COMP 352 – SUMMER 2018

Tutorial Session 5

Outline

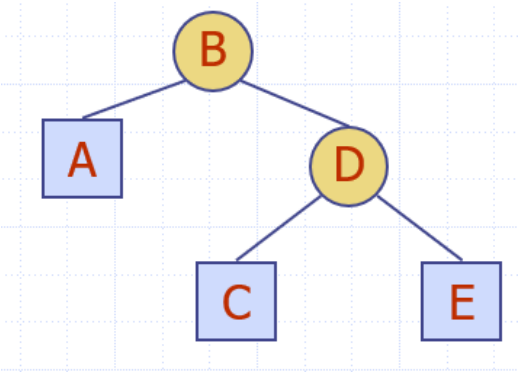
- Quick Overview on Trees
  - Definition
  - Different Implementations
- Depth and Height
- Tree Traverse Algorithm
- Problem Solving

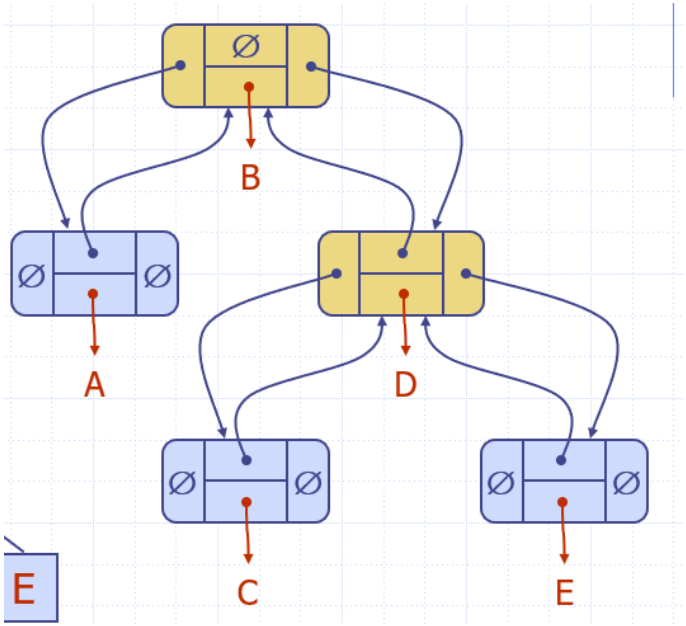
Tree definition

- A Tree is an ADT that stores its elements hierarchically.
- Each element in a tree has 1 parent element (with the exception of the root element) and 0 or more children elements.
- Two nodes that are children of the same parent are siblings. A node v is **external** if v has no children. A node v is **internal** if it has one or more children.
- External nodes are also known as **leaves**.

Implementation: linked list

- A node of binary tree is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node





Drawback?

Linked list implementation

The idea is simple. A node in the tree has:

- a data field
- a left child field with a pointer to another tree node
- a right child field with a pointer to another tree node
- optionally, a parent field with a pointer to the parent node

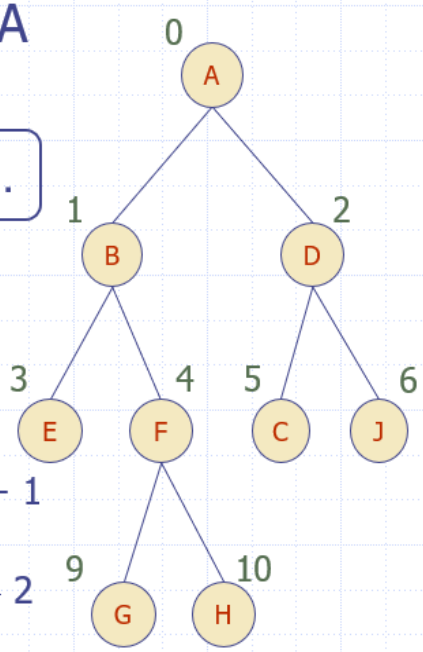
!!!!!IMPORTANT: a tree is represented by a pointer to the root not a node !!!!!!

