giraph

a language for manipulating graphs

Jessie Liu
"jessall.sh"
jll2219

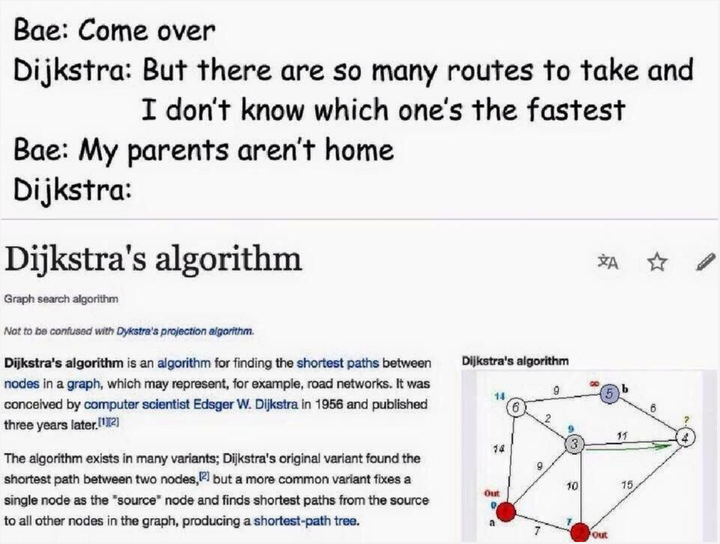
Seth Benjamin
"sethant.ml"
sjb2190

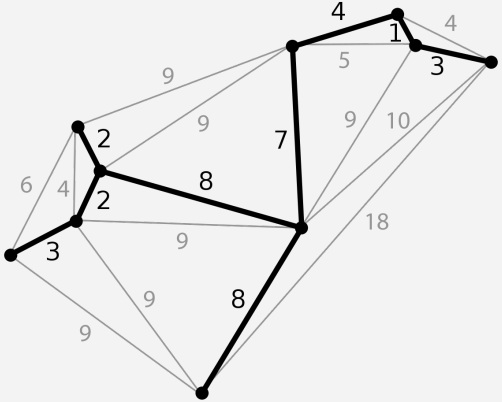
Daniel Benett
"danner.mll"
deb2174

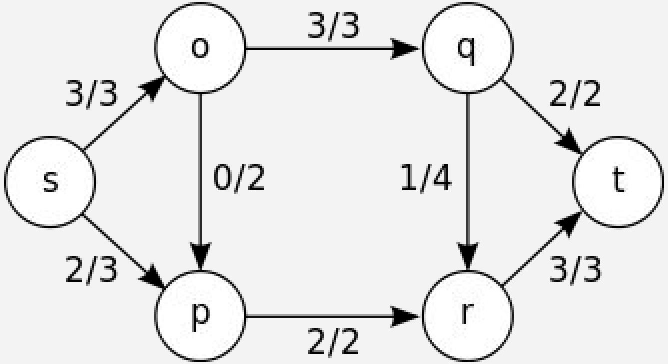
Jennifer Bi
"codejen.ml"
jb3495

motivation

graph algorithms are *everywhere*!







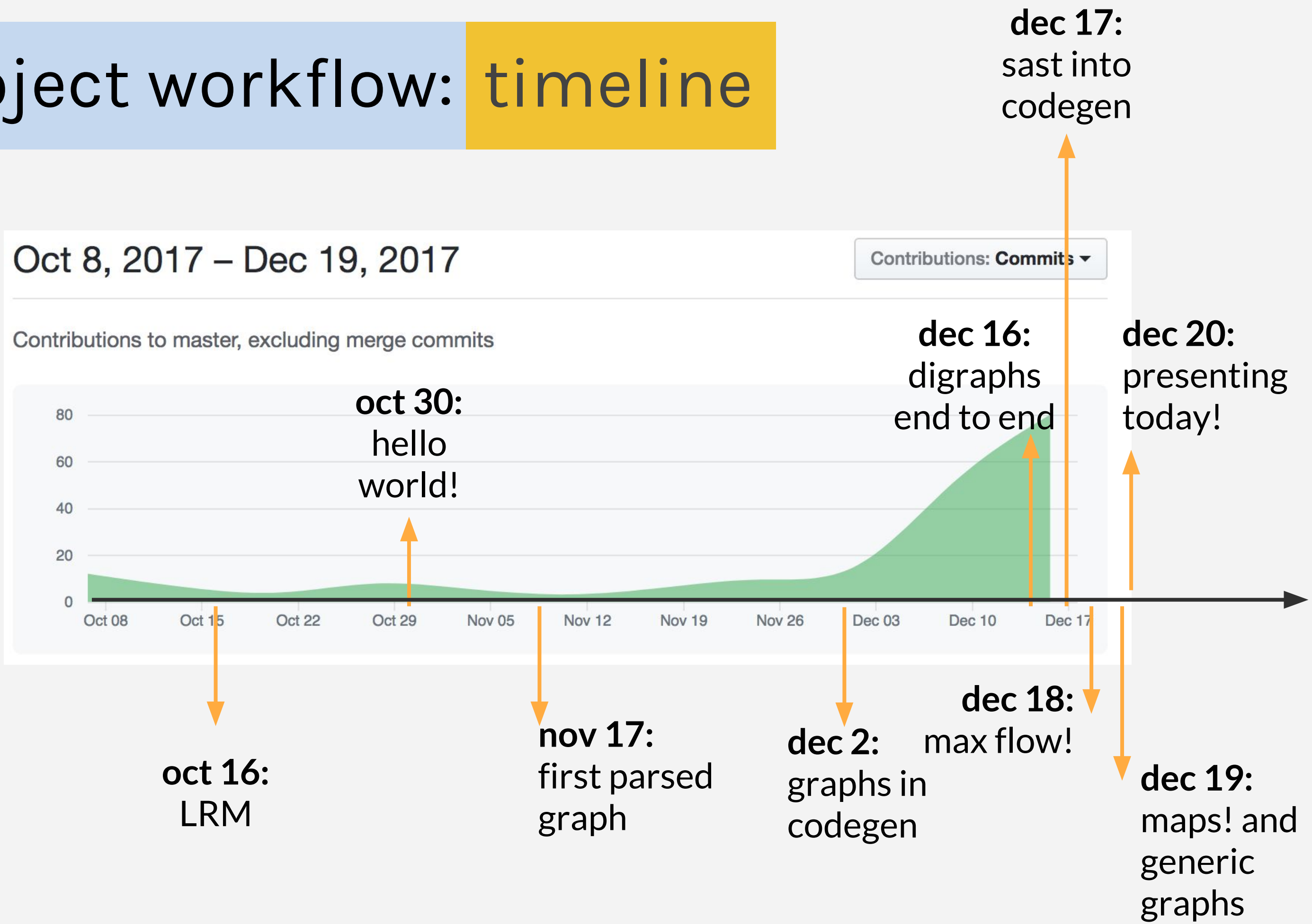
project workflow: tools

OCaml

Vim

THE C PROGRAMMING LANGUAGE

project workflow: timeline



language overview

operators

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

comments

```
!~ this is a
comment in giraph
~!
```

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

```
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];
```

language overview: graph operations

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()
```

language overview: graph iteration

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());
    }
```

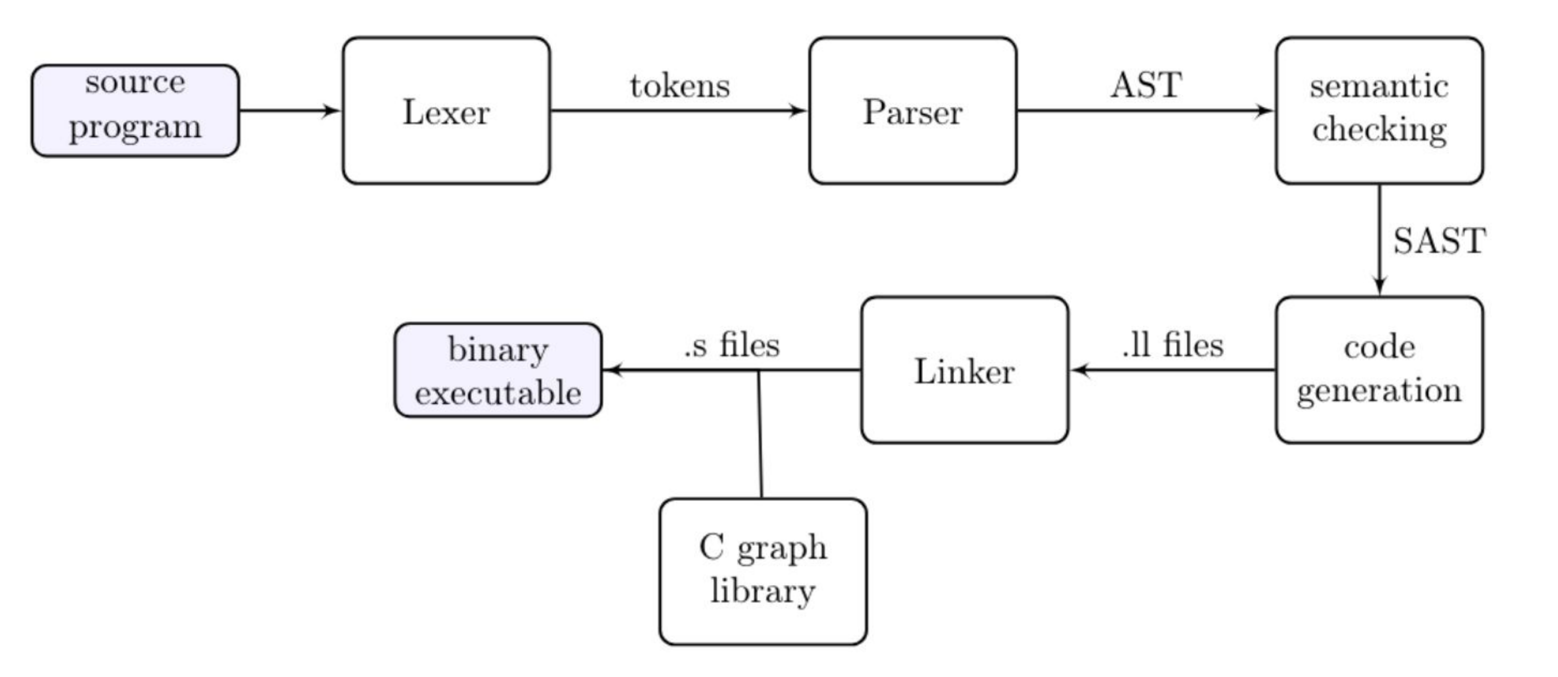
bfs:

```
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());
    }
```

architecture



implementation: graphs

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:

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

```
struct vertex_list_node {
    void *data;
    struct adj_list_node *adjacencies;
    struct vertex_list_node *next;
};
```

implementation: edges

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.

```
struct adj_list_node {
    struct vertex_list_node *vertex;
    struct adj_list_node *next;
    int weight;
};
```

