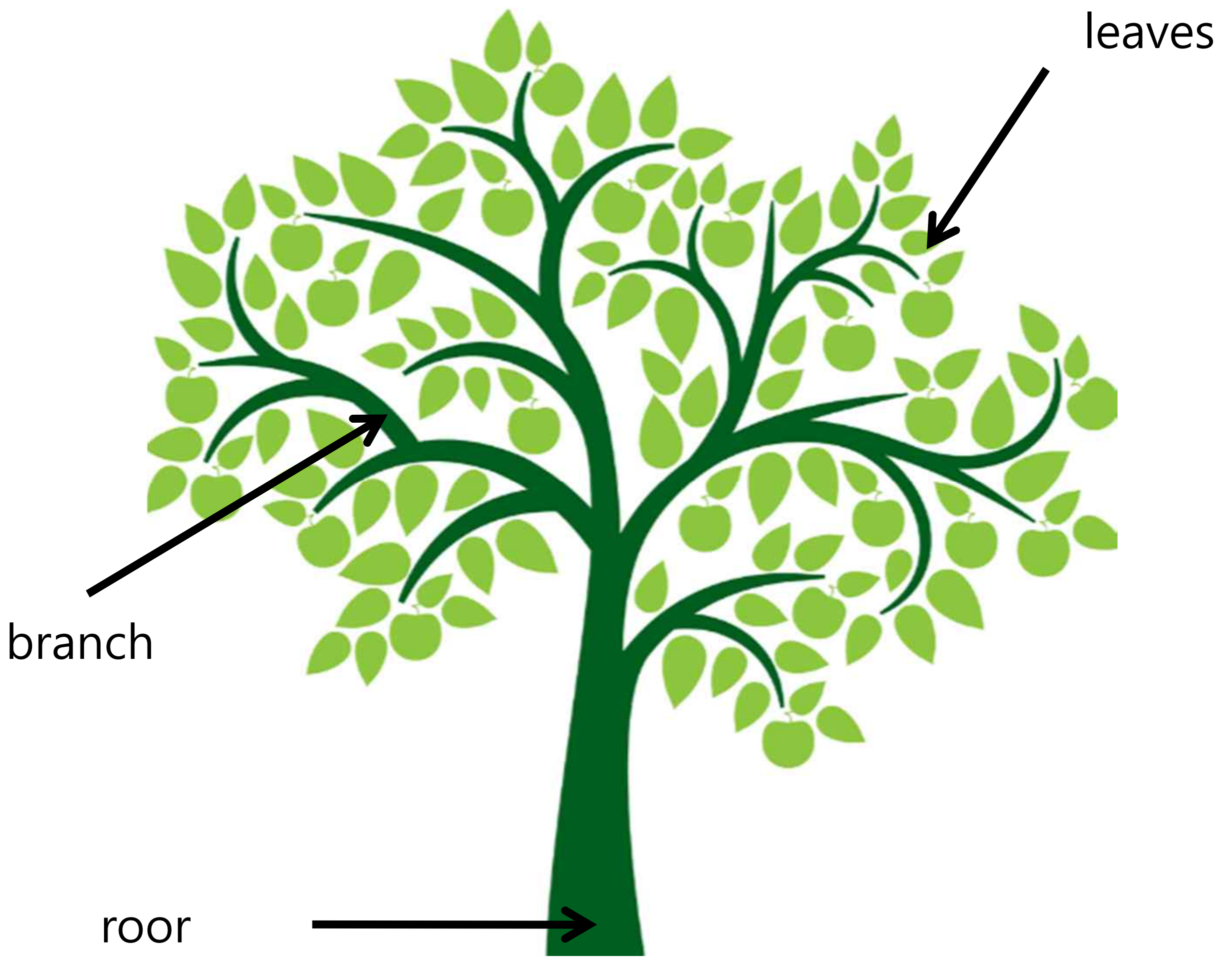
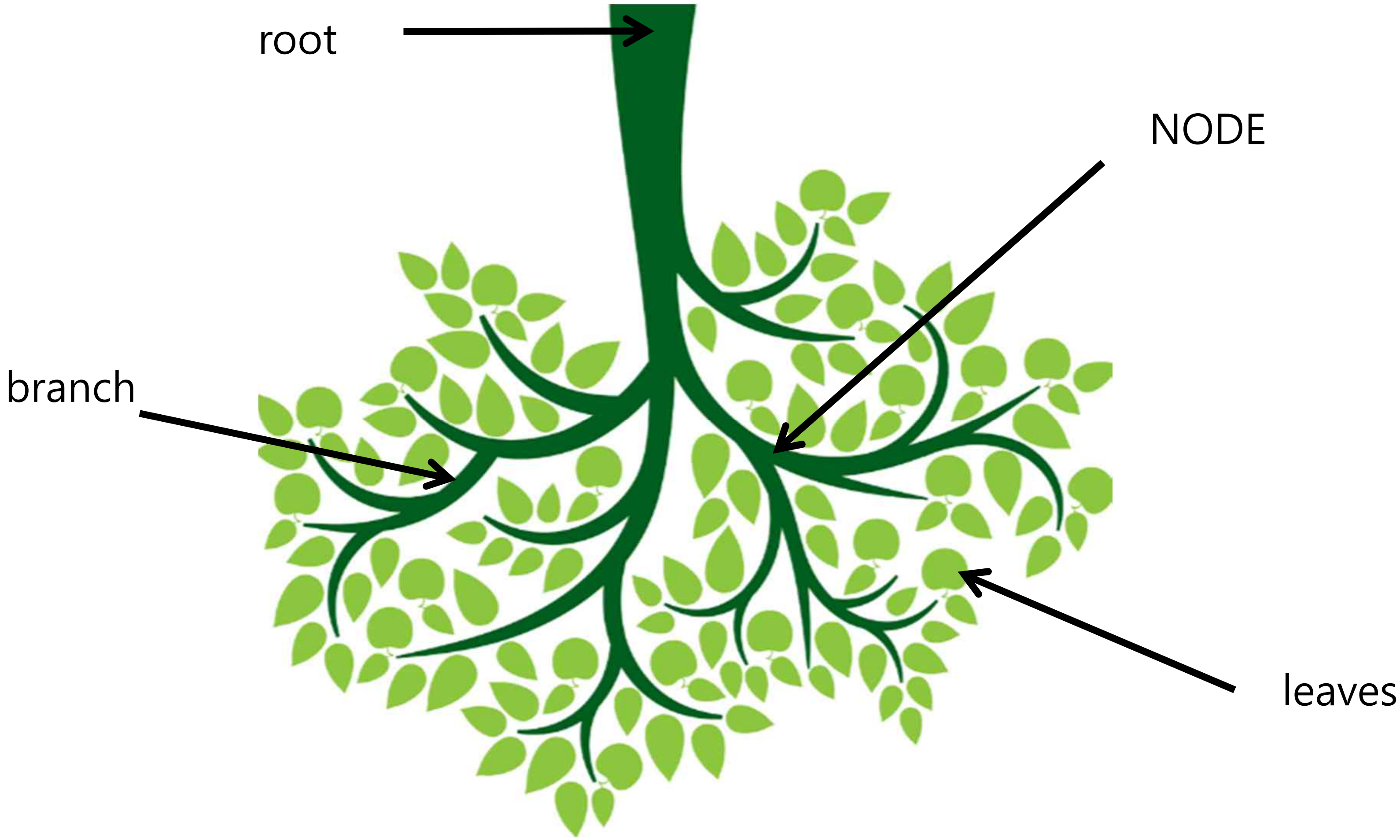
