

Chapter 13

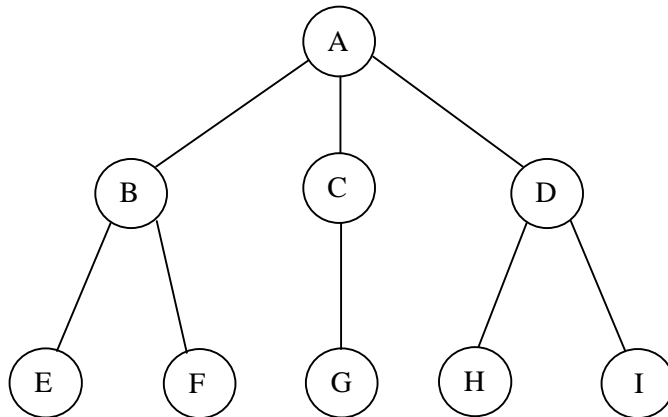
Trees

Data Structures in C++: Using the STL
Timothy Budd

- Tree Terminology
- Tree Definition
- Tree Representation
- Tree Traversal
- Binary Trees

Tree Terminology

- A sample tree:



Vertex or node: the labeled circles

Edge or arc: the links joining the nodes

Parent/Child relationship (top-down)

Siblings: nodes with a common parent

Predecessors: Ancestors

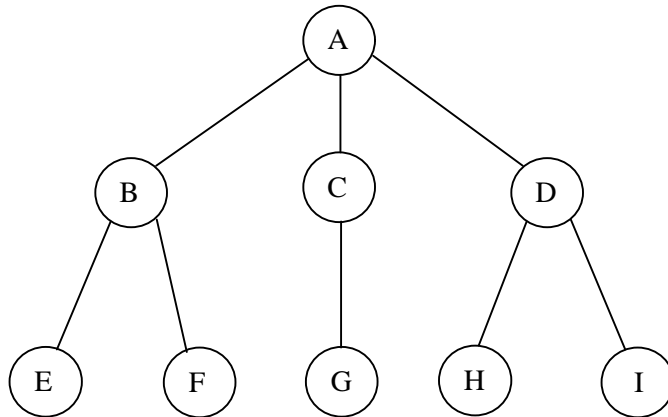
Successors: Descendants

Root: One per tree ($\text{root} \in V$)

Leaf: a node w/o descendants ($\text{leaves} \subseteq V$)

Interior node: node \notin leaves (could be root)

Tree Terminology



Height: No. of levels

Level: Starts with root (level 0)

Branching Factor: Max no. of children/node

Given: b = branching factor

h = height

ℓ = level

b^ℓ = max no. of nodes @ level ℓ

$b^h - 1$ = max no. of nodes in tree

Full: Every level fully populated

Complete: No nodes @ level $i+1$ until level i is full

Tree Definition

A three-part definition:

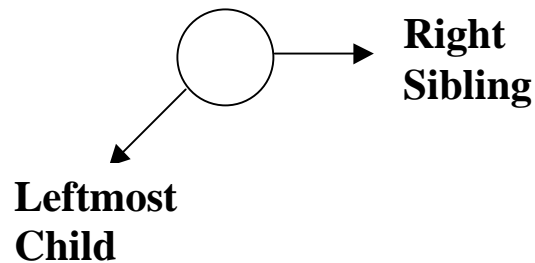
- A connected graph of n nodes
- No cycles (circuits)
- Exactly $n-1$ edges

All **three** are *necessary*

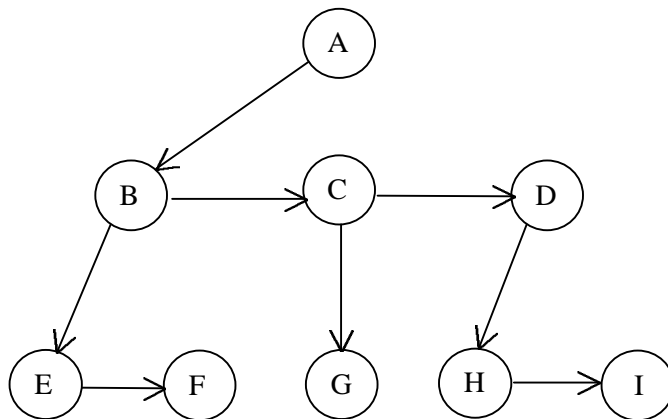
Any **two** are *sufficient*

Tree Representation

Every tree can be represented as a binary tree. At each node store two pointers:



The previous tree would be converted to:



(Only non-null pointers are shown above)

Tree Traversal

Given these 3 members in each node:

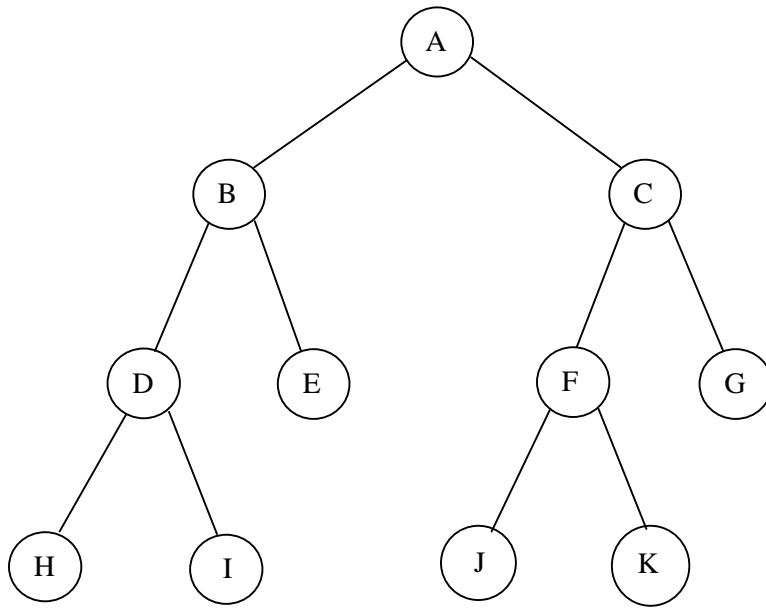
LChild	Data	RChild
--------	------	--------

Three different orderings of DFS:

- LDR – Inorder
- DLR – Preorder
- LRD – Postorder

Process the 3 members at each node in the appropriate order.

Binary Tree



Inorder:

Preorder:

Postorder:

Binary Tree Traversals

Inorder

```
template <class T>
class tree
{
public:
    void inorder(tree<T>* t);
    void preorder(tree<T>* t);
    void postorder(tree<T>* t);
private:
    tree<T>* Lchild;
    T Data;
    Tree<T>* Rchild;
};

template <class T>
void tree<T>::inorder(tree<T>* t)
{
    if (t != 0)
    {
        inorder(t->Lchild);
        cout << t->Data;
        inorder(t->Rchild);
    }
}
```

Binary Tree Traversals

Preorder

```
template <class T>
class tree
{
public:
    void inorder(tree<T>* t);
    void preorder(tree<T>* t);
    void postorder(tree<T>* t);
private:
    tree<T>* Lchild;
    T Data;
    Tree<T>* Rchild;
};

template <class T>
void tree<T>::preorder(tree<T>* t)
{
    if (t != 0)
    {
        cout << t->Data;
        preorder(t->Lchild);
        preorder(t->Rchild);
    }
}
```

Binary Tree Traversals

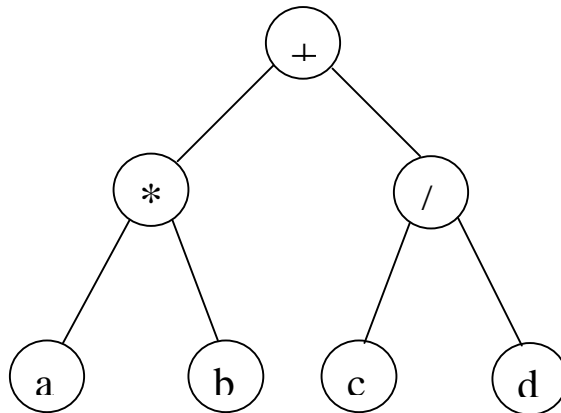
Postorder

```
template <class T>
class tree
{
public:
    void inorder(tree<T>* t);
    void preorder(tree<T>* t);
    void postorder(tree<T>* t);
private:
    tree<T>* Lchild;
    T Data;
    Tree<T>* Rchild;
};

template <class T>
void tree<T>::postorder(tree<T>* t)
{
    if (t != 0)
    {
        postorder(t->Lchild);
        postorder(t->Rchild);
        cout << t->Data;
    }
}
```

Expression Trees

- Binary trees containing:
 - Binary operators
 - operands
- Interior nodes hold operators
- Leaves hold operands



Preorder:

Inorder:

Postorder: