Binary Trees

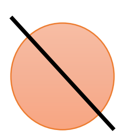
CSE 2020 Computer Science II

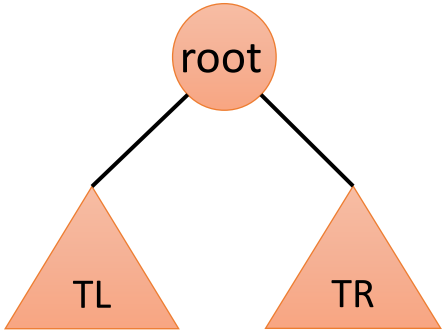
Learning Objectives

- define binary tree ADT
- explain the concepts related to binary trees, including path, length, level, depth, complete binary trees, and full binary trees
- implement pre-order, in-order, post-order, and level-order traversal

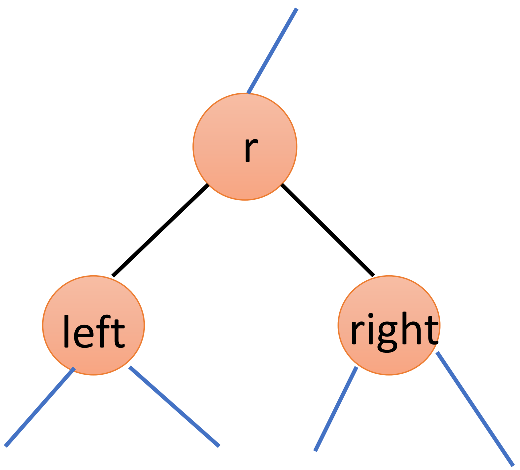
Basic Concepts

- A binary tree is a collection of nodes. This collection can be empty, or consists of a distinguished node called root, together with left binary subtree and right binary subtree, whose roots are connected by a directed edge from root.
  - recursive definition
  - two subtrees are disjoint from each other





Basic Concepts



- Parent and child
  - edge from parent to its child nodes
  - r is parent of left and right
  - left is the left child of r
  - right is the right child of r
- Leaf node is any node that has no child or has two empty children

