Data Structure

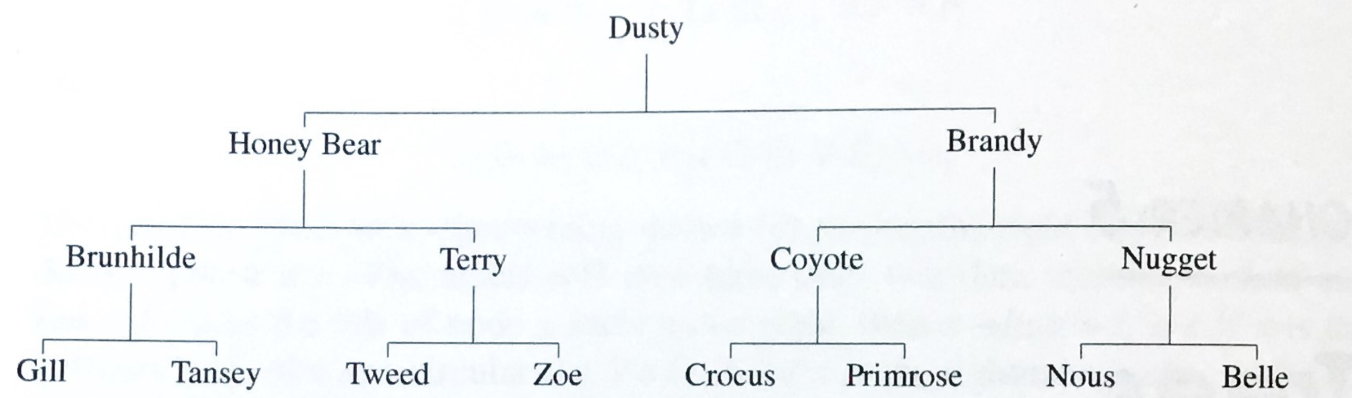
Trees

Shin Hong

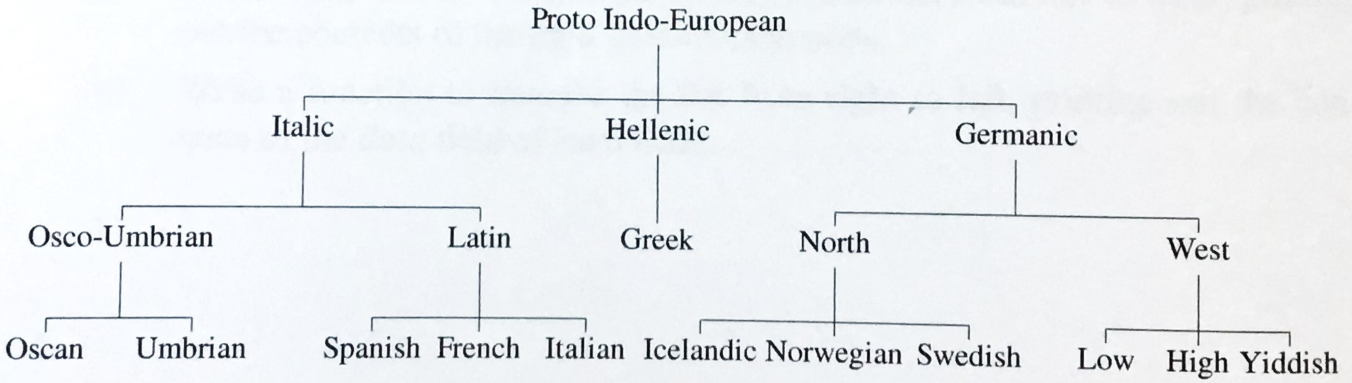
28 Apr 2023

DS&A. Chapter 7. Trees

Motivation



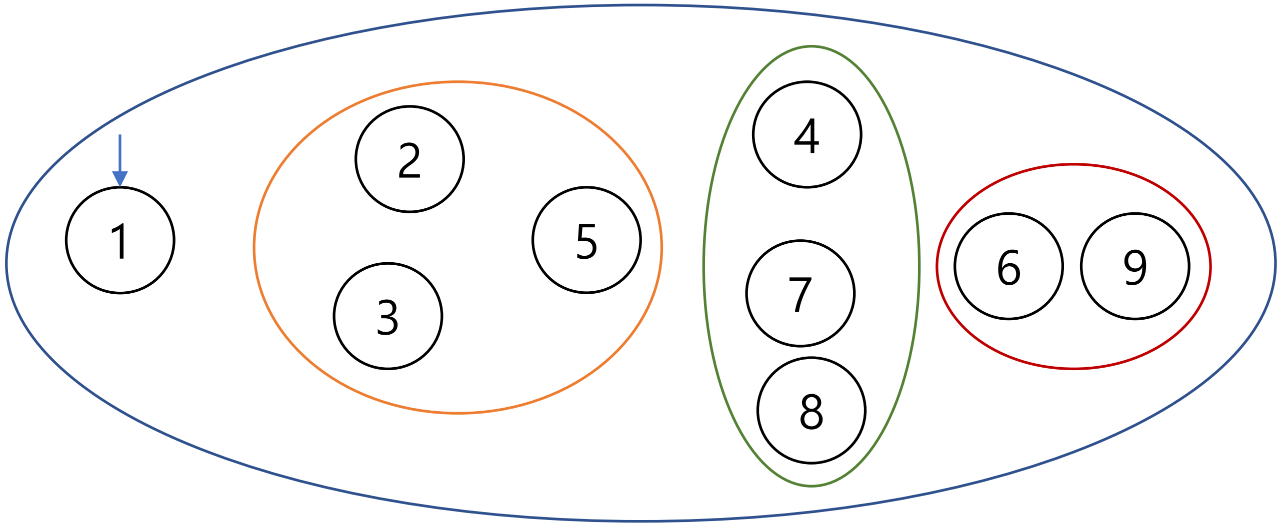
(a) Pedigree



(b) Lineal

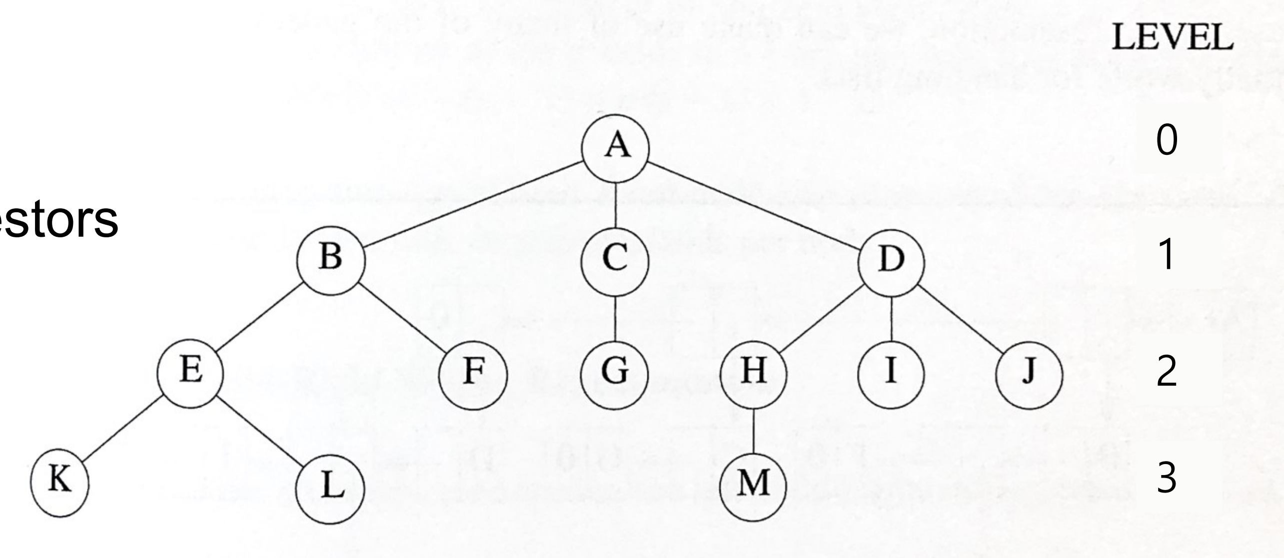
Tree

- A tree is a finite set of one or more nodes such that:
  - there exists a specifically designated node called the *root*, and
  - the remaining nodes are partitioned into disjoint sets T_1, T_2, ⋯, T_n, where each of these sets is a tree (subtree)