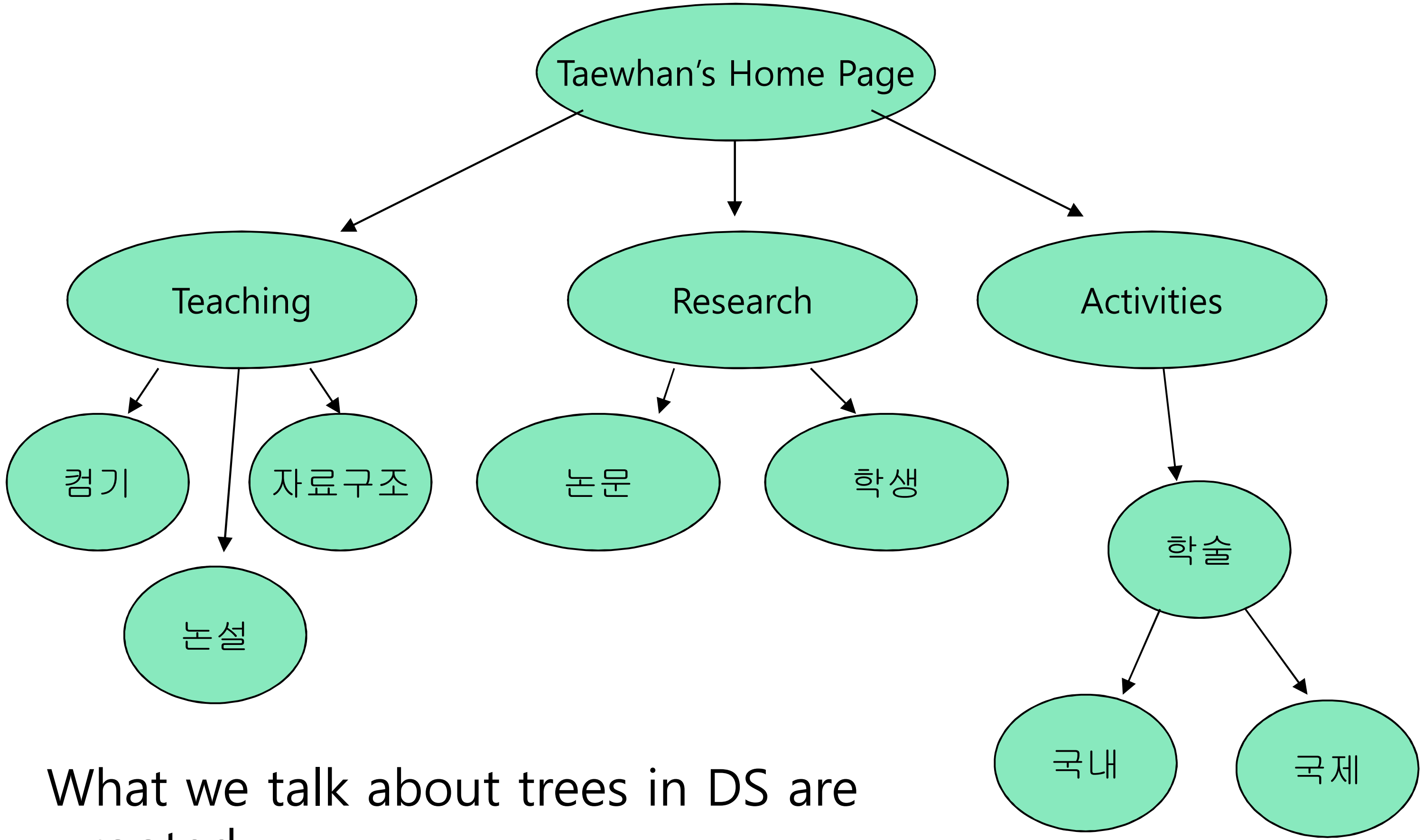
Look at “Trees” Outside.



