Introduction to Algorithms	Problem Set 2
CS 4820, Spring 2019	Due 11:59pm Thursday, February 7

Hand in your solutions electronically using CMS. Each solution should be submitted as a separate file. Collaboration is encouraged while solving the problems, but:

1. **list the names of those with whom you collaborated;**
2. **you must write up the solutions in your own words;**
3. **you must write your own code.**

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size.

(1) *(10 points)* Suppose n (≥ 1) people are stranded on a freeway due to a particularly heavy snowstorm. Let us model the freeway as the real line. The locations of the stranded people are given by real numbers. An emergency rescue operation found n hotels to potentially accommodate the n people. Assume that each hotel has the capacity to shelter 1 person. However, due the prevailing road conditions access to these hotels are severely limited. For each hotel H, there is a specific segment of the freeway such that only people stranded in this segment can make it to H. Given as input the locations of the stranded people, and a set of n segments of the freeway (each segement corresponding to a particular hotel), your task is to design an efficient algorithm to decide if it is possible for all n people to find accommodation for the night. (Assume that the input contains real numbers of finite precision, so that any arithmetic operation on two real numbers takes constant time. Neither the list of people's locations nor the list of segments are assumed to be sorted in any particular order.)

(2) *(15 points)* Consider the following simplified model of how a law enforcement organization such as the FBI apprehends the members of an organized crime ring. The crime ring has n members, denoted by $x_1, \ldots, x_n$. A *law enforcement plan* is a sequence of n actions, each of which is either:

- apprehending a member x_j directly: this succeeds with probability q_j; or
- apprehending a member x_j using another member, x_i, as a decoy: this action can only be taken if x_i was already apprehended in a previous step. The probability of success is p_{ij}.

Let's assume that, if an attempt to apprehend x_j fails, then x_j will go into hiding in a country that doesn't allow extradition, and hence x_j can never be apprehended after a failed attempt. Therefore, a law enforcement plan is only considered *valid* if for each of the crime ring's n members, the plan contains only one attempt to apprehend him or her.

Assume we are given an input that specifies the number of members in the crime ring, n, and the probabilities q_j and p_{ij} for each $i \neq j$. You can assume these numbers are strictly positive and that $p_{ij} = p_{ji}$ for all $i \neq j$.

(2a) *(5 points)* Design an algorithm to compute a valid law enforcement plan that maximizes the probability of apprehending all of the crime ring's members, *in the order* $x_1, x_2, \ldots, x_n$. In other words, for this part of the problem you should assume that the j^{th} action in the sequence should be an attempt to apprehend x_j, and the only thing your algorithm needs to decide is whether to apprehend x_j directly or to use one of the earlier members as a decoy, and if so, which decoy to use.

(2b) *(10 points)* Design an algorithm to compute a valid law enforcement plan that maximizes the probability of apprehending all of the crime ring's members, *in any order*. In other words, for this

part of the question, your algorithm must decide on the order in which to apprehend the crime ring's members *and* the sequence of operations to use to apprehend them in that order.

(3) *(10 points)* In this problem you are asked to implement Kruskal's Minimum Spanning Tree Algorithm and Prim's Minimum Spanning Tree Algorithm. Your implementation should run in $O(M \log N)$ time, as explained on page 150 and page 157 of the textbook. Implement the algorithm in Java. The only libraries you are allowed to `import` are the ones in `java.util.*`. There is no `Framework.java` provided for this assignment. You will need to implement your Java program from scratch.

Warning: Be aware that the running time of calling a method of a built-in Java class is usually not constant-time, and take this into account when you think about the overall running time of your code. For instance, if you use a LinkedList, and use the indexOf method, this will take time linear in the number of elements in the list.

Your Java program should take in input from `stdin` and write output to `stdout`. You can use the Java class `Scanner` to read input from `System.in` and use `System.out.println` to write to `stdout`. An autograder will be online (**the URL will be released on Piazza shortly**) and come with options to upload and test your code on a small number of public test cases. When we grade the assignment, we will run it on a larger number of more complex test cases.

Your code should run in $O(M \log N)$ time. In particular, we impose a runtime limit of 2 seconds on all runs, so that an asymptotically slower algorithm will likely not be able to complete the larger test cases within the time limit. (In that situation, you will receive partial credit for the test cases that your implementation computes in time.)

Your algorithm is to read data, representing a graph with edge costs, from the standard input in the following format.

- The first line contains three integers separated by spaces, NMP, where $1 \leq N \leq 5000$ is the number of nodes, $1 \leq M \leq 25000000$ is the number of edges, and $P \in \{0, 1\}$. If $P = 0$, Kruskal's Algorithm should be used. If $P = 1$, Prim's Algorithm should be used.
- Each of the following M lines contains three integers XYC, where $1 \leq X < Y \leq N$ are two node IDs in increasing order, and $0 \leq C \leq 2^{31}/M$ is a cost. Every such line denotes the presence of an undirected edge between nodes X and Y of cost C.

Your algorithm should output data in the following format:

- $N - 1$ lines, containing one number E, where $1 \leq E \leq M$. Each line should encode one edge contained in the minimum spanning tree computed by your algorithm. Moreover, if $P = 0$, then the edges should appear in the order they are added to the spanning tree by Kruskal's Algorithm, so the first line contains the first edge added (i.e. the cheapest edge in the graph); if $P = 1$, then the edges should appear in the order they are added by Prim's Algorithm. (Ties should be broken by adding lower-numbered edges first.)

Example:

- Input:

```
3 3 0          # 3 nodes, 3 edges, use Kruskal's
1 2 100        # edge 1: cost({1,2}) = 100
2 3 200        # edge 2: cost({2,3}) = 200
1 3 150        # edge 3: cost({1,3}) = 150
```

- Expected Output:

```
1          # Kruskal's first selects edge 1
3          # Kruskal's then selects edge 3, completing the MST
```

We will impose a runtime limit of **2 seconds** on each instance. In particular, this means that for a maximally-sized instance (with $M = 25000000$), on a 2GHz machine, you only have 160 clock cycles per edge – an algorithm with complexity $O(MN)$ would not finish in time even if the constant hidden by big-O notation is 1. (We may slightly adjust this number or size of test cases if Java turns out to be too slow on the testing rig.)

We will use Java 8 for compiling and testing your program. More details on the build and testing environment will be made available on the autograder website once it comes online.

Remember: The problem asks you to implement both the Kruskal's and the Prim's MST algorithms. Use class `Scanner` to read input from `stdin` and use `System.out.println` to output to `stdout`. You may find the class `PriorityQueue` useful.