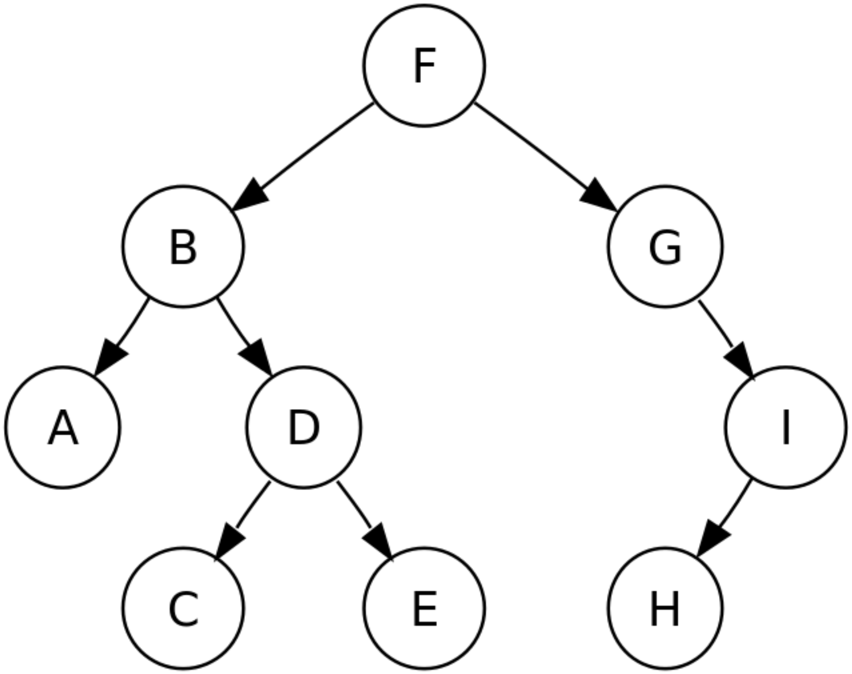
Basic Concepts

- A **path** from node n_1 to n_k is a sequence of nodes $n_1, n_2, \ldots\ldots, n_i, n_{i+1}, \ldots\ldots, n_k$ such that n_i is the parent of n_{i+1}
- The **length** of a path is the number of *edges* in this path. $n_1, n_2, \ldots\ldots, n_k$, the length is k-1
- The **depth** of a node n_i is the length of the unique path from the root to n_i. depth(root) = 0
- The **level** of a node is i if the depth of the node is i, the node is at the level i. The level of root is 0.
- The **height** of a node is the length of the longest path from this node to a leaf. The height of all leaves are 0. The height of a tree is the height of the root.

Binary Tree Example

- root F
- leaves: A, C, E, H
- height = 3

  FBDE or FBDC or FGIH
- depth(I) = 2
- depth(E) = 3
- 4 levels, level 0 to 3
  - level 0 F
  - level 1 B G
  - level 2 A D I
  - level 3 C E H

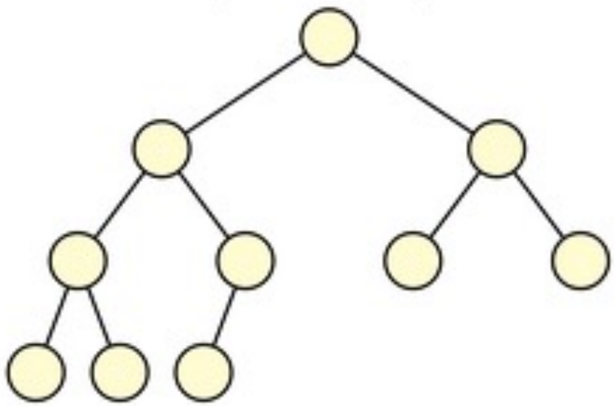


Basic Concepts

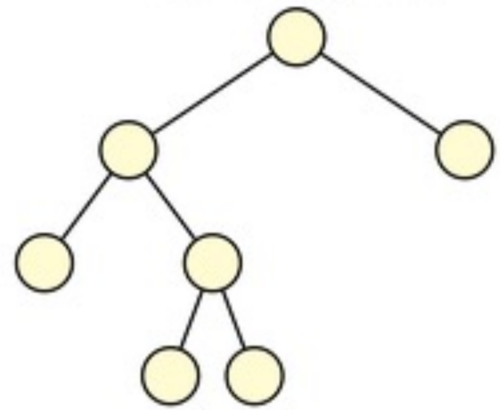
- A binary tree is **full** if each node is either a leaf or has two non-empty children
- A binary tree is **complete** if all levels except the last are completely filled and the last level has all its nodes filled in from left side.
- Please note that there is no particular relationship between above two tree shapes
- **Perfect binary tree** is a binary tree where all levels are completely filled.

Examples

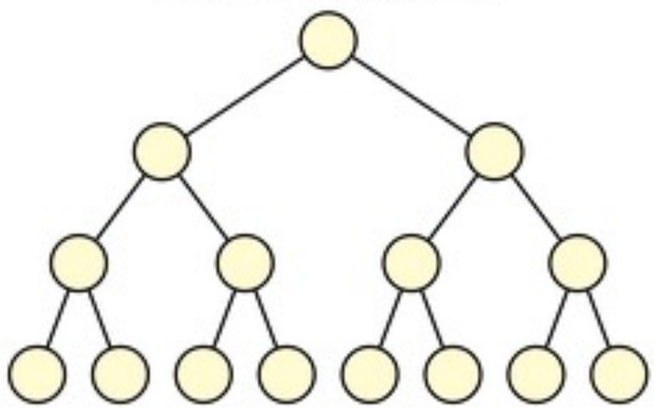
Complete Binary Tree



Full Binary Tree



Perfect Binary Tree



Complete Binary Tree

- A complete binary tree with height h (h + 1 Levels)

