XJTLU | Computing

CPT108: Data Structures and Algorithms

Semester 2, 2023-24

Tutorial 2

Stack and Queue

Problem 1. What is the output of the following program?

```
Queue q = new Queue();
Stack s = new Stack();
s.push(new Integer(5));
s.push(new Integer(6));
s.push(s.peek());
s.push(new Integer(7));
q.enqueue(s.pop());
q.enqueue(new Integer(5));
q.enqueue(new Integer(6));
System.out.println(q.peek());
s.push(q.dequeue());
System.out.println(s.pop());
s.pop();
System.out.println(s.pop());
```

A. 567

B. 765

C. 776

D. 766

E. None of the above

Problem 2. Consider the implementation of the Queue using an array. What would go wrong if we try to keep all the items at the front of a partially-filled array (so that data[0] is always the front of the queue)?

A. The constructor would require linear time

B. The `enqueue` method would require linear time

C. The `isEmpty` method would require linear time

D. The `dequeue` method would require linear time

E. The peek method would require linear time

Tree

Problem 3. What is the different between the binary search tree (BST) property and the min-heap property on page 163 (i.e., $A[\textsc{Parent}(i)] \leq A[i]$)? [Cormen 12.1-2]

Can the min-heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

Problem 4. Professor Kilmer claims to have discovered a remarkable property of binary search tree (BST). Suppose that the search for key k in a BST ends up at a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Kilmer claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest *counterexample* to the professor's claim. [Cormen 12.2-4]

Problem 5. Suppose that we have numbers between 1 and 100 in a binary search tree (BST) and want to search for the number 45. Which (possibly multiple) of the following sequences could be the sequence of nodes examined?

(a) $5, 2, 1, 10, 39, 34, 77, 63$

(b) $1, 2, 3, 4, 5, 6, 7, 8$

(c) $9, 8, 63, 0, 4, 3, 2, 1$

(d) $8, 7, 6, 5, 4, 3, 2, 1$

(e) $50, 25, 26, 27, 40, 44, 42$

(f) $50, 25, 26, 27, 40, 44$

Problem 6. Suppose you have a set of numbers where you over some period of time do a constant number of insertions and a linear number of lookups of the maximum. To maintain your set of numbers, you can choose between using either a heap or a binary search tree (BST).

(a) Assume you are interested in minimizing the average total runtime over the time period. For each structure, what is the asymptotic upper bound on the average case runtime, and which of the two structures should you choose?

(b) Suppose you realize that you actually interested in the worst case. For each structure, what is the asymptotic upper bound on the worst case runtime, and which of the two structures should you chose now?

Each justification should be no longer than two sentences.

Solutions

1 (C)

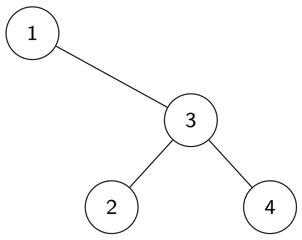
2 (D)

3 The BST property guarantees that all nodes in the left subtree are smaller, and respectively, and all nodes in the right subtree are larger, than the root. Whereas the min-heap property only guarantees that the child is larger than its parent, but does not distinguish between the relation between the left and right children. Therefore, the min-heap property cannot be used to print out the keys in sorted order in $O(n)$ time as we have no way to distinguish which subtree contains the smallest element.

4 When search for 4 in the BST on the right,

we have: $A = \{2\}$, $B = \{1, 3, 4\}$ and $C = \{\emptyset\}$.

Hence, Professor Kilmer's claim fails since $1 < 2$.



5 (a) NOT possible. The first element in the sequence must be the root element. When we compare the root key 5 with the sought key 45, we see that $5 < 45$, and so 45, if present in the tree, must be in the right subtree of the root. The second element in the sequence (2) should therefore be the root of the right subtree of the root node. But all keys in the right subtree are strictly greater than 5, and therefore 2 cannot be the key root of the right subtree.

(b) Possible. It is the sequence that is investigated in the degenerated tree consisting of elements $\{1, 2, 3, 4, 5, 6, 7, 8\}$, where no node has a left child.

(c) NOT possible. If we go to 8 from 9, then we are going in the wrong direction to find 45.

(d) NOT possible. Similar to the case above, it is not possible.

(e) NOT possible. All steps are in the right direction except the last one. From 44 we should check the right child, which has a key greater than 44. Instead we go to 42.

(f) Possible.

6 Let k be the number of insertions to be made. Let n be the number of elements in the set. By assumption there will be $\Theta(n)$ lookups of the maximum element.

(a) Insertion an element by bubbling a leaf up towards the root in a balanced tree is $O(\lg n)$. Insertion into a BST is done by traversing the tree from root to a leaf. In the average case, assuming that the tree to be somewhat balanced, the time consumption of traversal is proportional to the height of the tree, and so also $O(\lg n)$.

In order to look up the maximum element in a heap, we only have to look at the root element, which can be done in $O(1)$ time. To lookup the maximum element in a BST, we have to find the rightmost node in the tree (FIND-MAX). In a somewhat balanced tree, the length of such traversal is likely to be close to the height of the tree. So, the time consumption is $O(\lg n)$ time.

Hence, the total time consumption in the average case is $k \cdot O(\lg n) + n \cdot O(1) = O(n)$ when a heap is used; and $k \cdot O(\lg n) + n \cdot O(\lg n) = O(n \lg n)$ when a tree is used.

Heap	$O(n)$
Tree:	$O(n \lg n)$

For heap, the time consumptions is dominated by a linear number of lookups, each performed in $O(1)$ time. For the tree, the time consumptions is dominated by a linear number of lookups, each performed in $O(\lg n)$ time.

(b) For the heap, lookup is always done in constant time, and insertion is always a traversal from leaf to root in a balanced tree. So the time consumption remains bounded by $O(n)$ even in the worst case.

For the BST, we must consider the degenerate case when no node has a left child. Then, the tree is effectively a list rooted in the smallest element, descending rightwards through ever increasing elements towards the greatest element. In the worst case, we have to insert elements that are even greater than the elements already present in the tree. Then we have to traverse the entire tree, going through all $O(n)$ nodes before reaching the correct place to insert the new element. Such an insertion runs in time bounded by $O(n)$.

Lookup of the maximal element in such a degenerated tree is equally bad. Since the maximal element is kept in the rightmost node, in order to find it we have to traversal all $O(n)$ nodes again. This is done in $O(n)$ time.

So, the total time consumption is $O(n)$ when a heap is used, and $k \cdot O(n) + n \cdot O(n) = O(n^2)$ when a BST is used.

Heap	$O(n)$
Tree:	$O(n^2)$

The heap behaves asymptotically the same in the worst case as in the average case. In a tree where no node has a left child, both insertion and lookup of maximal element is done in $O(n)$ time.