1 Instructions

This is due at the start of the class on October 16, 2013. Please write clearly and answer questions precisely. As a thumb rule, the answer should be limited to < 2 written pages, with ample spacing between lines and in margins, per question. Always start a new question on a new page, starting with Question 1, followed by Question 2, ..., Question n. Please cite all the references (including web-sites, names of friends, etc.) which you have used/consulted as the source of information for each of the questions. BTW, when a question asks you to design an algorithm - it requires you to (1) Clearly spell out the steps of your algorithm in pseudocode (2) Prove that your algorithm is correct and (3) Analyze the running time. By default a graph $G = (V, E)$ is simple, undirected and connected.

2 Problems

1. Consider the Proposal Algorithm that was discussed in the class (and is on the course-webpage as well). The input consists of a list of $n$ men $M$ and $n$ women $W$, where each man has an ordered list of $n$ women (a permutation) whom he will like to marry. Similarly, each woman has an ordered list (a permutation) of $n$ men whom she will like to marry. Output is a set of $n$ pairs $M$ forming a stable perfect matching, where each pair in $M$ consists of a man and a woman. Moreover, each man is present in a unique pair, and each woman is present in a unique pair in $M$. Prove that the output of this algorithm is always the same and is independent of in which order the ‘unmarried men’ are chosen in the while loop. That is, show that any execution of this algorithm will always result in the stable matching corresponding to $M = \langle m, \text{best}(m) \rangle$, $\forall m \in M$.

2. Suppose that there are more men than women, i.e. $|M| > |W|$. As before, each men has a permutation of women according to his preference list, and similarly each women has a permutation of men corresponding to her preference list. Define what will be the notion of stable matching in this case, and design an algorithm for this new definition of stable matching.

3. Running time of many divide-and-conquer algorithms are expressed by a recurrence relation. Consider the following recurrence relation

$$T(n) = T(xn) + T((1 - x)n) + cn$$
expressed in terms of $x$ and $n$ where $x$ is a constant in the range $0 < x < 1$ and $n$ is the size of the problem, typically very large. Note that $c > 0$ is a fixed constant. (BTW, $T(n)$ represents the time to solve the problem of size $n$. It is solved recursively by partitioning it in two subproblems, one of size $xn$ and other of size $(1-x)n$, and the time required to partition the problem and merging the solution of the subproblems is $cn$.) Is the asymptotic complexity the same when $x = 0.5, 0.1$ and $0.001$? What happens to the constants hidden in the $O()$ notation. (Some of you may be tempted to use Master’s Theorem directly - best is to ignore the statement of Master’s Theorem, and concentrate on its proof. Purpose of this exercise is to learn the proof of Master’s theorem, as well as remind you of recurrence relations.)

4. Let $T = (V, E)$ be a tree such that each vertex has degree at most 3. Let $n=|V|$. Show that $T$ has an edge (called a centroid edge) whose removal disconnects $T$ into two disjoint subtrees with no more than $(2n+1)/3$ vertices each. Give a linear time algorithm to find such an edge; prove its correctness. Next think of doing the same in a recursive manner. For each of the subtrees, obtained after deleting the centroid edge, recursively compute centroid edge. What will be the running time? Can you use the recurrence of the previous problem? BTW, can you think of some strategy to do this recursive partitioning in linear time? Also, can you imagine a nice way to represent (i.e. a data structure) elegantly to represent this recursive partitioning.

5. Assume that you are given $n$ positive integers, $d_1 \geq d_2 \geq \cdots \geq d_n > 0$. You need to design an algorithm to test whether these integers form the degrees of an $n$ vertex simple undirected graph $G = (V,E)$. You may think of a simple greedy algorithm. (If these integers represent degrees of a simple graph, then the sequence is called a degree sequence or graphical.) For example if the sequence is $(2,2,2)$, then it represents a cycle consisting of three vertices.

If you want to get more insight into degree sequences, then try to prove the following theorem of Erdős-Gallai.

A sequence $d_1 \geq d_2 \geq \cdots \geq d_n > 0$ of integers is graphical if and only if

(a) $\sum_{i=1}^{n} d_i$ is even.

(b) For all $1 \leq k \leq n$, $\sum_{i=1}^{k} d_i \leq (k-1)k + \sum_{j=k+1}^{n} \min(d_j, k)$.

6. Assume all edges in a weighted simple connected graph $G = (V, E)$ have distinct cost.

(a) Show that the edge with the maximum weight in any cycle in $G$ cannot be any minimum spanning tree of $G$.

(b) Consider an edge $e = (uv)$ which is not in a spanning tree $T$ of $G$. (Note that $T$ may not be a MST.) Consider the unique path, $\pi(u,v)$, between nodes $u$ and $v$ in $T$. Prove the following. If the weight of each edge on $\pi(u,v)$ is less than the weight of $e$ then $e$ cannot occur in any minimum spanning tree of $G$.

7. Let $G = (V, E)$ be a weighted simple connected graph, and assume that all edge weights are distinct. Define the weight of a spanning tree to be the sum total of the weights of edges in that tree. By definition, a minimum spanning tree $T$ of $G$ has the smallest sum total of the
weight among all possible spanning trees of $G$. Suppose we are not interested in minimizing the sum total of the weights, but just the weight of the heaviest edge in a spanning tree. Call such a tree a light spanning tree (LST). First show that any MST of $G$ is also a LST. Next show that a LST may not always be a MST. To compute LST, we can use an algorithm to compute MST and report that MST as a LST. You are asked to think of an alternate algorithm, running in $O(|V| + |E|)$ time, to find a LST. (Hint: Let $e_m$ be the edge with the median weight among edges in $G = (V, E)$. Consider the subgraph $G'$ formed by all edges in $E$, whose weight is at most the weight of $e_m$. Can you deduce something about LST from the connectivity of $G'$.)

8. Show that in a depth-first search of a graph $G$, if we output a left parenthesis ‘(’ when a node is accessed for the first time and output a right parenthesis ‘)’ when a node is accessed for the last time, then resulting parenthesization (or bracketing sequence) is proper. Each left ‘(’ is properly matched with each right ‘)’. (BTW, this property is basis for several graph algorithms, e.g. Euler Tour technique - which we will see during the course.)

9. Recall the algorithm for biconnectivity of a connected undirected simple graph $G$. Modify the DFS search procedure, so that it decomposes $G$ into biconnected components. A subgraph $G'$ of $G$ is a biconnected component, if $G'$ is biconnected, and is maximal in the sense that we cannot add any more vertex of $G$ to $G'$ and keep it still biconnected. (Alternatively, one can define the biconnected components using the equivalence classes.) Argue why your modified search procedure produces biconnected components.

10. Assume that $G = (V, E)$ is biconnected. Our task is to identify those edges $E' \subseteq E$, so that if we remove any edge $e \in E'$ from $G$, then the resulting graph is not biconnected. Intuitively, edges in $E'$ are essential in maintaining the biconnectivity of $G$. It is fairly straightforward to test whether an edge $e \in E$ is critical, by just removing $e$ from $G$ and running the biconnectivity algorithm to test whether the resulting graph is biconnected. A question worth trying is to compute the set $E' \subseteq E$ in $o(|E|(|V| + |E|))$ time. Even if you don’t succeed, list some of your strategies and state why you failed. (Hint: I wanted 10 problems - so I just cooked this up - I don’t know how to solve this though.)