...a graph language
Introduction
Motivation

- Graphs appear naturally in many disciplines
- Solutions to graph problems can be extremely useful
- Writing your own graph library can be difficult and waste time
- Aiming for a language that can truly utilize and manipulate graphs with ease
Summary

- Mathematical style without the object-oriented fuss
- Native types for list, map, node, and graph built in
- Compiles to LLVM IR for cross-platform functionality
- Topological sort for scheduling and dependency checking
- Search graphs and find shortest path easily
Syntax
## Comments

/*
 * Multiple line comment
 */

## Operators

```
+  -  *  /  %  <  >
==  !=  <=  >=  =  !
&&  ||
```

## Keywords

- if
- else
- while
- main
- return
- int
- bool
- float
- string
- list
- hashmap
- node
- edge
- graph
- null
- void
Conditionals

if ( x == y ) {
    doSomething();
}

if ( i <= j ) {
    doSomething();
} else {
    doSomethingElse();
}

Loops

int i;
int i = 0;
while( i < 3 ) {
    doSomething(i);
    i++;
}

Functions

int add(int a, int b) {
    return a + b;
}

void endl() {
    print("\n");
}

int main(){
    int x, y;
    x = 4;
    y = 12
    print( add( x, y ) );
    return 0;
}
int main(){
    list < int > stuff = [4];
    int i = 0;
    while (i < 10) {
        stuff.add( stuff, i );
    }

    print( stuff );
}

int main(){
    hashmap < int > hash =
    {“cat” : 2, “dog” : 4};
    hash.put(“mouse”, 5);
    print( hash.get( “mouse” ) );
}
Implementation
3 IMPLEMENTATION

Planning

- Define exciting features we hope to implement
- Identify roles and distribute initial workloads
- Meet often to stay on the same page
- Create rough deadlines and push back unnecessary features
- Refine language usage and functionality
3 IMPLEMENTATION

Roles

- Dianya Jiang: Project planning, Test case, Scanner, Parser
- Tongyun Wu: AST, Parser, Code generation, Writing C Libraries
- Vince Pallone: Scanner, Linking C Libraries, Code generation
- Yoki Yuan: Semantics, Parser, Sast, Checker, Writing C Library
Tools

- Slack: Group communication
- GitHub: Public code repo and version control kept us sane
- Ubuntu: Consistent operating system for testing
- VirtualBox: Software for running virtual environment
- LLVM: The only reason we have a language
Architecture
Abstract Syntax Tree

- **program**
  - **functions**
    - **formals**
  - **globals**
    - **locals**
    - **statements**
      - **expressions**
Lessons Learned

- Start early with something small that works
- Use branches and add only one feature at a time
- Start with tests, then make them work
- Aim for a small set of highly focused features
Features
C Library – List and Map

- List
  - Initialize List, Add, Get, Set, Contains, Remove Data, Push, Get Size, Print List
  - *Special: Concat List -> [1,2,3] + [4,5,6] = [1,2,3,4,5,6]
  - Remove Data -> [1,2,3,4,5] - 5 = [1,2,3,4]

- HashMap
  - Put, Get, Contains, Remove, Keys,
C Library - Queue and Minheap

- Queue
  - Initialize, Push Back, Pop Front, get Size, print

- Minheap
  - Initialize, Swap, Compare, Heapify, Insert, Get Min Value, Decrease Priority, Print
4 FEATURES

C Library - Node and Edge

- Node
  - Create Node, Print Node, Get Node Value, * Set Visited, * Get Visited
  - Add Node, Has Node, Remove Node

- Edge
  - Create Edge, Print Edge, Get Edge Weight
  - Add Edge, Contains Edge, Remove Edge
C Library – Graph

- Graph
  - Graph Define: $a \rightarrow 1$, $b \rightarrow 3$, $c + b \rightarrow 2$, $d + c \rightarrow 5$, $e$
  - Graph Manipulation: Link Graph, Split Graph, Copy Graph, Get All Nodes, Set All Node Unvisited

- * Cool Functions
  - Breadth First Search, Depth First Search
  - Dijkstra Algorithm
Node & Edge Functions