Upside down





What we talk about trees in DS are
- rooted,
- ordered,
- directed

A (unrooted) tree is acyclic and connected graph

Graph-theoretic definition of a (Rooted) Tree:

A tree is a graph for which there exists a node, called root, such that:

-- for any node x, there exists exactly one path from the root to x

Recursive Definition of a (Rooted) Tree:

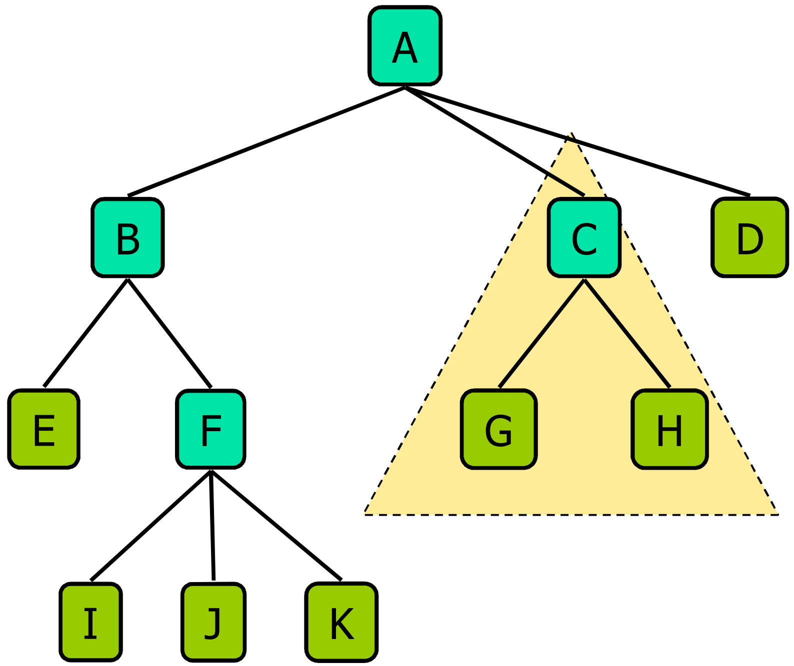
A tree is either:

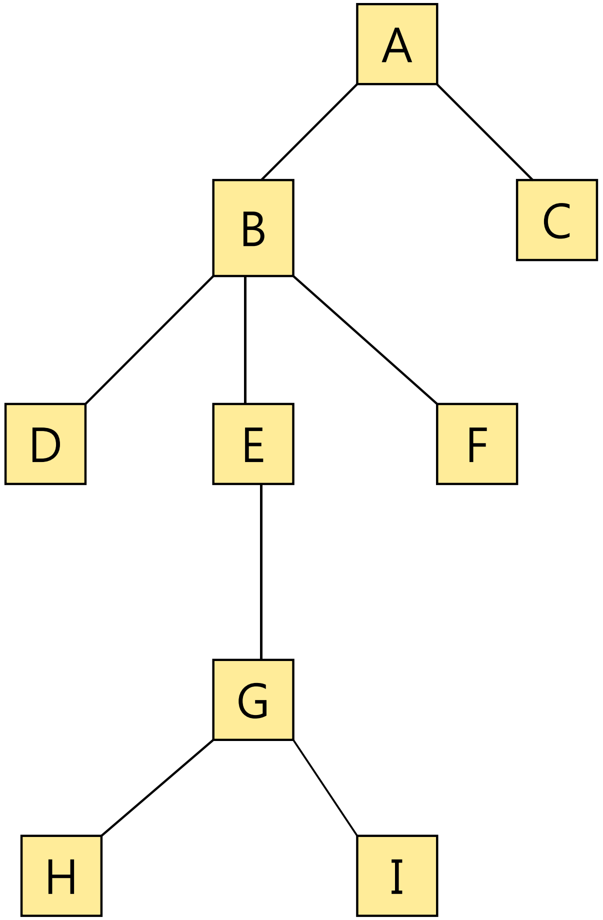
a. empty, or

b. it has a node called the root, followed by zero or more trees called subtrees

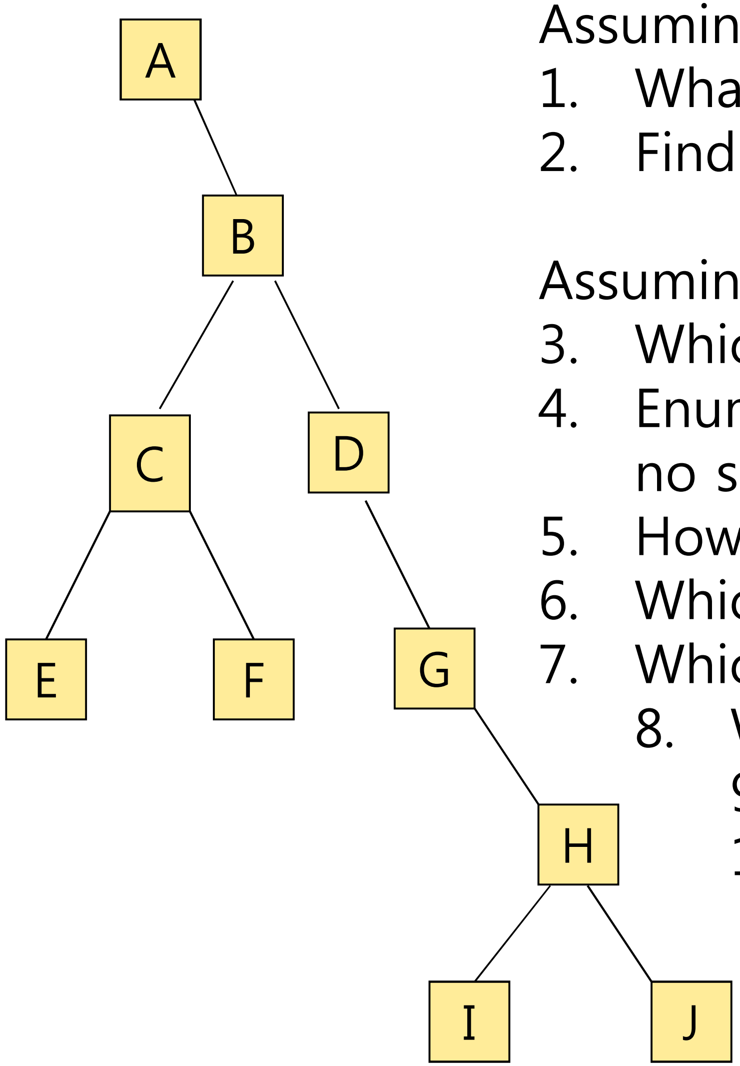
Terminology

- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Degree** of a tree: the maximum number of its node.
- **Subtree**: tree consisting of a node and its descendants