  Level 0   1 node

  Level 1   2 nodes

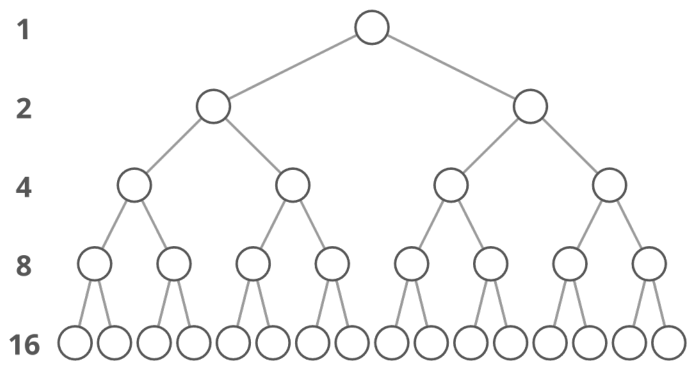
  Level 2   4 nodes

  ……

  Level k 2^k nodes

  ……

  Level h min 1 node, max 2^h nodes



- Perfect binary tree with height h, the number of nodes is

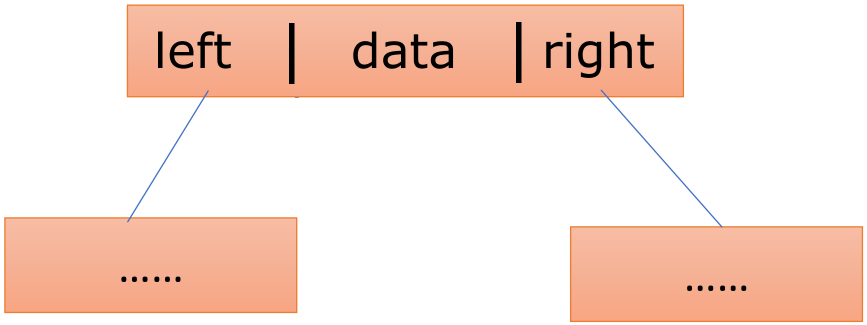
$$\sum_{i=0}^{h} 2^i = \frac{1-2^{h+1}}{1-2} = 2^{h+1} - 1$$

- Complete binary tree with n nodes, the height h

$$2^h \le n \le 2^{h+1} - 1 \Rightarrow \mathrm{h} = \lfloor log_2 n \rfloor \text{ or } h = floor(log_2 n)$$