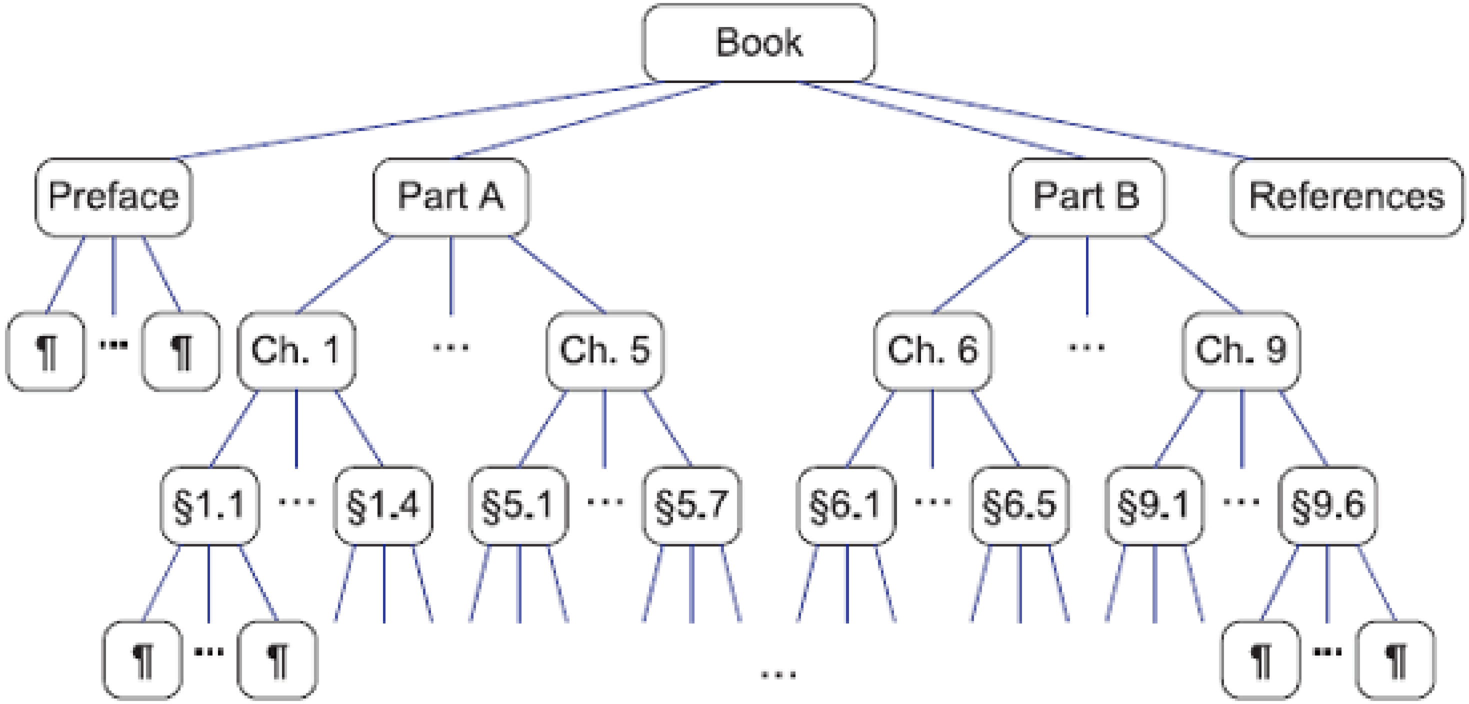
Terminologies

- Node: the item of information
- Branch (edge): links between two nodes (a parent and a child)
- Degree of a node: the number of subtrees
  - Degree of a tree
- Leaf (terminal, external) node: node with degree zero
  - non-terminal (internal) nodes
- Children, Parent, Siblings, Ancestors
- Level of a node: the number of the ancestors
  - depth of a node
- Height of a tree



