# Data Structures and Algorithms Design (DSECLZG519)

BITS Pilani
Hyderabad Campus

**Febin.A.Vahab**
**Asst.Professor(Offcampus)**
**BITS Pilani,Bangalore**

# CONTACT SESSION 4 -PLAN

innovate achieve lead

| Contact Sessions(#) | List of Topic Title | Text/Ref Book/external resource |
|---|---|---|
| 4 | Trees: Terms and Definition, Tree ADT, Applications Binary Trees : Terms and Definition, Properties, Properties, Representations (Vector Based and Linked), Binary Tree traversal (In Order, Pre Order, Post Order), Applications | T1:2.3 |

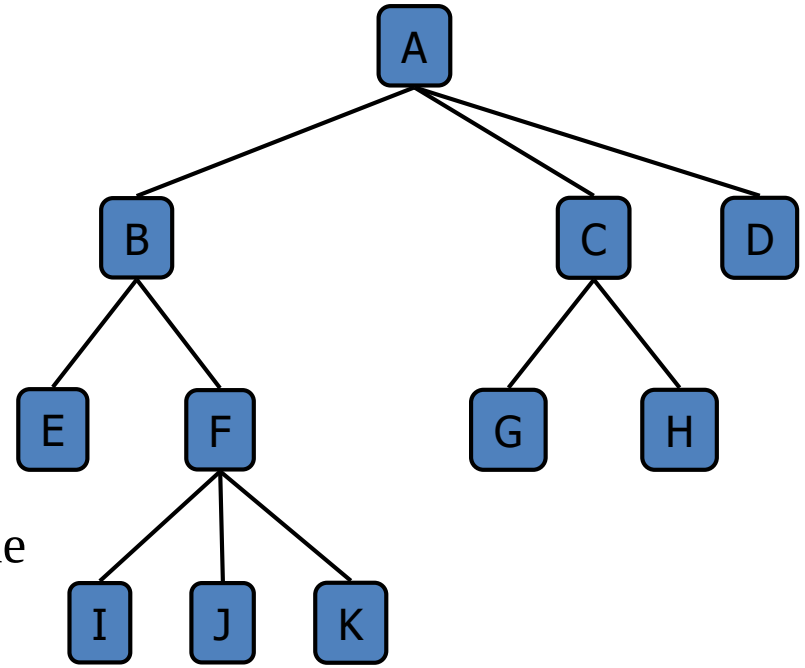
# TREES



- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- **Applications:**
  - Organization charts
  - File systems
  - Compiler Design/Text processing (syntax analysis & to display the structure of a sentence in a language)
  - Searching Algorithms
  - Evaluating a mathematical expression.

# TREE Terminologies

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



# TREE ADT-Depth and Height



- Let v be a node of a tree T.
- The depth of v is the number of ancestors of v, excluding v itself
- 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 T. isRoot(v) then
return 0
else
return 1 + depth (T, T. parent(v))
```

- The running time of algorithm depth ( T, v) is $O( 1 + d_v)$, where $d_v$ denotes the depth of the node v in the tree T.
- In the worst case, the depth algorithm runs in O (n) time, where n is the total number of nodes in the tree T

```
Algorithm depth(T, v):
if T. isRoot(v) then
return 0
else
return 1 + depth (T, T. parent(v)
```

# TREE ADT-Depth and Height

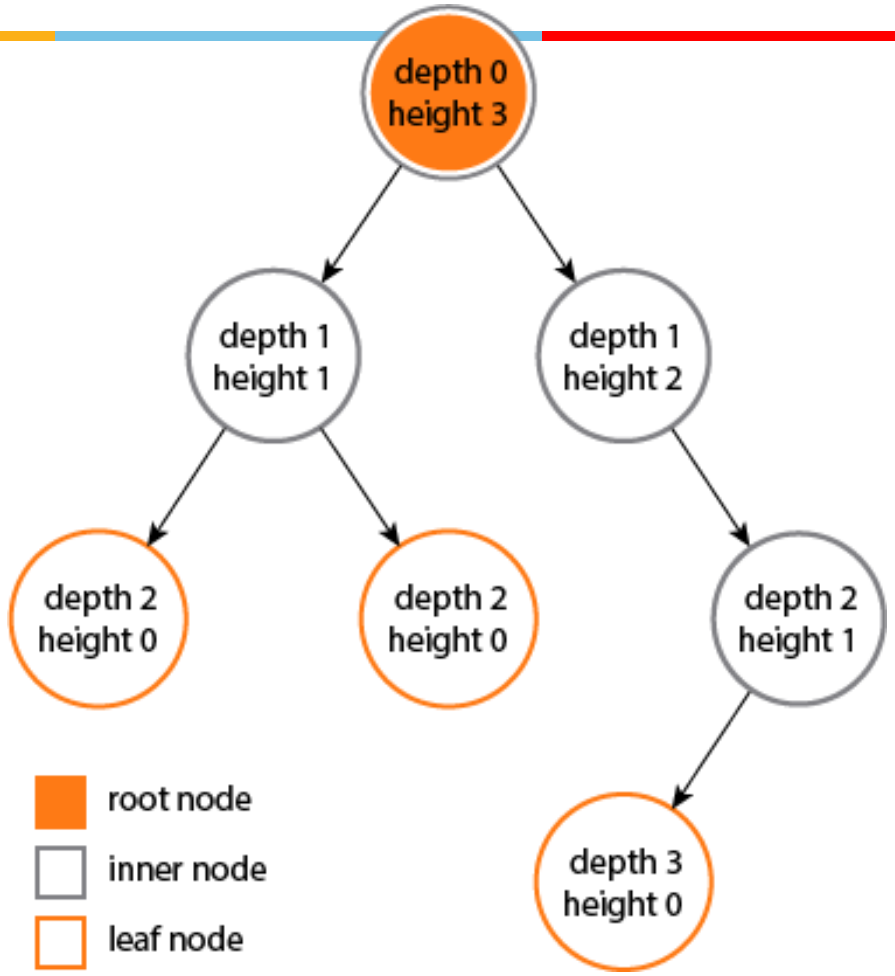


- The ***height of a tree*** T is equal to the maximum depth of an external node of T.
- If v is an external node, then the height o f v is 0.
