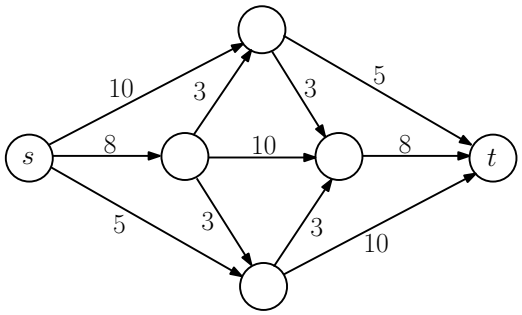
Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Hamid Ghodselahi December 12, 2012

Algorithm Theory, Winter Term 2012/13
Problem Set 5

hand in by Wednesday, January 9, 2013

Exercise 1: Maximum Flow Algorithms [10 Points]

Consider the following flow network:



(a) Solve the maximum flow problem on the above network by using the Ford Fulkerson algorithm. Give all intermediate results.

(b) Solve the maximum flow problem on the above network by using the preflow-push algorithm. Use the variant of the algorithm that always applies a push or relabel operation to a maximum height node with positive excess. Give all the push and relabel operations and describe in what way they change preflow or labeling.

(c) Give a minimum capacity s-t cut of the given network. Describe how you can get the cut from one of the maximum flows computed in (a) and (b).

Exercise 2: Forward-Only Paths [5 Points]

A friend of you has written some very fast maximum flow code. Unfortunately it turns out that the program doesn't always compute a correct maximum flow. When inspecting the solution you realize that your friend's program implements a simplified variant of the Ford-Fulkerson algorithm. When computing augmenting paths, the program only considers forward edges of the residual graph and it does not consider backward edges at all. We have seen in the lecture that backward edges are necessary to get a correct algorithm. However your friend claims that his algorithm (let's call it the forward-edge-only algorithm) always computes a solution that is within a constant factor of the optimal one. That is, there is an absolute constant $b > 1$ such that the forward-edge-only algorithm computes a flow of value at least $1/b$ times the value of an optimal flow. Is your friend right? If yes, prove it, otherwise show that the ratio of the maximum flow value and the flow computed by the forward-edge-only algorithm can be arbitrarily large. Assume that the forward-edge-only implementation always takes an arbitrary (possibly worst-case) augmenting path of only forward edges as long as such an augmenting path exists. You can also assume that all edge capacities are positive integers.

Exercise 3: [8 Points]

In the lecture we have seen that if the preflow-push algorithm always applies a push or relabel operation to a maximum height node with positive excess, the total number of push and relabel operations is at most $O(n^3)$. The goal of this exercise is to show that this algorithm can also be implemented to run in $O(n^3)$ time.

(a) First describe an algorithm (and a necessary simple data structure) that allows to always find the next maximum height node with positive excess in constant time.

Hint: First think about the possible maximum heights of a node with positive excess after applying a push or relabel operation at a node at height H.

(b) Once we have a maximum height node v with excess $e_f(v) > 0$, it remains to find an edge (v, w) in the residual graph G_f on which we can apply a push operation, or to find that no such edge exists so that we can relabel v. To do this efficiently, we maintain a linked list of all incident edges in the residual graph for every node v. For each edge in the list, we store its capacity and its flow value. Note that for every edge $e = (v, w)$ in the original graph, we have two copies, a forward copy at node v and a backward copy at node w. We also maintain pointers between the two copies so that they both can be updated efficiently when changing the flow on e. Each node keeps a pointer *current* to one of the edges in the list (initially, *current* points to the first edge on the list). If after a push operation, $e_f(v) = 0$, the pointer *current* is forwarded to the next edge on the list with positive residual capacity. If after a push operation, $e_f(v)$ remains positive, the pointer *current* is not moved. A relabel operation is performed whenever the pointer reaches the end of the edge list, before resetting it to the first element. Show that this gives a correct implementation of the algorithm. Further show that the number of list accesses of node v is at most $O(mn + k)$, where k is the number of saturating push operations at node v. Show that together with (a), this implies that the algorithm has running time $O(n^3)$.

Exercise 4: Bipartite Matching [7 Points]

We consider the Bipartite Matching Problem on a bipartite graph $G = (V, E)$. As usual, we assume that V is partitioned into sets X and Y, and each edge has one end in X and the other in Y.

If M is a matching in G, we say that a node $y \in Y$ is *covered* by M if y is an end of one of the edges in M.

(a) Consider the following problem. We are given G and a matching M in G. For a given number k, we want to decide if there is a matching M' in G so that

(i) M' has k more edges than M does, and

(ii) every node $y \in Y$ that is covered by M is also covered by M'.

We call this the *Coverage Expansion Problem*, with input G, M, and k and we will say that M' is a *solution* to the instance.

Give a polynomial-time algorithm that takes an instance of Coverage Expansion and either returns a solution M' or reports (correctly) that there is no solution. (You should include an analysis of the running time and a brief proof of why it is correct.)

Hint: You may wish to also look at part (*b*) to help in thinking about this.

Example. Consider Figure 1 below, and suppose M is the matching consisting of the edge (x_1, y_2). Suppose we are asked the above question with $k = 1$. Then the answer to this instance of Coverage Expansion is yes. We can let M' be the matching consisting (for example) of the two edges (x_1, y_2) and (x_2, y_4); M' has one more edge than M, and y_2 is still covered by M'.

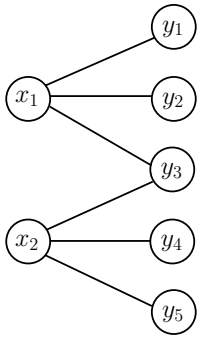


Figure 1: An instance of Coverage Expansion

(b) Give an example of an instance of Coverage Expansion, specified by G, M, and k, so that the following situation happens.

The instance has a solution; but in any solution M', the edges of M do not form a subset of the edges of M'.