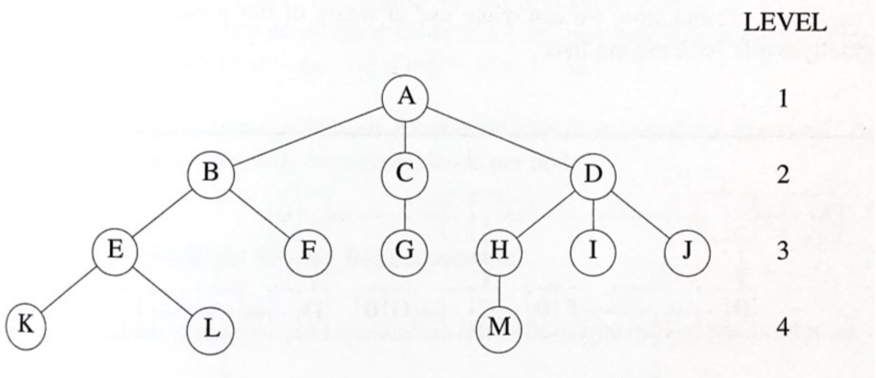
Ordered Tree

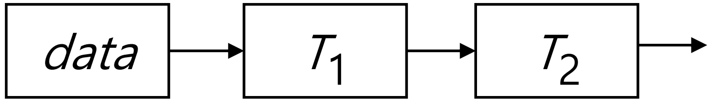
- A tree is ordered if there is a linear ordering defined for children of each node
  - an ordering determines how the tree is used

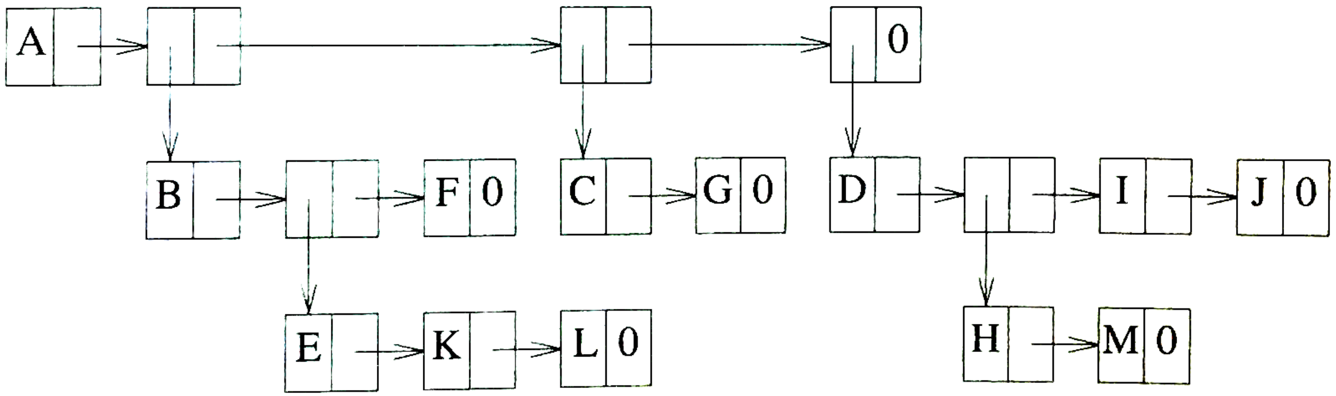


Tree Representation

- List representation
  - *Data, or* (*Data* (T_1, T_2, ... , T_N))
  - E.g., $(A(B(E(K,L),F),C(G),D(H(M),I,J)))$



