

Binary Trees

Computing Laboratory

<http://www.isical.ac.in/~dfs/lab>

(Recursive) Definition

A *binary tree* over a domain D is either:

- the empty set (called an *empty binary tree*); or
- a 3-tuple $\langle S_1, S_2, S_3 \rangle$ where
 - $S_1 \in D$, (called the *root*) and
 - S_2 and S_3 are *binary trees* over D (called the *left* and *right* subtree resp.)

Binary tree properties

- *Height: h*
- *Number of nodes: n*
- *Number of leaves: l*

Binary tree traversals

- *Preorder*
- *Inorder*
- *Postorder*

Conventional implementation:

```
typedef struct tnode {
    DATA d;
    struct tnode *left, *right;
    struct tnode *parent; // optional
} TNODE;
```

- One malloc per node
- Nodes may be scattered all over the heap area

Alternative implementation:

```
typedef struct tnode {
    DATA d;
    int left, right;
    int parent; //optional
} TNODE;

typedef struct {
    uint capacity, num_nodes;
    // num_nodes is optional
    int root, free;
    TNODE *nodelist;
} TREE;
```

- One initial malloc and reallocs as needed
- All nodes located within the same array

Binary tree implementation

Alternative implementation:

```
typedef struct tnode {
    DATA d;
    int left, right;
    int parent; //optional
} TNODE;

typedef struct {
    uint capacity, num_nodes;
    // num_nodes is optional
    int root, free;
    TNODE *nodelist;
} TREE;
```

Initially:

root = -1

	DATA	left	right
free → 0	—	1	-1
1	—	2	-1
2	—	3	-1
3	—	4	-1
:		:	
:		:	
n-1	—	-1	-1

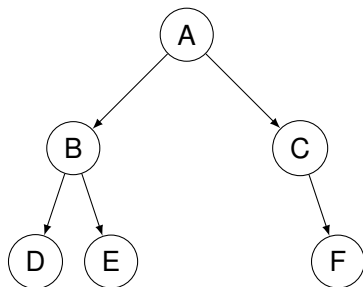
- One initial malloc and reallocs as needed
- All nodes located within the same array

Binary tree implementation

Alternative implementation:

root = 0

	DATA	left	right
0	A	1	2
1	B	3	4
2	C	-1	5
3	D	-1	-1
4	E	-1	-1
5	F	-1	-1
free → 6	—	7	-1
⋮		⋮	
n-1	—	-1	-1



Problems – I

For the following problems, some test cases (example trees) are available from the course home page. You may adapt the following code snippet to read in a tree and store it in an array (using the Alternative Implementation).

```
scanf("%u", &numNodes);
if (NULL == (tree = (NODE *) malloc(numNodes * sizeof(NODE)))) {
    fprintf(stderr, "Out of memory\n");
    exit(1);
}
for (node = tree, i = 0; i < numNodes; node++, i++)
    scanf("%d %d %d", &(node->data), &(node->left), &(node->right));
```

1. Using the code fragment given in the previous slide, write a `read_tree()` function with the following prototype.

```
int read_tree(TREE *t);
```

Modify your structure by adding a `parent` field, and modify your `read_tree()` function so that the `parent` field is filled in while reading the tree, without requiring a separate pass.

2. Given a binary tree with integer-valued nodes, and a *target* value, determine whether there exists a
 - (a) root-to-leaf path
 - (b) leaf-to-leaf path
 - (c) node-to-node path

in the tree such that the sum of all node values along that path equals the target.

3. Given a binary tree stored in an array (as in the Alternative Implementation), and the indices of two nodes in the tree, find the index of the node that is the lowest common ancestor of the given nodes.
4. 2024 / Lab Test 2 / Question 5
5. 2024 / Lab Test 4 / Question 4