- Otherwise, the height of v is one plus the maximum height of a child of v.

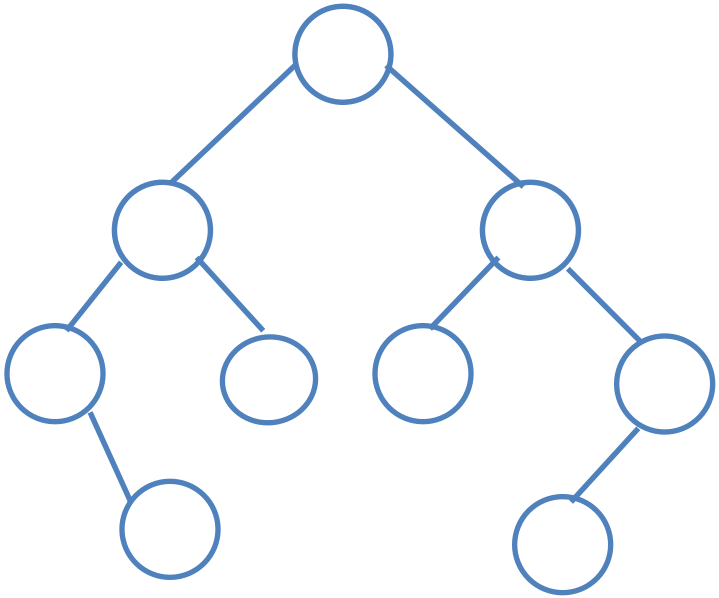
***Algorithm height(T, v) :***

```
if T. isExternal (v) then
return 0
else
h = 0
for each w ∈ T.children (v) do
h = max(h, height(T, w))
return 1 + h
```

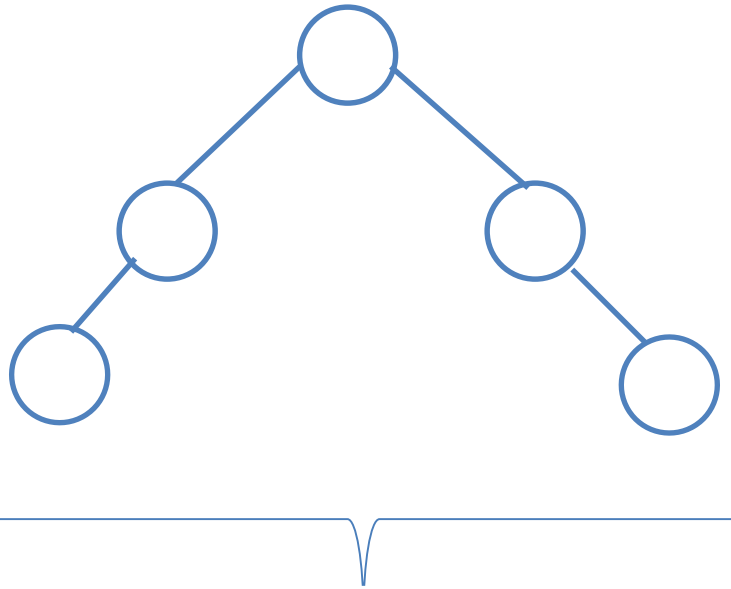
# TREE ADT-Depth and Height



# Binary Tree



Each node can have at most 2 – child

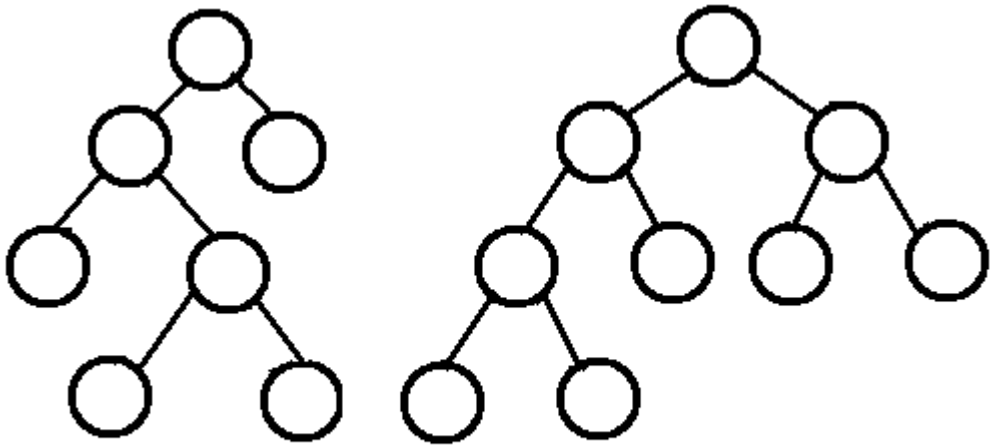


This is also binary tree

What about this ?

# Complete and Full(Proper) Binary tree



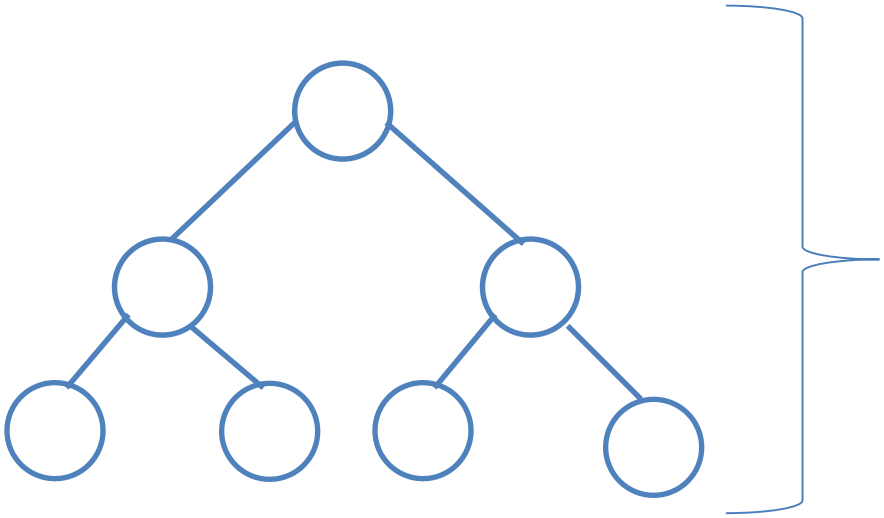


full tree complete tree

A **full binary tree** (sometimes proper **binary tree** or 2-**tree**) is a **tree** in which every node other than the leaves has two children

In Complete binary tree All levels except possibly the last are completely filled and all nodes are as left as possible.