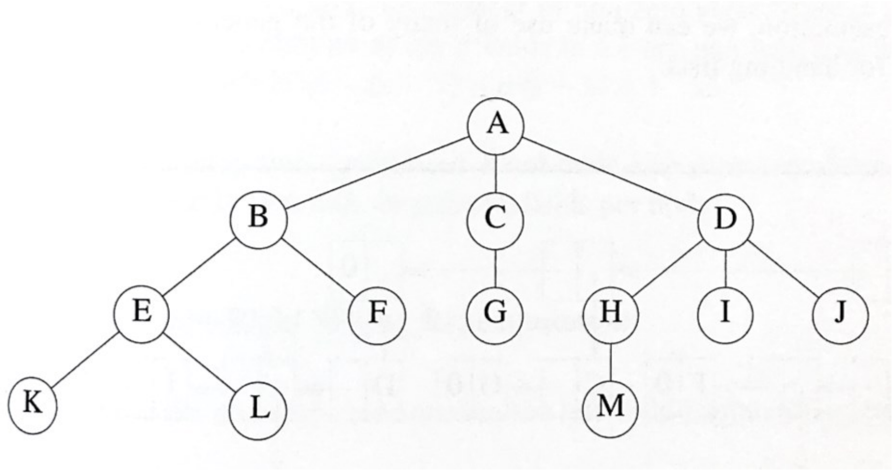


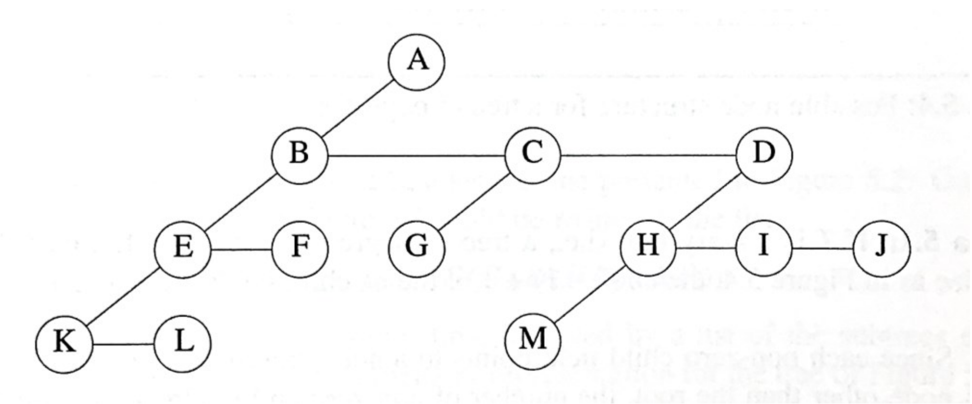


Tree Representation

- Left child-right sibling representation

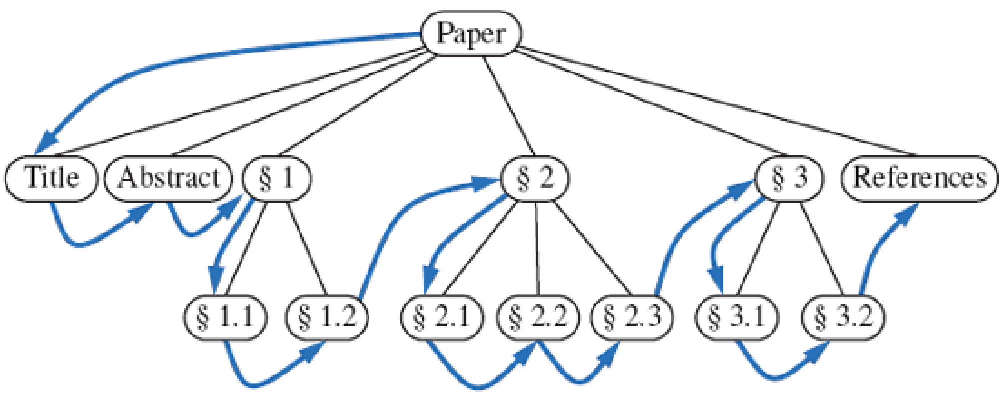
<table>
<tr><td colspan="2" align="center">data</td></tr>
<tr><td align="center">child</td><td align="center">sibling</td></tr>
</table>

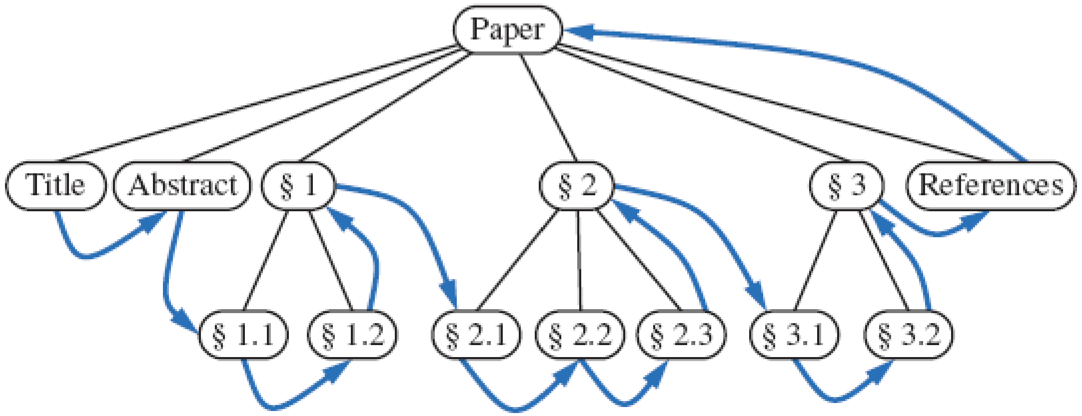




Tree Traversal

- A traversal of a tree is a systematic way of accessing (visiting) all nodes
- preorder traversal: visit the root node first, and then visit the sub-trees recursively
- postorder traversal: recursively visit the sub-tree first, and then visit the root node



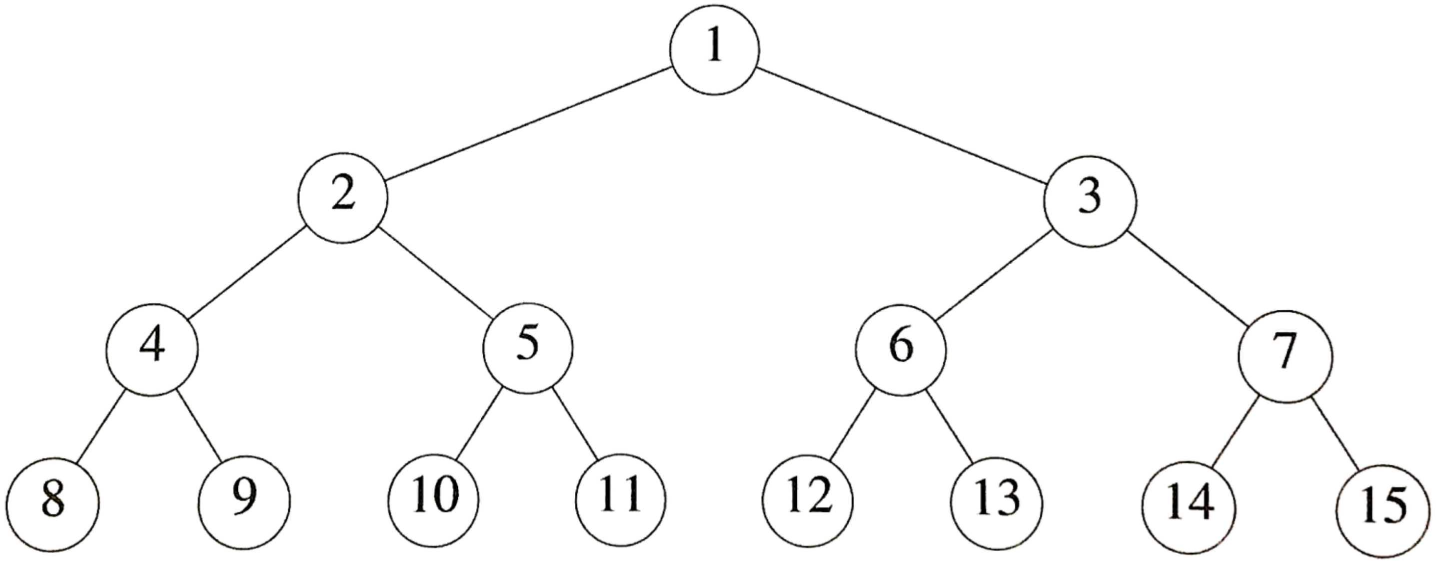


Binary Trees

- A binary tree is an ordered tree in which every node has at most two children
  - each child node is labeled as either left or right
  - a left child precedes a right child in the ordering of children
- A binary tree is empty, or it consists of (1) a root node, (2) a binary tree as a left subtree, and (3) a binary tree as a right subtree
- a binary tree is *proper* iff each node has either zero child or two children

