Due: 28th of August 2017 at 11:59pm

COMP 2907 – Assignment 1

All submitted work must be done individually without consulting someone elses solutions in accordance with the Universitys Academic Dishonesty and Plagiarism policies.

IMPORTANT! Questions 1a–c, 2a–c and 3a–c should be submitted via Blackboard as pdf (no handwriting!). The implementation required for Questions 1d, 2d and 3d should be submitted via Ed.

Questions

1. **[20 pts]** We are given an undirected graph $G = (V, E)$ with n vertices and m edges, a positive (not necessarily unique) edge cost c_e for each edge in E. We are also given q pairs of vertices $Q = \{(u_1, v_1), \ldots, (u_q, v_q)\}$. Decide for each pair (u, v) in Q if there is a path between u and v in G.

   Your task is to design an algorithm that solves this problem in $O(q \cdot (m+n))$ time. Your solution must include:

   (a) Description of how your algorithm works ("in plain English"). [5 points]

   (b) Argue why your algorithm is correct. [5 points]

   (c) Prove an upper bound on the time complexity of your algorithm. [5 points]

   (d) Implement your algorithm (in Ed) and test it on the the following graph instances `Graph8, Graph250, Graph1000`. Each instance is given as a text file using the following format:

   ```
   n
   m
   vertexId vertexId weight
   vertexId vertexId weight
   ...
   vertexId vertexId weight
   q
   vertexId vertexId
   vertexId vertexId
   ...
   ```

For example, the following text describes the instance depicted in Fig. 1. Note that the vertices are numbered from 0 to $n-1$ and that the two queries are $(0,1)$ and $(0,2)$.

```
4
3
0 2 2
0 3 5
2 3 3
2
0 1
0 2
```

Your program should read input from the text file. Your program only needs to print out '1' (= yes, there is a path in G between u and v) or '0' (= no, there is no path connecting u and v in G) for each of the q query pairs in order. You should separate each 1/0 with a new line. A scaffold is provided for Python 3 for you, but if you would like to use any other language, simply delete a1.py and replace with your file, and edit build.sh and run.sh to compile (if required) and run your code respectively.

Your code will not be benchmarked or tested for time complexity, but for full points it must be able to run instances similar to "Graph1000", and each test will time out after 5 seconds. Your program will also be tested on several hidden test cases. [5 points]

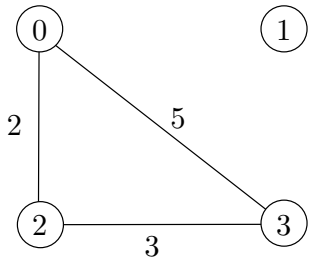

Figure 1: Illustrating the graph described in Question 1, with $n = 4$ and $m = 3$. If the query is $0, 1$ on the graph then the answer should be '0' while the answer to the query $0, 2$ on the same graph should be '1'.

2. **[30 pts]** Consider the following variant of the MST problem. We are given an undirected graph $G = (V, E)$ with n vertices and m edges, a positive (not necessarily unique) edge cost c_e for each edge in E, and a subset of edges $A \subset E$. Suppose that E represents a set of possible direct fibre links between vertices and A represents the existing set of direct fibre links

between vertices. We would like to find the cheapest way to connect all the vertices into one connected network.

The abstract problem we are interested in solving is to find a subset $X \subseteq E \setminus A$ of edges of minimum cost such that $(V, X \cup A)$ is connected.

Your task is to design an algorithm that solves this problem in $O(m \log n)$ time. Your solution must include:

(a) Description of how your algorithm works ("in plain English"). [5 points]

(b) Argue why your algorithm is correct. [10 points]

(c) Prove an upper bound on the time complexity of your algorithm. [5 points]

(d) Implement your algorithm (in Ed) and test it on the the following instances `Graph8, Graph250, Graph1000`. Each graph instance is given in a text file using the following format (where a is the number of edges in A):

```
n
m
vertexId vertexId weight
vertexId vertexId weight
...
a
vertexId vertexId
vertexId vertexId
```

For example, the following text describes the instance depicted in Fig. 2. Note that the vertices are numbered from 0 to $n - 1$.

```
4
5
0 2 2
0 1 6
0 3 5
1 3 8
2 3 3
2
0 2
1 3
```

Your program should read input from a text file, and calculate the cost of the spanning tree generated by your algorithm, that is, the cost of the edges in $X \cup A$. Your program does not need to return the actual edges, just print out the total cost rounded to **2 decimal places** to standard output.