FEATURES

```python
node a = node("A");
node b = node("B");
node c = node("C");
node d = node("D");
node e = node("E");
node f = node("F");
node m = node("M");
node h = node("H");
node x = node("X");
node y = node("Y");
node z = node("Z");

print(" A -> E + A -> B -> D + A -> C ->F ");

graph g = a -> 1s + a -> 2s + b -> 3s + d + a -> 4s + c -> 5s + f;
print(g);

print("Get Graph Size: ");
print(g.size());

print("Add Nodes to Graph");
g.addNode(m);
print(g.size());
print(g);

print("Judge whether Nodes exist");
print(g.hasNode(x));
print(g.hasNode(a));

print("Add Edge to Graph, Support new Nodes");
g.addEdge(e, f, 5);
g.addEdge(a, z, 10);
print(g.size());
list<node> nodes = g.getAllNodes();
print(nodes);
print(g);

print("Judge whether Edges exist between two Nodes");
print(g.hasEdge(c, d));
print(g.hasEdge(a, b));
```

```
A -> E + A -> B -> D + A -> C -> F

Nodes:
node A
node E
node B
node D
node C
node F

Edges:
edgeA ->E: 1
dgeB ->D: 3
dgeA ->B: 1
dgeC ->F: 5
dgeA ->C: 4

Get Graph Size:
6
Add Nodes to Graph
7

Nodes:
node A
node E
node B
node D
node C
node F
node M
node Z

Edges:
edgeA ->E: 1
dgeB ->D: 3
dgeA ->B: 2
dgeC ->F: 5
dgeE ->F: 5
dgeA ->Z: 10

Judge whether Nodes exist
false
true
Add Edge to Graph, Support new Nodes
8
[node A
node E
node B
node D
node C
node F
node M
node Z
]

Nodes:
node A
node E
node B
node D
node C
node F
node M
node Z

Edges:
edgeA ->E: 1
dgeB ->D: 3
dgeA ->B: 2
dgeC ->F: 5
dgeE ->F: 5
dgeA ->Z: 10

Judge whether Edges exist between two Nodes
false
true
node a = node("A");
node b = node("B");
node c = node("C");
node d = node("D");
node e = node("E");

print("Using + to link graph");

print("Link graphs: A -> 0$B -> 3$E + C -> 1$A + A -> 2$D");
print("Shared nodes: A");

graph g = a -> 0$b -> 3$e + c -> 1$a + a -> 2$d;

print(g);

using + to link graph
Link graphs: A -> 0$B -> 3$E + C -> 1$A + A -> 2$D
Shared nodes: A

Nodes:
node B
node E
node A
node C
node D

Edges:
edgeB -> E: 3
edgeA -> B: 0
edgeC -> A: 1
edgeA -> D: 2
Topological Sort

```java
node a = node("COMS 1004: Introduction to Computer Science and Programming in Java");
node b = node("COMS 4111: Introduction to Databases");
node c = node("COMS 6111: Advanced Database Systems");
node d = node("CSEE 4119: Computer Networks");
node e = node("CSEE 4140: Networking Laboratory");
node f = node("COMS 4118: Operating Systems");
node g = node("COMS 3101: Programming Language Python");
node h = node("CSOR 4231: Analysis of Algorithms I");
node i1 = node("COMS 6232: Analysis of Algorithms II");
node j = node("COMS 4771: Machine Learning");
node k = node("COMS 4772: Advanced Machine Learning");
node l = node("COMS 4995: Deep Learning for Computer Vision");

graph ga = a -> b -> f + b -> c + a -> d -> f + d -> e + h -> f + g -> h -> j -> l + h -> i1 + j -> k;

int i=0;
list<node> n = ga.getAllNodes();
list<node> tmp = n;
list<node> res = n;
tmp = B;
ga.getAllUnvisited();

while(i=ga.size()){
  if(!n.get(i).isVisited()){
    i = i + 1;
  }else{
    tmp = ga.dfs(n.get(i));
    tmp = tmp.size();
    while(!tmp.isEmpty()) {
      res.addAll(tmp.get(step-1));
      tmp = tmp - 1;
    }
    i = i + 1;
  }
}

while(i > 0) {
  System.out.println(res.get(i-1-ga.size()));
  i = i-1;
}
```

COMS 3101: Programming Language Python
COMS 4111: Introduction to Databases
COMS 4118: Operating Systems
COMS 4140: Networking Laboratory
COMS 6111: Advanced Database Systems
COMS 6232: Analysis of Algorithms II
COMS 4771: Machine Learning
COMS 4772: Advanced Machine Learning
COMS 4995: Deep Learning for Computer Vision
COMS 1004: Introduction to Computer Science and Programming in Java
CSEE 4119: Computer Networks
CSEE 4140: Networking Laboratory
COMS 4111: Introduction to Databases
- Find specific types of locations by using our filter

- Will generate out the shortest path between the source and those specific locations
"DO OR DO NOT. THERE IS NO TRY."
THANK YOU