Introduction to Algorithms	Problem Set 6
CS 4820, Spring 2019	Due 11:59pm Thursday, March 14

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)* You are trying to sabotage a flow network $G = (V, E)$ on n nodes and m edges by removing at most k edges from G to form a new flow network G'. Assume that every edge in G has unit capacity. Design an algorithm that given as input a flow network G and an integer $k > 0$ outputs a list of at most k edges to be removed from G such that value of the maximum flow of the resulting flow network G' is minimized. Your algorithm should run in time $O(mn)$.

(2) *(15 points)* m tourists are wandering in the streets of Manhattan, and they are hungry. Fortunately, there are m hotels as well, and you have been assigned the task to match tourists with hotels. We make the following assumptions:

- Model Manhattan as the $n \times n$ grid $[0, n-1] \times [0, n-1]$.
- Tourists can walk either horizontally or vertically on this grid. The time taken by a tourist to walk from (x_1, y_1) to (x_2, y_2) is given by $|x_1 - x_2| + |y_1 - y_2|$.
- Tourists can walk in parallel on a segment. (This models the fact that roads are wide enough for multiple people to walk simultaneously.)
- Each hotel can accommodate at most one tourist.
- Each tourist has a list of at most r hotels (which is a subset of the m available hotels) which they would be happy to go to.

You are given as input the initial locations of the m tourists, their hotel preferences, and the locations of the m hotels. Assume that time $T = 0$ initially. Define the cost of an allocation to be the minimum time T_0 such that all tourists reach their respective hotels (starting from their initial locations) at time $T = T_0$. Recall that we are assuming that the tourists walk simultaneously to their hotels.

Design an algorithm to find the minimum cost allocation that makes all the tourists happy (the algorithm should output 'No' if no such allocations exist). The running time of your algorithm should be $O(m^2 r \log n)$.

Example: Suppose there are 2 tourists a_1, a_2 located at $(0, 0)$ and $(1, 0)$ respectively. Further suppose there are 2 hotels h_1, h_2 located at $(0, 1)$ and $(1, 1)$ respectively. Let the lists of both a_1 and a_2 be $\{h_1, h_2\}$ and $r = 2$. Then, an assignment of a_1 to t_1 and a_2 to t_2 makes both of them happy. Further, under this allocation t_1 and t_2 reach their respective hotels at time $T = 1$. In the other assignment, which is a_1 to h_2 and a_2 to h_1 also makes both of them happy. However, they both reach their hotels at time $T = 2$, which is worse than the previous allocation. Thus, in this case your output should be (t_1, h_1) and (t_2, h_2).

(3) *(15 points)* In this problem, you are asked to implement the *Ford-Fulkerson algorithm* to solve the maximum flow problem, as described in sections 7.1 and 7.2 in the textbook. You should read these before getting started. For full credit, some parts of section 7.3 may also be of interest.

Implement the algorithm in Java. The only libraries you are allowed to `import` are the ones in `java.util.*, java.io.*`. There is no `Framework.java` provided for this assignment, because choosing the right data structures to hold the input data will be important. **Warning**: be aware of the space and time complexity of any data structures / methods you use from a built-in Java class, as these will count towards the complexity of your code.

Your Java program should take input from `stdin` and write output to `stdout`. We recommend using the `BufferedReader` and `BufferedWriter` classes, as these have much better performance than `Scanner` and `System.out.println`.

We will use an online autograder (`https://cs4820.cs.cornell.edu/`) that will allow you to upload your code and test it on a small number of public test cases. When we grade the assignment, we will run it on a larger number of more complex private test cases.

Your algorithm should read data from standard input in the following format:

- The first line contains two positive integers `n m` separated by a space, $2 \leq n \leq 10^3$, $0 \leq m \leq 10^5$, representing the number of nodes and number of edges in the directed graph, respectively.
- The next m lines each contain three numbers, `x y c`, separated by spaces, denoting a directed edge from the xth node to the yth node with capacity c. These numbers will satisfy $1 \leq x, y \leq n$ and $0 \leq c$.

We will take the first node (1) to be the source, and the second node (2) to be the sink. You will need to calculate a max flow between these vertices. Your algorithm should output data in the following format:

- The first (and only) line should contain a single integer `f`, $0 \leq f \leq 10^8$, representing the maximum flow that can be sent from node 1 to node 2 under the capacity constraints.

At least 80% **of all test cases will satisfy** $m \cdot f \leq 10^8$**. The remaining ones will be smaller: they will satisfy** $2 \leq n \leq$ **250,** $0 \leq m \leq$ **2000.**

Example (taken from these slides made by Cornell PhD alumnus Kevin Wayne; debugging using these slides may be useful):

- Input:

```
6 9             # 6 nodes, 9 edges
1 3 10          # edge from 1 to 3 with capacity 10
1 4 10          # edge from 1 to 4 with capacity 10
4 3 2           # edge from 4 to 3 with capacity 2
3 6 9           # edge from 3 to 6 with capacity 9
4 6 8           # edge from 4 to 6 with capacity 8
4 5 4           # edge from 4 to 5 with capacity 4
6 5 6           # edge from 6 to 5 with capacity 6
6 2 10          # edge from 6 to 2 with capacity 10
5 2 10          # edge from 5 to 2 with capacity 10
```

- Output:

```
19              # value of max flow
```

We will impose a runtime limit of **5 seconds** on each instance. You should reason about the runtime of your code on the possible test cases that satisfy the declared bounds on the size of the parameters. In particular, whether or not you can pass the up to 20% of test cases that do not satisfy $m \cdot f \leq 10^8$ may depend on choices that you make during the implementation.