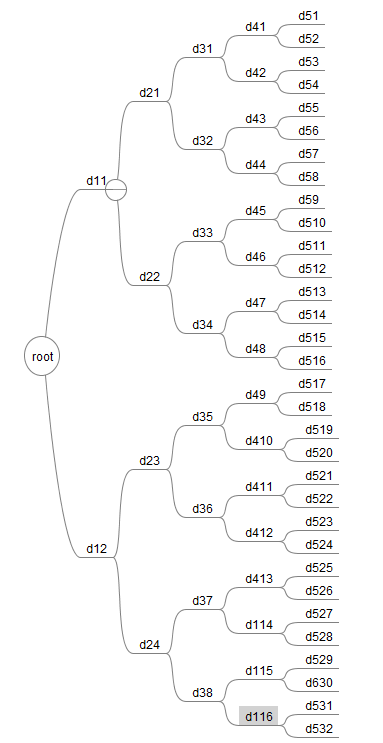
Assignment 3

R-2.7 Let T be a binary tree with n nodes that is implemented with a vector, S, and let p be the level numbering of the nodes in T, as given in Section 2.3.4. Give pseudo-code descriptions of each of the methods root, parent, leftChild, rightChild, isInternal, isExternal, and isRoot.

|  |
| --- |
| Algorithm root()  Output: position of the root of the tree T in S  return 1 |
| Algorithm parent(p)  Input : position p of a node in the vector S  Output: position of the root of the tree T in S  return p / 2 |
| isRoot(v)  Input: node v in the vector S  Output: true of false whether the node is root node of the tree T in S  if S.elementAtRank(1) = v do  return true  return false |
| isExternal(p)  Input: position p of a node in the vector S  Output: true of false whether the node is external node of the tree T in S  if p > 1 /\ p < S.size() then  if p\*2 >= S.size() then  return true  return false |
| isInternal(p)  Input: position p of a node in the vector S  Output: true of false whether the node is internal node of the tree T in S  if not isExternal(p) then  return true  return false |
| leftChild(p)  Input : position p of a node in the vector S  Output: position of the left child of the tree T in S  if isInternal(p) then  position <- p\*2  if position < S.size() then  return position  return null |
| rightChild(p)  Input : position p of a node in the vector S  Output: position of the right child of the tree T in S  if leftChild(p) <> null then  position <- leftChild(p) + 1  if position < S.size() then  return position  return null |

R-2.8 Answer the following questions so as to justify Theorem 2.8.

a. Draw a binary tree with height 5 and with the maximum number of external nodes.



b. What is the minimum number of external nodes for a binary tree with height h? Justify your answer.

We have: i >= h

e = i + 1 🡪 I = e – 1

So i = e - 1 >= h 🡪 e >= h + 1

Therefore, the minimum external nodes is h + 1

c. What is the maximum number of external nodes for a binary tree with height h? Justify your answer.

We have: i <= 2h – 1 🡪 i+1 <= 2h 🡪 e <= 2h

Therefore, the maximum external nodes is 2h

d. Let T be a binary tree with height h and n nodes. Show that log(n+1) -1 < h < (n-1)/2

We have: n <= 2h+1 – 1 🡪 n+1 <= 2h+1  🡪 log(n+1) <= h + 1 🡪 h >= log(n+1) – 1

Besides, n >= 2h+1 🡪 n – 1 >= 2h 🡪 h <= (n-1)/2

e. For which values of n and h can the above lower and upper bounds on h be attained with equality?

From the formula above, we must find n value so that h = log(n+1) – 1 = (n – 1)/2

log(n+1) = (n – 1)/2 + 1 = n/2 + ½ = (n + 1)/2

Base on this log(n+1) = (n+1)/2, this equality just happens if only n = 1

C-2.2 Analyze your implementation of the queue ADT that used two stacks (from assignment 2). What is the amortized running time for dequeue and enqueue, assuming that the stacks support constant time push, pop, and size methods?

C-2.7 Using the Sequence ADT, describe an efficient way of putting a sequence representing a deck of n cards into random order. Use the function randomInt(n), which returns a random number between 0 and n-1, inclusive. Your method should guarantee that every possible ordering is equally likely. What is the running time of your method, if the sequence is implemented with an array? What if it is implemented with a linked list?

|  |  |  |
| --- | --- | --- |
| Algorithm cardDesk(D)  L <- initialize a vector  e <- false  s <- D.size()  while e = false do  c <- randomInt(size)  L.insertAtRank(i, D.elemAtRank(c))  D.removeAtRank(c)  s <- D.size()  if s == 0 then  e <- true    return L | **List**  O(1)  O(1)  O(1)  O(n)  O(n)  O(n\*n)  O(n\*n)  O(n\*n)  O(n\*n)  O(n\*n)  O(1)  The running time is: O(n\*n) | **Array**  O(1)  O(1)  O(1)  O(n)  O(n)  O(n\*n)  O(n\*n)  O(n)  O(n)  O(n)  O(1)  The running time is: O(n\*n) |