Property	Value
Number of nodes	
Height	
Root Node	
Leaves	
Internal nodes	
Ancestors of H	
Descendants of B	
Siblings of E	
Right subtree of A	
Degree of this tree	



Assuming the tree is a non-rooted (general) tree,

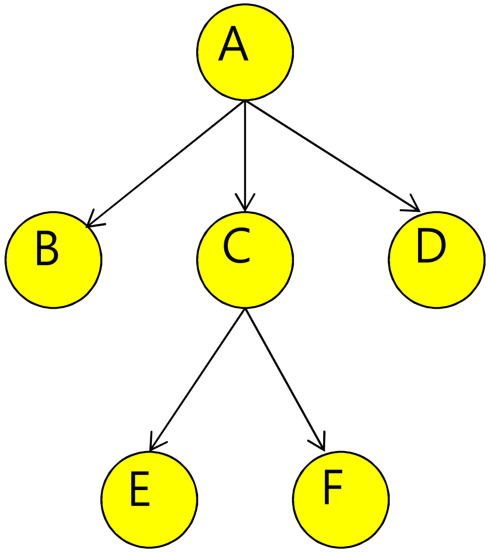
1. What is the longest path in the tree?
2. Find an edge that is not on the longest path.

Assuming the tree is a rooted and ordered tree,

3. Which node you guess root?
4. Enumerate the nodes that have a parent but no sibling.
5. How many parents does each node have?
6. Which node has the fewest children?
7. Which node has the most ancestors?
8. Which node has the most descendants?
9. List all the nodes in B's left subtree.
10. List all the leaves in the tree.

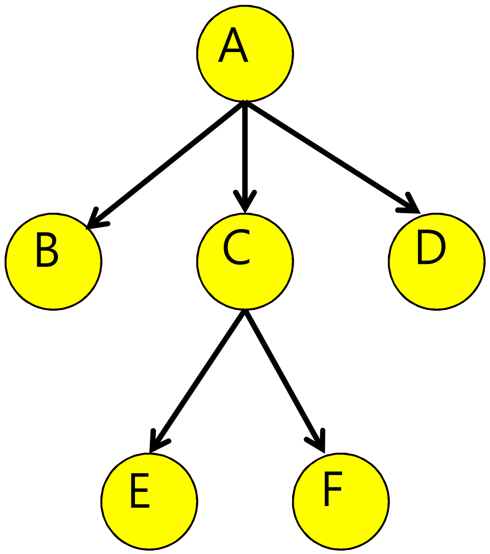
Implementation of Trees

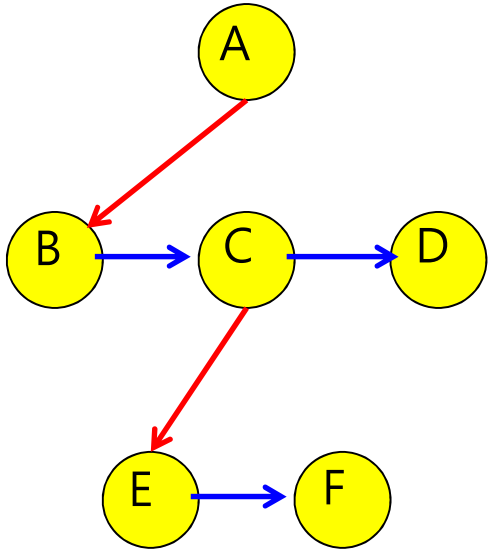
- Obvious pointer-based implementation



1. ______________ for trees with varying degrees.
2. ______________ for trees with a fixed degree of nodes

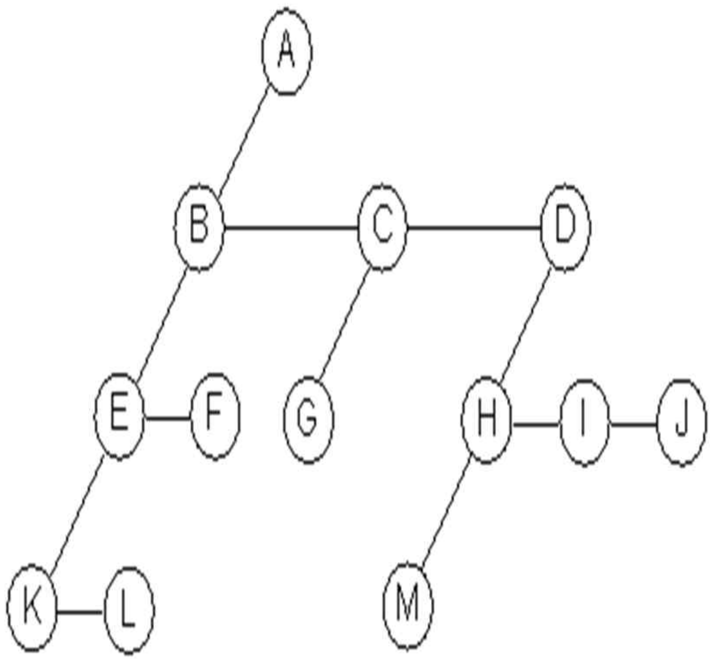
Left child – right sibling representation

