Algorithms Homework 3

University of Virginia

Gabriel Robins

Please make all algorithms as efficient as you can, and state their time and space complexities.

1-17. Solve the following problems from the [Cormen, 2nd Edition] algorithms textbook:

p. 161-163: 7-3, 7-4, 7-6
p. 170: 8.2-4
p. 173: 8.3-2, 8.3-4
p. 177: 8.4-2, 8.4-4
p. 178-179: 8-2, 8-3
p. 185: 9.1-1
p. 192-193: 9.3-1, 9.3-4, 9.3-5, 9.3-6, 9.3-8, 9.3-9

18. Give an algorithm that given a weighted graph, finds a spanning tree having the least possible <u>product</u> of its edge weights. Name a practical application of this problem.

19. True or false: if all edge weights of a graph are unique, then the MST is unique as well.

20. Give an algorithm for finding the next-to-minimum spanning tree of a weighted graph.

21. The shortest path between two nodes in a weighted graph may be not unique. Give an algorithm to find a shortest path between two nodes with a minimum <u>number</u> of edges.

22. Prove whether there exist a data structure where the operations INSERT, DELETE, and MIN each requires O(1) <u>worst-case</u> time each.

23. Does there exist a data structure where add/delete/find require O(1) expected-time and O(log n) worst-case time?

24. A "probe" at a pair of nodes A and B in a tree T determines whether all edges along the path in T from A to B are "intact" (e.g., we are looking for "open faults" in an electrical circuit).

    a) What is the minimum # of probes (in terms of the # of nodes & leaves of the tree) required to <u>completely</u> test all edges in a given tree?

    b) Give an algorithm that finds such a minimum set of probes for an arbitrary tree.

25. We would like to make a height-balanced binary search tree **<u>persistent</u>**, in the following sense. At the end of an arbitrarily long mixed series of node ADD and/or DELETE operations, the state of the tree after each individual operation is still explicitly represented. After N such arbitrary ADD and/or DELETE operations are performed (starting with an empty tree), within O(1) time we can obtain a pointer to the complete tree as it was right after the i^{th} operation, for any given i. Similarly, we need to support FIND queries on each of the N past versions of the tree, without asymptotic time penalty over normal tree searches. How can such a scheme be implemented efficiently, without asymptotically slowing down the worst-case ADD and DELETE times? What is the space penalty (in terms of N) required to implement this scheme?