giraph

a language for manipulating graphs

Jessie Liu
“jessall.sh”
jll2219

Seth Benjamin
“sethant.ml”
sjb2190

Daniel Benett
“danner.mll”
deb2174

Jennifer Bi
“codejen.ml”
jb3495
motivation

graph algorithms are everywhere!

Bae: Come over
Dijkstra: But there are so many routes to take and
          I don't know which one's the fastest
Bae: My parents aren't home
Dijkstra:

Dijkstra's algorithm
Graph search algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between
nodes in a graph, which may represent, for example, road networks. It was
conceived by computer scientist Edsger W. Dijkstra in 1956 and published
three years later. The algorithm exists in many variants; Dijkstra's original
variant found the shortest path between two nodes, but a more common
variant fixes a single node as the "source" node and finds shortest paths
from the source to all other nodes in the graph, producing a shortest-path
tree.
project workflow: tools
project workflow: timeline

Oct 8, 2017 – Dec 19, 2017

Contributions to master, excluding merge commits

- oct 16: LRM
- oct 30: hello world!
- Nov 17: first parsed graph
- Dec 2: max flow!
- Dec 16: digraphs end to end
- Dec 18: sast into codegen
- Dec 19: maps! and generic graphs
- Dec 20: presenting today!
- Dec 20: presenting today!
**Language overview**

**Comments**
- `!~ this is a comment in giraph ~!`

**Operators**
- `+, -, *, /, %, >, <, >=, <=, ==, :`

**Control Flow**
- `for (i = 0; i < 5; i = i + 1) {}`
- `while (i > 5) {}`
- `if(i == true) {} else {}`

**Non-Graph Types**
- `int, bool, void, float, string, map<> node`

**Function Declarations**
- `int main() {return 0;}`
- `map<int> foo(){map<int> m; return m;}`
language overview: graphs

**types**
- graph
- digraph
- wegraph
- wedigraph

**syntax**
```c
graph<int> g = [A:1 -- B:2 -- C:3 -- A ; D:4 -- A];

digraph<float> g = [A:1.0 <-> B:2.0 ; E:5.0 <- A];

wegraph<string> g = [A:“hi” -{1}- B:“there”];

wedigraph<int> g = [A:1 -{1}-> B:2 <-{2}-> C:3 <-{3}-> D:4];
```
**graph methods:**
add_node(node n)
add_edge(node from, node to)
remove_node(node n)
remove_edge(node from, node to)
has_node(node n)
has_edge(node from, node to)
set_edge_weight(node from, node to, int weight)
get_edge_weight(node from, node to)
neighbors(node n)
print()
for_edge:
graph g = [A:1];
  for_edge(e : g) {
    print(e.from().data());
  }

for_node:
graph g = [A:1 -- B:2];
  for_node(n : g) {
    print(n.data());
  }

bfs:
digraph g = [A:3 -> B:4];
  bfs(n : g ; B) {
    print(n.data());
  }

dfs:
graph g = [A:1 -- C:3; C -- E:5];
  dfs(b : g ; C) {
    print(b.data());
  }
LLVM-side, a graph is represented as a void pointer. This pointer is passed into C library functions. It is a pointer to the head of a linked list of vertex_list_node’s:

```c
struct graph {
    struct vertex_list_node *head;
};

struct vertex_list_node {
    void *data;
    struct adj_list_node *adjacencies;
    struct vertex_list_node *next;
};
```
Edges are represented with an adjacency list. Each `vertex_list_node` has an adjacency list which contains all nodes it has an edge to. Undirected graphs are represented internally with directed edges in both directions.

```c
struct adj_list_node {
    struct vertex_list_node *vertex;
    struct adj_list_node *next;
    int weight;
};
```
Nodes are also represented as void pointers LLVM-side. This is the node’s data pointer, which points to space allocated C-side that is large enough for any of the potential data types (i.e. sizeof(union data_type)).

```
union data_type {
  int i;
  float f;
  char *s;
  void *v;
};
```
A rule of thumb: At any given point, each new feature in codegen is semantically checked.

Used regression test suite with target pass/fail test cases, ensure that other features still worked.
  ○ Node and edge data: assignment and access
  ○ Graph declaration: consistency within graph type
  ○ Graph iteration
  ○ Scoping, nesting
  ○ Maps

If necessary, perform manual checks
  ○ E.g., Parser exception => Run programs with ocamlrun’s parser trace
testing
edmonds-karp code example

Flow network

Max s-t flow
demo!
thank you!
special thanks to our TA Lizzie