Advanced Topics in Algorithms
Computer Science 181b
Spring 2012
Homework 1
Due Wednesday, January 25

There are a number of places where you are asked to give proofs in this assignment. While you may be tempted to use mathematical induction, shorter and equally rigorous proofs are possible in every case. Each of these proofs can be done more simply and elegantly without induction. *For full credit, keep your solutions rigorous and concise.*

Please remember that the homework policy in this course does not permit consulting any external resources (books, websites, etc.). You may use your own lecture notes from previous courses and you may *talk* to other students in the class, the grutors, and Ran.

1. **[20 Points] Professor Lai's Greedy Matching Algorithms.** Professor I. Lai of the Pasadena Institute of Technology is giving a colloquium talk at Harvey Mudd.

   (a) Prof. Lai begins the lecture this way. "That algorithm that Prof. Ran showed in class for unweighted bipartite graphs is nice, but it's too complicated and too slow. I have a much simpler and faster greedy algorithm! The algorithm simply chooses an arbitrary edge and places it in the matching. Then, it chooses another edge that has no vertex in common with the previously chosen edge. It repeatedly chooses an arbitrary edge that has no vertices in common with any previously chosen edge until no such edge exists. Greed rocks!"

      i. Give a simple counterexample that shows that Prof. Lai's algorithm does not necessarily find a maximum matching.

      ii. Prove that Professor Lai's algorithm is guaranteed to find a matching whose size at least one half the optimal size! (That's neat - it's an approximation algorithm!)

      iii. Show that the ratio of one half is tight by describing how arbitrarily large graphs can be constructed such that Prof. Lai's algorithm may construct solutions whose size is exactly one half the maximum size.

      iv. Describe a fast implementation of Lai's algorithm and derive its asymptotic running time. Assume that the graph is represented as an adjacency list.

(b) Next, Prof. Lai says. "Aha! Very nice work clever Mudders. Now, let's consider weighted bipartite graphs (specifically, fully connected bipartite graphs $K_{n,n}$ with positive edge weights). The objective is to find a perfect matching of maximum weight. My algorithm sorts the edges by decreasing edge weight. Then, it chooses edges from this sorted list, one-by-one, adding an edge if it doesn't share a vertex with any previously chosen edge. I think that this algorithm will always find a matching whose total weight is at least half of the maximum weighted matching." State whether this claim is true or false and prove your claim.

2. **[15 Points] A Game on Graphs!** Here's a fun 2-player game played on a general undirected graph (i.e. the graph is not necessarily bipartite). Player 1 chooses some vertex v_1 and marks it. Player 2 then chooses an unmarked vertex v_2 adjacent to v_1 and marks it. Player 1 now chooses an unmarked vertex v_3 adjacent to v_2, etc. In other words, the two players alternate play such that the sequence of vertices $v_1, v_2, \ldots, v_k$ (chosen in that order) forms a simple (i.e. cycle-free) path in the graph. The last player able to select a vertex wins. Prove that the first player has a winning strategy if the graph has no perfect matching and the second player has a winning strategy if the graph has a perfect matching. As we'll see next week, there is a polynomial-time algorithm for finding a maximum matching in a general graph and thus this algorithm can determine whether or not a graph has a perfect matching. (For those of you who have taken Complexity Theory, this looks remarkably similar to the famous Generalized Geography game that is known to be PSPACE-complete. If you've seen that game in proof, you might ask yourself how we can reconcile the PSPACE-completeness of Generalized Geography with this result.)

3. **[15 Points] Bipartite Graph Matching.** Consider a bipartite graph with bipartition X, Y. We say that M is a matching of X *into* Y if M matches every vertex of X. In this problem, you will prove a famous result that states precisely when there exists a matching of X into Y. For any $S \subseteq X$, let $N(S)$ denote the set of neighbors of S. Your task is to prove that a bipartite graph has a matching of X into Y if and only if $|N(S)| \geq |S|$ for all $S \subseteq X$.

