CS 577: Introduction to Algorithms Fall 2020

Homework 7

Instructor: Dieter van Melkebeek TA: Bryan Jin

This homework covers network flow. **Problems 1 and 3 must be submitted for grading by 11:59pm on 11/02.** Problem 1 will be discussed in your discussion section on Friday. Please refer to the homework guidelines on Canvas for detailed instructions.

Warm-up problems

1. [Graded, 2 points] Suppose we are given an integer k, together with a flow network $N = (V, E, c)$ in which every edge has capacity 1. Design an algorithm to identify k edges in N such that after deleting those k edges, the maximum value of a flow in the remaining network is as small as possible. Your algorithm should run in time polynomial in n and m.

2. Given a flow network $N = (V, E, c)$ with source s and sink t, we say that a node $v \in V$ is *upstream* if, for all minimum s-t cuts (S, T) of G, $v \in S$. In other words, v lies on the s-side of every minimum s-t cut. Analogously, we say that v is *downstream* if $v \in T$ for every minimum s-t cut (S, T) of G. We call v *central* if it is neither upstream nor downstream.

   Design an algorithm that takes N and a flow f of maximum value in N, and classifies each of the nodes of N as being upstream, downstream, or central. Your algorithm should run in linear time.

Regular problems

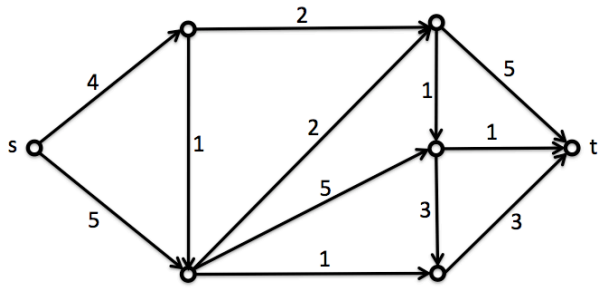
3. [Graded, 3 points] A given network G with integer capacities may have more than one minimum s-t cut. Define the *densest* minimum s-t cut to be any minimum s-t cut (S, T) of G with the greatest number of edges crossing from S to T.

   (a) Suppose we have access to a black box called MaxFlow. MaxFlow takes as input a network N and outputs a flow of maximum value for N. Design an algorithm to find a densest minimum s-t cut of G using at most one call to MaxFlow. Outside of MaxFlow, your algorithm should run in linear time. You may assume that standard arithmetic operations can be done in constant time.

   (b) Design an algorithm to determine whether G contains a unique densest minimum s-t cut. Once again, you can make at most one call to MaxFlow. Outside of MaxFlow, your algorithm should run in linear time. You may assume that standard arithmetic operations can be done in constant time.

4. Consider a network with integer capacities. An edge is called *upper-binding* if increasing its capacity by one unit increases the maximum flow value in the network. An edge is called *lower-binding* if reducing its capacity by one unit decreases the maximum flow value in the network.

(a) For the network G below determine a maximum flow f^*, the residual network G_{f^*}, and a minimum cut. Also identify all of the upper-binding edges and all of the lower-binding edges.



(b) Develop an algorithm for finding all the upper-binding edges in a network G when given G and a maximum flow f^* in G. Your algorithm should run in linear time.

(c) Develop an algorithm for finding all the lower-binding edges in a network G when given G and an integer maximum flow f^* in G. Your algorithm should run in time polynomial in n and m. Can you make it run in linear time?

5. A given network can have many minimum st-cuts.

(a) Determine precisely how large the number of minimum st-cuts in a graph can be as a function of n.

(b) Show that if (S_1, T_1) and (S_2, T_2) are both minimum st-cuts in a given network, then so is $(S_1 \cup S_2, T_1 \cap T_2)$. How does this generalize to more than 2 st-cuts?

(c) Design an algorithm that, given a network, generates a collection of minimum st-cuts $(S_1, T_1), (S_2, T_2), \ldots$ such that every minimum cut of the network can be written as

$$(\cup_{i \in I} S_i, \cap_{i \in I} T_i)$$

for some subset I of indices. Your algorithm should run in time polynomial in n and m.

Challenge problem

6. Problem J from the 2007 ACM-ICPC World Finals (see next page). Your algorithm should run in time polynomial in $n \doteq R + T$.

Programming problem

7. SPOJ problem Potholers (problem code POTHOLE).

Problem J

Tunnels

Input File: tunnels.in

Curses! A handsome spy has somehow escaped from your elaborate deathtrap, overpowered your guards, and stolen your secret world domination plans. Now he is running loose in your volcano base, risking your entire evil operation. He must be stopped before he escapes!

Fortunately, you are watching the spy's progress from your secret control room, and you have planned for just such an eventuality. Your base consists of a complicated network of rooms connected by non-intersecting tunnels. Every room has a closed-circuit camera in it (allowing you to track the spy wherever he goes), and every tunnel has a small explosive charge in it, powerful enough to permanently collapse it. The spy is too quick to be caught in a collapse, so you'll have to strategically collapse tunnels to prevent him from traveling from his initial room to the outside of your base.

Damage to your base will be expensive to repair, so you'd like to ruin as few tunnels as possible. Find a strategy that minimizes the number of tunnels you'll need to collapse, no matter how clever the spy is. To be safe, you'll have to assume that the spy knows all about your tunnel system. Your main advantage is the fact that you can collapse tunnels whenever you like, based on your observations as the spy moves through the tunnels.

Input

The input consists of several test cases. Each test case begins with a line containing integers R ($1 \le R \le 50$) and T ($1 \le T \le 1000$), which are the number of rooms and tunnels in your base respectively. Rooms are numbered from 1 to R. T lines follow, each with two integers x, y ($0 \le x,y \le R$), which are the room numbers on either end of a tunnel; a 0 indicates that the tunnel connects to the outside. More than one tunnel may connect a pair of rooms.

The spy always starts out in room 1. Input is terminated by a line containing two zeros.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the minimum number of tunnels that must be collapsed, in the worst case. Use the sample output format and print a blank line after each test case.

Sample Input	Output for the Sample Input
4 6 1 2 1 3 2 4 3 4 4 0 4 0 4 6 1 2 1 3 1 4 2 0 3 0 4 0 0 0	Case 1: 2 Case 2: 2