implementation: nodes

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;
};
```

testing

- 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

giraph — bash — 51×40

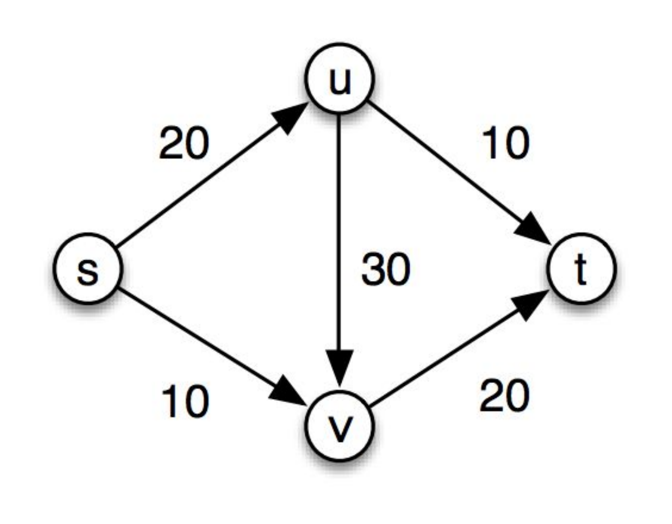
```
OK
-n test-dfs6...
OK
-n test-dfs7...
OK
-n test-dfs8...
OK
-n test-dfs9...
OK
-n test-digraph1...
OK
-n test-foredge1...
OK
-n test-foredge2...
OK
-n test-foredge3...
OK
-n test-foredge4...
OK
-n test-fornode1...
OK
-n test-fornode2...
OK
-n test-fornode3...
OK
-n test-fornode5...
OK
-n test-funcallgraphlit...
OK
-n test-getsetweight1...
OK
-n test-getsetweight2...
OK
-n test-hasedge1...
OK
-n test-hasedge2...
OK
-n test-hasnode1...
OK
-n test-hasnode2...
```

giraph — vim — 84×40

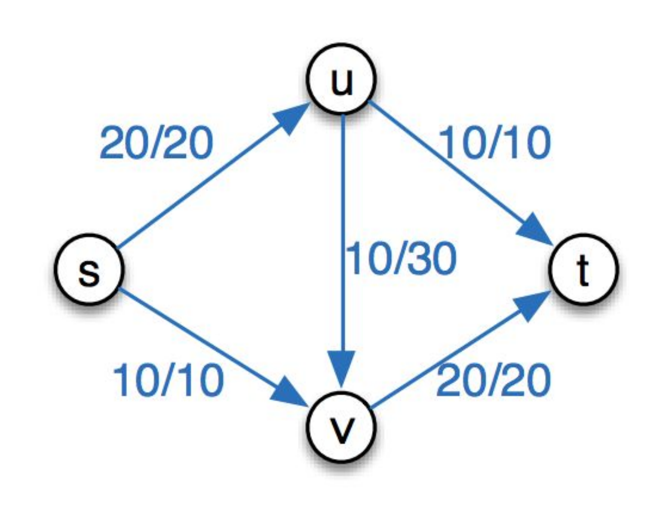
```

###### Testing test-addnode3
./giraph.native tests/test-addnode3.gir > test-addnode3.ll
/usr/local/opt/llvm/bin/llc test-addnode3.ll > test-addnode3.s
cc -o test-addnode3.exe test-addnode3.s graph.o
./test-addnode3.exe
diff -b test-addnode3.out tests/test-addnode3.out > test-addnode3.diff
###### SUCCESS

###### Testing test-addwedge1
./giraph.native tests/test-addwedge1.gir > test-addwedge1.ll
/usr/local/opt/llvm/bin/llc test-addwedge1.ll > test-addwedge1.s
cc -o test-addwedge1.exe test-addwedge1.s graph.o
./test-addwedge1.exe
diff -b test-addwedge1.out tests/test-addwedge1.out > test-addwedge1.diff
###### SUCCESS

###### Testing test-andor1
./giraph.native tests/test-andor1.gir > test-andor1.ll
/usr/local/opt/llvm/bin/llc test-andor1.ll > test-andor1.s
cc -o test-andor1.exe test-andor1.s graph.o
./test-andor1.exe
diff -b test-andor1.out tests/test-andor1.out > test-andor1.diff
###### SUCCESS

###### Testing test-andor2
./giraph.native tests/test-andor2.gir > test-andor2.ll
/usr/local/opt/llvm/bin/llc test-andor2.ll > test-andor2.s
cc -o test-andor2.exe test-andor2.s graph.o
./test-andor2.exe
diff -b test-andor2.out tests/test-andor2.out > test-andor2.diff
###### SUCCESS

###### Testing test-bfs1
./giraph.native tests/test-bfs1.gir > test-bfs1.ll
/usr/local/opt/llvm/bin/llc test-bfs1.ll > test-bfs1.s
cc -o test-bfs1.exe test-bfs1.s graph.o
./test-bfs1.exe
diff -b test-bfs1.out tests/test-bfs1.out > test-bfs1.diff
```

edmonds-karp code example

Flow network



Max s-t flow



demo!

thank you!

special thanks to our TA Lizzie