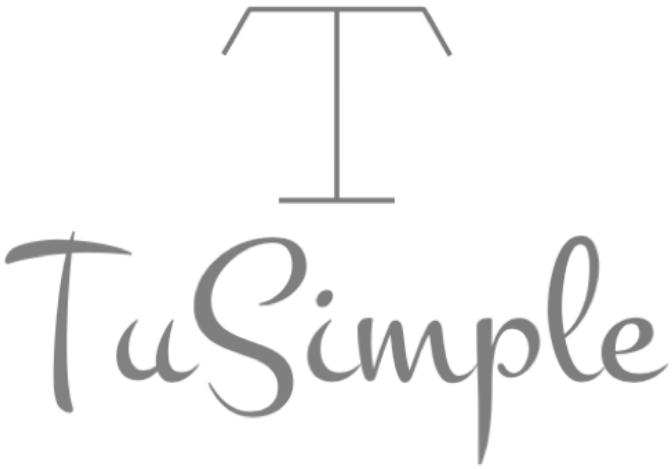


TuSimple

An Easy Graph Language

Jihao Zhang(jz2791)
Zicheng Xu(zx2197)
Shen Zhu(sz2609)
Ziyi Mu(zm2263)
Yunzi Chai(yc3228)

Content

1. Introduction

1.1. Motivation

In program algorithm, Graph is a very important and useful data structure. A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. Graph can present the relationship between different events, it is very intuitive and easy to understand. They are used in compiler and database. They are used to represent networks of communication, data organization, computational devices.

Although the current program language can support the implementation of graph, it is complicated and not intuitive. The nodes and edges between nodes cannot be shown intuitively, it contracts with the nature of graph. Besides, they can only provide limited functionality and the implementation of these graph algorithms are complex. So it is important to develop a programming language that is mainly used to deal with graph.

1.2. Objectives

In order to meet the need of implementing graph algorithms easily, we decide to develop a programming language that is used to deal with Graph, Tusimple. Tusimple can make coding graph related problems as easy as drawing graphs on a paper. It is should be a user-friendly Graph programming language that can give user an intuitive representation of graph and easy way to write code.

Because Tusimple is to deal with graph, so it can define nodes and edges of a graph, and assign value to all nodes and edges, define the type of nodes and edges, provide information about the relationship between different nodes and different graphs. To make it stand out from other current programming languages, the use of Tusimple should be intuitive and easy to understand. It is also easier to implement all these graph related algorithms, at the same time, Tusimple can provide some common graph algorithms build in function, so when users want to implement these algorithms, all they have to write is just one-line code. Of course, as a programming language, Tusimple can provide basic data structure, like list, set, map. And user can build up all common functions.

As a graph language, Tusimple has its own unique data structure, node and graph. Node has its name and value, nodes can be connected with each other to construct the edge. These edges can be assigned weight. As a graph data structure, we can add nodes and edges into these graph. Graph has its own basic build in function such as output the number of the nodes, and advanced build in function such as dfs and bfs. Tusimple has an intuitive and easy syntax to describe a graph. The definition and the usage of variables and functions is quite similar to C.

For example, if we want to create a directed graph with four nodes a,b,c and d, and edges a-b, a-c and a-d, here is how to create it in Tusimple:

```
node a, b, c, d;
graph g;
a->@{b, c, d} = @{1,2,3};
g.addNode(a);
g.addNode(b);
g.addNode(c);
g.addNode(d);
```

We use `->` connect nodes and represent edges, and we can batch assign value to these edges, this is very straightforward and easy to understand.

In the next chapters, we will give a language tutorial that give novice simple introduction of our language Tusimple, then a detailed and complete language reference manual will be given out in the next chapter. After this, we will introduce our project plan, timeline and everyone's responsibilities. Language architecture and test plans will be introduced in the later part.

2. Language Tutorial

2.1. Language Description

The TuSimple language is designed to make coding graphs as simple as drawing them on paper as well as representing and calculating graphs as simple as possible.

In order to achieve this goal, TuSimple has easy syntax and supports List, Set, Hashmap, Node and Graph. The optimal goal of this language is to provide a clear view on Node and Graph relationship in reality. Users could use this language to build their own graph and run some algorithms such as breadth-frist search, depth-first search and Dijkstra's single source shortest path algorithm on graphs.

2.2. Environment Setup

To compile this TuSimple program, you firstly need to compile the compiler. Open the TuSimple folder in the terminal and run **make**.

```
plt@ubuntu-plt:~/TuSimple-Graph-Language$ make
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,llvm.linker,llvm.bitreader,llvm.irreader -cflags -w,+a-4 \
        tusimple.native
Finished, 22 targets (0 cached) in 00:00:01.
cc    -c -o printbig.o printbig.c
plt@ubuntu-plt:~/TuSimple-Graph-Language$
```

And to test this compiler works well, you can run **./testall.sh**, then a series of test cases would run in terminal. If all of them success, it shows that this compiler is working well.

```
plt@ubuntu-plt:~/TuSimple-Graph-Language$ ./testall.sh
test-Dijkstra1...SUCCESS
test-arith1-add...SUCCESS
test-arith2-sub...SUCCESS
test-arith3-mult...SUCCESS
test-arith4-division...SUCCESS
test-bfs1...SUCCESS
test-bfs2...SUCCESS
test-bool...SUCCESS
test-data1...SUCCESS
test-dfs1...SUCCESS
test-dfs2...SUCCESS
test-for1...SUCCESS
test-for2...SUCCESS
test-for3...SUCCESS
test-fun1...SUCCESS
test-fun2...SUCCESS
test-graph1...SUCCESS
test-graph2...SUCCESS
test-graph3...SUCCESS
test-if1...SUCCESS
test-if2...SUCCESS
test-int...SUCCESS
test-list1...SUCCESS
test-list2...SUCCESS
test-list3...SUCCESS
test-list4...SUCCESS
test-list5...SUCCESS
test-map1...SUCCESS
```

Now you can start writing your program in TuSimple, the file extension of TuSimple program is ".tu". After writing your program, you can just compile this program using **./compile.sh <your file name>**, this compiler will generate ".exe" file and you can just run it.

2.3. Hello World Example

Let's write a simple hello world programming in TuSimple. In the first line of this program we first declare two nodes that contain integer values. Then, we initialize the nodes and then connect them with a undirected edge which has a weight of 1. Now you have coded the most simple graph in TuSimple!

```
int main(){
        node@{int} node1, node2;
        new node1; new node2;
        node1 -- node2 = 1;
}
```

2.4. Data Structures

2.4.1. List

TuSimple language provides some basic data structures such as list, set and hashmap for users to build some complex programs on top of them. For example, users can code breath-first search and depth-first search with the help of these data structures.

List is a built-in data structure, it can store int, float, string and node. You can declare a list by passing type of its stored variables:

```
list@{int} newList;
```

This will declaure a list "newList" that stores integers.

Now you will have two choices to inititaize this list.

```
new newList;
```

This will create an empty list.

```
newList = @{1, 2, 3};
```

In this way, you will have a list that contains integer 1, 2 and 3.

If you want to add more elements in this list, you can use

```
newList += @{4, 5, 6};
```

List also supports some operators and has built-in functions. You can get the first element in a List by

```
newList[0];
newList.get(0);
```

And to know the length of this List, just use

```
newList.length();
```

To remove the first element, you can use the remove built-in function

```
newList.remove(0);
```

Finally, if you want to print this list, printList here comes to help.

```
newList.printList();
```

2.4.2. Hashmap

Hashmap is a data structure that stores key-value pair.

To use hashmap, you will first need to declare and initialize it.

```
map@{int, string} intToString;
new intToString;
```

This defines a hashmap that maps integer to string. Then you can put something in hashmap and get the value by the key you passed.

```
intToString.put(1, "hello");
intToString.put(2, "world");

intToString.get(1);
```

And it should return “hello”. You can also check if this hashmap has some key and remove key-value pair by passing its key.

```
intToString.haskey(1);
intToString.haskey(3);

intToString.remove(1);
```

The first line should return true and the second line should return false.

2.4.3. Set

We also included set in our containers. The elements in the set can only be repeated once. You should first declare and initialize the set as other containers. Later you can add elements into the set using the += operator.

```
set@{int} set;
set += @{1, 2, 3};
```

set += @{1, 1, 1}; // Error! Try to add repeated elements.

2.5. Graph manipulation

To calculate and manipulate graphs with TuSimple, you have to initlized the graphs first. This is very intuitive and simple as drawing graphs on the paper. The first step is to declare the nodes and graphs. For nodes, you have to specify the value type that the node will store in the between the “@{”and “}”. Then you should declare the graph and initialize the nodes and graph. Then you can start to connect the nodes with edges. There are two ways of doing this. One is to link the nodes one by one using the “--” operator for undirected edge and “->” for directed edge. Another way is to link the nodes in batch, which also uses those two operators. However, one of the operands can be a node list. The example are as follows.

```
node@{int} s, a, b, c, d; //declare nodes
graph g; // declare graph
```

```
list@{int} lst; // declare list
// initialize all graph components and containers
new s; new a; new b; new c; new d; new lst;
// assign edges in batch. This is equivalent to s -- a = 1;s--b = 1;s--c = 1;
s -- {a , b , c} = {1, 1, 1};
d -- {a , b, c} = {1, 1, 1};
```

Cool! Now you already built the graph. Let's do something interesting with it. You can call our built-in functions by using a dot and the function name after the graph components and containers. In the following example, we showed how to do the bfs and print the graph in two lines! This will need a lot more codes in Java!

```
// Do a dfs on the graph
lst = g.dfs(s);
// output: s a d b c
lst.printList();
```

Now, let's try something more harder. Remember what's the most important steps in the shortest path algorithms? The Bellman-Ford and Dijkstra's. Yes, that's the relaxation. Let's do a little bit of review here. To relax an edge v->w means to test whether the best known way from s to w is to go from s to v, then take the edge from v to w, and, if so, update our data structures. Vertex relaxation is to relax all the edges pointing from a given vertex. To do all the steps in Java, you need the following lines of code:

```
private void relax(EdgeWeightedDigraph G, int v) {
    for (DirectedEdge e : G.adj(v)) {
        int w = e.to();
        if (distTo[w] > distTo[v] + e.weight()) {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
        }
    }
}
```

But in TuSimple, you only need one line after the initilizations.

```
g.relax(x);
```

Pretty cool huh? To know more about out built-in functions, you can check our language reference menu.

3. Language Reference Manual

3.1. Tokens

There are five classes of tokens: identifiers, keywords, literals, operators, constants and delimiters. White spaces are ignored.

3.1.1. Comments

The characters /* introduce a multi-line comment, which terminates with the characters */. Comments do not nest, and they do not occur within a string or character literals.The characters // introduce a single-line comment.

3.1.2. Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore _ counts as a letter. Upper and lower case letters are different. Identifiers may have any length.

3.1.3. Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise: int float bool string list set node map graph if else for while continue break return NULL print TRUE FALSE

3.1.4. Literals

String Literals are a sequence of zero or more letters, spaces, digits, other ASCII characters numbers 32 to 126, excluding the single quote (number 39). These strings should be enclosed in single quotes. Integer literals are a sequence of one or more digits. Only numbers in decimal format are recognized. Float literals are a sequence of one or more digits and a decimal point with another sequence of one or more digits. Only numbers in decimal format are recognized.

3.1.1. Delimiters

3.1.1.1. Parentheses and Braces

Parentheses are used to force evaluation of parts of a program in a specific order. They are also used to enclose arguments for a function.

3.1.1.2. Commas

Commas are used to separate arguments in declaration and function arguments.

3.1.1.3. Square Brackets

Square brackets are used for accessing of array elements.

3.1.1.4. Semicolon

Semicolons are used to terminate a sequence of code.

3.1.1.5. Curly Braces

Curly braces are used to enclose function definitions and for/while loops, as well as node declarations.

3.1.1.6. Constants

maxInt = 2147483647

minInt = -2147483638

3.2. Data Types

3.2.1. Integers

Possible value:
32-bit signed Integer (-2147483638 ~ 2147483647)

Example:

int i1 = 0;

int i2 = -123

int i3 = 43;

3.2.2. Floats

Possible value:
A IEEE 754 double-precision (64-bit) numbers

Example:

float f1 = 0.356;

float f2 = 3.4e-16;

float f3 = 1;

3.2.3. Booleans

Possible value:

bool b1 = true;

bool b2 = false;

3.2.4. Strings

Possible value:
A sequence of ASCII enclosed by double quotes

Example:

string s1 = “PLT”;

string s2 = “” ;

string s3 = “Hello”;

3.2.5. Lists

A list of homogeneous elements.

List should be declared in the format: list (<type of the list element>) <name of list >;

Example:

list@{int} l;

list@{node@{int}} queue;

{1, 1, 1, 1}

{node1, node2, node3}

3.2.6. Sets

A set of homogeneous elements, in which each element should be unique.

Set should be declared in the format: set (<type of the set element>) <name of set >;

Example:

set@{int} s;

set@{node@{int}} set_of_node;

{1, 2, 3, 4}

{node1, node2, node3}

3.2.7. Nodes

A node of the map. Contains fields of name, value and a list of all the nodes connected to it.

Node should be declared in the format: node (<type of the node value>) <name of node >;

Example:

node@{string} n;

node@{int} a, b, c, d;

3.2.8. Maps

A hashmap of keys and its values. The manipulation of the hashmap is based on the keys.

Map should be declared in the format: map (<pair of the map values>) <name of map >;

Example:

map@{node@{int}, int} distant;

3.2.9. Graphs

A graph contains all nodes and edges. This type provides the built-in functions to implement the core function of this language.

Graph should be declared in the format: graph <name of map >;

Example:

graph g;

3.3. Operators

Name	Operator	Int & float	Bool	string	list/set/map	node
PLUS	+	Add two values Example: 1 + 2 (evaluates 3) 3.15 + 5.20 (evaluates 8.35)	/	Connect two string Example: “I” + “love PLT” (evaluates “Ilove PLT”)	Add new element Example: List l Node n l += @{n} (evaluates a list with the node n)	/
MINUS	-	Substract Example: 1 - 2 (evaluates -1) 3.15 - 5.20 (evaluates -2.05)	/	/	/	/

MULTIPLY	*	Multiply Example: 1 * 2 (evaluates 2) 3.15 * 1.0 (evaluates 3.15)	/	/	/	/
DIVIDE	/	Divide Example: 1 / 2 (evaluates 0) 3.15 / 1.0 (evaluates 3.15)	/	/	/	/
ASSIGN	=	Set the left variable with the value of right side	Set the left variable with the value of right side	Set the left variable with the value of right side	Set the left variable with the value of right side	Set the node with the value of right side
EQUAL	==	Compare the value	Compare the value	Compare the name string	Compare the name string	Compare the name string
AND	&&	Calculate using boolean value	&	Calculate using boolean value	/	/
OR	||	Calculate using boolean value	|	Calculate using boolean value	/	/
NOT	!	Calculate using boolean value	!	Calculate using boolean value	/	/
GT	>	Compare the value	/	/	/	/
LT	<	Compare the value	/	/	/	/
GE	>=	Compare the value	/	/	/	/

LE	<=	Compare the value	/	/	/	/
LINK	->	/	/	/	/	Link current node to another
DI-LINK	--	/	/	/	/	Link both nodes to each other
NEXT	++	Add value by 1	/	/	Move to the next element	/
BRACE	{}	/		/	Take value from set	/
BRACKET	[]	/		Take character with index	Take value from list	/
PARENTH	()	/		/	Take value from map	/

3.4. Built-in Functions

Public Function		
Name	**Signature**	**Description**
max	min(T a, T b)	*Return the major value*
min	max(T a, T b)	*Return the minor value*
prints	prints(string s)	*Print the target string*
print	print(int i)	*Print the target number*

API of list(list@{T} l)		
Name	**Signature**	**Description**
length	l.length()	*The total number of elements in the list*
pop	l.pop()	*Delete the last element in the list*
remove	l.remove(int i)	*Delete the corresponding element using the index i*
get	l.get(int i)	*Access the corresponding element using the index i*

concat	l.concat(list l2)	*Add all the elements of input list to the end of the list*
printList	l.printList()	*Output all the elements of the list*
add(is not built-in function)	l += @{int i}	*Add the element to the end of the list*

API of set(set @{T} s)		
Name	**Signature**	**Description**
length	s.length()	*The total number of elements in the set*
put	s.put(T a)	*Put the input value into the set; If the key value is already existed, it would not be included*
remove	s.remove(T a)	*Delete the corresponding element with the value a*
contain	s.contain(T a)	*Test whether the set has the value a*

API of map(map @{T1, T2} m)		
Name	**Signature**	**Description**
size	m.size()	*The total number of elements in the hashmap*
put	m.put(T1 a, T2 b)	*Put the input pair into the hashmap; If the key value is already existed, update the value*
get	m.get(T1 a)	*Access the corresponding value using the key value a*
remove	m.remove(T1 a)	*Delete the corresponding pair using the key value a*
haskey	m.haskey(T1 a)	*Test whether the map has the key a*

API of node(node@{T} n)		
Name	**Signature**	**Description**
value	n.value()	*Return the value of the node*
name	n.name()	*Return the name of the node*
setValue	n.setValue(T a)	*Set the value of the node using the input value a*
iterNode	n.iterNode(int i)	*Return the i-th node linked to node n*

length	n.length()	*Return the total number of nodes linked to node n*
weightIter	n.weightIter(int i)	*Return the value of the edge linking the i-th node and node n*

API of graph(graph g)		
Name	**Signature**	**Description**
init_tag	g.init_tag()	*Tagging each node in the given map with initial value*
relax	g.reduce(node n)	*Updating the value of connected nodes, select the smaller value; It's the "relaxation" operation in the graph algorithms*
expand	g.expand(node n)	*Updating the value of connected nodes, select the bigger value*
combine	g.combine(graph g2)	*Return the graph combining g1 and g2*
bfs	g.bfs(node n)	*Execute the BFS to the graph, return the access sequence of nodes*
dfs	g.dfs(node n)	*Execute the DFS to the graph, return the access sequence of nodes*
findGraphNode	g.findGraphNode(node n)	*Return the node in the graph using the name of the input node*
addNode	g.addNode(node n)	*Add the node n into the graph*
addEdge	g.addEdge(node n1, node n2, int w)	*Add the edge linking node n1 and node n2 with the weight of w into the graph*
length	g.length()	*Return the total number of nodes in the graph*
printGraph	g.printGraph()	*Output all the nodes and edges of the graph*

3.5. Statements

3.5.1. The if Statement

The if statement is the similar with c in definition.

Example:

if (condition c1) then { sentence; } else { sentence; }

The condition c1 could be bool or int.

3.5.2. The while Statement

The while statement could be consider as a expansion of if statement.

Example:

while (condition c1) { sentence s1; }

The sentence s1 would be executed constantly while the condition is satisfied.

4. Project Plan

4.1. Project Process Overview:

We started to contrive our language from the beginning of the class. At the end of January we have assigned our role and had several meetings on the basic design of our language. Our group met each other more than one times every week, and discussed with our TA Julie Chen time to time throughout the development process. After several group meeting at the end of January and beginning of February, a draft language proposal have been determined and was composed by our team member, especially by our language guru Zicheng Xu.

After meeting with our TA for the feedback, we started to sketch some of our basic built-in types and build-in functions for the purpose of implementation of an easy graph language. Since we have assigned each member a role, the responsible member with the assigned role was in charge of the part he was assigned to, and the rest of the team followed the path that responsible member is heading. During the reading week in mid March, we had several meetings to carefully examine the Micro-C language compiler and every team member started to have a big picture on how to extend the existing well-written code base to accommodate our language design.

Determining our implementation built upon the OCaml Micro-C language is the first important design decision of our team, and it was lead by manager Jihao Zhang. Although Micro-C is more like a prototype of a complete language, it has lots of great basic features like function calls and variable declaration. Also its code is very clear and easy to understand, which helps us to understand the magic of a language compiler. Then it became easier for us to complete the scanner and parser to accommodate our build-in types, and then semantic check and codegen to make the end-to-end implementation of some basic features.

But after several weeks tryout we found that it would be very hard to implement everything in OCaml, especially for the graph algorithm part and memory management. So the beginning of April we made our second important design decision, leading by our system architect Shen Zhu, which is incorporating the C code implementation to our OCaml compiler code base by linking C library containing algorithm code.

After the decision our development process was moving much faster than before. A few weeks after we finished majority of the memory management and graph algorithm. Then all of our team members were working hard on the linking process of OCaml and C algorithm library. This involved a lot of understanding of the OCaml LLVM assembly level code. At the end of April we have implemented an end-to-end including the graph algorithm workflow for our project.

The final week is going smoothly since we have finished implementing the overall framework of our language, and only adding additional build-in functions and other features to make our code easier to use. With the help of the our test suite implementation, we could speed up of development with no worries.

Finally we implemented the dijkstra and neural network algorithm using our Tusimple easy graph language, as everything needed for the graph algorithm had been fully implemented and tested. So we just put up all the pieces and assemble the entire complex algorithm like a breeze.

4.2. Project Timeline:

1.25	Start the first meeting and purpose the basic design of our language.
2.6	Another meeting to outline our language and determine the preliminary design direction.
2.8	Submit the Proposal determining the basic functionality of our Tusimple language.
2.18	Have meetings to determine the basic built-in type and build-in functions for easy graph manipulation.
2.22	Submit the LRM specifying our graph language reference.
3.13-18	Have several reading week meetings on Micro-C compiler code and LLVM system.
3.20	Work on scanner and parser to accommodate our build-in types.
3.24	Work on Semantic and Codegen to make the first end-to-end implementation.
3.27	Demo the Helloworld and impress our TA :)
4.8	Start to incorporate C to our Ocaml code base by linking C.
4.9	Start to implement basic graph algorithm using C.
4.16	Successfully link our Ocaml to C on build-in type creation.

4.25	Finish majority of the graph manipulation algorithm.
5.1	Start to write unit test suit and finish the compiler parser and AST.
5.4	Finish all the build-in types and functions in compiler back end.
5.6	Finish all the required graph manipulation algorithm.
5.8	Implement the dijkstra and neural network algorithm using our Tusimple easy graph language.
5.9	Finalize the test suite and bug fix.
5.10	Finish the final demo.
5.10	Submit the final report and final code base.

4.3. Roles and Responsibility:

Jihao Zhang: Jihao Zhang is the Manager of our team. He organized the regular team meeting, implemented most of the compiler front end, and worked on semantic check and code generation end-to-end. He also helped to make the important design decisions to speed up our implementation.

Shen Zhu: Shen Zhu is the System Architecture of the team. He implemented Ocaml to C linking interface and implemented build-in type creating and manipulation as C library. He had lots of implementation of the code generation with LLVM moe, and resolved difficult problems regarding memory management.

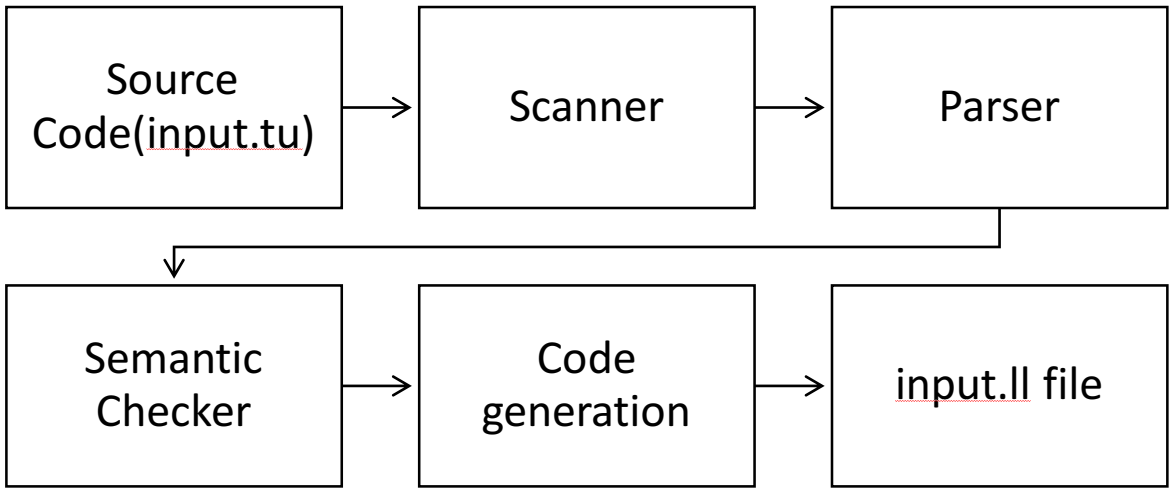
Ziyi Mu: Ziyi Mu is the assistant System Architecture of the team. He worked on the Ocaml to C interface, implemented part of the compiler front end and a holistic semantic check of the language, and implemented several features including operator overloading and built-in function incorporation.

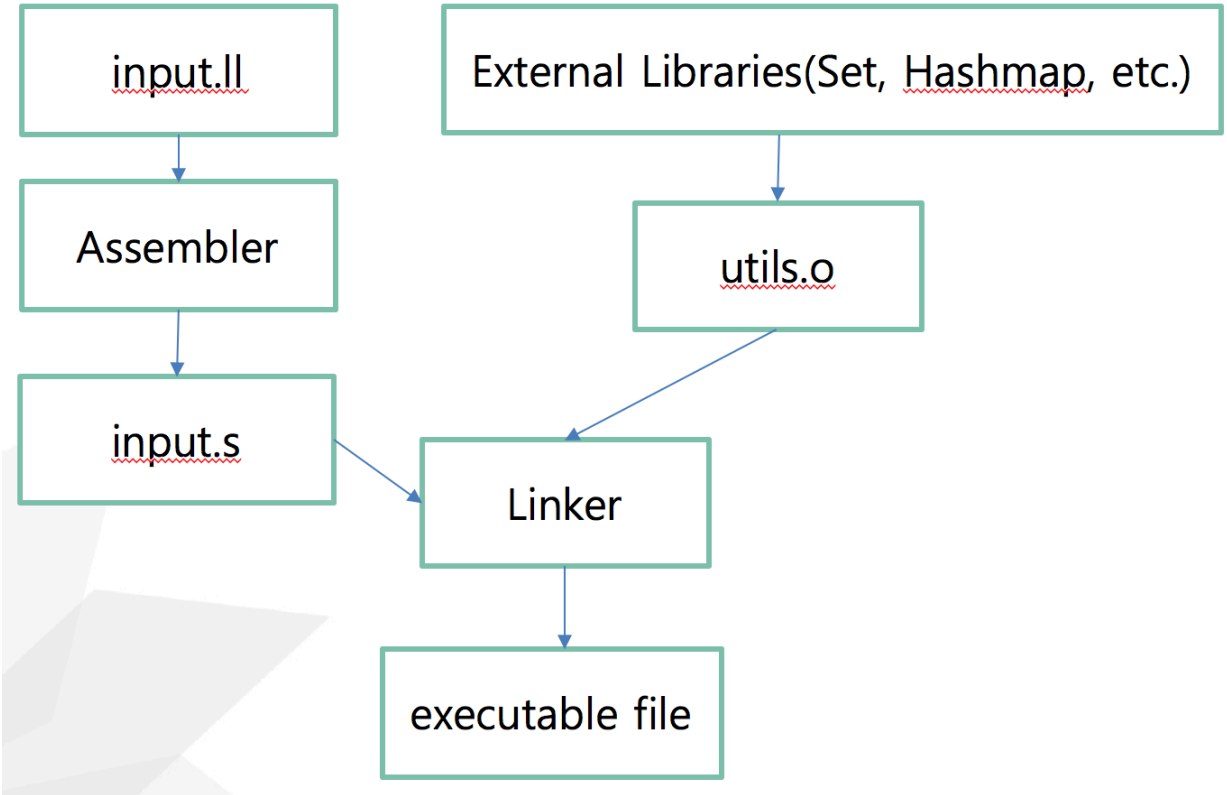
Zicheng Xu: Zicheng Xu is the Language Guru of the team. He designed the outline of what our language look like, wrote and refined the language reference manual. He also implemented the core graph algorithm both in C for build-in functions, and in our Tusimple language for testing and demo propose.

Yunzi Chai: Yunzi Chai is the Tester of our team. She wrote the entire test suite and testing scripts to make sure our language development process going smoothly. She implemented test cases covering every corner of the language, so our team member could easily run regression tests to make sure implementing new feature won't break precious parts of the code.

5. Language Architecture

5.1. Diagram





5.2. Compiler

The compiler includes the following parts:

ast.ml: A module which lays out the possible nodes in an AST representation of a syntactically correct program.

codegen.ml: Takes in a semantically checked AST (SAST), assuming it is both syntactically and semantically valid, and translates each AST node into LLVM IR.

semant.ml: semantic checker of the program

tusimple.ml: Top-level of the TuSimple compiler: scan & parse the input, check the resulting AST, generate LLVM IR, and dump the module.

scanner.mll: Converts the source code into tokens.

parser.mly: Takes in tokens from the scanner, checks its syntactic correctness. If these checks are passed, it generates an abstract syntax tree (AST) representation of the program.

5.3. Scanner

The scanner module takes in the source file from and converts it to a list of tokens. Comments are removed from the tokens. The scanner also discards whitespace, tabs, and newlines.

5.4. Parser

The parser is generated by ocamlyacc. Builds the AST from tokens and also will throw exceptions relating to syntax errors. The parser scans through the tokens passed from the scanner and construct an AST.

5.5. Abstract Syntax Tree

The abstract syntax tree is generated by the parser and represents the overall structure of the program. All phases proceeding lexical analysis utilize the AST.

5.6. Semantic Analyzer

The abstract sysntax tree are then passed to the semantic checker. It converts it to a map of strings which utilizes each function's name as its key. The semantic checker then passes through the list of global statements. It ensures any variable definitions matching types while also raising errors if it finds any illigal types.

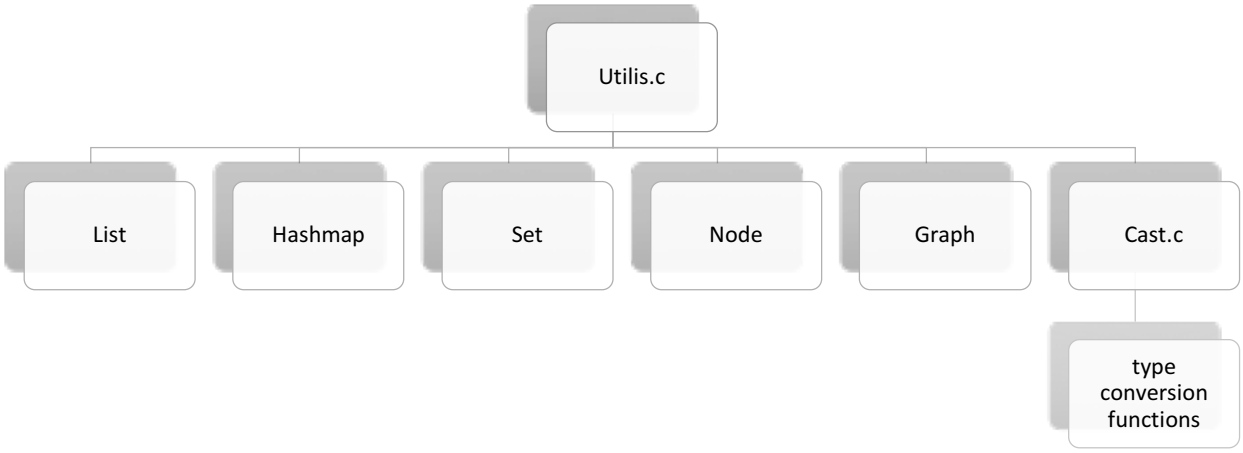
5.7. Code generator

The code generator translates the given abstract syntax tree into LLVM IR. The entire building process is performed by iterating through the statements and expressions. The code generator takes the LLVM IR and dumps it into a LLVM module which is written to an indicated output.ll file.

5.8. External libraries

We wrote external libraries in C, which includes containers and graph data structures. For containers, we implemented set, map, list. For graph data structures, we implemented node, graph, and edge representations.

The library structures are as follows:



6. Test Plan

6.1. Source Code and Generated Target Code

6.1.1. Example 1: Graph Depth First Search

Source Language Program:

```
int main(){
int a, b, i;
bool c, d;
int size, now;
node@{int} node1,node2,node3,node4,node5,node6,node7,node8,node9,node10;
list@{node@{int}} l, l2, rec;
set@{node@{int}} visited;
map@{string,int} hash;
graph g;

new node1;
new node2;
new node3;
new node4;
new node5;
new node6;
new node7;
new node8;
new node9;
new node10;
new l;
new l2;
new rec;
new visited;
new hash;
new g;

node1 -> node2 = 11;
node2 -> node4 = 12;
node1 -> node3 = 14;
node3 -> node4 = 15;

node1.setvalue(1);
node2.setvalue(2);
node3.setvalue(3);
node4.setvalue(4);

g.addNode(node1);
g.addNode(node2);
g.addNode(node3);
g.addNode(node4);

l = g.dfs(node1);

node5 = l.get(0);
```

```
  prints(node5.name());

  node6 = l.get(1);
  prints(node6.name());

  node7 = l.get(2);
  prints(node7.name());

  node8 = l.get(3);
  prints(node8.name());

  l2 += @{node1};
  rec += @{node1};
  hash.put(node1.name(), 0);

  while (l2.length()!=0){
    node9 = l2.get(l2.length()-1);
    size = node9.length();
    now = hash.get(node9.name());
    if (now<size){
      node10 = node9.iterNode(now);
      if (!hash.haskey(node10.name())){
        l2 += @{node10};
        rec += @{node10};
        hash.put(node10.name(),0);
      }
      hash.put(node9.name(), now+1);
    } else {
      l2.remove(l2.length()-1);
    }
  }

  print(rec.length());

  node5 = rec.get(0);
  prints(node5.name());
    print(node5.value());

  node6 = rec.get(1);
  prints(node6.name());
    print(node6.value());

  node7 = rec.get(2);
  prints(node7.name());
    print(node7.value());

  node8 = rec.get(3);
  prints(node8.name());
    print(node8.value());

}
```

Target Language Program:

```
; ModuleID = 'TuSimple'

%struct.List = type { i32, i32, i8**, i32 }
%struct.hashmap = type { i32, i32, i32, i32, %struct.hashmap_element* }
%struct.hashmap_element = type { i8*, i32, [2 x i8*] }
%struct.Set = type { i32, i32, %struct.List* }
%struct.Node = type { i32, i8*, i8*, %struct.List*, %struct.List* }
%struct.Graph = type { i8*, %struct.List*, %struct.hashmap* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@0 = private unnamed_addr constant [6 x i8] c"node1\00"
@1 = private unnamed_addr constant [6 x i8] c"node2\00"
@2 = private unnamed_addr constant [6 x i8] c"node3\00"
@3 = private unnamed_addr constant [6 x i8] c"node4\00"
@4 = private unnamed_addr constant [6 x i8] c"node5\00"
@5 = private unnamed_addr constant [6 x i8] c"node6\00"
@6 = private unnamed_addr constant [6 x i8] c"node7\00"
@7 = private unnamed_addr constant [6 x i8] c"node8\00"
@8 = private unnamed_addr constant [6 x i8] c"node9\00"
@9 = private unnamed_addr constant [7 x i8] c"node10\00"
@10 = private unnamed_addr constant [2 x i8] c"g\00"

declare i32 @get_list_size(%struct.List*)

declare i8* @pop_list_element(%struct.List*)

declare i8* @remove_list_element(%struct.List*, i32)

declare i8* @get_list_element(%struct.List*, i32)

declare %struct.List* @concat_list(%struct.List*, %struct.List*, ...)

declare %struct.List* @create_list(i32)

declare %struct.List* @plus_list(%struct.List*, ...)

declare i8* @print_list(%struct.List*)

declare %struct.hashmap* @create_hashmap(i32, i32)

declare %struct.hashmap* @hashmap_put(%struct.hashmap*, ...)

declare i8* @hashmap_get(%struct.hashmap*, ...)

declare i32 @hashmap_length(%struct.hashmap*)

declare i1 @hashmap_haskey(%struct.hashmap*, ...)

declare %struct.hashmap* @hashmap_remove(%struct.hashmap*, ...)
```

```
declare %struct.Set* @put_set_from_list(%struct.Set*, %struct.List*)

declare %struct.Set* @create_set(i32)

declare %struct.Set* @put_set(%struct.Set*, ...)

declare i32 @get_set_size(%struct.Set*)

declare i1 @check_set_element(%struct.Set*, ...)

declare %struct.Set* @remove_set_element(%struct.Set*, ...)

declare %struct.Node* @createNode(i8*, i32)

declare i32 @getEdgeValue(%struct.Node*, %struct.Node*)

declare i32 @addNodeEdge(%struct.Node*, %struct.Node*, i32)

declare i32 @addReverseEdge(%struct.Node*, %struct.Node*, i32)

declare i8* @getNodeName(%struct.Node*)

declare %struct.Node* @setNodeValue(%struct.Node*, ...)

declare %struct.Node* @iterNode(%struct.Node*, i32)

declare i32 @getNodeLength(%struct.Node*)

declare i8* @get_node_value(%struct.Node*)

declare i32 @weightIterNode(%struct.Node*, i32)

declare i32 @graphLength(%struct.Graph*)

declare %struct.Graph* @createGraph(i8*)

declare %struct.Graph* @addGraphNode(%struct.Graph*, %struct.Node*)

declare %struct.Graph*
@addGraphEdge(%struct.Graph*, %struct.Node*, %struct.Node*, i32)

declare %struct.Node* @iterGraph(%struct.Graph*, i32)

declare %struct.Node* @findGraphNode(%struct.Graph*, i8*)

declare %struct.Node* @init_tag(%struct.Graph*)

declare %struct.Node* @reduce(%struct.Graph*, %struct.Node*)

declare %struct.Node* @expand(%struct.Graph*, %struct.Node*)

declare %struct.Graph* @combine(%struct.Graph*, %struct.Graph*)
```

```
declare %struct.List* @bfs(%struct.Graph*, %struct.Node*)

declare %struct.List* @dfs(%struct.Graph*, %struct.Node*)

declare i8* @print_graph(%struct.Graph*)

declare i32 @voidToint(i8*)

declare double @voidTofloat(i8*)

declare i1 @voidTobool(i8*)

declare i8* @voidTostring(i8*)

declare %struct.Node* @voidTonode(i8*)

declare %struct.Graph* @voidTograph(i8*)

declare i32 @printf(i8*, ...)

declare i32 @printbig(i32)

define i32 @main() {
entry:
  %i = alloca i32
  %b = alloca i32
  %a = alloca i32
  %d = alloca i1
  %c = alloca i1
  %now = alloca i32
  %size = alloca i32
  %node10 = alloca %struct.Node*
  %node9 = alloca %struct.Node*
  %node8 = alloca %struct.Node*
  %node7 = alloca %struct.Node*
  %node6 = alloca %struct.Node*
  %node5 = alloca %struct.Node*
  %node4 = alloca %struct.Node*
  %node3 = alloca %struct.Node*
  %node2 = alloca %struct.Node*
  %node1 = alloca %struct.Node*
  %rec = alloca %struct.List*
  %l2 = alloca %struct.List*
  %l = alloca %struct.List*
  %visited = alloca %struct.Set*
  %hash = alloca %struct.hashmap*
  %g = alloca %struct.Graph*
  %createNode = call %struct.Node* @createNode(i8* getelementptr inbounds ([6
x i8]* @0, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode, %struct.Node** %node1
  %createNode1 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @1, i32 0, i32 0), i32 0)
```

```
  store %struct.Node* %createNode1, %struct.Node** %node2
  %createNode2 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @2, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode2, %struct.Node** %node3
  %createNode3 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @3, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode3, %struct.Node** %node4
  %createNode4 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @4, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode4, %struct.Node** %node5
  %createNode5 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @5, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode5, %struct.Node** %node6
  %createNode6 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @6, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode6, %struct.Node** %node7
  %createNode7 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @7, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode7, %struct.Node** %node8
  %createNode8 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @8, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode8, %struct.Node** %node9
  %createNode9 = call %struct.Node* @createNode(i8* getelementptr inbounds
([7 x i8]* @9, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode9, %struct.Node** %node10
  %create_list = call %struct.List* @create_list(i32 4)
  store %struct.List* %create_list, %struct.List** %l
  %create_list10 = call %struct.List* @create_list(i32 4)
  store %struct.List* %create_list10, %struct.List** %l2
  %create_list11 = call %struct.List* @create_list(i32 4)
  store %struct.List* %create_list11, %struct.List** %rec
  %create_set = call %struct.Set* @create_set(i32 4)
  store %struct.Set* %create_set, %struct.Set** %visited
  %create_hashmap = call %struct.hashmap* @create_hashmap(i32 3, i32 0)
  store %struct.hashmap* %create_hashmap, %struct.hashmap** %hash
  %createGraph = call %struct.Graph* @createGraph(i8* getelementptr inbounds
([2 x i8]* @10, i32 0, i32 0))
  store %struct.Graph* %createGraph, %struct.Graph** %g
  %node212 = load %struct.Node** %node2
  %node113 = load %struct.Node** %node1
  %addNodeEdge = call i32
@addNodeEdge(%struct.Node* %node113, %struct.Node* %node212, i32 11)
  %node414 = load %struct.Node** %node4
  %node215 = load %struct.Node** %node2
  %addNodeEdge16 = call i32
@addNodeEdge(%struct.Node* %node215, %struct.Node* %node414, i32 12)
  %node317 = load %struct.Node** %node3
  %node118 = load %struct.Node** %node1
  %addNodeEdge19 = call i32
@addNodeEdge(%struct.Node* %node118, %struct.Node* %node317, i32 14)
  %node420 = load %struct.Node** %node4
  %node321 = load %struct.Node** %node3
```

```
  %addNodeEdge22 = call i32
@addNodeEdge(%struct.Node* %node321, %struct.Node* %node420, i32 15)
  %node123 = load %struct.Node** %node1
  %setNodeValue = call %struct.Node* (%struct.Node*, ...)*
@setNodeValue(%struct.Node* %node123, i32 1)
  %node224 = load %struct.Node** %node2
  %setNodeValue25 = call %struct.Node* (%struct.Node*, ...)*
@setNodeValue(%struct.Node* %node224, i32 2)
  %node326 = load %struct.Node** %node3
  %setNodeValue27 = call %struct.Node* (%struct.Node*, ...)*
@setNodeValue(%struct.Node* %node326, i32 3)
  %node428 = load %struct.Node** %node4
  %setNodeValue29 = call %struct.Node* (%struct.Node*, ...)*
@setNodeValue(%struct.Node* %node428, i32 4)
  %node130 = load %struct.Node** %node1
  %g31 = load %struct.Graph** %g
  %addGraphNode = call %struct.Graph*
@addGraphNode(%struct.Graph* %g31, %struct.Node* %node130)
  %node232 = load %struct.Node** %node2
  %g33 = load %struct.Graph** %g
  %addGraphNode34 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g33, %struct.Node* %node232)
  %node335 = load %struct.Node** %node3
  %g36 = load %struct.Graph** %g
  %addGraphNode37 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g36, %struct.Node* %node335)
  %node438 = load %struct.Node** %node4
  %g39 = load %struct.Graph** %g
  %addGraphNode40 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g39, %struct.Node* %node438)
  %node141 = load %struct.Node** %node1
  %g42 = load %struct.Graph** %g
  %dfs = call %struct.List* @dfs(%struct.Graph* %g42, %struct.Node* %node141)
  store %struct.List* %dfs, %struct.List** %l
  %l43 = load %struct.List** %l
  %get_list_element = call i8* @get_list_element(%struct.List* %l43, i32 0)
  %voidTonode = call %struct.Node* @voidTonode(i8* %get_list_element)
  store %struct.Node* %voidTonode, %struct.Node** %node5
  %node544 = load %struct.Node** %node5
  %getNodeName = call i8* @getNodeName(%struct.Node* %node544)
  %node545 = load %struct.Node** %node5
  %getNodeName46 = call i8* @getNodeName(%struct.Node* %node545)
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName46)
  %l47 = load %struct.List** %l
  %get_list_element48 = call i8* @get_list_element(%struct.List* %l47, i32 1)
  %voidTonode49 = call %struct.Node* @voidTonode(i8* %get_list_element48)
  store %struct.Node* %voidTonode49, %struct.Node** %node6
  %node650 = load %struct.Node** %node6
  %getNodeName51 = call i8* @getNodeName(%struct.Node* %node650)
  %node652 = load %struct.Node** %node6
  %getNodeName53 = call i8* @getNodeName(%struct.Node* %node652)
```

```
  %printf54 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName53)
  %l55 = load %struct.List** %l
  %get_list_element56 = call i8* @get_list_element(%struct.List* %l55, i32 2)
  %voidTonode57 = call %struct.Node* @voidTonode(i8* %get_list_element56)
  store %struct.Node* %voidTonode57, %struct.Node** %node7
  %node758 = load %struct.Node** %node7
  %getNodeName59 = call i8* @getNodeName(%struct.Node* %node758)
  %node760 = load %struct.Node** %node7
  %getNodeName61 = call i8* @getNodeName(%struct.Node* %node760)
  %printf62 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName61)
  %l63 = load %struct.List** %l
  %get_list_element64 = call i8* @get_list_element(%struct.List* %l63, i32 3)
  %voidTonode65 = call %struct.Node* @voidTonode(i8* %get_list_element64)
  store %struct.Node* %voidTonode65, %struct.Node** %node8
  %node866 = load %struct.Node** %node8
  %getNodeName67 = call i8* @getNodeName(%struct.Node* %node866)
  %node868 = load %struct.Node** %node8
  %getNodeName69 = call i8* @getNodeName(%struct.Node* %node868)
  %printf70 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName69)
  %node171 = load %struct.Node** %node1
  %create_list72 = call %struct.List* @create_list(i32 4)
  %node173 = load %struct.Node** %node1
  %plus_list = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list72, %struct.Node* %node173)
  %l274 = load %struct.List** %l2
  %concat_list = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %l274, %struct.List* %create_list72)
  store %struct.List* %concat_list, %struct.List** %l2
  %node175 = load %struct.Node** %node1
  %create_list76 = call %struct.List* @create_list(i32 4)
  %node177 = load %struct.Node** %node1
  %plus_list78 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list76, %struct.Node* %node177)
  %rec79 = load %struct.List** %rec
  %concat_list80 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %rec79, %struct.List* %create_list76)
  store %struct.List* %concat_list80, %struct.List** %rec
  %node181 = load %struct.Node** %node1
  %getNodeName82 = call i8* @getNodeName(%struct.Node* %node181)
  %hash83 = load %struct.hashmap** %hash
  %hashmap_put = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash83, i8* %getNodeName82, i32 0)
  br label %while

while:                                            ; preds = %merge, %entry
  %l2131 = load %struct.List** %l2
  %get_list_size132 = call i32 @get_list_size(%struct.List* %l2131)
  %tmp133 = icmp ne i32 %get_list_size132, 0
  br i1 %tmp133, label %while_body, label %merge134
```

```
while_body:                                       ; preds = %while
  %l284 = load %struct.List** %l2
  %get_list_size = call i32 @get_list_size(%struct.List* %l284)
  %tmp = sub i32 %get_list_size, 1
  %l285 = load %struct.List** %l2
  %get_list_element86 = call i8* @get_list_element(%struct.List* %l285,
i32 %tmp)
  %voidTonode87 = call %struct.Node* @voidTonode(i8* %get_list_element86)
  store %struct.Node* %voidTonode87, %struct.Node** %node9
  %node988 = load %struct.Node** %node9
  %getNodeLength = call i32 @getNodeLength(%struct.Node* %node988)
  store i32 %getNodeLength, i32* %size
  %node989 = load %struct.Node** %node9
  %getNodeName90 = call i8* @getNodeName(%struct.Node* %node989)
  %hash91 = load %struct.hashmap** %hash
  %hashmap_get = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %hash91, i8* %getNodeName90)
  %voidToint = call i32 @voidToint(i8* %hashmap_get)
  store i32 %voidToint, i32* %now
  %now92 = load i32* %now
  %size93 = load i32* %size
  %tmp94 = icmp slt i32 %now92, %size93
  br i1 %tmp94, label %then, label %else125

merge:                                            ; preds
= %else125, %merge101
  br label %while

then:                                             ; preds = %while_body
  %now95 = load i32* %now
  %node996 = load %struct.Node** %node9
  %iterNode = call %struct.Node* @iterNode(%struct.Node* %node996,
i32 %now95)
  store %struct.Node* %iterNode, %struct.Node** %node10
  %node1097 = load %struct.Node** %node10
  %getNodeName98 = call i8* @getNodeName(%struct.Node* %node1097)
  %hash99 = load %struct.hashmap** %hash
  %hashmap_haskey = call i1 (%struct.hashmap*, ...)*
@hashmap_haskey(%struct.hashmap* %hash99, i8* %getNodeName98)
  %tmp100 = xor i1 %hashmap_haskey, true
  br i1 %tmp100, label %then102, label %else

merge101:                                         ; preds = %else, %then102
  %now119 = load i32* %now
  %tmp120 = add i32 %now119, 1
  %node9121 = load %struct.Node** %node9
  %getNodeName122 = call i8* @getNodeName(%struct.Node* %node9121)
  %hash123 = load %struct.hashmap** %hash
  %hashmap_put124 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash123, i8* %getNodeName122, i32 %tmp120)
  br label %merge

then102:                                          ; preds = %then
```

```
  %node10103 = load %struct.Node** %node10
  %create_list104 = call %struct.List* @create_list(i32 4)
  %node10105 = load %struct.Node** %node10
  %plus_list106 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list104, %struct.Node* %node10105)
  %l2107 = load %struct.List** %l2
  %concat_list108 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %l2107, %struct.List* %create_list104)
  store %struct.List* %concat_list108, %struct.List** %l2
  %node10109 = load %struct.Node** %node10
  %create_list110 = call %struct.List* @create_list(i32 4)
  %node10111 = load %struct.Node** %node10
  %plus_list112 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list110, %struct.Node* %node10111)
  %rec113 = load %struct.List** %rec
  %concat_list114 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %rec113, %struct.List* %create_list110)
  store %struct.List* %concat_list114, %struct.List** %rec
  %node10115 = load %struct.Node** %node10
  %getNodeName116 = call i8* @getNodeName(%struct.Node* %node10115)
  %hash117 = load %struct.hashmap** %hash
  %hashmap_put118 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash117, i8* %getNodeName116, i32 0)
  br label %merge101

else:                                             ; preds = %then
  br label %merge101

else125:                                          ; preds = %while_body
  %l2126 = load %struct.List** %l2
  %get_list_size127 = call i32 @get_list_size(%struct.List* %l2126)
  %tmp128 = sub i32 %get_list_size127, 1
  %l2129 = load %struct.List** %l2
  %remove_list_element = call i8* @remove_list_element(%struct.List* %l2129,
i32 %tmp128)
  %voidTonode130 = call %struct.Node* @voidTonode(i8* %remove_list_element)
  br label %merge

merge134:                                         ; preds = %while
  %rec135 = load %struct.List** %rec
  %get_list_size136 = call i32 @get_list_size(%struct.List* %rec135)
  %rec137 = load %struct.List** %rec
  %get_list_size138 = call i32 @get_list_size(%struct.List* %rec137)
  %printf139 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %get_list_size138)
  %rec140 = load %struct.List** %rec
  %get_list_element141 = call i8* @get_list_element(%struct.List* %rec140,
i32 0)
  %voidTonode142 = call %struct.Node* @voidTonode(i8* %get_list_element141)
  store %struct.Node* %voidTonode142, %struct.Node** %node5
  %node5143 = load %struct.Node** %node5
  %getNodeName144 = call i8* @getNodeName(%struct.Node* %node5143)
  %node5145 = load %struct.Node** %node5
```

```
  %getNodeName146 = call i8* @getNodeName(%struct.Node* %node5145)
  %printf147 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName146)
  %node5148 = load %struct.Node** %node5
  %get_node_value = call i8* @get_node_value(%struct.Node* %node5148)
  %voidToint149 = call i32 @voidToint(i8* %get_node_value)
  %node5150 = load %struct.Node** %node5
  %get_node_value151 = call i8* @get_node_value(%struct.Node* %node5150)
  %voidToint152 = call i32 @voidToint(i8* %get_node_value151)
  %printf153 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint152)
  %rec154 = load %struct.List** %rec
  %get_list_element155 = call i8* @get_list_element(%struct.List* %rec154,
i32 1)
  %voidTonode156 = call %struct.Node* @voidTonode(i8* %get_list_element155)
  store %struct.Node* %voidTonode156, %struct.Node** %node6
  %node6157 = load %struct.Node** %node6
  %getNodeName158 = call i8* @getNodeName(%struct.Node* %node6157)
  %node6159 = load %struct.Node** %node6
  %getNodeName160 = call i8* @getNodeName(%struct.Node* %node6159)
  %printf161 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName160)
  %node6162 = load %struct.Node** %node6
  %get_node_value163 = call i8* @get_node_value(%struct.Node* %node6162)
  %voidToint164 = call i32 @voidToint(i8* %get_node_value163)
  %node6165 = load %struct.Node** %node6
  %get_node_value166 = call i8* @get_node_value(%struct.Node* %node6165)
  %voidToint167 = call i32 @voidToint(i8* %get_node_value166)
  %printf168 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint167)
  %rec169 = load %struct.List** %rec
  %get_list_element170 = call i8* @get_list_element(%struct.List* %rec169,
i32 2)
  %voidTonode171 = call %struct.Node* @voidTonode(i8* %get_list_element170)
  store %struct.Node* %voidTonode171, %struct.Node** %node7
  %node7172 = load %struct.Node** %node7
  %getNodeName173 = call i8* @getNodeName(%struct.Node* %node7172)
  %node7174 = load %struct.Node** %node7
  %getNodeName175 = call i8* @getNodeName(%struct.Node* %node7174)
  %printf176 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName175)
  %node7177 = load %struct.Node** %node7
  %get_node_value178 = call i8* @get_node_value(%struct.Node* %node7177)
  %voidToint179 = call i32 @voidToint(i8* %get_node_value178)
  %node7180 = load %struct.Node** %node7
  %get_node_value181 = call i8* @get_node_value(%struct.Node* %node7180)
  %voidToint182 = call i32 @voidToint(i8* %get_node_value181)
  %printf183 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint182)
  %rec184 = load %struct.List** %rec
  %get_list_element185 = call i8* @get_list_element(%struct.List* %rec184,
i32 3)
  %voidTonode186 = call %struct.Node* @voidTonode(i8* %get_list_element185)
```

```
  store %struct.Node* %voidTonode186, %struct.Node** %node8
  %node8187 = load %struct.Node** %node8
  %getNodeName188 = call i8* @getNodeName(%struct.Node* %node8187)
  %node8189 = load %struct.Node** %node8
  %getNodeName190 = call i8* @getNodeName(%struct.Node* %node8189)
  %printf191 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt1, i32 0, i32 0), i8* %getNodeName190)
  %node8192 = load %struct.Node** %node8
  %get_node_value193 = call i8* @get_node_value(%struct.Node* %node8192)
  %voidToint194 = call i32 @voidToint(i8* %get_node_value193)
  %node8195 = load %struct.Node** %node8
  %get_node_value196 = call i8* @get_node_value(%struct.Node* %node8195)
  %voidToint197 = call i32 @voidToint(i8* %get_node_value196)
  %printf198 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint197)
  ret i32 0
}
```

6.1.2. Example 2: Graph Single Source Shortest Path

Source Language Program:

```
int main(){

    int max, i, size, tmp, now;
    string name;
    node@{int} node1, node2, node3, node4, node5, node6, node7, node8, node9,
node10;
    list@{node@{int}} l;
    list@{node@{int}} queue;
    set@{string} inList;
    map@{string, int} dis;
    graph g;
    new node1; new node2; new node3; new node4; new node5; new node6; new
node7; new node8; new node9; new node10;
    new l; new queue; new inList; new dis; new g;
    node1 -> node2 = 1;
    node2 -> node4 = 2;
    node1 -> node3 = 3;
    node3 -> node4 = 3;
    node1 -> node4 = 1;
    max = 10000;


    /* Implementation of Shortest Path */
    queue+=@{node1};
    inList.put(node1.name());
    dis.put(node1.name(), 0);
    while (queue.length()!=0){
      node9 = queue.get(0);
      size = node9.length();
      now = dis.get(node9.name());
      for (i=0;i<size;i+=1){
```

```
        node10 = node9.iterNode(i);
        tmp = now + node9.weightIter(i);
        if (!dis.haskey(node10.name())){
          dis.put(node10.name(), max);
        }
        if (tmp<dis.get(node10.name())){

          dis.put(node10.name(), tmp);
          if (!inList.contain(node10.name())){
            queue += @{node10};
            inList.put(node10.name());
          }
        }
      }
      inList.remove(node9.name());
      queue++;
    }
    print(dis.get(node1.name()));
    print(dis.get(node2.name()));
    print(dis.get(node3.name()));
    print(dis.get(node4.name()));

}
```

Target Language Program:

```
; ModuleID = 'TuSimple'

%struct.List = type { i32, i32, i8**, i32 }
%struct.hashmap = type { i32, i32, i32, i32, %struct.hashmap_element* }
%struct.hashmap_element = type { i8*, i32, [2 x i8*] }
%struct.Set = type { i32, i32, %struct.List* }
%struct.Node = type { i32, i8*, i8*, %struct.List*, %struct.List* }
%struct.Graph = type { i8*, %struct.List*, %struct.hashmap* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@0 = private unnamed_addr constant [6 x i8] c"node1\00"
@1 = private unnamed_addr constant [6 x i8] c"node2\00"
@2 = private unnamed_addr constant [6 x i8] c"node3\00"
@3 = private unnamed_addr constant [6 x i8] c"node4\00"
@4 = private unnamed_addr constant [6 x i8] c"node5\00"
@5 = private unnamed_addr constant [6 x i8] c"node6\00"
@6 = private unnamed_addr constant [6 x i8] c"node7\00"
@7 = private unnamed_addr constant [6 x i8] c"node8\00"
@8 = private unnamed_addr constant [6 x i8] c"node9\00"
@9 = private unnamed_addr constant [7 x i8] c"node10\00"
@10 = private unnamed_addr constant [2 x i8] c"g\00"

declare i32 @get_list_size(%struct.List*)
```

```
declare i8* @pop_list_element(%struct.List*)

declare i8* @remove_list_element(%struct.List*, i32)

declare i8* @get_list_element(%struct.List*, i32)

declare %struct.List* @concat_list(%struct.List*, %struct.List*, ...)

declare %struct.List* @create_list(i32)

declare %struct.List* @plus_list(%struct.List*, ...)

declare i8* @print_list(%struct.List*)

declare %struct.hashmap* @create_hashmap(i32, i32)

declare %struct.hashmap* @hashmap_put(%struct.hashmap*, ...)

declare i8* @hashmap_get(%struct.hashmap*, ...)

declare i32 @hashmap_length(%struct.hashmap*)

declare i1 @hashmap_haskey(%struct.hashmap*, ...)

declare %struct.hashmap* @hashmap_remove(%struct.hashmap*, ...)

declare %struct.Set* @put_set_from_list(%struct.Set*, %struct.List*)

declare %struct.Set* @create_set(i32)

declare %struct.Set* @put_set(%struct.Set*, ...)

declare i32 @get_set_size(%struct.Set*)

declare i1 @check_set_element(%struct.Set*, ...)

declare %struct.Set* @remove_set_element(%struct.Set*, ...)

declare %struct.Node* @createNode(i8*, i32)

declare i32 @getEdgeValue(%struct.Node*, %struct.Node*)

declare i32 @addNodeEdge(%struct.Node*, %struct.Node*, i32)

declare i32 @addReverseEdge(%struct.Node*, %struct.Node*, i32)

declare i8* @getNodeName(%struct.Node*)

declare %struct.Node* @setNodeValue(%struct.Node*, ...)

declare %struct.Node* @iterNode(%struct.Node*, i32)
```

```
declare i32 @getNodeLength(%struct.Node*)

declare i8* @get_node_value(%struct.Node*)

declare i32 @weightIterNode(%struct.Node*, i32)

declare i32 @graphLength(%struct.Graph*)

declare %struct.Graph* @createGraph(i8*)

declare %struct.Graph* @addGraphNode(%struct.Graph*, %struct.Node*)

declare %struct.Graph*
@addGraphEdge(%struct.Graph*, %struct.Node*, %struct.Node*, i32)

declare %struct.Node* @iterGraph(%struct.Graph*, i32)

declare %struct.Node* @findGraphNode(%struct.Graph*, i8*)

declare %struct.Node* @init_tag(%struct.Graph*)

declare %struct.Node* @reduce(%struct.Graph*, %struct.Node*)

declare %struct.Node* @expand(%struct.Graph*, %struct.Node*)

declare %struct.Graph* @combine(%struct.Graph*, %struct.Graph*)

declare %struct.List* @bfs(%struct.Graph*, %struct.Node*)

declare %struct.List* @dfs(%struct.Graph*, %struct.Node*)

declare i8* @print_graph(%struct.Graph*)

declare i32 @voidToint(i8*)

declare double @voidTofloat(i8*)

declare i1 @voidTobool(i8*)

declare i8* @voidTostring(i8*)

declare %struct.Node* @voidTonode(i8*)

declare %struct.Graph* @voidTograph(i8*)

declare i32 @printf(i8*, ...)

declare i32 @printbig(i32)

define i32 @main() {
entry:
  %now = alloca i32
  %tmp = alloca i32
```

```
  %size = alloca i32
  %i = alloca i32
  %max = alloca i32
  %name = alloca i8*
  %node10 = alloca %struct.Node*
  %node9 = alloca %struct.Node*
  %node8 = alloca %struct.Node*
  %node7 = alloca %struct.Node*
  %node6 = alloca %struct.Node*
  %node5 = alloca %struct.Node*
  %node4 = alloca %struct.Node*
  %node3 = alloca %struct.Node*
  %node2 = alloca %struct.Node*
  %node1 = alloca %struct.Node*
  %l = alloca %struct.List*
  %queue = alloca %struct.List*
  %inList = alloca %struct.Set*
  %dis = alloca %struct.hashmap*
  %g = alloca %struct.Graph*
  %createNode = call %struct.Node* @createNode(i8* getelementptr inbounds ([6
x i8]* @0, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode, %struct.Node** %node1
  %createNode1 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @1, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode1, %struct.Node** %node2
  %createNode2 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @2, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode2, %struct.Node** %node3
  %createNode3 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @3, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode3, %struct.Node** %node4
  %createNode4 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @4, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode4, %struct.Node** %node5
  %createNode5 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @5, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode5, %struct.Node** %node6
  %createNode6 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @6, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode6, %struct.Node** %node7
  %createNode7 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @7, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode7, %struct.Node** %node8
  %createNode8 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @8, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode8, %struct.Node** %node9
  %createNode9 = call %struct.Node* @createNode(i8* getelementptr inbounds
([7 x i8]* @9, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode9, %struct.Node** %node10
  %create_list = call %struct.List* @create_list(i32 4)
  store %struct.List* %create_list, %struct.List** %l
  %create_list10 = call %struct.List* @create_list(i32 4)
  store %struct.List* %create_list10, %struct.List** %queue
```

```
  %create_set = call %struct.Set* @create_set(i32 3)
  store %struct.Set* %create_set, %struct.Set** %inList
  %create_hashmap = call %struct.hashmap* @create_hashmap(i32 3, i32 0)
  store %struct.hashmap* %create_hashmap, %struct.hashmap** %dis
  %createGraph = call %struct.Graph* @createGraph(i8* getelementptr inbounds
([2 x i8]* @10, i32 0, i32 0))
  store %struct.Graph* %createGraph, %struct.Graph** %g
  %node211 = load %struct.Node** %node2
  %node112 = load %struct.Node** %node1
  %addNodeEdge = call i32
@addNodeEdge(%struct.Node* %node112, %struct.Node* %node211, i32 1)
  %node413 = load %struct.Node** %node4
  %node214 = load %struct.Node** %node2
  %addNodeEdge15 = call i32
@addNodeEdge(%struct.Node* %node214, %struct.Node* %node413, i32 2)
  %node316 = load %struct.Node** %node3
  %node117 = load %struct.Node** %node1
  %addNodeEdge18 = call i32
@addNodeEdge(%struct.Node* %node117, %struct.Node* %node316, i32 3)
  %node419 = load %struct.Node** %node4
  %node320 = load %struct.Node** %node3
  %addNodeEdge21 = call i32
@addNodeEdge(%struct.Node* %node320, %struct.Node* %node419, i32 3)
  %node422 = load %struct.Node** %node4
  %node123 = load %struct.Node** %node1
  %addNodeEdge24 = call i32
@addNodeEdge(%struct.Node* %node123, %struct.Node* %node422, i32 1)
  store i32 10000, i32* %max
  %node125 = load %struct.Node** %node1
  %create_list26 = call %struct.List* @create_list(i32 4)
  %node127 = load %struct.Node** %node1
  %plus_list = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list26, %struct.Node* %node127)
  %queue28 = load %struct.List** %queue
  %concat_list = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %queue28, %struct.List* %create_list26)
  store %struct.List* %concat_list, %struct.List** %queue
  %node129 = load %struct.Node** %node1
  %getNodeName = call i8* @getNodeName(%struct.Node* %node129)
  %inList30 = load %struct.Set** %inList
  %put_set = call %struct.Set* (%struct.Set*, ...)*
@put_set(%struct.Set* %inList30, i8* %getNodeName)
  %node131 = load %struct.Node** %node1
  %getNodeName32 = call i8* @getNodeName(%struct.Node* %node131)
  %dis33 = load %struct.hashmap** %dis
  %hashmap_put = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %dis33, i8* %getNodeName32, i32 0)
  br label %while

while:                                            ; preds = %merge93, %entry
  %queue98 = load %struct.List** %queue
  %get_list_size = call i32 @get_list_size(%struct.List* %queue98)
  %tmp99 = icmp ne i32 %get_list_size, 0
```

```
  br i1 %tmp99, label %while_body, label %merge100

while_body:                                       ; preds = %while
  %queue34 = load %struct.List** %queue
  %get_list_element = call i8* @get_list_element(%struct.List* %queue34, i32
0)
  %voidTonode = call %struct.Node* @voidTonode(i8* %get_list_element)
  store %struct.Node* %voidTonode, %struct.Node** %node9
  %node935 = load %struct.Node** %node9
  %getNodeLength = call i32 @getNodeLength(%struct.Node* %node935)
  store i32 %getNodeLength, i32* %size
  %node936 = load %struct.Node** %node9
  %getNodeName37 = call i8* @getNodeName(%struct.Node* %node936)
  %dis38 = load %struct.hashmap** %dis
  %hashmap_get = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis38, i8* %getNodeName37)
  %voidToint = call i32 @voidToint(i8* %hashmap_get)
  store i32 %voidToint, i32* %now
  store i32 0, i32* %i
  br label %while39

while39:                                          ; preds
= %merge63, %while_body
  %i90 = load i32* %i
  %size91 = load i32* %size
  %tmp92 = icmp slt i32 %i90, %size91
  br i1 %tmp92, label %while_body40, label %merge93

while_body40:                                     ; preds = %while39
  %i41 = load i32* %i
  %node942 = load %struct.Node** %node9
  %iterNode = call %struct.Node* @iterNode(%struct.Node* %node942, i32 %i41)
  store %struct.Node* %iterNode, %struct.Node** %node10
  %now43 = load i32* %now
  %i44 = load i32* %i
  %node945 = load %struct.Node** %node9
  %weightIterNode = call i32 @weightIterNode(%struct.Node* %node945,
i32 %i44)
  %tmp46 = add i32 %now43, %weightIterNode
  store i32 %tmp46, i32* %tmp
  %node1047 = load %struct.Node** %node10
  %getNodeName48 = call i8* @getNodeName(%struct.Node* %node1047)
  %dis49 = load %struct.hashmap** %dis
  %hashmap_haskey = call i1 (%struct.hashmap*, ...)*
@hashmap_haskey(%struct.hashmap* %dis49, i8* %getNodeName48)
  %tmp50 = xor i1 %hashmap_haskey, true
  br i1 %tmp50, label %then, label %else

merge:                                            ; preds = %else, %then
  %tmp56 = load i32* %tmp
  %node1057 = load %struct.Node** %node10
  %getNodeName58 = call i8* @getNodeName(%struct.Node* %node1057)
  %dis59 = load %struct.hashmap** %dis
```

```
  %hashmap_get60 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis59, i8* %getNodeName58)
  %voidToint61 = call i32 @voidToint(i8* %hashmap_get60)
  %tmp62 = icmp slt i32 %tmp56, %voidToint61
  br i1 %tmp62, label %then64, label %else87

then:                                             ; preds = %while_body40
  %max51 = load i32* %max
  %node1052 = load %struct.Node** %node10
  %getNodeName53 = call i8* @getNodeName(%struct.Node* %node1052)
  %dis54 = load %struct.hashmap** %dis
  %hashmap_put55 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %dis54, i8* %getNodeName53, i32 %max51)
  br label %merge

else:                                             ; preds = %while_body40
  br label %merge

merge63:                                          ; preds = %else87, %merge74
  %i88 = load i32* %i
  %tmp89 = add i32 %i88, 1
  store i32 %tmp89, i32* %i
  br label %while39

then64:                                           ; preds = %merge
  %tmp65 = load i32* %tmp
  %node1066 = load %struct.Node** %node10
  %getNodeName67 = call i8* @getNodeName(%struct.Node* %node1066)
  %dis68 = load %struct.hashmap** %dis
  %hashmap_put69 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %dis68, i8* %getNodeName67, i32 %tmp65)
  %node1070 = load %struct.Node** %node10
  %getNodeName71 = call i8* @getNodeName(%struct.Node* %node1070)
  %inList72 = load %struct.Set** %inList
  %check_set_element = call i1 (%struct.Set*, ...)*
@check_set_element(%struct.Set* %inList72, i8* %getNodeName71)
  %tmp73 = xor i1 %check_set_element, true
  br i1 %tmp73, label %then75, label %else86

merge74:                                          ; preds = %else86, %then75
  br label %merge63

then75:                                           ; preds = %then64
  %node1076 = load %struct.Node** %node10
  %create_list77 = call %struct.List* @create_list(i32 4)
  %node1078 = load %struct.Node** %node10
  %plus_list79 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list77, %struct.Node* %node1078)
  %queue80 = load %struct.List** %queue
  %concat_list81 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %queue80, %struct.List* %create_list77)
  store %struct.List* %concat_list81, %struct.List** %queue
  %node1082 = load %struct.Node** %node10
```

```
  %getNodeName83 = call i8* @getNodeName(%struct.Node* %node1082)
  %inList84 = load %struct.Set** %inList
  %put_set85 = call %struct.Set* (%struct.Set*, ...)*
@put_set(%struct.Set* %inList84, i8* %getNodeName83)
  br label %merge74

else86:                                           ; preds = %then64
  br label %merge74

else87:                                           ; preds = %merge
  br label %merge63

merge93:                                          ; preds = %while39
  %node994 = load %struct.Node** %node9
  %getNodeName95 = call i8* @getNodeName(%struct.Node* %node994)
  %inList96 = load %struct.Set** %inList
  %remove_set_element = call %struct.Set* (%struct.Set*, ...)*
@remove_set_element(%struct.Set* %inList96, i8* %getNodeName95)
  %queue97 = load %struct.List** %queue
  %remove_list_element = call i8*
@remove_list_element(%struct.List* %queue97, i32 0)
  br label %while

merge100:                                         ; preds = %while
  %node1101 = load %struct.Node** %node1
  %getNodeName102 = call i8* @getNodeName(%struct.Node* %node1101)
  %dis103 = load %struct.hashmap** %dis
  %hashmap_get104 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis103, i8* %getNodeName102)
  %voidToint105 = call i32 @voidToint(i8* %hashmap_get104)
  %node1106 = load %struct.Node** %node1
  %getNodeName107 = call i8* @getNodeName(%struct.Node* %node1106)
  %dis108 = load %struct.hashmap** %dis
  %hashmap_get109 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis108, i8* %getNodeName107)
  %voidToint110 = call i32 @voidToint(i8* %hashmap_get109)
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint110)
  %node2111 = load %struct.Node** %node2
  %getNodeName112 = call i8* @getNodeName(%struct.Node* %node2111)
  %dis113 = load %struct.hashmap** %dis
  %hashmap_get114 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis113, i8* %getNodeName112)
  %voidToint115 = call i32 @voidToint(i8* %hashmap_get114)
  %node2116 = load %struct.Node** %node2
  %getNodeName117 = call i8* @getNodeName(%struct.Node* %node2116)
  %dis118 = load %struct.hashmap** %dis
  %hashmap_get119 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis118, i8* %getNodeName117)
  %voidToint120 = call i32 @voidToint(i8* %hashmap_get119)
  %printf121 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint120)
  %node3122 = load %struct.Node** %node3
```

```
  %getNodeName123 = call i8* @getNodeName(%struct.Node* %node3122)
  %dis124 = load %struct.hashmap** %dis
  %hashmap_get125 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis124, i8* %getNodeName123)
  %voidToint126 = call i32 @voidToint(i8* %hashmap_get125)
  %node3127 = load %struct.Node** %node3
  %getNodeName128 = call i8* @getNodeName(%struct.Node* %node3127)
  %dis129 = load %struct.hashmap** %dis
  %hashmap_get130 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis129, i8* %getNodeName128)
  %voidToint131 = call i32 @voidToint(i8* %hashmap_get130)
  %printf132 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint131)
  %node4133 = load %struct.Node** %node4
  %getNodeName134 = call i8* @getNodeName(%struct.Node* %node4133)
  %dis135 = load %struct.hashmap** %dis
  %hashmap_get136 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis135, i8* %getNodeName134)
  %voidToint137 = call i32 @voidToint(i8* %hashmap_get136)
  %node4138 = load %struct.Node** %node4
  %getNodeName139 = call i8* @getNodeName(%struct.Node* %node4138)
  %dis140 = load %struct.hashmap** %dis
  %hashmap_get141 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %dis140, i8* %getNodeName139)
  %voidToint142 = call i32 @voidToint(i8* %hashmap_get141)
  %printf143 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x
i8]* @fmt, i32 0, i32 0), i32 %voidToint142)
  ret i32 0
}
```

