COP 5536 Fall 2013

Programming Project

Due Date: October 25, 2013, 11:55 PM Eastern Time

Submission Website: Sakai

1. General

1. Problem description

You are required to implement Prim's Minimum Cost Spanning Tree algorithm, say **MST**, using a simple data structure, say ***simple scheme***, as well as Fibonacci heap, say ***f-heap scheme***, and measure the relative performance of the two implementations.

Both of the schemes **MUST** use the ***adjacency lists*** representation for graphs. The *simple scheme* uses an array to determine the next edge to include while the f-heap scheme uses a Fibonacci heap to do this. The simple scheme should run in $O(n^2)$ time, where n is the number of vertices in the graph.

2. Programming Environment

You may implement this assignment in Java or C++. Your program must be compilable and runable on the Thunder CISE server using gcc/g++ or standard JDK. You may access the server using Telnet or SSH client on thunder.cise.ufl.edu.

2. Input/Output Requirements

Running modes:

The name of your program should be **mst.cpp** for C++ and **mst.java** for Java. Your program **MUST** support all of the following modes.

(i) random mode:

Run with graphs generated by random number generator. The command line for this mode is:

$ mst –r n d

// run in a random connected graph with *n* vertices and *d*% of density.

// See Performance measurements section for details.

The following points should be noted in generating graphs in the random mode. Assume we have a function *random(k)* that returns a random integer in the range 0 to k-1. Also assume that the n vertices of a graph are labeled from 0 to n-1.

1. To generate an edge set i = random(n), j = random(n) and cost = random(1000) + 1, and add the edge into the graph when edge (i, j) is *not* in the graph.
2. After all edges are generated for a graph you need to make sure that the graph is connected. You can do this by running a depth-first search on the graph. Repeat the process of generating graphs until the graph is connected.

(ii) user input mode:

In the user input mode, your program has to support redirected input from a file "file-name" which contains undirected graph representation. The command line for this mode is:

$mst -s file-name // read the input from a file 'file-name' for simple scheme
$mst -f file-name // read the input from a file 'file-name' for f-heap scheme

In the user input mode, your program must get the following input format:

```
n m             // The number of vertices and edges respectively in the first line
v1  v2  c1      // the edge (v1, v2) with cost c1
v2  v3  c2      // the edge (v2, v3) with cost c2
 …              // a total of m edges
```

Assume that vertices are labeled from 0 to n-1.
An example input is shown below:

```
3 2
0 1 5
1 2 8
```

The graph consists of three vertices {0, 1, 2} and two edges (0,1) and (1,2) with costs 5 and 8 respectively.

In the user input mode, your program should display the cost of this tree in the first line and the edges in the constructed spanning tree in the following n-1 lines. Print the output to the **standard output stream**.
The output for the example shown above is as follows:

```
13
0 1
1 2
```

3. Performance Measurements

The performances will be measured **only** in the **random mode**. The experiment is to generate 10 connected undirected graphs with different edge densities (10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%) for each of the cases n = 1000, 3000, and 5000. Note that an undirected graph of n vertices can have at most $\boldsymbol{n(n-1)/2}$ edges. For each graph assign random costs to the edges in the range between 1 and 1000. Measure the runtimes of the *simple scheme* and the *f-heap scheme* on the same graph.
To get reliable results, run your program at least 5 times for each case specified and average them by dividing the number of runs.

4. Submission

The following contents are required for submission:
1. Makefile: If you do not use this file, provide detail instructions on how compile and run in your REPORT file.
2. Source Program: Provide comments.
3. REPORT:

- The report should be in PDF format.
- The report should contain your basic info: Name, UFID and UF Email account (Not CISE account).
- State what compiler you use, how to compile, and etc.

- Function prototypes showing the **structure** of your programs.
- A summary of result comparison: You should put first your expectation of the comparison before running your program: i.e. what you think about the relative performance of each scheme, and why.
- **Please include the structure of your program. List of function prototypes is not enough.**

4. Draw a plot or a table with execution time followed by comments whether it confirms your expectation. If different, why it is the case and state the factors that influence the results.

To submit, Please compress all your files together using a **zip** or the **tar** utility and submit to the **Sakai** system. You should look for Assignment->Project for the submission. Your submission should be named ***LastName_FirstName.*zip(tar)**. Please make sure the name you provided is the same as the same that appears on the Sakai system.
Please DO NOT submit directly to a TA**. All email submission will be ignored without further notification.**
Please note that the due day is a hard deadline. No late submission will be allowed. Any submission after the deadline will not be accepted.

5. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.
Correct implementation and execution: 60%
Comments and readability: 15%
Report: 25%

Note: If you do not follow the input/output or submission requirements above, 25% of your score will be deduced. In addition we may ask you to demonstrate your projects.

6. Miscellanies

Your implementations should be your own. **You have to work by yourself for this project.**
You can measure execution time like below:

```
#include <time.h>
clock_t Start, Time;
Start = clock();
.......... (your algorithm)
Time = clock() - Start; // time in micro seconds

Measuring execution time (Java)
long start = 0;
start = System.currentTimeMillis();
.....................(your algorthim)
stop = System.currentTimeMillis();
// execution time for your algorithm.
time = stop - start;
```

If you have any question, please contact Eyup Serdar Ayaz at ayaz@cise.ufl.edu.