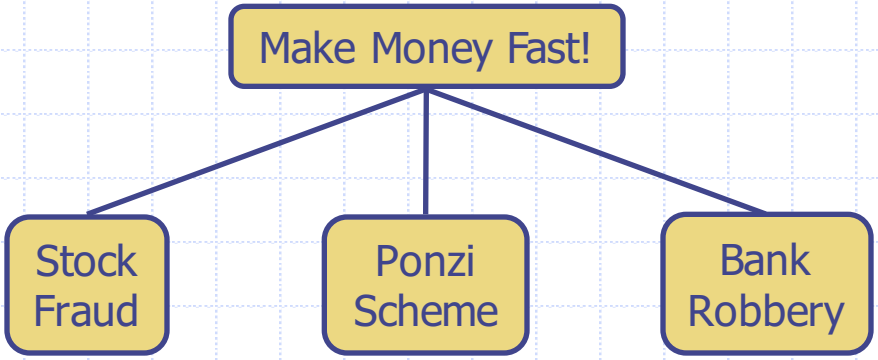
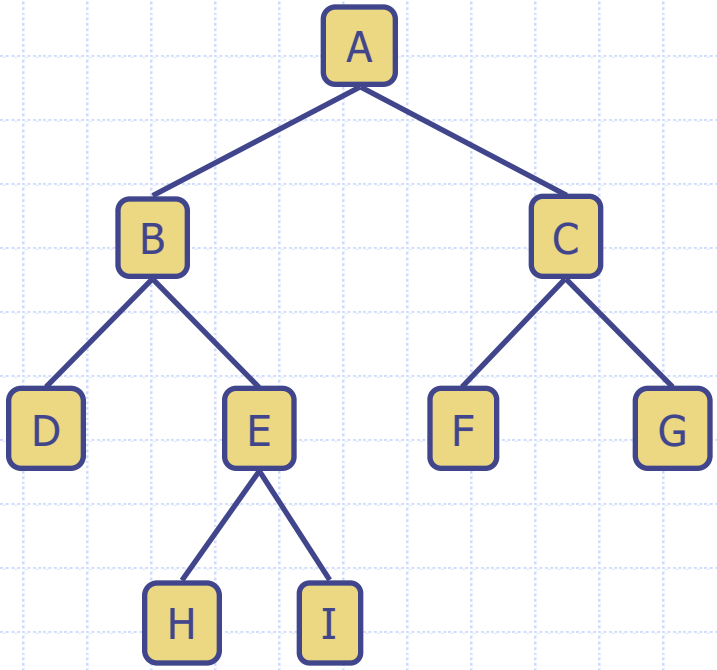
Trees



Binary Trees

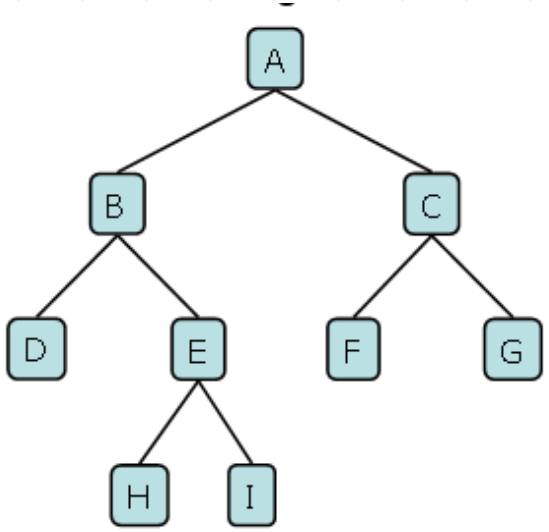
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



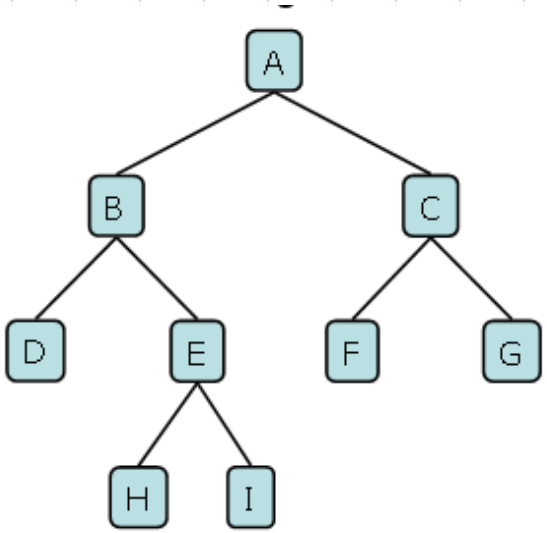
Binary Tree Traversals

- Trees can be traversed in different ways
- **Breadth-first traversal**
  - Visiting each node starting from the lowest (or highest) level and moving down (or up) level by level, visiting nodes on each level from left to right (or from right to left)
  - **XXX what is it??**
- **Depth-First Traversals**
  - Inorder (Left, Root, Right) : **xxx what is it?**
  - Preorder (Root, Left, Right) : **xxx what is it?**
  - Postorder (Left, Right, Root): **xxx what is it?**