6.1.3. Example 3: Neutral Network XOR Function

Source Language Program:

```
int func(int in){
        if (in>0) return in;
        else return 0;
}

int main(){

        int i, j, size, size_j, now, sum, error, de, update, hidden_to_outer,
count;
        node@{int} node1, node2, node3, node4, node5, node6, node7, node8;
        graph g1, g2, g3;
        map@{string, int} hash;
        list@{int} l, w, update_l, delta_w, update_w;

        new node1;new node2;new node3;new node4;new node5;new node6;new
node7;new node8;
        new g1;new g2;new g3;new hash;
        new l;new w;new update_l;new delta_w;new update_w;
```

```
        node1 -> @{node3, node4, node5} = @{8, 4, 3};

        node2 -> @{node3, node4, node5} = @{2, 9, 5};

        node3 -> node6 = 3;
        node4 -> node6 = 5;
        node5 -> node6 = 9;

        w += @{8, 4, 3, 2, 9, 5, 3, 5, 9};
        w.printList();

        g1.addNode(node1);
        g1.addNode(node2);

        g2.addNode(node3);
        g2.addNode(node4);
        g2.addNode(node5);

        g3.addNode(node6);

        hidden_to_outer = g1.length() * g2.length();

        size = g1.length();
        for (i=0;i<size;i+=1){
                node7 = g1.iterGraph(i);
                size_j = node7.length();
                for (j=0;j<size_j;j+=1){
                        node8 = node7.iterNode(j);
                        if (hash.haskey(node8.name())){
                                now = i * g2.length() + j;
                                hash.put(node8.name(), hash.get(node8.name()) +
w.get(now));
                        }
                        else {
                                now = i * g2.length() + j;
                                hash.put(node8.name(), w.get(now));
                        }
                }
        }

        size = g2.length();
        for (i=0;i<size;i+=1){
                node7 = g2.iterGraph(i);
                now = func(hash.get(node7.name()));
                l += @{now};
        }

        sum = 0;
        size = g2.length();
        for (i=0;i<size;i+=1){
                node7 = g2.iterGraph(i);
                sum += l.get(i) * w.get(hidden_to_outer+i) / 10;
```

```
}

error = 0 - func(sum);
de = ((func(sum+1) - func(sum))/1 + (func(sum) - func(sum-1))/1)/2;
sum = de * error / 10;

for(i=0;i<size;i+=1){
        update = sum * w.get(hidden_to_outer+i) * 1;
        delta_w += @{update};
}

count = 0;
for (i=0;i<hidden_to_outer;i+=1){
        update = w.get(i) + delta_w.get(count);
        update_w += @{update};

        count += 1;
        if (count==g2.length()) count = 0;
}

size = g2.length();
for (i=0;i<size;i+=1){
        node7 = g2.iterGraph(i);
        update = w.get(hidden_to_outer+i) + sum * l.get(i) / 10;
        update_l += @{update};
}
// update_l.printList();
update_w.cancat(update_l);

w = update_w;
w.printList();

/*
         second iteration
*/

new l;new update_l;new delta_w;new update_w;

size = g2.length();
for (i=0;i<size;i+=1){
        node7 = g2.iterGraph(i);
        hash.remove(node7.name());
}

size = g1.length();
for (i=0;i<size;i+=1){
        node7 = g1.iterGraph(i);
        size_j = node7.length();
        for (j=0;j<size_j;j+=1){
                node8 = node7.iterNode(j);
                if (hash.haskey(node8.name())){
                        now = i * g2.length() + j;
```

```
                                hash.put(node8.name(), hash.get(node8.name()) +
w.get(now));
                        }
                        else {
                                now = i * g2.length() + j;
                                hash.put(node8.name(), w.get(now));
                        }
                }
        }

        size = g2.length();
        for (i=0;i<size;i+=1){
                node7 = g2.iterGraph(i);
                now = func(hash.get(node7.name()));
                l += @{now};
        }

        sum = 0;
        size = g2.length();
        for (i=0;i<size;i+=1){
                node7 = g2.iterGraph(i);
                sum += l.get(i) * w.get(hidden_to_outer+i) / 10;

        }

        error = 0 - func(sum);
        de = ((func(sum+1) - func(sum))/1 + (func(sum) - func(sum-1))/1)/2;
        sum = de * error / 10;

        for(i=0;i<size;i+=1){
                update = sum * w.get(hidden_to_outer+i) * 1;
                delta_w += @{update};
        }

        count = 0;
        for (i=0;i<hidden_to_outer;i+=1){
                update = w.get(i) + delta_w.get(count);
                update_w += @{update};

                count += 1;
                if (count==g2.length()) count = 0;
        }

        size = g2.length();
        for (i=0;i<size;i+=1){
                node7 = g2.iterGraph(i);
                update = w.get(hidden_to_outer+i) + sum * l.get(i) / 10;
                update_l += @{update};
        }
        // update_l.printList();
        update_w.cancat(update_l);

        w = update_w;
```

```
        w.printList();

}
```

Target Language Program:

```
; ModuleID = 'TuSimple'

%struct.List = type { i32, i32, i8**, i32 }
%struct.hashmap = type { i32, i32, i32, i32, %struct.hashmap_element* }
%struct.hashmap_element = type { i8*, i32, [2 x i8*] }
%struct.Set = type { i32, i32, %struct.List* }
%struct.Node = type { i32, i8*, i8*, %struct.List*, %struct.List* }
%struct.Graph = type { i8*, %struct.List*, %struct.hashmap* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@0 = private unnamed_addr constant [6 x i8] c"node1\00"
@1 = private unnamed_addr constant [6 x i8] c"node2\00"
@2 = private unnamed_addr constant [6 x i8] c"node3\00"
@3 = private unnamed_addr constant [6 x i8] c"node4\00"
@4 = private unnamed_addr constant [6 x i8] c"node5\00"
@5 = private unnamed_addr constant [6 x i8] c"node6\00"
@6 = private unnamed_addr constant [6 x i8] c"node7\00"
@7 = private unnamed_addr constant [6 x i8] c"node8\00"
@8 = private unnamed_addr constant [3 x i8] c"g1\00"
@9 = private unnamed_addr constant [3 x i8] c"g2\00"
@10 = private unnamed_addr constant [3 x i8] c"g3\00"
@fmt2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare i32 @get_list_size(%struct.List*)

declare i8* @pop_list_element(%struct.List*)

declare i8* @remove_list_element(%struct.List*, i32)

declare i8* @get_list_element(%struct.List*, i32)

declare %struct.List* @concat_list(%struct.List*, %struct.List*, ...)

declare %struct.List* @create_list(i32)

declare %struct.List* @plus_list(%struct.List*, ...)

declare i8* @print_list(%struct.List*)

declare %struct.hashmap* @create_hashmap(i32, i32)

declare %struct.hashmap* @hashmap_put(%struct.hashmap*, ...)
```

```
declare i8* @hashmap_get(%struct.hashmap*, ...)

declare i32 @hashmap_length(%struct.hashmap*)

declare i1 @hashmap_haskey(%struct.hashmap*, ...)

declare %struct.hashmap* @hashmap_remove(%struct.hashmap*, ...)

declare %struct.Set* @put_set_from_list(%struct.Set*, %struct.List*)

declare %struct.Set* @create_set(i32)

declare %struct.Set* @put_set(%struct.Set*, ...)

declare i32 @get_set_size(%struct.Set*)

declare i1 @check_set_element(%struct.Set*, ...)

declare %struct.Set* @remove_set_element(%struct.Set*, ...)

declare %struct.Node* @createNode(i8*, i32)

declare i32 @getEdgeValue(%struct.Node*, %struct.Node*)

declare i32 @addNodeEdge(%struct.Node*, %struct.Node*, i32)

declare i32 @addReverseEdge(%struct.Node*, %struct.Node*, i32)

declare i8* @getNodeName(%struct.Node*)

declare %struct.Node* @setNodeValue(%struct.Node*, ...)

declare %struct.Node* @iterNode(%struct.Node*, i32)

declare i32 @getNodeLength(%struct.Node*)

declare i8* @get_node_value(%struct.Node*)

declare i32 @weightIterNode(%struct.Node*, i32)

declare i32 @graphLength(%struct.Graph*)

declare %struct.Graph* @createGraph(i8*)

declare %struct.Graph* @addGraphNode(%struct.Graph*, %struct.Node*)

declare %struct.Graph*
@addGraphEdge(%struct.Graph*, %struct.Node*, %struct.Node*, i32)

declare %struct.Node* @iterGraph(%struct.Graph*, i32)

declare %struct.Node* @findGraphNode(%struct.Graph*, i8*)
```

```
declare %struct.Node* @init_tag(%struct.Graph*)

declare %struct.Node* @reduce(%struct.Graph*, %struct.Node*)

declare %struct.Node* @expand(%struct.Graph*, %struct.Node*)

declare %struct.Graph* @combine(%struct.Graph*, %struct.Graph*)

declare %struct.List* @bfs(%struct.Graph*, %struct.Node*)

declare %struct.List* @dfs(%struct.Graph*, %struct.Node*)

declare i8* @print_graph(%struct.Graph*)

declare i32 @voidToint(i8*)

declare double @voidTofloat(i8*)

declare i1 @voidTobool(i8*)

declare i8* @voidTostring(i8*)

declare %struct.Node* @voidTonode(i8*)

declare %struct.Graph* @voidTograph(i8*)

declare i32 @printf(i8*, ...)

declare i32 @printbig(i32)

define i32 @main() {
entry:
  %count = alloca i32
  %hidden_to_outer = alloca i32
  %update = alloca i32
  %de = alloca i32
  %error = alloca i32
  %sum = alloca i32
  %now = alloca i32
  %size_j = alloca i32
  %size = alloca i32
  %j = alloca i32
  %i = alloca i32
  %node8 = alloca %struct.Node*
  %node7 = alloca %struct.Node*
  %node6 = alloca %struct.Node*
  %node5 = alloca %struct.Node*
  %node4 = alloca %struct.Node*
  %node3 = alloca %struct.Node*
  %node2 = alloca %struct.Node*
  %node1 = alloca %struct.Node*
  %g3 = alloca %struct.Graph*
```

```
  %g2 = alloca %struct.Graph*
  %g1 = alloca %struct.Graph*
  %hash = alloca %struct.hashmap*
  %update_w = alloca %struct.List*
  %delta_w = alloca %struct.List*
  %update_l = alloca %struct.List*
  %w = alloca %struct.List*
  %l = alloca %struct.List*
  %createNode = call %struct.Node* @createNode(i8* getelementptr inbounds ([6
x i8]* @0, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode, %struct.Node** %node1
  %createNode1 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @1, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode1, %struct.Node** %node2
  %createNode2 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @2, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode2, %struct.Node** %node3
  %createNode3 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @3, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode3, %struct.Node** %node4
  %createNode4 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @4, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode4, %struct.Node** %node5
  %createNode5 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @5, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode5, %struct.Node** %node6
  %createNode6 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @6, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode6, %struct.Node** %node7
  %createNode7 = call %struct.Node* @createNode(i8* getelementptr inbounds
([6 x i8]* @7, i32 0, i32 0), i32 0)
  store %struct.Node* %createNode7, %struct.Node** %node8
  %createGraph = call %struct.Graph* @createGraph(i8* getelementptr inbounds
([3 x i8]* @8, i32 0, i32 0))
  store %struct.Graph* %createGraph, %struct.Graph** %g1
  %createGraph8 = call %struct.Graph* @createGraph(i8* getelementptr inbounds
([3 x i8]* @9, i32 0, i32 0))
  store %struct.Graph* %createGraph8, %struct.Graph** %g2
  %createGraph9 = call %struct.Graph* @createGraph(i8* getelementptr inbounds
([3 x i8]* @10, i32 0, i32 0))
  store %struct.Graph* %createGraph9, %struct.Graph** %g3
  %create_hashmap = call %struct.hashmap* @create_hashmap(i32 3, i32 0)
  store %struct.hashmap* %create_hashmap, %struct.hashmap** %hash
  %create_list = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list, %struct.List** %l
  %create_list10 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list10, %struct.List** %w
  %create_list11 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list11, %struct.List** %update_l
  %create_list12 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list12, %struct.List** %delta_w
  %create_list13 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list13, %struct.List** %update_w
```

```
  %node314 = load %struct.Node** %node3
  %node415 = load %struct.Node** %node4
  %node516 = load %struct.Node** %node5
  %node117 = load %struct.Node** %node1
  %addNodeEdge = call i32
@addNodeEdge(%struct.Node* %node117, %struct.Node* %node314, i32 8)
  %node118 = load %struct.Node** %node1
  %addNodeEdge19 = call i32
@addNodeEdge(%struct.Node* %node118, %struct.Node* %node415, i32 4)
  %node120 = load %struct.Node** %node1
  %addNodeEdge21 = call i32
@addNodeEdge(%struct.Node* %node120, %struct.Node* %node516, i32 3)
  %node322 = load %struct.Node** %node3
  %node423 = load %struct.Node** %node4
  %node524 = load %struct.Node** %node5
  %node225 = load %struct.Node** %node2
  %addNodeEdge26 = call i32
@addNodeEdge(%struct.Node* %node225, %struct.Node* %node322, i32 2)
  %node227 = load %struct.Node** %node2
  %addNodeEdge28 = call i32
@addNodeEdge(%struct.Node* %node227, %struct.Node* %node423, i32 9)
  %node229 = load %struct.Node** %node2
  %addNodeEdge30 = call i32
@addNodeEdge(%struct.Node* %node229, %struct.Node* %node524, i32 5)
  %node631 = load %struct.Node** %node6
  %node332 = load %struct.Node** %node3
  %addNodeEdge33 = call i32
@addNodeEdge(%struct.Node* %node332, %struct.Node* %node631, i32 3)
  %node634 = load %struct.Node** %node6
  %node435 = load %struct.Node** %node4
  %addNodeEdge36 = call i32
@addNodeEdge(%struct.Node* %node435, %struct.Node* %node634, i32 5)
  %node637 = load %struct.Node** %node6
  %node538 = load %struct.Node** %node5
  %addNodeEdge39 = call i32
@addNodeEdge(%struct.Node* %node538, %struct.Node* %node637, i32 9)
  %create_list40 = call %struct.List* @create_list(i32 0)
  %plus_list = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 8)
  %plus_list41 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 4)
  %plus_list42 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 3)
  %plus_list43 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 2)
  %plus_list44 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 9)
  %plus_list45 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 5)
  %plus_list46 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 3)
  %plus_list47 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 5)
```

```
  %plus_list48 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list40, i32 9)
  %w49 = load %struct.List** %w
  %concat_list = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %w49, %struct.List* %create_list40)
  store %struct.List* %concat_list, %struct.List** %w
  %w50 = load %struct.List** %w
  %print_list = call i8* @print_list(%struct.List* %w50)
  %node151 = load %struct.Node** %node1
  %g152 = load %struct.Graph** %g1
  %addGraphNode = call %struct.Graph*
@addGraphNode(%struct.Graph* %g152, %struct.Node* %node151)
  %node253 = load %struct.Node** %node2
  %g154 = load %struct.Graph** %g1
  %addGraphNode55 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g154, %struct.Node* %node253)
  %node356 = load %struct.Node** %node3
  %g257 = load %struct.Graph** %g2
  %addGraphNode58 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g257, %struct.Node* %node356)
  %node459 = load %struct.Node** %node4
  %g260 = load %struct.Graph** %g2
  %addGraphNode61 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g260, %struct.Node* %node459)
  %node562 = load %struct.Node** %node5
  %g263 = load %struct.Graph** %g2
  %addGraphNode64 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g263, %struct.Node* %node562)
  %node665 = load %struct.Node** %node6
  %g366 = load %struct.Graph** %g3
  %addGraphNode67 = call %struct.Graph*
@addGraphNode(%struct.Graph* %g366, %struct.Node* %node665)
  %g168 = load %struct.Graph** %g1
  %graphLength = call i32 @graphLength(%struct.Graph* %g168)
  %g269 = load %struct.Graph** %g2
  %graphLength70 = call i32 @graphLength(%struct.Graph* %g269)
  %tmp = mul i32 %graphLength, %graphLength70
  store i32 %tmp, i32* %hidden_to_outer
  %g171 = load %struct.Graph** %g1
  %graphLength72 = call i32 @graphLength(%struct.Graph* %g171)
  store i32 %graphLength72, i32* %size
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge117, %entry
  %i120 = load i32* %i
  %size121 = load i32* %size
  %tmp122 = icmp slt i32 %i120, %size121
  br i1 %tmp122, label %while_body, label %merge123

while_body:                                       ; preds = %while
  %i73 = load i32* %i
  %g174 = load %struct.Graph** %g1
```

```
  %iterGraph = call %struct.Node* @iterGraph(%struct.Graph* %g174, i32 %i73)
  store %struct.Node* %iterGraph, %struct.Node** %node7
  %node775 = load %struct.Node** %node7
  %getNodeLength = call i32 @getNodeLength(%struct.Node* %node775)
  store i32 %getNodeLength, i32* %size_j
  store i32 0, i32* %j
  br label %while76

while76:                                          ; preds
= %merge, %while_body
  %j114 = load i32* %j
  %size_j115 = load i32* %size_j
  %tmp116 = icmp slt i32 %j114, %size_j115
  br i1 %tmp116, label %while_body77, label %merge117

while_body77:                                     ; preds = %while76
  %j78 = load i32* %j
  %node779 = load %struct.Node** %node7
  %iterNode = call %struct.Node* @iterNode(%struct.Node* %node779, i32 %j78)
  store %struct.Node* %iterNode, %struct.Node** %node8
  %node880 = load %struct.Node** %node8
  %getNodeName = call i8* @getNodeName(%struct.Node* %node880)
  %hash81 = load %struct.hashmap** %hash
  %hashmap_haskey = call i1 (%struct.hashmap*, ...)*
@hashmap_haskey(%struct.hashmap* %hash81, i8* %getNodeName)
  br i1 %hashmap_haskey, label %then, label %else

merge:                                            ; preds = %else, %then
  %j112 = load i32* %j
  %tmp113 = add i32 %j112, 1
  store i32 %tmp113, i32* %j
  br label %while76

then:                                             ; preds = %while_body77
  %i82 = load i32* %i
  %g283 = load %struct.Graph** %g2
  %graphLength84 = call i32 @graphLength(%struct.Graph* %g283)
  %tmp85 = mul i32 %i82, %graphLength84
  %j86 = load i32* %j
  %tmp87 = add i32 %tmp85, %j86
  store i32 %tmp87, i32* %now
  %node888 = load %struct.Node** %node8
  %getNodeName89 = call i8* @getNodeName(%struct.Node* %node888)
  %hash90 = load %struct.hashmap** %hash
  %hashmap_get = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %hash90, i8* %getNodeName89)
  %voidToint = call i32 @voidToint(i8* %hashmap_get)
  %now91 = load i32* %now
  %w92 = load %struct.List** %w
  %get_list_element = call i8* @get_list_element(%struct.List* %w92,
i32 %now91)
  %voidToint93 = call i32 @voidToint(i8* %get_list_element)
  %tmp94 = add i32 %voidToint, %voidToint93
```

```
  %node895 = load %struct.Node** %node8
  %getNodeName96 = call i8* @getNodeName(%struct.Node* %node895)
  %hash97 = load %struct.hashmap** %hash
  %hashmap_put = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash97, i8* %getNodeName96, i32 %tmp94)
  br label %merge

else:                                             ; preds = %while_body77
  %i98 = load i32* %i
  %g299 = load %struct.Graph** %g2
  %graphLength100 = call i32 @graphLength(%struct.Graph* %g299)
  %tmp101 = mul i32 %i98, %graphLength100
  %j102 = load i32* %j
  %tmp103 = add i32 %tmp101, %j102
  store i32 %tmp103, i32* %now
  %now104 = load i32* %now
  %w105 = load %struct.List** %w
  %get_list_element106 = call i8* @get_list_element(%struct.List* %w105,
i32 %now104)
  %voidToint107 = call i32 @voidToint(i8* %get_list_element106)
  %node8108 = load %struct.Node** %node8
  %getNodeName109 = call i8* @getNodeName(%struct.Node* %node8108)
  %hash110 = load %struct.hashmap** %hash
  %hashmap_put111 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash110, i8* %getNodeName109,
i32 %voidToint107)
  br label %merge

merge117:                                         ; preds = %while76
  %i118 = load i32* %i
  %tmp119 = add i32 %i118, 1
  store i32 %tmp119, i32* %i
  br label %while

merge123:                                         ; preds = %while
  %g2124 = load %struct.Graph** %g2
  %graphLength125 = call i32 @graphLength(%struct.Graph* %g2124)
  store i32 %graphLength125, i32* %size
  store i32 0, i32* %i
  br label %while126

while126:                                         ; preds
= %while_body127, %merge123
  %i144 = load i32* %i
  %size145 = load i32* %size
  %tmp146 = icmp slt i32 %i144, %size145
  br i1 %tmp146, label %while_body127, label %merge147

while_body127:                                    ; preds = %while126
  %i128 = load i32* %i
  %g2129 = load %struct.Graph** %g2
  %iterGraph130 = call %struct.Node* @iterGraph(%struct.Graph* %g2129,
i32 %i128)
```

```
  store %struct.Node* %iterGraph130, %struct.Node** %node7
  %node7131 = load %struct.Node** %node7
  %getNodeName132 = call i8* @getNodeName(%struct.Node* %node7131)
  %hash133 = load %struct.hashmap** %hash
  %hashmap_get134 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %hash133, i8* %getNodeName132)
  %voidToint135 = call i32 @voidToint(i8* %hashmap_get134)
  %func_result = call i32 @func(i32 %voidToint135)
  store i32 %func_result, i32* %now
  %now136 = load i32* %now
  %create_list137 = call %struct.List* @create_list(i32 0)
  %now138 = load i32* %now
  %plus_list139 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list137, i32 %now138)
  %l140 = load %struct.List** %l
  %concat_list141 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %l140, %struct.List* %create_list137)
  store %struct.List* %concat_list141, %struct.List** %l
  %i142 = load i32* %i
  %tmp143 = add i32 %i142, 1
  store i32 %tmp143, i32* %i
  br label %while126

merge147:                                         ; preds = %while126
  store i32 0, i32* %sum
  %g2148 = load %struct.Graph** %g2
  %graphLength149 = call i32 @graphLength(%struct.Graph* %g2148)
  store i32 %graphLength149, i32* %size
  store i32 0, i32* %i
  br label %while150

while150:                                         ; preds
= %while_body151, %merge147
  %i183 = load i32* %i
  %size184 = load i32* %size
  %tmp185 = icmp slt i32 %i183, %size184
  br i1 %tmp185, label %while_body151, label %merge186

while_body151:                                    ; preds = %while150
  %i152 = load i32* %i
  %g2153 = load %struct.Graph** %g2
  %iterGraph154 = call %struct.Node* @iterGraph(%struct.Graph* %g2153,
i32 %i152)
  store %struct.Node* %iterGraph154, %struct.Node** %node7
  %i155 = load i32* %i
  %l156 = load %struct.List** %l
  %get_list_element157 = call i8* @get_list_element(%struct.List* %l156,
i32 %i155)
  %voidToint158 = call i32 @voidToint(i8* %get_list_element157)
  %hidden_to_outer159 = load i32* %hidden_to_outer
  %i160 = load i32* %i
  %tmp161 = add i32 %hidden_to_outer159, %i160
  %w162 = load %struct.List** %w
```

```
  %get_list_element163 = call i8* @get_list_element(%struct.List* %w162,
i32 %tmp161)
  %voidToint164 = call i32 @voidToint(i8* %get_list_element163)
  %tmp165 = mul i32 %voidToint158, %voidToint164
  %tmp166 = sdiv i32 %tmp165, 10
  %sum167 = load i32* %sum
  %i168 = load i32* %i
  %l169 = load %struct.List** %l
  %get_list_element170 = call i8* @get_list_element(%struct.List* %l169,
i32 %i168)
  %voidToint171 = call i32 @voidToint(i8* %get_list_element170)
  %hidden_to_outer172 = load i32* %hidden_to_outer
  %i173 = load i32* %i
  %tmp174 = add i32 %hidden_to_outer172, %i173
  %w175 = load %struct.List** %w
  %get_list_element176 = call i8* @get_list_element(%struct.List* %w175,
i32 %tmp174)
  %voidToint177 = call i32 @voidToint(i8* %get_list_element176)
  %tmp178 = mul i32 %voidToint171, %voidToint177
  %tmp179 = sdiv i32 %tmp178, 10
  %tmp180 = add i32 %sum167, %tmp179
  store i32 %tmp180, i32* %sum
  %i181 = load i32* %i
  %tmp182 = add i32 %i181, 1
  store i32 %tmp182, i32* %i
  br label %while150

merge186:                                         ; preds = %while150
  %sum187 = load i32* %sum
  %func_result188 = call i32 @func(i32 %sum187)
  %tmp189 = sub i32 0, %func_result188
  store i32 %tmp189, i32* %error
  %sum190 = load i32* %sum
  %tmp191 = add i32 %sum190, 1
  %func_result192 = call i32 @func(i32 %tmp191)
  %sum193 = load i32* %sum
  %func_result194 = call i32 @func(i32 %sum193)
  %tmp195 = sub i32 %func_result192, %func_result194
  %tmp196 = sdiv i32 %tmp195, 1
  %sum197 = load i32* %sum
  %func_result198 = call i32 @func(i32 %sum197)
  %sum199 = load i32* %sum
  %tmp200 = sub i32 %sum199, 1
  %func_result201 = call i32 @func(i32 %tmp200)
  %tmp202 = sub i32 %func_result198, %func_result201
  %tmp203 = sdiv i32 %tmp202, 1
  %tmp204 = add i32 %tmp196, %tmp203
  %tmp205 = sdiv i32 %tmp204, 2
  store i32 %tmp205, i32* %de
  %de206 = load i32* %de
  %error207 = load i32* %error
  %tmp208 = mul i32 %de206, %error207
  %tmp209 = sdiv i32 %tmp208, 10
```

```
  store i32 %tmp209, i32* %sum
  store i32 0, i32* %i
  br label %while210

while210:                                         ; preds
= %while_body211, %merge186
  %i229 = load i32* %i
  %size230 = load i32* %size
  %tmp231 = icmp slt i32 %i229, %size230
  br i1 %tmp231, label %while_body211, label %merge232

while_body211:                                    ; preds = %while210
  %sum212 = load i32* %sum
  %hidden_to_outer213 = load i32* %hidden_to_outer
  %i214 = load i32* %i
  %tmp215 = add i32 %hidden_to_outer213, %i214
  %w216 = load %struct.List** %w
  %get_list_element217 = call i8* @get_list_element(%struct.List* %w216,
i32 %tmp215)
  %voidToint218 = call i32 @voidToint(i8* %get_list_element217)
  %tmp219 = mul i32 %sum212, %voidToint218
  %tmp220 = mul i32 %tmp219, 1
  store i32 %tmp220, i32* %update
  %update221 = load i32* %update
  %create_list222 = call %struct.List* @create_list(i32 0)
  %update223 = load i32* %update
  %plus_list224 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list222, i32 %update223)
  %delta_w225 = load %struct.List** %delta_w
  %concat_list226 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %delta_w225, %struct.List* %create_list222)
  store %struct.List* %concat_list226, %struct.List** %delta_w
  %i227 = load i32* %i
  %tmp228 = add i32 %i227, 1
  store i32 %tmp228, i32* %i
  br label %while210

merge232:                                         ; preds = %while210
  store i32 0, i32* %count
  store i32 0, i32* %i
  br label %while233

while233:                                         ; preds
= %merge256, %merge232
  %i261 = load i32* %i
  %hidden_to_outer262 = load i32* %hidden_to_outer
  %tmp263 = icmp slt i32 %i261, %hidden_to_outer262
  br i1 %tmp263, label %while_body234, label %merge264

while_body234:                                    ; preds = %while233
  %i235 = load i32* %i
  %w236 = load %struct.List** %w
```

```
  %get_list_element237 = call i8* @get_list_element(%struct.List* %w236,
i32 %i235)
  %voidToint238 = call i32 @voidToint(i8* %get_list_element237)
  %count239 = load i32* %count
  %delta_w240 = load %struct.List** %delta_w
  %get_list_element241 = call i8*
@get_list_element(%struct.List* %delta_w240, i32 %count239)
  %voidToint242 = call i32 @voidToint(i8* %get_list_element241)
  %tmp243 = add i32 %voidToint238, %voidToint242
  store i32 %tmp243, i32* %update
  %update244 = load i32* %update
  %create_list245 = call %struct.List* @create_list(i32 0)
  %update246 = load i32* %update
  %plus_list247 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list245, i32 %update246)
  %update_w248 = load %struct.List** %update_w
  %concat_list249 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_w248, %struct.List* %create_list245)
  store %struct.List* %concat_list249, %struct.List** %update_w
  %count250 = load i32* %count
  %tmp251 = add i32 %count250, 1
  store i32 %tmp251, i32* %count
  %count252 = load i32* %count
  %g2253 = load %struct.Graph** %g2
  %graphLength254 = call i32 @graphLength(%struct.Graph* %g2253)
  %tmp255 = icmp eq i32 %count252, %graphLength254
  br i1 %tmp255, label %then257, label %else258

merge256:                                         ; preds
= %else258, %then257
  %i259 = load i32* %i
  %tmp260 = add i32 %i259, 1
  store i32 %tmp260, i32* %i
  br label %while233

then257:                                          ; preds = %while_body234
  store i32 0, i32* %count
  br label %merge256

else258:                                          ; preds = %while_body234
  br label %merge256

merge264:                                         ; preds = %while233
  %g2265 = load %struct.Graph** %g2
  %graphLength266 = call i32 @graphLength(%struct.Graph* %g2265)
  store i32 %graphLength266, i32* %size
  store i32 0, i32* %i
  br label %while267

while267:                                         ; preds
= %while_body268, %merge264
  %i294 = load i32* %i
  %size295 = load i32* %size
```

```
  %tmp296 = icmp slt i32 %i294, %size295
  br i1 %tmp296, label %while_body268, label %merge297

while_body268:                                    ; preds = %while267
  %i269 = load i32* %i
  %g2270 = load %struct.Graph** %g2
  %iterGraph271 = call %struct.Node* @iterGraph(%struct.Graph* %g2270,
i32 %i269)
  store %struct.Node* %iterGraph271, %struct.Node** %node7
  %hidden_to_outer272 = load i32* %hidden_to_outer
  %i273 = load i32* %i
  %tmp274 = add i32 %hidden_to_outer272, %i273
  %w275 = load %struct.List** %w
  %get_list_element276 = call i8* @get_list_element(%struct.List* %w275,
i32 %tmp274)
  %voidToint277 = call i32 @voidToint(i8* %get_list_element276)
  %sum278 = load i32* %sum
  %i279 = load i32* %i
  %l280 = load %struct.List** %l
  %get_list_element281 = call i8* @get_list_element(%struct.List* %l280,
i32 %i279)
  %voidToint282 = call i32 @voidToint(i8* %get_list_element281)
  %tmp283 = mul i32 %sum278, %voidToint282
  %tmp284 = sdiv i32 %tmp283, 10
  %tmp285 = add i32 %voidToint277, %tmp284
  store i32 %tmp285, i32* %update
  %update286 = load i32* %update
  %create_list287 = call %struct.List* @create_list(i32 0)
  %update288 = load i32* %update
  %plus_list289 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list287, i32 %update288)
  %update_l290 = load %struct.List** %update_l
  %concat_list291 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_l290, %struct.List* %create_list287)
  store %struct.List* %concat_list291, %struct.List** %update_l
  %i292 = load i32* %i
  %tmp293 = add i32 %i292, 1
  store i32 %tmp293, i32* %i
  br label %while267

merge297:                                         ; preds = %while267
  %update_l298 = load %struct.List** %update_l
  %update_w299 = load %struct.List** %update_w
  %concat_list300 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_w299, %struct.List* %update_l298)
  %update_w301 = load %struct.List** %update_w
  store %struct.List* %update_w301, %struct.List** %w
  %w302 = load %struct.List** %w
  %print_list303 = call i8* @print_list(%struct.List* %w302)
  %create_list304 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list304, %struct.List** %l
  %create_list305 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list305, %struct.List** %update_l
```

```
  %create_list306 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list306, %struct.List** %delta_w
  %create_list307 = call %struct.List* @create_list(i32 0)
  store %struct.List* %create_list307, %struct.List** %update_w
  %g2308 = load %struct.Graph** %g2
  %graphLength309 = call i32 @graphLength(%struct.Graph* %g2308)
  store i32 %graphLength309, i32* %size
  store i32 0, i32* %i
  br label %while310

while310:                                         ; preds
= %while_body311, %merge297
  %i320 = load i32* %i
  %size321 = load i32* %size
  %tmp322 = icmp slt i32 %i320, %size321
  br i1 %tmp322, label %while_body311, label %merge323

while_body311:                                    ; preds = %while310
  %i312 = load i32* %i
  %g2313 = load %struct.Graph** %g2
  %iterGraph314 = call %struct.Node* @iterGraph(%struct.Graph* %g2313,
i32 %i312)
  store %struct.Node* %iterGraph314, %struct.Node** %node7
  %node7315 = load %struct.Node** %node7
  %getNodeName316 = call i8* @getNodeName(%struct.Node* %node7315)
  %hash317 = load %struct.hashmap** %hash
  %hashmap_remove = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_remove(%struct.hashmap* %hash317, i8* %getNodeName316)
  %i318 = load i32* %i
  %tmp319 = add i32 %i318, 1
  store i32 %tmp319, i32* %i
  br label %while310

merge323:                                         ; preds = %while310
  %g1324 = load %struct.Graph** %g1
  %graphLength325 = call i32 @graphLength(%struct.Graph* %g1324)
  store i32 %graphLength325, i32* %size
  store i32 0, i32* %i
  br label %while326

while326:                                         ; preds
= %merge384, %merge323
  %i387 = load i32* %i
  %size388 = load i32* %size
  %tmp389 = icmp slt i32 %i387, %size388
  br i1 %tmp389, label %while_body327, label %merge390

while_body327:                                    ; preds = %while326
  %i328 = load i32* %i
  %g1329 = load %struct.Graph** %g1
  %iterGraph330 = call %struct.Node* @iterGraph(%struct.Graph* %g1329,
i32 %i328)
  store %struct.Node* %iterGraph330, %struct.Node** %node7
```

```
  %node7331 = load %struct.Node** %node7
  %getNodeLength332 = call i32 @getNodeLength(%struct.Node* %node7331)
  store i32 %getNodeLength332, i32* %size_j
  store i32 0, i32* %j
  br label %while333

while333:                                         ; preds
= %merge342, %while_body327
  %j381 = load i32* %j
  %size_j382 = load i32* %size_j
  %tmp383 = icmp slt i32 %j381, %size_j382
  br i1 %tmp383, label %while_body334, label %merge384

while_body334:                                    ; preds = %while333
  %j335 = load i32* %j
  %node7336 = load %struct.Node** %node7
  %iterNode337 = call %struct.Node* @iterNode(%struct.Node* %node7336,
i32 %j335)
  store %struct.Node* %iterNode337, %struct.Node** %node8
  %node8338 = load %struct.Node** %node8
  %getNodeName339 = call i8* @getNodeName(%struct.Node* %node8338)
  %hash340 = load %struct.hashmap** %hash
  %hashmap_haskey341 = call i1 (%struct.hashmap*, ...)*
@hashmap_haskey(%struct.hashmap* %hash340, i8* %getNodeName339)
  br i1 %hashmap_haskey341, label %then343, label %else364

merge342:                                         ; preds
= %else364, %then343
  %j379 = load i32* %j
  %tmp380 = add i32 %j379, 1
  store i32 %tmp380, i32* %j
  br label %while333

then343:                                          ; preds = %while_body334
  %i344 = load i32* %i
  %g2345 = load %struct.Graph** %g2
  %graphLength346 = call i32 @graphLength(%struct.Graph* %g2345)
  %tmp347 = mul i32 %i344, %graphLength346
  %j348 = load i32* %j
  %tmp349 = add i32 %tmp347, %j348
  store i32 %tmp349, i32* %now
  %node8350 = load %struct.Node** %node8
  %getNodeName351 = call i8* @getNodeName(%struct.Node* %node8350)
  %hash352 = load %struct.hashmap** %hash
  %hashmap_get353 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %hash352, i8* %getNodeName351)
  %voidToint354 = call i32 @voidToint(i8* %hashmap_get353)
  %now355 = load i32* %now
  %w356 = load %struct.List** %w
  %get_list_element357 = call i8* @get_list_element(%struct.List* %w356,
i32 %now355)
  %voidToint358 = call i32 @voidToint(i8* %get_list_element357)
  %tmp359 = add i32 %voidToint354, %voidToint358
```

```
  %node8360 = load %struct.Node** %node8
  %getNodeName361 = call i8* @getNodeName(%struct.Node* %node8360)
  %hash362 = load %struct.hashmap** %hash
  %hashmap_put363 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash362, i8* %getNodeName361, i32 %tmp359)
  br label %merge342

else364:                                          ; preds = %while_body334
  %i365 = load i32* %i
  %g2366 = load %struct.Graph** %g2
  %graphLength367 = call i32 @graphLength(%struct.Graph* %g2366)
  %tmp368 = mul i32 %i365, %graphLength367
  %j369 = load i32* %j
  %tmp370 = add i32 %tmp368, %j369
  store i32 %tmp370, i32* %now
  %now371 = load i32* %now
  %w372 = load %struct.List** %w
  %get_list_element373 = call i8* @get_list_element(%struct.List* %w372,
i32 %now371)
  %voidToint374 = call i32 @voidToint(i8* %get_list_element373)
  %node8375 = load %struct.Node** %node8
  %getNodeName376 = call i8* @getNodeName(%struct.Node* %node8375)
  %hash377 = load %struct.hashmap** %hash
  %hashmap_put378 = call %struct.hashmap* (%struct.hashmap*, ...)*
@hashmap_put(%struct.hashmap* %hash377, i8* %getNodeName376,
i32 %voidToint374)
  br label %merge342

merge384:                                         ; preds = %while333
  %i385 = load i32* %i
  %tmp386 = add i32 %i385, 1
  store i32 %tmp386, i32* %i
  br label %while326

merge390:                                         ; preds = %while326
  %g2391 = load %struct.Graph** %g2
  %graphLength392 = call i32 @graphLength(%struct.Graph* %g2391)
  store i32 %graphLength392, i32* %size
  store i32 0, i32* %i
  br label %while393

while393:                                         ; preds
= %while_body394, %merge390
  %i412 = load i32* %i
  %size413 = load i32* %size
  %tmp414 = icmp slt i32 %i412, %size413
  br i1 %tmp414, label %while_body394, label %merge415

while_body394:                                    ; preds = %while393
  %i395 = load i32* %i
  %g2396 = load %struct.Graph** %g2
  %iterGraph397 = call %struct.Node* @iterGraph(%struct.Graph* %g2396,
i32 %i395)
```

```
  store %struct.Node* %iterGraph397, %struct.Node** %node7
  %node7398 = load %struct.Node** %node7
  %getNodeName399 = call i8* @getNodeName(%struct.Node* %node7398)
  %hash400 = load %struct.hashmap** %hash
  %hashmap_get401 = call i8* (%struct.hashmap*, ...)*
@hashmap_get(%struct.hashmap* %hash400, i8* %getNodeName399)
  %voidToint402 = call i32 @voidToint(i8* %hashmap_get401)
  %func_result403 = call i32 @func(i32 %voidToint402)
  store i32 %func_result403, i32* %now
  %now404 = load i32* %now
  %create_list405 = call %struct.List* @create_list(i32 0)
  %now406 = load i32* %now
  %plus_list407 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list405, i32 %now406)
  %l408 = load %struct.List** %l
  %concat_list409 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %l408, %struct.List* %create_list405)
  store %struct.List* %concat_list409, %struct.List** %l
  %i410 = load i32* %i
  %tmp411 = add i32 %i410, 1
  store i32 %tmp411, i32* %i
  br label %while393

merge415:                                         ; preds = %while393
  store i32 0, i32* %sum
  %g2416 = load %struct.Graph** %g2
  %graphLength417 = call i32 @graphLength(%struct.Graph* %g2416)
  store i32 %graphLength417, i32* %size
  store i32 0, i32* %i
  br label %while418

while418:                                         ; preds
= %while_body419, %merge415
  %i451 = load i32* %i
  %size452 = load i32* %size
  %tmp453 = icmp slt i32 %i451, %size452
  br i1 %tmp453, label %while_body419, label %merge454

while_body419:                                    ; preds = %while418
  %i420 = load i32* %i
  %g2421 = load %struct.Graph** %g2
  %iterGraph422 = call %struct.Node* @iterGraph(%struct.Graph* %g2421,
i32 %i420)
  store %struct.Node* %iterGraph422, %struct.Node** %node7
  %i423 = load i32* %i
  %l424 = load %struct.List** %l
  %get_list_element425 = call i8* @get_list_element(%struct.List* %l424,
i32 %i423)
  %voidToint426 = call i32 @voidToint(i8* %get_list_element425)
  %hidden_to_outer427 = load i32* %hidden_to_outer
  %i428 = load i32* %i
  %tmp429 = add i32 %hidden_to_outer427, %i428
  %w430 = load %struct.List** %w
```

```
  %get_list_element431 = call i8* @get_list_element(%struct.List* %w430,
i32 %tmp429)
  %voidToint432 = call i32 @voidToint(i8* %get_list_element431)
  %tmp433 = mul i32 %voidToint426, %voidToint432
  %tmp434 = sdiv i32 %tmp433, 10
  %sum435 = load i32* %sum
  %i436 = load i32* %i
  %l437 = load %struct.List** %l
  %get_list_element438 = call i8* @get_list_element(%struct.List* %l437,
i32 %i436)
  %voidToint439 = call i32 @voidToint(i8* %get_list_element438)
  %hidden_to_outer440 = load i32* %hidden_to_outer
  %i441 = load i32* %i
  %tmp442 = add i32 %hidden_to_outer440, %i441
  %w443 = load %struct.List** %w
  %get_list_element444 = call i8* @get_list_element(%struct.List* %w443,
i32 %tmp442)
  %voidToint445 = call i32 @voidToint(i8* %get_list_element444)
  %tmp446 = mul i32 %voidToint439, %voidToint445
  %tmp447 = sdiv i32 %tmp446, 10
  %tmp448 = add i32 %sum435, %tmp447
  store i32 %tmp448, i32* %sum
  %i449 = load i32* %i
  %tmp450 = add i32 %i449, 1
  store i32 %tmp450, i32* %i
  br label %while418

merge454:                                         ; preds = %while418
  %sum455 = load i32* %sum
  %func_result456 = call i32 @func(i32 %sum455)
  %tmp457 = sub i32 0, %func_result456
  store i32 %tmp457, i32* %error
  %sum458 = load i32* %sum
  %tmp459 = add i32 %sum458, 1
  %func_result460 = call i32 @func(i32 %tmp459)
  %sum461 = load i32* %sum
  %func_result462 = call i32 @func(i32 %sum461)
  %tmp463 = sub i32 %func_result460, %func_result462
  %tmp464 = sdiv i32 %tmp463, 1
  %sum465 = load i32* %sum
  %func_result466 = call i32 @func(i32 %sum465)
  %sum467 = load i32* %sum
  %tmp468 = sub i32 %sum467, 1
  %func_result469 = call i32 @func(i32 %tmp468)
  %tmp470 = sub i32 %func_result466, %func_result469
  %tmp471 = sdiv i32 %tmp470, 1
  %tmp472 = add i32 %tmp464, %tmp471
  %tmp473 = sdiv i32 %tmp472, 2
  store i32 %tmp473, i32* %de
  %de474 = load i32* %de
  %error475 = load i32* %error
  %tmp476 = mul i32 %de474, %error475
  %tmp477 = sdiv i32 %tmp476, 10
```

```
  store i32 %tmp477, i32* %sum
  store i32 0, i32* %i
  br label %while478

while478:                                         ; preds
= %while_body479, %merge454
  %i497 = load i32* %i
  %size498 = load i32* %size
  %tmp499 = icmp slt i32 %i497, %size498
  br i1 %tmp499, label %while_body479, label %merge500

while_body479:                                    ; preds = %while478
  %sum480 = load i32* %sum
  %hidden_to_outer481 = load i32* %hidden_to_outer
  %i482 = load i32* %i
  %tmp483 = add i32 %hidden_to_outer481, %i482
  %w484 = load %struct.List** %w
  %get_list_element485 = call i8* @get_list_element(%struct.List* %w484,
i32 %tmp483)
  %voidToint486 = call i32 @voidToint(i8* %get_list_element485)
  %tmp487 = mul i32 %sum480, %voidToint486
  %tmp488 = mul i32 %tmp487, 1
  store i32 %tmp488, i32* %update
  %update489 = load i32* %update
  %create_list490 = call %struct.List* @create_list(i32 0)
  %update491 = load i32* %update
  %plus_list492 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list490, i32 %update491)
  %delta_w493 = load %struct.List** %delta_w
  %concat_list494 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %delta_w493, %struct.List* %create_list490)
  store %struct.List* %concat_list494, %struct.List** %delta_w
  %i495 = load i32* %i
  %tmp496 = add i32 %i495, 1
  store i32 %tmp496, i32* %i
  br label %while478

merge500:                                         ; preds = %while478
  store i32 0, i32* %count
  store i32 0, i32* %i
  br label %while501

while501:                                         ; preds
= %merge524, %merge500
  %i529 = load i32* %i
  %hidden_to_outer530 = load i32* %hidden_to_outer
  %tmp531 = icmp slt i32 %i529, %hidden_to_outer530
  br i1 %tmp531, label %while_body502, label %merge532

while_body502:                                    ; preds = %while501
  %i503 = load i32* %i
  %w504 = load %struct.List** %w
```

```
  %get_list_element505 = call i8* @get_list_element(%struct.List* %w504,
i32 %i503)
  %voidToint506 = call i32 @voidToint(i8* %get_list_element505)
  %count507 = load i32* %count
  %delta_w508 = load %struct.List** %delta_w
  %get_list_element509 = call i8*
@get_list_element(%struct.List* %delta_w508, i32 %count507)
  %voidToint510 = call i32 @voidToint(i8* %get_list_element509)
  %tmp511 = add i32 %voidToint506, %voidToint510
  store i32 %tmp511, i32* %update
  %update512 = load i32* %update
  %create_list513 = call %struct.List* @create_list(i32 0)
  %update514 = load i32* %update
  %plus_list515 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list513, i32 %update514)
  %update_w516 = load %struct.List** %update_w
  %concat_list517 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_w516, %struct.List* %create_list513)
  store %struct.List* %concat_list517, %struct.List** %update_w
  %count518 = load i32* %count
  %tmp519 = add i32 %count518, 1
  store i32 %tmp519, i32* %count
  %count520 = load i32* %count
  %g2521 = load %struct.Graph** %g2
  %graphLength522 = call i32 @graphLength(%struct.Graph* %g2521)
  %tmp523 = icmp eq i32 %count520, %graphLength522
  br i1 %tmp523, label %then525, label %else526

merge524:                                         ; preds
= %else526, %then525
  %i527 = load i32* %i
  %tmp528 = add i32 %i527, 1
  store i32 %tmp528, i32* %i
  br label %while501

then525:                                          ; preds = %while_body502
  store i32 0, i32* %count
  br label %merge524

else526:                                          ; preds = %while_body502
  br label %merge524

merge532:                                         ; preds = %while501
  %g2533 = load %struct.Graph** %g2
  %graphLength534 = call i32 @graphLength(%struct.Graph* %g2533)
  store i32 %graphLength534, i32* %size
  store i32 0, i32* %i
  br label %while535

while535:                                         ; preds
= %while_body536, %merge532
  %i562 = load i32* %i
  %size563 = load i32* %size
```

```
  %tmp564 = icmp slt i32 %i562, %size563
  br i1 %tmp564, label %while_body536, label %merge565

while_body536:                                    ; preds = %while535
  %i537 = load i32* %i
  %g2538 = load %struct.Graph** %g2
  %iterGraph539 = call %struct.Node* @iterGraph(%struct.Graph* %g2538,
i32 %i537)
  store %struct.Node* %iterGraph539, %struct.Node** %node7
  %hidden_to_outer540 = load i32* %hidden_to_outer
  %i541 = load i32* %i
  %tmp542 = add i32 %hidden_to_outer540, %i541
  %w543 = load %struct.List** %w
  %get_list_element544 = call i8* @get_list_element(%struct.List* %w543,
i32 %tmp542)
  %voidToint545 = call i32 @voidToint(i8* %get_list_element544)
  %sum546 = load i32* %sum
  %i547 = load i32* %i
  %l548 = load %struct.List** %l
  %get_list_element549 = call i8* @get_list_element(%struct.List* %l548,
i32 %i547)
  %voidToint550 = call i32 @voidToint(i8* %get_list_element549)
  %tmp551 = mul i32 %sum546, %voidToint550
  %tmp552 = sdiv i32 %tmp551, 10
  %tmp553 = add i32 %voidToint545, %tmp552
  store i32 %tmp553, i32* %update
  %update554 = load i32* %update
  %create_list555 = call %struct.List* @create_list(i32 0)
  %update556 = load i32* %update
  %plus_list557 = call %struct.List* (%struct.List*, ...)*
@plus_list(%struct.List* %create_list555, i32 %update556)
  %update_l558 = load %struct.List** %update_l
  %concat_list559 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_l558, %struct.List* %create_list555)
  store %struct.List* %concat_list559, %struct.List** %update_l
  %i560 = load i32* %i
  %tmp561 = add i32 %i560, 1
  store i32 %tmp561, i32* %i
  br label %while535

merge565:                                         ; preds = %while535
  %update_l566 = load %struct.List** %update_l
  %update_w567 = load %struct.List** %update_w
  %concat_list568 = call %struct.List* (%struct.List*, %struct.List*, ...)*
@concat_list(%struct.List* %update_w567, %struct.List* %update_l566)
  %update_w569 = load %struct.List** %update_w
  store %struct.List* %update_w569, %struct.List** %w
  %w570 = load %struct.List** %w
  %print_list571 = call i8* @print_list(%struct.List* %w570)
  ret i32 0
}

define i32 @func(i32 %in) {
```

```
entry:
  %in1 = alloca i32
  store i32 %in, i32* %in1
  %in2 = load i32* %in1
  %tmp = icmp sgt i32 %in2, 0
  br i1 %tmp, label %then, label %else

merge:                                            ; No predecessors!
  ret i32 0

then:                                             ; preds = %entry
  %in3 = load i32* %in1
  ret i32 %in3

else:                                             ; preds = %entry
  ret i32 0
}
```