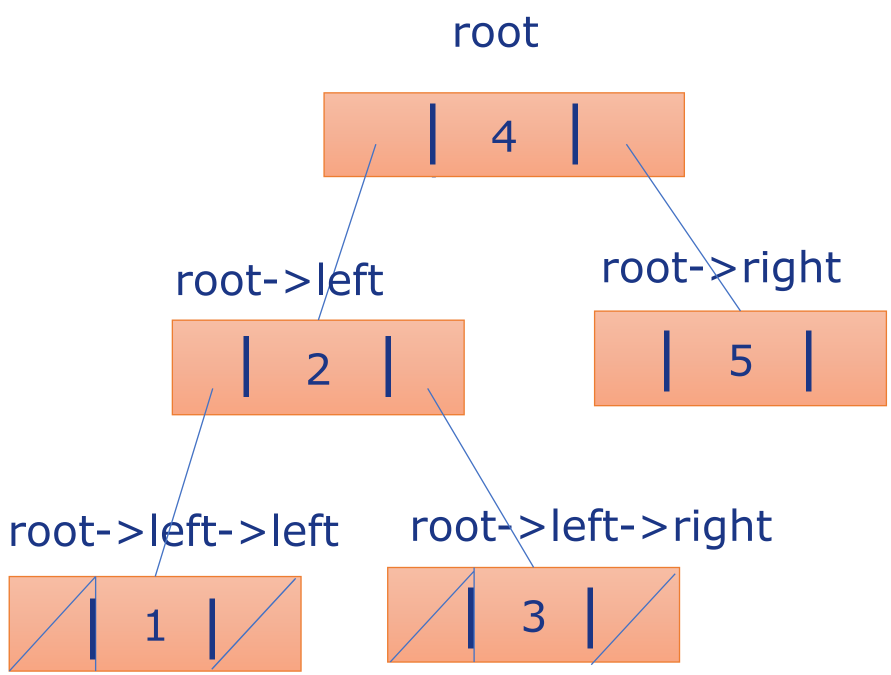
Binary Tree Node struct template

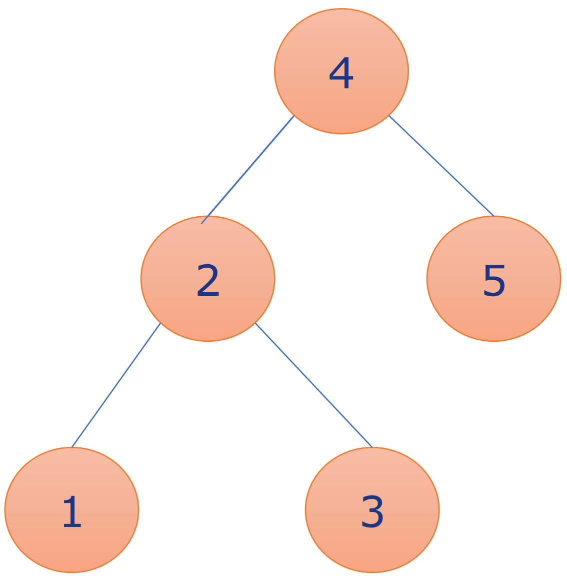
```
template <typename T>
struct BinaryNode
{
    T data;
    BinaryNode* left;
    BinaryNode* right;

    BinaryNode(const T& d = T()):
                data(d),left(nullptr),right(nullptr)
    {}
};
```



Example

- BinaryNode<int>* root;





Binary Tree Traversal

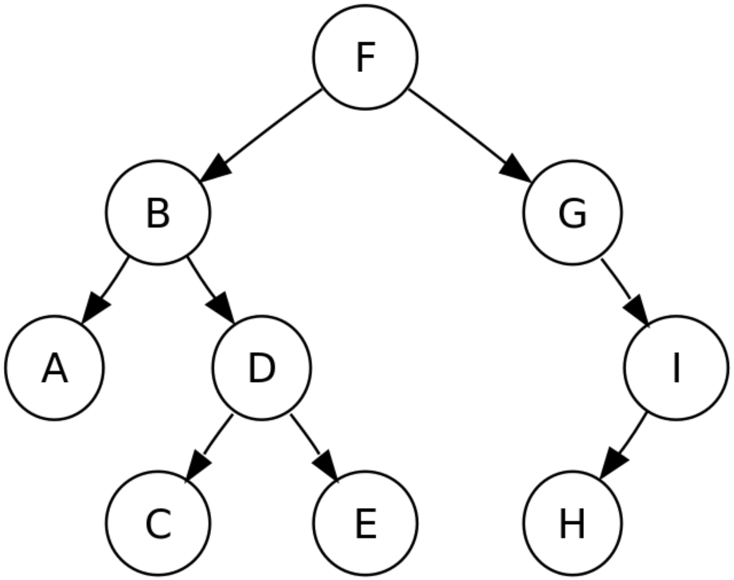
- Traversal means visiting or performing a specific action (such as print) to all nodes in some order.
- Preorder traverse the tree with root node:
  - access root node
  - preorder traverse left subtree of root
  - preorder traverse right subtree of root
- Inorder traverse:
  - inorder traverse left subtree of root
  - access root node
  - inorder traverse right subtree of root

Binary Tree Traversal (cont.)

- Postorder traversal:
  - postorder traverse left subtree of root
  - postorder traverse right subtree of root
  - access root node
- Level-order traversal
  - access root at level 0
  - access nodes at level 1 from left to right
  - access nodes at level 2 from left to right
  - …….

