CSC 226 SUMMER 2020
ALGORITHMS AND DATA STRUCTURES II
ASSIGNMENT 2 - PROGRAM
UNIVERSITY OF VICTORIA

1 Programming Assignment

The assignment is to implement an algorithm to determine if the minimum weight spanning tree of an edge-weighted, connected graph G, with no self-loops or parallel edges, is the same when using Prim's algorithm as it is when using Kruskal's algorithm. The edge weights in G will be real numbers greater than 0 and will not necessarily be distinct. A Java template has been provided containing an empty method PrimVsKruskal, which takes a single argument consisting of a weighted adjacency matrix for an edge-weighted graph G with real number edge weights all greater than 0. The expected behavior of the method is as follows:

Input: An $n \times n$ array G, of type double, representing an edge-weighted graph.

Output: A boolean value which is true if the Prim's MST equals the Kruskal's MST and false otherwise.

A correct implementation of the PrimVsKruskal class will find the minimum weight spanning tree using the textbook's implementation of Prim's algorithm (eager version) and the minimum weight spanning tree using the textbook's implementation of Kruskal's algorithm and compare. If they are the same it returns true, otherwise false.

You must use the provided Java template as the basis of your submission, and put your implementation inside the PrimVsKruskal method in the template. You may not change the name, return type or parameters of the PrimVsKruskal method. You may add additional methods as needed. You may use any of the classes provided by the textbook as your code will be run with the algs4.jar file. Most likely you will use some or all of the following: UF class, IndexMinPQ class, MinPQ class, Edge class, Queue class, etc. The main method in the template contains code to help you test your implementation by reading an adjacency matrix from a file. You may modify the main method to help with testing, but only the contents of the PrimVsKruskal method (and any methods you have added) will be marked, since the main function will be deleted before marking begins. Please read through the comments in the template file before starting.

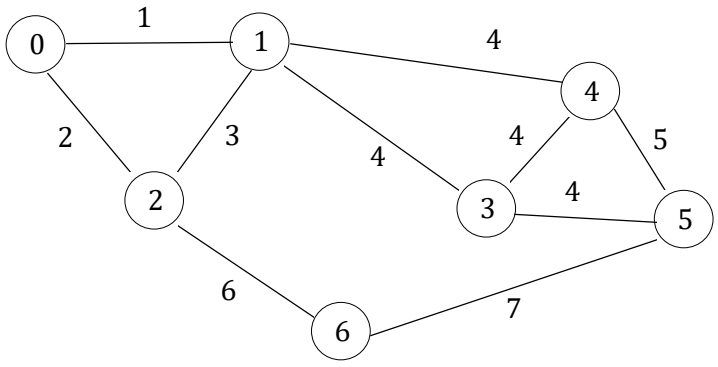
I foresee three ways of solving this problem, each increasingly more difficult and worth increasingly more marks. (1) The simplest way to solve this problem is to convert the adjacency matrix into an EdgeWeightedGraph object, running PrimMST and KruskalMST on the graph and then comparing the output trees to one another. (2) Another way to solve the problem is to leave the graph as an adjacency matrix and modifying the code of PrimMST and KruskalMST (inside your PrimVsKruskal class) to work directly with the adjacency matrix. You would still generate the two MSTs and compare them. (3) The most difficult but most efficient way to solve it also involves working directly with the adjacency matrix but here you build the two trees concurrently, testing the consistency of the trees after adding each edge. We will call this an "early detection system", which will potentially recognize if the two trees are unequal before completion of the trees.

2 Input Format

The testing code in the main function of the template reads a graph in a weighted adjacency matrix format and uses the PrimVsKruskal class to compare the two minimum spanning trees. A weighted adjacency matrix A for an edge-weighted graph G on n vertices is an $n \times n$ matrix where entry (i, j) gives the weight of the edge between vertices i and j (or 0 if no edge exists). For example, the matrix

$$A = \begin{bmatrix} 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 4 & 4 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 & 6 \\ 0 & 4 & 0 & 0 & 4 & 4 & 0 \\ 0 & 4 & 0 & 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 & 5 & 0 & 7 \\ 0 & 0 & 6 & 0 & 0 & 7 & 0 \end{bmatrix}$$

corresponds to the edge-weighted graph below. Note that the weighted adjacency matrix for an undirected graph is always symmetric.



The input format used by the testing code in main consists of the number of vertices n followed by the $n \times n$ weighted adjacency matrix. The graph above would be specified as follows:

```
7
0 1 2 0 0 0 0
1 0 3 4 4 0 0
2 3 0 0 0 0 6
0 4 0 0 4 4 0
0 4 0 4 0 5 0
0 0 0 4 5 0 7
0 0 6 0 0 7 0
```

3 Test Datasets

A collection of randomly generated edge-weighted graphs with positive, real number edge weights will be used to test your code. I will provide some example files but you are encouraged to create your own test inputs to ensure that your implementation functions correctly in all cases.

4 Sample Run

The output of a model solution on the graph above is given in the listing below.

```
Reading input values from test1.txt.
Does Prim MST = Kruskal MST? true
```

5 Evaluation Criteria

The programming assignment will be marked out of 25, based on a combination of automated testing and human inspection. To receive full marks, the algorithm will determine if the MST found by Prim equals the MST by Kruskal, while building both trees simultaneously and including some kind of early false detection technique. That is, to get full marks you cannot simply generate the MSTs then test if they are the same. You will need to build both simultaneously, while efficiently checking as they grow if the answer is sure to be false, insuring a more efficient best-case run time.

Score (/50)	Description
0 − 5	Submission does not compile or does not conform to the provided template.
5 − 15	The implemented algorithm is inaccurate on the tested inputs or only calls PrimMST and KruskalMST.
15 − 20	The implemented algorithm is accurate on all tested inputs and you rewrite Prim's and Kruskal's to work directly on the adjacency matrix.
20 − 25	The implemented algorithm is accurate and has an efficient early false detection scheme.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template. You may only submit one source file. **If your submission does not compile for any reason (even trivial mistakes like typos), or was not based on the template, it will receive at most 5 out of 25.** The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.