6.2. Automated Test Suite

6.2.1. Test Cases

Out test suite includes 70 test cases, among which there are 26 should-fail test cases and 44 should-pass test cases. The test cases include all features of our language, data structure and types, operations, control flow and complicated functions.

- Data structure and types

We test all the data structure and types of our language. This include the basic types int, float, bool, string, and all the other data structures of Tusimple, list, set, map, node and graph. Test assures that these data structure and types can function properly.

- Binary and unary operators

The test suite tests all the basic binary and unary operators. These operators can work well on normal condition. Besides, we test them under different exceptional conditions, for example, type checking.

- Control flow

The test of this part is to check whether for loop, while loop and if/else blocks can work properly. Besides, we need to test them under different exceptional conditions, to check whether it can report correct error information.

- Complicated functions

We utilize all the data structure, types, operators and control flow to write complicated function that can implement simple and complicated algorithm. For example, depth-first-search, single source shortest path etc.

6.2.2. Test Results

We use the similar shell script as micro-c's test script to work as automated test script. And we test all the test cases by just running `./testall.sh`. And the test result are as follows, all the test cases are successful.

```
plt@ubuntu-plt:~/TuSimple-Graph-Language$ ./testall.sh
test-Dijkstra1...SUCCESS
test-arith1-add...SUCCESS
test-arith2-sub...SUCCESS
test-arith3-mult...SUCCESS
test-arith4-division...SUCCESS
test-bfs1...SUCCESS
test-bfs2...SUCCESS
test-bool...SUCCESS
test-data1...SUCCESS
test-dfs1...SUCCESS
test-dfs2...SUCCESS
test-for1...SUCCESS
test-for2...SUCCESS
test-for3...SUCCESS
test-fun1...SUCCESS
test-fun2...SUCCESS
test-graph1...SUCCESS
test-graph2...SUCCESS
test-graph3...SUCCESS
test-if1...SUCCESS
test-if2...SUCCESS
test-int...SUCCESS
test-list1...SUCCESS
test-list2...SUCCESS
test-list3...SUCCESS
test-list4...SUCCESS
test-list5...SUCCESS
test-map1...SUCCESS
test-map2...SUCCESS
test-map3...SUCCESS
test-map4...SUCCESS
test-node1...SUCCESS
test-node2...SUCCESS
test-node3...SUCCESS
test-node4...SUCCESS
```

```
test-node5...SUCCESS
test-node6...SUCCESS
test-ops1...SUCCESS
test-ops2...SUCCESS
test-set1...SUCCESS
test-set2...SUCCESS
test-set3...SUCCESS
test-set4...SUCCESS
test-while1...SUCCESS
test-while2...SUCCESS
fail-add1...SUCCESS
fail-fun1...SUCCESS
fail-fun2...SUCCESS
fail-fun3...SUCCESS
fail-fun4...SUCCESS
fail-list1...SUCCESS
fail-list2...SUCCESS
fail-fun3...SUCCESS
fail-fun4...SUCCESS
fail-list1...SUCCESS
fail-list2...SUCCESS
fail-list3...SUCCESS
fail-list4...SUCCESS
fail-map1...SUCCESS
fail-map2...SUCCESS
fail-node1...SUCCESS
fail-node2...SUCCESS
fail-op-and1...SUCCESS
fail-op-and2...SUCCESS
fail-op-not1...SUCCESS
fail-op-not2...SUCCESS
fail-op-or1...SUCCESS
fail-op-or2...SUCCESS
fail-set1...SUCCESS
fail-set2...SUCCESS
fail-set3...SUCCESS
fail-set4...SUCCESS
fail-set5...SUCCESS
fail-test...SUCCESS
fail-undeclaredfun...SUCCESS
```

7. Conclusions

7.1. Jihao Zhang

This is one of the most interesting course I took at Columbia. Not only did I learn how compilers work, but also how to programming in functional languages. I think it's really important to get going on OCaml earlier in this course for the project. The second step is to completely understand the example codes of a compiler, which is helpful of understanding the compiler architecture and how to write a fully functional compiler.

Another important aspect of this project is to cooperate with other group members. We must meet regularly and work on the compiler together. I really enjoyed working with my team members, they are super intelligent and diligent. We often work on the project through late nights and discuss on the details of our programming language. This is not an easy process, but I really enjoyed it a lot.

As a group manager, apart from wring code for the project, I also have to arrange meetings, divide works and make sure that we are keeping up with the schedule. I had a lot of fun working on this team, because we all shared in this very same goal of getting this project up and running. And we work productively through these open communication channels. We all respected each other's work and help other people as much as I can. All of those helped contribute to the overall goal of the team.

7.2. Zicheng Xu

The design of a language is a balance of specification and generalization. When we want to narrow the application scope of our language to graph algorithms, the methods could not be easily used in neural networks. The final version of our language is intended to simplify the implementation of neural networks. Some of our rules may looks strange from the view of traditional graph algorithms.

For example, we allow a graph to contain some edges linking a node in the graph and a node out of the graph. This means when you remove a node, which is linked by another node in the graph, from the graph, you can still access it using bfs(), dfs() or traversal algorithms written by hand. It would enable us to define a particular layer in the neural network using graph because a layer would definitely has links with other layers. This feature would be really helpful in developments of iterations based on layers. When it comes to normal graph, we need to record a hashmap using reverse edges. Hence when we want to delete a node, we could query the hashmap to delete all the original edges. In fact, in other languages, we should also do the same thing. So this tradeoff is acceptable from my point of view. Please believe that all the features of our language are specified according to our coding experiences and the principle of optimization. The coding samples in the document should provide you a clear understand of our thoughts.

At the beginning of our project, we do have a great ambition to build a most comprehensive language for late-comers. This mission should be delivered to the readers of this document. I would recommend you to consider on the following parts:

1. The abstraction of common parts in graph algorithms: by now, our abstraction is based on finite and static graph. The improvements could be done when including more parallel and real-time operations.

2. The visualization of complex graph: by now, our printGraph() function could render the details of the graph. But in practice, we have far more complicated graphs to process. If we could implement a function like the rendering function of networkX, the debugging work would be improved tremendously.

Last but not least, the programming skills of Ocaml is quite interesting to me since functional language is a new toy. It expanded my thinking of programming languages and helped a lot in understanding the essence of translator, which inspired the specification of TuSimple at the same time. I highly recommend others who are interested in underlying mechanism of programming to study Ocaml.

7.3. Shen Zhu

This project is definitely hard and challenging, but finally when I saw the running of breath-first search, depth-first search and single srouce shortest path algorithm, suddenly I feel everything is worth it.

There are a lost of obstacles during the development of TuSimple language. I have never used OCaml before, so it takes quite a while for to get used to this functional programming language. And the documentation of LLVM for OCaml is not quite clear. As system architect in this team, I need to figure out the structure of our compiler, so I spent quite a lot of time in reading and understanding the code of MicroC compiler. And thinking the functions of our language, we need to support some basic data strucutres such as List, Set and Hashmap, I consulted students who have registered this class and our TA, then figured out the whole structure of our compiler.

Another thing I think very important is to have a project plan and complete each step according to this plan. Having a good team can also go a long way in achieving the goal of your project.

So my advice for future students to start early, because getting familiar with OCaml and understanding how your codes compile downto LLVM takes quite a while. Another advice is to read the MicorC compiler, I also want to strengthen the importance of “build something first”, after building this prototype, you can add features on it gradually.

7.4. Yunzi Chai

I think I have learned a lot from this PLT project. First, about the theory part. At the beginning of our project, I am really confused about what is going on. Later, I found out that it is quite important to really understand what Edward taught in class and the basic knowledge. And the homework is also really helpful, it help me to test whether I really digest the knowledge and apply them in our project. Then we started from Micro-C compiler. We tried to figure out the basic files of Micro-C, like scanner, parser, ast, semantic check and codegen. And then we can start from mimicing the micro-c. In the end of project, I get a quite comprehensive understand of how a programming language is designed and how a compiler is implemented.

As a tester, at first, I think it is quite a simple job, all you have to do is just to use the language. And later on, I found out that testing is not as simple as it seems to be. You need to test all the basic function of basic data structure, control flows and types. And what is much more important is to test these corners cases. So you need to think about various usage of our language. And testing is quite an important part of software development, if you miss any tiny mistakes, it may cause severe disaster in the future. At the same time, because there are a lot of cases that need to be tested, you should learn how to do it systematically.

As it is a team project that five persons work together on, how to work efficiently is key point to the success of the project. Everyone should be responsible for this project and do not always rely on other people. Because it is a long-time project, we should have a clear timeline and plan for this project, it can assure the project work smoothly. If you just start working on it in the last week, it would be a nightmare and disaster. Manager is a quite import role, he should be the person that give a final decision when everyone cannot come to an agreement. Finally, everyone should have a clear role and understand clearly what they should do, otherwise, several members may be doing the same job, and no one is working on certain part. Communication is very important, whenever you have a confusion about other member's work, you should ask them immediately.

Special thanks to Professor Edward and our TA Julie and all our team members!

7.5. Ziyi Mu

This project is one of the largest and most complex projects I have ever done in my academic life, as it consists several large components and the interaction of each component is very complex and requires lots of effort to complete even a simple end-to-end workflow.

It is fortunate for me to collaborate in a great team with several accountable teammates. The pleasant team collaboration is based on the regular team meetings and reflection. Also since all the teammates are very familiar with github, collaborating our code becomes easier than we expected.

One lesson learned is that throughout a large project like this, the team will need to make several important design decisions. The decisions should be made as early as possible and need to make

sure every member understand the merit of this decision and start developing along the way. I observed a speed boosting after we have made those decisions and yield nicer result.

7.6. Advice for Future Teams

7.6.1. The initial step of construction

The imagination and the implementation would vary a lot. We recommend future teams to determine a core function and build rules based on it. In the following process, you have a high chance of changing parameters and functionalities. The changes may make you want to scrap it and start all over again. But as long as you are stick to your core function, the painful process would be acceptable.

7.6.2. The tip on debugging

You should let all of your team members go through the details of code in codegen.ml. This is the key part to connect other parts. 80% of the debugging work is associated with this file. It would save a lot of time if your members have the ability to debug on their own.

7.6.3. The last advice

Start the final project as early as possible. The programming language has some common features with natural languages. There is always something remained could be improved. Leave no regret, try your best. Good luck!

8. Appendix : Code Listing

8.1. Lib/

This folder contains the C library we wrote to support TuSimple language.

8.1.1. config.h

```
#ifndef LIB_CONFIG_H
#define LIB_CONFIG_H


#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "cast.h"


#define INT 0
#define FLOAT 1
#define BOOL 2
#define STRING 3
#define NODE 4
#define GRAPH 5


#endif //LIB_CONFIG_H
```

8.1.2. cast.h

```
#include "config.h"
#include "utils.h"


#ifndef TUSIMPLELIB_CAST_H
#define TUSIMPLELIB_CAST_H


// Methods that convert void to other types
int32_t voidToint(void* pointer);
double voidTofloat(void* pointer);
bool voidTobool(void* pointer);
char* voidTostring(void* pointer);
struct Node* voidTonode(void* pointer);
struct Graph* voidTograph(void* pointer);


// Methods that convert other types to void
```

```
void* intTovoid(int32_t value);
void* floatTovoid(double value);
void* boolTovoid(bool value);
void* stringTovoid(char* value);
void* nodeTovoid(struct Node* value);
void* graphTovoid(struct Graph* value);

#endif //TUSIMPLELIB_CAST_H
```

8.1.3. cast.c

```
#include "cast.h"


int32_t voidToint(void* pointer) {
    return *((int32_t*) pointer);
}

double voidTofloat(void* pointer) {
    return *((double*) pointer);
}

bool voidTobool(void* pointer) {
    return *((bool*) pointer);
}

char* voidTostring(void* pointer) {
    return (char*) pointer;
}

struct Node* voidTonode(void* pointer) {
    return (struct Node*) pointer;
}

struct Graph* voidTograph(void* pointer) {
    return (struct Graph*) pointer;
}


void* intTovoid(int32_t value) {
    int* value_pointer = (int*)malloc(sizeof(int32_t));
    *value_pointer = value;
    return (void*)value_pointer;
}

void* floatTovoid(double value) {
    double* value_pointer = (double*)malloc(sizeof(double));
    *value_pointer = value;
    return (void*)value_pointer;
}

void* boolTovoid(bool value) {
    bool* value_pointer = (bool*)malloc(sizeof(bool));
```

```
    *value_pointer = value;
    return (void*)value_pointer;
}

void* stringTovoid(char* value) {
    return (void*) value;
}

void* nodeTovoid(struct Node* value) {
    return (void*) value;
}

void* graphTovoid(struct Graph* value) {
    return (void*) value;
}
```

8.1.4. list.h

```
#ifndef TUSIMPLELIB_LIST_H
#define TUSIMPLELIB_LIST_H


#include "config.h"


struct List {
    int32_t size;
    int32_t type;
    void **value;
    int32_t currPos;
};



// Functions for list
struct List *create_list(int32_t type);

struct List *plus_list_helper(struct List *list, void *data);

struct List *plus_list(struct List *list, ...);

struct List *concat_list(struct List *list1, struct List *list2);

void *get_list_element(struct List *list, int index);

void *pop_list_element(struct List *list);

void *remove_list_element(struct List *list, int index);

int get_list_size(struct List *list);

bool check_list_element(struct List *list, ...);
```

```
void change_list_element(struct List* list, int index, ...);

#endif //TUSIMPLELIB_LIST_H
```

8.1.5. list.c

```
#include "list.h"
#include "utils.h"


struct List *create_list(int32_t type) {
    struct List *newList = (struct List *) malloc(sizeof(struct List));

    // Init value of newly created list
    newList->type = type;
    newList->size = 1;
    newList->currPos = 0;
    newList->value = (void *) malloc(newList->size * sizeof(void *));
    return newList;
}


struct List *plus_list_helper(struct List *list, void *value) {
    if (list->currPos >= list->size) {
        // Double list size
        list->size = list->size * 2;
        list->value = (void **) realloc(list->value, list->size * sizeof(void
*));
    }

    // Add element and reset size
    *(list->value + list->currPos) = value;
    list->currPos++;
    return list;
}


struct List *plus_list(struct List *list, ...) {
    if (list == NULL) {
        printf("Error! plus_list() : List does not exist. \n");
        exit(1);
    }

    // Extract data using variable-argument
    va_list arg_ptr;
    va_start(arg_ptr, list);

    void *data;
    switch (list->type) {
        case INT:
            data = intTovoid(va_arg(arg_ptr, int));
            break;
```

```
        case FLOAT:
            data = floatTovoid(va_arg(arg_ptr, double));
            break;

        case BOOL:
            data = boolTovoid(va_arg(arg_ptr, bool));
            break;

        case STRING:
            data = stringTovoid(va_arg(arg_ptr, char*));
            break;

        case NODE:
            data = nodeTovoid(va_arg(arg_ptr, struct Node*));
            break;

        case GRAPH:
            data = graphTovoid(va_arg(arg_ptr, struct Graph*));

        default:
            break;
    }

    va_end(arg_ptr);

    return plus_list_helper(list, data);
}


void *get_list_element(struct List *list, int index) {
    // Corner case
    if (list == NULL) {
        printf("Error! get_list_element() : List does not exist. \n");
        exit(1);
    } else if (list->size == 0 || list->size <= index || list->size <= -
index) {
        printf("Error! get_list_element() : Index out of range. \n");
        exit(1);
    } else if (index < 0) {
        index += list->size;
    }

    return *(list->value + index);
}


int get_list_size(struct List *list) {
    // Corner case
    if (list == NULL) {
        // printf("Error! get_list_size() : List does not exist. \n");
        // exit(1);
        return 0;
    }
```

```
    return list->currPos;
}


struct List *concat_list(struct List *list1, struct List *list2) {
    int size2 = list2->currPos;
    int i;

    switch (list1->type) {
        case INT:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidToint(*(list2->value + i)));
            }
            break;

        case BOOL:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidTobool(*(list2->value + i)));
            }
            break;

        case FLOAT:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidTofloat(*(list2->value + i)));
            }
            break;

        case STRING:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidTostring(*(list2->value + i)));
            }
            break;

        case NODE:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidTonode(*(list2->value + i)));
            }
            break;

        case GRAPH:
            for (i = 0; i < size2; i++) {
                list1 = plus_list(list1, voidTograph(*(list2->value + i)));
            }
            break;

        default:
            break;
    }

    return list1;
}
```

```
void *pop_list_element(struct List *list) {
    if (list == NULL) {
        printf("Error! pop_list_element() : List does not exist.\n");
        exit(1);
    } else if (list->currPos < 1) {
        printf("Error! pop_list_element() : No element to pop.\n");
        exit(1);
    }

    void *value = *(list->value + list->currPos - 1);
    list->currPos--;

    return value;
}


void *remove_list_element(struct List *list, int index) {
    if (list == NULL) {
        printf("Error! remove_list_element() : List does not exist.\n");
        exit(1);
    } else if (list->size <= index || list->size == 0) {
        printf("Error! remove_list_element() : Index out of range.\n");
        exit(1);
    }

    void *elementToRemove = *(list->value + index);

    for (int i = index; i < list->currPos; i++) {
        *(list->value + i) = *(list->value + i + 1);
    }

    // decrease size
    list->currPos--;

    return elementToRemove;
}


bool check_list_element(struct List *list, ...) {
    if (list == NULL) {
        printf("%s\n", "Error! check_list_element : List does not exist.\n");
        exit(1);
    }

    void *target;
    bool exist = 0;

    va_list args_ptr;
    va_start(args_ptr, list);

    switch (list->type) {
        case INT:
            target = intTovoid(va_arg(args_ptr, int));
```

```
            break;

        case BOOL:
            target = boolTovoid(va_arg(args_ptr, bool));
            break;

        case FLOAT:
            target = floatTovoid(va_arg(args_ptr, double));
            break;

        case STRING:
            target = stringTovoid(va_arg(args_ptr, char*));
            break;

        case NODE:
            target = nodeTovoid(va_arg(args_ptr, struct Node*));
            break;

        case GRAPH:
            target = graphTovoid(va_arg(args_ptr, struct Graph*));
            break;

        default:
            break;
    }

    va_end(args_ptr);

    // Perform linear scan for target
    for (int i = 0; i < list->currPos; i++) {
        switch (list->type) {
            case INT:
                if (voidToint(target) == voidToint(*(list->value + i))) {
                    exist = 1;
                    return exist;
                }
                break;

            case BOOL:
                if (voidTobool(target) == voidTobool(*(list->value + i))) {
                    exist = 1;
                    return exist;
                }
                break;

            case FLOAT:
                if (fabs(voidTofloat(target) - voidTofloat(*(list->value +
i))) < 0.00001) {
                    exist = 1;
                    return exist;
                }
                break;
```

```
            case STRING:
                if (strcmp(voidTostring(target), voidTostring(*(list->value +
i))) == 0) {
                    exist = 1;
                    return exist;
                }
                break;

            case NODE:
                if (strcmp(voidTonode(target)->name, voidTonode(*(list->value
+ i))->name) == 0) {
                    exist = 1;
                    return exist;
                }
                break;

            case GRAPH:
                if (strcmp(voidTograph(target)->name,
voidTograph(*(list->value + i))->name) == 0) {
                    exist = 1;
                    return exist;
                }
                break;

            default:
                break;
        }
    }

    return exist;
}

void change_list_element(struct List* list, int index, ...) {
    if (list == NULL) {
        printf("%s\n", "Error! change_list_element : List does not exist!");
        exit(1);
    }

    va_list args_ptr;
    va_start(args_ptr, index);

    switch (list->type) {
        case INT:
            *(list->value + index) = intTovoid(va_arg(args_ptr, int));
            break;

        case FLOAT:
            *(list->value + index) = floatTovoid(va_arg(args_ptr, double));
            break;

        case BOOL:
            *(list->value + index) = boolTovoid(va_arg(args_ptr, bool));
            break;
```

```
        case STRING:
            *(list->value + index) = stringTovoid(va_arg(args_ptr, char*));
            break;

        case NODE:
            *(list->value + index) = nodeTovoid(va_arg(args_ptr, struct
Node*));
            break;

        case GRAPH:
            *(list->value + index) = va_arg(args_ptr, struct Graph*);
            break;

        default:
            break;
    }

    va_end(args_ptr);

    return;
}
```

8.1.6. hashmap.h

```
#ifndef TUSIMPLELIB_HASHMAP_H
#define TUSIMPLELIB_HASHMAP_H


#include "config.h"
#include "utils.h"

#define MAP_MISSING -3  /* No such element */
#define MAP_FULL -2    /* Hashmap is full */
#define MAP_OMEM -1    /* Out of Memory */
#define MAP_OK 0    /* OK */


Define data structure
struct hashmap_element {
    char *key;
    int used;
    void *data[2];
};

struct hashmap {
    int tableSize;
    int size;
    int32_t keyType;
    int32_t valueType;
    struct hashmap_element *data;
};
```

```
typedef int (*Func)(void *, void *, void *);


// Define functions
struct hashmap *create_hashmap(int32_t keyType, int32_t valueType);

struct hashmap *hashmap_put(struct hashmap *map, ...);

int hashmap_length(struct hashmap *map);

int32_t hashmap_keytype(struct hashmap *map);

int32_t hashmap_valuetype(struct hashmap *map);

bool hashmap_haskey(struct hashmap *map, ...);

void *hashmap_get(struct hashmap *map, ...);

struct hashmap *hashmap_remove(struct hashmap *map, ...);

struct List *hashmap_keys(struct hashmap *map);

int hashmap_iterate(struct hashmap *map, Func f);

#endif //TUSIMPLELIB_HASHMAP_H
```

