Anthony Rusignuolo

CS 600WS – Advanced Algorithms

Reza Peyrovian

Homework 2

I pledge my honor that I have abided by the Stevens Honor System.

1. C-2.8 Describe the structure and pseudocode for an array-based implementation of an index-based list that achieves O(1) time for insertions and removals at index 0, as well as insertions and removals at the end of the list. Your implementation should also provide for a constant-time get method.
   1. This requires the functionality of stacks and queues. The structure would be an N element array S with elements stored from S[first] to S[last], where N is the maximum array size, first is the index indicating the first or bottom element in S, and last is the index indicating the last element in S. Both first and last are initially set to 0 for an empty stackQueue. There also needs to be a way of handling errors for indicating when the stack is full or empty. This is achieved in the methods of the object’s class.

class stackQueue() {

N = maximum array size  
this = array(N)

this.first = 0

this.last = 0

method increment(variable) {

if variable + 1 = N

variable = 0

else

variable = variable + 1

}

method decrement(variable) {

if variable – 1 < 0

variable = N – 1

else

variable = variable - 1

}

method length() {

if this.last < this.first

return N + 1 - this.first + this.last

return this.last – this.first + 1

}

method insertEnd(o) {

increment this.last

if this.last == this.first

decrement(this.last)

return SQFullError

else

this[t] = o

return

}

method removalEnd() {

if this.last == this.first

return SQEmpty Error

else

temp = this[this.last]

this[this.last] = null

decrement(this.last)

return temp

}

method insertBeginning(o) {

if length(this) == N

return SQFullError

else

decrement(this.first)

this[this.first] = o

return

}

method removalBeginning() {

if this.first == this.last

return SQEmptyError

else

temp = this[this.first]

this[this.first] = null

increment(this.first)

return temp

}

method get(index) {

if index >= N

return indexError

else

return this[((this.first + index) % N)]

}

1. C-2.20 Let T be a binary tree with n nodes. Give a linear-time method that uses the methods of the BinaryTree interface to traverse the nodes of T by increasing values of the level numbering function p given in Exercise R-2.8. This traversal is known as the level order traversal.
   1. levelOrderTraversal(BinaryTree T): {  
       q = new Queue;  
       q.enqueue(T.root());  
       while q: {  
       node v = q.dequeue();  
       if T.isInternal() {  
       q.enqueue(v.leftChild());  
       q.enqueue(v.rightChild());
2. A-2.2 Suppose you work for a company, iPuritan.com, that has strict rules for when two employees, x and y, may date one another, requiring approval from their lowest-level common supervisor. The employees at iPuritan.com are organized in a tree, T, such that each node in T corresponds to an employee and each employee, z, is considered a supervisor for all of the employees in the subtree of T rooted at z (including z itself). The lowest-level common supervisor for x and y is the employee lowest in the organizational chart, T, that is a supervisor for both x and y. Thus, to ﬁnd a lowest-level common supervisor for the two employees, x and y, you need to ﬁnd the lowest common ancestor (LCA) between the two nodes for x and y, which is the lowest node in T that has both x and y as descendants (where we allow a node to be a descendant of itself). Given the nodes corresponding to the two employees x and y, describe an efﬁcient algorithm for ﬁnding the supervisor who may approve whether x and y may date each other, that is, the LCA of x and y in T. What is the running time of your method?
   1. lowestCommonAncestor(node x, node y): {  
       parentX = parent(x)  
       parentY = parent(y)  
       if parentX == null or x == parentY  
       return x  
       if parentY == null or y == parentX  
       return y  
       if parentX == parentY  
       return parentX  
       X = LCA(parentX, y)  
       Y = LCA(x, parentY)  
       if isRoot(X)  
       return Y  
       else  
       return X  
        
      Run time is O(h2)
3. R-3.6 Give a pseudocode description of an algorithm to ﬁnd the element with smallest key in a binary search tree. What is the running time of your method?
   1. Assuming this is an ordered binary tree  
      smallestKey(root)  
       min = root  
       while min.left  
       min = min.left  
       return min  
        
      Run time is O(h)
4. C-3.3 Describe how to perform the operation ﬁndAllElements(k), which returns every element with a key equal to k (allowing for duplicates) in an ordered set of n key-value pairs stored in an ordered array, and show that it runs in time O(log n+s),where s is the number of elements returned.
   1. high = highest value  
      low = lowest value  
      s = 0  
      findAllElements(high, low, k)  
       mid = midpoint between high and low  
       if mid > k  
       FAE(high, mid, k)  
       if mid < k  
       FAE(mid, low, k)  
       if mid = k  
       s++  
       FAE(high-2, low, k)  
       FAE(high, low+2, k)
5. A-3.2 Imagine that you work for a database company, which has a popular system for maintaining sorted sets. After a negative review in an inﬂuential technology web-site, the company has decided it needs to convert all of its indexing software from using sorted arrays to an indexing strategy based on using binary search trees, so as to be able to support insertions and deletions more efﬁciently. Your job is to write a program that can take a sorted array, A, of n elements, and construct a binary search tree, T, storing these same elements, so that doing a binary search for any element in T will run in O(log n) time. Describe an O(n)-time algorithm for doing this conversion.
   1. arrayToBST(array) {  
       mid = midpoint of array  
       leftArray = array[0 to mid-1] //includes mid-1  
       rightArray = array[mid+1 to -1]  
       helper(root, leftArray, rightArray){  
       if leftArray.length == 1  
       left = new node(array[0])  
       else  
       left = arrayToBST(leftArray)  
       if rightArray.length == 1  
       right = new node(array[0])  
       else  
       right = arrayToBST(rightArray)  
       left.parent = root  
       right.parent = root  
       root.child(left)  
       root.child(right)  
       return root  
       return helper(root, leftArray, rightArray)
6. R-4.4 A certain Professor Amongus claims that the order in which a ﬁxed set of elements is inserted into an AVL tree does not matter—the same tree results every time. Give a small example that proves Professor Amongus wrong.
   1. taking the values 1, 2, 3, 4 you can generate the following AVL trees  
       2  
       / \  
       1 3  
       \  
       4  
        
       3  
       / \  
       2 1  
       /  
       4  
      which are clearly different
7. R-4.7 What is the minimum number of nodes in a red-black tree of height 8?
8. A-4.4 Suppose you are working for a victim-support group to build a website for maintaining a set, S, containing the names of all the registered sex offenders in a given area. The system should be able to list out the names of the people in S ordered by their Zip codes, and, within each Zip code, ordered alphabetically. It should also be able to list out the names of the people in S just for a given Zip code. The running time for a full listing should be O(n), where n is the number of people in S, and the running time for a listing for a given Zip code should be O(log n+s),where s is the number of names returned. Insertions and removals from S should run in O(log n) time. Describe a scheme for achieving these bounds.