# CS 456 : Advanced Algorithms

**Programming Assignment #01**

**Total Points: 150**

**Assigned Date** : Wednesday, January 29, 2020
**Due Date** : Wednesday, February 12, 2020 @ 09:59:59 a.m.

## Objectives

Your first programming assignment is to **implement Kruskal's and Prim's minimum spanning tree algorithms**. The main objective of this assignment is to **empirically validate** the most efficient implementation of a minimum spanning tree algorithm. More importantly, you are fully expected to firsthand experience how, even for the same algorithm, the underlying data structure influences the overall runtime. Hence, the sub-objectives of this assignment are to:

1. implement Priority Queue -based Prim's and Kruskal's algorithms;
2. implement a Min-Heap -based Prim's algorithm;
3. implement a Union-Find -based Kruskal's algorithms; and
4. validate the theoretical best runtime complexities for both algorithms using empirical results.

Also, you are expected to think about and address the following questions in your report:

a. At what size $n_0$ does your implementation(s) start to exhibit asymptotic complexity?
b. How does the graph density affect runtimes of each algorithm?
c. What is the characteristic of your input required to generate *average* complexity. How about best case and worst case scenarios?
d. How do you plan to generate the appropriate input required to exhibit different asymptotic behaviors?
e. How does the measured run time correspond to the abstract complexity analysis using operation counting (as discussed in class)?

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!!**. Use of a version control software is highly recommended!
- Make sure to test your program properly before your final submission. It is **highly** recommended to test build and run your submission on the home server, **home.cs.siue.edu**.

- You may use any programming language of your choice out of `C, C++, Java` or `Python` (or a language you have mutually agreed with the instructor) as these are all available on the home server. However, you **must** make sure that your code compiles and runs on a typical Linux machine.
- Absolutely **DO NOT** include executables with your submissions.
- A **Makefile** is mandatory. Whether or not your program needs to be compiled, have it echo instructions to run the program.
- For input, the program must take as a command line argument the file path to the input file containing the graph information and another argument which is to be the output file name. For example `./kruskal <inputfile> <outputfile>`. **Be aware that the file locations will most likely NOT be in the same directory as the executable**
- For output, the program should print to a file that includes measured runtime, the MST found, and the cost of the MST. See I/O specifications below.
- The report part of your solution must be produced using a word processor. LaTeX is highly recommended (not a requirement).
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. If using LaTeX, the pgfplots package is very useful for making all sorts of graphs.
- Be professional with your reports; properly label and title your graphs; properly caption and cross-reference your figures; make sure to include all sections/subsection mentioned below.
- Your final report should be in **PDF** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.

## I/O Specifications

### Input

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

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

An example line from the input file would look like `0 1:4 2:4 3:1 4:4 5:4 6:2 7:1 8:1 9:4`

Each line corresponds to edges coming from one vertex. This example has an edge from 0 to 1 with weight 4, 0 to 2 with weight 4, 0 to 3 with weight 1, etc.

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. It is also possible that there are multiple graphs per file meaning that there could be two groups of vertices that are not connected to each other in any way.

### Output

Your output file should list the time that it took your program to find the MST(s), along with the total cost of each MST.

The format of the graph should use the format of the input.

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 **Wednesday, February 12, 2020 @ 09:59:59 a.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- [**120 points**] A report that includes the followings:
  - Motivation and background of the experiment. Use this section to explain succinctly your approach to achieving the objectives listed above. [**5 points**]
  - A separate section for each implementation [**45 x 2 points**]:
    - Pseudocode of the algorithms used in the implementation. If multiple implementations use the same algorithm, only one pseudocode is needed. If it is repeated, state as such. The pseudo code will count for each implementation, so if it is used twice, it will be worth 10 points total. [**5 points**]
    - Correctness proof of the algorithms used in the implementation. Again, only one is needed even if the algorithm is used in multiple implementations just like the pseudo code. [**10 points**]
    - Testing Plan (for best/average/worse cases). Discuss 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. [**5 points**]
    - **Expected Results**: A description of the expected asymptotic behavior of your program. You may (and probably are advised to) use your pseudocode to aid this explanation. [**5 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. Use enough data points such that a clear pattern of execution time is developed. You must include a graph of the run time and convince the reader that the graph follows the trend you say it does. [**10 points**]
    - Justification of your observations. You must be able to justify and/or argue the empirical asymptotic behavior you are observing. [**10 points**]
  - Conclusion and overall performance comparisons. [**25 points**]

  **Note:** Professionally format and write your report. Treat your report as if you would be submitting it for publication. Failing to do so can reduce your score by as much as 10%.
- [**30 points**] A compressed tarball of the directory containing **ONLY** your source code files, and a Makefile. Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name-111-pr1.tar.gz source/`. Obviously, change the name to your last name and 111 to the last three digits of your SIUE ID.
  - Correct implementation of algorithms. [**15 points**]
  - Correct input procedures. [**5 points**]
  - Correct output procedures. [**5 points**]
  - Good coding practices e.g. naming conventions, readable code, commenting, etc. [**5 points**]

Each implementation should have its own program. In this project, that would be one for Kruskal's with union-find, one for Kruskal's with priority queue, and one for Prim's.

## Generating Test Graphs

To generate random graphs, the following program is available to you via `Github` https://github.com/SnugglyCoder/graph-generator. Clone it, build it (a `makefile` is included), and use it on a Linux machine with `make`. The repository `readme` contains instructions on how to use and run the program. There are also issues listed in the repository. Hence, feel free to contribute to the open source project and help it grow and become more useful for future use!
Some other useful graph generators and large datasets (that can be converted to graphs):

- http://graphstream-project.org/
- http://law.di.unimi.it/datasets.php
- https://dnac.ssri.duke.edu/datasets.php
- https://snap.stanford.edu/data/
- https://github.com/sepandhaghighi/pyrgg
- http://networkrepository.com/

## Extra Credit Options

### Faster Algorithms

Runtime of both these algorithms can be further improved using more sophisticated (and tedious to implement) data structures such as `Fibonacci Heaps` (**CLRS** Ch.19) and `Relaxed Heaps` (https://www.cs.princeton.edu/research/techreps/TR-109-87). Time permitting, you may attempt to empirically validate and compare the runtime complexity of one (or both) spanning tree algorithms based-off of one of these advance data structures against the core assignment tasks. There are also linear-time randomized algorithms that you may be interested in validating. Some information can be found here https://algs4.cs.princeton.edu/lectures/43MinimumSpanningTrees.pdf.
If you go down this path, your project documentation should be properly augmented to reflect your "extra credit" work. Good for extra [**45 points**].