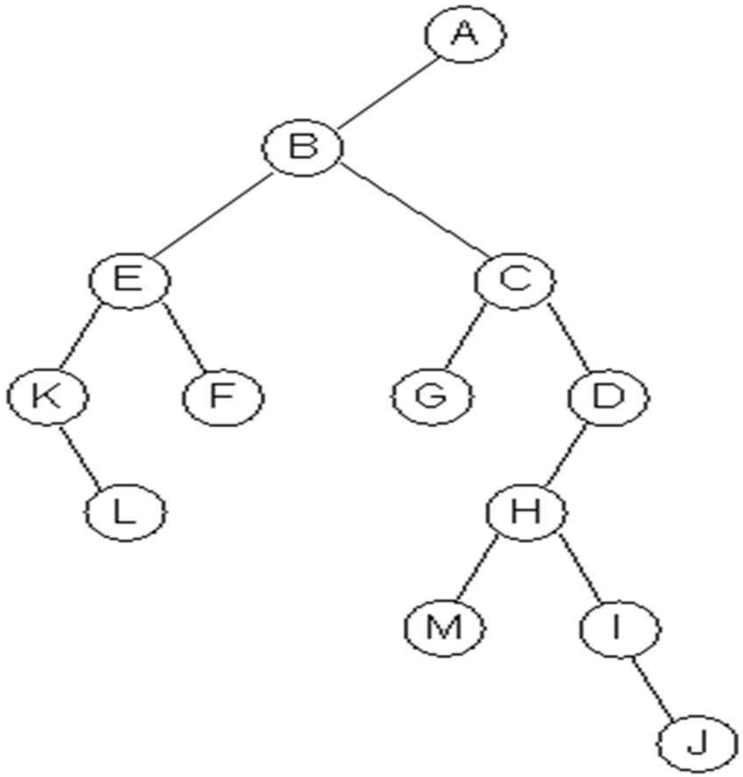




1. Efficient usage of memory for any tree
2. ____________ may be limited.

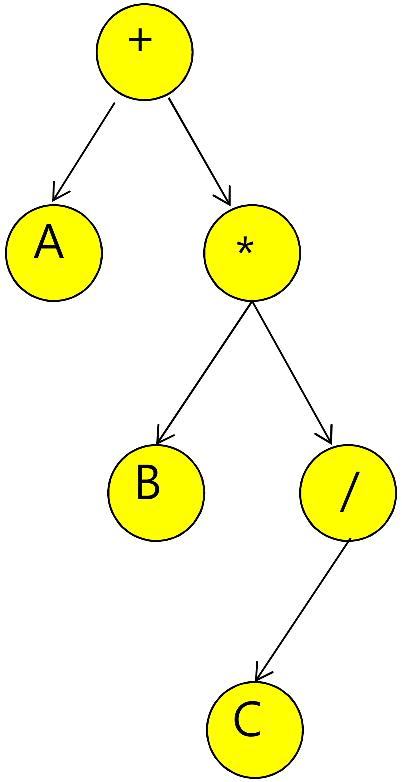
Left child – right child representation





1. _______ tree(s) can be repres
2. Efficient usage of memory.
3. But, the implication on the original tree may be lost. Unique?

