Data Structures and Algorithms
Lecture notes: Trees

Lecturer: Michel Toulouse

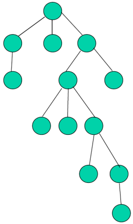
Hanoi University of Science and Technology
michel.toulouse@soict.hust.edu.vn

31 mai 2021

Outline

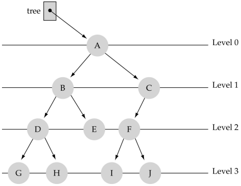
What is a tree

- Here a tree is a data structure (like arrays, stacks, queues), but more generally a tree is a form of graph, so trees have nodes and edges
- Nodes in a tree, like nodes in a queue, have a data field and have pointer field(s).



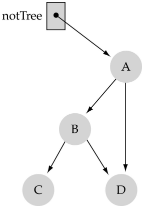
What is a tree

- Like in queues, nodes are connected by pointers to form a sequence (called levels in a tree), usually the beginning of the sequence is the node at the highest level in the tree
- Unlike queues, nodes in a tree can branch out in several directions from level i to level $i+1$



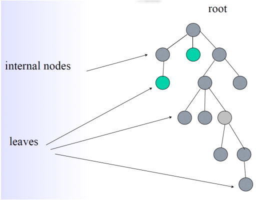
Not a tree

- In a queue, the pointer to the first node in the sequence was called the "head", for a tree it will be named the root
- A tree has a unique path from the root to every other node
- Thus the graph on the right is not a tree as there exists two paths from the root A to the leaf D



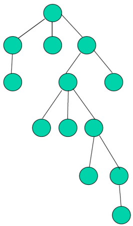
Types of nodes in a tree

- The first node in a tree is also called the root of the tree
- The nodes at the end of a branch in the tree are called leaves
- The other nodes are named internal nodes



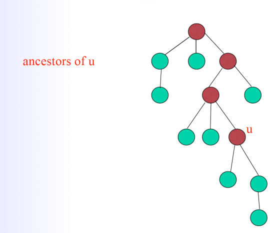
Tree : parent, child, siblings

- A node y at level i connected to a node x at level $i-1$ is said to be the child of node x and x is the parent of node y
- The root node has no parent
- Leave nodes have no child
- The child nodes w, y, z of a same parent node x are said to be siblings with respect to each other



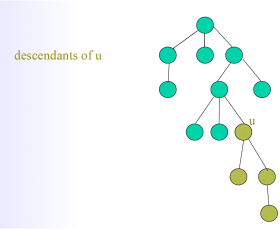
Tree : ancestors, height, depth

- The ancestors of a node u are all the nodes on the path from u to the root
- According to CLRS textbook, u is his own ancestor
- The root has only itself as ancestor
- The height of a node u in a tree is the length of the longest path from u to any leaf
- The height of a leaf is 0
- The height of u in the tree is 2
- The height of a tree is the height of its root
- The depth of tree is the number of levels in the tree, the present tree has depth 5
- The depth of a node is the level of that node in the tree. The depth of the root is 0, the depth of node u in the tree is 3



Tree : descendants and subtrees

- The descendants of a node u are the nodes on all the paths from node u to leaves
- According to CLRS textbook, u is his own descendant
- The descendants of a node u form of subtree rooted at node u
- If n is the number of nodes in a tree, there are $n-1$ subtrees
- A subtree where the root is a leave has only itself as descendant
- The degree of a node u is the number children of u
- The degree of a tree is equal to the largest degree of its nodes



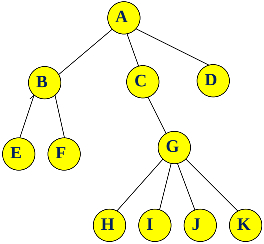
Node declaration in C

Nodes in a tree are declared in the same way as for nodes of a queue. Nodes have two pointers, one for the leftmost child and one for the right-sibling of the leftmost child

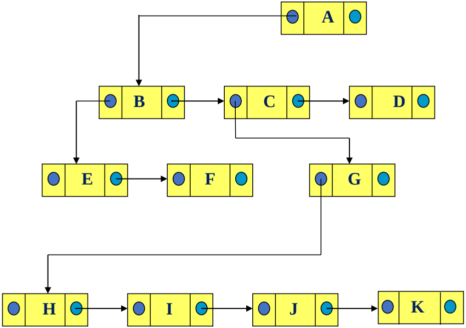
Data	
Leftmost Child	Right-sibling

```
typedef struct
{
 int data;  // data of each node
 struct treeNode  * leftmost_child;
 struct treeNode  * right_sibling;
}treeNode;
treeNode * Root;
```

Tree example

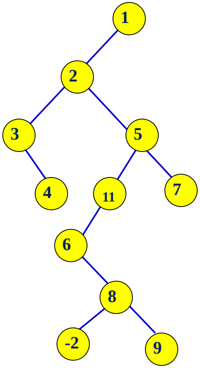


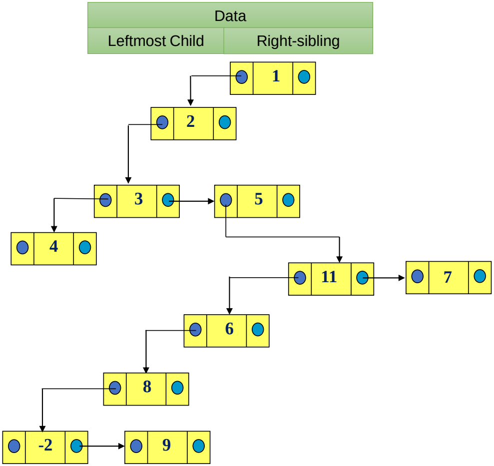
Data	
Leftmost Child	Right-sibling




```
typedef struct
{
  char data;  // data of each node
  struct treeNode  * leftmost_child;
  struct treeNode  * right_sibling;
}treeNode;
treeNode * Root;
```

Tree second example



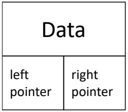


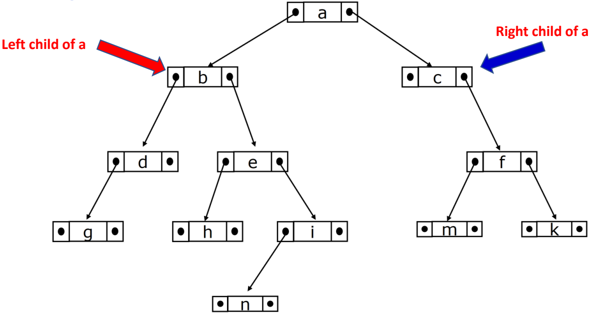

```
typedef struct
{
  int data;  // data of each node
  struct treeNode  * leftmost_child;
  struct treeNode  * right_sibling;
}treeNode;
treeNode * Root;
```

Binary trees

A binary tree is a tree such that

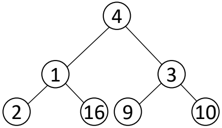
- every node has at most 2 children
- each node is labeled as being either a left child or a right child





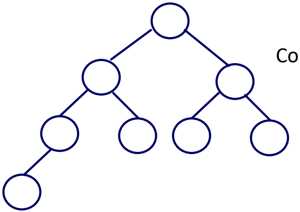
Full versus complete binary trees

- ***Full*** binary tree: a binary tree in which
  - every parent has 2 children,
  - every leaf has equal depth

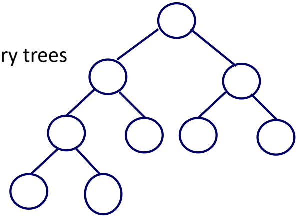


Full binary tree

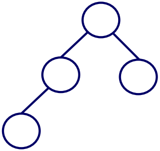
- ***Complete*** binary tree: a binary tree in which
  - every level is full except possibly the deepest level
  - if the deepest level isn't full, leaf nodes are as far to the left as possible



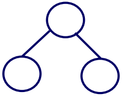
Complete binary trees



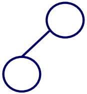
Examples



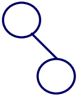
Complete binary tree



Full and complete

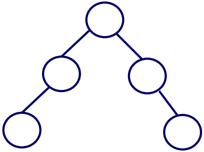


Complete



Neither full nor complete

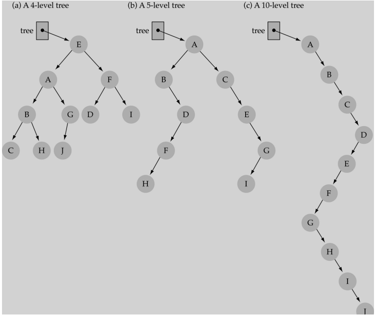
Full and complete



Neither full nor complete

Height of binary tree

- The maximum height of a binary tree with n nodes is the same as the length of a link list with n nodes, i.e. n
- The minimum height of a binary tree with n nodes is $\lceil \log(n+1) \rceil - 1$
- Full and complete binary trees have minimum height