Binary Tree Traversal Example

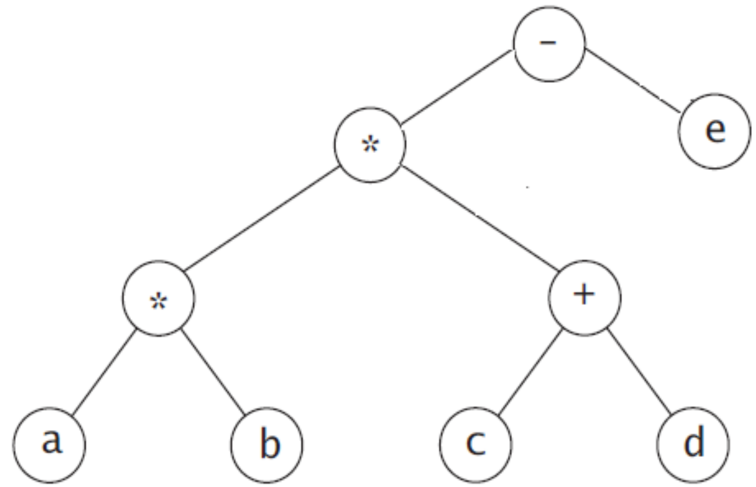
- Preorder
  F B A D C E G I H
- Inorder
  A B C D E F G H I
- Postorder
  A C E D B H I G F
- Level-order
  F B G A D I C E H



- Question:
  - can the combination of preorder and inorder traversal sequences uniquely identify a binary tree?
  - how about postorder and inorder?
  - how about preorder and postorder?

Expression Tree

- Leaves are operands
- Other nodes contain operators
- prefix representation
- infix representation
- postfix representation



Inorder Traversal - Recursive

- Recursive inorder traversal

```
template <typename T>
void inorder(BinaryNode<T>* t)
{
   if (t != nullptr)
   {
         inorder(t->left);
         cout << t->data;
         inorder(t->right);
   }
}
```

Recursive Function Call Process

- inorder(root)
  - root is passed to inorder function
  - if statement, root is not nullptr, execute if body
  - inorder(root->left)
    - root->left is passed to inorder function
    - if statement, root->left is not nullptr, execute if body
    - inorder(root->left->left)
      - root->left->left is passed to inorder function
      - If statement, root->left->left is not nullptr, execute if body
      - inorder(root->left->left->left)
        - root->left->left->left is passed to inorder function
        - if statement, root->left->left->left is nullptr, exit inorder function
    - cout << root->left->left->data;
    - inorder(root->left->left->right)
      - root->left->left->right is passed to inorder function
      - if statement, ……

Inorder Traversal – Non-recursive

- Non-recursive inorder traversal

```
Stack<BinaryNode<T>*> s;
cur = root;
while (cur is not nullptr OR s is not empty)
   while (cur is not nullptr)
       s.push(cur)
       cur = cur->left
  cur = s.top();
  access cur->data
  s.pop()
  cur = cur->right
```

Preorder Traversal - Recursive

- Recursive preorder traversal

```
template <typename T>
void preorder(BinaryNode<T>* t)
{
    if (t != nullptr)
    {
        cout << t->data;
        preorder(t->left);
        preorder(t->right);
    }
}
```

Preorder Traversal – Non-recursive

- Non-recursive preorder traversal

```
Stack<BinaryNode<T>*> s;
if root is not nullptr then
   s.push(root)
   while s is not empty
       cur = s.top();
       access cur element
       s.pop();
       push non-empty right child of cur
       push non-empty left child of cur
```

Postorder Traversal - Recursive

- Recursive postorder traversal

```
template <typename T>
void postorder(BinaryNode<T>* t)
{
   if (t != nullptr)
   {
        postorder(t->left);
        postorder(t->right);
        cout << t->data;
   }
}
```

Postorder Traversal Application

- Clear Function

```
template <typename T>
void clear(BinaryNode<T>*& t)
{
        if (t != nullptr)
        {
             clear(t->left);
             clear(t->right);
             delete t;
        }
}
```

Level-order Traversal

```
Queue<BinaryNode<T>*> q;
if root is not nullptr
     q.enqueue(root);
     while q is not empty
          cur = q.front_element();
          access the cur element
          q.dequeue();
          enqueue non-empty left child of cur
          enqueue non-empty right child of cur
```