Assignment 4 (yes, the last one)

CS3310 Compiler Design Lab

due November 2, 2015 at 23:55 on moodle (compiler, SSSP, DFS)
and November 7, 2015 at 23:55 on moodle (new algorithm)

Design a language for simple graph algorithms. Implement two graph algorithms using it. Submit Single Source Shortest Path and Depth-First Search. The output code must be C. We should be able to use your code-generator to generate C code for SSSP and DFS, compile them using gcc and run on some large graphs.

Here are the algorithms. You are allowed to modify them per your wish, or use the following syntax. For instance, you may want to pre-specify that each node would have dist as the attribute in SSSP or dfsnum in DFS or you may have a different syntax for for-loop, or you may support booleans apart from integers. You can assume all graph attributes to be integers.

SSSP.

```
foreach node u {
   u.dist = 999999999;      // some large value
}
0.dist = 0;                 // 0 is always the source.
do {
     continue = 0;
    foreach node u {
      foreach edge(u, v) {
        if u.dist + weight(u, v) < v.dist {
           v.dist = u.dist + weight(u, v);
           continue = 1;
        }
      }
    }
} while continue == 1;

foreach node u {
   print u.dist;  // should print a newline also.
}
```

The C code you generate should take argv[1] as the char *filename which contains the graph in the edge-list format. Edge-list format is as follows:
nnodes nedges
src dst wt
src dst wt
… // nedges lines

The first two entries are number of nodes and number of edges in the graph. The next nedges lines are edges in the form of source, destination, weight (without comma). The graph may have self-loops, duplicate edges, multiple edges between the same pair of nodes and the graph may as well be disconnected. See the sample graphs.

Your SSSP code would be stored in some file, say sssp.algo. We will run your compiler (a.out produced from a4.l and a4.y) as *a.out <sssp.algo >sssp.c*. This C code should be compilable with gcc in DCF. You are allowed to use header files which can be included in sssp.c. For instance, you may write your graph.h structure and *#include "graph.h"*.

gcc sssp.c -o b.out would create SSSP implementation, which would take graph as the second argument argv[1].
b.out p2p-Gnutell04_1.edges

This should print the shortest distances of all nodes (as per sssp.algo above). The distances should match the expected distances (which the evaluation script will compute separately).

Similarly, the **DFS traversal** would be specified in your language. A sample syntax is given below.

```
function DFS(u, num) {
  u.dfsnum = num;
  print u.dfsnum;  // should print a newline also.
  foreach edge (u, v) {
    if v.dfsnum == 0 {
       num = num + 1;
       DFS(v, num);       // recursive.
    }
  }
}
foreach node u {
  u.dfsnum = 0;
}
DFS(0, 1);   // node ids are 0 based, DFS numbering starts from 1.
```

a4.l, a4.y and all required headers should be submitted at the first deadline.
We will then declare a new algorithm on moodle, which you should write in your language and submit by the second deadline. We will run your new algorithm using your compiler and see if it works fine.

Sample graphs.

http://www.cse.iitm.ac.in/~rupesh/temp/p2p-Gnutell04_1.edges
http://www.cse.iitm.ac.in/~rupesh/temp/out.flickr-groupmemberships1.edges
http://www.cse.iitm.ac.in/~rupesh/temp/soc-LiveJournal11.edges

Some assumptions.

- The algorithm would not add / remove nodes / edges. Edge weights would not be changed. Thus, the graph is static.
- All values would be integral (node ids, edge weights, node attributes).
- Number of nodes and number of edges would fit within a word (32 bits), and all values would be non-negative.
- Node ids range from 0 to N-1 where N is the number of nodes.
- Source node in SSSP and DFS would always be 0.
- Each edge in the input graph is directed.
- In the SSSP the distance for unreachable nodes will be 999999999 do not choose any other value, and it is guaranteed in every input Graph any reachable node will always have shortest distance less then 999999999.
- For the DFS as we know that many DFS are possible for a Graph depends on different available branches to proceed. So here we have restriction about which branch should be taken at any point when multiple branches are available. Suppose we have many branches available to proceed at a point then choose the branch which introduced before all other branches in the list of edges.

eg. 1:

```
4       5
0       1       5
0       2       5
1       2       5
1       3       6
2       3       6
```

DFS Traversal will be

```
        0, 1, 2, 3
```

eg. 2:

```
4       5
0       2       5
0       1       5
1       2       5
1       3       6
2       3       6
```

DFS Traversal will be

```
        0, 2, 3, 1
```

Any other traversal will not be accepted even if it is a correct DFS traversal.

- Do not print unreachable nodes in DFS traversal.

  eg.

```
4       5
0       1       4
0       2       6
1       2       5
3       1       2
3       0       3
```

  DFS Traversal will be

```
        0, 1, 2
```

Submission instructions:

Same as before. Please do not include graphs and a.out in ROLLNO.tar.gz.