Your code will not be benchmarked or tested for time complexity, but for full points it must be able to run instances similar to "Graph1000", and each test will time out after 5 seconds. Your program will also be tested on hidden test cases. [10 points]

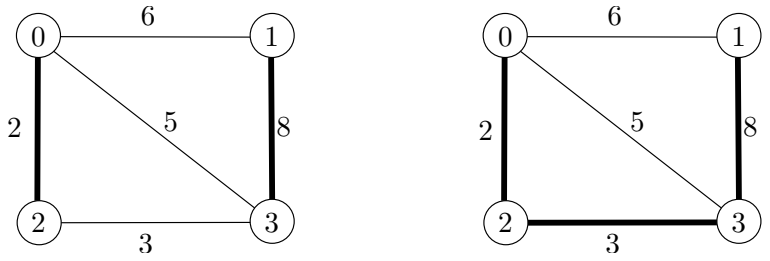

Figure 2: (left) Illustrating the graph described in Question 2, with $n = 4$, $m = 5$ and $a = 2$. (right) Showing an optimal solution having cost 13.

3. **[50 pts]** Consider the following extension of the problem in question 2. We are given an undirected graph $G = (V, E)$ with n vertices and m edges, a positive (not necessarily unique) edge cost c_e for each edge in E, a subset of edges $A \subset E$ and a subset of vertices $B \subset V$. Suppose that V represents a set of computers and B is the subset of these computers that have a single network port to connect to another computer. We would like to find the cheapest way to design a computer network that will connect all computers given the existing network (V, A).

   The abstract problem we are interested in solving is to find a subset $X \subseteq E \setminus A$ of edges of minimum cost such that $(V, X \cup A)$ is connected and $\deg_{X \cup A}(u) = 1$ for all $u \in B$ (that is, for every vertex u in B there is only one edge in $X \cup A$ that is incident to u).

   Your task is to design an algorithm that solves this problem in $O(m \log n)$ time. Your solution must include:

   (a) Description of how your algorithm works ("in plain English"). [10 points]

   (b) Argue why your algorithm is correct. [15 points]

   (c) Prove an upper bound on the time complexity of your algorithm. [5 points]

   (d) Implement your algorithm (in Ed) and test it on the the following instances `Graph8, Graph250, Graph1000`. Each instance is given in a text file as described in question 1.

   Each graph instance is given in a text file using the following format (where a is the number of edges in A and b is the number of vertices in B):

```
n
m
vertexId vertexId weight
vertexId vertexId weight
...
a
vertexId vertexId
vertexId vertexId
...
b
vertexId
vertexId
...
```

For example, the following text describes the instance depicted in Fig. 3. Note that the vertices are numbered from 0 to $n - 1$.

```
4
5
0 2
0 1 6
0 2 2
0 3 5
1 3 8
2 3 3
2
0 2
1 3
2
2
3
```

Your program should read input from a text file, and calculate the cost of the spanning tree generated by your algorithm, that is the cost of the edges in $X \cup A$. Your program does not need to return the actual edges, just print out the total weight rounded to **2 decimal places** to standard output.

Your code will not be benchmarked or tested for time complexity, but for full points it must be able to run instances similar to "Graph-Q3-1000", and each test will time out after 5 seconds. [20 points]

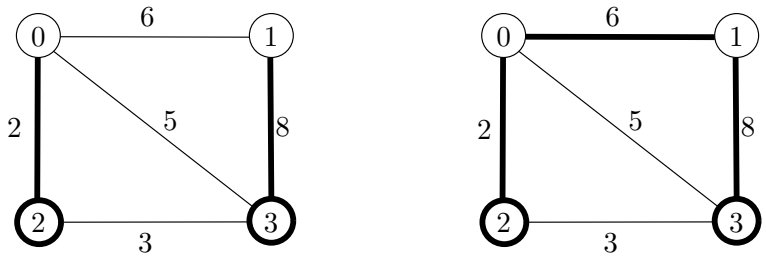

Figure 3: (left) Illustrating the graph described in Question 3, with $n = 4$, $m = 5$, $a = 2$ and $b = 2$. (right) Showing an optimal solution having cost 16.