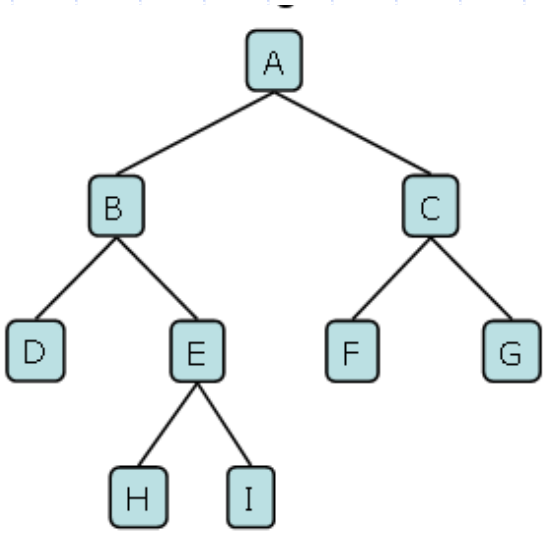
Binary Tree Traversals

- **Breadth-first traversal**
  - Visiting each node starting from the lowest (or highest) level and moving down (or up) level by level, visiting nodes on each level from left to right (or from right to left)
  - **→A,B,C,D,E,F,G,H,I**
- Algorithm:
  - Print all nodes at a given level (printCurrentLevel)
  - Print the level order traversal of the tree (printLevelorder). printLevelorder makes use of printCurrentLevel to print nodes at all levels one by one starting from the root.



Binary Tree Traversals

- **Breadth-first traversal**
  - Visiting each node starting from the lowest (or highest) level and moving down (or up) level by level, visiting nodes on each level from left to right (or from right to left)
  - **→A,B,C,D,E,F,G,H,I**
- Algorithm: method 1
  - Print all nodes at a given level (printCurrentLevel)
  - Print the level order traversal of the tree (printLevelorder). printLevelorder makes use of printCurrentLevel to print nodes at all levels one by one starting from the root.



Binary Tree Traversals
- Breadth-first traversal

- **Class Node → ?**
- **Class BinaryTree → ?**
  - **Def heigh(self, node)→?**
  - **Def printCurrentLevel(self, node, level):→ ?**
  - **Def printLevelOrder(self):→ ?**

Do by your self

Binary Tree Traversals

- Breadth-first traversal

```
class Node:
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self, node):
        self.root= node
```

Binary Tree Traversals
- Breadth-first traversal

```
def height(self, node):
    if node is None:
        return 0
    else:
        # Compute the height of each subtree
        lheight = self.height(node.left)
        rheight = self.height(node.right)
        # Use the larger one
        if lheight > rheight:
            return lheight + 1
        else:
            return rheight + 1
```

Binary Tree Traversals
- Breadth-first traversal

```
# Print nodes at a current level
def printCurrentLevel(self,node, level):
    if node is None:
        return
    if level == 1:
        print(node.data, end=" ")
    elif level > 1:
        self.printCurrentLevel(node.left, level-1)
        self.printCurrentLevel(node.right, level-1)
```

Binary Tree Traversals
- Breadth-first traversal

```
def printLevelOrder(self):
    h = self.height(self.root)
    for i in range(1, h+1):
        self.printCurrentLevel(self.root,i)
```

Binary Tree Traversals

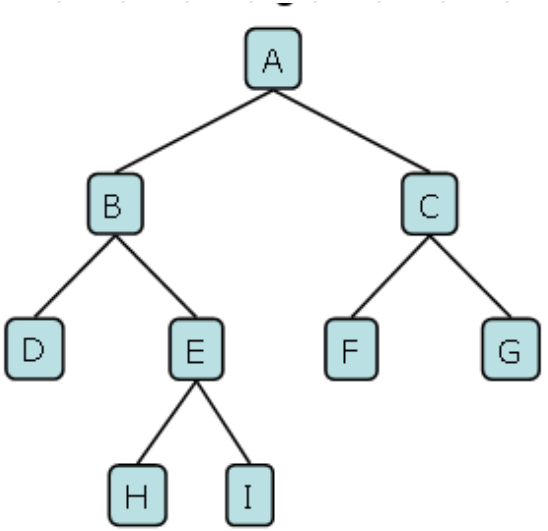
- Breadth-first traversal

```
btree= BinaryTree(Node("A"))
btree.root.left = Node("B")
btree.root.right = Node("C")
btree.root.left.left = Node("D")
btree.root.left.right = Node("E")
btree.root.right.left = Node("F")
btree.root.right.right = Node("G")
btree.root.left.right.left = Node("H")
btree.root.left.right.right = Node("I")

print("Level order traversal of binary tree is -")
btree.printLevelOrder()
```