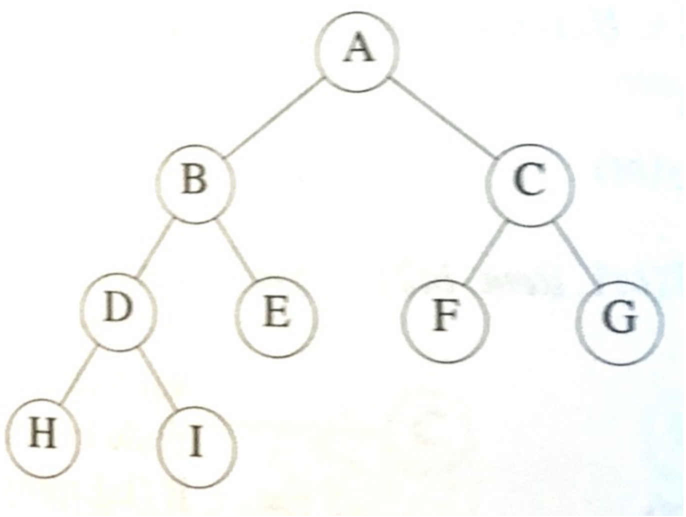
Terminologies (1/2)

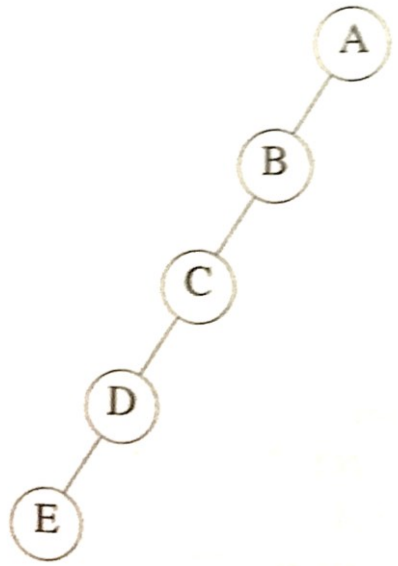
- A **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes



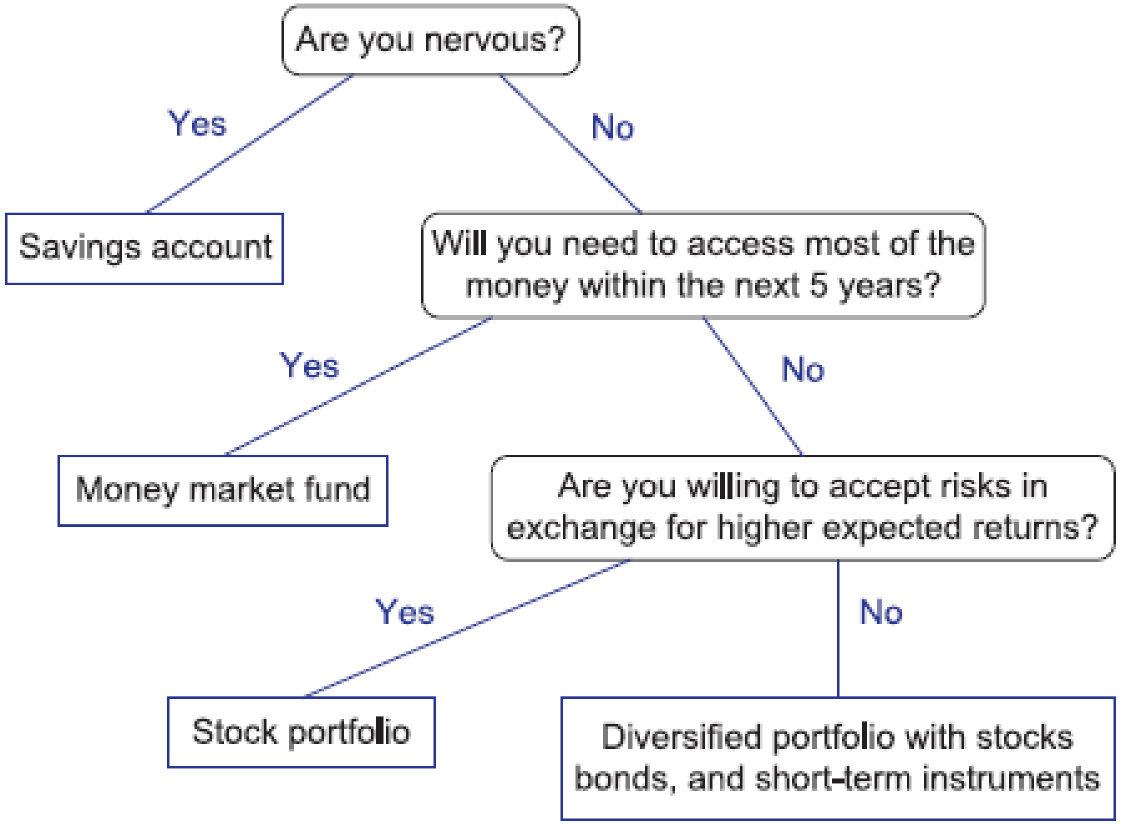
Terminologies (2/2)

- A binary tree with n nodes and depth k is **complete** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k
- The hiehgt of a complete binary tree with n nodes is $\lceil \log_2(n+1) \rceil$
- A tree is called skewed if nodes are skewed at left or right subtrees