# Perfect binary tree



In Strict binary tree all levels will be complete filled.

Maximum no of nodes in a binary tree with height h ,$\mathbf{n = 2^{h+1} - 1}$

Height of complete binary tree $\mathbf{h = \log_2 n}$

# TREE ADT



- We use **Positions** to abstract nodes
- Generic methods:

Back

  - ***integer size():*** Retum the number of nodes in the tree.
  - ***boolean isEmpty()*** :
  - ***ObjectIterator elements():*** Return an iterator of all the elements stored at nodes of the tree.
  - ***positionIterator positions():*** Return an iterator of all the nodes of the tree
- Accessor methods:
  - ***position root():*** Return the root of the tree
  - ***position parent(p):*** Return the parent of node v; error if v is root.
  - ***positionIterator children(p):*** Return an iterator of the children of node v.

# TREE ADT

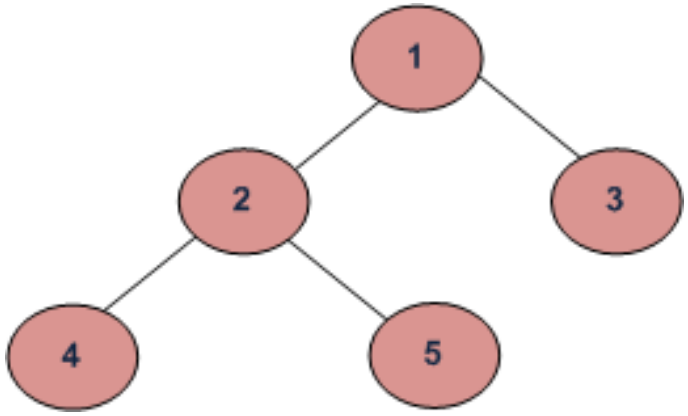


- Query methods:
  - ***boolean isInternal(p):*** Test whether node v is internal.
  - ***boolean isExternal(p):*** Test whether node v is external
  - ***boolean isRoot(p):*** Test whether node v is the root.
- Update methods:
  - ***swapElements(v, w):*** Swap the elements stored at the nodes v and w.
  - ***object replaceElement(v, e):*** Replace with e and return the element stored at node v

# Assumptions

- The accessor methods root() and parent($v$) take $O(1)$ time.
- The query methods isInternal($v$), isExternal($v$), and isRoot($v$) take $O(1)$ time, as well.
- The accessor method children($v$) takes $O(c_v)$ time, where $c_v$ is the number of children of $v$.
- The generic methods swapElements($v, w$) and replaceElement($v, e$) take $O(1)$ time.
- The generic methods elements() and positions(), which return iterators, take $O(n)$ time, where $n$ is the number of nodes in the tree.
- For the iterators returned by methods elements(), positions(), and children($v$), the methods hasNext(), nextObject() or nextPosition() take $O(1)$ time each.

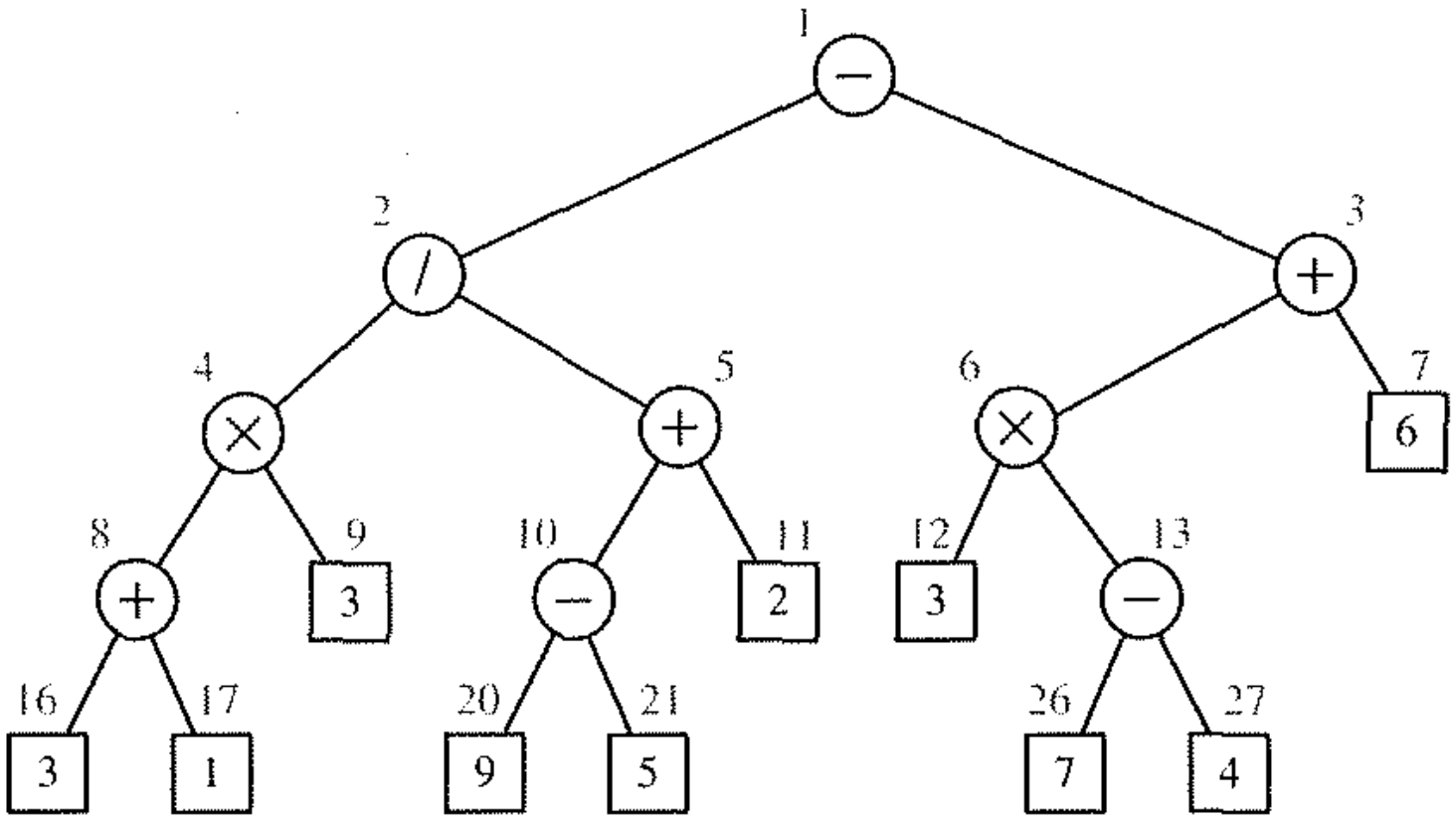
# TREE Traversals




(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

# TREE Traversals-Qn1-HW-Discuss in CANVAS





# QN-2-Class Discussion



- Find the preorder traversal for the binary tree using
- Inorder traversal : C D E N P  X Y
- Postorder traversal: D C E P Y X N
- Also construct the tree.

