CSCI 570 - HW 2

Rubric: This homework will have 100 points in total. Each question 25 points. It is more important that you give feedback, rather than the points. Please make sure to explain why you deduct points.

Solutions

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

**Solution**. Let C denote the number of nodes with two children, and L denote the number of leaves. We need to prove that C = L-1.

Base case. A tree with one node. The identity holds

Inductive Hypothesis. Assume that is true for a tree with *n*-1 nodes.

Inductive step. Prove it for a tree with *n* nodes.

We start with a tree of n-1 nodes and insert a new node. There are two cases to consider:

Case 1. the inserted node is only a child. The identity C = L-1 does not change

Case 2. the inserted node is the second child. In this case L++ and C++, so the identity holds.

Rubric: Base case (10 pts). Induction on #node of a tree (5 pts). Case 1 (5 pts). Case 2 (5 pts).

1. Suppose we perform a sequence of *n* operations on a data structure in which the *ith* operation costs *i* if *i* is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

**Solution.**

In a sequence of *n* operations there are 1+log *n* exact powers of 2, namely 1, 2, 4, ..., 2log *n* .

Summing them up yields 1 +2 + 4 + … + 2log *n* = 2·2log *n* – 1 ≤ 2*n.* Thus the total cost of all operations is T(*n*) ≤ 2*n* + *n* = 3*n*, which means O(3) amortized cost per operation.

Rubric: Sum of costs for i is power of 2 operations. (10 pts). Sum of costs for other operations (10 pts). Amortized cost. (5 pts) Note that O(3) = O(1).

1. When we have two sorted lists of numbers in non-descending order, and we need to merge them into one sorted list, we can simply compare the first two elements of the lists, extract the smaller one and attach it to the end of the new list, and repeat until one of the two original lists become empty, then we attach the remaining numbers to the end of the new list and it's done. This takes linear time. Now, try to give an algorithm using O(*n* log *k*) time to merge *k* sorted lists (you can also assume that they contain numbers in non-descending order) into one sorted list, where *n* is the total number of elements in **all** the input lists. Use a heap for *k*-way merging.

**Solution.** Construct a min-heap of the minimum elements from each of *k* lists. The creation of this min-heap will cost O(*k*) time. Next we run deleteMin and move the minimum element to the output array. Each extraction takes O(log *k*) time. Then insert into the heap the next element from the list from which the element was extracted. Since we extract *n* elements in total, the running time is O(*n* log *k* + *k*) = O(*n* log *k*).

Rubric:

1. Construct min-heap of the minimum elements from each of k lists. (5 pts)
2. Run deleteMin and move the minimum element to the output array. (5 pts)
3. Insert into the heap the next element from the list from which the element was extracted. (5 pts)
4. Complexity analysis: Construct heap (5 pts). DeleteMin and Insert (5 ptrs)
5. You are given a weighted graph G, two designated vertices *s* and *t*. Your goal is to find a path from *s* to *t* in which the minimum edge weight is maximized i.e. if there are two paths with weights 1015 and 273 then the second path is considered better since the minimum weight (2) is greater than the minimum weight of the first (1). Describe an efficient algorithm to solve this problem and show its complexity.

Solution.

1. find a path from s to t in which every edge weight is at least W.

Run BFS, ignoring any edges of weight less than W. This will take O(V + E) time. Or Remove all edges with weight less than W and run BFS from s.

2)

Sort all edges e1, e2, …., em. Then run the Part 1) algorithm with W = ek, testing for each W whether you can find a path in the graph using only edges of weight at least W. The largest W is the solution. How to choose W among e1, e2, …., em? We do it in a binary search fashion. Runtime O( (V + E) log E).

If we go through all edges, then O( (V + E) E).

Solution2. Greedy algorithm. Start with s in the set. At each step of the algorithm, find a vertex not in the set that has a maximum edge to the set. Add the vertex to the set. Runtime O( (V + E) log E)

Rubric:

1. Part 1 (10 pts)
2. Binary search in all edge weights. (10 pts)
3. Complexity analysis (5 pts)

If the student goes through all edges to find the maximum W with the right complexity analysis O( (V + E) E), he/she gets 20 pts.