8.1.7. hashmap.c

```
#include "hashmap.h"


#define INITIAL_SIZE 256
#define MAX_CHAIN_LENGTH 8


// Functions that calculate the hash
static unsigned long crc32_tab[] = {
        0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
        0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
        0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
        0x90bf1d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
        0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,
        0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L,
        0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
        0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
        0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L,
        0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
        0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L,
        0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
        0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
```

```
        0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
        0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0x0f00f934L, 0x9609a88eL,
        0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
        0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
        0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
        0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
        0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
        0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
        0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
        0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
        0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
        0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
        0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
        0x03b6e20cL, 0x74b1d29aL, 0xead54739L, 0x9dd277afL, 0x04db2615L,
        0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
        0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
        0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
        0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
        0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
        0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
        0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
        0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL,
        0x4669be79L, 0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
        0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
        0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
        0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
        0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
        0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL,
        0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
        0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
        0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
        0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
        0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
        0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
        0x37d83bf0L, 0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
        0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
        0xcdd70693L, 0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
        0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
        0x2d02ef8dL
};

/* Return a 32-bit CRC of the contents of the buffer. */
unsigned long crc32(const unsigned char *s, unsigned int len) {
    unsigned int i;
    unsigned long crc32val;

    crc32val = 0;
    for (i = 0; i < len; i++) {
        crc32val =
                crc32_tab[(crc32val ^ s[i]) & 0xff] ^
                (crc32val >> 8);
    }
    return crc32val;
```

```
}

/*
 * Hashing function for a string
 */
unsigned int hashmap_hash_int(struct hashmap *map, char *keystring) {

    unsigned long key = crc32((unsigned char *) (keystring),
strlen(keystring));

    /* Robert Jenkins' 32 bit Mix Function */
    key += (key << 12);
    key ^= (key >> 22);
    key += (key << 4);
    key ^= (key >> 9);
    key += (key << 10);
    key ^= (key >> 2);
    key += (key << 7);
    key ^= (key >> 12);

    /* Knuth's Multiplicative Method */
    key = (key >> 3) * 2654435761;

    return key % map->tableSize;
}

int hashmap_hash(struct hashmap *map, char *key) {
    int curr;

    // Test if map is full
    if (map->size >= map->tableSize / 2) return MAP_FULL;

    curr = hashmap_hash_int(map, key);
    for (int i = 0; i < MAX_CHAIN_LENGTH; i++) {
        if (map->data[curr].used == 0) {
            return curr;
        }

        if (map->data[curr].used == 1 && strcmp(map->data[curr].key, key) ==
0) {
            return curr;
        }

        curr = (curr + 1) % map->tableSize;
    }

    return MAP_FULL;
}

int hashmap_rehash(struct hashmap *map) {
    struct hashmap_element *curr;

    // Apply for new space, twice as large as before
```

```
    struct hashmap_element *temp = (struct hashmap_element *) calloc(2 *
map->tableSize,

sizeof(struct hashmap_element));
    if (!temp) return MAP_OMEM;

    // Update data pointers
    curr = map->data;
    map->data = temp;

    // Update size
    int oldSize = map->tableSize;
    map->tableSize = 2 * map->tableSize;
    map->size = 0;

    // Rehash
    for (int i = 0; i < oldSize; i++) {
        // Only hash those used position
        if (curr[i].used == 0) continue;

        hashmap_put(map, curr[i].key, curr[i].data);
    }

    free(curr);

    return MAP_OK;
}


// Functions for hashmap features
struct hashmap *create_hashmap(int32_t keyType, int32_t valueType) {
    // Apply memory
    struct hashmap *newMap = (struct hashmap *) malloc(sizeof(struct
hashmap));

    // Set properties
    newMap->keyType = keyType;
    newMap->valueType = valueType;
    newMap->size = 0;
    newMap->data = (struct hashmap_element *) calloc(INITIAL_SIZE,
sizeof(struct hashmap_element));
    newMap->tableSize = INITIAL_SIZE;

    return newMap;
}

struct hashmap *hashmap_put(struct hashmap *map, ...) {
    // Variables to store var_args
    void *keyData;
    void *valueData;
    char *key;

    va_list arg_ptr;
```

```
va_start(arg_ptr, map);

// Extract and format key
switch (map->keyType) {
    case INT:
        keyData = intTovoid(va_arg(arg_ptr, int));
        key = malloc(16);
        snprintf(key, 16, "%d", voidToint(keyData));
        break;

    case STRING:
        keyData = stringTovoid(va_arg(arg_ptr, char*));
        key = voidTostring(keyData);
        break;

    default:
        break;
}

// Extract and format value
switch (map->valueType) {
    case INT:
        valueData = intTovoid(va_arg(arg_ptr, int));
        break;

    case FLOAT:
        valueData = floatTovoid(va_arg(arg_ptr, double));
        break;

    case BOOL:
        valueData = boolTovoid(va_arg(arg_ptr, bool));
        break;

    case STRING:
        valueData = stringTovoid(va_arg(arg_ptr, char*));
        break;

    default:
        break;
}

va_end(arg_ptr);

// Check if element already exists
if (!hashmap_haskey(map, key)) {
    map->size++;
}

// Find where to put
int index = hashmap_hash(map, key);
while (index == MAP_FULL) {
    if (hashmap_rehash(map) == MAP_OMEM) {
        printf("Error! hashmap_put() : Out of Memory.\n");
```

```
            exit(1);
        }
        index = hashmap_hash(map, key);
    }

    // Put
    map->data[index].data[0] = keyData;
    map->data[index].data[1] = valueData;
    map->data[index].key = key;
    map->data[index].used = 1;
    //map->size++;

    return map;
}

bool hashmap_haskey(struct hashmap *map, ...) {
    char *key;

    va_list args_ptr;
    va_start(args_ptr, map);

    switch (map->keyType) {
        case INT:
            key = malloc(16);
            snprintf(key, 16, "%d", va_arg(args_ptr,
                    int));
            break;

        case STRING:
            key = va_arg(args_ptr, char*);
            break;

        default:
            break;
    }

    va_end(args_ptr);

    int index = hashmap_hash_int(map, key);
    for (int i = 0; i < MAX_CHAIN_LENGTH; i++) {
        int usedFlag = map->data[index].used;
        if (usedFlag == 1) {
            if (strcmp(map->data[index].key, key) == 0) {
                return 1;
            }
        }
        index = (index + 1) % map->tableSize;
    }

    return 0;
}

void *hashmap_get(struct hashmap *map, ...) {
```

```
    char *key;

    va_list args_ptr;
    va_start(args_ptr, map);

    switch (map->keyType) {
        case INT:
            key = malloc(16);
            snprintf(key, 16, "%d", va_arg(args_ptr,
                    int));
            break;

        case STRING:
            key = va_arg(args_ptr, char*);
            break;

        default:
            break;
    }

    va_end(args_ptr);

    int index = hashmap_hash_int(map, key);
    for (int i = 0; i < MAX_CHAIN_LENGTH; i++) {
        int useFlag = map->data[index].used;
        if (useFlag == 1) {
            if (strcmp(map->data[index].key, key) == 0) {
                return map->data[index].data[1];
            }
        }

        index = (index + 1) % map->tableSize;
    }

    // printf("Error! hashmap_get() : Key does not exist.\n");
    // exit(1);
    return NULL;
    //return boolTovoid(false);
}

struct hashmap *hashmap_remove(struct hashmap *map, ...) {
    char *key;

    va_list args_ptr;
    va_start(args_ptr, map);

    switch (map->keyType) {
        case INT:
            key = malloc(16);
            snprintf(key, 16, "%d", va_arg(args_ptr,
                    int));
            break;
```

```
            case STRING:
                key = va_arg(args_ptr, char*);
                break;

            default:
                break;
        }

        va_end(args_ptr);

        int index = hashmap_hash_int(map, key);

        for (int i = 0; i < MAX_CHAIN_LENGTH; i++) {
            int useFlag = map->data[index].used;
            if (useFlag == 1) {
                if (strcmp(map->data[index].key, key) == 0) {
                    map->data[index].used = 0;
                    map->data[index].data[0] = NULL;
                    map->data[index].data[1] = NULL;
                    map->data[index].key = NULL;
                    map->size--;

                    return map;
                }
            }
            index = (index + 1) % map->tableSize;
        }
        printf("Error! hashmap_remove() : No such key");
        exit(1);
    }

    struct List *hashmap_keys(struct hashmap *map) {
        if (map == NULL) {
            printf("Error! hashmap_keys() : Map does not exist.\n");
            exit(1);
        } else if (hashmap_length(map) <= 0) {
            printf("Error! hashmap_keys() : Map does not have key.\n");
            exit(1);
        }

        struct List *keys = create_list(map->keyType);
        for (int i = 0; i < map->tableSize; i++) {
            if (map->data[i].used != 0) {
                switch (map->keyType) {
                    case INT:
                        plus_list(keys, voidToint(map->data[i].data[0]));
                        break;

                    case STRING:
                        plus_list(keys, voidTostring(map->data[i].data[0]));
                        break;

                    default:
```

```
                    break;
                }
            }
        }

        return keys;
    }

    int hashmap_iterate(struct hashmap *map, Func f) {
        if (map == NULL) {
            printf("Error! hashmap_iterate() : Map does not exist.\n");
            exit(1);
        } else if (hashmap_length(map) <= 0) {
            return MAP_MISSING;
        }

        for(int i = 0; i < map->tableSize; i++) {
            if (map->data[i].used != 0) {
                int status = f(map->data[i].key, map->data[i].data[0],
    map->data[i].data[1]);
                if (status != MAP_OK) {
                    return status;
                }
            }
        }

        return MAP_OK;
    }

    int hashmap_length(struct hashmap *map) {
        if (map == NULL) {
            printf("Error! hashmap_length() : Map does not exist.\n");
            exit(1);
        }
        return map->size;
    }

    int32_t hashmap_keytype(struct hashmap *map) {
        if (map == NULL) {
            printf("Error! hashmap_length() : map does not exist.\n");
            exit(1);
        }
        return map->keyType;
    }

    int32_t hashmap_valuetype(struct hashmap *map) {
        if (map == NULL) {
            printf("Error! hashmap_length() : map does not exist.\n");
            exit(1);
        }
        return map->valueType;
    }
```

```
int test_inttoint_hashmap_iterate_func(void* key, void* keyData, void*
keyValue) {
    printf("%s ", "MAPKEY:");
    printf("%s\n", voidTostring(key));
    printf("%s ", "KEY DATA:");
    printf("%d\n", voidToint(keyData));
    printf("%s ", "VALUE DATA");
    printf("%d\n", voidToint(keyValue));

    return MAP_OK;
}
```

8.1.8. set.h

```
#ifndef TUSIMPLELIB_SET_H
#define TUSIMPLELIB_SET_H

#include "config.h"
#include "list.h"

#define SET_MISSING -1
#define SET_OK 1

struct Set {
    int32_t type;
    int32_t size;
    struct List *data;
};

typedef int (*Func2)(void **);

struct Set *create_set(int type);

bool check_set_element(struct Set *set, ...);

struct Set *put_set(struct Set *set, ...);

int32_t get_set_element_index(struct Set *set, ...);

struct Set *remove_set_element(struct Set *set, ...);

struct List *get_set_elements(struct Set *set);

int set_iterate(struct Set *set, Func2 f);

int32_t get_set_type(struct Set *set);

int32_t get_set_size(struct Set *set);

struct Set *put_set_from_list(struct Set *set, struct List * list);
```

```
#endif //TUSIMPLELIB_SET_H
```

8.1.9. set.c

```
#include "set.h"


struct Set *create_set(int32_t type) {
    // Apply for memory
    struct Set *newSet = (struct Set *) malloc(sizeof(struct Set));

    // Set property
    newSet->type = type;
    newSet->size = 0;
    newSet->data = create_list(type);

    return newSet;
}

bool check_set_element(struct Set *set, ...) {
    // Corner case
    if (set == NULL) {
        printf("%s\n", "Error! check_set_element : Set does not exist.\n");
        exit(1);
    }

    bool exist = 0;

    va_list args_ptr;
    va_start(args_ptr, set);

    switch (set->type) {
        case INT:
            exist = check_list_element(set->data, va_arg(args_ptr, int));
            break;

        case FLOAT:
            exist = check_list_element(set->data, va_arg(args_ptr, double));
            break;

        case BOOL:
            exist = check_list_element(set->data, va_arg(args_ptr, bool));
            break;

        case STRING:
            exist = check_list_element(set->data, va_arg(args_ptr, char*));
            break;

        case NODE:
            exist = check_list_element(set->data, va_arg(args_ptr, struct
Node*));
            break;
```

```
        default:
            break;
    }

    va_end(args_ptr);

    return exist;
}

/*
 * Set does not have order, this function is a helper function for removing
 * elements in set
 */
int32_t get_set_element_index(struct Set *set, ...) {
    // Corner case
    if (set == NULL) {
        printf("%s\n", "Error! get_set_element_index : Set does not
exist.\n");
        exit(1);
    }

    int index = 0;

    int intTemp;
    double floatTemp;
    bool boolTemp;
    char* stringTemp;
    struct Node* nodeTemp;

    va_list args_ptr;
    va_start(args_ptr, set);

    switch (set->type) {
        case INT:
            intTemp = va_arg(args_ptr, int);
            while (index < (set->data->currPos)) {
                if (intTemp == voidToint(*(set->data->value + index))) {
                    return index;
                }
                index++;
            }
            return -1;

        case BOOL:
            boolTemp = va_arg(args_ptr, bool);
            while (index < (set->data->currPos)) {
                if (boolTemp == voidTobool(*(set->data->value + index))) {
                    return index;
                }
                index++;
            }
            return -1;
```

```
        case FLOAT:
            floatTemp = va_arg(args_ptr, double);
            while (index < (set->data->currPos)) {
                if (fabs(floatTemp - voidTofloat(*(set->data->value +
index))) < 0.00001) {
                    return index;
                }
                index++;
            }
            return -1;

        case STRING:
            stringTemp = va_arg(args_ptr, char*);
            while (index < (set->data->currPos)) {
                if (strcmp(stringTemp, voidTostring(*(set->data->value +
index))) == 0) {
                    return index;
                }
                index++;
            }
            return -1;

        case NODE:
            nodeTemp = va_arg(args_ptr, struct Node*);
            while (index < (set->data->currPos)) {
                // printf("%s\n", nodeTemp->name);
                // printf("%s\n", voidTonode(*(set->data->value +
index))->name);
                if (strcmp(nodeTemp->name, voidTonode(*(set->data->value +
index))->name) == 0) {
                    return index;
                }
                index++;
            }
            return -1;

        default:
            break;
    }

    return -1;
}

struct Set *put_set(struct Set *set, ...) {
    // Corner case
    if (set == NULL) {
        printf("%s\n", "Error! put_set : Set does not exist.\n");
        exit(1);
    }

    void *addData;
```

```
va_list args_ptr;
va_start(args_ptr, set);
switch (set->type) {
    case INT:
        addData = intTovoid(va_arg(args_ptr, int));
        if (!check_set_element(set, voidToint(addData))) {
            set->data = plus_list(set->data, voidToint(addData));
            set->size++;
        } else {
            printf("Error! put_set : Element Already exist.\n");
            exit(1);
        }
        break;

    case FLOAT:
        addData = floatTovoid(va_arg(args_ptr, double));
        if (!check_set_element(set, voidTofloat(addData))) {
            set->data = plus_list(set->data, voidTofloat(addData));
            set->size++;
        } else {
            printf("Error! put_set : Element Already exist.\n");
            exit(1);
        }
        break;

    case BOOL:
        addData = boolTovoid(va_arg(args_ptr, bool));
        if (!check_set_element(set, voidTobool(addData))) {
            set->data = plus_list(set->data, voidTobool(addData));
            set->size++;
        } else {
            printf("Error! put_set : Element Already exist.\n");
            exit(1);
        }
        break;

    case STRING:
        addData = stringTovoid(va_arg(args_ptr, char*));
        if (!check_set_element(set, voidTostring(addData))) {
            set->data = plus_list(set->data, voidTostring(addData));
            set->size++;
        } else {
            printf("Error! put_set : Element Already exist.\n");
            exit(1);
        }
        break;

    case NODE:
        addData = nodeTovoid(va_arg(args_ptr, struct Node*));
        if (!check_set_element(set, voidTonode(addData))) {
            set->data = plus_list(set->data, voidTonode(addData));
            set->size++;
        } else {
```

```
                printf("%s\n", "Error! put_set : Element Already exist.");
                exit(1);
            }
            break;

        default:
            break;
    }

    va_end(args_ptr);

    return set;
}

struct Set *remove_set_element(struct Set *set, ...) {
    // Corner case
    if (set == NULL) {
        printf("%s\n", "Error! remove_element : Set does not exist.\n");
        exit(1);
    }

    int intTemp;
    bool boolTemp;
    double floatTemp;
    char *stringTemp;
    struct Node* nodeTemp;

    int index;

    va_list args_ptr;
    va_start(args_ptr, set);

    switch (set->type) {

        case INT:
            intTemp = va_arg(args_ptr, int);
            index = get_set_element_index(set, intTemp);
            if (index == -1) {
                printf("Error! remove_set_element : Element does not
exist.\n");
                exit(1);
            } else {
                remove_list_element(set->data, index);
                set->size--;
            }
            break;

        case BOOL:
            boolTemp = va_arg(args_ptr, bool);
            index = get_set_element_index(set, boolTemp);
            if (index == -1) {
                printf("Error! remove_set_element : Element does not
exist.\n");
```

```
                exit(1);
            } else {
                remove_list_element(set->data, index);
                set->size--;
            }
            break;

        case FLOAT:
            floatTemp = va_arg(args_ptr, double);
            index = get_set_element_index(set, floatTemp);
            if (index == -1) {
                printf("Error! remove_set_element : Element does not
exist.\n");
                exit(1);
            } else {
                remove_list_element(set->data, index);
                set->size--;
            }
            break;

        case STRING:
            stringTemp = va_arg(args_ptr, char*);
            index = get_set_element_index(set, stringTemp);
            if (index == -1) {
                printf("Error! remove_set_element : Element does not
exist.\n");
                exit(1);
            } else {
                remove_list_element(set->data, index);
                set->size--;
            }
            break;

        case NODE:
            nodeTemp = va_arg(args_ptr, struct Node*);
            index = get_set_element_index(set, nodeTemp);
            if (index == -1) {
                printf("%s\n", "Error! remove_set_element : Node does not
exist.");
                exit(1);
            } else {
                remove_list_element(set->data, index);
                set->size--;
            }
            break;

        default:
            break;
    }

    va_end(args_ptr);

    return set;
```

```
}

struct List *get_set_elements(struct Set *set) {
    // Corner case
    if (set == NULL) {
        printf("%s\n", "Error! get-set_elements : Set does not exist.\n");
        exit(1);
    }

    struct List *list;
    int i;

    switch (set->type) {
        case INT:
            list = create_list(INT);
            list->type = INT;
            list->size = set->size;
            for (i = 0; i < set->size; i++) {
                list = plus_list(list, voidToint(get_list_element(set->data,
i)));
            }
            return list;

        case FLOAT:
            list = create_list(FLOAT);
            list->type = FLOAT;
            list->size = set->size;
            for (i = 0; i < set->size; i++) {
                list = plus_list(list,
voidTofloat(get_list_element(set->data, i)));
            }
            return list;

        case BOOL:
            list = create_list(BOOL);
            list->type = BOOL;
            list->size = set->size;
            for (i = 0; i < set->size; i++) {
                list = plus_list(list, voidTobool(get_list_element(set->data,
i)));
            }
            return list;

        case STRING:
            list = create_list(STRING);
            list->type = STRING;
            list->size = set->size;
            for (i = 0; i < set->size; i++) {
                list = plus_list(list,
voidTostring(get_list_element(set->data, i)));
            }
            return list;
```

```
        case NODE:
            list = create_list(NODE);
            list->type = NODE;
            list->size = set->size;
            for (i = 0; i < set->size; i++) {
                list = plus_list(list, voidTonode(get_list_element(set->data,
i)));
            }
            return list;

        default:
            break;
    }

    return list;
}

int set_iterate(struct Set *set, Func2 f) {
    // Corner case
    if (set == NULL) {
        printf("Error! set_iterate : Set does not exist.\n");
        exit(1);
    } else if (get_set_size(set) < 0) {
        return SET_MISSING;
    }

    for (int i = 0; i < get_set_size(set); i++) {
        int status = f(set->data->value + i);
        if (status != SET_OK) {
            return status;
        }
    }

    return SET_OK;
}

int32_t get_set_type(struct Set *set) {
    if (set == NULL) {
        printf("%s\n", "Error! get_set_type : Set does not exist.\n");
        exit(1);
    }

    return set->type;
}

int32_t get_set_size(struct Set *set) {
    if (set == NULL) {
        printf("%s\n", "Error! get_set_type : Set does not exist.\n");
        exit(1);
    }

    return set->size;
}
```

```
struct Set *put_set_from_list(struct Set *set, struct List * list) {
    if (set == NULL) {
        printf("%s\n", "Error! put_set_from_list : Set does not exist.\n");
        exit(1);
    }

    int i;
    switch (set->type) {
        case INT:
            for (i = 0; i < get_list_size(list); i++) {
                set = put_set(set, voidToint(get_list_element(list, i)));
            }
            break;

        case BOOL:
            for (i = 0; i < get_list_size(list); i++) {
                set = put_set(set, voidTobool(get_list_element(list, i)));
            }
            break;

        case FLOAT:
            for (i = 0; i < get_list_size(list); i++) {
                set = put_set(set, voidTofloat(get_list_element(list, i)));
            }
            break;

        case STRING:
            for (i = 0; i < get_list_size(list); i++) {
                set = put_set(set, voidTostring(get_list_element(list, i)));
            }
            break;

        case NODE:
            for (i = 0; i < get_list_size(list); i++) {
                set = put_set(set, voidTonode(get_list_element(list, i)));
            }
            break;

        default:
            break;
    }

    return set;
}

int test_int_set_iterate(void **data) {
    printf("%s", "SET ELEMENT: ");
    printf("%d\n", voidToint(*data));

    return SET_OK;
}
```

```
int test_int_set_iterate_2(void **data) {
    int value = voidToint(*data) + 1;
    *data = intTovoid(value);
    return SET_OK;
}
```

8.1.10. node.h

```
#ifndef _NODE_H_
#define _NODE_H_

#include "config.h"
#include "utils.h"
#include "list.h"

struct Node {
        int32_t type;
        void* value;

        char* name;
        struct List* nodes;
        struct List* weight;
};

struct Node* createNode(char* name, int32_t type);
struct Node* setNodeValue(struct Node* node, ...);
char* getNodeValue(struct Node* node, int32_t type, ...);
char* getNodeName(struct Node* node);
void addNodeEdge(struct Node* node1, struct Node* node2, double weight);
void addReverseEdge(struct Node* node1, struct Node* node2, double weight);
struct Node* iterNode(struct Node* node, int index);
double weightIterNode(struct Node* node, int index);
int getNodeLength(struct Node* node);

#endif
```

8.1.11. node.c

```
#include "node.h"

struct Node* createNode(char* name, int32_t type) {
        struct Node* new = (struct Node*) malloc(sizeof(struct Node));
         new->name = name;
         new->type = type;
         new->nodes = NULL;
         new->weight = NULL;

         switch (type) {
               case INT:
                       // new->value = intTovoid(va_arg(ap, int));
                       new->value = intTovoid(0);
                       break;
               case FLOAT:
                       // new->value = floatTovoid(va_arg(ap, double));
                       new->value = floatTovoid(0);
```

```
                        break;
                case BOOL:
                        // new->value = boolTovoid(va_arg(ap, bool));
                        new->value = boolTovoid(0);
                        break;
                case STRING:
                        // new->value = stringTovoid(va_arg(ap, char*));
                        new->value = "";
                        break;
                default:
                        break;
         }

        return new;
}

struct Node* setNodeValue(struct Node* node, ...) {

        va_list ap;
        va_start(ap, node);
        switch (node->type) {
                case INT:
                        node->value = intTovoid(va_arg(ap, int));
                        break;
                case FLOAT:
                        node->value = floatTovoid(va_arg(ap, double));
                        break;
                case BOOL:
                        node->value = boolTovoid(va_arg(ap, bool));
                        break;
                case STRING:
                        node->value = stringTovoid(va_arg(ap, char*));
                        break;
                default:
                        break;
        }
        va_end(ap);

        return node;
}

char* getNodeValue(struct Node* node, int32_t type, ...) {
        if (node == NULL) {
                printf("Node does not exist.\n");
                return NULL;
        }
        va_list ap;
        va_start(ap, type);
        switch (type) {
                case INT:
                        printf("INT\n");
                        (*va_arg(ap, int*)) = voidToint(node->value);
                        break;
```

```
                case FLOAT:
                        (*va_arg(ap, double*)) = voidTofloat(node->value);
                        break;
                case BOOL:
                        (*va_arg(ap, bool*)) = voidTobool(node->value);
                        break;
                case STRING:
                        return voidTostring(node->value);
                default:
                        break;
        }
        va_end(ap);

        return "";
}

int getNodeLength(struct Node* node){
        return get_list_size(node->nodes);
}

char* getNodeName(struct Node* node) {
        if (node == NULL) {
                printf("Node does not exist.\n");
                return NULL;
        }
        return node->name;
}

void addNodeEdge(struct Node* node1, struct Node* node2, double weight){
        if (node1->nodes==NULL){
                node1->nodes = create_list(NODE);
                node1->weight = create_list(FLOAT);
        }
        // printf("EXECUTED at addNodeEdge");
        plus_list(node1->nodes, node2);
        plus_list(node1->weight, weight);
        // printf("%d\n", voidToint(newEdge->value));
        // printf("%d\n", voidToint(node->nodes->value));
}

void addReverseEdge(struct Node* node1, struct Node* node2, double weight){
        if (node2->nodes==NULL){
                node2->nodes = create_list(NODE);
                node2->weight = create_list(FLOAT);
        }
        // printf("EXECUTED at addNodeEdge");
        plus_list(node2->nodes, node1);
        plus_list(node2->weight, weight);
        // printf("%d\n", voidToint(newEdge->value));
        // printf("%d\n", voidToint(node->nodes->value));
}

struct Node* iterNode(struct Node* node, int index){
```

```
        // printf("Node %s: %d\n", node->name, index);
        int size = get_list_size(node->nodes);
        // printf("Size: %d\n", size);
        if (0<=index && index<size)
                return voidTonode(get_list_element(node->nodes, index));
        else{
                printf("Node does not exist.\n");
                return NULL;
        }
}

double weightIterNode(struct Node* node, int index){
        int size = get_list_size(node->weight);
        // printf("%d\n", size);
        if (0<=index && index<size)
                return voidTofloat(get_list_element(node->weight, index));
        else{
                printf("Node does not exist.\n");
                return 0;
        }
}

double getEdgeValue(struct Node* node1, struct Node* node2) {
        int size1 = get_list_size(node1->weight);

        for (int i = 0; i < size1; i++) {
                if (strcmp(voidTonode(get_list_element(node1->nodes,
i))->name, node2->name) == 0) {
                        // Found node
                        return voidTofloat(get_list_element(node1->weight, i));
                }
        }

        printf("%s\n", "Error! getEdgeValue : Node not found!");
        return 0;
}
```

8.1.12. graph.h

```
#ifndef _GRAPH_H_
#define _GRAPH_H_

#include "config.h"
#include "utils.h"
#include "hashmap.h"
#include "set.h"
#include "node.h"

struct Graph {
        char* name;
        struct List* nodes;
        // struct List* weight;
```

```
        struct hashmap* hashmap;
};

struct Graph* createGraph(char* name);
void addGraphNode(struct Graph* graph, struct Node* node);
int addGraphEdge(struct Graph* graph, struct Node* node1, struct Node* node2,
int weight);
struct Node* iterGraph(struct Graph* graph, int index);
struct Node* findGraphNode(struct Graph* graph, char* nodeName);

/* built-in function */
struct Node* init_tag(struct Graph* g);
struct Node* reduce(struct Graph* g, struct Node* n);
struct Node* expand(struct Graph* g, struct Node* n);
struct Graph* combine(struct Graph* g1, struct Graph* g2);
struct List* component(struct Graph* g);
struct List* bfs(struct Graph* g, struct Node* n);
struct List* dfs(struct Graph* g, struct Node* n);
struct Node* find(struct Graph* g, struct Node* n, char* lambda);// only need
one of the parameters
struct List* find_path(struct Graph* g, struct Node* n1, struct Node* n2);
struct Graph* assign(struct Graph* g, char* lambda);
struct Graph* reverse(struct Graph* g);

#endif
```

8.1.13. graph.c

```
#include "graph.h"

void print_list(struct List* l){
        if (l==NULL){
                printf("print_list NULL\n");
        }
        int size = get_list_size(l);
        int i;
        // printf("size: %d", size);

        switch (l->type) {
                case INT:
                        for (i = 0; i < size; i++) {
                                printf("%s ", voidToint(get_list_element(l,
i)));
                        }
                        printf("\n");
                        break;

                case STRING:
                        for (i = 0; i < size; i++) {
                                printf("%s ", voidTostring(get_list_element(l,
i)));
                        }
                        printf("\n");
```

```
                        break;

                case NODE:
                        for (i = 0; i < size; i++) {
                                printf("%s ", voidTonode(get_list_element(l,
i))->name);
                        }
                        printf("\n");
                        break;

                default:
                        break;
        }

        // for (int i=0;i<size;i++){
        //      struct Node* n = get_list_element(l, i);
        //      printf("%s ", n->name);
        // }
        // printf("\n");
        return;
}

void print_graph(struct Graph* g){
        printf("Printing graph %s :\n", g->name);
        int size = get_list_size(g->nodes);
        printf("size : %d\n", size);
        for (int i=0;i<size;i++){
                struct Node* n = iterGraph(g, i);
                // printf("%d: %s -> \n", i+1, n->name);
                printf("%s : %d ->\n", n->name, voidToint(n->value));
                int w_size = get_list_size(n->nodes);
                // printf("w_size : %d\n", w_size);
                for (int j=0;j<w_size;j++){
                        // printf("EXE HERE, j = %d\n", j);
                        // struct Node* m =
voidTonode(get_list_element(n->nodes, j));
                        struct Node* m = iterNode(n, j);
                        printf("( %s, %f) ", m->name, weightIterNode(n, j));
                }
                printf("\n");
        }
}

struct Graph* createGraph(char* name){
        struct Graph* new = (struct Graph*) malloc(sizeof(struct Graph));
        new->name = name;
        new->nodes = NULL;
        // new->weight = NULL;
        new->hashmap = create_hashmap(STRING, INT);
        return new;
}

struct Graph* addGraphNode(struct Graph* graph, struct Node* node){
```

```
        if (graph==NULL){
                printf("Graph does not exist.\n");
                return;
        }
        if (graph->nodes==NULL){
                graph->nodes = create_list(NODE);
                // graph->weight = create_list(LIST);
        }
        // printf("Graph List Type: %d\n", graph->nodes->type);
        // printf("addGraphNode: %s\n", node->name);
        plus_list(graph->nodes, node);
        // plus_list(graph->weight, node->weight);
        // printf("%s\n", node->name);
        // printf("%d\n", get_list_size(graph->nodes)-1);
        graph->hashmap = hashmap_put(graph->hashmap, node->name,
get_list_size(graph->nodes)-1);

        return graph;
}

struct Graph* addGraphEdge(struct Graph* graph, struct Node* node1, struct
Node* node2, int weight){
        if (hashmap_get(graph->hashmap, node1->name)==NULL){
                addGraphNode(graph, node1);
        }
        if (hashmap_get(graph->hashmap, node2->name)==NULL){
                addGraphNode(graph, node2);
        }
        addNodeEdge(node1, node2, weight);

        return graph;
}

struct Node* iterGraph(struct Graph* graph, int index){
        int size = get_list_size(graph->nodes);
        // printf("%d\n", size);
        if (0<=index && index<size)
                return (struct Node*)get_list_element(graph->nodes, index);
        else{
                printf("Node does not exist.\n");
                return NULL;
        }
}


struct Node* findGraphNode(struct Graph* graph, char* nodeName){
        void* tmp;
        if ((tmp=hashmap_get(graph->hashmap, nodeName))!=NULL){
                int index = voidToint(tmp);
                return iterGraph(graph, index);
        } else {
                return NULL;
        }
```

```
}

/* --- built-in function --- */

struct Node* init_tag(struct Graph* g){
        int size = get_list_size(g->nodes);
        for (int i=0;i<size;i++){
                struct Node* n = iterGraph(g, i);
                int w_size = get_list_size(n->nodes);
                for (int j=0;j<w_size;j++){
                        change_list_element(n->weight, j, 0);
                }
        }
}

// the type of nodes must be FLOAT
struct Node* reduce(struct Graph* g, struct Node* n0){
        struct Node* n = findGraphNode(g, n0->name);
        double v_n = 0;
        getNodeValue(n, n->type, &v_n);

        int size = get_list_size(n->nodes);
        for (int i=0;i<size;i++){
                struct Node* m = iterNode(n, i);
                double v_m = 0, tmp;
                getNodeValue(m, m->type, &v_m);
                if ((tmp = v_n + weightIterNode(n, i)) < v_m){
                        m->value = floatTovoid(tmp);
                }
        }
}

// the type of nodes must be FLOAT
struct Node* expand(struct Graph* g, struct Node* n0){
        struct Node* n = findGraphNode(g, n0->name);
        double v_n = 0;
        getNodeValue(n, n->type, &v_n);

        int size = get_list_size(n->nodes);
        for (int i=0;i<size;i++){
                struct Node* m = iterNode(n, i);
                double v_m = 0, tmp;
                getNodeValue(m, m->type, &v_m);
                if ((tmp = v_n + weightIterNode(n, i)) > v_m){
                        m->value = floatTovoid(tmp);
                }
        }
}

struct Graph* combine(struct Graph* g1, struct Graph* g2){
        int size2 = get_list_size(g2->nodes);
        for (int i=0;i<size2;i++){
                struct Node* n = iterGraph(g2, i);
```

```
                if (findGraphNode(g1, n->name)==NULL){
                        addGraphNode(g1, n);
                }
        }
        return g1;
}


struct List* bfs(struct Graph* g, struct Node* n){
        // printf("g: %s\n", g->name);
        // printf("n: %s\n", n->name);
        struct List* l = create_list(NODE);
        struct List* rec = create_list(NODE);
        struct Set* visited = create_set(NODE);

        plus_list(l, n);
        plus_list(rec, n);
        put_set(visited, n);

        while (get_list_size(l)!=0){
                struct Node* n = get_list_element(l, 0);
                int size = get_list_size(n->nodes);
                for (int i=0;i<size;i++){
                        struct Node* m = iterNode(n, i);
                        if (check_set_element(visited, m)==false){
                                plus_list(l, m);
                                plus_list(rec, m);
                                put_set(visited, m);
                        }
                }
                remove_list_element(l, 0);
        }
        return rec;
}

struct List* dfs(struct Graph* g, struct Node* n){
        struct List* l = create_list(NODE);
        struct List* rec = create_list(NODE);
        struct hashmap* m = create_hashmap(STRING, INT);

        plus_list(l, n);
        plus_list(rec, n);
        hashmap_put(m, n->name, 0);

        while (get_list_size(l)!=0){
                // print_list(l);
                struct Node* n = get_list_element(l, get_list_size(l)-1);
                // printf("dfs: %s\n", n->name);
                int size = get_list_size(n->nodes);
                int now = voidToint(hashmap_get(m, n->name));
                // printf("%s -- %d\n", n->name, now);
                if (now<size){
                        struct Node* x = iterNode(n, now);
```

```
                        if (hashmap_haskey(m, x->name)==false){
                                plus_list(l, x);
                                plus_list(rec, x);
                                hashmap_put(m, x->name, 0);
                                // printf("ADD NEW ELEMENT %s\n", x->name);
                        }
                        hashmap_remove(m, n->name);
                        hashmap_put(m, n->name, now+1);
                } else {
                        remove_list_element(l, get_list_size(l)-1);
                }
        }
        return rec;
}
```

8.1.14. utils.h

```
#ifndef UTILITIES_H
#define UTILITIES_H

#include "config.h"


#define MAP_MISSING -3  /* No such element */
#define MAP_FULL -2    /* Hashmap is full */
#define MAP_OMEM -1    /* Out of Memory */
#define MAP_OK 0    /* OK */

#define INITIAL_SIZE 256
#define MAX_CHAIN_LENGTH 8

#define SET_MISSING -1
#define SET_OK 1


typedef int (*Func)(void *, void *, void *);

typedef int (*Func2)(void **);


// Define Data Structures
struct List {
    int32_t size;
    int32_t type;
    void **value;
    int32_t currPos;
};

struct hashmap_element {
    char *key;
    int used;
    void *data[2];
};
```

```
struct hashmap {
    int tableSize;
    int size;
    int32_t keyType;
    int32_t valueType;
    struct hashmap_element *data;
};

struct Set {
    int32_t type;
    int32_t size;
    struct List *data;
};

struct Node {
    int32_t type;
    void* value;

    char* name;
    struct List* nodes;
    struct List* weight;
};

struct Graph {
    char* name;
    struct List* nodes;
    // struct List* weight;
    struct hashmap* hashmap;
};


// Define functions
/**********************************
    List Methods
**********************************/
struct List *create_list(int32_t type);

struct List *plus_list_helper(struct List *list, void *data);

struct List *plus_list(struct List *list, ...);

struct List *concat_list(struct List *list1, struct List *list2);

void *get_list_element(struct List *list, int index);

void *pop_list_element(struct List *list);

void *remove_list_element(struct List *list, int index);

int get_list_size(struct List *list);

bool check_list_element(struct List *list, ...);
```

```
/***********************************
    Hashmap Methods
***********************************/
struct hashmap *create_hashmap(int32_t keyType, int32_t valueType);

struct hashmap *hashmap_put(struct hashmap *map, ...);

int hashmap_length(struct hashmap *map);

int32_t hashmap_keytype(struct hashmap *map);

int32_t hashmap_valuetype(struct hashmap *map);

bool hashmap_haskey(struct hashmap *map, ...);

void *hashmap_get(struct hashmap *map, ...);

struct hashmap *hashmap_remove(struct hashmap *map, ...);

struct List *hashmap_keys(struct hashmap *map);

int hashmap_iterate(struct hashmap *map, Func f);


/***********************************
    Set Methods
***********************************/
struct Set *create_set(int type);

bool check_set_element(struct Set *set, ...);

struct Set *put_set(struct Set *set, ...);

int32_t get_set_element_index(struct Set *set, ...);

struct Set *remove_set_element(struct Set *set, ...);

struct List *get_set_elements(struct Set *set);

int set_iterate(struct Set *set, Func2 f);

int32_t get_set_type(struct Set *set);

int32_t get_set_size(struct Set *set);


/***********************************
    Node Methods
***********************************/
struct Node* createNode(char* name, int32_t type);
struct Node* setNodeValue(struct Node* node, ...);
```

```
char* getNodeValue(struct Node* node, int32_t type, ...);
void addNodeEdge(struct Node* node1, struct Node* node2, int weight);
void addReverseEdge(struct Node* node1, struct Node* node2, double weight);
char* nameIterNode(struct Node* node, int index);
int weightIterNode(struct Node* node, int index);
int getNodeLength(struct Node* node);
void* get_node_value(struct Node* node);

/**********************************
    Graph Methods
**********************************/
struct Graph* createGraph(char* name);
struct Graph* addGraphNode(struct Graph* graph, struct Node* node);
struct Graph* addGraphEdge(struct Graph* graph, struct Node* node1, struct
Node* node2, double weight);
struct Node* iterGraph(struct Graph* graph, int index);
struct Node* findGraphNode(struct Graph* graph, char* nodeName);

struct Node* init_tag(struct Graph* g);
struct Node* reduce(struct Graph* g, struct Node* n0);
struct Node* expand(struct Graph* g, struct Node* n0);
struct Graph* combine(struct Graph* g1, struct Graph* g2);
struct List* bfs(struct Graph* g, struct Node* n);
struct List* dfs(struct Graph* g, struct Node* n);
int graphLength(struct Graph* g);

#endif
```

8.1.15. main.c

```
#include "config.h"
#include "list.h"
#include "cast.h"
#include "hashmap.h"
#include "set.h"
#include "node.h"
#include "graph.h"
#include "utils.h"


int main() {
        // Test function: create_list
    printf("%s\n", "TEST: create_list");
    struct List *intList = create_list(0);
    printf("%d\n", intList->type);

    struct List *doubleList = create_list(1);
    printf("%d\n", doubleList->type);


    // Test function: plus_list
    printf("%s\n", "TEST: plus_list");
```

```
struct List *intListTest = create_list(INT);
struct List *doubleListTest = create_list(FLOAT);
struct List *stringListTest = create_list(STRING);
intListTest = plus_list(intListTest, 10);
doubleListTest = plus_list(doubleListTest, 10.123);
char str1[12] = "Hello";
char str2[12] = "World";
stringListTest = plus_list(stringListTest, str1);
stringListTest = plus_list(stringListTest, str2);
printf("%s\n", voidTostring(pop_list_element(stringListTest)));
printf("%s\n", voidTostring(pop_list_element(stringListTest)));
printf("%s\n", "PRINT PRINT_LIST");
print_list(intListTest);


// Test function: get_list_element, get_list_size
printf("%s\n", "TEST: get_list_element, get_list_size");
struct List *intListTest1;
intListTest1 = create_list(INT);
intListTest1 = plus_list(intListTest1, 10);
printf("%d\n", get_list_size(intListTest1));

intListTest1 = plus_list(intListTest1, 20);
printf("%d\n", get_list_size(intListTest1));

intListTest1 = plus_list(intListTest1, 30);
printf("%d\n", get_list_size(intListTest1));

printf("%d\n", intListTest1->type);
printf("%d\n", intListTest1->size);
void *intVoidPointerTest = get_list_element(intListTest1, 2);
int intTest = voidToint(intVoidPointerTest);
printf("%d\n", intTest);


// Test function: pop_list_element
printf("%s\n", "TEST: pop_list_element");
struct List *intListTest2;
intListTest2 = create_list(INT);
intListTest2 = plus_list(intListTest2, 10);
intListTest2 = plus_list(intListTest2, 20);
intListTest2 = plus_list(intListTest2, 30);
printf("%d\n", voidToint(pop_list_element(intListTest2)));
printf("%d\n", voidToint(pop_list_element(intListTest2)));
printf("%d\n", voidToint(pop_list_element(intListTest2)));
intListTest2 = plus_list(intListTest2, 10);
intListTest2 = plus_list(intListTest2, 20);
intListTest2 = plus_list(intListTest2, 30);


// Test function: remove_list_element
printf("%s\n", "TEST: remove_list_element");
struct List *intListTest3;
```

```
intListTest3 = create_list(INT);
intListTest3 = plus_list(intListTest3, 10);
intListTest3 = plus_list(intListTest3, 20);
intListTest3 = plus_list(intListTest3, 30);
printf("%d\n", get_list_size(intListTest3));
printf("%d\n", voidToint(remove_list_element(intListTest3, 0)));
printf("%d\n", get_list_size(intListTest3));
printf("%d\n", voidToint(remove_list_element(intListTest3, 0)));
printf("%d\n", get_list_size(intListTest3));
printf("%d\n", voidToint(remove_list_element(intListTest3, 0)));
printf("%d\n", get_list_size(intListTest3));
intListTest3 = plus_list(intListTest3, 40);


// Test function: check_list_element
printf("%s\n", "TEST: check_list_element");
printf("%d\n", check_list_element(intListTest2, 10));
assert(check_list_element(intListTest2, 10) == 1);
printf("%d\n", check_list_element(intListTest2, 20));
assert(check_list_element(intListTest2, 20) == 1);
printf("%d\n", check_list_element(intListTest2, 30));
assert(check_list_element(intListTest2, 30) == 1);
printf("%d\n", check_list_element(intListTest2, 40));
assert(check_list_element(intListTest2, 40) == 0);

struct List* stringListTest2 = create_list(STRING);
stringListTest2 = plus_list(stringListTest2, "hello");
stringListTest2 = plus_list(stringListTest2, "world");
printf("%s\n", voidTostring(get_list_element(stringListTest2, 1)));
printf("%d\n", check_list_element(stringListTest2, "hello"));
printf("%d\n", check_list_element(stringListTest2, "world"));
printf("%d\n", check_list_element(stringListTest2, "columbia"));


// Test function: concat_list
printf("TEST: concat_list\n");
printf("%d\n", get_list_size(intListTest2));
printf("%d\n", get_list_size(intListTest3));
struct List* concatListTest = concat_list(intListTest2, intListTest3);
printf("%d\n", get_list_size(concatListTest));


// Test function: create_hashmap
printf("%s\n", "TEST: create_hashmap");
struct hashmap *intToInt = create_hashmap(INT, INT);
struct hashmap *intToString = create_hashmap(INT, STRING);
printf("%d\n", intToInt->keyType);
printf("%d\n", intToInt->valueType);
printf("%d\n", intToString->keyType);
printf("%d\n", intToString->valueType);


// Test function: hashmap_hash_int
```

```
    printf("%s\n", "TEST: hashmap_hash_int");
    struct hashmap *intToIntTwo = create_hashmap(INT, INT);
    printf("%d\n", hashmap_hash_int(intToIntTwo, "Hello"));
    printf("%d\n", hashmap_hash_int(intToIntTwo, "World"));


    // Test function: hashmap_put
    printf("%s\n", "TEST: hashmap_put");
    struct hashmap *intToInt3 = create_hashmap(INT, INT);
    intToInt3 = hashmap_put(intToInt3, 10, 99);
    printf("%d\n", intToInt3->size);

    intToInt3 = hashmap_put(intToInt3, 11, 88);
    printf("%d\n", intToInt3->size);

    struct hashmap *stringToInt1 = create_hashmap(STRING, INT);
    printf("%d\n", stringToInt1->size);
    stringToInt1 = hashmap_put(stringToInt1, "hello", 10);
    printf("%d\n", stringToInt1->size);
    stringToInt1 = hashmap_put(stringToInt1, "world", 11);
    printf("%d\n", stringToInt1->size);


    // Test function: hashmap_length, hashmap_keytype, hashmap_valuetype
    printf("%s\n", "TEST: hashmap_length, hashmap_keytype,
hashmap_valuetype");
    printf("%d\n", hashmap_length(stringToInt1));
    printf("%d\n", hashmap_keytype(stringToInt1));
    printf("%d\n", hashmap_valuetype(stringToInt1));


    // Test function: hashmap_hashkey
    printf("%s\n", "TEST: hashmap_haskey");
    printf("%d\n", hashmap_haskey(stringToInt1, "hello"));
    printf("%d\n", hashmap_haskey(stringToInt1, "world"));
    printf("%d\n", hashmap_haskey(stringToInt1, "columbia"));
    printf("%d\n", hashmap_haskey(intToInt3, 10));
    printf("%d\n", hashmap_haskey(intToInt3, 11));
    printf("%d\n", hashmap_haskey(intToInt3, 12));


    // Test function: hashmap_get
    printf("%s\n", "TEST: hashmap_get");
    printf("%d\n", voidToint(hashmap_get(stringToInt1, "hello")));
    printf("%d\n", voidToint(hashmap_get(stringToInt1, "world")));
    //printf("%d\n", voidToint(hashmap_get(stringToInt1, "columbia")));
    printf("%d\n", voidToint(hashmap_get(intToInt3, 10)));
    printf("%d\n", voidToint(hashmap_get(intToInt3, 11)));
    //printf("%d\n", voidToint(hashmap_get(intToInt3, 12)));


    // Test function: hashmap_remove
    printf("%s\n", "TEST: hashmap_remove");
```

```
    stringToInt1 = hashmap_remove(stringToInt1, "hello");
    assert(hashmap_length(stringToInt1) == 1);
    stringToInt1 = hashmap_remove(stringToInt1, "world");
    assert(hashmap_length(stringToInt1) == 0);
    stringToInt1 = hashmap_put(stringToInt1, "hello", 10);
    assert(hashmap_length(stringToInt1) == 1);
    stringToInt1 = hashmap_put(stringToInt1, "world", 100);
    assert(hashmap_length(stringToInt1) == 2);

    intToInt3 = hashmap_remove(intToInt3, 10);
    assert(hashmap_length(intToInt3) == 1);
    intToInt3 = hashmap_put(intToInt3, 20, 100);
    assert(hashmap_length(intToInt3) == 2);


    // Test function: hashmap_keys
    struct List *stringKeysList = hashmap_keys(stringToInt1);
    for (int i = 0; i < get_list_size(stringKeysList); i++) {
        printf("%s\n", voidTostring(get_list_element(stringKeysList, i)));
    }
    struct List *intKeysList = hashmap_keys(intToInt3);
    for (int i = 0; i < get_list_size(intKeysList); i++) {
        printf("%d\n", voidToint(get_list_element(intKeysList, i)));
    }


    // Test function: hashmap_iterate
    struct hashmap* intToInt4 = create_hashmap(INT, INT);
    intToInt4 = hashmap_put(intToInt4, 1, 10);
    intToInt4 = hashmap_put(intToInt4, 2, 20);
    intToInt4 = hashmap_put(intToInt4, 3, 30);
    intToInt4 = hashmap_put(intToInt4, 4, 40);
    intToInt4 = hashmap_put(intToInt4, 5, 50);
    printf("%s\n", "TEST: hashmap_iterate");
    int status = hashmap_iterate(intToInt4,
test_inttoint_hashmap_iterate_func);


    // Test function: create_set, get_set_type
    printf("%s\n", "TEST: create_set");
    struct Set *intSet = create_set(INT);
    printf("%d\n", intSet->type);
    printf("%d\n", intSet->data->type);
    printf("%d\n", intSet->size);
    printf("%d\n", get_set_type(intSet));

    struct Set *stringSet = create_set(STRING);
    printf("%d\n", stringSet->type);
    printf("%d\n", stringSet->data->type);
    printf("%d\n", stringSet->size);
    printf("%d\n", get_set_type(stringSet));
```

```
// Test function: put_set, get_set_size
printf("%s\n", "TEST: put_set, get_set_size");
struct Set *intSet2 = create_set(INT);
intSet2 = put_set(intSet2, 1);
printf("%d\n", get_set_size(intSet2));
assert(get_set_size(intSet2) == 1);
intSet2 = put_set(intSet2, 2);
printf("%d\n", get_set_size(intSet2));
assert(get_set_size(intSet2) == 2);
intSet2 = put_set(intSet2, 3);
printf("%d\n", get_set_size(intSet2));
assert(get_set_size(intSet2) == 3);

struct Set *stringSet2 = create_set(STRING);
stringSet2 = put_set(stringSet2, "hello");
printf("%d\n", get_set_size(stringSet2));
assert(get_set_size(stringSet2) == 1);
stringSet2 = put_set(stringSet2, "world");
printf("%d\n", get_set_size(stringSet2));
assert(get_set_size(stringSet2) == 2);
printf("%d\n", check_set_element(stringSet2, "hello"));
printf("%d\n", check_set_element(stringSet2, "world"));
printf("%d\n", check_set_element(stringSet2, "columbia"));


// Test function: get_set_elements
printf("%s\n", "TEST: get_set_elements");
struct List *intList1 = get_set_elements(intSet2);
printf("%d\n", get_list_size(intList1));
struct List *stringList1 = get_set_elements(stringSet2);
printf("%d\n", get_list_size(stringList1));


// Test function: set_iterate
printf("%s\n", "TEST: set_iterate");
status = set_iterate(intSet2, test_int_set_iterate);
status = set_iterate(intSet2, test_int_set_iterate_2);
status = set_iterate(intSet2, test_int_set_iterate);


// Test function: get_set_element_index
printf("%s\n", "TEST: get_set_element_index");
printf("%d\n", get_set_element_index(intSet2, 1));
printf("%d\n", get_set_element_index(intSet2, 2));
printf("%d\n", get_set_element_index(intSet2, 3));
printf("%d\n", get_set_element_index(intSet2, 4));
// struct Set* nodeSet1 = create_set(NODE);
// struct Node* setTestNode1 = createNode("setTestNode1", INT);
// struct Node* setTestNode2 = createNode("setTestNode2", INT);
// nodeSet1 = put_set(nodeSet1, setTestNode1);
// nodeSet1 = put_set(nodeSet1, setTestNode2);
// printf("%d\n", get_set_element_index(nodeSet1, setTestNode1));
```

```
    // Test function: remove_set_element
    printf("%s\n", "TEST: remove_set_element");
    intSet2 = remove_set_element(intSet2, 2);
    printf("%d\n", get_set_size(intSet2));
    printf("%d\n", intSet2->data->currPos);
    intSet2 = remove_set_element(intSet2, 3);
    printf("%d\n", get_set_size(intSet2));
    printf("%d\n", intSet2->data->currPos);
    struct Set* nodeSet1 = create_set(NODE);
    struct Node* setTestNode1 = createNode("setTestNode1", INT);
    struct Node* setTestNode2 = createNode("setTestNode2", INT);
    nodeSet1 = put_set(nodeSet1, setTestNode1);
    nodeSet1 = put_set(nodeSet1, setTestNode2);
    printf("%d\n", get_set_size(nodeSet1));
    nodeSet1 = remove_set_element(nodeSet1, setTestNode1);
    printf("%d\n", get_set_size(nodeSet1));


    // Test function: addNodeEdge
    printf("%s\n", "TEST: addNodeEdge");
    struct Node* addNodeEdgeTestNode1 = createNode("addNodeEdgeTestNode1",
INT);
    struct Node* addNodeEdgeTestNode2 = createNode("addNodeEdgeTestNode2",
INT);
    addNodeEdge(addNodeEdgeTestNode1, addNodeEdgeTestNode2, 10);
    printf("%f\n", getEdgeValue(addNodeEdgeTestNode1, addNodeEdgeTestNode2));


    return 0;
}
```

8.2. ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or  | Mod

type uop = Neg | Not

type typ = Int | Bool | Void | Node of typ | Float | String | List of typ |
Set of typ | Map of typ * typ | Graph | Edge | Null_t

type bind = typ * string

type expr =
    Literal of int
  | BoolLit of bool
  | Null
  | FloatLit of float
  | StringLit of string
```

```
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | AddAssign of string * expr
  | MinusAssign of string * expr
  | AddAdd of string
  | SingleEdge of  string * string
  | Call of string * expr list
  | DotCall of string * string * expr list
  | Subscript of string * expr
  | ListLiteral of expr list
  | Noexpr
  | SingleLinkAssign of expr * expr
  | DoubleLinkAssign of string * string * expr
(*   | SubscriptAssign of expr * expr
 *)  | BatchSingleLinkAssign of string * expr * expr
  | New of string
  | BatchDoubleLinkAssign of string * expr * expr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
```

```
  | Mod -> "%"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | FloatLit(f) -> string_of_float f
  | StringLit(s) -> "\"" ^ s ^ "\""
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Null -> "null"
  | Id(s) -> s
  | AddAdd(s) -> s ^ "++"
  | New(s) -> "new " ^ s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
(*   | SubscriptAssign(subscript, e) -> string_of_expr subscript ^ " = " ^
string_of_expr e
 *)  | AddAssign(v, e) -> v ^ " += " ^ string_of_expr e
  | MinusAssign(v, e) -> v ^ " -= " ^ string_of_expr e
  | SingleEdge(e1, e2) ->  e1 ^ " -> " ^  e2
  | SingleLinkAssign(e1 , e) -> string_of_expr e1 ^ " = " ^ string_of_expr e
  | DoubleLinkAssign(n1, n2, e) -> n1 ^ " -- " ^ n2 ^ " = " ^ string_of_expr
e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | DotCall(d, f, el) ->
     d ^ "." ^ f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
")"
  | Noexpr -> ""
  | Subscript(var1, e) -> var1 ^ "[" ^ string_of_expr e ^ "]"
  | ListLiteral(el) -> "@{" ^ String.concat ", " (List.map string_of_expr el)
^ "}"
  | BatchSingleLinkAssign(var1, e2, e3) ->  var1 ^ " -> " ^ string_of_expr e2
^ " = " ^ string_of_expr e3
  | BatchDoubleLinkAssign(var1, e2, e3) ->  var1 ^ " -- " ^ string_of_expr e2
^ " = " ^ string_of_expr e3

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt
s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
```

```
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Node(t1) -> "node" ^ "@{" ^ string_of_typ t1 ^ "}"
  | Float -> "float"
  | String -> "string"
  | List(t1) -> "list" ^ "@{" ^ string_of_typ t1 ^ "}"
  | Set(t1) -> "set" ^ "@{" ^ string_of_typ t1 ^ "}"
  | Map(t1, t2) -> "map" ^ "@{" ^ string_of_typ t1 ^ ", " ^ string_of_typ t2
^ "}"
  | Graph -> "graph"
  | Edge -> "edge"
  | Null_t -> "null"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.3. parser.mly

```
/* Ocamlyacc parser for TuSimple */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA LEFTSQUAREBRACKET
RIGHTSQUAREBRACKET
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT ADDASSIGN MINUSASSIGN MOD DOT NEW
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR SINGLELINK DOUBLELINK ADDADD AT
NULL MAXINT MININT
%token RETURN IF ELSE FOR WHILE INT BOOL VOID NODE FLOAT STRING LIST SET MAP
GRAPH

%token <string> ID

%token <int> LITERAL
%token <float> FLOAT_LITERAL
```

```
%token <string> STRING_LITERAL
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN ADDASSIGN MINUSASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT NEG
%left ADDADD
%left SINGLELINK DOUBLELINK

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { [], [] }
 | decls vdecl { ($2 :: fst $1), snd $1 }
 | decls fdecl { fst $1, ($2 :: snd $1) }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $1;
         fname = $2;
         formals = $4;
         locals = List.rev $7;
         body = List.rev $8 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    typ ID                   { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | STRING { String }
  | GRAPH { Graph }
  | LIST AT LBRACE typ RBRACE { List($4) }
```

```
  | SET AT LBRACE typ RBRACE { Set($4) }
  | NODE AT LBRACE typ RBRACE { Node($4) }
  | MAP AT LBRACE typ COMMA typ RBRACE { Map($4, $6) }


vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl_batch { $2 @ $1 }

vdecl:
    typ ID SEMI { ($1, $2) }

vdecl_batch:
    typ id_list { List.map (fun id -> ($1, id)) $2 }

id_list:
    id_element SEMI { List.rev $1 }

id_element:
    ID    { [$1] }
  | id_element COMMA ID { $3 :: $1 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    LITERAL          { Literal($1) }
  | NEW ID           { New($2) }
  | MAXINT           { Literal(1073741823) }
  | MININT           { Literal(-1073741824) }
  | FLOAT_LITERAL    { FloatLit($1) }
  | STRING_LITERAL   { StringLit($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | NULL             { Null }
  | ID               { Id($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
```

```
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr MOD    expr { Binop($1, Mod,   $3) }
  | ID ADDADD { AddAdd($1) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ADDASSIGN expr { AddAssign($1, $3) }
  | ID MINUSASSIGN expr { MinusAssign($1, $3) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | singleEdge ASSIGN expr { SingleLinkAssign($1, $3) }
  | ID DOUBLELINK ID ASSIGN expr { DoubleLinkAssign($1, $3, $5) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | ID DOT ID LPAREN actuals_opt RPAREN { DotCall($1, $3, $5) }
  | LPAREN expr RPAREN { $2 }
  | ID SINGLELINK lists ASSIGN lists { BatchSingleLinkAssign($1, $3, $5) }
  | ID DOUBLELINK lists ASSIGN lists { BatchDoubleLinkAssign($1, $3, $5) }
  | lists { $1 }
  | singleEdge { $1 }
  | subscript { $1 }



singleEdge:
   ID SINGLELINK ID { SingleEdge($1, $3) }

subscript:
   ID LEFTSQUAREBRACKET expr RIGHTSQUAREBRACKET { Subscript($1, $3) }

lists:
  | AT LBRACE list_literals { ListLiteral($3) }


 list_literals:
    RBRACE { [] }
 | listElements RBRACE { List.rev $1 }

listElements:
    expr    { [$1] }
  | listElements COMMA expr  { $3 :: $1 }



actuals_opt:
```

```
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

8.4. codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)


let translate (globals, functions) =
        let context = L.global_context () in
        let llctx = L.global_context () in
        let customM = L.MemoryBuffer.of_file "Lib/utils.bc" in
        let llm = Llvm_bitreader.parse_bitcode llctx customM in
        let the_module = L.create_module context "TuSimple"
        and i32_t  = L.i32_type  context
        and i8_t   = L.i8_type   context
        and i1_t   = L.i1_type   context
        and void_t = L.void_type context
        and float_t  = L.double_type context
        and string_t  = L.pointer_type (L.i8_type context)
        and list_t = L.pointer_type (match L.type_by_name llm "struct.List"
with
                None -> raise (Failure "struct.List doesn't defined.")
        | Some x -> x)
        and set_t = L.pointer_type (match L.type_by_name llm "struct.Set" with
                None -> raise (Failure "struct.Set doesn't defined.")
        | Some x -> x)
        and map_t = L.pointer_type (match L.type_by_name llm "struct.hashmap"
with
                None -> raise (Failure "struct.hashmap doesn't defined.")
        | Some x -> x)
```

```
        and node_t = L.pointer_type (match L.type_by_name llm "struct.Node"
with
                None -> raise (Failure "struct.Node doesn't defined.")
        | Some x -> x)
        and graph_t = L.pointer_type (match L.type_by_name llm "struct.Graph"
with
                None -> raise (Failure "struct.Graph doesn't defined.")
        | Some x -> x)

        in

        let ltype_of_typ = function
                A.Int -> i32_t
        | A.Bool -> i1_t
        | A.Void -> void_t
        | A.List(_) -> list_t
        | A.Set(_) -> set_t
        | A.Map(_, _) -> map_t
        | A.String -> string_t
        | A.Float -> float_t
        | A.Node(_) -> node_t
        | A.Graph -> graph_t
        |  _ -> raise (Failure ("[Error] Type Not Found for ltype_of_typ."))
        in

        let lconst_of_typ = function
                A.Int -> L.const_int i32_t 0
        | A.Float -> L.const_int i32_t 1
        | A.Bool -> L.const_int i32_t 2
        | A.String -> L.const_int i32_t 3
        | A.Node _ -> L.const_int i32_t 4
        | A.Graph -> L.const_int i32_t 5
        | _ -> raise (Failure ("[Error] Type Not Found for lconst_of_typ."))

        in

(*
=====================================================================
        List Methods
=====================================================================
*)

        let get_list_size_t = L.function_type i32_t [| list_t |]
        in
        let get_list_size_f = L.declare_function "get_list_size"
get_list_size_t the_module
        in
        let get_list_size list_ptr llbuilder =
                        let actuals = [| list_ptr |] in (
                                L.build_call get_list_size_f actuals
"get_list_size" llbuilder
                        )
        in
```

```
        let pop_list_element_t = L.function_type (L.pointer_type i8_t) [|
list_t |]
        in
        let pop_list_element_f = L.declare_function "pop_list_element"
pop_list_element_t the_module
        in
        let pop_list_element list_ptr llbuilder =
                        let actuals = [| list_ptr |] in (
                                L.build_call pop_list_element_f actuals
"pop_list_element" llbuilder
                        )
        in

        let real_remove_list_element_t = L.function_type (L.pointer_type i8_t)
[| list_t; i32_t |]
        in
        let real_remove_list_element_f = L.declare_function
"remove_list_element" real_remove_list_element_t the_module
        in
        let real_remove_list_element list_ptr index llbuilder =
                        let actuals = [| list_ptr; index |] in (
                                L.build_call real_remove_list_element_f actuals
"remove_list_element" llbuilder
                        )
        in

        let remove_list_element_t = L.function_type (L.pointer_type i8_t)
[|list_t; i32_t|]
        in
        let remove_list_element_f = L.declare_function "remove_list_element"
remove_list_element_t the_module
        in
        let remove_list_element list_ptr llbuilder =
                        let actuals = [|list_ptr; L.const_int i32_t 0|] in(
                                L.build_call remove_list_element_f actuals
"remove_list_element" llbuilder
                        )
        in
        let get_list_element_t = L.function_type (L.pointer_type i8_t) [|
list_t ; i32_t |]
        in
        let get_list_element_f = L.declare_function "get_list_element"
get_list_element_t the_module
        in
        let get_list_element list_ptr i llbuilder =
                let actuals =[|list_ptr; i|] in (
                        L.build_call get_list_element_f actuals
"get_list_element" llbuilder
                )
        in
```

```
        let concat_list_t = L.var_arg_function_type list_t [| list_t ; list_t
|]
        in

        let concat_list_f = L.declare_function "concat_list" concat_list_t
the_module
        in

        let concat_list l1_ptr l2_ptr llbuilder =
                let actuals = [|l1_ptr; l2_ptr|] in (
                        L.build_call concat_list_f actuals "concat_list"
llbuilder
                )

        in

        let create_list_t  = L.function_type list_t [| i32_t |]
        in
        let create_list_f  = L.declare_function "create_list" create_list_t
the_module
        in
        let create_list typ llbuilder =
        let actuals = [|lconst_of_typ typ|]in (
                L.build_call create_list_f actuals "create_list" llbuilder
        )
        in

        let add_list_t  = L.var_arg_function_type list_t [| list_t |]
        in

        let add_list_f  = L.declare_function "plus_list" add_list_t the_module
        in

        let add_list l_ptr llbuilder data =
        let actuals = [| l_ptr; data|] in
                ignore (L.build_call add_list_f actuals "plus_list" llbuilder)
        in

        let print_list_t = L.function_type (L.pointer_type i8_t)  [| list_t |]
        in
        let print_list_f = L.declare_function "print_list" print_list_t
the_module
        in
        let print_list l_ptr llbuilder =
                let actuals = [|l_ptr|] in (
                        L.build_call print_list_f actuals "print_list"
llbuilder
                )
        in


(*
============================================================
```

```
        Hashmap Methods
=============================================================
*)
        let create_hashmap_t = L.function_type map_t [| i32_t; i32_t |]
        in

        let create_hashmap_f = L.declare_function "create_hashmap"
create_hashmap_t the_module
        in
        let create_hashmap kType vType llbuilder =
                let actuals = [| lconst_of_typ kType; lconst_of_typ vType |]
in
                        (L.build_call create_hashmap_f actuals "create_hashmap"
llbuilder)
        in

        let hashmap_put_t = L.var_arg_function_type map_t [| map_t |]
        in

        let hashmap_put_f = L.declare_function "hashmap_put" hashmap_put_t
the_module
        in

        let hashmap_put m_ptr key value llbuilder  =
                let actuals = [| m_ptr; key; value |] in
                        L.build_call hashmap_put_f actuals "hashmap_put"
llbuilder
        in

        let hashmap_get_t = L.var_arg_function_type (L.pointer_type i8_t) [|
map_t |]
        in

        let hashmap_get_f = L.declare_function "hashmap_get" hashmap_get_t
the_module
        in

        let hashmap_get m_ptr data llbuilder  =
                let actuals = [| m_ptr; data |] in
                        L.build_call hashmap_get_f actuals "hashmap_get"
llbuilder
        in

        let hashmap_length_t = L.function_type i32_t [| map_t |]
        in
        let hashmap_length_f = L.declare_function "hashmap_length"
hashmap_length_t the_module
        in
        let hashmap_length m_ptr llbuilder =
                let actuals = [| m_ptr |] in
                        L.build_call hashmap_length_f actuals "hashmap_length"
llbuilder
        in
```

```
        let hashmap_haskey_t = L.var_arg_function_type i1_t [| map_t |]
        in
        let hashmap_haskey_f = L.declare_function "hashmap_haskey"
hashmap_haskey_t the_module
        in
        let hashmap_haskey m_ptr data llbuilder =
                let actuals = [| m_ptr; data |] in (
                        L.build_call hashmap_haskey_f actuals "hashmap_haskey"
llbuilder
                )
        in

        let hashmap_remove_t = L.var_arg_function_type map_t [| map_t |]
        in
        let hashmap_remove_f = L.declare_function "hashmap_remove"
hashmap_remove_t the_module
        in
        let hashmap_remove m_ptr data llbuilder =
                let actuals = [| m_ptr; data |] in (
                        L.build_call hashmap_remove_f actuals "hashmap_remove"
llbuilder
                )
        in
(*
=============================================================
        Set Methdos
=============================================================
*)

let put_set_from_list_t = L.function_type set_t [| set_t; list_t |]
in
let put_set_from_list_f = L.declare_function "put_set_from_list"
put_set_from_list_t the_module
in
let put_set_from_list s_ptr l_ptr llbuilder =
        let actuals = [| s_ptr; l_ptr |] in (
                L.build_call put_set_from_list_f actuals "put_set_from_list"
llbuilder
        )
in

let create_set_t = L.function_type set_t [| i32_t |]
in
let create_set_f = L.declare_function "create_set" create_set_t the_module
in
let create_set typ llbuilder =
        let actuals = [|lconst_of_typ typ|] in (
                L.build_call create_set_f actuals "create_set" llbuilder
        )
in

let add_set_t = L.var_arg_function_type set_t [| set_t |]
```

```
in
let add_set_f = L.declare_function "put_set" add_set_t the_module
in
let add_set s_ptr data llbuilder =
        let actuals = [|s_ptr; data|] in
                L.build_call add_set_f actuals "put_set" llbuilder
in


        let add_all_elements_into_list element_list l_ptr llbuilder =
                List.iter (add_list l_ptr llbuilder) element_list
        in

        let get_set_size_t = L.function_type i32_t [| set_t |]
        in
        let get_set_size_f = L.declare_function "get_set_size" get_set_size_t
the_module
        in
        let get_set_size s_ptr llbuilder =
                        let actuals = [| s_ptr |] in (
                                L.build_call get_set_size_f actuals
"get_set_size" llbuilder
                        )
        in

        let check_set_element_t = L.var_arg_function_type i1_t [| set_t |]
        in
        let check_set_element_f = L.declare_function "check_set_element"
check_set_element_t the_module
        in
        let check_set_element s_ptr data llbuilder =
                        let actuals = [| s_ptr; data |] in (
                                L.build_call check_set_element_f actuals
"check_set_element" llbuilder
                        )
        in

        let remove_set_element_t = L.var_arg_function_type set_t [| set_t |]
        in
        let remove_set_element_f = L.declare_function "remove_set_element"
remove_set_element_t the_module
        in
        let remove_set_element s_ptr data llbuilder =
                        let actuals = [| s_ptr; data |] in (
                                        L.build_call remove_set_element_f
actuals "remove_set_element" llbuilder
                        )
        in


(*
=============================================================
        Node Methods
```

```
============================================================
*)
let createNode_t = L.function_type node_t [| string_t; i32_t |]
in
let createNode_f = L.declare_function "createNode" createNode_t the_module
in
let createNode s_ptr nodeType llbuilder =
        let actuals = [| s_ptr; lconst_of_typ nodeType |] in (
                L.build_call createNode_f actuals "createNode" llbuilder
        )
in

let getEdgeValue_t = L.function_type i32_t [| node_t; node_t |]
in
let getEdgeValue_f = L.declare_function "getEdgeValue" getEdgeValue_t
the_module
in
let getEdgeValue n1_ptr n2_ptr llbuilder =
        let actuals = [| n1_ptr; n2_ptr |] in (
                L.build_call getEdgeValue_f actuals "getEdgeValue" llbuilder
        )
in

let addNodeEdge_t = L.function_type i32_t [| node_t; node_t; i32_t |]
in
let addNodeEdge_f = L.declare_function "addNodeEdge" addNodeEdge_t the_module
in
let addNodeEdge n1_ptr n2_ptr weight llbuilder =
        let actuals = [| n1_ptr; n2_ptr; weight |] in (
                ignore(L.build_call addNodeEdge_f actuals "addNodeEdge"
llbuilder)
        )
in

let addReverseEdge_t = L.function_type i32_t [| node_t; node_t; i32_t |]
in
let addReverseEdge_f = L.declare_function "addReverseEdge" addReverseEdge_t
the_module
in
let addReverseEdge n1_ptr n2_ptr weight llbuilder =
        let actuals = [| n1_ptr; n2_ptr; weight |] in (
                ignore(L.build_call addReverseEdge_f actuals "addReverseEdge"
llbuilder)
        )
in

let getNodeName_t = L.function_type string_t [| node_t |]
in
let getNodeName_f = L.declare_function "getNodeName" getNodeName_t the_module
in
let getNodeName n_ptr llbuilder =
        let actuals = [| n_ptr |] in (
                L.build_call getNodeName_f actuals "getNodeName" llbuilder
```

```
        )
in

let setNodeValue_t = L.var_arg_function_type node_t [| node_t |]
in
let setNodeValue_f = L.declare_function "setNodeValue" setNodeValue_t
the_module
in
let setNodeValue n_ptr data llbuilder =
        let actuals = [|n_ptr; data|] in
                L.build_call setNodeValue_f actuals "setNodeValue" llbuilder
in

let iterNode_t = L.function_type node_t [| node_t; i32_t|]
in
let iterNode_f = L.declare_function "iterNode" iterNode_t the_module
in
let iterNode n_ptr index llbuilder =
                let actuals = [|n_ptr; index|] in (
                        L.build_call iterNode_f actuals "iterNode" llbuilder
                )
in

let getNodeLength_t = L.function_type i32_t [| node_t |]
in
let getNodeLength_f = L.declare_function "getNodeLength" getNodeLength_t
the_module
in
let getNodeLength n_ptr llbuilder =
                let actuals = [| n_ptr |] in (
                        L.build_call getNodeLength_f actuals "getNodeLength"
llbuilder
                )
in

let get_node_value_t = L.function_type (L.pointer_type i8_t) [| node_t |]
in
let get_node_value_f = L.declare_function "get_node_value" get_node_value_t
the_module
in
let get_node_value n_ptr llbuilder =
                let actuals = [| n_ptr |] in (
                                L.build_call get_node_value_f actuals
"get_node_value" llbuilder
                )
in

let weigthIterNode_t = L.function_type i32_t [| node_t; i32_t |]
in
let weightIterNode_f = L.declare_function "weightIterNode" weigthIterNode_t
the_module
in
let weightIterNode n_ptr index llbuilder =
```

```
                let actuals = [|n_ptr; index|] in (
                                L.build_call weightIterNode_f actuals
"weightIterNode" llbuilder
                )
in


(*
=============================================================
        Graph Methods
=============================================================
*)

let graphLength_t = L.function_type i32_t [| graph_t |]
in
let graphLength_f = L.declare_function "graphLength" graphLength_t the_module
in
let graphLength g_ptr llbuilder =
        let actuals = [| g_ptr |] in (
                L.build_call graphLength_f actuals "graphLength" llbuilder
        )
in

let createGraph_t = L.function_type graph_t [| string_t |]
in
let createGraph_f = L.declare_function "createGraph" createGraph_t the_module
in
let createGraph s_ptr llbuilder =
        let actuals = [| s_ptr |] in (
                L.build_call createGraph_f actuals "createGraph" llbuilder
        )
in

let addGraphNode_t = L.function_type graph_t [| graph_t; node_t |]
in
let addGraphNode_f = L.declare_function "addGraphNode" addGraphNode_t
the_module
in
let addGraphNode g_ptr n_ptr llbuilder =
        let actuals = [| g_ptr; n_ptr |] in
                L.build_call addGraphNode_f actuals "addGraphNode" llbuilder
in

let addGraphEdge_t = L.function_type graph_t [| graph_t; node_t; node_t;
i32_t |]
in
let addGraphEdge_f = L.declare_function "addGraphEdge" addGraphEdge_t
the_module
in
let addGraphEdge g_ptr n1_ptr n2_ptr w llbuilder =
        let actuals = [| g_ptr; n1_ptr; n2_ptr; w |] in
                L.build_call addGraphEdge_f actuals "addGraphEdge" llbuilder
in
```

```
let iterGraph_t = L.function_type node_t [| graph_t; i32_t |]
in
let iterGraph_f = L.declare_function "iterGraph" iterGraph_t the_module
in
let iterGraph g_ptr id llbuilder=
        let actuals = [| g_ptr; id |] in
                L.build_call iterGraph_f actuals "iterGraph" llbuilder
in

let findGraphNode_t = L.function_type node_t [| graph_t; string_t |]
in
let findGraphNode_f = L.declare_function "findGraphNode" findGraphNode_t
the_module
in
let findGraphNode g_ptr name llbuilder=
        let actuals = [| g_ptr; name |] in
                L.build_call findGraphNode_f actuals "findGraphNode" llbuilder
in

let initTag_t = L.function_type node_t [| graph_t |]
in
let initTag_f = L.declare_function "init_tag" initTag_t the_module
in
let initTag g_ptr llbuilder=
        let actuals = [| g_ptr |] in
                L.build_call initTag_f actuals "init_tag" llbuilder
in

let reduce_t = L.function_type node_t [| graph_t; node_t |]
in
let reduce_f = L.declare_function "reduce" reduce_t the_module
in
let reduce g_ptr n_ptr llbuilder=
        let actuals = [| g_ptr; n_ptr |] in
                L.build_call reduce_f actuals "reduce" llbuilder
in

let expand_t = L.function_type node_t [| graph_t; node_t |]
in
let expand_f = L.declare_function "expand" expand_t the_module
in
let expand g_ptr n_ptr llbuilder=
        let actuals = [| g_ptr; n_ptr |] in
                L.build_call expand_f actuals "expand" llbuilder
in

let combine_t = L.function_type graph_t [| graph_t; graph_t |]
in
let combine_f = L.declare_function "combine" combine_t the_module
in
let combine g1_ptr g2_ptr llbuilder=
        let actuals = [| g1_ptr; g2_ptr |] in
```

```
                L.build_call combine_f actuals "combine" llbuilder
in

let bfs_t = L.function_type list_t [| graph_t; node_t |]
in
let bfs_f = L.declare_function "bfs" bfs_t the_module
in
let bfs g_ptr n_ptr llbuilder=
        let actuals = [| g_ptr; n_ptr |] in
                L.build_call bfs_f actuals "bfs" llbuilder
in

let dfs_t = L.function_type list_t [| graph_t; node_t |]
in
let dfs_f = L.declare_function "dfs" dfs_t the_module
in
let dfs g_ptr n_ptr llbuilder=
        let actuals = [| g_ptr; n_ptr |] in
                L.build_call dfs_f actuals "dfs" llbuilder
in

let print_graph_t = L.function_type (L.pointer_type i8_t)  [| graph_t |]
in
let print_graph_f = L.declare_function "print_graph" print_graph_t the_module
in
let print_graph g_ptr llbuilder =
        let actuals = [| g_ptr |] in
                L.build_call print_graph_f actuals "print_graph" llbuilder
in

(*
=====================================================================
        Cast Methods
=====================================================================
*)
let voidToint_t = L.function_type i32_t [|L.pointer_type i8_t|]
in
let voidToint_f = L.declare_function "voidToint" voidToint_t the_module
in
let voidToint v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidToint_f actuals "voidToint" llbuilder
in

let voidTofloat_t = L.function_type float_t [|L.pointer_type i8_t|]
in
let voidTofloat_f = L.declare_function "voidTofloat" voidTofloat_t the_module
in
let voidTofloat v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidTofloat_f actuals "voidTofloat" llbuilder
in
```

```
let voidTobool_t = L.function_type i1_t [|L.pointer_type i8_t|]
in
let voidTobool_f = L.declare_function "voidTobool" voidTobool_t the_module
in
let voidTobool v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidTobool_f actuals "voidTobool" llbuilder
in

let voidTostring_t = L.function_type string_t [|L.pointer_type i8_t|]
in
let voidTostring_f = L.declare_function "voidTostring" voidTostring_t
the_module
in
let voidTostring v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidTostring_f actuals "voidTostring" llbuilder
in

let voidTonode_t = L.function_type node_t [|L.pointer_type i8_t|]
in
let voidTonode_f = L.declare_function "voidTonode" voidTonode_t the_module
in
let voidTonode v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidTonode_f actuals "voidTonode" llbuilder
in

let voidTograph_t = L.function_type graph_t [|L.pointer_type i8_t|]
in
let voidTograph_f = L.declare_function "voidTograph" voidTograph_t the_module
in
let voidTograph v_ptr llbuilder =
        let actuals = [|v_ptr|] in
        L.build_call voidTograph_f actuals "voidTograph" llbuilder
in

        (* Declare each global variable; remember its value in a map *)
        let global_vars =
                let global_var m (t, n) =
                        let init = L.const_int (ltype_of_typ t) 0
                        in StringMap.add n (L.define_global n init the_module,
t) m in
                List.fold_left global_var StringMap.empty globals in

        (* Declare printf(), which the print built-in function will call *)
        let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
in
        let printf_func = L.declare_function "printf" printf_t the_module in

        (* Declare the built-in printbig() function *)
        let printbig_t = L.function_type i32_t [| i32_t |] in
```

```
        let printbig_func = L.declare_function "printbig" printbig_t
the_module in

        (* Define each function (arguments and return type) so we can call it
*)
        let function_decls =
                let function_decl m fdecl =
                        let name = fdecl.A.fname
                        and formal_types = Array.of_list (List.map (fun (t,_)
-> ltype_of_typ t) fdecl.A.formals)
                        in let ftype = L.function_type (ltype_of_typ
fdecl.A.typ) formal_types in
                        StringMap.add name (L.define_function name ftype
the_module, fdecl) m in
                List.fold_left function_decl StringMap.empty functions in

        (* Fill in the body of the given function *)
        let build_function_body fdecl =
                let (the_function, _) = StringMap.find fdecl.A.fname
function_decls in
                let builder = L.builder_at_end context (L.entry_block
the_function) in

                let int_format_str = L.build_global_stringptr "%d\n" "fmt"
builder in
                let string_format_str = L.build_global_stringptr "%s\n" "fmt"
builder
                in
                (* Construct the function's "locals": formal arguments and
locally
                        declared variables.  Allocate each on the stack,
initialize their
                        value, if appropriate, and remember their values in
the "locals" map *)
                let local_vars =
                        let add_formal m (t, n) p = L.set_value_name n p;
                                let local = L.build_alloca (ltype_of_typ t) n
builder in
                                ignore (L.build_store p local builder);
                                StringMap.add n (local, t) m in

                        let add_local m (t, n) =
                                let local_var = L.build_alloca (ltype_of_typ t)
n builder
                                in StringMap.add n (local_var, t) m in

                        let formals = List.fold_left2 add_formal
StringMap.empty fdecl.A.formals
                                (Array.to_list (L.params the_function)) in
                                List.fold_left add_local formals fdecl.A.locals
in
```

```
                (* Return the value/type tuple for a variable or formal
argument *)
                let lookup n = try StringMap.find n local_vars
                       with Not_found -> StringMap.find n global_vars
                in

                let get_llvm_from_llvm_asttype_tuple (ltype,_) = ltype
                in
                let type_conversion typ elementPtr builder = match typ with
                         A.Int -> voidToint elementPtr builder
                       | A.Float -> voidTofloat elementPtr builder
                       | A.Bool -> voidTobool elementPtr builder
                       | A.Node _ -> voidTonode elementPtr builder
                       | A.Graph -> voidTograph elementPtr builder
                       | A.String -> voidTostring elementPtr builder
                       | _ -> raise (Failure (" undefined operator[] "))
                 in
                (* Construct code for an expression; return its value/type
tuple *)
                let rec expr builder = function
                               A.Literal i -> (L.const_int i32_t i, A.Int)
                       | A.FloatLit i -> (L.const_float float_t i, A.Float)
                       | A.BoolLit b -> (L.const_int i1_t (if b then 1 else
0), A.Bool)
                       | A.StringLit s -> (L.build_global_stringptr s s
builder, A.String)
                       | A.Null ->  (L.const_null list_t, A.List(A.Int))
                       | A.Noexpr -> (L.const_int i32_t 0, A.Void)
                       | A.Id s -> (L.build_load (fst (lookup s)) s builder,
(snd (lookup s)))
                       | A.ListLiteral el ->
                                       let listLiteral_type = snd (expr
builder (List.hd el))
                                       in
                                       let listPtr = create_list
listLiteral_type builder
                                       in
                                       ignore(add_all_elements_into_list
                                               (List.map
get_llvm_from_llvm_asttype_tuple (List.map (expr builder) el)) listPtr
builder);
                                               (listPtr, listLiteral_type)
                       | A.MinusAssign(var, e) ->
                                       let (var', typ) =  lookup var in
                                               ((match typ with
                                               | A.Int ->
                                                               let e1' =
L.build_load var' var builder
                                                               and (e2', _) =
expr builder e
                                                               in
                                                               L.build_store
(L.build_sub e1' e2' "tmp" builder) var' builder
```

```
                                                | A.Float ->
                                                                let e1' =
L.build_load var' var builder
                                                                and (e2', _) =
expr builder e
                                                                in
                                                                L.build_store
(L.build_fsub e1' e2' "tmp" builder) var' builder
                                                | _ -> raise (Failure (" minus
assign error "))), typ)
                        | A.Subscript (var, e) -> let (var', typ) = lookup var
and (s', _) = expr builder e in
                                                ((match typ with
                                                | A.List typeList -> (
                                                        let elementPtr =
get_list_element (L.build_load var' var builder) s' builder
                                                                in
type_conversion typeList elementPtr builder
                                                                )
                                                | A.Map(_, t2) -> (
                                                        let elementPtr =
hashmap_get (L.build_load var' var builder) s' builder
                                                                in
type_conversion t2 elementPtr builder
                                                                )
                                                | _ -> raise (Failure ("
undefined operator[] "))), typ)
                        | A.New id -> let (_, typ) = lookup id in
                                        ((match typ with
                                                | A.List listType ->
L.build_store (create_list listType builder) (fst (lookup id)) builder
                                                | A.Set setType -> L.build_store
(create_set setType builder) (fst (lookup id)) builder
                                                | A.Map(kType, vType) ->
L.build_store (create_hashmap kType vType builder) (fst (lookup id)) builder
                                                | A.Node nodeType -> let
nodeName = L.build_global_stringptr id "" builder
                                                                in L.build_store
(createNode nodeName nodeType builder) (fst (lookup id)) builder
                                                | A.Graph -> let graphName =
L.build_global_stringptr id "" builder
                                                                in L.build_store
(createGraph graphName builder) (fst (lookup id)) builder
                                                | _ -> raise (Failure (" Type
not found "))), typ)
                        | A.AddAdd var -> let (var', typ) = lookup var in
                                        ignore(remove_list_element
(L.build_load var' var builder) builder);
                                        (var',typ)
                        | A.Binop (e1, op, e2) ->
                                        let (e1', t1') = expr builder e1
                                        and (e2', _) = expr builder e2 in
                                        ((match op with
```

```
                                                    A.Add    -> L.build_add
                                                  | A.Sub     -> L.build_sub
                                                  | A.Mult    -> L.build_mul
                                                  | A.Div     -> L.build_sdiv
                                                  | A.And     -> L.build_and
                                                  | A.Or      -> L.build_or
                                                  | A.Mod -> L.build_srem
                                                  | A.Equal   -> L.build_icmp
L.Icmp.Eq
                                                  | A.Neq     -> L.build_icmp
L.Icmp.Ne
                                                  | A.Less    -> L.build_icmp
L.Icmp.Slt
                                                  | A.Leq     -> L.build_icmp
L.Icmp.Sle
                                                  | A.Greater -> L.build_icmp
L.Icmp.Sgt
                                                  | A.Geq     -> L.build_icmp
L.Icmp.Sge
                                                  ) e1' e2' "tmp" builder, t1')
                          | A.Unop(op, e) ->
                                        let (e', t') = expr builder e in
                                        ((match op with
                                                    A.Neg     -> L.build_neg
                                                  | A.Not     -> L.build_not) e'
"tmp" builder, t')
                          | A.Assign (s, e) ->
                                        let (e', t') = expr builder e in
                                        ignore (L.build_store e' (fst (lookup
s)) builder);
                                        (e', t')
                          | A.AddAssign (var, e) ->
                                let (var', typ) =  lookup var and (s', _) =
expr builder e in
                                ((match typ with
                                        | A.List _ -> L.build_store
(concat_list (L.build_load var' var builder) s' builder) (fst (lookup var))
builder
                                        | A.Int ->
                                                    let e1' = L.build_load
var' var builder
                                                        and (e2', _) =
expr builder e
                                                        in
                                                        L.build_store
(L.build_add e1' e2' "tmp" builder) var' builder
                                        | A.Float  ->
                                                    let e1' = L.build_load
var' var builder
                                                        and (e2', _) =
expr builder e
                                                        in
```

```
                                                                L.build_store
(L.build_fadd e1' e2' "tmp" builder) var' builder
                                        | A.Set _ -> let s1 = L.build_load var'
var builder
                                                        and (l1', _) = expr
builder e
                                                in put_set_from_list s1 l1'
builder
                                        | _ -> raise (Failure (" undefined +=
")))), typ)

                        | A.SingleEdge (n1, n2) ->
                                        ((let (n1', _) = lookup n1 and (n2', _)
= lookup n2 in
                                                        getEdgeValue
(L.build_load n1' n1 builder) (L.build_load n2' n2 builder) builder),
A.Float)
                        | A.DoubleLinkAssign (var1, var2, e) ->
                                        (let (var1', _) = lookup var1 and
(var2', _) = lookup var2 in
                                                let (s', _) = expr builder e in
                                                        ignore(addNodeEdge
(L.build_load var1' var1 builder) (L.build_load var2' var2 builder) s'
builder);
                                                        ignore(addNodeEdge
(L.build_load var2' var2 builder) (L.build_load var1' var1 builder) s'
builder)); (L.const_int i32_t 0, A.Float)
                        | A.SingleLinkAssign(e1, e) ->
                                        ((match e1 with
                                                        | A.SingleEdge(n1, n2)
-> let (n1', _) = lookup n1 and (n2', _) = lookup n2
                                                                        in let(s', _) =
expr builder e in

        ignore((addNodeEdge (L.build_load n1' n1 builder) (L.build_load n2' n2
builder) s' builder)); s'
                                                        | _ -> raise
(Failure("illegal edge assignment. "))), A.Int)

                        | A.BatchSingleLinkAssign(var, n1, n2) ->
                                (let getListsOfLit n =
                                        (match n with
                                        | A.ListLiteral li -> li
                                        | _ -> raise (Failure("Error at
BatchSingleLinkAssign: match fail")))
                                in
                                        let f elem = fst (expr builder elem)
                                        in
                                        let n1_sequence = List.map f
(getListsOfLit n1)
                                        in
                                        let n2_sequence = List.map f
(getListsOfLit n2)
```

```
                                        in
                                        let (var', _) = lookup var
                                        in
                                        let addNodeIter v1 v2 = addNodeEdge
(L.build_load var' var builder) v1 v2 builder
                                        in
                                                ignore(List.iter2 addNodeIter
n1_sequence n2_sequence)

                                        );
                                        (L.const_int i32_t 0, A.Float)

                        | A.BatchDoubleLinkAssign(var, n1, n2) ->
                                (let getListsOfLit n =
                                        (match n with
                                        | A.ListLiteral li -> li
                                        | _ -> raise (Failure("Error at
BatchDoubleLinkAssign : match fail")))
                                in
                                        let f elem = fst (expr builder elem)
                                        in
                                        let n1_sequence = List.map f
(getListsOfLit n1)
                                        in
                                        let n2_sequence = List.map f
(getListsOfLit n2)
                                        in
                                        let (var', _) = lookup var
                                        in
                                        let addNodeIter v1 v2 = addNodeEdge
(L.build_load var' var builder) v1 v2 builder
                                        in
                                        let addReverseIter v1 v2 =
addReverseEdge (L.build_load var' var builder) v1 v2 builder
                                        in
                                                ignore(List.iter2 addNodeIter
n1_sequence n2_sequence);
                                                ignore(List.iter2 addReverseIter
n1_sequence n2_sequence)
                                        );
                                        (L.const_int i32_t 0, A.Float)

                        | A.DotCall (dname, fname, actuals) -> let (dname',
dtype) = lookup dname in
                                (match dtype with
                                        A.Node n_type ->
                                                (match fname with
                                                        "value" ->  (let
nodeValuePtr = get_node_value (L.build_load dname' dname builder) builder

                                                in type_conversion n_type
nodeValuePtr builder, n_type)
```

```
                                                    | "name" -> (getNodeName
(L.build_load dname' dname builder) builder, A.String)
                                                    | "setvalue" ->
(setNodeValue (L.build_load dname' dname builder)

                                                    (fst (expr builder
(List.nth actuals 0) )) builder, A.Void)
                                                    | "iterNode" -> (let arg
= List.nth actuals 0

                                                    in let index = fst (expr
builder arg)

                                                    in iterNode
(L.build_load dname' dname builder) index builder, A.Node(n_type))
                                                    | "length" ->
(getNodeLength (L.build_load dname' dname builder) builder, A.Int)
                                                    | "weightIter" -> (let
arg = List.nth actuals 0

                                                            in let index =
fst (expr builder arg)

                                                            in
weightIterNode (L.build_load dname' dname builder) index builder, A.Int)
                                                    | _ -> raise (Failure
("Error! Node has no such method")))
                                | A.List ele_type ->
                                        (match fname with
                                                    "get" -> (let arg =
List.nth actuals 0
                                                            in let index =
fst (expr builder arg)
                                                            in let
listElementPtr = get_list_element (L.build_load dname' dname builder) index
builder
                                                            in
type_conversion ele_type listElementPtr builder, ele_type)
                                                | "pop" -> (let listElementPtr =
pop_list_element (L.build_load dname' dname builder) builder

                                    in type_conversion ele_type listElementPtr
builder, ele_type)
                                                | "remove" -> (let arg =
List.nth actuals 0
                                                            in let index =
fst (expr builder arg)
                                                            in let
listElementPtr = real_remove_list_element (L.build_load dname' dname builder)
index builder
                                                            in
type_conversion ele_type listElementPtr builder, ele_type)
```

```
                                                | "length" -> (get_list_size
(L.build_load dname' dname builder) builder, A.Int)
                                                | "cancat" -> (let arg =
List.nth actuals 0
                                                                in let listPtr =
fst (expr builder arg)
                                                                in concat_list
(L.build_load dname' dname builder) listPtr builder, ele_type)
                                                | "printList" -> (print_list
(L.build_load dname' dname builder) builder, A.Void)
                                                | _ -> raise (Failure ("Error!
List has no such method")))
                        | A.Set ele_type ->
                                (match fname with
                                                "put" -> (add_set (L.build_load
dname' dname builder)
                                                                        (fst
(expr builder (List.nth actuals 0) )) builder, A.Void)
                                                | "length" -> (get_set_size
(L.build_load dname' dname builder) builder, A.Int)
                                                | "contain" -> (check_set_element
(L.build_load dname' dname builder)
                                                                (fst (expr
builder (List.nth actuals 0) )) builder, A.Bool)
                                                | "remove" -> (remove_set_element

        (L.build_load dname' dname builder)
                                                                        (fst
(expr builder (List.nth actuals 0)))
                                                                        builder,
A.Set(ele_type))
                                                        | _ -> raise (Failure
("Error! Set has no such method")))
                        | A.Map (_, v_type) ->
                                (match fname with
                                                "put" -> (hashmap_put
                                                (L.build_load dname' dname
builder)
                                                (fst (expr builder (List.nth
actuals 0)))
                                                (fst (expr builder (List.nth
actuals 1) )) builder
                                                                        ,
A.Void)
                                 | "get" -> (let listElementPtr = (hashmap_get

                                                        (L.build_load dname'
dname builder)
                                                        (fst (expr builder
(List.nth actuals 0))) builder)
                                                        in type_conversion
v_type listElementPtr builder, v_type)
```

```
                                | "size" -> (hashmap_length (L.build_load
dname' dname builder) builder, A.Int)
                                | "haskey" -> (let arg = List.nth actuals 0
                                                                        in let
mapKey = fst (expr builder arg)
                                                                        in
hashmap_haskey (L.build_load dname' dname builder) mapKey builder, A.Bool)
                                | "remove" -> (let arg = List.nth actuals 0
                                                                    in let mapKey =
fst (expr builder arg)
                                                                    in
hashmap_remove (L.build_load dname' dname builder) mapKey builder, A.Void)
                                | _ -> raise (Failure ("Error! Map has no such
method")))
                        | A.Graph ->
                                (match fname with
                                        "bfs" -> (let arg = List.nth actuals 0
                                                                    in let nodePtr =
fst (expr builder arg)
                                                                    in bfs
(L.build_load dname' dname builder) nodePtr builder
                                                                    ,
A.List(A.Node(A.Int)))
                                | "dfs" -> (let arg = List.nth actuals 0
                                                                in let nodePtr = fst
(expr builder arg)
                                                                in dfs (L.build_load
dname' dname builder) nodePtr builder
                                                                , A.List(A.Node(A.Int)))
                                | "iterGraph" -> (iterGraph
                                                                (L.build_load
dname' dname builder)
                                                                (fst (expr
builder (List.nth actuals 0)))
                                                                builder
                                                                , A.Node(A.Int))
                                | "findGraphNode" -> (findGraphNode

        (L.build_load dname' dname builder)
                                                                        (fst
(expr builder (List.nth actuals 0)))
                                                                        builder
                                                                        ,
A.Node(A.Int))
                                | "init" -> (initTag (L.build_load dname' dname
builder) builder, A.Void)
                                | "addNode" -> (addGraphNode (L.build_load
dname' dname builder)
                                                                         (fst
(expr builder (List.nth actuals 0)))
                                                                         builder
                                                                , A.Void)
```

```
                                | "addEdge" -> (addGraphEdge (L.build_load
dname' dname builder)
                                                            (fst (expr
builder (List.nth actuals 0)))
                                                            (fst (expr
builder (List.nth actuals 1)))
                                                            (fst (expr
builder (List.nth actuals 2)))
                                                            builder
                                                           , A.Void)
                                | "relax" -> (reduce (L.build_load dname' dname
builder)
                                                           (fst (expr
builder (List.nth actuals 0)))
                                                           builder
                                                           , A.Node(A.Int))
                                | "expand" -> (expand (L.build_load dname'
dname builder)
                                                           (fst (expr
builder (List.nth actuals 0)))
                                                           builder
                                                           , A.Node(A.Int))
                                | "combine" -> (combine (L.build_load dname'
dname builder)
                                                           (fst (expr
builder(List.nth actuals 0)))
                                                           builder
                                                           , A.Graph)
                                | "length" -> (graphLength (L.build_load dname'
dname builder) builder, A.Int)
                                | "printGraph" -> (print_graph (L.build_load
dname' dname builder) builder, A.Void)
                                | _ -> raise (Failure ("Error! Graph has no
such method")))
                                | _ -> raise (Failure ("Error! Do not support
such type")))
                        | A.Call ("print", [e]) -> (L.build_call printf_func [|
int_format_str ; (fst (expr builder e)) |]
                                "printf" builder, (snd (expr builder e)))
                        | A.Call ("prints", [e]) ->
                                (L.build_call printf_func [|
string_format_str ; (fst (expr builder e)) |]
                                "printf" builder, (snd (expr builder
e)))
                        | A.Call ("printbig", [e]) ->
                                (L.build_call printbig_func [| (fst (expr
builder e)) |]
                                "printbig" builder, (snd (expr builder e)))
                        | A.Call (f, act) ->
                                let (fdef, fdecl) = StringMap.find f
function_decls in
                                let actuals = List.rev (List.map fst (List.map
(expr builder) (List.rev act))) in
```

```
                                let result = (match fdecl.A.typ with A.Void ->
""


                | _ -> f ^ "_result") in
                                        (L.build_call fdef (Array.of_list actuals)
result builder, fdecl.A.typ)
                in

                (* Invoke "f builder" if the current block doesn't already
                        have a terminal (e.g., a branch). *)
                let add_terminal builder f =
                        match L.block_terminator (L.insertion_block builder)
with
                                Some _ -> ()
                        | None -> ignore (f builder) in

                (* Build the code for the given statement; return the builder
for
                        the statement's successor *)
                let rec stmt builder = function
                                A.Block sl -> List.fold_left stmt builder sl
                        | A.Expr e -> ignore (fst (expr builder e)); builder
                        | A.Return e -> ignore (match fdecl.A.typ with
                                A.Void -> L.build_ret_void builder
                                | _ -> L.build_ret (fst (expr builder e))
builder); builder
                        | A.If (predicate, then_stmt, else_stmt) ->
                                let bool_val = fst (expr builder predicate) in
                                let merge_bb = L.append_block context "merge"
the_function in

                                let then_bb = L.append_block context "then"
the_function in
                                add_terminal (stmt (L.builder_at_end context
then_bb) then_stmt)
                                        (L.build_br merge_bb);

                                let else_bb = L.append_block context "else"
the_function in
                                add_terminal (stmt (L.builder_at_end context
else_bb) else_stmt)
                                        (L.build_br merge_bb);

                                ignore (L.build_cond_br bool_val then_bb
else_bb builder);
                                L.builder_at_end context merge_bb

                        | A.While (predicate, body) ->
                                let pred_bb = L.append_block context "while"
the_function in
                                ignore (L.build_br pred_bb builder);
```

```
                                let body_bb = L.append_block context
"while_body" the_function in
                                add_terminal (stmt (L.builder_at_end context
body_bb) body)
                                        (L.build_br pred_bb);

                                let pred_builder = L.builder_at_end context
pred_bb in
                                let bool_val = fst (expr pred_builder
predicate) in

                                let merge_bb = L.append_block context "merge"
the_function in
                                ignore (L.build_cond_br bool_val body_bb
merge_bb pred_builder);
                                L.builder_at_end context merge_bb

                        | A.For (e1, e2, e3, body) -> stmt builder
                        ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ;
A.Expr e3]) ] )
                in

                (* Build the code for each statement in the function *)
                let builder = stmt builder (A.Block fdecl.A.body) in

                (* Add a return if the last block falls off the end *)
                add_terminal builder (match fdecl.A.typ with
                                A.Void -> L.build_ret_void
                        | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
        in

        List.iter build_function_body functions;
        the_module
```