   One direction of this statement will be very easy to prove! There are a number of ways to prove the harder direction. The approach that I would like you to use here starts like this. Assume that $|N(S)| \geq |S|$ for all $S \subseteq X$ but that a maximum matching M in the graph leaves one or more vertices in X unmatched (or "unsaturated"). Then, we wish to construct a set $S \subseteq X$ such

that $|N(S)| < |S|$ in order to obtain a contradiction. To this end, let $x \in X$ be an unsaturated vertex with respect to M (also called "M-unsaturated"). Construct set S appropriately to obtain the desired contradiction.

4. **[20 Points] A Min-Max Theorem.** Recall that we mentioned in class that the size of a smallest vertex cover in a bipartite graph is equal to the size of a maximum matching in that graph. We proved this result using the Max Flow-Min Cut Theorem, but now you will prove it more directly using the theorem that you just proved above. Consider a bipartite graph with bipartition X, Y. Start by positing a minimum vertex cover C. Let $S = C \cap X$ and let $T = C \cap Y$. Split the graph into two disjoint bipartite graphs, one containing S (and some vertices in Y) and the other containing T (and some vertices in X). Now, show that there exists a matching in the first graph that matches all vertices in S and a matching in the second graph that matches all vertices in T. Use the Theorem above to do this. Conclude that there exists a matching whose size is equal to the size of the vertex cover.

5. **[30 Points] Fast Bipartite Matching.** Our first bipartite matching algorithm took $O(n(n+m))$ time: There were $O(n)$ iterations of depth-first search (to find an augmenting path) and each iteration took $O(n+m)$ time. In class we described a second approach that uses breadth-first search rather than depth-first search. This algorithm finds *many* disjoint augmenting paths simultaneously. In particular, at each iteration the algorithm finds the length of the shortest augmenting path and a *maximal* (but not necessarily maximum!) set of augmenting paths of that length. We then use these disjoint augmenting paths to construct a new matching and move on to the next iteration. Each iteration of the algorithm takes $O(n+m)$ time and we wish to show that after $O(\sqrt{n})$ iterations, the algorithm will terminate with a maximum matching, resulting in a total running time of $O(\sqrt{n}(n+m))$. We do this in a few steps.

   (a) Let M denote the matching at the beginning of one iteration of the algorithm and let M' denote the matching at the beginning of some later iteration of the algorithm. Show that there are exactly $|M'| - |M|$ vertex-disjoint odd length paths in $M \oplus M'$ and these paths are augmenting paths with respect to M.

   (b) Next, consider a matching M at a given iteration and a single augmenting path P of minimum length. As we saw in class, the matching that results from augmenting M by P is $M \oplus P$. Let P' denote a shortest augmenting path with respect to $M \oplus P$. Prove that $|P'| \geq |P| + 2|P \cap P'|$. That

is, the number of edges in P' is at least the number of edges in P plus twice the number of edges that P and P' have in common. To do this, Let $M' = (M \oplus P) \oplus P'$ denote the matching that results after augmenting M first by P and then augmenting that new new matching by P'. Now, argue the following:

i. $M \oplus M' = P \oplus P'$ (this is short and sweet).

ii. Among the paths in $P \oplus P'$ are at least two augmenting paths for matching M.

iii. $|P'| \geq |P| + 2|P \cap P'|$. .

(c) That's nice, but at a given iteration, the algorithm finds a maximal set of vertex disjoint shortest augmenting paths for the current matching M and augments M with all of these augmenting paths before it continues on to the next iteration. Use the result that you just proved to show that in the next iteration, the shortest augmenting path is at least 2 longer than the lengths of the augmenting paths that were used in this current iteration.

(d) Finally, prove that the algorithm terminates after $O(\sqrt{n})$ iterations. To do this, first perform exactly $\sqrt{n}$ iterations of the algorithm. Then show that the number of additional iterations required by the algorithm is $O(\sqrt{n})$ and conclude that the total number of iterations is therefore $O(\sqrt{n})$.