GaE

Graphs Ain’t Easy

Andrew Jones (adj2129)
Kevin Zeng (ksz2109)
Samara Nebel (srn2134)

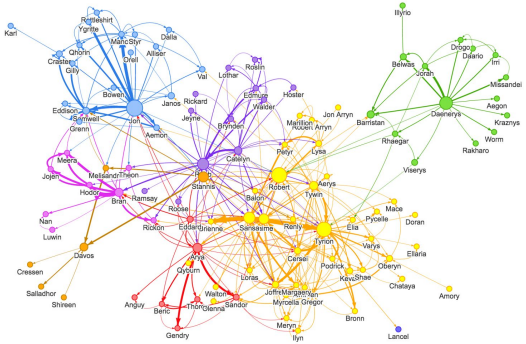
Introduction

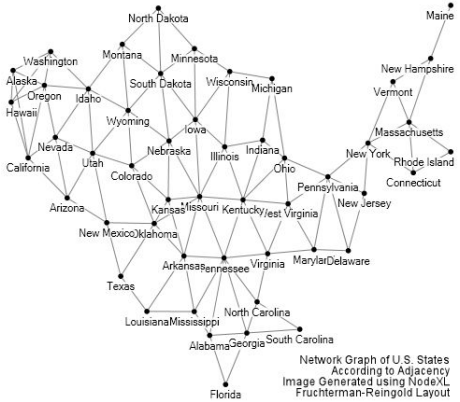
Graphs

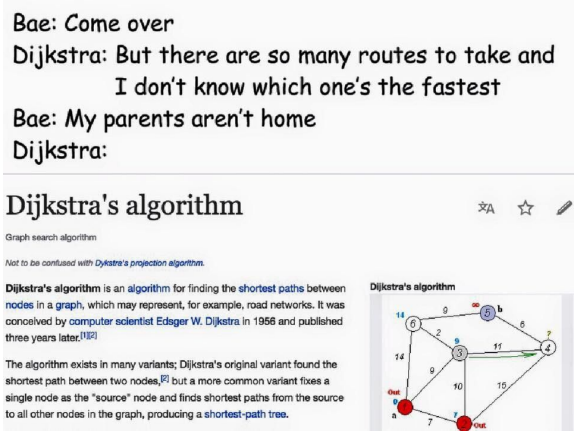
- Complex data structure
- Ubiquitous and fundamental

Goal:

- We want to provide the end user a streamlined interface to easily write programs that read and parse graphs.







Architecture

Scanner | Parser | Semant | Codegen | Linker

Input: source program

Output: tokens

Input: tokens

Output: ast

Input: ast

Output: sast

Input: sast

Output: LLVM IR

Input: LLVM IR and C Library

Output: executable

Data Types

`int`	32-bit signed integer
`double`	32-bit floating point number
`bool`	Boolean - 0 == false, 1 == true
`char`	ASCII character
`string`	An array of ASCII characters
`array`	A list that can store elements of a single type
`map<k,v>`	Variable-size mapping that associates key of type k to value of type v
`graph<n,e>`	Weighted and directed graph with nodes of type n and edge weights of type e
`edge<n, w>`	A three-tuple consisting of source node, destination node, and edge weight where n is the node type and w is the edge weight type
`struct`	A group of data elements grouped together under one name as a type definition

Base types

Container types

Keywords

func	int	double
bool	char	string
map	graph	edge
struct	in	if
else	for	while
return	true	false

Operators

`+, -, *, /, %, ++, --`	Integer operators (add, subtract, multiply, divide, mod, increment, decrement)
`+. , -. , *. , /. , %.`	Double operators (add, subtract, multiply, divide, mod)
`||, &&, !`	Boolean logic operators (or, and, not)
`<, >, <=, >=, ==, !=`	Relational and equality operators (less than, greater than, less than/equal, greater than/equal, equal, not equal)
`:=, =`	Assignment operators
`+`	String operator (concatenation)
`[]`	Array and map operator (index)
`in`	Array, map, and graph operator (in)

Variable Declaration and Instantiation

Variables must be declared before they are instantiated

```
int x;
x := 0;
x = 5;
```

NOTE: formally, `:=` is the assignment operator and = is the re-assign operator, but in practice using either operator will exhibit the same outcomes.