Binary tree



Recursive definition:

A binary tree T is either

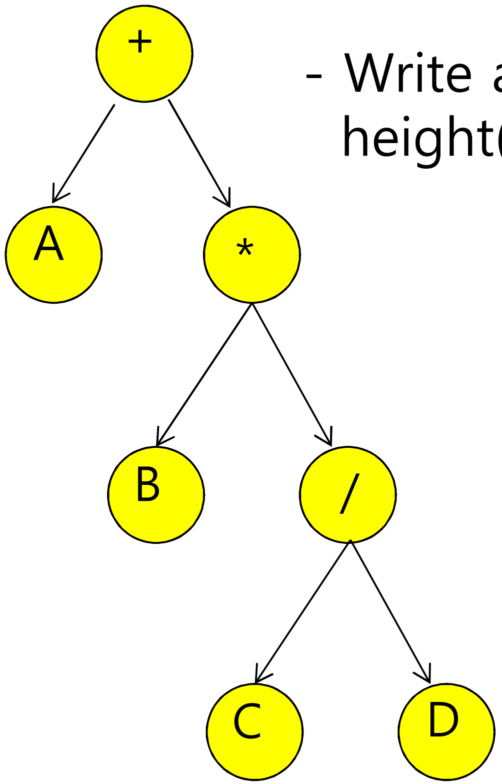
1. ____________________, or
2. ____.

height(T)

- One of the most frequently used functions on a binary tree

- Returns the length of the longest path from root to a leaf

- Write a recursive definition of the height of binary tree T, height(T):

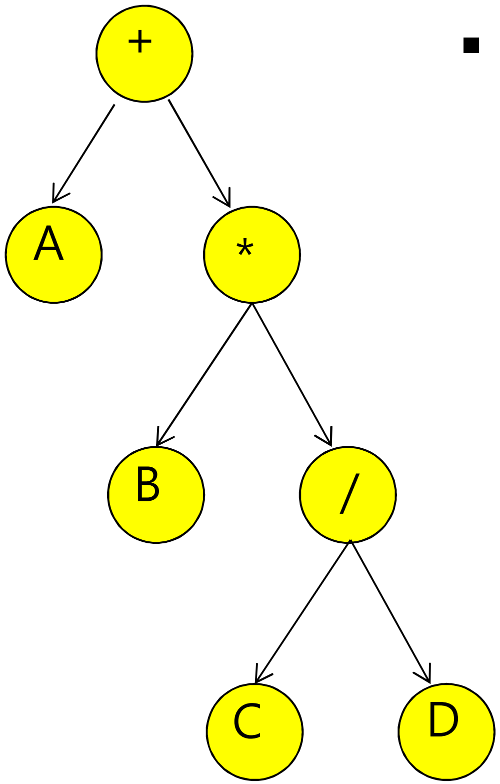


Full Binary tree

- A tree in which every node has exactly 0 or 2 children

- F is a full binary tree if and only if:

  - F = { }, or
  - F = { r, T_L,T_R }, and



Perfect Binary tree

Perfect binary tree of height h, P_h:

- P_{-1} is an empty tree, and
- if $h > -1$, then P_h is { r, T_L,T_R }, where T_L and T_R are P_{h-1}

P_0:

P_1:

P_2:

How many nodes in a perfect binary tree of height h?

Complete Binary tree

for any level k in [0, h-1], level k has 2^k nodes, and on level h, all nodes are “pushed to the left”.

Complete tree of height h, C_h:

1. if $h = -1$, then C_h is { }

2. if $h > -1$, then C_h is { r, T_L,T_R }, and either:

   T_L is __________ and T_R is __________

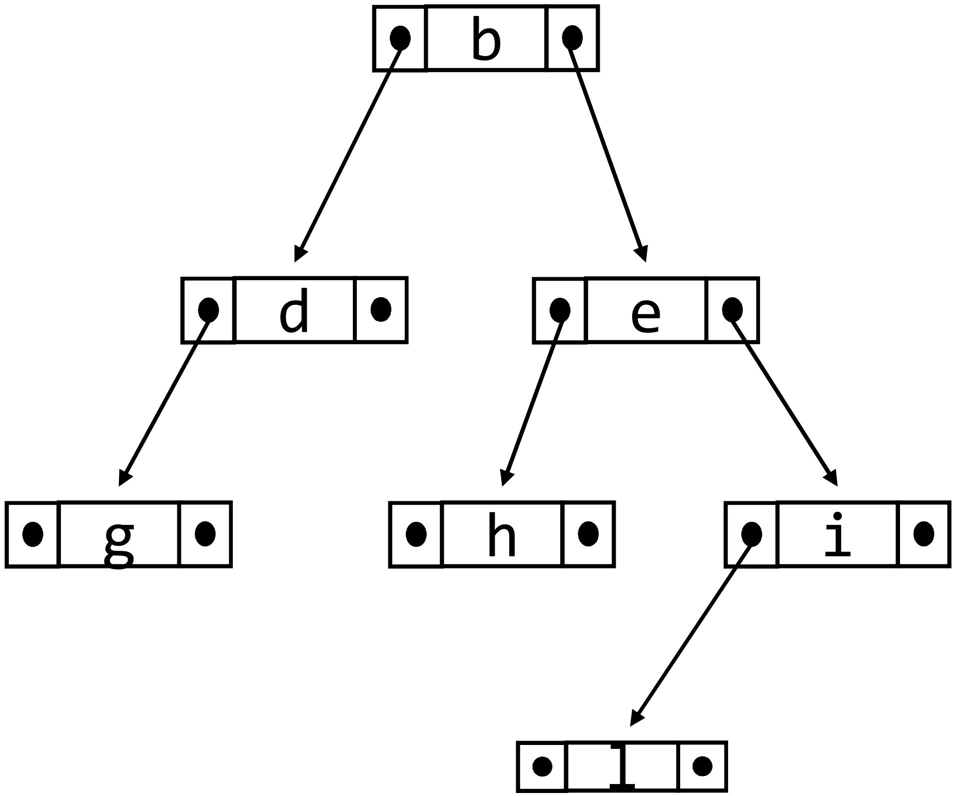
   OR

   T_L is __________ and T_R is __________

Is every full tree complete ?

Is every complete tree full ?

Rooted, directed, ordered binary tree




```
template <class T>
class tree {
private:
   class Node {
   public:
      T data;
      Node *left, *right;
      Node (int d, Node *left, Node *right) {
          this->data = d;
          this->left = left;
          this->right = right;
      }
      ~Node ( ) {
           if (this->left) delete this->left;
           if (this ->right) delete this->right;
       }
      Node *root;
   public:
         .....

      }
```

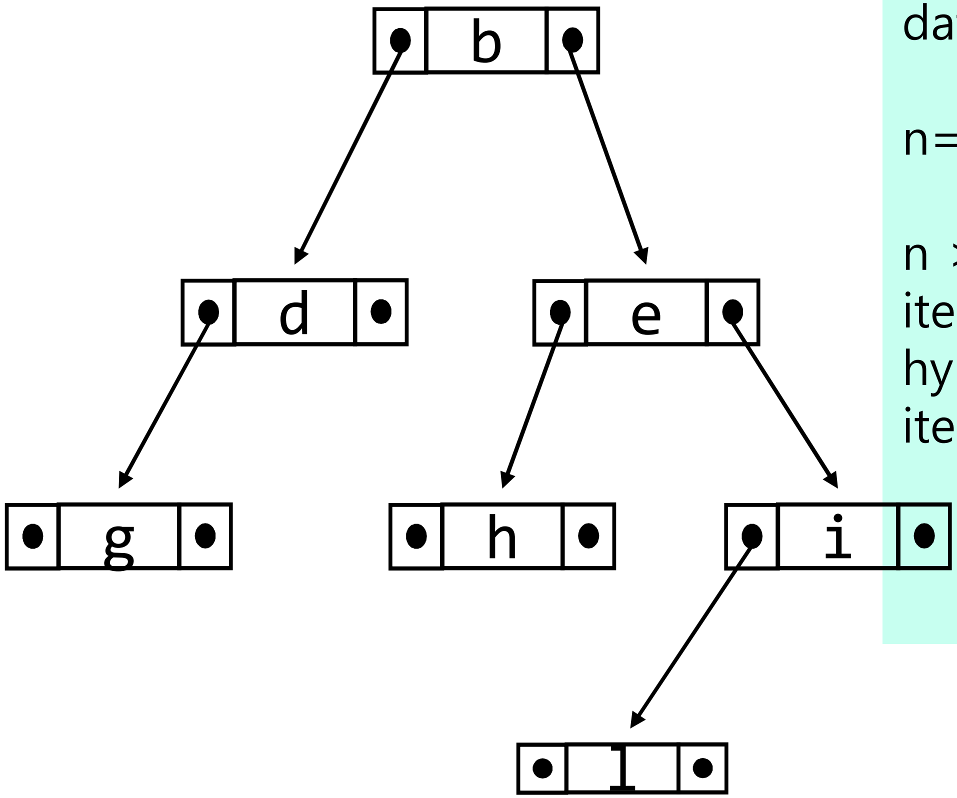
Tree ADT:

traverse

insert

remove

Theorem: if there are n data items in a binary tree, then there are _________ null pointers.



Proof.

Consider an arbitrary binary tree T with n data items.

n= 0: we represent T with 1 null pointer.

n > 0: T = {r, TL, TR} with size(TL) = a items, size(TR) = b items. By induction hypothesis that for all k < n, a BT of k items has k+1 null pointers, we know that

TL has a+1 nulls and TR has b+1 nulls. Thus, T has a+b+2 = n+1 nulls.