Chapter-9: C-9.26

R-8.1 The following questions refer to the tree of Figure 8.3.

a. Which node is the root?

Root =/user/rt/courses/

b. What are the internal nodes?

/user/rt/courses/

cs016/

cs252/

homeworks/

programs/

projects/

papers/

demos/

c. How many descendants does node cs016/ have?

Grades, homeworks/, programs/, hw1, hw2, hw3, pr1, pr2, pr3

d. How many ancestors does node cs016/ have?

1

e. What are the siblings of node homeworks/?

Grades, programs

f. Which nodes are in the subtree rooted at node projects/?

projects/, papers/, demos/, buylow, sellhigh, market

g. What is the depth of node papers/?

3

h. What is the height of the tree? 4

R-8.4 What is the running time of a call to T. height2(p) when called on a position p distinct from the root of T? (See Code Fragment 8.5.)

O(n-1) = O(n)

R-8.4 What is the running time of a call to T. height2(p) when called on a position p distinct from the root of T? (See Code Fragment 8.5.)

- / \* + 3 1 3 + - 9 5 2 + \* 3 - 7 4 6 [Count total we have 19 nodes]

3 1 + 3 \* 9 5 - 2 + / 3 7 4 - \* 6 + - [Count total we have 19 nodes]

C-8.42 Describe how to clone a LinkedBinaryTree instance representing a (not necessarily proper) binary tree, with use of the add left and add right methods.

Assuming root has left, right pointer and data field.

CLONE(root)

if root == NULL then return NULL

new\_root = new LinkedBinaryTree(root.data) //create new node for root

new\_root.\_add\_left(CLONE(root.left)) //recursively clone left subtree and add it as left to new root

new\_root.\_add\_right(CLONE(root.right)) //recursively clone right subtree and add it as right to new root

return new\_root //return new\_root

R-9.3 What does each remove min call return within the following sequence of priority queue ADT methods: add(5,A), add(4,B), add(7,F), add(1,D), remove min( ), add(3,J), add(6,L), remove min( ), remove min( ), add(8,G), remove min( ), add(2,H), remove min( ), remove min( )?

(1,D), (3, J), (4,B), (5,A), (2,H), (6,L)

R-9.5 The min method for the UnsortedPriorityQueue class executes in O(n) time, as analyzed in Table 9.2. Give a simple modification to the class so that min runs in O(1) time. Explain any necessary modifications to other methods of the class.

public interface Entry {

K getKey( );

V getValue( );

}

public Entry insert(K key, V value) throws IllegalArgumentException {

checkKey(key); // auxiliary key-checking method (could throw exception)

Entry newest = new PQEntry<>(key, value);

list.addLast(newest);

return newest;

public Entry min( ) { 30 if (list.isEmpty( )) return null;

return findMin( ).getElement( );

}

public Entry removeMin( ) {

if (list.isEmpty( )) return null;

return list.remove(findMin( ));

}

R-9.21 Show all the steps of the algorithm for removing the entry (16,X) from the heap of Figure 9.1, assuming the entry had been identified with a locator.

Step 1:Delete a node from the array

Step 2: Replace the deletion node with the "farthest right node" on the lowest level of the Binary Tree

Step 3: value in replacement node < its parent node so have to replace with parent

C-9.26 Show how to implement the stack ADT using only a priority queue and one additional integer instance variable.

*structure Stack:*

priorityQueue : MinPriorityQueue<Item>

topPriority : integer

*procedure initialize(stack : Stack):*

stack.priorityQueue

stack.topPriority

*procedure push(stack : Stack, item : Item):*

stack.priorityQueue.insert(topPriority, item)

topPriority ← topPriority - 1

*procedure pop(stack : Stack):*

topPriority ← (topPriority + 1)

poppedItem ← stack.priorityQueue.removeMin()

return poppedItem