

The project is due in hardcopy by Monday April 27, 5pm in Prof. Khardon's mailbox in the main office.

Note: If you want to do another project of your own choice, please come to talk to me. I will accept any project that exercises and tests some of the material we covered, as long as it is not too easy or too hard. BUT you must negotiate such a project in advance.

1 Introduction

In this project you will implement two versions of Kruskal's algorithm and compare their run times. Recall that in class we discussed first a simple implementation of Kruskal's algorithm without a special data structure. Pseudocode is as follows:

```
Kruskal-1(V,E,weights):
1 Initialize:  $A = \emptyset$ 
2 Sort  $E$  in increasing order by edge weight
3 For each edge  $(u, v) \in E$  in sorted order do
4   Let  $B = A \cup (u, v)$  (add the edge to  $A$ )
5   if  $B$  does not contain a cycle then
6      $A = B$ 
7 Return  $A$ .
```

In this version, cycle tests (on line 5) are implemented via DFS on the graph of the set B . When implementing this version you should decide how to represent the sets A and B so as to facilitate running DFS on the graph of B . Notice that we can quit the for-loop early if A already includes $V - 1$ edges.

The second version uses a data structure for disjoint sets and the following pseudocode:

```
Kruskal-2(V,E,weights):
1 Initialize:  $A = \emptyset$ 
2 For each  $v \in V$  do Make-Set( $v$ )
3 Sort  $E$  in increasing order by edge weight
4 For each edge  $(u, v) \in E$  in sorted order do
5   if Find-Set( $u$ )  $\neq$  Find-Set( $v$ ) then
6      $A = A \cup (u, v)$ 
7   Union( $u, v$ )
8 Return  $A$ .
```

You should implement this version using the linked list representation with the weighted union heuristic.

2 Input Graphs

We prepared some input graphs for you for testing. These are available at `/comp/160/files/project/`. The files are named using the number of nodes in the graph and the size of E (as a fraction of the max possible). The format is as in the following example where each line after the 2nd gives an edge and its weight.

```
Start
5 nodes 7 edges
1 2 10
3 1 20
4 5 3
2 3 8
1 4 9
4 3 8
5 3 9
End
```

3 Your Program

Your program should include a reader for input graphs (formatted as above) and implementation of the two variants of the algorithm. Choose efficient version of subroutines whenever you can. You may use C, C++, or Java for your program. But you must implement all data structures and algorithms yourself and in particular you should not use any libraries providing facilities beyond the basic types.

4 Experimenting

After developing and debugging your program you should run some timing experiments to see whether you observe differences between the algorithm variants and if so when these are observed. The extent of timing and testing is up to you but at the minimum you should use our graph examples. A more thorough investigation can test the performance as a function of number of nodes and edge density (or other parameter of the graph of your choice) on a family of graphs you generate.

5 What to Submit

Submit (hardcopy only): (1) Copies of your program and any scripts used to run it. Make sure your code is well documented, that you explain how to run the code, and that you explain any

choices made in the implementation (e.g. using XYZ sorting algorithm, using XYZ representation), either within the code or on an extra note. (2) A short report on the timing experiments and their results and any conclusions you can draw from them.

The projects will be graded based on the (1) amount of work done, (2) quality of the code, (3) documentation of the code, (4) explanation of choices of algorithms and data structures used, and (5) quality of the report.