# QN-3-HW-Discuss in CANVAS



- Suppose that you are using a doubly linked list, maintaining a reference to the first and last node in the list, along with its size, to implement the operations below
- **addFirst(item):** prepend the item to the beginning of the list
- **get(i)** :return the item at position i in the list
- **set(i, item)** :replace position i in the list with the item
- **removeLast()** delete and return the item at the end of the list

- What is the worst-case running time of each of the operation above?

# QN-4-HW-Discuss in CANVAS

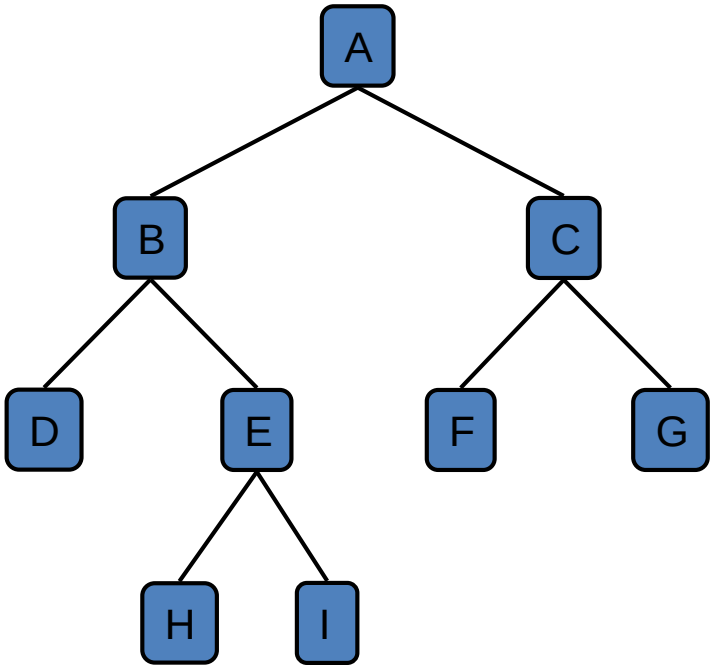
- We can construct a full binary tree from pre order and post order traversals. Discuss with an example.

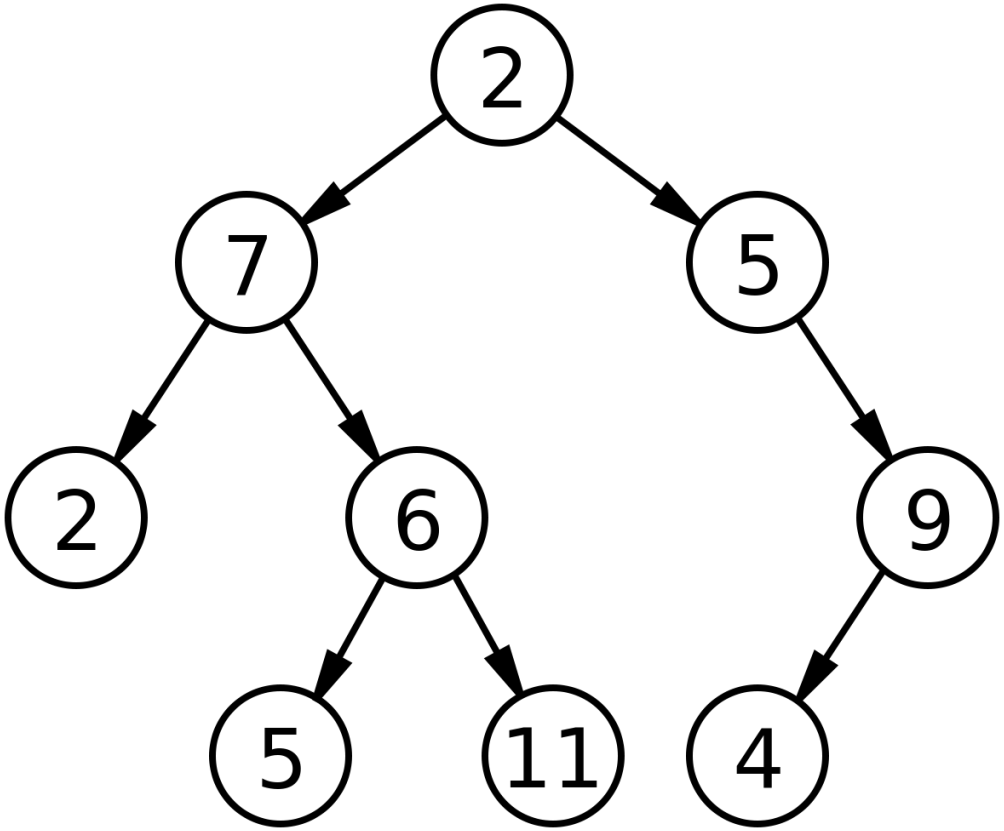
# Binary Trees



- A binary tree is a tree with the following properties:
  - Each internal node has two children
  - Each internal node has **ATMOST** two children
