Homework 7

Due by **12:30pm, Friday, November 11, 2016**.

Make sure you follow all the homework policies (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/policies/hw-policy.html).

All submissions should be done via Autolab (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/autolab.html).

Question 1 (Programming Assignment) [40 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

In this problem, we will explore weighted graphs.

We are given a starting node s and an ending node e, for some undirected graph G with n nodes. Further, each node u has its own weight, w_u `(0 <= w_u <= 50)`. The graph is formatted as an adjacency list, with the leading number being the node weight, meaning that for each node, u, we can access all of u's neighbors, as well as u's weight. The goal is to output the array of nodes in a shortest path from s to e. (The weight of a path $P = (u_1, \ldots, u_\ell)$ is $\sum_{i=1}^{\ell} w_{u_i}$ and the shortest path is the one with the minimum weight.)

Input

The input file is given with the first two lines being the start node and the end node, and the remainder of the file is an adjacency list with node weights for graph G (we assume that the set of vertices is $\{0, 1, \ldots, n-1\}$). The adjacency list assumes that the current node is the index of the line under consideration. For instance the third line of the input file has the list of all nodes adjacent to node 0 (the first two being the start and end nodes) and so on.

```
s   <- Staring node (some node between u0 and un-1).
e   <- Ending node (some node between u0 and un-1).
w0 u1 u4 u6      <- All nodes that share edges with u0
w1 u3 uu         <- All nodes that share edges with u1
w2 u0            <- All nodes that share edges with u2
.
.
.
wn-1 u0 u4 u2 u7  <- All nodes that share edges with un-1
```

Output

The output should be a list/ArrayList/vector (depending on your language of choice) where every 2 adjacent nodes have an edge between them. The 1st node in the output should be startNode and the last node in the output should be endNode. For example, if startNode was 2 and endNode was 7, and the minimum path between them is 2-9-8-4-7 then you should output [2, 9, 8, 4, 7]. If there is no path between the start and end nodes, output an empty output.

```
[s, n0 n1 n2... nm, e]                  <- Each entry is a node on the shortest path between 's' and
```

</> Note

There is **no** guarantee that the graph is completely connected. In the situation where a node can not be reached from the starting node, you should return an empty array.

</> Note

For this problem there can be multiple shortest paths so it is possible that your solution (assuming it is correct) might not match the corresponding sample output.

Hint

The best possible algorithm for this problem that we are aware of runs in time $O((m+n)\log n)$ where m in the total number of edges and n is the number of vertices.

! Note

Both the input and output parsers in each of the three languages are already written for you.
Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java | Python | C++

Download Java Skeleton Code (HW7Java.zip)

Directory Structure

```
├── Driver.java
├── HW7_Student.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given two coding files: `Driver.java` and `HW7_Student.java`. `Driver.java` takes the input file, parses it and creates an instance of the class and prints result in output file. You only need to update the `HW7_Student.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Hint

For this problem you *might* find the Java PriorityQueue (https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) or the TreeSet (https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) data structure useful.

Method you need to write:

```
                                    /**
                                     * This method must be filled in by you. You may add other methods and subclasses
                                     * but they must remain within the HW7_Student_Solution class.
                                     */
 5.                                 public ArrayList outputPath(){
                                    /*
                                    * Find the smallest weighted path between _startNode and _endNode
                                    * The first number of graph's adjacency list is the weight of it's node
                                    */
10.
                                    return new ArrayList();
                                    }
```

The `HW7_Student` class has 3 instance variables:

- `_startNode` which is of type int and stores the starting node id.
- `_endNode` which is of type int and stores the ending node id.
- `graph` which is of type `HashMap<Integer, ArrayList<Integer>>` where the key is the node id and the value is an arraylist of adjacent nodes. The 0th index of each arraylist is the weight of the key node

The output should be an `ArrayList` where every 2 adjacent nodes have an edge between them. Also, the 1st node in the `ArrayList` should be `_startNode` and the last node in the `ArrayList` should be `_endNode`. For example, if startNode was 2 and endNode was 7, and the minimum path between them is 2-9-8-4-7 then you should output [2, 9, 8, 4, 7]. If there is no path between the start and end nodes, output an empty `ArrayList`.

Compiling and executing from command line:

Assuming you're in the same directory level as `Driver.java`. Run `javac Driver.java` to compile.

To execute your code on `input1.txt`, run `java Driver testcases/input1.txt`. The output array will be printed to `stdout`.

Submission

You only need to submit `HW7_Student.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (MSTs for special graphs) [45 points]

The Problem

In this problem, we will consider the special case of MST where there are only two possible edge costs. In particular, the input to the problem is an undirected connected graph $G = (V, E)$ where every edge cost satisfies $c_e \in \{5, 10\}$ (i.e. the edge costs are either 5 or 10 and no other value). Present an $O(m + n)$ time algorithm to compute an MST of such graphs.

Note

The above run time is better than any known MST algorithm for the general cost case. Thus, your algorithm really needs to exploit the fact that there are only two edge costs.

Hint

It *might* be useful to start from the $O(m + n)$ time algorithm for the case when $c_e = 5$ for every $e \in E$ and then try to extend the algorithm to the current two cost scenario. In the solution that I have in mind the *proof of correctness* of the algorithm goes via the proof of correctness of Kruskal's algorithm. *As usual, there can be more than one way of solving a problem, so feel free to ignore the hint if you have another way to solve the problem.*

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Algorithm idea:` `10` points.
2. `Algorithm details:` `10` points.
3. `Proof of correctness idea:` `10` points.
4. `Proof details:` `10` points.
5. `Runtime analysis:` `5` point.

> **! Note**
> **If you do not have labeled and separated out proof idea, algorithm idea, proof details, algorithm details and runtime analysis you will get a zero(0) irrespective of the technical correctness of your solution.**

Questions 3 (Shortest path in DAGs) [15 points]

The Problem

Design an $O(m+n)$ time algorithm for the following problem. Given a Directed Acyclic Graph (or DAG) G with arbitrary weights on the edges (i.e. they can be positive or negative) with n vertices and m edges, and given two vertices s and t, compute the shortest path from s to t in G. (You can assume that G is given to you in the adjacency list format.)

> **Note**
> In case you missed it, note that the weights can be negative and the run time of the algorithm needs to be a linear in the input size.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Algorithm idea:` `3` points.
2. `Algorithm details:` `3` points.
3. `Proof of correctness idea:` `3` points.
4. `Proof details:` `3` points.
5. `Runtime analysis:` `3` point.

> **! Note**
> **If you do not have labeled and separated out proof idea, algorithm idea, proof details, algorithm details and runtime analysis you will get a zero(0) irrespective of the technical correctness of your solution.**