Container types (array, map, and graph) must be instantiated with either a literal or their respective `_init()` function

```
int arr1[];
int arr2[];
arr1 := [1, 2, 3];
arr2 := arr_init();
append(arr2, 1);
```

Control Flow (if, for, while)

If:

```
int x;
x := 5;
if x == 6 {
     printi(1);
}
else {
     printi(2);
}

/* this will print 2 */
```

For:

```
int i;
for i := 0; i < 10; i++ {
     printi(i);
}

/* this will print 0-9 */
```

While:

```
int x;
x := 0;
while (x != 10) {
     printi(x);
     x++;
}

/* this will print 0-9 */
```

Functions

A function declaration has the form:

```
func func_name(parameter-list) return-type
```

Parameter list: A series of variable types separated by commas (can be empty)

Return type must be specified.

Inside the function:

- Variables must be declared at the beginning
- There must be a return statement at the end which returns the corresponding return type

Every program must have a main function:

```
func main() int {}
```

Example:

```
func average_of_two(int x, int y) int {

    int tmp;

    tmp := (x + y) / 2;

    return tmp;

}
```

Arrays and Maps

Arrays:

```
string[] arr;

arr := [“hello”, “world”]
```

Types:

- Primitives: int, double, string, char, bool
- Structs
- Edges

Maps:

```
map<string, int> my_map;

my_map := [“zero”: 0, “one”: 1];
```

Key Types:

- string, int, char, struct

Value Types:

- Primitives

Array and Map Built-in Functions

Arrays:

- `lena(arr)` Returns length of the array.
- `arr[index]` Returns element from the array.
- `arr[index] = value` Utilizes the index operator to change the value stored at the index to the new value.
- `append(arr, value)` Appends the value to the end of the array.
- `arr_init()` Initializes an empty array.
- `el in arr` Returns boolean for whether `arr` contains `el`

Maps:

- `lenm(my_map)` Returns length of the map.
- `my_map[key]` Returns value corresponding to the stored key-value pair.
- `my_map[key] = value` Utilizes the index operator to change the value corresponding to the key. If the key does not exist, this will add a new key-value pair to the map.
- `map_init()` Initializes an empty map.
- `getKeys(my_map)` Returns an array of the keys from the map.
- `key in my_map` Returns boolean for whether `key` is a key in `my_map`

Structs

Declared at the beginning of the program in the global scope. Example:

```
struct My_struct {

    value: int,

    name: string

}
```

Struct attributes may only be base types, i.e. char, bool, int, double, and string.

Variables of this struct type can then be assigned as follows:

```
My_struct var;

var:= { value: 1, name: “hello” };
```

Individual fields can be accessed as well:

```
prints(var.name);

/* this will print “hello” */
```

Edges

Edge: a three-tuple of structs, i.e. (src, dst, val)

Edge is a generic type:

- First type parameter is node type
- Second type parameter is edge value type
- Both types MUST be a struct type

Each Edge represents one directed edge between the two specified nodes with the specified edge value.

```
struct Node {
     name: string
}
struct Value {
     value: int
}
…
edge<Node, Value> e;
e := (
     {name: “src”},
     {name: “dst”},
     {value: 10}
);
```

Graphs

Graph: a collection of edges

Graph is a generic type, with type parameter definitions and restrictions the same as Edge.

Nodes are uniquely identified based on struct equality, i.e. node1 and node2 refer to the same node iff all their attributes are the same.

At most one edge can exist in a graph with the same source and destination node.

```
struct Int {
     value: int
}
…
graph<Int, Int> g;
g := {
  ({value: 1}, {value: 2}, {value: 10}),
  ({value: 1}, {value: 3}, {value: 5}),
  ({value: 1}, {value: 4}, {value: 12}),
  ({value: 2}, {value: 3}, {value: 8}),
};
```

Graph And Edge Built-in Functions

Graphs:

- `graph_init()`
  - Initializes an empty graph. Edges can then be added to the graph using the `addEdge()` function.
- `getNodes(graph)`
  - Returns an array of node structs.
- `getEdges(graph)`
  - Returns an array of edges.
- `addEdge(graph, new_edge)`
  - Adds edge `new_edge` to the graph.
- `n in graph`
  - Returns boolean for whether `n` is a node inside `graph`

Edges:

- `getSrc(edge)`
  - Returns source node struct.
- `getDst(edge)`
  - Returns destination node struct.
- `getVal(edge)`
  - Returns edge value struct.
- `setSrc(edge, node_struct)`
  - Sets the source node of `edge` to `node_struct`.
- `setDst(edge, node_struct)`
  - Sets the destination node of `edge` to `node_struct`.
- `setVal(edge, node_struct)`
  - Sets the edge value of `edge` to `node_struct`.

Demo

Thank you!