Introduction to Algorithms **Problem Set 6**
CS 4820, Spring 2018 **Due 11:59pm Thursday, March 15**

Hand in your solution electronically using CMS. 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.**

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time.

(1) Computing a maximum flow in a network is a fairly time-consuming task, but various related problems have *linear-time* algorithms. Design linear-time algorithms for each of the following problems in a flow network with n vertices and m edges. Prove the correctness of your algorithm and prove that its running time is $O(m+n)$. In both problems, you are allowed to assume that the edge capacities are integers.

(1a) Max-flow detection. *(5 points)*
Given a flow network G and a flow f in G, decide whether f is a maximum flow.

(1b) Flow improvement. *(5 points)*
Given a flow network G, and a flow f_0 in G that is *not* a maximum flow, find another flow f_1 such that $v(f_1) > v(f_0)$.

HINT: Don't over-think this problem. If you understood the lectures on the Ford-Fulkerson algorithm, designing both algorithms should be almost trivial.

(2) *(10 points)* In a flow network, let us define an edge e to be *useless* if the relation $f(e) = 0$ is satisfied by **every maximum flow** f. Design a polynomial-time algorithm that takes a flow network G and a maximum flow $\bar{f}$ on this network, and outputs a list of all of its useless edges.

For full credit, your algorithm's running time should be $O(m^2)$ or faster.

(3) *(10 points)* Implement the Ford-Fulkerson algorithm in java using the environment provided on CMS. Use the framework code `FrameworkFlow.java` to read the input and write the output in a specific form. (This makes it easy for us to test your algorithm.) The only thing you need to implement is the algorithm, and you are restricted to implement this between the lines `//YOUR CODE STARTS HERE` and `//YOUR CODE ENDS HERE`. This is to make sure you can only use classes from `java.util` (imported at the start of the file), and the nested class `Edge`. Your implementation should run in $O(mC)$ time as discussed in lecture.

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.

You can test your code with the test cases provided on CMS. `FrameworkFlow.java` takes two command line arguments, the first is the name of the input file, and the second is the name of the output file.

The format of the input file is the following:

- The first three lines each contain one number: n, the number of nodes, s, the node number of the source, and t, the node number of the sink, respectively.
- The next n lines consist of the adjacency lists of the nodes and the capacity of the corresponding edges. To be precise: line $4+i$ corresponds to node i (the nodes are numbered 0 to $n1$), and lists a node j, and directly next to it the capacity of edge (i, j), for all nodes j reachable from i. For instance, if line 4 is `1 5 6 8`, then nodes 1 and 6 can be reached from node 0, and the capacity of arc (0, 1) is 5, while the capacity of arc (0, 6) is 8.

The output file is similar, except that the capacities are replaced by the flow value on the edge.

The code reads in the input, and stores the adjacency list (as objects of the class `Edge`) of node i as entry i of the array `adjacencyList`, i.e. `adjacencyList[i][k]` is the $(k+1)^{\text{st}}$ edge leaving node i. The nested class `Edge`, with fields `headNode`, `tailNode`, `capacity`, `flow`, `originalEdge` and `isForwardEdge` is given for your convenience, and you are free to use it or ignore (parts of) it. Your code should assign values that correspond to a maximum flow to the `flow` field of the edges.

We use Java 8 for compiling and testing your program.