6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms

Problem Set 1

Collaboration policy: collaboration is *encouraged*. Here are more precise rules:

1. You must first think about the problems on your own,
2. You must write up your own solutions, independently,
3. You must record the name of every collaborator,
4. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration group limited to 3 or 4 people in a given week.

Problems:

1. Show a sequence of operations that can lead to a Fibonacci heap on n elements which is a chain.
2. Suppose that Fibonacci heaps were modified so that a node is marked after losing $k-1$ children (as opposed to just one child in the original setting), and thus a node is cut after losing k children.
   (a) Assuming that k is a constant, show that this will improve the amortized cost of decrease key (to a better constant) while it will increase the amortized cost of delete-min (by a constant factor). You may need to change the potential function for the analysis.
   (b) Could one take k growing with n and still have logarithmic ranks for every node?
3. Consider the Ford-Fulkerson augmenting path algorithm for the maximum flow problem seen in lecture, and assume that at every iteration the *shortest* (in terms of number of edges) augmenting path in the residual graph is selected (recall that the algorithm first finds the shortest path in the residual graph corresponding to the current flow, then pushes as much flow on it so as not to violate the capacity constraints, and then repeats until there is no augmenting path in the residual graph).

(a) Show that the length of the shortest path in the residual graph does not decrease from one iteration to the next.

(b) Show that the length of the shortest path in the residual graph must increase at least every m iterations, where m is the number of edges in the graph.

(c) Deduce that the total number of augmentations must be $O(nm)$ where n is the number of vertices.

(d) Deduce that the total running time of the algorithm is $O(nm^2)$.

4. Consider a maximum $s-t$ flow problem with *integer* capacities u. We saw in lecture that the Ford and Fulkerson generic algorithm could take $O(mU)$ iterations, where U is the maximum capacity of any edge, and this is not polynomial. In this problem, you will show that the algorithm can be made polynomial through a simple technique known as *scaling*.

   (a) Suppose we have a maximum flow f for our capacitated network, and we increase the capacity of some of the edges by 1 unit. f may not remain optimal. Prove that the maximum flow value increases by at most m, where m is the number of edges. What is the running time of Ford and Fulkerson if we start from our given flow f?

   (b) Let $k = \lfloor \log_2 U \rfloor$ and consider $k+1$ maximum flow instances. In the jth one where $j = 0, 1, \cdots, k$, the capacities $u(v, w)$ are replaced by

$$u_j(v, w) = \left\lfloor \frac{u(v, w)}{2^j} \right\rfloor .$$

   When $j = 0$, the problem is our original maximum flow instance. When $j = k$, all capacities are either 0 or 1. Describe how we can efficiently solve the maximum flow problem for the original capacities by solving the problem with the capacities u_j for j from k down to 1. What is the running time of the resulting algorithm?

5. Consider a capacitated network $G = (V, E)$. Let λ_{uv} denote the maximum flow value from u to v. Prove that

$$\lambda_{st} \geq \min(\lambda_{su}, \lambda_{ut})$$

   for any $s, t, u \in V$.

6. We have seen in lecture that the maximum cardinality matching problem in a bipartite graph $G = (V, E)$ with $V = A \cup B$ can be formulated as a maximum flow problem in a network H with vertex set $V \cup \{s, t\}$ and appropriately defined arcs and capacities. For any matching M in G, there is a corresponding $0-1$

flow f (i.e. with all (raw) flow values 0 or 1) in H with $|f| = |M|$, and vice versa.

Use the max-flow min-cut theorem to show that there exists a matching M in G of cardinality $|A|$ (i.e. every vertex in A is incident to an edge of M) if and only if for every $S \subseteq A$, we have $|N(S)| \geq |S|$ where $N(S) = \{v \in B : \exists u \in S$ with $(u, v) \in E\}$ is the set of neighbors of S (or *neighborhood*).

7. Consider the following 2-player game between Alice and Bob. We are given a bipartite graph $G = (V, E)$ where V is partitioned into $A \cup B$ (thus A and B are disjoint). Alice first chooses a vertex $a_1 \in A$. Then Bob selects a vertex $b_1 \in B$ which is adjacent to a_1. Alice now needs to select a vertex $a_2 \in A$ which is adjacent to b_1, and different from a_1. And we keep alternating between the 2 players. At any stage, a player needs to select a vertex which is (i) adjacent to the last vertex selected by its opponent and (ii) distinct from all previously selected vertices. A player loses as soon as he/she can't select such a vertex.

   For example, if the graph is a path $u - v - w - x$ on 4 vertices with $A = \{u, w\}$ and $B = \{v, x\}$ then Bob has a winning strategy. If Alice first plays u then Bob will play v (and then Alice w and Bob x), and if she plays w, he will play x. It is also easy to construct instances where Alice has a winning strategy (e.g. a complete bipartite graph with $|A| > |B|$).

   Design a polynomial-time algorithm which given a bipartite graph decides whether Alice or Bob has a winning strategy. And explain what the corresponding winning strategy (for that player) is.

8. Which question did you like the most (excluded this one...)? Which question did you like the least?