Comp 251: Assignment 2

Answers must be returned online by February 26th (11:59pm), 2017.

- Your solution must be returned electronically on MyCourse.
- Written answers and programming questions must be returned in two separate submission folders on MyCourse.
- The only format accepted for written answers are PDF or text files (i.e. .txt r .rtf). PDF files must open on SOCS computers. Any additional files (e.g. images) must be included in the PDF.
- Do not submit a compressed repository with all your files. Upload instead each PDF or text file individually.
- The solution of programming questions must be written in java. Your program should compile and execute of SOCS computers. Java files that do not compile or execute properly on SOCS computer will not be graded.
- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments the names of the persons with who you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators" at the beginning of your document. If asked, you should be able to orally explain your solution to a member of the course staff.
- Unless specified, all answers must be justified.
- When applicable, your pseudo-code should be commented and indented.
- The clarity and presentation of your answers is part of the grading. Be neat!
- Violation of all rules above may result in penalties or even absence of grading (Please, refer to the course webpage for a full description of the policy).
- Partial answers will receive credits.
- The course staff will answer questions about the assignment during office hours or in the online forum at `https://osqa.cs.mcgill.ca/`. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time.

Exercise 1 (Disjoint sets (20 points)) We want to implement a disjoint set data structure with union and find operations. The template for this program is available on the course website and named `DisjointSets.java`.
In this question, we model a partition of n elements with distinct integers ranging from 0 to $n-1$ (i.e. each element is represented by an integer in $[0, \cdots, n-1]$, and each integer in $[0, \cdots, n-1]$ represent one element). We choose to represent the disjoint sets with trees, and to implement the forest of trees with an array named `par`. More precisely, the value stored in `par[i]` is parent of the element `i`, and `par[i]==i` when `i` is the root of the tree and thus the representative of the disjoint set.
In addition, we also choose to implement the *path compression* technique seen in class.
Download the file `DisjointSets.java`, and complete the methods `find(int i)` as well as

`union(int i, int j)`. The constructor takes one argument n (a strictly positive integer) that indicates the number of elements in the partition, and initialize it by assigning a separate set to each element. The method `find(int i)` will return the representative of the disjoint set that contains `i` (do not forget to implement path compression here!). The method `union(int i, int j)` will merge the set containing `i` in the disjoint set containing `j` (i.e. the root of the tree containing `i` will become a child of the root of the tree containing `j`), and return the representative (as an integer) of the new merged set.
Once completed, compile and run the file `DisjointSets.java`. It should produce the output available in the file `unionfind.txt` available on the course website.
Note: You will need to complete this question to implement Question 2.

Exercise 2 (Minimum Spanning trees (40 points)) We will implement the Kruskal algorithm to calculate the minimum spanning tree (MST) of a undirected weighted graph. Here, you will use the file `DisjointSets.java` completed in the previous question, and two other files `WGraph.java`, `Kruskal.java` available on the course website. You will need the classes `DisjointSets` and `WGraph` to execute `Kruskal.java`. Your role will be to complete two methods in the template `Kruskal.java`.
The file `WGraph.java` implements two classes `WGraph` and `Edge`. An object of `Edge` stores all informations about edges (i.e. the two vertices and the weight of the edge), which are used to build graphs. The class `WGraph` has two constructors `WGraph()` and `WGraph(String file)`. The first one creates an empty graph and the second uses a file to initialize a graph. Graphs are encoded using the following format. The first line of this file is a single integer n that indicates the number of nodes in the graph. Each vertex is labelled with an number in $[0, \cdots, n-1]$, and each integer in $[0, \cdots, n-1]$ represents one and only one vertex. The following lines respect the syntax "n_1 n_2 w", where n_1 and n_2 are integers representing the nodes connected by an edge, and w the weight of this edge. n_1, n_2, and w must be separated by space(s). An example of such file can be found on the course website with the file `g1.txt`. These files will be used as an input in the program `Kruskal.java` to initialize the graphs. Thus, an example of a command line is `java Kruskal g1.txt`.
The class `WGraph` also provide a method `addEdge(Edge e)` that adds an edge to a graph (i.e. an object of this class). Another method `listOfEdgesSorted()` allows you to access the list of edges of a graph in increasing order of their weight.
You task will be to complete the two static methods `isSafe(DisjointSets p, Edge e)` and `kruskal(WGraph g)` in `Kruskal.java`. The method `isSafe` considers a partition of the nodes p and an edge e, and should return `True` if it is safe to add the edge e to the MST, and `False` otherwise. The method `kruskal` will take a graph object of the class `WGraph` as an input, and return another `WGraph` object which will be the MST of the input graph.
Once completed, compile all the java files and run the command line `java Kruskal g1.txt`. An example of the expected output is available in the file `mst1.txt`. You are invited to run other examples of your own to verify that your program is correct.

Exercise 3 (Greedy algorithms (15 points)) Describe a greedy algorithm that, given a set of points $S = \{x_1, x_2, \cdots, x_n\}$ on the real line (i.e. $x_i \in \mathbb{R}$), determines the smallest set of unit-length closed intervals that contains all of the given points.
For instance, if $S = \{0.8, 5.1, 0.5, 1.4\}$, then a solution would be $\{[0.8, 1.8], [5.1, 6.1]\}$.
Your algorithm must return an *optimal answer*. Indicate what is the greedy choice and the optimal substructure, and give an upper bound of the worst-case running time of your algorithm (i.e. using the big O notation).
Note 1: We do not ask you to provide a complete proof of correctness of your algorithm, but a *complete*

and *valid* proof will receive bonus points.
Note 2: We do not assume that the points are initially sorted.

Exercise 4 (Shortest paths (10 points)) Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Illustrate your answer.

Exercise 5 (Bipartite graphs (15 points)) Show that a graph is bipartite if and only if does not have an odd cycle. (Note the "if and only if". The proof needs to go both ways.)