ELL781: Assignment 2

Submission deadline: **22 October, 23:59**

Maximum Marks: 10 (20% extra credit for single-person attempts)

1 Problem statement

This assignment involves implementing the Prim and Kruskal algorithms for finding a Minimum Spanning Tree (MST) of an undirected, weighted graph.

2 Procedure

Here are the steps you should follow for this assignment:

1. *Prim's algorithm*: Start with the pseudocode discussed in class, or the Pascal code given in Fig. 7.8 of the *AHU* book. This is essentially what you need to implement, in a programming language of your choice (ideally C/C++/Java/Python). However, one change needs to be made: rather than just taking a cost matrix as input and printing a set of edges as output, you need to take as input and return as output instances of a `GRAPH` data structure. You may re-use your `GRAPH` data structure from Assignment 1, but you are also allowed to make changes to it, if needed (for example, you will need to allow for weighted, undirected graphs now). However, the modularity must be preserved: there must be a separate, generic `GRAPH` data structure with a defined set of functions/operations, and the main program (for Prim's algorithm) must interact with the data structure only through these functions.

2. *Kruskal's algorithm*: Start with the pseudocode given in Fig. 7.10 of *AHU*. As you will notice right away, this is considerably more complicated to implement than Prim's algorithm. In particular, this pseudocode makes use of two new ADTs: `PRIORITYQUEUE` and `MFSET`. So first of all these will need to be implemented. Here is how you should proceed (please follow these steps carefully):

    - Implement a `PRIORITYQUEUE` data structure. In addition to the discussion in class, please look at Sections 4.10 and 4.11 of *AHU* for details on how this can be done. We are not asking you to submit separate written pseudocode this time; but your final code should be well-commented and it should be easy for anyone reading it to understand how each part works.
    - Impelment an `MFSET` data structure. In addition to the discussion in class, look at Section 5.5 of *AHU* to understand how it can be implemented. There are multiple ways of doing so; you may pick one as you prefer, but should aim to minimise time complexity. Again, the code should be well-structured and commented for easy readability.
    - Using both the above data structures, implement Kruskal's algorithm. Like for Prim's, your input and output should be in the form of an instance of your `GRAPH` data structure.

3. Write a top-level program for creating a `GRAPH` instance and running both Prim's and Kruskal's algorithms on it, and printing out the respective MSTs returned, along with the total cost and runtime (in milliseconds) for each one. The input to this top-level program

should be an $n \times n$ cost matrix for a graph, n being the number of vertices (assume the diagonal elements will always be 0, and elements corresponding to unconnected vertices will be set to -1); this can be read in from a text file. A sample input file is provided at `http://web.iitd.ac.in/~sumeet/input_graph.txt`, which corresponds to the graph depicted in Fig. 7.4 of *AHU*.

Logically, your code overall should consist of the following 6 components/modules (it is desirable to have each component in a separate file):

- A data structure (plus associated functions) which implements the `GRAPH` ADT (you may re-use your code from Assignment 1, with appropriate changes). **[1]**
- An implementation of Prim's algorithm, as a function which takes a `GRAPH` instance as input and returns another `GRAPH` instance as output. **[2]**
- A data structure (plus associated functions) which implements the `PRIORITYQUEUE` ADT. **[2]**
- A data structure (plus associated functions) which implements the `MFSET` ADT. **[2]**
- An implementation of Kruskal's algorithm, as a function which uses the two data structures just mentioned, and takes a `GRAPH` instance as input and returns another `GRAPH` instance as output. **[2]**
- A top-level program which is a function that takes as input the cost matrix for a graph in the given format, creates a `GRAPH` instance using it, calls the Prim and Kruskal functions on this instance and gets the `GRAPH` instances returned by each of them (whilst also keeping track of the runtime for each call), and prints out these MSTs in the given format along with the respective total costs and runtimes. **[1]**

2.1 Output format

Your top-level program should print out the MSTs from the Prim and Kruskal algorithms, in the form of a sequence of $n-1$ edges (vertex pairs), where n is the number of vertices in the input graph. Each MST should be preceded by a line which gives the name of the algorithm that generated it, the total cost of the MST, and the runtime it took your code to generate the given MST. For the sample input file linked to above, the output might look as follows:

```
Prim's algorithm MST (total cost:  15; runtime:  17ms)
(1,3)
(2,3)
(2,5)
(3,6)
(4,6)
Kruskal's algorithm MST (total cost:  15; runtime:  41ms)
(1,3)
(2,3)
(2,5)
(3,6)
(4,6)
```

3 What to submit

You need to submit all your code, properly commented and documented for easy readability as described. Please package everything into a single zip/tar/rar file. You should also include a README file which explains to the user exactly how they should use your code to generate the MSTs for a given graph.

Submission will be via Moodle (`https://moodle.iitd.ac.in/`); the exact submission procedure will be announced there in due course. The submission deadline for this assignment will be **22 October, 23:59**. Any late submissions will be penalised, and may not be evaluated at all, depending on the extent of delay.

4 Collaboration policy

You are free to discuss any aspect of the assignment with others; however, your code must be entirely written by you (and your partner, if applicable), without any copying from anywhere (the only exception is code re-used from Assignment 1). We will be using plagiarism-detection tools to ensure that this is the case, and any violations will lead to the loss of all marks for the assignment, plus further disciplinary action depending on the severity of the offence.