Array implementation

- Nodes are stored in an array A

Node v is stored at A[rank(v)]

- rank(root) = 0
- if node is the left child of parent(node),
  rank(node) = 2 · rank(parent(node)) + 1
- if node is the right child of parent(node),
  rank(node) = 2· rank(parent(node)) + 2



Get the parent
index(parent) = (N-1)/2 (integer division with truncation)

Pro representation list vs array

- Pro using list representation
  - Dynamic size
  - Represent different type of tree
  - Insertions and deletions can be made directly without data movements
  - Node can be placed anywhere in the memory
- Pro using array
  - Easy to understand
  - Easy to move from parent to child
  - Programming is easy

Cons representation list vs array

- Cons list:
  - It is difficult to understand
  - Accessing a particular node is difficult
- Cons array:
  - A lot of movement if you want to insert delete a node
  - Lot a memory wasted if you do not use all the allocated memory

DEPTH DEFINITION

The **depth** of a node v is the number of ancestors of v, excluding v itself. The depth of a node v can also be recursively defined as follows:

- If v is the root, then the depth of v is 0
- Otherwise, the depth of v is one plus the depth of the parent of v

Algorithm depth(T , v):
  if v is the root of T **then**
    return 0
else
    return 1 + depth(T , w),
      where w is the parent of v in T

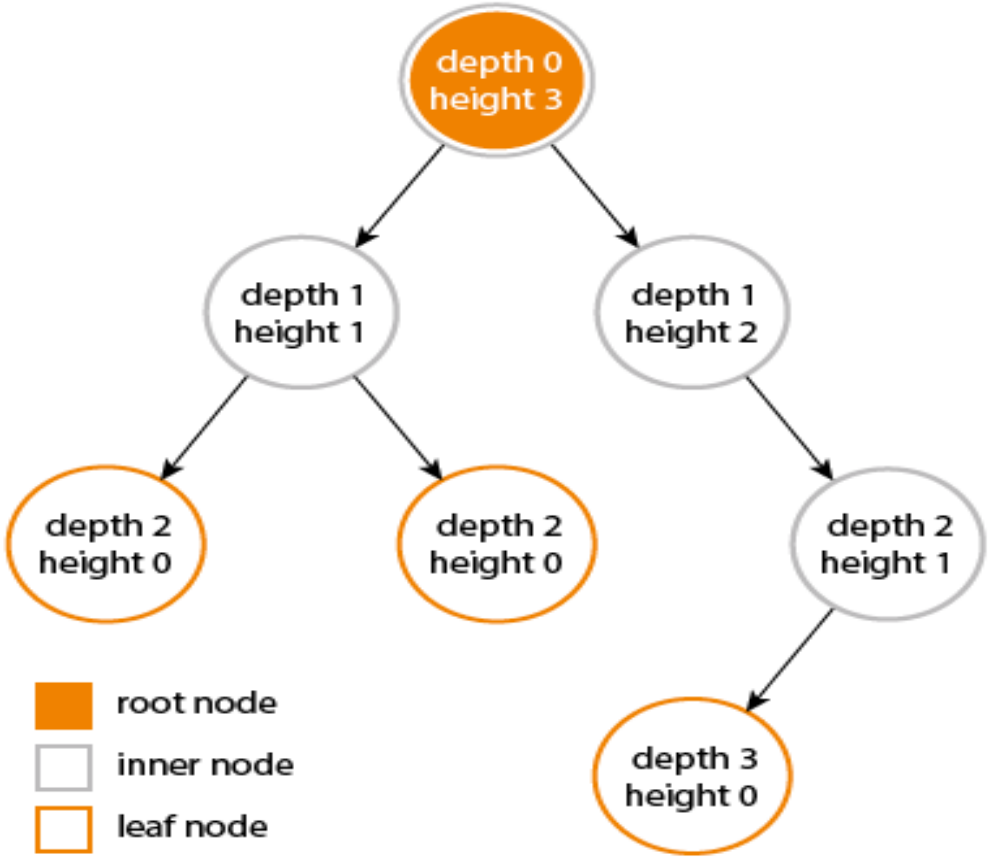
HEIGHT DEFINITION

The height of a node is the number of edges on the *longest path* from the node to a leaf. Recursive definition:

- A leaf node will have a height of 0.
- If v is an external node, then the height of v is 0
- Otherwise, the height of v is one plus the maximum height of a child of v

```
Algorithm height(T , v):
  if v is an external node in T then
    return 0
  else
    h ← 0
    for each child w of v in T do
      h ← max(h, height(T , w))
    return 1 + h
```

Difference height vs depth



Preorder traversal

In a **preorder traversal** of a tree T, the root of T is visited first and then the subtrees rooted at its children are traversed recursively.

