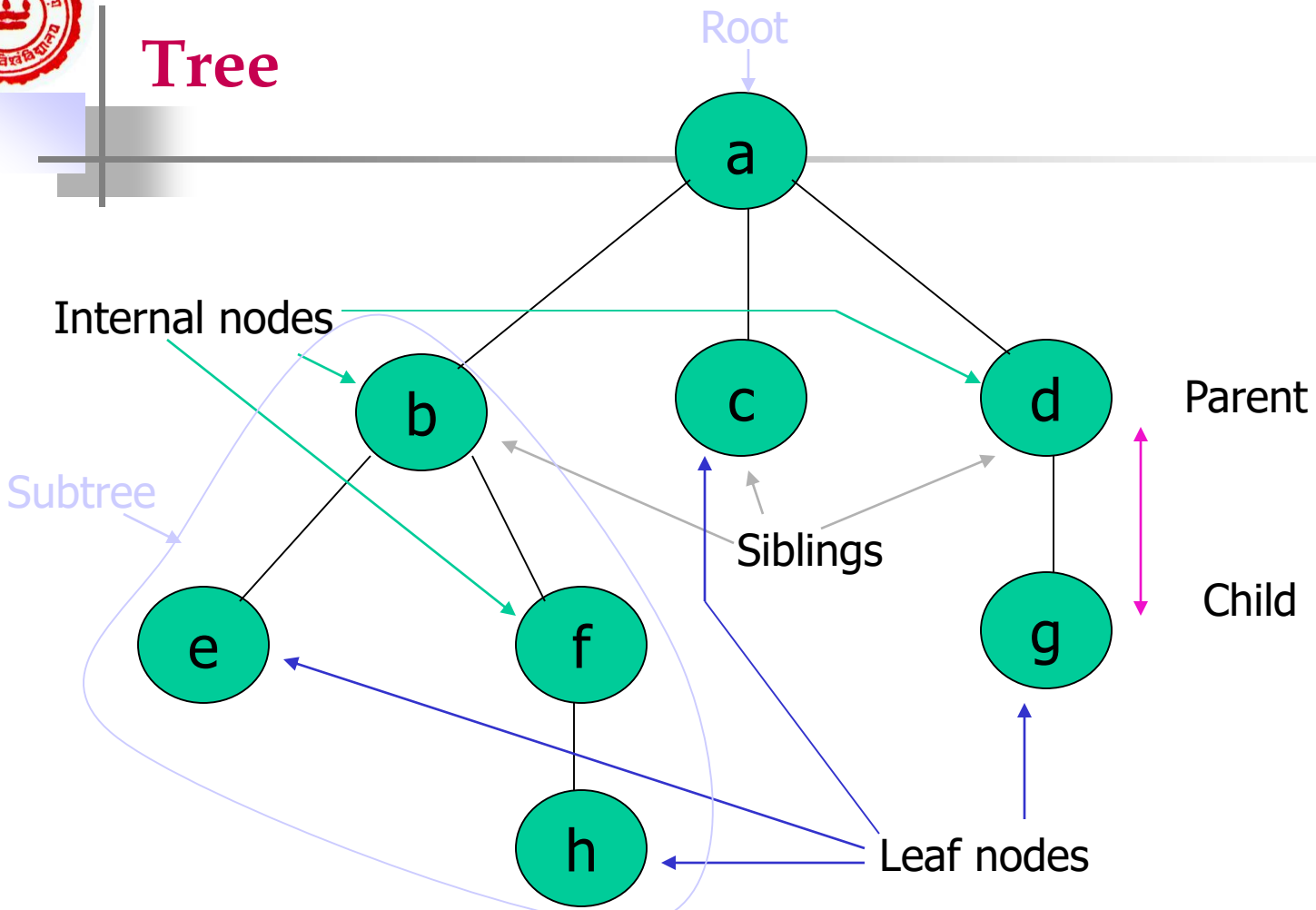
TREE

Tree



Edges – ab, ac, ad, be, bf, dg, fh

Path from a to e – abe

Path from a to h – abfh

Height = 4

A tree with n nodes has n-1 edges

Binary Tree

- A tree in which every node has at most two children is a Binary Tree.
- One can distinguish between the left child and the right child.

- A Full Binary Tree of height n is one that contains maximum possible number of nodes up to level n.

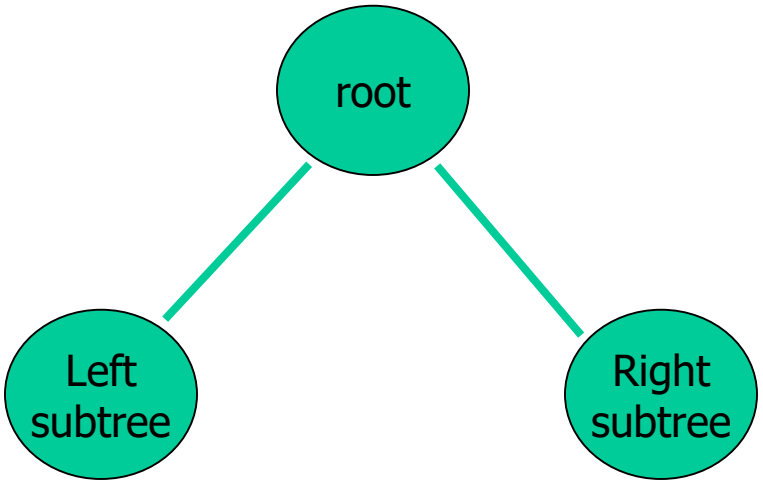
- A complete binary tree is one where

1. All the leaves are on level n or n-1.
2. On level 1 through n-2, all nodes have exactly two children.
3. On level n, the leaves are as far to the left as possible.

