EE 2331 Data Structures and Algorithms, Semester B, 2009/10

Tutorial 10: Binary Trees

Week 10 (25th March, 2010)

The tasks of tutorial exercises are divided into three levels. Level 1 is the basic tasks. You should have enough knowledge to complete them after attending the lecture. Level 2 is the advanced tasks. You should able to tackle them after revision. Level 3 is the challenge tasks which may be out of the syllabus and is optional to answer. I expect you to complete task A in the tutorial.

Outcomes of this tutorial

1. Able to understand the implicit array representation of binary trees
2. Able to convert implicit array tree to linked list tree using recursion
3. Able to implement different tree traversals using recursion or queues

Binary trees are a commonly used data structure. The basic operations on binary trees are the tree constructions and traversals.

In this tutorial, we will use the dot character '.' to denote the "absent" nodes of the implicit array representation of a binary tree.

The implicit array tree is stored in a text file with the following format:

Row	Content	Remark
1st	C_1 C_2 C_3 … C_n	The implicit array representation of a binary tree C_i is the ith element (char type) of the binary tree

The structure of tree is defined as follows:

```
typedef char TreeElement;                 //define the tree element (char)
typedef struct _treenode {                //define the tree node structure
        TreeElement data;
        struct _treenode *left, *right;
} TreeNode;

typedef struct _tree {                    //define the tree structure
        struct _treenode *root;
} Tree;
```

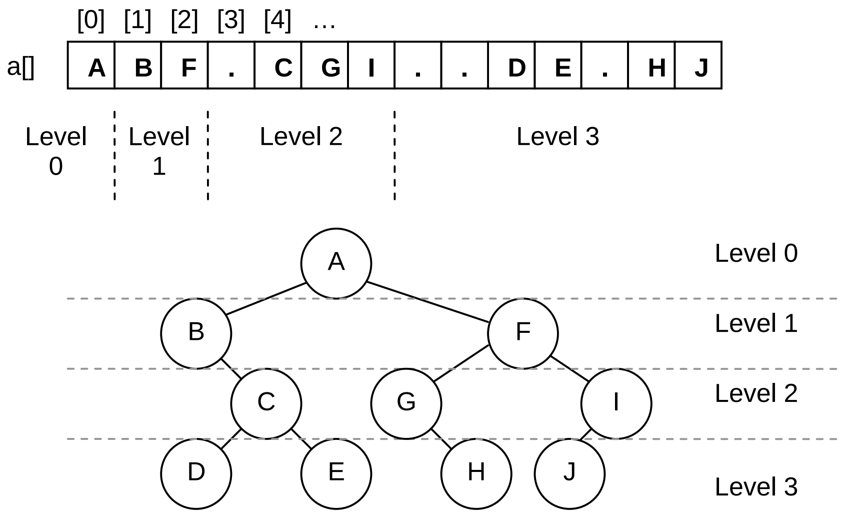

Figure 1. An example of implicit array tree representation.

Task A (Level 2): Construction of binary tree

Design the recursive function **TreeNode* arrayToLinkedTree(TreeElement a[], int index, int size)** to convert a tree in an implicit array representation to linked list representation.

The function accepts an implicit array tree (*a*[]), the current index of the array tree node (*index*), and the maximum size of the array tree (*size*). It is expected to return the pointer of a tree node structure that representing *a*[index] in the tree.

Expected Output:

```
Enter the file name for testing: test1.txt
The array tree is: [ABF.CGI..DE.HJ]

Enter your action ( 1) task a, 2) task b, 3) task c ): 1

The linked list tree is:
    I
      J
  F
      H
    G
A
      E
    C
      D
  B
```

Discussion: This function is now implemented by recursion. Can this function be implemented by iteration? Any extra data structures do you need?

Task B (Level 1): Double-order traversal

Besides inorder (LVR), preorder (VLR) and postorder (LRV), there are other traversal orders. One of them is called double-order (VLVR). For double-order, it is the combination of preorder and inorder. It will visit itself twice.

Implement the recursive function **print_double_order_sequence(TreeNode *p)** to traverse and print the sequence of the binary tree *t* in double-order.

Expected Output:

```
Enter the file name for testing: test1.txt
The array tree is: [ABF.CGI..DE.HJ]

Enter your action ( 1) task a, 2) task b, 3) task c ): 2

The double-order sequence is: A B B C D D C E E A F G G H H F I J J I
```

Discussion: Can you reconstruct the tree structure by considering the double-order sequence alone?

Task C (Level 2): Breadth-first traversal

In a breadth-first (level-order) traversal of a binary tree, the nodes are visited in an order according to their level. First visit the node at level 0, the root node. Then visit the nodes at level 1, in left-to-right order, and so on.

This task is to be solved by a non-recursive algorithm with the use of a queue (*Note: a queue of tree node pointers*). The idea is as follows. Firstly, we enqueue the pointer *p* (pointed at the root) into the queue. For each cycle, we dequeue a pointer *p* from the queue, print the node value, and enqueue *p*'s left and right pointers (if any) into the queue. Repeat until the queue is empty.

Design the non-recursive function **print_level_order_sequence(Tree *t)** to traverse and print the sequence of the binary tree *t* in level-order with the help of a queue.

Expected Output:

```
Enter the file name for testing: test1.txt
The array tree is: [ABF.CGI..DE.HJ]

Enter your action ( 1) task a, 2) task b, 3) task c ): 3

The level-order sequence is: A B F C G I D E H J
```