Algorithm preorder(T , v):
perform the "visit" action for node v
for each child w of v in T **do**
    preorder(T , w)
{recursively traverse the subtree rooted at w}

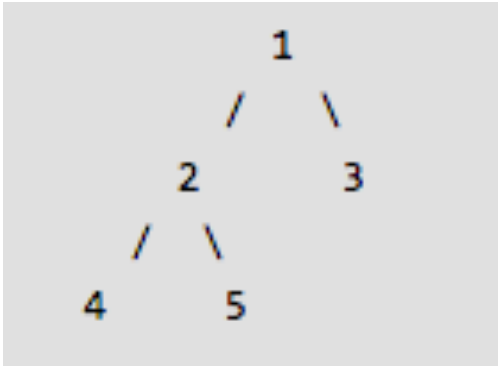
Post-order traversal

A **post-order traversal** recursively traverses the subtrees rooted at the children of the root first, and then visits the root.

Algorithm postorder(T, v):
for each child w of v in T **do**
    postorder(T, w)
{recursively traverse the subtree rooted at w}
perform the "visit" action for node v

Exercise 1

Given a binary Tree, print the nodes in in-order fashion without recursion



In order : 4 2 5 1 3

EXERCISE 2:

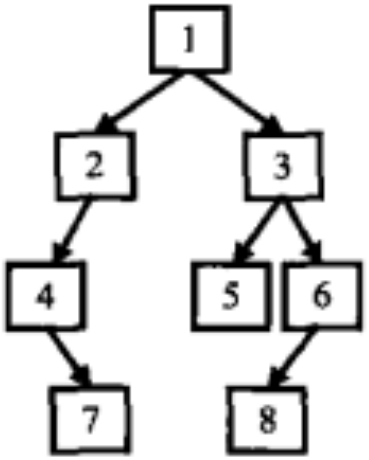
Given preorder and inorder traversal of a tree, construct the binary tree. You may assume that duplicates do not exist in the tree.

For example, given

preorder = [1, 2, 4, 7, 3, 5, 6, 8]

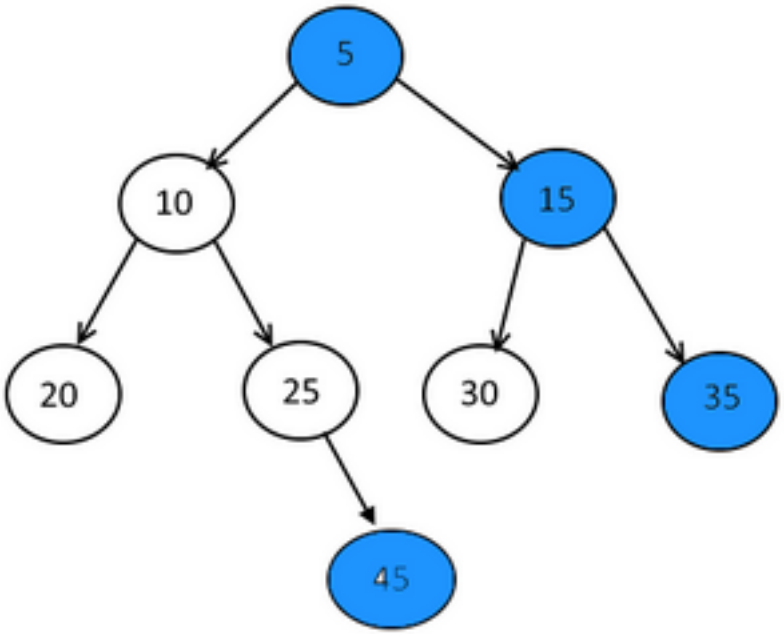
inorder = [4, 7, 2, 1, 5, 3, 8, 6]

you get this binary tree.



Exercise 3:

Given a Binary Tree, print right view of it. Right view of a Binary Tree is set of nodes visible when tree is visited from right side.



Right view: 5 15 35 45

Concordia UNIVERSITY