8.5. scanner.mll

```
(* Ocamllex scanner for TuSimple *)

{ open Parser
  let unescape s =
        Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let digit = ['0'-'9']
let float = '-'?(digit+) ['.'] digit+
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let ascii = ([' '-'!' '#'-'[' ']'-'~'])

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
```

```
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['      { LEFTSQUAREBRACKET }
| ']'      { RIGHTSQUAREBRACKET }
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '='       { ASSIGN }
| "+="      { ADDASSIGN }
| "-="     { MINUSASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "&&"      { AND }
| "||"      { OR }
| "!"       { NOT }
| "->"     { SINGLELINK }
| "--"     { DOUBLELINK }
| "@"       { AT }
| "//"     { signleLineComment lexbuf }
| "."       { DOT }

| "maxint" { MAXINT }
| "minint" { MININT }


| "%"      { MOD }
| "++"     { ADDADD }
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "return" { RETURN }

| "int"     { INT }
| "float"   { FLOAT }
| "bool"    { BOOL }
| "void"    { VOID }
| "string" { STRING }

| "node"    { NODE }
| "list"    { LIST }
| "set"     { SET }
| "map"     { MAP }
| "graph"   { GRAPH }
```

```
| "true"   { TRUE }
| "false"  { FALSE }
| "null"   { NULL }

| "new" { NEW }

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| float as lxm { FLOAT_LITERAL(float_of_string lxm) }
| '"' ((ascii | escape)* as lit) '"' { STRING_LITERAL(unescape lit) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }

and signleLineComment = parse
  "\n" { token lexbuf }
| _ {signleLineComment lexbuf}
```

8.6. semant.ml

```
(* Semantic checking for the TuSimple compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
        n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned to
```

```
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet = rvaluet then lvaluet else
     if rvaluet = Null_t
     then (match lvaluet with
        List t1 -> List(t1)
      | Set t1 -> Set(t1)
      |  _ -> raise err)
    else raise err
  in

  let checkAddAssign lvaluet rvaluet err =
     if ((lvaluet = Int || lvaluet = Float) && (rvaluet = Int || rvaluet =
Float)) || ((rvaluet = String && rvaluet = String))
     then if lvaluet = rvaluet
                then lvaluet
                else raise err
     else match lvaluet with
       List type1  when type1 = rvaluet -> Void
     | List _ when lvaluet = rvaluet -> Void
     | Set type1  when type1 = rvaluet -> Void
     | Set type1 -> (match rvaluet with
                   List type2 when type1 = type2 -> Void
                 |  Set type2 when type1 = type2 -> Void
                 | _ -> raise err)
     | Map (type1, _) when  type1 = rvaluet -> Void
     | Map (type1, _ ) -> (match rvaluet with
                   List type2 when type1 = type2 -> Void
                 |  Set type2 when type1 = type2 -> Void
                 | _ -> raise err)
     | _ -> raise err
  in

  let checkMinusAssign lvaluet rvaluet err =
     if (lvaluet = Int || lvaluet = Float) && (rvaluet = Int || rvaluet =
Float)
     then if lvaluet = rvaluet then lvaluet else raise err
     else match lvaluet with
       List type1  when type1 = rvaluet -> Void
     | Set type1  when type1 = rvaluet -> Void
     | Map (type1, _) when  type1 = rvaluet -> Void
     | _ -> raise err
  in

  let checkLinkAssign node1Type node2Type exprType err=
     if node1Type = node2Type
     then match node1Type with
     | Node(_) -> (if exprType = Int || exprType = Float
                then Int
                else raise err)
     | _ -> raise err
     else raise err
  in
```

```
  let checkBatchLinkAssign t1 t2 t3 err =
    match t1 with
    | Node _ -> (match t2 with
          List _ -> (match t3 with
                     | List _-> Int
                     | _ -> raise err)
        | _ -> raise err)
    | _ -> raise err
  in

  let checkSubscript varType rt err =
    if varType = String && rt = Int
      then varType
    else match varType with
       List type1  when rt = Int -> type1
     | Map (type1, type2) when  type1 = rt -> type2
     | _ -> raise err
  in

  let checkAddAdd tp err =
    match tp with
       List _ -> tp
     | _          -> raise err
  in
  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();

  report_duplicate (fun n -> "duplicate function " ^ n)
    (List.map (fun fd -> fd.fname) functions);

  (* Function declaration for a named function *)
  let built_in_decls =  StringMap.add "print"
     { typ = Void; fname = "print"; formals = [(Int, "x")];
       locals = []; body = [] } (StringMap.add "prints"
     { typ = Void; fname = "prints"; formals = [(String, "x")];
       locals = []; body = [] } (StringMap.singleton "printbig"
     { typ = Void; fname = "printbig"; formals = [(Int, "x")];
       locals = []; body = [] }))
   in

  let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd
m)
                         built_in_decls functions
  in
```

```
  let function_decl s = try StringMap.find s function_decls
       with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

  let _ = function_decl "main" in (* Ensure "main" is defined *)

  let check_function func =

    List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
      " in " ^ func.fname)) func.formals;

    report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
      (List.map snd func.formals);

    List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
      " in " ^ func.fname)) func.locals;

    report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
      (List.map snd func.locals);

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
          StringMap.empty (globals @ func.formals @ func.locals )
    in

    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    let checkHomogeneous elementType err tp = if tp = elementType then
ignore(Int) else raise err
    in

    (* Return the type of an expression or throw an exception *)
    let rec expr = function
            Literal _ -> Int
      | BoolLit _ -> Bool
      | FloatLit _ -> Float
      | StringLit _ -> String
      | Null -> Null_t
      | New id -> let idTyp = type_of_identifier id
      in
      let checkNew typ err =
        (match typ with
        | Node _ -> Null_t
        | List _ -> Null_t
        | Set _  -> Null_t
        | Map(_,_)  -> Null_t
        | Graph -> Null_t
        | _ -> raise err)
```

```
      in checkNew idTyp (Failure ("illegal new operation of " ^ string_of_typ
idTyp))
      | ListLiteral el ->  let checkListLit el err =
         match List.length el  with
           0 -> raise (Failure ("can't define empty list in this way in
TuSimple "))
         | _ -> let elementType = expr (List.hd el)
                in ignore(List.iter (checkHomogeneous elementType err)
(List.map expr el)) ; List(elementType)
        in
        checkListLit el (Failure ("illegal list definition" ^ " {" ^
String.concat ", " (List.map string_of_expr el)
          ^ "}. Type Must be homogeneous in a list"))
      | Id s -> type_of_identifier s
      | AddAdd s -> let tp = type_of_identifier s in
        checkAddAdd tp (Failure ("illegal next operation " ^ string_of_typ tp
^
             " ++ "))
      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
        (match op with
            Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
          | Add | Sub | Mult | Div when t1 = Float && t2 = Float -> Float
          | Add | Sub | Mult | Div when t1 = Float && t2 = Int -> Float
          | Add | Sub | Mult | Div when t1 = Int && t2 = Float -> Float
          | Add | Sub | Mult | Div when (t1 = Int || t1 = Int) && (t2 = Int
|| t2 = Int) -> Int
          | Add | Sub | Mult | Div when (t1 = Int || t1 = Float) && (t2 =
Float || t2 = Int) -> Float
          | Add when t1 = String && t2 = String -> String
          | Add when t1 = Bool && t2 = Bool -> Bool
          | Mod when t1 = Int && t2 = Int -> Int
              | Equal | Neq when t1 = t2 -> Bool
          | Equal | Neq when t1 = Null_t -> Bool
          | Equal | Neq when t2 = Null_t -> Bool
              | Less | Leq | Greater | Geq when (t1 = Int || t1 = Float) &&
(t1 = Float || t2 = Int) -> Bool
              | And | Or when t1 = Bool && t2 = Bool -> Bool
          | _ -> raise (Failure ("illegal binary operator " ^
                    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                    string_of_typ t2 ^ " in " ^ string_of_expr e))
        )
      | Unop(op, e) as ex -> let t = expr e in
         (match op with
           Neg when t = Int -> Int
         | Neg when t = Float -> Float
         | Not when t = Bool -> Bool
         | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op
^
                         string_of_typ t ^ " in " ^ string_of_expr ex)))
      | Noexpr -> Void
      | Subscript(var, e) as ex -> let varType = type_of_identifier var
                          and rt = expr e in
```

```
        checkSubscript varType rt (Failure ("illegal subscript  " ^
string_of_typ varType ^
             " = " ^ string_of_typ rt ^ " in " ^
             string_of_expr ex))
      | Assign(var, e) as ex -> let lt = type_of_identifier var
                                and rt = expr e in
        check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt
^
                                     " = " ^ string_of_typ rt ^ " in " ^
                                     string_of_expr ex))
      | AddAssign(var, e) as ex -> let lt = type_of_identifier var
                                   and rt = expr e in
        checkAddAssign lt rt
                       (Failure ("illegal add assignment " ^ string_of_typ
lt ^ " += " ^
                         string_of_typ rt ^ " in " ^ string_of_expr ex))
      | MinusAssign(var, e) as ex -> let lt = type_of_identifier var
                                     and rt = expr e in
        checkMinusAssign lt rt
                       (Failure ("illegal add assignment " ^ string_of_typ
lt ^ " -= " ^
                         string_of_typ rt ^ " in " ^ string_of_expr ex))
      | SingleEdge(e1, e2) as ex -> let lt = type_of_identifier e1
                                    and rt = type_of_identifier e2 in
        let err = (Failure ("illegal get edge " ^ string_of_typ lt ^ " -> " ^
                         string_of_typ rt ^ " in " ^ string_of_expr ex))
      in
        (match lt with
        | Node _ -> (match rt with
          | Node _ -> Int
          | _ -> raise err)
        | _ -> raise err)
      | SingleLinkAssign(e1, e) as ex -> let tp1 = expr e1
                                         and tp2 = expr e in
        if tp1 = Int && (tp2 = Int || tp2 = Float)
        then tp2
        else raise (Failure ("illegal single directed edge assignment " ^
string_of_typ tp1 ^ " = "
                     ^ string_of_typ tp2 ^ " in " ^ string_of_expr ex))
      | DoubleLinkAssign(var1, var2, e) as ex -> let node1Type =
type_of_identifier var1
                                                 and node2Type =
type_of_identifier var2
                                                 and exprType = expr e in
        checkLinkAssign node1Type node2Type exprType
                  (Failure ("illegal undirected edge assignment " ^
string_of_typ node1Type ^ " -> " ^
                         string_of_typ node2Type ^ " = " ^ string_of_typ
exprType ^ " in " ^ string_of_expr ex))
      | BatchSingleLinkAssign(var1, e2, e3) as ex -> let type1 =
type_of_identifier var1
                                                     and type2 = expr e2
                                                     and type3 = expr e3 in
```

```
        checkBatchLinkAssign type1 type2 type3
                  (Failure ("illegal batch singlelink assignment" ^
string_of_expr ex ^ " in "^ string_of_typ type1 ^ " -> "
                    ^ string_of_typ type2 ^ " = " ^ string_of_typ type3))
      | BatchDoubleLinkAssign(var1, e2, e3) as ex -> let type1 =
type_of_identifier var1
                                                     and type2 = expr e2
                                                     and type3 = expr e3 in
        checkBatchLinkAssign type1 type2 type3
                  (Failure ("illegal batch undirected edge assignment" ^
string_of_expr ex ^ " in "^ string_of_typ type1 ^ " -> "
                    ^ string_of_typ type2 ^ " = " ^ string_of_typ type3))
      | Call(fname, actuals) as call -> let fd = function_decl fname in
         if List.length actuals != List.length fd.formals then
           raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^ string_of_expr
call))
         else
           List.iter2 (fun (ft, _) e -> let et = expr e in
              ignore (check_assign ft et
                (Failure ("illegal actual argument found " ^ string_of_typ et
^
                " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr
e))))
             fd.formals actuals;
           fd.typ
       | DotCall(dname, fname, actuals) -> let typ = type_of_identifier dname
in
         (match typ with
            Node typn ->
            (match fname with
                "value" ->
                  if actuals = [] then typn else raise (Failure ("Node get
method error"))
              | "name" ->
                  if actuals = [] then String else raise (Failure ("Node name
method error"))
              | "setvalue" ->
                  (match actuals with
                      [x] when ((expr x) = Int) || ((expr x) = String)
                            || ((expr x) = Float) || ((expr x) = Bool) ->
Null_t
                    | _ -> raise (Failure ("Set setvalue method error")))
              | "iterNode" ->
                  (match actuals with
                      [x] when (expr x) = Int -> Node typn
                    | _ -> raise (Failure ("Error! Wrong argument in
iterNode")))
              | "length" ->
                  if actuals = [] then Int else raise (Failure ("Error! Wrong
argument in Node.length()"))
              | "weightIter" ->
                  (match actuals with
```

```
                    [x] when (expr x) = Int -> Int
                  | _ -> raise (Failure ("Set weightIter method error")))
            | _ -> raise (Failure ("Node has no such method"))
          )
        | List ele_type ->
          (match fname with
              "get" ->
                (match actuals with
                    [x] when (expr x) = Int -> ele_type
                  | _ -> raise (Failure ("List get method error")))
            | "pop" ->
                if actuals = [] then Null_t else raise (Failure ("List pop
method error"))
            | "remove" ->
                (match actuals with
                    [x] when (expr x) = Int -> Null_t
                  | _ -> raise (Failure ("List remove method error")))
            | "length" ->
                if actuals = [] then Int else raise (Failure ("List length
method error"))
            | "cancat" ->
                (match actuals with
                    [x] when (expr x) = List ele_type -> List ele_type
                  | _ -> raise (Failure ("List cancatenate method error")))
            | "printList" ->
                if actuals = [] then Null_t else raise (Failure ("List
print_list method error"))
            | _ -> raise (Failure ("List has no such method"))
          )
        | Set ele_type ->
          (match fname with
              "put" ->
                (match actuals with
                    [x] when (expr x) = ele_type -> Null_t
                  | _ -> raise (Failure ("Set put method error")))
            | "length" ->
                if actuals = [] then Int else raise (Failure ("Set length
method error"))
            | "contain" ->
                (match actuals with
                    [x] when (expr x) = ele_type -> Bool
                  | _ -> raise (Failure ("Set contain method error")))
            | "remove" ->
                (match actuals with
                    [x] when (expr x) = ele_type -> Set ele_type
                  | _ -> raise (Failure ("Set remove method error")))
            | _ -> raise (Failure ("Set has no such method"))
          )
        | Map (k_type, v_type) ->
          (match fname with
              "get" ->
                (match actuals with
                    [x] when (expr x) = k_type -> v_type
```

```
                    | _ -> raise (Failure ("Map get method error")))
              | "put" ->
              (let x = List.nth actuals 0
                   and y =  List.nth actuals 1
                    in  if (expr x) = k_type  && (expr y) = v_type
                  then Void
                else raise (Failure ("Map put method error")))
              | "size" ->
                if actuals = [] then Int else raise (Failure ("Map length
method error"))
              | "haskey" ->
                (match actuals with
                    [x] when (expr x) = k_type -> Bool
                  | _ -> raise (Failure ("Map haskey method error")))
              | "remove" ->
                   (match actuals with
                     [x] when (expr x) = k_type -> Null_t
                   | _ -> raise (Failure ("Map remove method error")))
              | _ -> raise (Failure ("Sermantic ERROR : Map has no such
method"))
            )
          | Graph ->
            (match fname with
                "bfs" ->
                (match actuals with
                  [x] when (expr x) = Node(Int) -> List(Node(Int))
                | _ -> raise (Failure ("Graph bfs method error")))
              | "dfs" ->
                (match actuals with
                  [x] when (expr x) = Node(Int) -> List(Node(Int))
                  | _ -> raise (Failure ("Graph dfs method error")))
              | "iterGraph" ->
                (match actuals with
                  [x] when (expr x) = Int -> Node(Int)
                  | _ -> raise (Failure ("Graph iterGraph method error")))
              | "findGraphNode" ->
                (match actuals with
                  [x] when (expr x) = String -> Node(Int)
                  | _ -> raise (Failure ("Graph findGraphNode method
error")))
              | "init" ->
                if actuals = [] then Void else raise (Failure ("Graph init
method error"))
              | "addNode" ->
                (match actuals with
                  [x] when Node(Int) = (expr x) -> Void
                  | _ -> raise (Failure ("Graph add node method error")))
              | "addEdge" ->
                (match actuals with
                  [x1;x2;x3] when (expr x1) = Node(Int) &&
                                  (expr x2) = Node(Int) &&
                                  (expr x3) = Int -> Void
                  | _ -> raise (Failure ("Graph add edge method error")))
```

```
              | "combine" ->
                    (match actuals with
                      [x] when (expr x) = Graph -> Graph
                    | _ -> raise (Failure ("Graph combine method error")))
              | "relax" ->
                    (match actuals with
                      [x] when (expr x) = Node(Int) -> Void
                    | _ -> raise (Failure ("Graph relax method error")))
              | "expand" ->
                    (match actuals with
                      [x] when (expr x) = Node(Int) -> Void
                    | _ -> raise (Failure ("Graph expand method error")))
              | "length" ->
                    if actuals = [] then Int else raise (Failure ("Graph
length method error"))
              | "printGraph" ->
                    if actuals = [] then Null_t else raise (Failure ("Graph
print_graph method error"))
              | _ -> raise (Failure ("Graph has no such method"))
            )
          | _ -> raise (Failure ("unsupported type for method call")))
    in

    let check_bool_expr e = if expr e != Bool
     then raise (Failure ("expected Boolean expression in " ^ string_of_expr
e))
     else () in

    (* Verify a statement or throw an exception *)
    let rec stmt = function
            Block sl -> let rec check_block = function
          [Return _ as s] -> stmt s
        | Return _ :: _ -> raise (Failure "nothing may follow a return")
        | Block sl :: ss -> check_block (sl @ ss)
        | s :: ss -> stmt s ; check_block ss
        | [] -> ()
       in check_block sl
      | Expr e -> ignore (expr e)
      | Return e -> let t = expr e in if t = func.typ then () else
       raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                       string_of_typ func.typ ^ " in " ^ string_of_expr e))

      | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
      | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                               ignore (expr e3); stmt st
      | While(p, s) -> check_bool_expr p; stmt s
    in

    stmt (Block func.body)

  in
  List.iter check_function functions
```

