Advanced Algorithms
Test 2
November 15, 2012

This is a take-home test. You have 43 hours to work on this test, from November 15 (Thursday) 17:00 until November 17 (Saturday) noon. This test is open-book, open-notes (including library and internet resources), but strictly individual—you cannot discuss any part of this test or communicate in any form about the problems and the material covered with anyone (physically present or not, known to you or purely anonymous) other than Prof. Moret and the TAs. (The prohibition on discussions includes any forum, such as `stackoverflow`*, where problems are "presented" and contributors pitch in. That is, your access to the net for this test is strictly read-only.)*

If you use any material from the library or the internet, you must cite your source *in the text at the place where you make use of it. (There is no need to cite the notes from the course; if you reuse a solution from a homework problem, simply mention the fact.)*

There are four problems, each worth 25 points. In all problems except the first, there are two questions, the first much simpler than the second; for those problems, the first question is worth 10pts, the second worth 15pts.

Problem 1. (Greedy Algorithms)

Recall that the subgraph G' of G induced by a subset V' of V is the graph $G' = (V', E')$, where E' is the subset of E containing exactly those edges with both endpoints in V'. You are given an undirected graph $G = (V, E)$ (as an array of adjacency lists) and a positive integer K and asked to find a maximum subset of vertices such that the subgraph of G induced by these vertices has degree at least K (i.e., every vertex in the induced subgraph has degree of at least K).

Consider the following greedy algorithm:

> repeatedly remove the remaining vertex of smallest (updated) degree, until all remaining vertices have degree at least K.

We use a priority queue to retrieve the remaining vertex of smallest degree; since removing a vertex decreases the degree of its neighbors, we use a Fibonacci heap to take advantage of its efficient `DecreaseKey` operation. Thus our algorithm begins by building a Fibonacci heap containing all vertices, using the degree of each vertex as the priority key. Our main loop then simply runs a `DeleteMin` operation on the heap. If the heap is empty, the algorithm stops and returns the empty set; otherwise, it tests the key (degree) of the returned vertex v against K. If the degree is at least K, the algorithm stops and returns the collection of vertices still in the heap. Otherwise, the algorithm updates the degrees of the neighbors of v (listed in the adjacency list of v) using `DecreaseKey` and begins the next iteration.

Prove the correctness of this algorithm and that it runs in $O(|E| + |V| \cdot \log(|V|))$ worst-case time.

Problem 2. (Minimum Spanning Trees)
Give a polynomial-time algorithm for each of the following problems—sketch a proof of their correctness and analyze their running time.

You are given an undirected graph $G = (V, E)$ with (not necessarily distinct) positive integral weights for the edges, and an edge $e_0 \in E$.

1. Decide whether *every* minimum spanning tree of G contains e_0.
2. Decide whether *some* minimum spanning tree of G contains e_0.

Problem 3. (Bipartite Matchings)
Give a polynomial-time algorithm for each of the following problems—sketch a proof of their correctness and analyze their running time.

You are given a bipartite graph $G = (V_1, V_2, E)$, $|V_1| = |V_2|$, and an edge $e_0 \in E$.

1. Decide whether *every* perfect matching of G contains e_0.
2. Decide whether *some* perfect matching of G contains e_0.

Problem 4. (Maximum Flow)
You are given an undirected network $N = (V, E)$ with integral edge capacities, a source, s, and a sink, t. Define N to be *k-stable* if and only if the value of the maximum s-t flow does not increase under any k-edge alteration. A k-edge alteration consists of picking k edges of the network and assigning them arbitrary new (positive integral) capacities.

1. Design an algorithm to test whether N is 1-stable; your algorithm should run in $O(|V|^2 \cdot |E|)$ time.
2. Design an algorithm to test whether N is 2-stable; your algorithm should also run in $O(|V|^2 \cdot |E|)$ time, although it is likely to involve significantly more work.