CS 456: Advanced Algorithms

Programming Assignment #02

Assigned Date	: Friday, October 24, 2014
Due Date	: Friday, October 31, 2014 @ 12:29:59 p.m.

Objectives

Your second programming assignment is to **implement Kruskal's minimum spanning tree algorithm**. The main objective of this assignment is to **empirically validate** the most efficient implementation of Kruskal's algorithm. Hence, the sub-objectives of this assignment are:

1. To implement the Union-Find data structure;
2. To use union-find data structure to develop an efficient implementation of Kruskal's algorithm; and
3. To validate the theoretical best runtime complexity for Kruskal's algorithm using empirical results;

Thus, you are expected to properly demonstrate that you were able to achieve these objectives through documented work listed in your project report.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- Read all instructions provided here thoroughly and carefully.
- The report part of your solution must be produced using a word processor. Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. Graphs/plots should be properly labeled.
- You may use **any programming language** of your choice. However, you should make sure that your code compiles and runs on a typical Linux machine.
- Your report must be in **PDF** format.
- Follow a good coding standard. Use the CS department programming styling guide found here `https://www.cs.siue.edu/programming-style-guide`.
- You are required to make your submission through Moodle. A dropbox will be opened within a 24 hour window of the deadline and will continue to be open for an additional 24 hours. There is no extra credit for submitting early but there will be a 15% penalty for being late. Please carefully read the Deliverables section below for more details.
- Total points: [**100 points**]

I/O Specifications

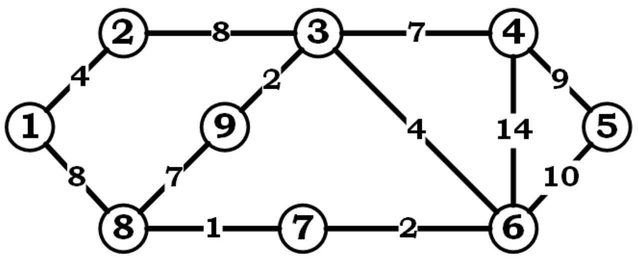
Your program should be capable of reading an input file that describes a graph using the following format:

```
<source-vertex> <destination-vertex> <weight>
```

You may consider having the first line of this file to list the total number of vertices and edges of your graph, if such a listing helps you create your data structure, but this is not mandatory. You are not to assume any particular ordering of the listing, or the type of edge weights (edge weights can be either integer or decimal) but you may assume the vertex IDs to be *non-decimal*.

Your program should use this input file to calculate the minimum spanning tree for the given graph and generate an output file that lists the MST based on Kruskal's greedy selection order. Furthermore, your output file should list the time that it took your program to find the MST, along with the total cost of the MST.

As an example, consider the following undirected graph[*] and its corresponding edge list `Input.dat` and `Output.dat` files.



Input.dat

```
9 12
1 2 4
2 3 8
3 4 7
4 5 9
1 8 8
3 9 2
7 8 1
8 9 7
4 6 14
5 6 10
6 7 2
3 6 4
```

Output.dat

```
Time   : x seconds
MST    :
         8 7 1
         6 7 2
         3 9 2
         1 2 4
         3 6 4
         3 4 7
         1 8 8
         4 5 9
Cost   : 37
```

The instructor will independently verify the programmatical correctness of your submitted solution using his own test file that may include a large number of vertices and edges. Thus, you are also not to assume any particular size limitation of the graph your program can handle.

Deliverables

The due date of this assignment is **Friday, October 31, 2014 @ 12:29:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- A report **in PDF format** that includes the followings:
  - **Motivation and Background :** Use this section to explain succinctly your approach to achieving the objectives listed above [**10 points**]
  - **Description :** A pseudocode listing of your program. If you are using classes, make sure to also include an appropriate UML class interaction diagram [**5 points**]
  - **Testing Plan :** In particular, what you consider to be the true runtime complexity of the algorithm and how you plan to separate that from auxiliary tasks such as input processing and output file generation. You must properly justify any assumptions you make. [**10 points**]

[*] edge 4-5 weight corrected on 10/26/14 @ 9:16am

- **Results and Analysis :**
  - **Theoretical Results**: A description of the expected asymptotic behavior of your program. You may (and probably are advised) use your pseudocode to aid this explanation [**15 points**]
  - **Empirical Results**: Observations from your experiments. You should probably repeat each experiment several times to eliminate any statistical errors, but list the outcomes of each run in tabular format. Also, you are expected to produce output for at least three problems – a smaller size problem, a medium size problem, and a large size problem. [**15 points**]
- **Analysis :** Prove (or disapprove) that your implementation of the Kruskal's algorithm achieves the theoretical runtime complexity. Provide a comparative plot between theoretical expectation vs. empirical outcome. Explain what you see. [**20 points**]
- **Conclusions :** Describe your observations and conclusions of your experiment including any problems encountered and/or any key insights you've discovered. [**10 points**]
- **Instructions :** How to compile and run your program [**5 points**]
- **Program listing** [**5 points**]
  - Good programming structure (headers, variable names, code re-use, functional decomposition, object-oriented design, and comments)
- References

- A compressed tarball of the directory containing your source codes. Do not include executables, test input or sample outputs in this tarball. I will generate them using my own test input files. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name111-pr2.tar.gz source/`. Obviously, change the name to your last name and 111 to the last three digits of your SIUE ID [**5 points**].

Finding large datasets to experiment will require you to either research what's out there or generate them using your own code. If you do use resources other than your own, make sure to properly cite them in your report and give due credit to the original authors. By absolutely no means, you are allowed to plagiarize code.

- SNAP Dataset found at `http://snap.stanford.edu/data/`
- Pajek Dataset found at `http://vlado.fmf.uni-lj.si/pub/networks/data/`

Here are couple of resources to look for large datasets. You may have to parse them to the input specification described above. Use your scripting skills (or write an appropriate parser). If you use an outside resource to do the parsing for you, make sure to cite the source.

Extra Credit (required for MS students)

Empirically validate the runtime complexity of a different implementation of Kruskal's algorithm that does not use the union-find data structure and compare the results against the findings above. This is a mandatory task for MS students but considered extra credit for undergraduate students. However, your project documentation should now be properly augmented to reflect your work. Good for [**50 points**].