  - The children of a node are an ordered pair
- We call the children of an internal node **-left child and right child**
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

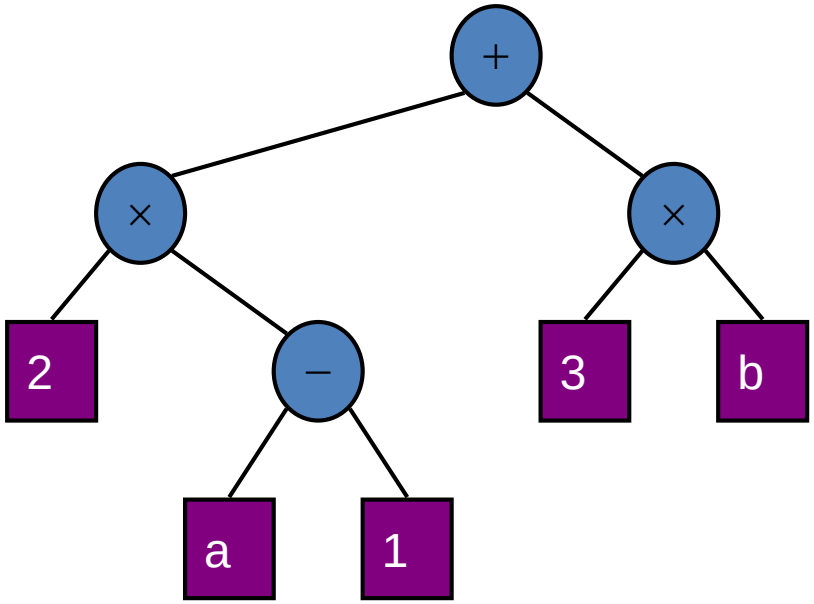
# Binary Trees



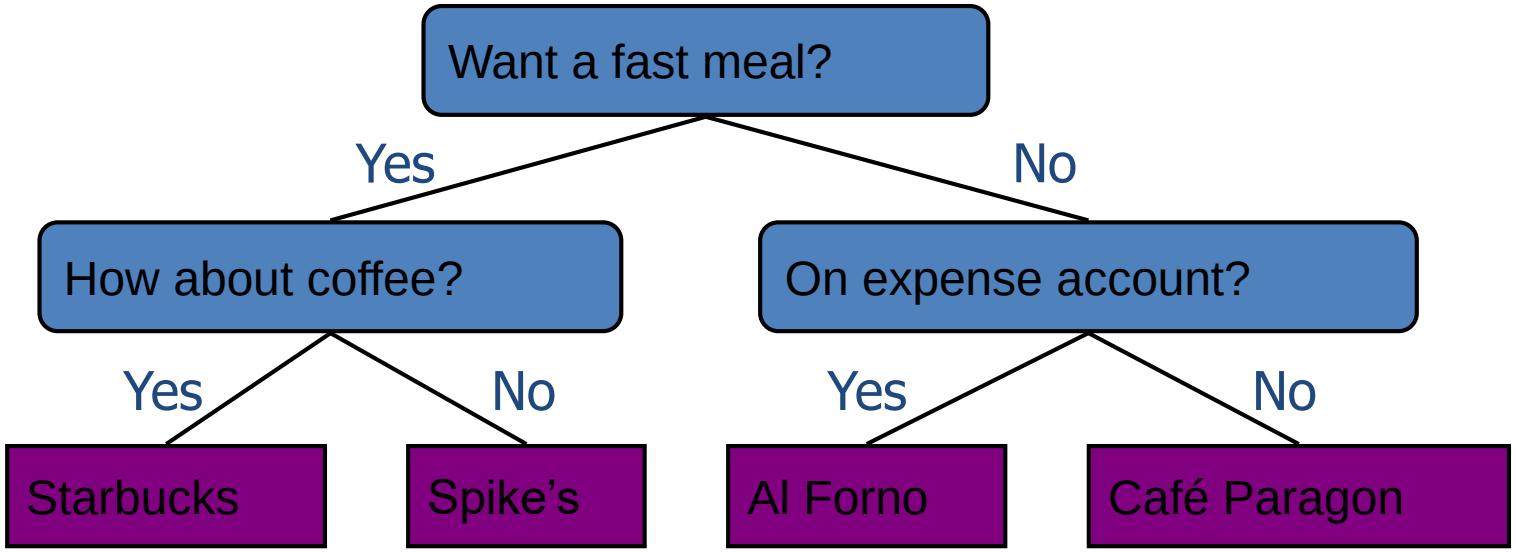
# Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression ((2 × (a – 1)) + (3 × b))



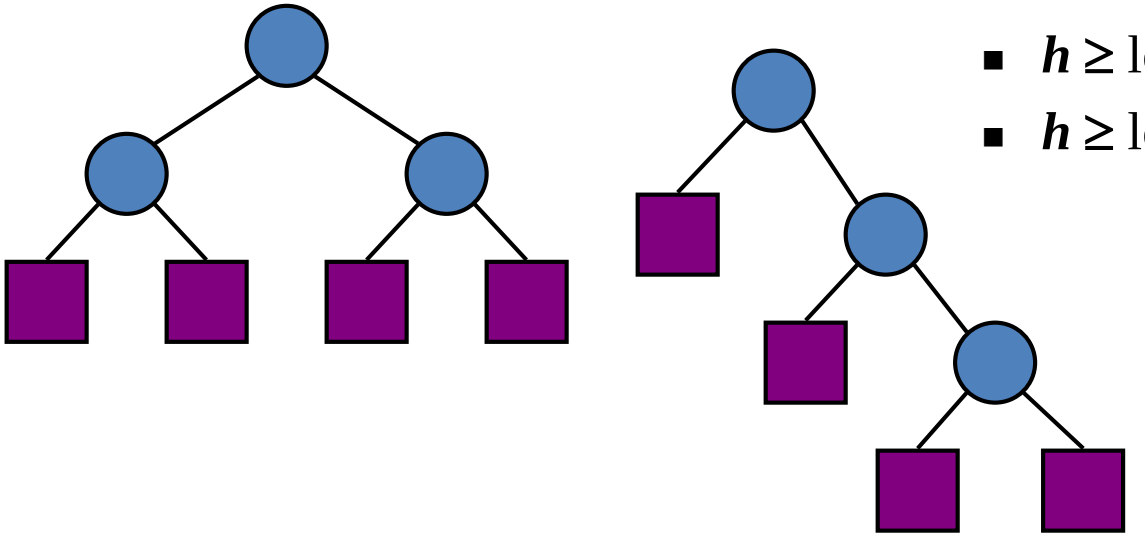
# Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



# Properties of (proper)Binary Trees

- Notation
  - ***n*** number of nodes
  - ***e*** number of external nodes
  - ***i*** number of internal nodes
  - ***h*** height



- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

# Properties of Binary Trees



Let T be a (proper) binary tree with n nodes, and let h denote the height of T. Then T has the following properties:

- The number of external nodes in T is at least h + 1 and at most $2^h$ .
