CO 353 Winter 2018: Project 1

Due: February 5 at 8pm

1 Objective

The project consists in implementing *both* Prim's algorithm and Kruskal's algorithm for the minimum spanning tree problem. The implementation must be *correct* and *efficient*. In particular, given a graph with n nodes and m edges,

- the implementation of Prim's algorithm must have $O(n^2)$ complexity, and
- the implementation of Kruskal's algorithm must be $O(m \log m)$, even in the special case where $m = O(n)$ instead of $O(n^2)$.

The above complexity bounds are in the arithmetic model, i.e. addition, subtraction, multiplication, division comparison and assignment are $O(1)$.

This project can be completed in groups of 1 or 2. You can also form groups of 3, but *only* if you first get my approval by email (lpoirrier (at uwaterloo)) by January 21st.

2 Input and output

The input is a connected graph $G = (V, E)$ with node-set V (with $n = |V|$) and edge-set E (with $m = |E|$). Positive integer edge costs $c_e \in \mathbb{Z}$ are given for each edge $e \in E$. An upper bound $c_{\max}$ is also given on the edge costs: for all $e \in E$, we have $1 \leq c_e \leq c_{\max}$. The graph is given in a file containing $1 + m$ lines. The first line is of the form:

n m $c_{\max}$

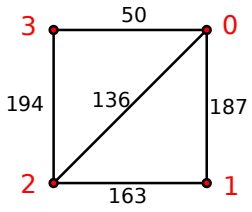
and is followed by m lines of the form:

u v c_{uv}

indicating an edge from node u to node v with cost c_{uv}. Note that both u and v describe the corresponding node by its index between 0 and $n-1$, i.e. $u, v \in \{0, 1, \ldots, n-1\}$.

A complete example is given by

```
4 5 200
0 1 187
0 2 136
0 3 50
1 2 163
2 3 194
```



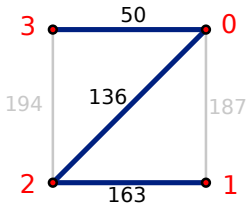
where the first line indicates that there are $n = 4$ nodes, $m = 5$ edges, and edge costs c_e are at most $c_{\max} = 200$. Then, the subsequent 5 lines give the 5 edges of the graph by specifying the adjacent nodes, and the corresponding edge costs. On the figure, the nodes are given by their indices.

The output must be a minimum spanning tree given in a file consisting of $n-1$ lines. Each line is of the form:

```
u   v
```

In our example, a correct solution file could be:

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



Note that, in order to (optionally) let you include more information in the output file, everything between a # sign and the end of a line will be ignored. So, for example, the following is also correct:

```
# n = 4, m = 5, cmax = 200.  MST cost 349.
0 3     # cost fifty
0 2     # cost 136
1 2     # some other comment
```

More example input and output files are given at `https://www.math.uwaterloo.ca/~lpoirrie/co353.html`.

3 Running the code

Your code must take exactly two arguments. The first argument is the name of a file containing the input graph. The second argument is the name of a file where the output tree is to be written.

You have a choice between the following languages for the implementation: C, C++ or Python.

3.1 In C or C++

Your code must be portable to any environment with a standards-conforming C or C++ compiler. It must be possible to compile and run it by using the following commands:

in C:

```
gcc -O3 -o prim prim.c
./prim input_graph.txt output_tree1.txt
gcc -O3 -o kruskal kruskal.c
./kruskal input_graph.txt output_tree2.txt
```

in C++:

```
g++ -O3 -o prim prim.cpp
./prim input_graph.txt output_tree1.txt
g++ -O3 -o kruskal kruskal.cpp
./kruskal input_graph.txt output_tree2.txt
```

In other words, you must implement your code for Prim's algorithm in a file called `prim.c` (`prim.cpp` in C++), and your code for Kruskal's algorithm in a file called `kruskal.c` (`kruskal.cpp` in C++).

Note: Optionally, you may provide a `Makefile`, in which case your code will be compiled by running `make`. If you choose to do this, it is your responsibility to ensure that the `Makefile` is correct, so do it only if you are familiar with `Makefile`s already.

3.2 In Python

It must be possible to run your code by executing the following commands:

```
python prim.py input_graph.txt output_tree1.txt
python kruskal.py input_graph.txt output_tree2.txt
```

In other words, you must implement your code for Prim's algorithm in a file called `prim.py`, and your code for Kruskal's algorithm in a file called `kruskal.py`.

4 Submitting the project

You submit your implementation by sending a single email to both `lpoirrier (at uwaterloo)` and `wjtoth (at uwaterloo)`, by 8pm on Monday, February 5th, 2018. Late submissions will not be accepted. The subject line of your email must contain the string `CO353`. Your email must contain **a single attachment file**: an archive in the format `.zip` or `.tgz`. The name of the archive is formed by the UWaterloo IDs of your group members, in any order, separated by underscores. For example: `jwtoth_lpoirrie.tgz`. The archive contains (at least) three files:

- the source file for your implementation of Prim's algorithm,
- the source file for your implementation of Kruskal's algorithm, and
- a file called `notes.txt` (or `notes.md` if you prefer), see below.

The file `notes.txt` mentions the names of the (1, 2 or 3) members of your group. It also mentions the algorithmic complexity of the two implementations, and contains a short explanation of the algorithms and data structures that you used to achieve this complexity. The file should be between 10 and 50 lines long, approximately.

5 Contribution

You may use the standard library in the language you chose. No other library is allowed. In case of doubt ask me. You can consult any source of information you want (books, internet, scientific papers, etc.), but copying code is not allowed, regardless of the source. Every single line of your code must be written by a member of the group. Each member of the group must understand (and must be able to justify) every line of the code, including the ones they did not write. After the due date, I may call an individual group member to come to my office hours; if that member does not show sufficient understanding of the code, *all* group members will get zero.

6 Grading criteria

The projects will be graded on a total of 10 marks, divided in 4 categories. Pay attention to the first two: if your code does not follow the specification or is incorrect, it will not be tested for speed, and you will get zero for the

fourth category (efficiency).

- Specification (1 mark). Your submission must follow the specifications given in this document (your email must have the correct recipients/subject/attachment, your source files must follow the template given above, your code must compile and take the right arguments). The rules are rigid because testing will be automated.
- Correctness (2 marks). The output must describe a minimum spanning tree.
- Code quality (2 marks). It should be reasonably easy to understand your code. Comments can be used to help in that regard, but with moderation. The code itself must be clear and readable.
- Efficiency (5 marks). Your code must have the prescribed complexity ($O(n^2)$ for Prim, $O(m \log m)$ for Kruskal). It is your responsibility to understand the computational complexity of the functions and data structures that you use, even if they are part of the core language. For example, if you use an array of elements, removing an element in the middle is $O(n)$, even if it appears to be a single operation. This can be done in $O(1)$ with a linked list.

  Note: if your code is so fast that reading the input becomes the bottleneck, we will measure the time taken specifically by the Prim/Kruskal parts of your code.