- Full and complete binary trees can be easily represented by arrays.

Binary Tree – Recursive definition

A binary tree can be viewed as composed of a root node, a left sub-tree and a right sub-tree. Both the sub-trees are binary.



Binary tree can be used to represent general trees.

How to represent a general n-ary tree using binary structure?

Properties of Binary Trees

1. The maximum no. of nodes on level i of a binary tree is 2^{i-1}, i >= 1.

2. The maximum no. of nodes in a binary tree of depth (height) k is 2^k-1, k>=1.

3. For any non-empty binary tree, if n_0 is the no. of terminal (leaf) nodes, and n_2 is the no. of nodes of degree 2 (having two children), then $n_0 = n_2 + 1$.

ADT Binary Tree

- **Objects:** A binary tree is a structure of nodes of the same type. Each node is the root of a sub-tree, having a left and right child which are binary trees. Each node has a key field that identifies the entry.
- **Operations:**

  init_t(t) – initialize an empty tree.

  empty_t(t) – boolean function to return true if the tree t is empty.

  height(t) – function to return the height of tree t.

  make_root(n) – procedure to make node n the root of a tree.

  addleft(t, n) – add node n as a left child of tree t.

  addright(t, n) – add node n as a right child of tree t.

  deleteleaf(n) – delete the leaf node n.

Binary Tree Traversals

Preorder Traverse

Root-Left-Right

Inorder Traverse

Left-Root-Right

Postorder Traverse

Left-Right-Root

Perform a traversal of the nodes of the tree in the specified order and visit each node. Visiting involves performing some computation with the data in the node.

A few problems

1. Delete a tree
2. Boolean function ***isleaf*** (t)
3. Function ***sizetree*** to count the no. of nodes in a tree
4. Function ***count leaf***
5. ***Equaltree*** (t_1 , t_2)
6. Function ***maxintree***
7. ***Copytree***(s,t)
8. ***Isintree***(t,n)
9. ***Printtree*** - LTR, TLR, LRT, RTL, TRL, RLT
10. Find
    a) In-order-predecessor
    b) In-order-successor

Tree Implementation using C Pointers

```
typedef struct nt {
                T   info;
                struct nt *left, *right;
              } bintree;

bintree *t, *cur;

bintree *init_t() {   return   NULL; }

int empty_t(bintree *t) { return (t == NULL) ;}

bintree *makeroot(T  n) {
                        bintree *t;
                        if((t = malloc(sizeof(bintree))) == NULL)
                                perror("malloc error");
                        else {
                                t -> info = n;
                                t -> left =NULL;
                                t -> right = NULL;
                        }
                        return t;
                }
```

Tree Implementation using C Pointers …

```
void addleft ( bintree *t , T  n ) {
                        t -> left = makeroot ( n );
                    }
 void addright ( bintree *t, T  n ) {
                        t -> right = makeroot ( n );
                     }
 int height ( bintree *t ) {
                if ( empty_t (t) )
                        return 0;
                else
                        return(1 + max(height(t -> left), height ( t -> right)));
            }

void preorder_traverse ( bintree *t ) {
                                    if ( !(empty_t (t)))
                                    {
                                                visit (t);
                                                preorder_traverse (t -> left);
                                                preorder_traverse (t ->right);
                                    }
                                }
```

- How to use different visit functions in the same traversal function?

Threaded Binary Tree

- Frequent inorder-traversals of long binary trees using recursive routines involve a lot of space and time.

- Use of right pointers in a node having no right child to point to the inorder-successor can avoid recursion. Such pointers are called Threads.

```
typedef struct nt {  T  info ;
                     struct nt *left, *right ;
                     int thread ;
                   } threaded_bintree
                     ;
```

Threaded Binary Tree ...

```
void th_inorder_traverse ( threaded_bintree *t) {
    threaded_bintree *cur ;
    cur = t ;
    while ( !empty_t (cur)) {
        while ( !(empty_t(cur -> left)))
            cur = cur -> left ;
        visit (cur) ;
        while((cur -> thread) & (!(empty_t(cur ->right)))) {
            cur = cur -> right ;
            visit (cur);
        }
        cur = cur -> right ;
    }
}
```

- Define a complete ADT for Threaded Binary Tree and write the functions for ADT operations

Summary

- Binary tree is a widely used ADT in many applications.
- The concept of two way navigation is basic to many high level problem solving
- Application of Binary tree are prevalent in
  - Set representation
  - Decision Trees
  - Game Trees
  - Searching
- Interesting constraints are imposed on Binary Tree to arrive at many useful data structures