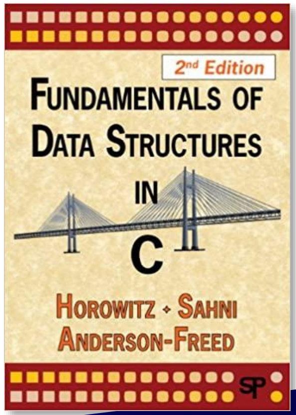
Data Structure

Tree

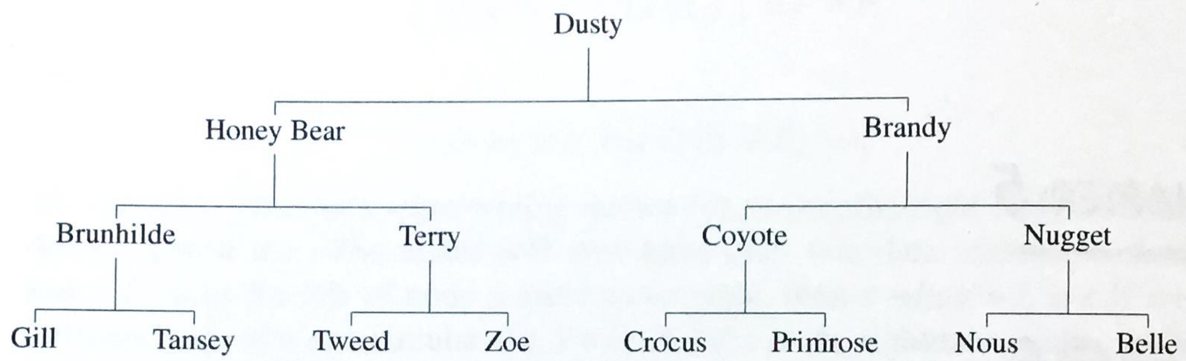
Shin Hong

May 26, 2020

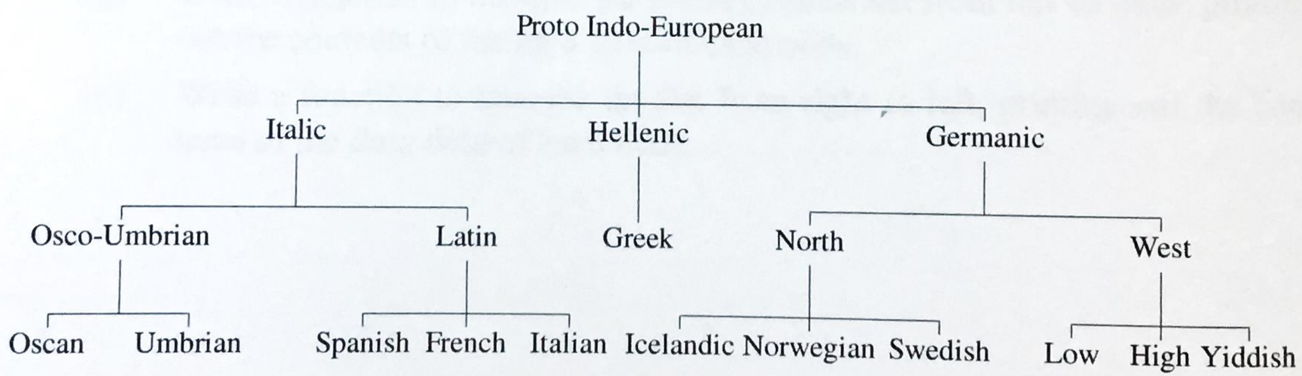
Ch. 5 Tree



Motivation



(a) Pedigree



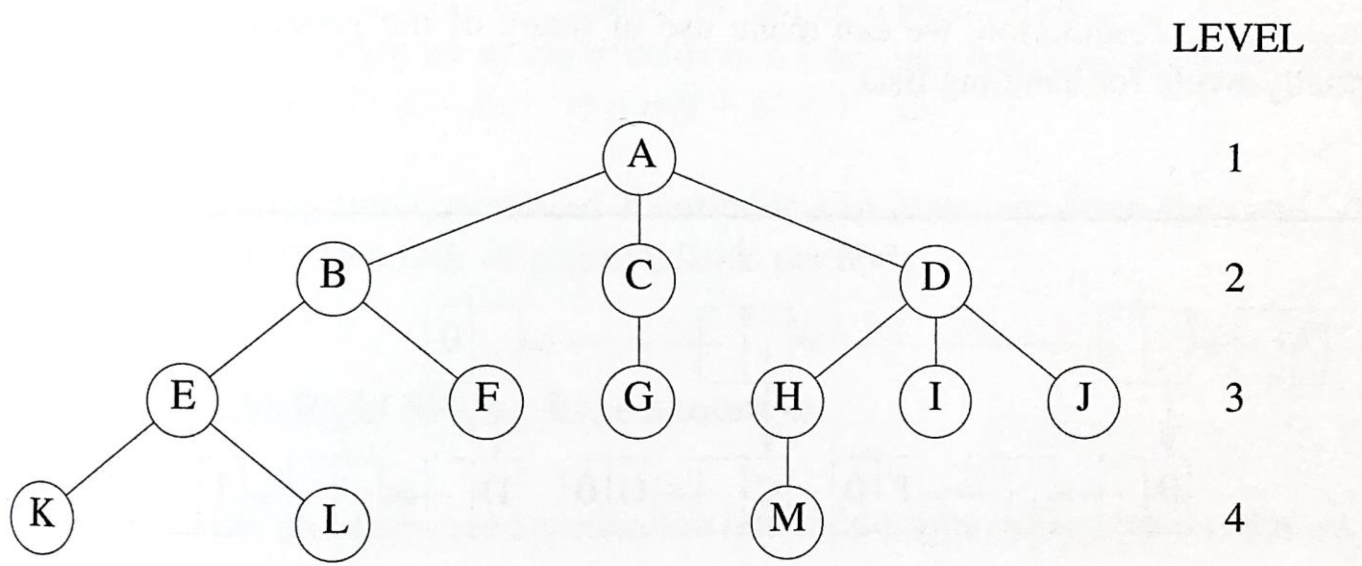
(b) Lineal

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 $T_1, T_2, \cdots, T_n$, where each of these sets is a tree (subtree)

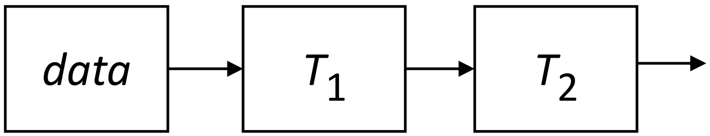
Terminologies

- Node: the iterm of information
- Branch: links between two nodes
- Degree of a node: the number of subtrees
  - Degree of a tree
- Leaf (terminal) node: node with degree zero
  - non-terminal nodes
- Children, Parent, Siblings, Ancestors
- Level of a node
- Height of a tree