Binary Tree Traversals

- **Breadth-first traversal**
  - Visiting each node starting from the lowest (or highest) level and moving down (or up) level by level, visiting nodes on each level from left to right (or from right to left)
  - **→A,B,C,D,E,F,G,H,I**
- Algorithm: method 2 using queue
  - printLevelorder(tree)
    - 1) Create an empty queue q
    - 2) temp_node = root
    - 3) Loop while temp_node is not NULL
      - a) print temp_node->data.
      - b) Enqueue temp_node's children (first left then right children) to q
      - c) Dequeue a node from q.



Binary Tree Traversals
- Breadth-first traversal

```
class Node:
    # A utility function to create a new node
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None
```

```
class BinaryTree:
    def __init__(self, node):
        self.root= node
```

Binary Tree Traversals
- Breadth-first traversal

```
def printLevelOrder(self):
    # Base Case
    if self.root is None:
        return
    queue = []
    # Enqueue Root and initialize height
    queue.append(self.root)
    while(len(queue) > 0):
        # Print front of queue and
        # remove it from queue
        print(queue[0].data)
        node = queue.pop(0)
        # Enqueue left child
        if node.left is not None:
            queue.append(node.left)
        # Enqueue right child
        if node.right is not None:
            queue.append(node.right)
```

Binary Tree Traversals
- Breadth-first traversal

```
btree= BinaryTree(Node("A"))
btree.root.left = Node("B")
btree.root.right = Node("C")
btree.root.left.left = Node("D")
btree.root.left.right = Node("E")
btree.root.right.left = Node("F")
btree.root.right.right = Node("G")
btree.root.left.right.left = Node("H")
btree.root.left.right.right = Node("I")

print("Level order traversal of binary tree is -")
btree.printLevelOrder()
```

Binary Tree Traversals- Depth First Traversals

```
Algorithm Inorder(tree)
    1. Traverse the left subtree, i.e., call Inorder(left-subtree)
    2. Visit the root.
    3. Traverse the right subtree, i.e., call Inorder(right-subtree)
```

```
Algorithm Preorder(tree)
    1. Visit the root.
    2. Traverse the left subtree, i.e., call Preorder(left-subtree)
    3. Traverse the right subtree, i.e., call Preorder(right-subtree)
```

is used to create a copy of the tree

```
Algorithm Postorder(tree)
    1. Traverse the left subtree, i.e., call Postorder(left-subtree)
    2. Traverse the right subtree, i.e., call Postorder(right-subtree)
    3. Visit the root.
```

is used to delete the tree.

Binary Tree Traversals- Depth First Traversals

- **Class Node → ?**
- **Class BinaryTree → ?**
  - **def printInorder(self,root):→?**
  - **def printPostorder(self,root):→ ?**
  - **def printPreorder(self,root):→ ?**

Do by your self

Binary Tree Traversals- Depth First Traversals

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.data = key
class BinaryTree:
    def __init__(self, node):
        self.root= node
```

Binary Tree Traversals- Depth First Traversals

```
def printInorder(self,root):
    if root:
        self.printInorder(root.left)
        print(root.data),
        self.printInorder(root.right)
```

Binary Tree Traversals- Depth First Traversals

```
def printPostorder(self,root):
    if root:
        self.printPostorder(root.left)
        self.printPostorder(root.right)
        print(root.data),
```

Binary Tree Traversals- Depth First Traversals

```
def printPreorder(self,root):
    if root:
        print(root.data),
        self.printPreorder(root.left)
        self.printPreorder(root.right)
```

Binary Tree Traversals- Depth First Traversals

```
btree= BinaryTree(Node("A"))
btree.root.left = Node("B")
btree.root.right = Node("C")
btree.root.left.left = Node("D")
btree.root.left.right = Node("E")
btree.root.right.left = Node("F")
btree.root.right.right = Node("G")
btree.root.left.right.left = Node("H")
btree.root.left.right.right = Node("I")
print ("Preorder traversal of binary tree is")
btree.printPreorder(btree.root)
print ("\nInorder traversal of binary tree is")
btree.printInorder(btree.root)
print ("\nPostorder traversal of binary tree is")
btree.printPostorder(btree.root)
```

Binary Search Tree Traversals

Slot sau