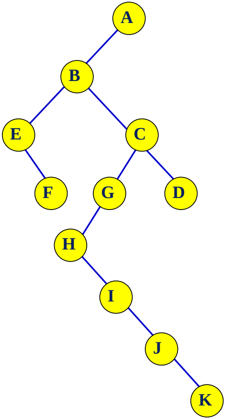
Binary tree representation

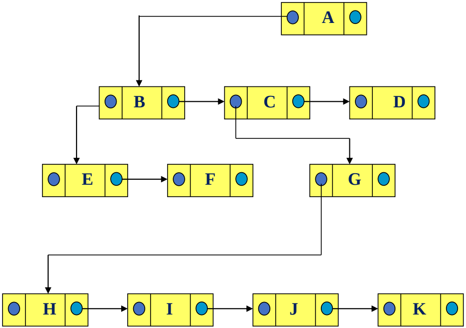
Binary trees are represented using pointers in a similar ways as ordinary trees :

- Each node contains the address of the left child and the right child
- If any node has its left or right child empty then it will have in its respective pointer a null value
- A leaf has null value in both of its pointers

```
typedef struct
{
   DataType data; /*data of node; DataType: int, char, double..*/
   struct node *left ;   /* points to the left child */
   struct node *right;   /* points to the right child */
}node;
```

Example





Binary tree traversal

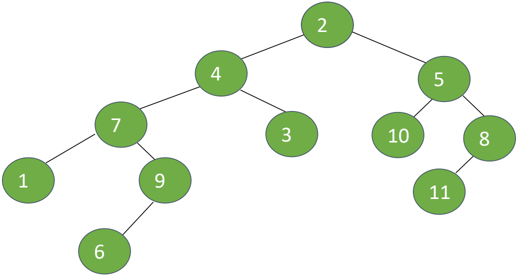
A traversal is a systematic way to visit all nodes of a graph, a binary tree in the present case

There are two very common traversals :

- Breadth First
- Depth First

Breadth First : In a breadth first traversal all of the nodes on a given level are visited and then all of the nodes on the next level are visited. Usually in a left to right fashion

On the tree below $Breadth - first - search(2)$ visits the nodes in this order : 2, 4, 5, 7, 3, 10, 8, 1, 9, 11, 6



Depth first traversals

In a depth first traversal all the nodes of a subtree are visited prior to visit another subtree

There are three common depth first traversals

- Inorder
- Preorder
- Postorder

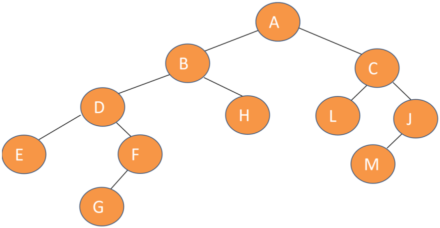
Inorder tree traversal

Traverse the left subtree ; Visit the root ; Traverse the right subtree

```
InorderTreeWalk(x)
   if x ≠ NIL
      InorderTreeWalk(x.left) ;
      print(x.key) ;
      InorderTreeWalk(x.right) ;
```

Call: InorderTreeWalk(A);

$$\underbrace{\mathbf{E\ D\ G\ F\ B\ H}}_{\text{left subtree}}\ \mathbf{A}\ \underbrace{\mathbf{L\ C\ M\ J}}_{\text{right subtree}}$$



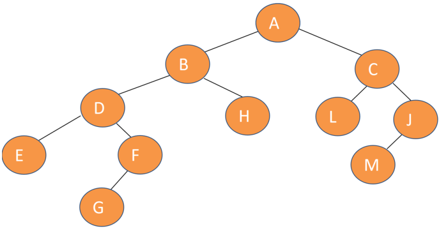
Preorder tree traversal

Visit the root ; Traverse the left subtree ; Traverse the right subtree

PreorderTreeWalk(x)
if $x \neq NIL$
  print(x.key) ;
  PreorderTreeWalk(x.left) ;
  PreorderTreeWalk(x.right) ;

`Call: PreorderTreeWalk(A);`

$$\mathbf{A}\ \underbrace{\mathbf{B\ D\ E\ F\ G\ H}}_{\text{left subtree}}\ \underbrace{\mathbf{C\ L\ J\ M}}_{\text{right subtree}}$$



Postorder tree traversal

Traverse the left subtree ; Traverse the right subtree ; Visit the root

PostorderTreeWalk(x)
if $x \neq NIL$
  PostorderTreeWalk(x.left) ;
  PostorderTreeWalk(x.right) ;
  print(x.key) ;

`Call: PostorderTreeWalk(A);`

$$\underbrace{\mathbf{E\ G\ F\ D\ H\ B}}_{\text{left subtree}}\ \underbrace{\mathbf{L\ M\ J\ C}}_{\text{right subtree}}\ \mathbf{A}$$