- The number of internal nodes in T is at least h and at most $2^h - 1$ .
- The total number of nodes in T is at least 2h + 1 and at most $2^{h+1} - 1$ .
- The height of T is at least log(n + 1 ) -1 and at most (n - 1 ) /2, that is, log(n + 1 ) - 1 <=h <= (n - 1 )/2.
- The number of external nodes is 1 more than the number of internal nodes.

# BinaryTree ADT-Methods



- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT

**Additional methods:**

- ***position leftChild(v)*** Return the left child of v; an error condition occurs if v is an external node.
- ***position rightChild(v)***
- ***position sibling(p)***

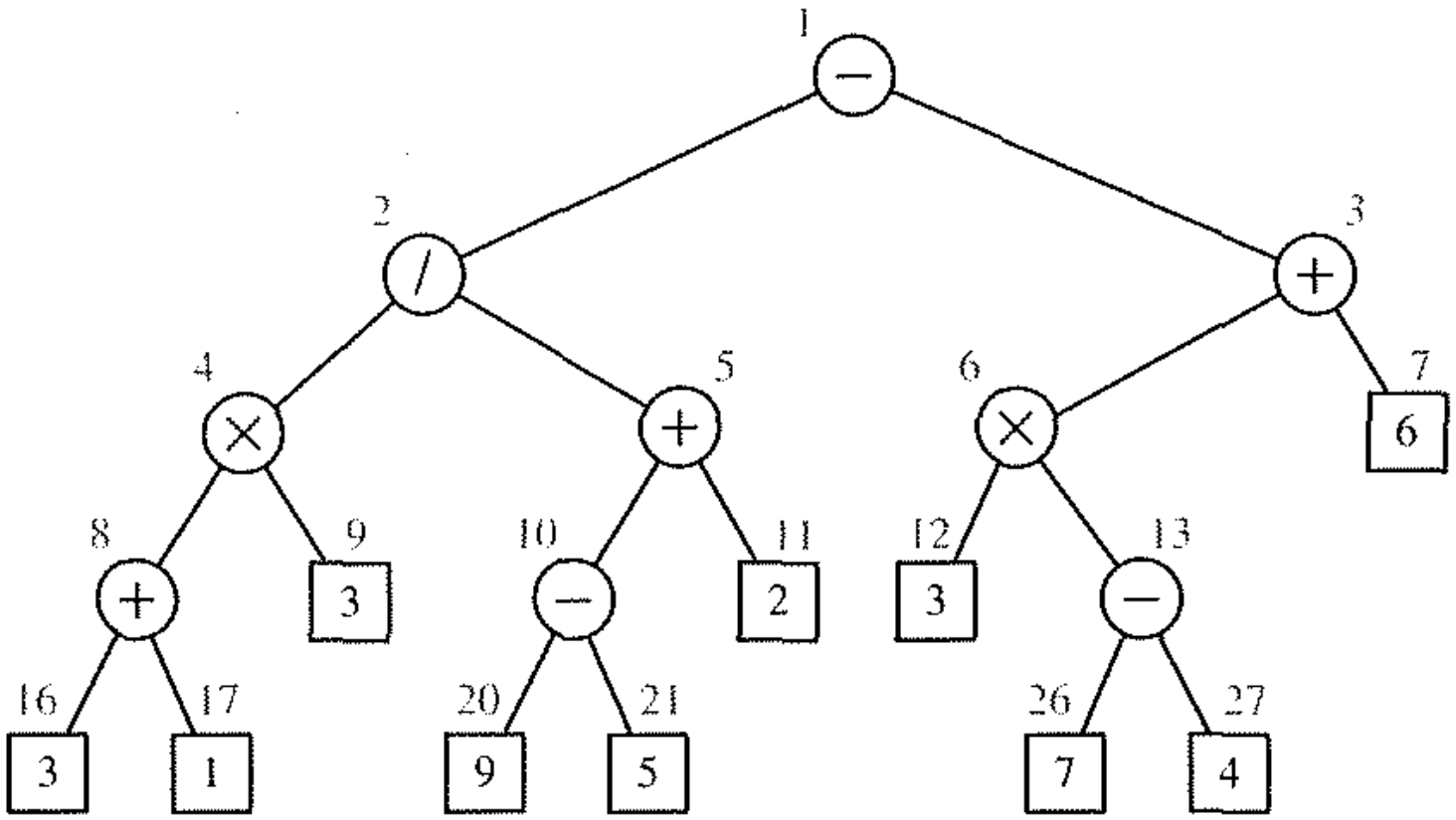
- Update methods may be defined by data structures implementing the BinaryTree ADT

# Binary Trees-Representations Vector Based



- For every node v of T , let p( v) be the integer defined as follows.
- If v is the root of T, then p(v) = 1 .
- If v is the left child of node u, then p( v) = 2p(u) .
- If v is the right child of node u, then p( v) = 2p (u) + 1 .