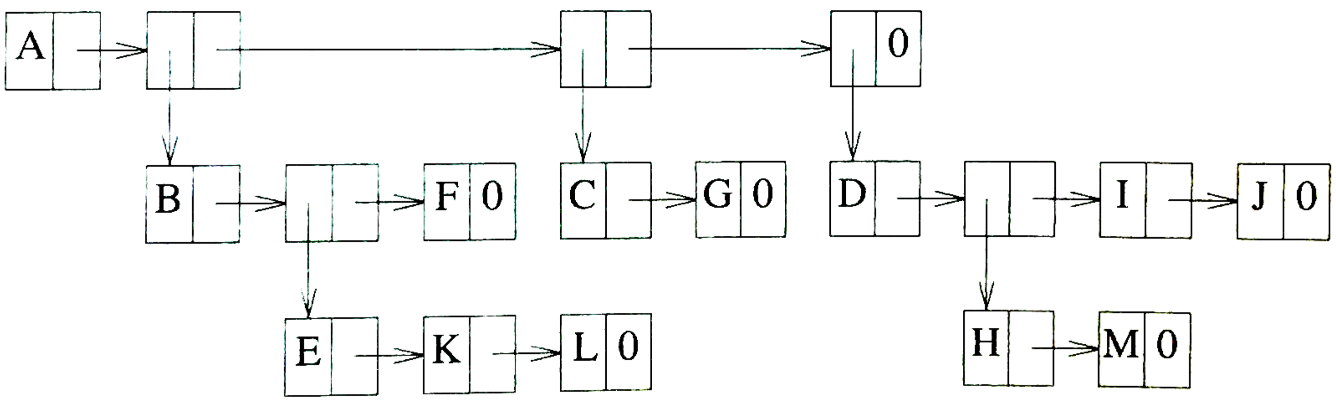


Tree Representation

- List representation
  - *Data, or (Data* $(T_1, T_2, \ldots, T_N)$)
  - E.g.,



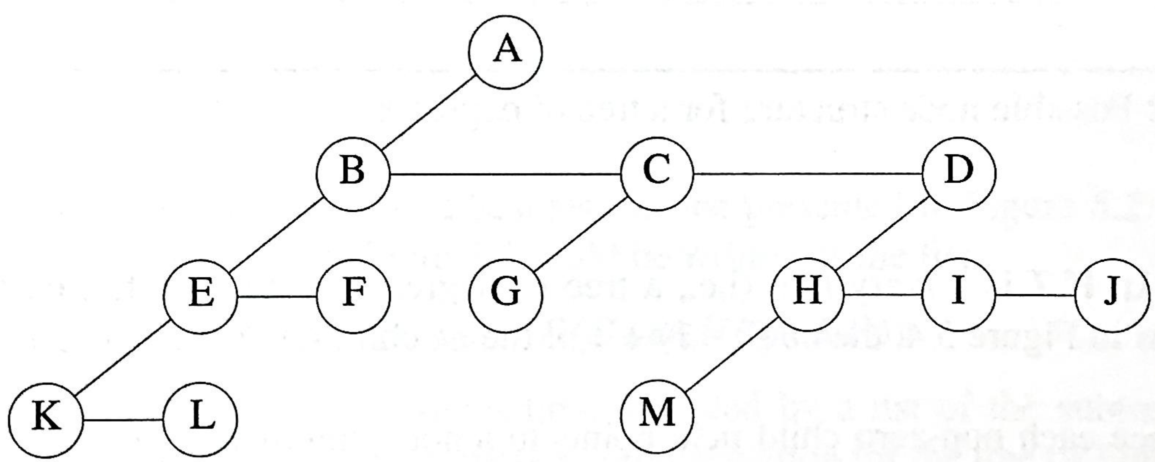
$(A\,(B\,(E\,(K,L),F),C\,(G),D\,(H\,(M),I,J)))$



Tree Representation

- Left child-right silbling representation

<table>
<tr><td colspan="2">data</td></tr>
<tr><td>left child</td><td>right child</td></tr>
</table>



Binary Tree

- A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees
  - A binary tree is a tree with a degree 2
  - each node may have a left child and a right child

- The definition of binary tree differs from the standard notion of a tree
  - no tree with zero node, but there's an empty binary tree
  - no ordering in children in a tree, but a binary has

Abstract Data Type *BinTree*

- Objects: a finite set of nodes consisting of left BinTree and right BinTree, or empty

- Functions
  - is_empty(bt)
  - make_bintree(left, right)
  - get_data(bt)
  - get_left_child(bt)
  - get_right_child(bt)

Properties of Binary Tree (1)

- The max number of nodes on level i of a binary tree is 2^{i-1} for $0 < i$
- The max number of nodes in a binary tree of depth k is 2^{k-1} for $0 < k$

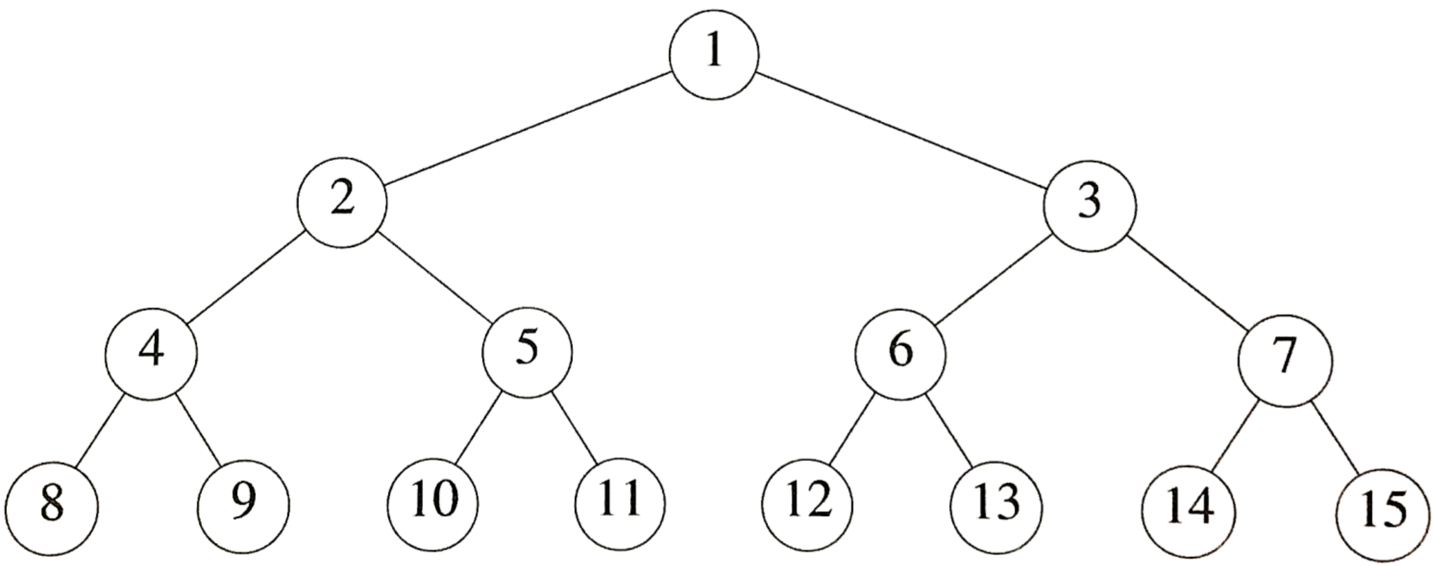
Properties of Binary Tree (2)

- For a non-empty binary tree, if b is the number of branches and n is the number of nodes, then $n = b + 1$

- For a non-empty binary tree, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2, then $n_0 = n_2 + 1$

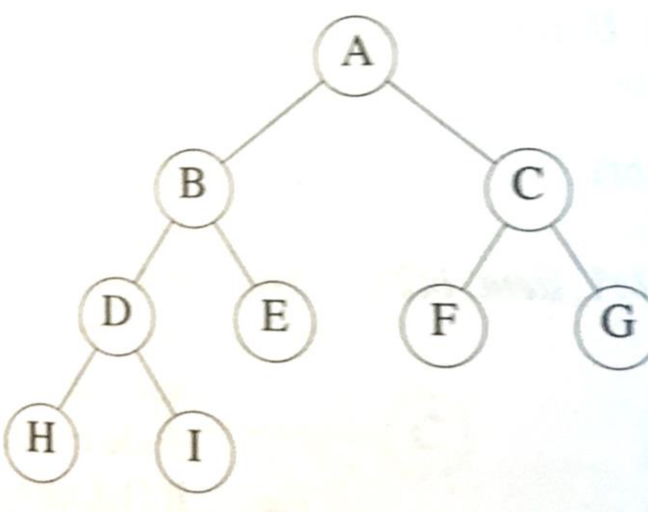
Terminologies (1/2)

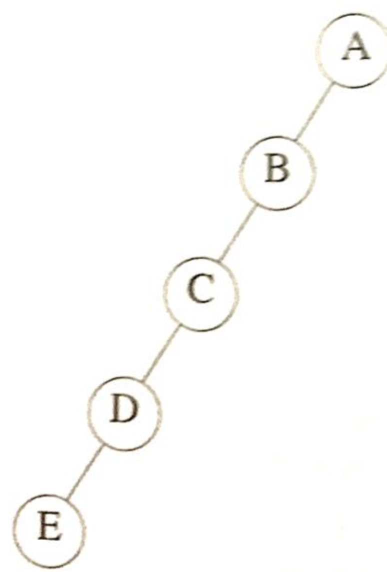
- A **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes



Terminologies (2/2)

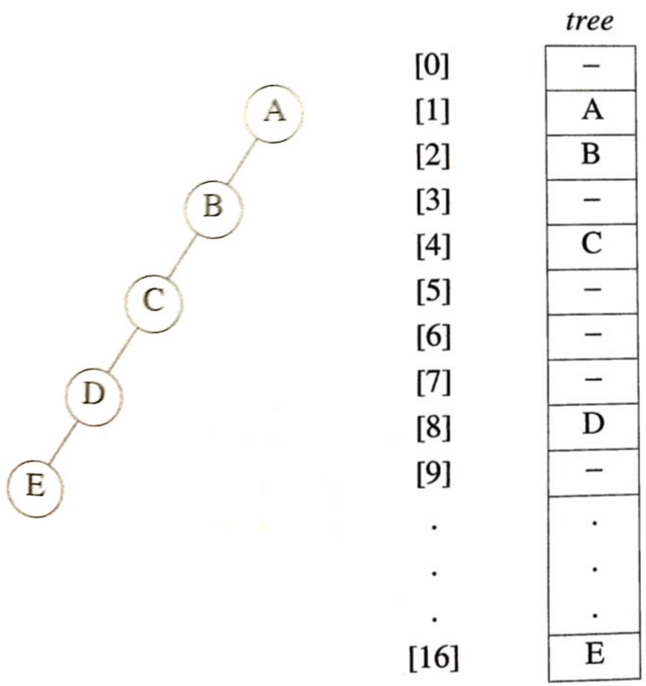
- A binary tree with n nodes and depth k is **complete** if and only if its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k
- The hiehgt of a complete binary tree with n nodes is $\lceil \log_2(n+1) \rceil$
- A tree is called skewed if nodes are skewed at left or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



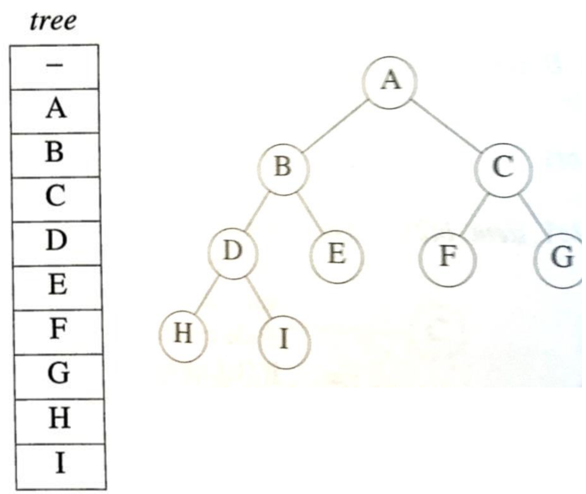


Representation: with array

- If a complete binary with n nodes is represented sequentially, then for any node with index i, the following properties hold
  1. parent(i) is at index floor $\lfloor i/2 \rfloor$ except the root node (i is 1)
  2. left_child(i) is at $2i$
  3. right_child(i) is at $2i+1$



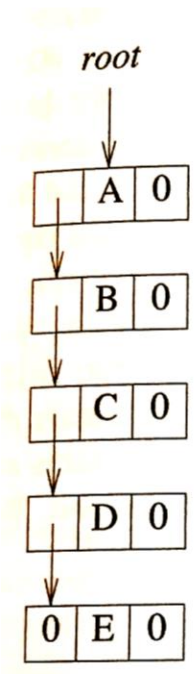
(a) Tree of Figure 5.10(a)