8.7. tusimple.ml

```
(* Top-level of the TuSimple compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);    (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check
*)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

8.8. compile.sh

```
#!/bin/sh
TUSIMPLE="./tusimple.native"
basename=`echo $1 | sed 's/.*\\///
                          s/.tu//'`
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

Run "$TUSIMPLE" "<" $1 ">" "${basename}.ll" &&
Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
Run "$CC" "-o" "${basename}.exe" "${basename}.s" "./Lib/utils.o" "printbig.o
" &&
Run "./${basename}.exe" > "${basename}.out"
```

8.9. tests/

8.9.1. fail-add1.tu

```
int main()
{
        int i;
        bool b;
        i = 20;
        b = true;
        print (i + b);
        return 0;
}
```

8.9.2. fail-add1.err

```
Fatal error: exception Failure("illegal binary operator int + bool in i + b")
```

8.9.3. fail-fun1.tu

```
int add(int x, int y) {
        int i;
        i = x + y;
        return i;
}

int main()
{
        int i;
        bool b;
        i = 20;
        b = true;
        add(i, b);
        return 0;
}
```

8.9.4. fail-fun1.err

```
Fatal error: exception Failure("illegal actual argument found bool expected
int in b")
```

8.9.5. fail-fun2.tu

```
int add(int x, int y) {
        int i;
        i = x + y;
        return i;
}

int main()
{
        int i;
        bool b;
        i = 20;
```

```
        b = true;
        add(3);
        return 0;
}
```

8.9.6. fail-fun2.err

```
Fatal error: exception Failure("expecting 2 arguments in add(3)")
```

8.9.7. fail-fun3.tu

```
int foo(int x, int y) {
        int i;
        i = x + y;
        return i;
}
bool bar(bool x) {
        return x;
}

int main()
{
        int i;
        bool b;
        i = 20;
        b = true;
        foo(3, bar(true));
        return 0;
}
```

8.9.8. fail-fun3.err

```
Fatal error: exception Failure("illegal actual argument found bool expected
int in bar(true)")
```

8.9.9. fail-fun4.tu

```
int foo(int x, int y) {
        int i;
        i = x + y;
        return i;
}
int bar(bool x) {
        return x;
}

int main()
{
        int i;
        bool b;
        i = 20;
```

```
        b = true;
        foo(3, bar(true));
        return 0;
}
```

8.9.10. fail-fun4.err

```
Fatal error: exception Failure("return gives bool expected int in x")
```

8.9.11. fail-list1.tu

```
int main()
{
        list @{int} l;
        new l;
        l = @{0.1, 3,4};

        return 0;
 }
```

8.9.12. fail-list1.err

```
Fatal error: exception Failure("illegal list definition {0.1, 3, 4}. Type
Must be homogeneous in a list")
```

8.9.13. fail-list2.tu

```
int main()
{
        list @{bool} l;
        new l;
        l = @{0.1, 3,4};

        return 0;
 }
```

8.9.14. fail-list2.err

```
Fatal error: exception Failure("illegal list definition {0.1, 3, 4}. Type
Must be homogeneous in a list")
```

8.9.15. fail-list3.tu

```
int main()
{
        list @{float} l;
        list @{int} l1;
        new l;
        new l1;
        l = @{0.4,-0.5};
        l1 = @{3,4,5};
```

```
        l += l1;

        return 0;
 }
```

8.9.16. fail-list3.err

```
Fatal error: exception Failure("illegal add assignment list@{float} +=
list@{int} in l += l1")
```

8.9.17. fail-list4.tu

```
int main()
{
        list @{string} l;
        list @{float} l1;

        l = @{"abc", "def", "ghi"};
        l1 = @{9.234,2.53,6.32};
        l.cancat(l1);
        print(l.length());
        l.pop();
        print(l.length());
        l.remove(3);
        print(l.length());

        return 0;
}
```

8.9.18. fail-list4.err

```
Fatal error: exception Failure("List cancatenate method error")
```

8.9.19. fail-map1.tu

```
int main()
{
        map @{string, int} m;
        new m;

        m.put("hello", 0.1);
        m.put("hello", 2);
        m.put("hi", 4);
        m.put("hello", 3);


        print(m.get("hello"));

        print(m.size());

        if (m.haskey("hello")) prints("true");
        else prints("false");
```

```
        m.remove("hello");

        if (m.haskey("hello")) prints("true");
        else prints("false");

        print(m.size());


        return 0;
}
```

8.9.20. fail-map1.err

```
Fatal error: exception Failure("Map put method error")
```

8.9.21. fail-map2.tu

```
int main()
{
        map @{string, int} m;
        new m;

        m.put("hello", 1);
        m.put("hello", 2);
        m.put("hi", 4);
        m.put("hello", 3);


        print(m.get(1));

        print(m.size());

        if (m.haskey("hello")) prints("true");
        else prints("false");

        m.remove("hello");

        if (m.haskey("hello")) prints("true");
        else prints("false");

        print(m.size());


        return 0;
}
```

8.9.22. fail-map2.err

```
Fatal error: exception Failure("Map get method error")
```

8.9.23. fail-node1.tu

```
int main()
{
        node @{int} n1;
        node @{int} n2;

        new n1;
        new n2;
        prints(n1.value());
        prints(n2.value());

        n1.setvalue("abc");
        n2.setvalue("def");

        prints(n1.value());
        prints(n2.value());

        prints(n1.name());
        prints(n2.name());

        return 0;
}
```

8.9.24. fail-node1.err

```
Fatal error: exception Failure("illegal actual argument found int expected
string in n1.value()")
```

8.9.25. fail-node2.tu

```
int main()
{
        node @{int} n1;
        node @{int} n2;

        new n1;
        new n2;
        prints(n1.value());
        prints(n2.value());

        n1.setvalue("abc");
        n2.setvalue("def");

        prints(n1.value());
        prints(n2.value());

        prints(n1.name());
        prints(n2.name());

        return 0;
}
```

8.9.26. fail-node2.err

```
Fatal error: exception Failure("expected Boolean expression in n1.value()")
```

8.9.27. fail-op-and1.tu

```
void foo(int x, float y) {
        if (x && y) prints("true");
        else prints("false");
}

int main()
{
        int a;
        float b;
        a = 1;
        b = 1.2;
        foo(a, b);

        return 0;
 }
```

8.9.28. fail-op-and1.err

```
Fatal error: exception Failure("illegal binary operator int && float in x &&
y")
```

8.9.29. fail-op-and2.tu

```
void foo(int x, bool y) {
        if (x && y) prints("true");
        else prints("false");
}

int main()
{
        int a;
        bool b;
        a = 1;
        b = true;
        foo(a, b);

        return 0;
 }
```

8.9.30. fail-op-and2.err

```
Fatal error: exception Failure("illegal binary operator int && bool in x &&
y")
```

8.9.31. fail-op-not1.tu

```
void foo(int x) {
        if (!x ) prints("true");
```

```
        else prints("false");
}

int main()
{
        int a;
        bool b;
        a = 1;
        b = true;
        foo(a);

        return 0;
 }
```

8.9.32. fail-op-not1.err

```
Fatal error: exception Failure("illegal unary operator !int in !x")
```

8.9.33. fail-op-not2.tu

```
void foo(float x) {
        if (!x ) prints("true");
        else prints("false");
}

int main()
{
        float a;
        bool b;
        a = 1.2;
        b = true;
        foo(a);

        return 0;
 }
```

8.9.34. fail-op-not2.err

```
Fatal error: exception Failure("illegal unary operator !float in !x")
```

8.9.35. fail-op-or1.tu

```
void foo(int x, bool y) {
        if (x || y) prints("true");
        else prints("false");
}

int main()
{
        int a;
        bool b;
        a = 1;
```

```
        b = true;
        foo(a, b);

        return 0;
 }
```

8.9.36. fail-op-or1.err

```
Fatal error: exception Failure("illegal binary operator int || bool in x ||
y")
```

8.9.37. fail-op-or2.tu

```
void foo(int x, float y) {
        if (x || y) prints("true");
        else prints("false");
}

int main()
{
        int a;
        float b;
        a = 1;
        b = 1.2;
        foo(a, b);

        return 0;
 }
```

8.9.38. fail-op-or2.err

```
Fatal error: exception Failure("illegal binary operator int || float in x ||
y")
```

8.9.39. fail-set1.tu

```
int main() {
        set @{int} s;
        new s;
        s.put(true);

        return 0;
}
```

8.9.40. fail-set1.err

```
Fatal error: exception Failure("Set put method error")
```

8.9.41. fail-set2.tu

```
int main() {
        set @{int} s;
        new s;
```

```
        s.put(true);

        return 0;
}
```

8.9.42. fail-set2.err

```
Fatal error: exception Failure("Set put method error")
```

8.9.43. fail-set3.tu

```
int main() {
        set @{bool} s;
        new s;
        s.put(1);

        return 0;
}
```

8.9.44. fail-set3.err

```
Fatal error: exception Failure("Set put method error")
```

8.9.45. fail-set4.tu

```
int main() {
        set @{node@{int}} s;
        new s;
        s.put(0.1);

        return 0;
}
```

8.9.46. fail-set4.err

```
Fatal error: exception Failure("Set put method error")
```

8.9.47. fail-set5.tu

```
int main() {
        set @{node@{int}} s;
        node @{int} n1;
        node @{int} n2;
        node @{int} n3;
        node @{int} n4;
        list @{node@{float}} l;

        new s;
        new n1;
        new n2;
        new n3;
        new n4;
```

```
        new l;

        n1.setvalue(11);
        n2.setvalue(22);
        n3.setvalue(33);
        n4.setvalue(0.44);

        s.put(n1);
        print (s.length());
        l = @{n4};
        s+=l;
        if (s.contain(n1)) prints("true");
        else prints("false");
        if (s.contain(n4)) prints("true");
        else prints("false");

        return 0;
}
```

8.9.48. fail-set5.err

```
Fatal error: exception Failure("illegal assignment list@{node@{float}} =
list@{node@{int}} in l = @{n4}")
```

8.9.49. fail-test.tu

```
int main() {
        int a;
        a = 0.2;
        return 0;
}
```

8.9.50. fail-test.err

```
Fatal error: exception Failure("illegal assignment int = float in a = 0.2")
```

8.9.51. fail-undeclaredfun.tu

```
int main()
{
        int i;
        i = 20;
        add(i);
        return 0;
}
```

8.9.52. fail-undeclaredfun.err

```
Fatal error: exception Failure("unrecognized function add")
```

8.9.53. test-arith1-add.tu

```
int main() {
```

```
        int a;
        int b;
        int c;
        a = 5;
        b = 2;
        c = a + b;
        print(c);
        return 0;
}
```

8.9.54. test-arith1-add.out

```
7
```

8.9.55. test-arith2-sub.tu

```
int main() {
        int a;
        int b;
        int c;
        a = 1;
        b = 2;
        c = a - b;
        print(c);
        return 0;
}
```

8.9.56. test-arith2-sub.out

```
-1
```

8.9.57. test-arith3-mult.tu

```
int main() {
        int a;
        int b;
        int c;
        a = 9;
        b = 2;
        c = a * b;
        print(c);
        return 0;
}
```

8.9.58. test-arith3-mult.out

```
18
```

8.9.59. test-arith4-division.tu

```
int main() {
        int a;
```

```
        int b;
        int c;
        a = 10;
        b = 2;
        c = a / b;
        print(c);
        return 0;
}
```

8.9.60. test-arith4-division.out

```
5
```

8.9.61. test-bfs1.tu

```
int main(){
        int a, b, i;
        bool c, d;
        int size;
        node@{int} node1, node2, node3, node4, node5, node6, node7, node8,
node9, node10;
        list@{node@{int}} l, l2, rec;
        set@{node@{int}} visited;
        graph g;
        new node1;
        new node2;
        new node3;
        new node4;
        new node5;
        new node6;
        new node7;
        new node8;
        new node9;
        new node10;
        new l;
        new l2;
        new rec;
        new visited;
        new g;
        node1 -> node3 = 14;
        node1 -> node2 = 11;

        node2 -> node4 = 12;
        node3 -> node4 = 15;
        g.addNode(node1);
        g.addNode(node2);
        g.addNode(node3);
        g.addNode(node4);
        l = g.bfs(node1);

        node5 = l.get(0);
        prints(node5.name());
```

```
        node6 = l.get(1);
        prints(node6.name());

        node7 = l.get(2);
        prints(node7.name());

        node8 = l.get(3);
        prints(node8.name());


}
```

8.9.62. test-bfs1.out

```
node1
node3
node2
node4
```

8.9.63. test-bfs2.tu

```
int main(){
 int a, b, i;
 bool c, d;
 int size;
 node@{int} node1, node2, node3, node4, node5, node6, node7, node8, node9,
node10;
 list@{node@{int}} l, l2, rec;
 set@{node@{int}} visited;
 graph g;
 new node1;
 new node2;
 new node3;
 new node4;
 new node5;
 new node6;
 new node7;
 new node8;
 new node9;
 new node10;
 new l;
 new l2;
 new rec;
 new visited;
 new g;
 node1 -> node3 = 14;
 node1 -> node2 = 11;

 node2 -> node4 = 12;
 node3 -> node4 = 15;
 g.addNode(node1);
 g.addNode(node2);
 g.addNode(node3);
```

```
  g.addNode(node4);


  l2 += @{node1};
  rec += @{node1};
  visited.put(node1);

  while (l2.length()!=0){
    node9 = l2.get(0);
    size = node9.length();

    print(size);
    for (i=0;i<size;){
      node10 = node9.iterNode(i);
      if (!visited.contain(node10)){
          prints("EXECUTED");
          l2 += @{node10};
          rec += @{node10};
          visited.put(node10);
      }
      i+=1;
    }
    l2.remove(0);
  }

  print(rec.length());

  node5 = rec.get(0);
  prints(node5.name());

  node6 = rec.get(1);
  prints(node6.name());

  node7 = rec.get(2);
  prints(node7.name());

  node8 = rec.get(3);
  prints(node8.name());

}
```

8.9.64. test-bfs2.out

```
2
EXECUTED
EXECUTED
1
EXECUTED
1
0
4
node1
node3
```

```
node2
node4
```

8.9.65. test-bool.tu

```
int main()
{
  bool i;
  i = true;
  if (!i) print(1);
  else print (0);
  return 0;
}
```

8.9.66. test-bool.out

```
0
```

8.9.67. test-data1.tu

```
int main() {
        set @{node@{int}} s;
        node @{int} n1;
        node @{int} n2;
        node @{int} n3;
        node @{int} n4;
        list @{node@{int}} l;

        new s;
        new n1;
        new n2;
        new n3;
        new n4;
        new l;

        n1.setvalue(11);
        n2.setvalue(22);
        n3.setvalue(33);
        n4.setvalue(44);

        s.put(n1);
        print (s.length());
        l = @{n2,n3};
        s+=l;
        if (s.contain(n1)) prints("true");
        else prints("false");
        if (s.contain(n4)) prints("true");
        else prints("false");

        return 0;
}
```

8.9.68. test-data1.out

```
1
true
false
```

8.9.69. test-dfs2.tu

```
int main(){
        int a;
        int b;
         int i;
        bool c;
        bool d;
        int size;
        int now;
        node@{int}
node1,node2,node3,node4,node5,node6,node7,node8,node9,node10;
        list@{node@{int}} l, l2, rec;
        set@{node@{int}} visited;
        map@{string,int} hash;
        graph g;

        new node1;
        new node2;
        new node3;
        new node4;
        new node5;
        new node6;
        new node7;
        new node8;
        new node9;
        new node10;
        new l;
        new l2;
        new rec;
        new visited;
        new hash;
        new g;

        node1 -> node3 = 14;
        node1 -> node2 = 11;
        node2 -> node4 = 12;
        node3 -> node4 = 15;


         node1.setvalue(1);
        node2.setvalue(2);
        node3.setvalue(3);
        node4.setvalue(4);


        g.addNode(node1);
```

```
        g.addNode(node3);
        g.addNode(node2);
        g.addNode(node4);


        l2 += @{node1};
        rec += @{node1};
        hash.put(node1.name(), 0);

        while (l2.length()!=0){
                node9 = l2.get(l2.length()-1);
                size = node9.length();
                now = hash.get(node9.name());
                if (now<size){
                        node10 = node9.iterNode(now);
                        if (!hash.haskey(node10.name())){
                                l2 += @{node10};
                                rec += @{node10};
                                hash.put(node10.name(),0);
                        }
                        hash.put(node9.name(), now+1);
                } else {
                        l2.remove(l2.length()-1);
                }
        }

        print(rec.length());

        node5 = rec.get(0);
        prints(node5.name());
        print(node5.value());

        node6 = rec.get(1);
        prints(node6.name());
        print(node6.value());

        node7 = rec.get(2);
        prints(node7.name());
        print(node7.value());

        node8 = rec.get(3);
        prints(node8.name());
        print(node8.value());

}
```

8.9.70. test-dfs2.out

```
4
node1
1
node3
3
```

```
node4
4
node2
2
```

8.9.71. test-Dijkstra1.tu

```
int main(){

    int max;
    int i;
    int size;
    int tmp;
    int now;
    string name;
    node@{int} node1;
    node@{int} node2;
    node@{int} node3;
    node@{int} node4;
    node@{int} node5;
    node@{int} node6;
    node@{int} node7;
    node@{int} node8;
    node@{int} node9;
    node@{int} node10;

    list@{node@{int}} l;
    list@{node@{int}} queue;
    set@{string} inList;
    map@{string, int} dis;
    graph g;

    new node1;
    new node2;
    new node3;
    new node4;
    new node5;
    new node6;
    new node7;
    new node8;
    new node9;
    new node10;

    new l;
    new queue;
    new inList;
    new dis;
    new g;

    node1 -> node2 = 11;
    node2 -> node4 = 11;
    node1 -> node3 = 3;
    node3 -> node4 = 3;
```

```
max = 10000;

queue+=@{node1};
inList.put(node1.name());
// print(inList.length());
dis.put(node1.name(), 0);


/*
l+=@{node2}; remain+=@{node2}; hash.put(node2.name(), 0);
l+=@{node3}; remain+=@{node3}; hash.put(node3.name(), 0);
l+=@{node4}; remain+=@{node4}; hash.put(node4.name(), 0);
l+=@{node5}; remain+=@{node5}; hash.put(node5.name(), 0);
l+=@{node6}; remain+=@{node6}; hash.put(node6.name(), 0);
l+=@{node7}; remain+=@{node7}; hash.put(node7.name(), 0);
l+=@{node8}; remain+=@{node8}; hash.put(node8.name(), 0);
l+=@{node9}; remain+=@{node9}; hash.put(node9.name(), 0);
l+=@{node10};remain+=@{node10};hash.put(node10.name(),0);
*/


while (queue.length()!=0){
  node9 = queue.get(0);
  // prints(node9.name());
  size = node9.length();
  now = dis.get(node9.name());
  for (i=0;i<size;i+=1){
    // print(i);
    node10 = node9.iterNode(i);
    // print(now);
    tmp = now + node9.weightIter(i);
    // print(node9.weightIter(i));
    if (!dis.haskey(node10.name())){
      dis.put(node10.name(), max);
    }
    if (tmp<dis.get(node10.name())){

      dis.put(node10.name(), tmp);
      if (!inList.contain(node10.name())){
        queue += @{node10};
        inList.put(node10.name());
        // print(inList.length());
      }
    }
    // i+=1;
  }
  // print(inList.length());
  inList.remove(node9.name());
  queue.remove(0);
  // print(inList.length());
}
```

```
    now = dis.get(node1.name());
    print(now);
    print(dis.get(node2.name()));
    print(dis.get(node3.name()));
    print(dis.get(node4.name()));

}
```

8.9.72. test-Dijkstra1.out

```
0
11
3
6
```

8.9.73. test-for1.tu

```
int main()
{
        int i;
        for (i = 0; i < 8; i = i+1) {
                print(i);
        }
        return 0;
}
```

8.9.74. test-for1.out

```
0
1
2
3
4
5
6
7
```

8.9.75. test-for2.tu

```
int main()
{
        int i;
        i = -4;
        for (; i < 8; ) {
                print(i);
                i = i+1;
        }
        return 0;
}
```

8.9.76. test-for2.out

```
-4
-3
-2
-1
0
1
2
3
4
5
6
7
```

8.9.77. test-for3.tu

```
int main()
{
        int i;
        i = 4;
        for (; i > 1; i = i-1) {
                print(i);
        }
        return 0;
}
```

8.9.78. test-for3.out

```
4
3
2
```

8.9.79. test-fun1.tu

```
int foo(int x, bool y) {
        if (y) {
                return x;
        }
        return 0;
}
bool bar(bool x) {
        return x;
}

int main()
{
        int i;
        bool b;
        i = 20;
        b = true;
        print(foo(5, bar(true)));
        return 0;
}
```

8.9.80. test-fun1.out

```
5
```

8.9.81. test-fun2.tu

```
int foo(list@{int} l) {
        return l[0];
}
bool bar(bool x) {
        return x;
}

int main()
{
        int i;
        bool b;
        list @{int} l;
        i = 20;
        b = true;
        l = @{1,2,3};
        print(foo(l));
        return 0;
}
```

8.9.82. test-fun2.out

```
1
```

8.9.83. test-graph1.tu

```
int main() {
        node @{int} n1, n2, n3,n4, n5,n6, n7, n8;
        graph g;
        int i;

        new n1;
        new n2;
        new n3;
        new n4;
        new n5;
        new n6;
        new n7;
        new n8;
        new g;

        n1.setvalue(1);
        n2.setvalue(2);
        n3.setvalue(3);
        n4.setvalue(4);
        n5.setvalue(5);
```

```
        n6.setvalue(6);
        n7.setvalue(7);

        g.addNode(n1);
        g.addNode(n2);
        g.addEdge(n1, n2, 33);

/*      n8 = g.IterGraph(0);
        prints(n8.name());
        print(n8.value());
*/
/*
        n1->@{n2, n3, n4} = @{22, 33, 44};
        n5--@{n1, n6, n7} = @{22,66, 77};
        print(n1.length());


        for (i = 0; i < n5.length(); i+=1) {
                n8 = n5.iterNode(i);
                prints(n8.name());
                print(n8.value());
        }
*/

        return 0;
}
```

8.9.84. test-graph1.out

8.9.85. test-graph2.tu

```
int main() {
        node @{int} n1, n2, n3,n4, n5,n6, n7, n8;
        graph g;
        int i;

        new n1;
        new n2;
        new n3;
        new n4;
        new n5;
        new n6;
        new n7;
        new n8;
        new g;

        n1.setvalue(1);
        n2.setvalue(2);
        n3.setvalue(3);
        n4.setvalue(4);
```

```
        n5.setvalue(5);
        n6.setvalue(6);
        n7.setvalue(7);

//      g.addNode(n1);
//      g.addNode(n2);
        g.addEdge(n1, n2, 33);
        g.addEdge(n3, n4, 44);
        g.addNode(n3);
        g.addNode(n4);

        for (i = 0; i < 6; i += 1) {
                n8 = g.iterGraph(i);
                prints(n8.name());
                print(n8.value());
        }

/*
        n1->@{n2, n3, n4} = @{22, 33, 44};
        n5--@{n1, n6, n7} = @{22,66, 77};
        print(n1.length());


        for (i = 0; i < n5.length(); i+=1) {
                n8 = n5.iterNode(i);
                prints(n8.name());
                print(n8.value());
        }

*/
        return 0;
}
```

8.9.86. test-graph2.out

```
n1
1
n2
2
n3
3
n4
4
n3
3
n4
4
```

8.9.87. test-graph3.tu

```
int main(){
  int a, b, i;
  bool c, d;
```

```
  int size;
        node@{int} node1, node2, node3, node4, node5, node6, node7, node8,
node9, node10;
  list@{node@{int}} l, l2, rec;
  set@{node@{int}} visited;
  graph g, g1;

  new node1;
  new node2;
  new node3;
  new node4;
  new node5;
  new node6;
  new node7;
  new node8;
  new node9;
  new node10;
  new l;
  new l2;
  new rec;
  new visited;
  new g;
  new g1;


  node1 -> node2 = 11;
  node1 -> node3 = 14;
  node2 -> node4 = 12;
  node3 -> node4 = 15;
  g.addNode(node1);
  g.addNode(node2);
  g.addNode(node3);
  g.addNode(node4);

  for (i = 0; i < g.length(); i+=1) {
    node7 = g.iterGraph(i);
    prints(node7.name());
  }

  g1.addEdge(node5, node6, 12);
  g.combine(g1);
  for (i = 0; i < g.length(); i+=1) {
    node7 = g.iterGraph(i);
    prints(node7.name());
  }

}
```

8.9.88. test-graph3.out

```
node1
node2
node3
```

```
node4
node1
node2
node3
node4
node5
node6
```

8.9.89. test-if1.tu

```
int main() {
        int a;
        a = 3;
        if (a < 0) {
                print(1);
        }
        else
                print(0);
        return 0;
}
```

8.9.90. test-if1.out

```
0
```

8.9.91. test-if2.tu

```
int main() {
        int a;
        int b;
        a = 3;
        b = 4;
        if (a < b) {
                print(1);
        }
        else if (a == b)
                print(0);
        else
                print (-1);
        return 0;
}
```

8.9.92. test-if2.tu

```
1
```

8.9.93. test-int.tu

```
int main(){
  int b;
```

```
  b = 10;
  print (b);
}
```

8.9.94. test-int.out

```
10
```

8.9.95. test-list1.tu

```
int main()
{
        list @{int} l;
        l = @{1,2,3};
        print(l[1]);
        return 0;
}
```

8.9.96. test-list1.out

```
2
```

8.9.97. test-list2.tu

```
int main()
{
        int i;
        list @{int} l;
        list @{int} l1;
        new l1;
        l = @{11,22,33};
        l += @{44,55,66};
        print (l.length());
        print (l[5]);
        for (i = 0; i < l.length(); i+=1) {
                print(l.get(i));
        }
        l.pop();
        l.pop();
        print(l.length());


        for (i = 0; i < l.length(); i+=1) {
                print(l.get(i));
        }

        return 0;
 }
```

8.9.98. test-list2.out

```
6
66
```

```
11
22
33
44
55
66
4
11
22
33
44
```

8.9.99. test-list3.tu

```
int main() {
        int i;
        list @{int} l;
        list @{int} l1;
        new l1;
        new l;
        print (l.length());
        l.cancat(l1);
        print (l.length());
        for (i = 0; i < l.length(); i+=1) {
                print(l.get(i));
        }
        l.cancat(@{11,22,33});
        l.remove(1);

        for (i = 0; i < l.length(); i+=1) {
                print(l.get(i));
        }

        l.pop();

        for (i = 0; i < l.length(); i+=1) {
                print(l.get(i));
        }

}
```

8.9.100. test-list3.out

```
0
0
11
33
11
```

8.9.101. test-list4.tu

```
int main()
{
        list @{float} l;
        list @{float} l1;

        l = @{0.23, 13.2};
        l1 = @{9.234,2.53,6.32};
        l.cancat(l1);
        print(l.length());
        l.pop();
        print(l.length());
        l.remove(3);
        print(l.length());

        return 0;
}
```

8.9.102. test-list4.out

```
5
4
3
```

8.9.103. test-list5.tu

```
int main()
{
        int i;
        list @{node@{int}} l;
        list @{node@{int}} l1;
        node@{int} n1;
        node@{int} n2;
        node@{int} n3;
        node@{int} n4;
        node@{int} n5;
        node@{int} n6;

        new l1;
        new l;
        new n1;
        new n2;
        new n3;
        new n4;
        new n5;
        new n6;

        n1.setvalue(1);
        n2.setvalue(2);
        n3.setvalue(3);
        n4.setvalue(4);
        n5.setvalue(5);

        l = @{n1,n2,n3};
```

```
        l += @{n4,n5};

        n6 = n1;
        print(n6.value());

        for (i = 0; i < l.length(); i+=1) {
                n6 = l[i];
                print(n6.value());
        }
        l.pop();
        l.pop();
        print(l.length());


        for (i = 0; i < l.length(); i+=1) {
                n6 = l[i];
                print(n6.value());
        }

        return 0;
 }
```

8.9.104. test-list5.out

```
1
1
2
3
4
5
3
1
2
3
```

8.9.105. test-map1.tu

```
int main()
{
        map @{int, string} m;
        new m;
        m.put(1,"lalalal");

        if(m.haskey(1)) print(1);
        else print(0);
        print(m.size());



        m.remove(1);
        if(m.haskey(1)) print(1);
        else print(0);
```

```
        return 0;
}
```

8.9.106. test-map1.out

```
1
1
0
```

8.9.107. test-map2.tu

```
int main()
{
    map @{string, float} m;
    new m;
    m.put("a", 0.23);
    m.put("b", 4.2);
    m.put("c", 1.45);
    m.put("d", 546.3);
    m.get("c");
    if(m.haskey("e")) print(1);
    else print(0);
    if(m.haskey("a")) print(1);
    else print(0);
    print(m.size());
    return 0;
}
```

8.9.108. test-map2.out

```
0
1
4
```

8.9.109. test-map3.tu

```
int main()
{
    map @{string, int} m;
    new m;
    m.put("a", 3434);
    m.put("b", 112);
    m.put("c", 65);
    m.put("d", 192);
    print(m.get("c"));

    print(m.size());

    return 0;
}
```

8.9.110. test-map3.out

```
65
4
```

8.9.111. test-map4.tu

```
int main()
{
          map @{string, int} m;
        new m;

        m.put("hello", 1);
    m.put("hello", 2);
    m.put("hi", 4);
    m.put("hello", 3);


        print(m.get("hello"));

        print(m.size());

        if (m.haskey("hello")) prints("true");
        else prints("false");

        m.remove("hello");

        if (m.haskey("hello")) prints("true");
        else prints("false");

        print(m.size());


          return 0;
}
```

8.9.112. test-map4.out

```
3
2
true
false
1
```

8.9.113. test-node1.tu

```
int main()
{

        node @{int} n1;
        node @{int} n2;
```

```
        new n1;
        new n2;

        print (n1.value());
        prints(n1.name());

        n1.setvalue(3);
        n2.setvalue(1);
        print (n1.value());
        prints(n1.name());
        print (n2.value());
        prints(n2.name());

        return 0;
}
```

8.9.114. test-node1.out

```
0
n1
3
n1
1
n2
```

8.9.115. test-node2.tu

```
int main()
{
        node @{float} n1;
        node @{float} n2;

        new n1;
        new n2;

        n1.setvalue(3.1);
        n2.setvalue(1.2);

        prints(n1.name());
        prints(n2.name());

        return 0;
}
```

8.9.116. test-node2.out

```
n1
n2
```

8.9.117. test-node3.tu

```
int main()
{
        node @{bool} n1;
        node @{bool} n2;

        new n1;
        new n2;
        if(n1.value()) print(1);
        else print(0);
        if(n2.value()) print(1);
        else print(0);


        prints(n1.name());
        prints(n2.name());

        return 0;
}
```

8.9.118. test-node3.out

```
0
0
n1
n2
```

8.9.119. test-node4.tu

```
int main()
{
        node @{string} n1;
        node @{string} n2;

        new n1;
        new n2;
        prints(n1.value());
        prints(n2.value());

        n1.setvalue("abc");
        n2.setvalue("def");

        prints(n1.value());
        prints(n2.value());

        prints(n1.name());
        prints(n2.name());

        return 0;
}
```

8.9.120. test-node4.out

```
abc
def
n1
n2
```

8.9.121. test-node5.tu

```
int main() {
        node @{int} n1, n2, n3,n4, n5,n6, n7, n8;
        graph g;
        int i;

        new n1;
        new n2;
        new n3;
        new n4;
        new n5;
        new n6;
        new n7;
        new n8;
        new g;

        n1.setvalue(1);
        n2.setvalue(2);
        n3.setvalue(3);
        n4.setvalue(4);
        n5.setvalue(5);
        n6.setvalue(6);
        n7.setvalue(7);


        n1->@{n2, n3, n4} = @{22, 33, 44};
        n5--@{n1, n6, n7} = @{22,66, 77};
        print(n1.length());

        for (i = 0; i < n5.length(); i+=1) {
                n8 = n5.iterNode(i);
                prints(n8.name());
                print(n8.value());
        }


        return 0;
}
```

8.9.122. test-node5.out

```
4
n1
1
n6
6
n7
```

```
7
```

8.9.123. test-node6.tu

```
int main() {
        node @{int} n1, n2, n3,n4, n5,n6, n7, n8;
        graph g;
        int i;

        new n1;
        new n2;
        new n3;
        new n4;
        new n5;
        new n6;
        new n7;
        new n8;
        new g;

        n1.setvalue(1);
        n2.setvalue(2);
        n3.setvalue(3);
        n4.setvalue(4);
        n5.setvalue(5);
        n6.setvalue(6);
        n7.setvalue(7);


        n1->@{n2, n3, n4} = @{22, 33, 44};
        n5--@{n1, n6, n7} = @{22,66, 77};
        print(n1.length());

        for (i = 0; i < n1.length(); i+=1) {
                n8 = n1.iterNode(i);
                prints(n8.name());
                print(n8.value());
        }


        return 0;
}
```

8.9.124. test-node6.out

```
4
n2
2
n3
3
n4
4
n5
5
```

8.9.125. test-ops1.tu

```
int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  if (1 == 2) print(1);
  else print(0);
  if(1 == 1) print(1);
  else print(0);
  print(99);
  if(1 != 2) print(1);
  else print(0);
  if(1 != 1) print(1);
  else print(0);
  print(99);
  if(1 < 2) print(1);
  else print(0);
  if(2 < 1) print(1);
  else print(0);
  print(99);
  if(1 <= 2) print(1);
  else print(0);
  if(1 <= 1) print(1);
  else print(0);
  if(2 <= 1) print(1);
  else print(0);
  print(99);
  if(1 > 2) print(1);
  else print(0);
  if(2 > 1) print(1);
  else print(0);
  print(99);
  if(1 >= 2) print(1);
  else print(0);
  if(1 >= 1) print(1);
  else print(0);
  if(2 >= 1) print(1);
  else print(0);
  return 0;
}
```

8.9.126. test-ops1.out

```
1
0
1
0
0
```

```
0
1
1
1
0
1
0
-10
```

8.9.127. test-ops2.tu

```
int main()
{
  if(true) print(1); else print(0);
  if(false) print(1); else print(0);
  if(true && true) print(1); else print(0);
  if(true && false) print(1); else print(0);
  if(false && true) print(1); else print(0);
  if(false && false) print(1); else print(0);
  if(true || true) print(1); else print(0);
  if(true || false) print(1); else print(0);
  if(false || true) print(1); else print(0);
  if(false || false) print(1); else print(0);
  if(!false) print(1); else print(0);
  if(!true) print(1); else print(0);
  print(-10);
}
```

8.9.128. test-ops2.out

```
1
0
1
0
0
0
1
1
1
0
1
0
-10
```

8.9.129. test-set1.tu

```
int main()
{
        set@{int} s;
        new s;
        s += @{1,2,3};
        s.put(9);
```

```
        print(s.length());
        if (s.contain(2)) prints("true");
        else prints("false");
        if (s.contain(8)) prints("true");
        else prints("false");
        return 0;
}
```

8.9.130. test-set1.out

```
4
true
false
```

8.9.131. test-set2.tu

```
int main()
{
        set@{float} s;
        new s;
        s += @{1.1,2.2,3.3};
        s.put(9.9);
        print(s.length());
        if (s.contain(2.2)) prints("true");
        else prints("false");
        if (s.contain(8.8)) prints("true");
        else prints("false");
        return 0;
}
```

8.9.132. test-set2.out

```
4
true
false
```

8.9.133. test-set3.tu

```
int main()
{
        set@{string} s;
        new s;
        s += @{"asdf" , "sdfsdf"};
        s.put("were");
        print (s.length());

        if (s.contain("asdf")) prints("true");
        else prints("false");
        if (s.contain("c")) prints("true");
        else prints("false");

        return 0;
}
```

8.9.134. test-set3.out

```
3
true
false
```

8.9.135. test-set4.tu

```
int main()
{
        set@{node@{int}} s;
        list@{node@{int}} l;

        node @{int} n1;
        node @{int} n2;

        new s;
        new l;
        new n1;
        new n2;

        n1.setvalue(3);
        n2.setvalue(1);
        l = @{n1,n2};
        s += @{n1, n2};
        print (s.length());
        return 0;
}
```

8.9.136. test-set4.out

```
2
```

8.9.137. test-while1.tu

```
int main() {
        int i;
        i = 5;
        while(i > 0) {
                print(i);
                i = i - 1;
        }
        return 0;
}
```

8.9.138. test-while1.out

```
5
4
3
2
```

```
1
```

8.9.139. test-while2.tu

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  print(foo(7));
  return 0;
}
```

8.9.140. test-while2.out

```
14
```