Data Structure & Algorithms
CS210/CS210A
Semester I, 2014-15, CSE, IIT Kanpur

Theoretical Assignment II

Deadline : 6:00 PM, 19th October

Note: Your solution for each problem must be formal and complete. The design of each algorithm or data structure must be provided in full details. You must provide analysis of time and space complexity of each algorithm and data structure. You should give suitable and precise arguments to justify your approach and/or prove the correctness of the algorithm (Do not write stories). All problems carry equal marks.

1 Splitting a red-black tree in $O(\log n)$ time

In the class we discussed an algorithm for *Special_merge*(T_1, T_2) on a red-black tree. Design an algorithm for *Split*(T, x) which splits T in to two red black trees T_1 and T_2 such that $T_1 < x < T_2$. Your algorithm must be based on *Special_merge*(T_1, T_2) such that it invokes *Special_merge* as a black box during its execution. Your algorithm must run in $O(\log n)$ time only where n is the number of nodes in T. (*You will lose at most 20% marks if your algorithm takes* $O(\log^2 n)$ *time.*)

2 Thinking beyond limits and beyond marks

There are problems which appear too difficult to be solved at the first glance. However, if you explore a problem closely and with open mind, you are likely to solve the problem, may be after many attempts.

In the class, we discussed the problem of articulation points in an undirected graph. To every person, who looks at this problem, at first the design of $O(m + n)$ time algorithm would appear too difficult a task. If you revisit the $O(m + n)$ time algorithm discussed today, you will find that it has too much to inspire all of us. The aim of this problem is to give every student (ordinary or extraordinary) a chance to experiment on his own and see whether what is said above has any truth.

There is a directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges. Each vertex v also stores a label $L(v)$ which is a real number. Let $min\_L(v)$ denote the label of the least label vertex reachable from v in the graph G.

1. Design an $O(m + n)$ time algorithm which receives graph $G = (V, E)$ and a vertex $s \in V$, and computes $\min\_L(s)$.

2. Suppose you wish to compute $\min\_L(v)$ for each vertex $v \in V$. Obviously, you can do it in $O(mn)$ time by repeating the algorithm of the first part n times. But can you do it in *close* to $O(m + n)$ time in total ?

   Surely, it sounds difficult. But yes, it is possible to design an algorithm which will run in $O(m + n \log n)$ time. Your task is to design this algorithm.

Note: There are many hurdles on your path to solve the above problem. For example, the problem is on directed graph but we discussed DFS/BFS traversal only for undirected graph. You might like to revisit these traversal algorithms to see whether there is really any difficulty in extending them to directed graphs. The problem is also giving you some hint. In particular, the expression $O(m + n \log n)$ is giving you a hint. Make best use of it.

3 Breadth of a special graph

There was a rooted tree T with n nodes. Somehow, the direction of its edges got lost and we have this *undirected* tree. Let G be the graph given in the form of adjacency lists corresponding to this undirected tree. It is easy to observe that there is a unique path between each pair of vertices in this graph G. The length of a path is defined as the number of edges on the path. We define breadth of this graph G (undirected tree) as the length of the longest path present in it. Design an $O(n)$ time algorithm to compute breadth of the tree.

Interestingly, this problem can be solved using BFS as well as DFS traversal. However, you must give the algorithm that is based on DFS traversal. Make use of *recursive* structure of the DFS traversal to solve this problem.

... Ponder over these problems with free mind ... you are welcome to discuss in case you get stuck ...