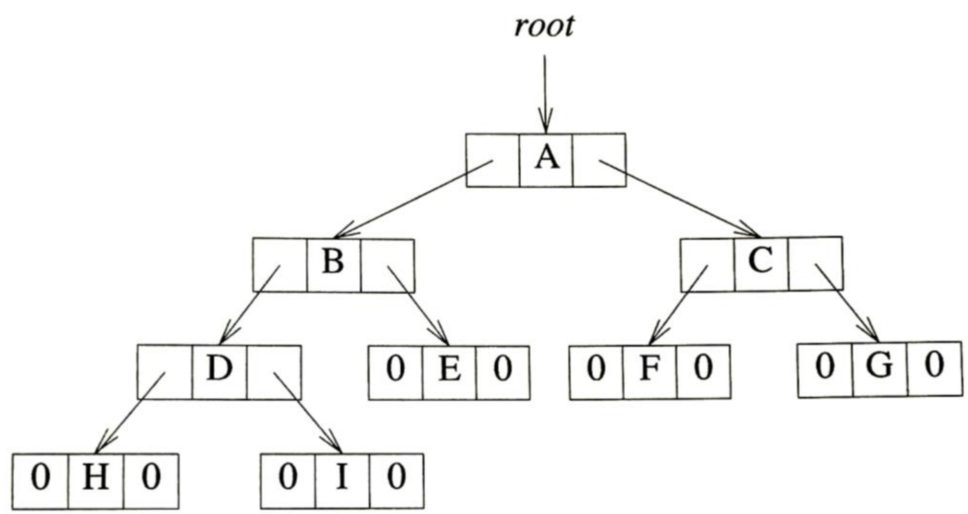


(b) Tree of Figure 5.10(b)

Representation: linked list

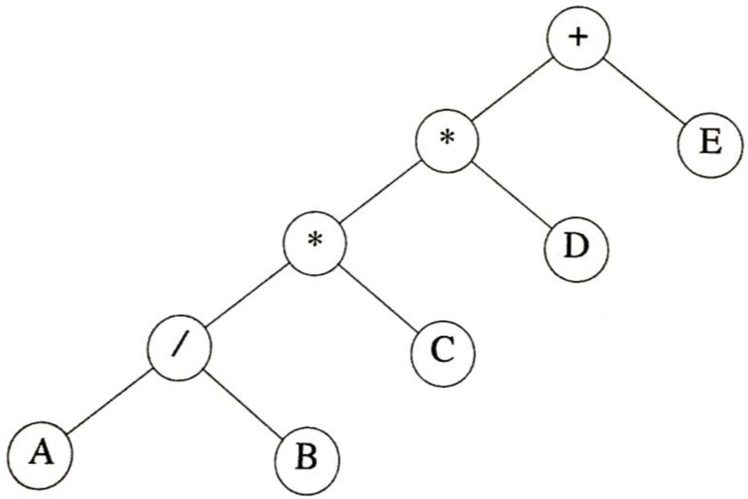
```
struct tree {
   int data ;
   struct tree * left ;
   struct tree * right ;
}
```





Binary Tree Traversal

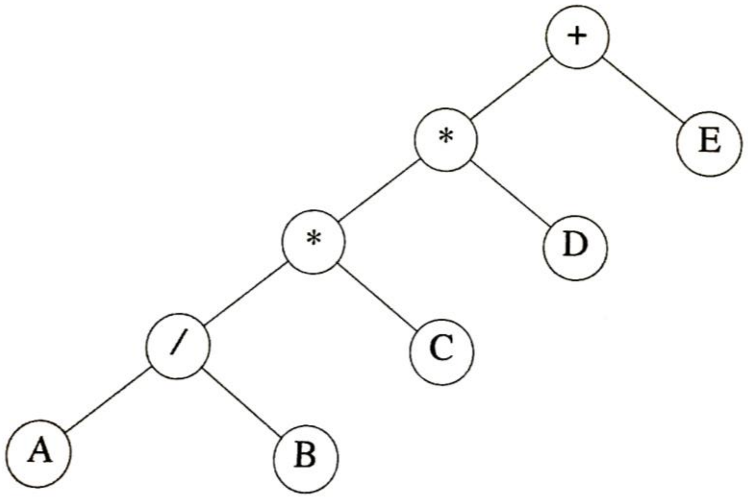
- A tree traversal is to visit each node in the tree exactly once and performs an operation at each visit of a node
- Traversal ordering: ordering of performing the operation
  - **Inorder traversal**: Left subtree → Visiting node → Right subtree
  - **Preorder traversal**: Visiting node → Left subtree → Right subtree
  - **Postorder traversal**: Left subtree → Right subtree → Visiting node



