CMPT 307 - Data Structures and Algorithms: Problem Set 1

Reminder: The homework assignment will **not** be collected and graded. However, the quiz given in class will be based on the material covered in the homework. If you solve the homework, you should be well-prepared for the quiz. You'll be able to check your solutions against the ones posted on the course webpage a week before the quiz.

1. **Asymptotics** Rank the following functions by order of growth; that is, arrange the following 7 functions $g_1, \ldots, g_7$ so that $g_1 = O(g_2), g_2 = O(g_3), \ldots, g_6 = O(g_7)$. Here are the functions: $2^{\sqrt{\log n}}$, 2^n, $n^{4/3}$, $n(\log n)^3$, $n^{\log n}$, 2^{2^n}, 2^{n^2}.

2. **Stable matchings** There are m hospitals, each with a certain number of available positions. There are n medical students looking for a position at a hospital. Each hospital has its ranking of the students in order of preference, and each student has the ranking of the hospitals in order of preference. Let us assume that the number n of students is greater than the total number of available positions (in all hospitals).

   We want an assignment where each student is assigned to at most one hospital so that all available positions in all hospitals are filled. Of course, since there are more students than positions, some students will end up without a position.

   We say that an assignment is *stable* if no instability of the following two types exists:

   - There is a hospital h and students s and s' so that: s is assigned to h; s' is not assigned to any hospital; and h prefers s' to s.
   - There are hospitals h and h', and students s and s' so that: s is assigned to h; s' is assigned to h'; h prefers s' to s; and s' prefers h to h'.

   Show that there is always a stable assignment of students to hospitals, and give an efficient algorithm to find one.

3. **Stable marriage** There are n men named $1, 2, \ldots, n$, and n women named $1', 2', \ldots, n'$. Suppose that all n men have the same preference list of women: $1', 2', \ldots, n'$, but women may have different preference lists of men. Prove that in this case there is only *one* stable matching. Describe this stable matching.

4. A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

5. Let $G = (V, E)$ be a connected graph, and let $u \in V$ be some vertex. Suppose we compute a DFS tree rooted at u, and obtain T that includes all nodes of G. Suppose then we compute a BFS tree rooted at u, and obtain the same tree T. Show that in this case $G = T$ (i.e., the graph G cannot contain any edges that are not in T).

6. **Counting shortest paths.** Write an algorithm for the following problem: Given an undirected graph $G = (V, E)$, and two nodes $s, t \in V$, find the *number* of shortest paths from s to t in G. (The algorithm should not list all the paths, but rather just compute the number of such paths.)

   The running time of your algorithm should be linear in the graph representation, i.e., $O(|V| + |E|)$. (Write an algorithm in pseudocode, argue its correctness, and analyze its running time.)

7. **Topological ordering** The algorithm for topological ordering of a given DAG that we saw in class repeatedly finds a node with no incoming edges and deletes it. It works if the input graph is indeed a DAG.

   But now suppose that the input graph may or may not be a DAG. Extend the topological ordering algorithm so that, given an input graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG; or (b) a cycle in G, thus establishing that G is not a DAG.

   The running time of your algorithm should be $O(m + n)$ for directed graphs with n nodes and m edges.

8. **Building Min-Heaps** Give a $O(n)$ time algorithm for building a Min-Heap, given an array of n numbers as input. (You need to present an algorithm in pseudo-code (as we do in class), prove that your algorithm is correct, and analyze its running time, proving that the algorithm always terminates within $O(n)$ steps.)