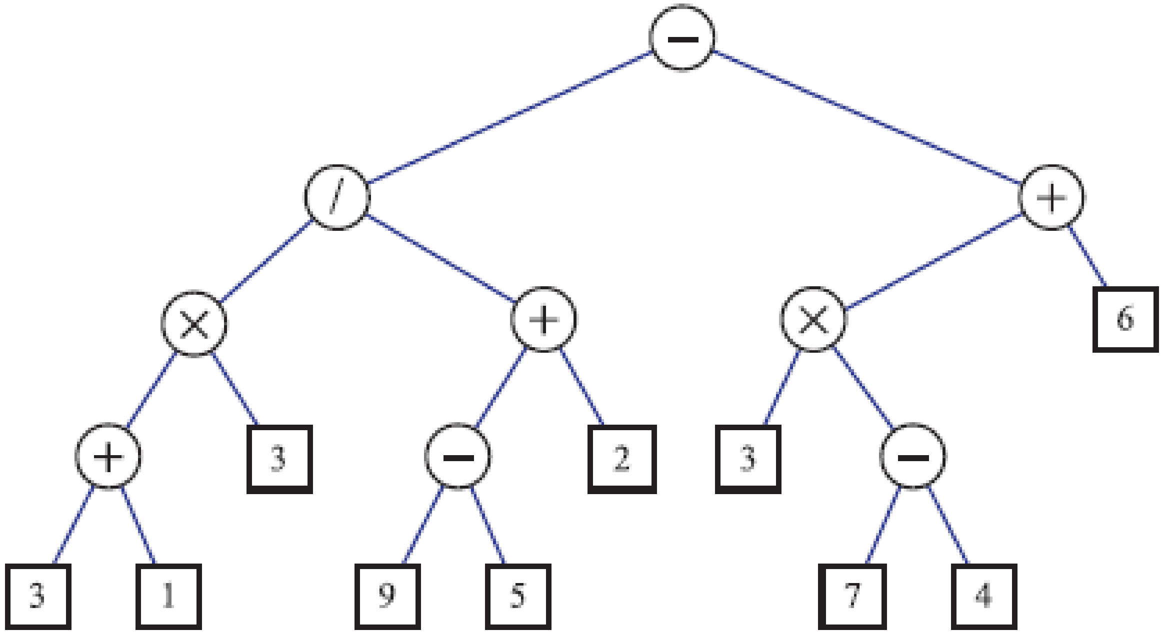




Examples



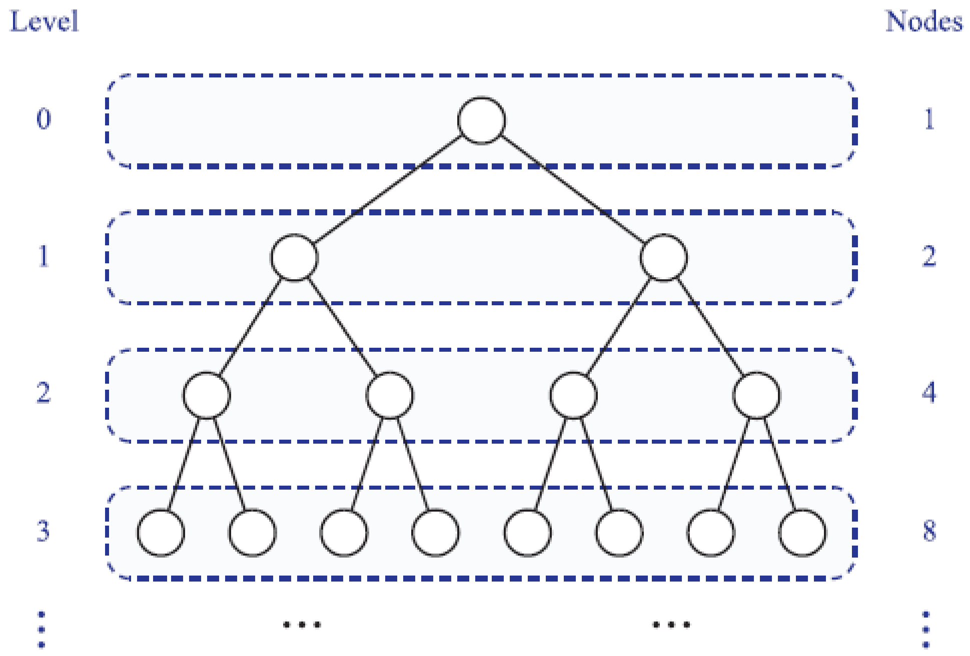
Are you nervous?
Yes
No
Savings account
Will you need to access most of the money within the next 5 years?
Yes
No
Money market fund
Are you willing to accept risks in exchange for higher expected returns?
Yes
No
Stock portfolio
Diversified portfolio with stocks bonds, and short-term instruments



−
/
+
×
+
×
6
+
3
−
2
3
−
3
1
9
5
7
4

Properties of Binary Trees

- there are at most 2^d nodes at level d
  - the root node is at level 0
- the relation of height h and the number of nodes n
  - $h+1 \le \; n \; \le 2^{h+1}-1$
  - $\log(n+1)-1 \; \le h \; \le \; n-1$
  - $1 \le \; n_E \; \le 2^h$ where n_E is the number of external nodes
  - $h \le \; n_I \; \le 2^h-1$ where n_I is the number of internal nodes