Binary Tree Traversal (Con'd)

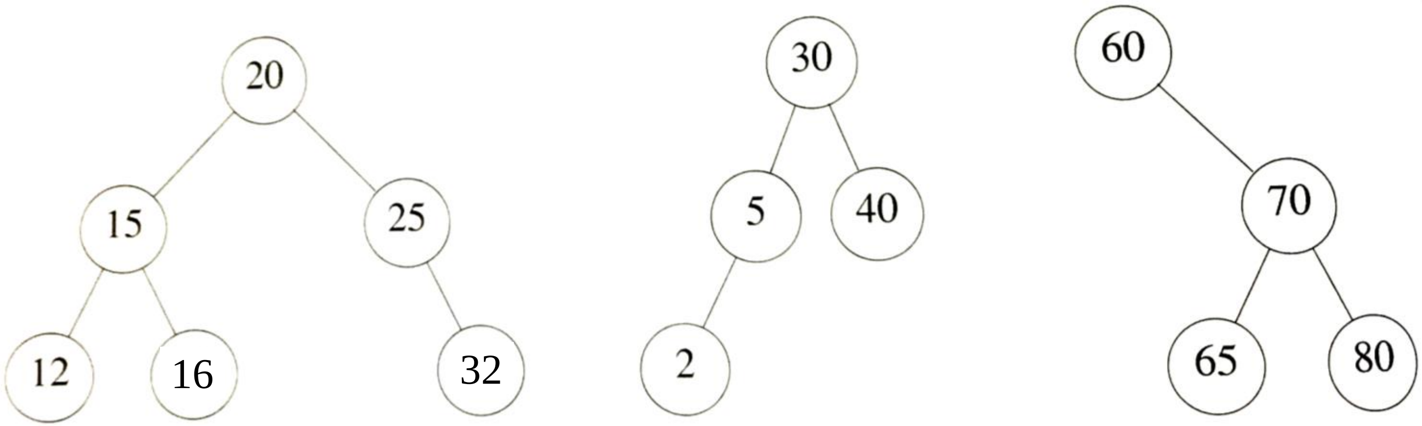
- Level-order traversal (breadth first search)
  - visit the nodes in their numbering order
  - a queue is needed for level-order traversal
  - algorithm

```
enqueue(root)
while queue is not empty do
    n = dequeue()
    visit(n)
    enqueue(left(n))
    enqueue(right(n))
done
```



Binary Search Tree

- A binary search tree is a binary tree with the following properties
  - (1) each node has a unique key
  - (2) keys in the left subtree are smaller than the key in the root
  - (3) keys in the right subtree are greater than the key in the root
- A binary search tree can be used for constructing a dictionary as a collection of key-value pairs
- Examples



Binary Search Tree - Operations

- search(T, K)
- insert(T, K, V)
- delete(T, K)

  locate node X whose key is K

  if it is a leaf, delete X

  if it has a single child, replace X with its child

  if it has two children:

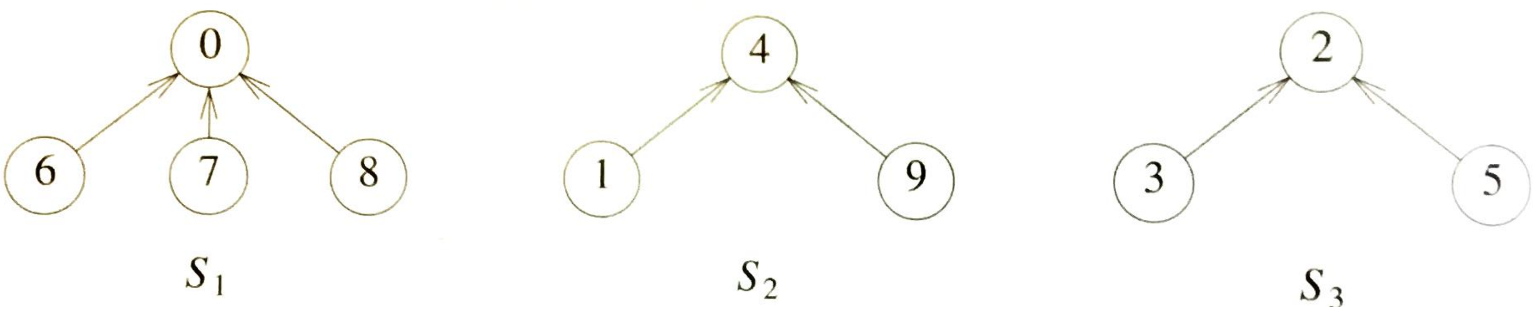
    find node Y immediately next to X

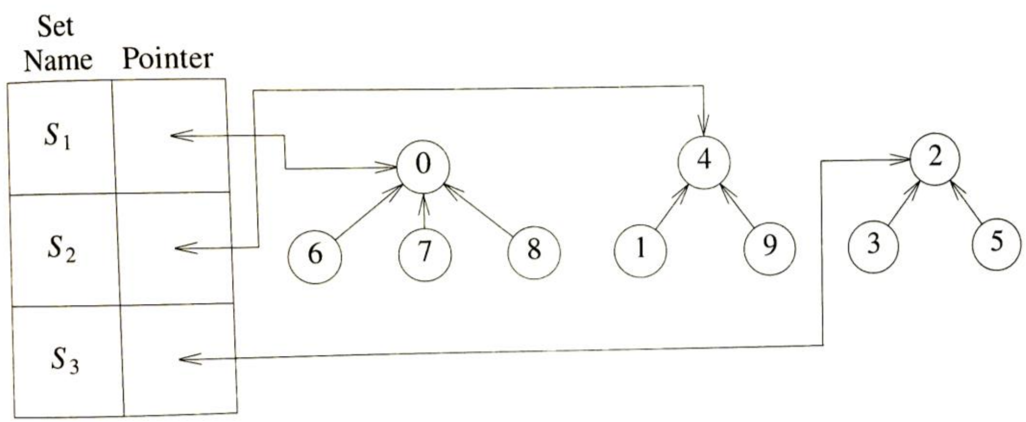
    replace the element of X with that of Y

    delete Y

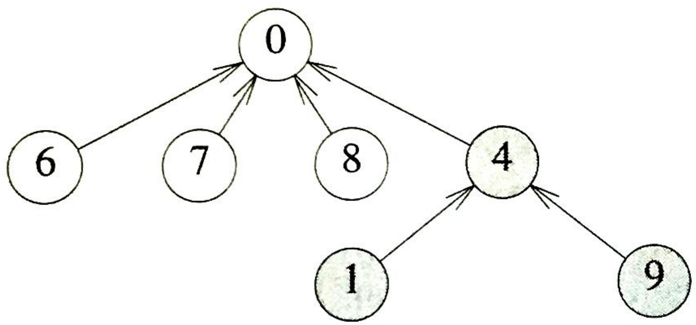
Representation of Disjoint Sets

Disjoint set union. If S_i and S_j are two disjoint sets, then their union $S_i \cup S_j$ = {all elements, x, such that x is in S_i or S_j}. Thus, $S_1 \cup S_2 = \{0, 6, 7, 8, 1, 4, 9\}$. Since we have assumed that all sets are disjoint, following the union of S_i and S_j we can assume that the sets S_i and S_j no longer exist independently. That is, we replace them by $S_i \cup S_j$.



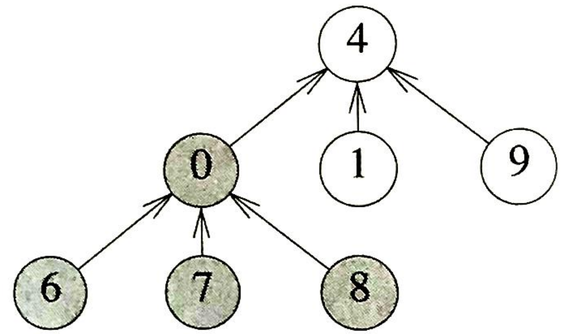


Union Operation



$S_1 \cup S_2$

or



$S_1 \cup S_2$

Find Operation

i	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
parent	−1	4	−1	2	−1	2	0	0	0	4