Problem Set 8

Data Structures and Algorithms, Fall 2022

Due: November 3.

Problem 1

Nanjing has many tall buildings, but only some of them have a clear view of the Xuanwu Lake. Suppose we are given an array $A[1 \cdots n]$ that stores the height of n buildings on a city block, indexed from west to east. Building i has a good view of the Xuanwu Lake if and only if every building to the east of i is shorter than i. The following is an algorithm that computes which buildings have a good view of the Xuanwu Lake. What is the running time of this algorithm? You need to give an asymptotic tight bound, and you need to prove your answer. *(Hint: Ordered stacks.)*

GOODVIEW$(A[1 \cdots n])$

```
1: Initialize a stack S.
2: for (i ← 1 to n) do
3:     while (S is not empty and A[i] > A[PEEK(S)]) do
4:         POP(S).
5:     PUSH(S, i).
6: return S.
```

Problem 2

To conserve space for a stack, it is proposed to shrink it when its size is some fraction of the number of allocated cells. This supplements the array-doubling strategy for growing it. Assume we stay with the policy that the array size is doubled whenever the stack size grows beyond the current array size. Evaluate each of the following proposed shrinking policies, using amortized costs if possible. Do they offer constant amortized time per stack operation (i.e., push and pop)? You need to prove your answer. (The current array size is denoted as N.)

(a) If a pop results in fewer than $N/2$ stack elements, reduce the array to $N/2$ cells.

(b) If a pop results in fewer than $N/4$ stack elements, reduce the array to $N/4$ cells.

(c) If a pop results in fewer than $N/4$ stack elements, reduce the array to $N/2$ cells.

Problem 3

Consider an ordinary binary search tree augmented by adding to each node x the attribute $x.size$ giving the number of keys stored in the subtree rooted at x. Let α be a constant in the range $1/2 \leq \alpha < 1$. We say that a given node x is α-balanced if $x.left.size \leq \alpha \cdot x.size$ and $x.right.size \leq \alpha \cdot x.size$. The tree as a whole is α-balanced if every node in the tree is α-balanced. The following amortized approach to maintaining weight-balanced trees was suggested by G. Varghese.

(a) A $1/2$-balanced tree is, in a sense, as balanced as it can be. Given a node x in an arbitrary binary search tree, show how to rebuild the subtree rooted at x so that it becomes $1/2$-balanced. Your algorithm should run in time $O(x.size)$, and it can use $O(x.size)$ auxiliary storage.

(b) Show that performing a search in an n-node α-balanced binary search tree takes $O(\lg n)$ time.

For the remainder of this problem, assume that the constant α is strictly greater than $1/2$. Suppose that we implement INSERT and DELETE as usual for an n-node binary search tree, except that after every such operation, if any node in the tree is no longer α-balanced, then we "rebuild" the subtree rooted at the highest such node in the tree so that it becomes $1/2$-balanced. We shall analyze this rebuilding scheme using the potential method. For a node x in a binary search tree T, we define

$$\Delta(x) = |x.left.size - x.right.size| ,$$

and we define the potential of T as

$$\Phi(T) = c \cdot \left(\sum_{x \in T : \Delta(x) \geq 2} \Delta(x) \right) ,$$

where c is a sufficiently large constant that depends on α.

(c) Argue that any binary search tree has nonnegative potential and a $1/2$-balanced tree has potential 0.

(d) Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α (i.e., express c as a function of α) in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced? You need to justify your answer.

(e) Argue that inserting a node into or deleting a node from an n-node α-balanced tree costs $O(\lg n)$ amortized time.

Problem 4

Recall the linked-list representation of disjoint sets discussed in class. Professor F. Lake suspects that it might be possible to keep just one pointer in each set object, rather than two ($head$ and $tail$), while keeping the number of pointers in each list element at two. Show that the professor's suspicion is well founded by describing how to represent each set by a linked list such that each operation has the same running time as the operations described in class. Describe also how the operations work. Your scheme should allow for the weighted-union heuristic, with the same effect as described in class.

Problem 5

Recall the rooted-tree representation of disjoint sets discussed in class. Suppose that we wish to add the operation PRINTSET(x), which is given a node x and prints all the members of x's set, in any order. Show how we can add just a single attribute to each node in a disjoint-set forest so that PRINTSET(x) takes time linear in the number of members of x's set and the asymptotic running times of the other operations are unchanged. (You may assume that we can print each member of the set in $O(1)$ time.) You need to give the pseudocode of the PRINTSET(x) procedure.