Properties of Proper Binary Trees

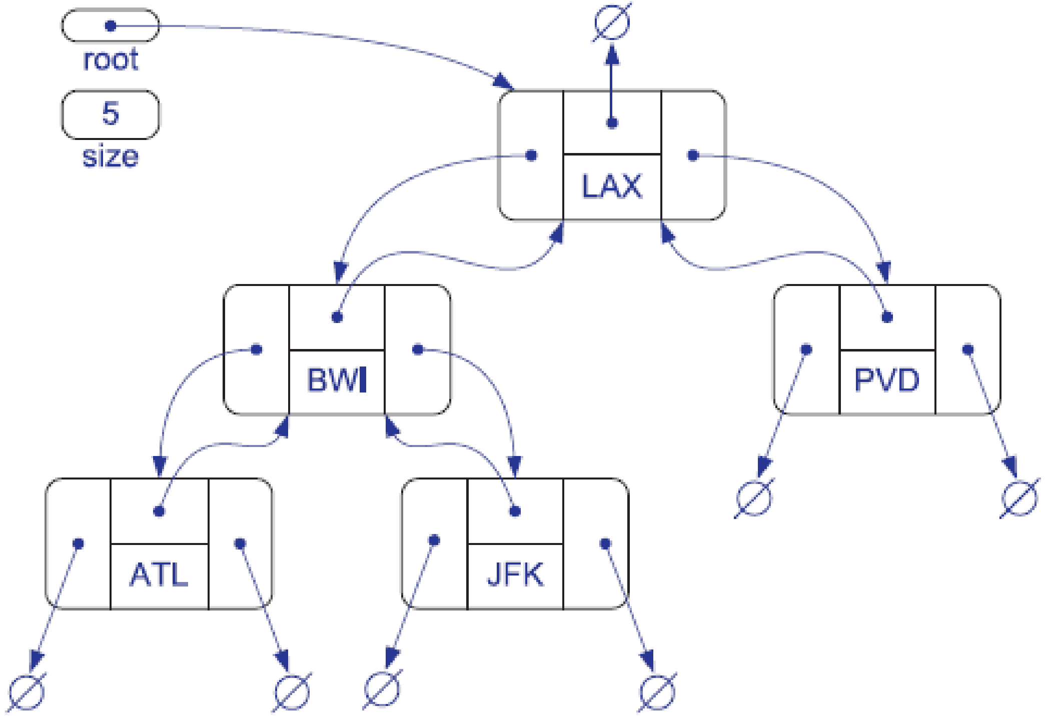
- the relation of height h and the number of nodes n
  - $2h + 1 \leq \; n \; \leq 2^{h+1} - 1$
  - $\log(n+1) - 1 \; \leq h \; \leq (n-1)/2$
  - $h + 1 \leq \; n_E \; \leq 2^h$ where n_E is the number of external nodes
  - $h \leq \; n_I \; \leq 2^h - 1$ where n_I is the number of internal nodes

- The number of external nodes is one more than the number of internal node for a non-empty proper binary tree

Linked Structure of Binary Tree

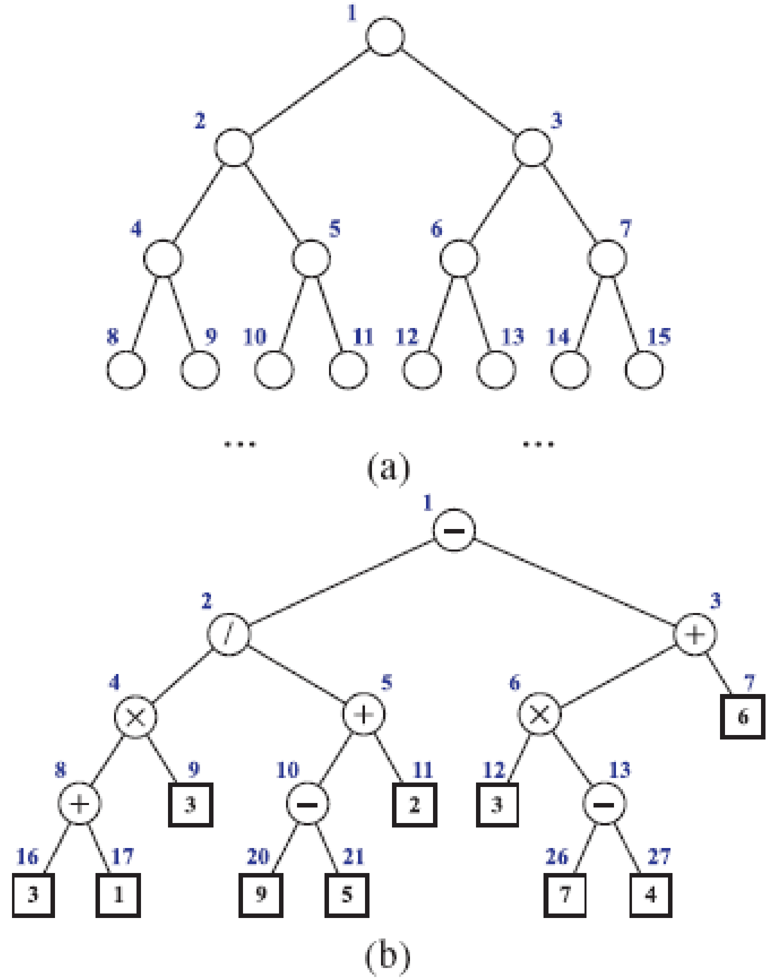
- Each node stores an associated element, and pointers to its parent and two children

- A tree has a pointer to the root node



Array-based Structure for Binary Tree

- For every node v, let $f(v)$ be the integer defined as follows:
  - if v is the root, then $f(v)$ = 1
  - if v is the left child of node u, then $f(v)$ = 2 $f(u)$
  - if v is the right child of node u, then $f(v)$ = 2 $f(u)$ + 1



Traversals of Binary Tree (1/2)

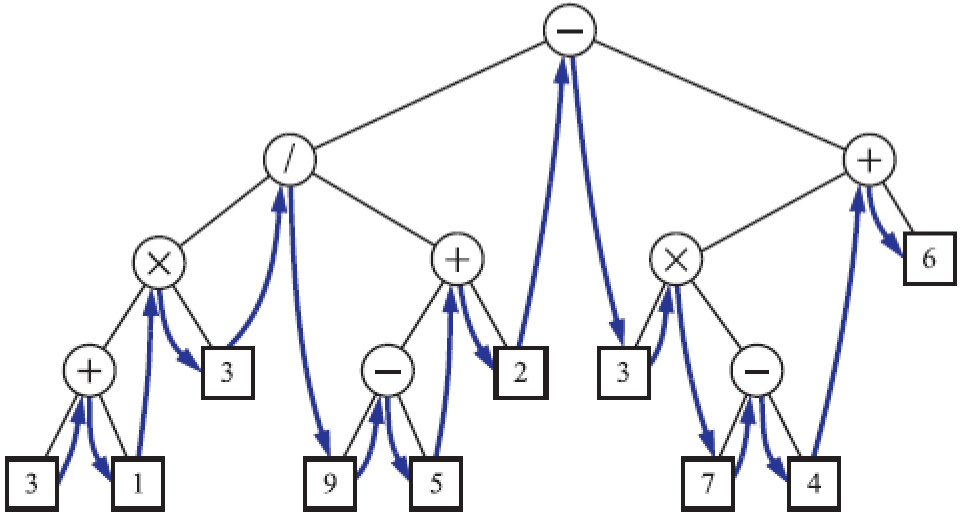
- Preorder traversal
- Postorder traversal
  - ex. evaluating expression

```
Algorithm evaluateExpression(T, p):
  if p is an internal node then
    x ← evaluateExpression(T, p.left())
    y ← evaluateExpression(T, p.right())
    Let • be the operator associated with p
    return x • y
  else
    return the value stored at p
```