- ***p is known as a level numbering***

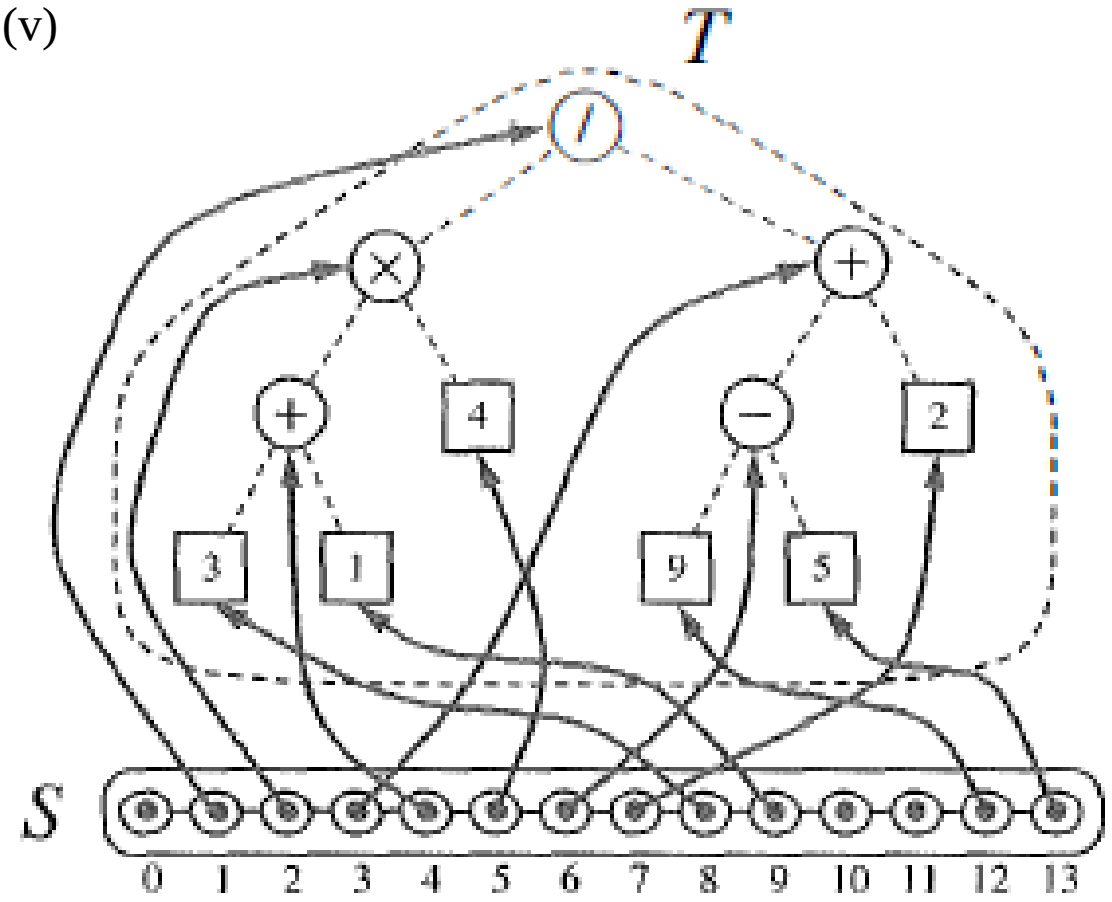
# Binary Trees-Representations Vector Based



# Binary Trees-Representations-Vector Based



Node v of T is associated with the element of S at rank p (v)



# Binary Trees-Representations Vector Based



- Let n be the number of nodes of T, and let $P_M$ be the maximum value of p (v) over all the nodes of T .
- Vector S has size $N = P_M + 1$ since the element of S at rank 0 is not associated with any node of T .
- Vector S will have, in general, a number of empty elements that do not refer to existing nodes of T.
- These empty slots could correspond to empty external nodes or even slots where descendants of such nodes would go
- In the worst case, $N = 2^{(n+1)/2}$

- //n is the number of nodes of T,N is the vector size

# Binary Trees-Representations Vector Based



- Operation Time
  - positions, elements O(n)
  - swap Elements, replaceElement O( 1 )
  - root, parent, children O( 1 )