Traversals of Binary Tree (2/2)

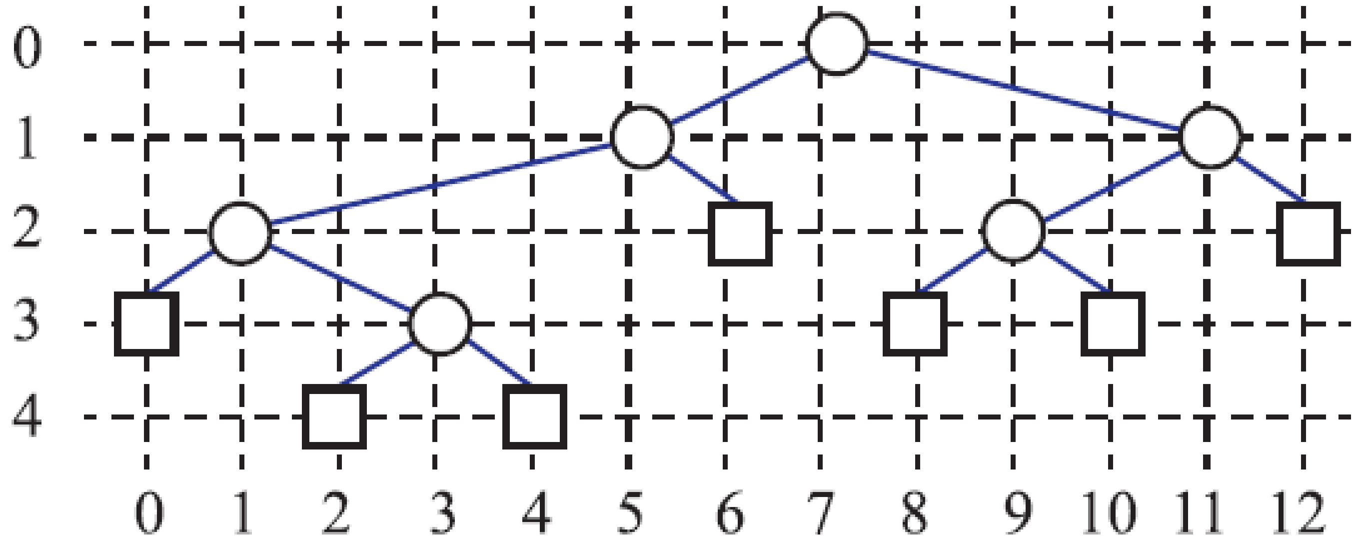
- Inorder traversal

```
Algorithm inorder(T, p):
  if p is an internal node then
    inorder(T, p.left())          {recursively traverse left subtree}
  perform the "visit" action for node p
  if p is an internal node then
    inorder(T, p.right ())        {recursively traverse right subtree}
```



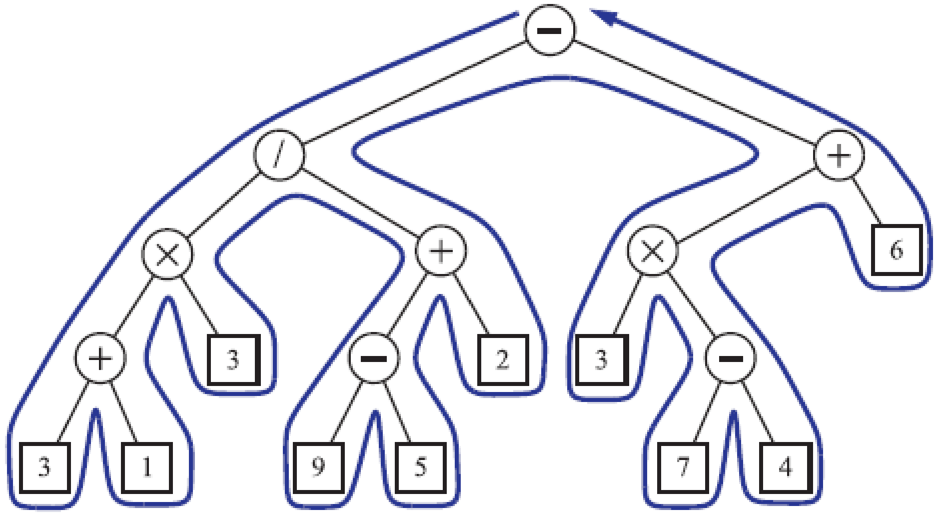
Using Inorder Traversal for Tree Drawing

- $x(p)$ is the number of nodes visited before p in T
- $y(p)$ is the depth p in T



Euler Tour Traversal

- The Euler tour traversal walks around a tree T by encountering each node three times as follows:
  - on the left (preorder)
  - from the below (inorder)
  - on the right (postorder)




```
Algorithm eulerTour(T, p):
  perform the action for visiting node p on the left
  if p is an internal node then
    recursively tour the left subtree of p by calling
eulerTour(T, p.left())
  perform the action for visiting node p from below
  if p is an internal node then
    recursively tour the right subtree of p by calling
eulerTour(T, p.right())
  perform the action for visiting node p on the right
```

Binary Search Tree

- A binary search tree is a proper binary tree such that
  - each internal node p stores an element $x(p)$
  - for each internal node p, the elements stored in the left subtree are less than or equal to $x(p)$
  - for each internal node p, the elements stored in the right subtree are greater than or equal to $x(p)$
  - the external nodes do not store any element

- An inorder traversal of internal nodes visit the elements in non-decending order