  - leftChild, rightChild, sibling O ( 1 )
  - islnternal , isExternal, isRoot O( 1 )

Methods

  - Fast and Simple
  - Space inefficient if the height of the tree is large

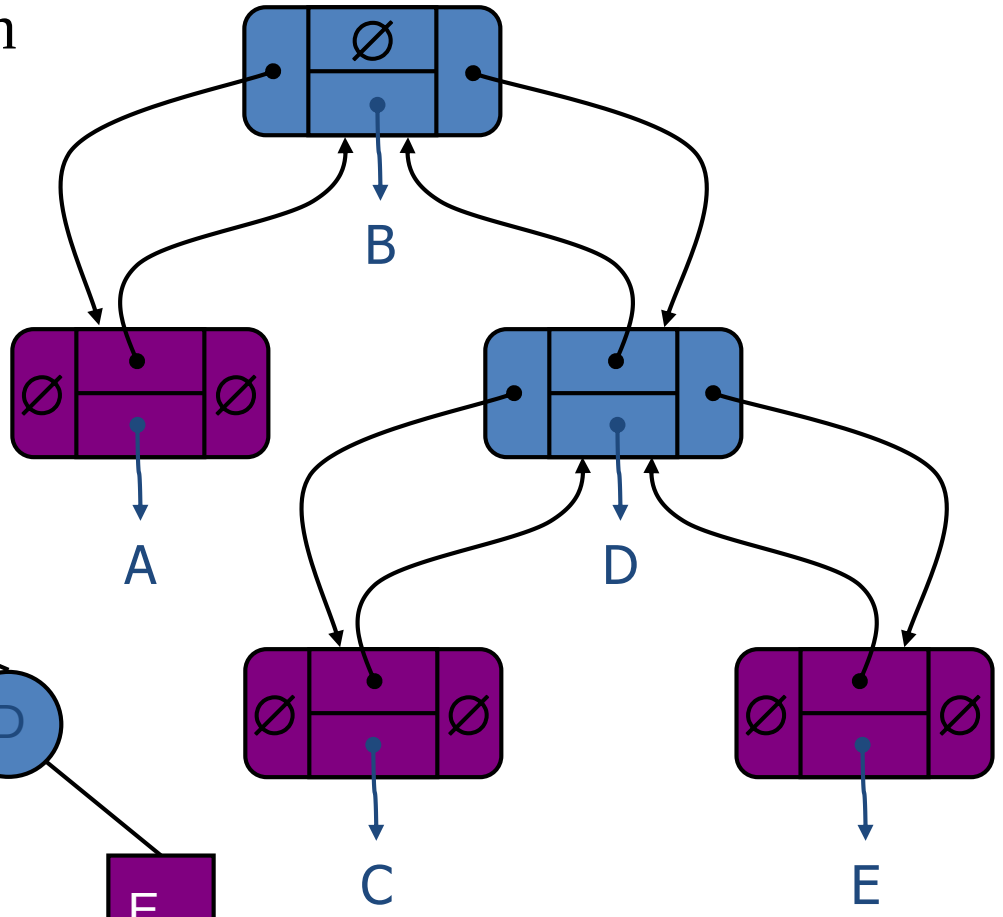
# Binary Trees-Representations Linked Structure

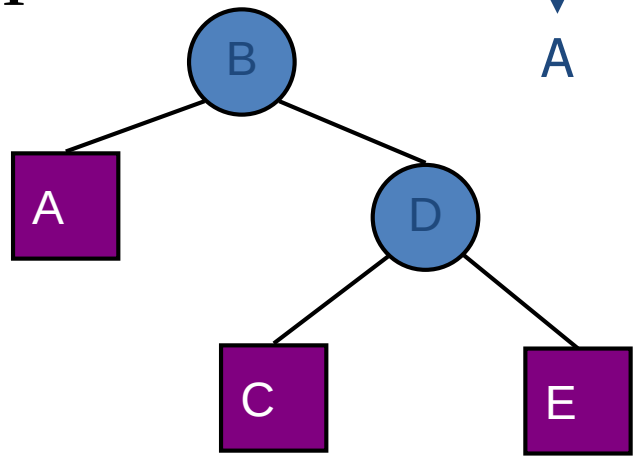


A node is represented by an object storing

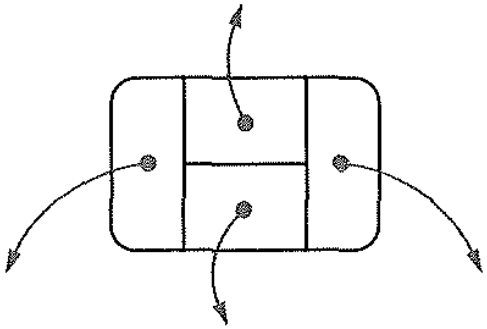
- Element
- Parent node
- Left child node
- Right child node

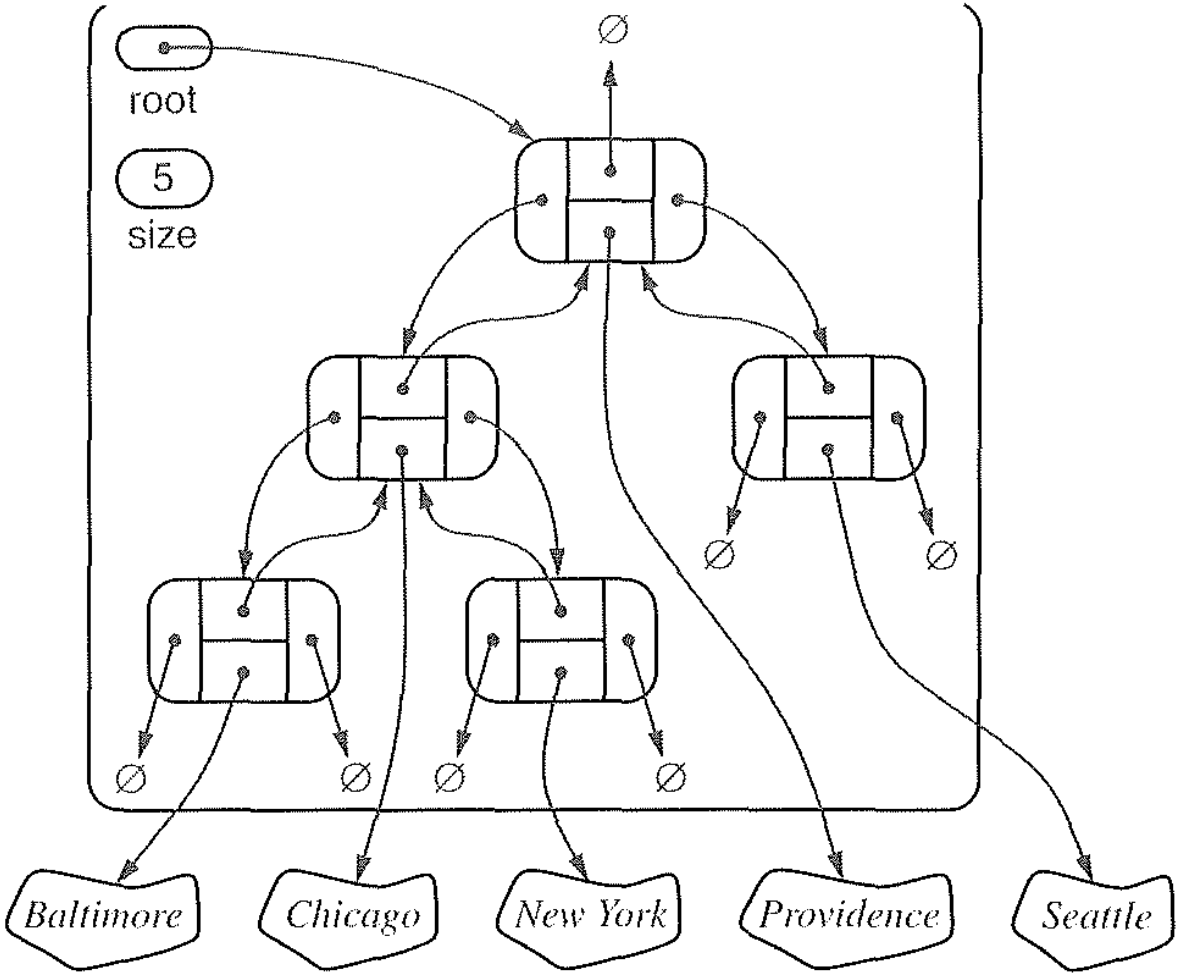
**Node objects implement the Position ADT**



# Binary Trees-Representations
# Linked Structure





# Binary Trees-Representations Linked Structure



- Operation Time
- Size()
- isEmpty()
- swapElements(v,w)
- replaceElement(v,e)

O(1)

- Positions()
- Elements()

O(n)

# Binary Trees-Applications



- Data Base indexing
- BSP in Video Games
- Path finding algorithms in AI applications
- Huffman coding
- Heaps
- SET AND MAP IN C++
- Syntax tree

# THANK YOU!

BITS Pilani
Hyderabad Campus