

15CSE201 : Data Structures and Algorithms

Lecture 9 : Non Linear Data Structures

Ritwik M

Based on the reference materials by Prof. Goodrich, OCW METU and Dr. Vidhya Balasubramanian

A Recap

- What we know so far:
 - Stacks
 - Queues – Linear, Circular, Deque
 - Vectors , Sequences and Iterators
 - Linked lists – SLL,DLL,CLL
- Where is it used?

Other Representations

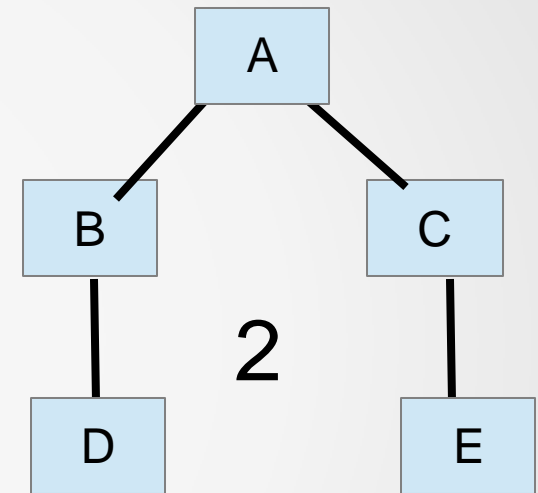
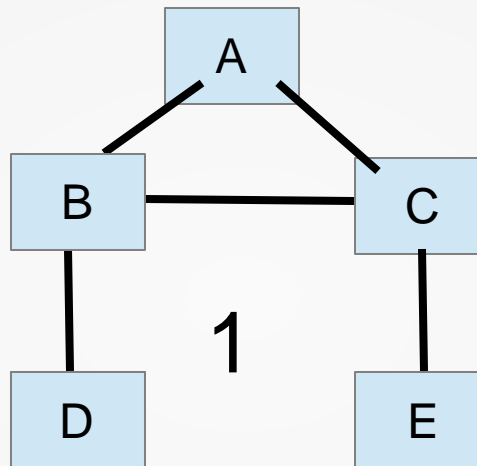
- Which is the optimal DS can we use for the following?
 - Job scheduling
 - Multiple print operations given to a single printer in a network
 - Undo and redo operations in a word processor
 - Buffering when watching videos on Youtube
 - Sorting a given set of data
 - Representing hierarchies in an organization
 - Displaying the map of a city

Why not Linear Data Structures?

- Difficult to represent relationships
- Have a fixed “order”
- How to represent unordered data?
- Solution: Non – Linear Data Structures
- Examples:
 - Trees
 - Graphs

Non Linear Structures

- The easiest way to represent hierarchies
- 2 Primary Types:
 - Graphs
 - Trees



- Which image above is a graph and which is a tree? Why?
- All trees are graphs, but all graphs need not be trees
- This session focuses more on trees.

Trees

- Concept:
 - Root : Primary category, The beginning...
 - Child :Sub category, The descendants...
- Why Trees:
 - they allow us to implement a host of algorithms much faster

Application of Trees

- Storing and manipulating information that naturally forms a hierarchy
- Used to represent decisions
- Can be used to represent paths and nodes of networks/ data
- Eg: the directory structure of your computer
- Any situation where branching happens

Trees – Definition

- An abstract data type that stores elements hierarchically
- Except the parent / root element, every other element in a tree must have a parent and zero or more child elements
- All trees must satisfy some special properties

Properties of trees

- Any tree **T** has the following properties:
 - T has special node **r** called the “**root**” which has no parent node
 - Each node **V** of T that is not r has a unique parent node u
 - If u is the parent node, v is the **child** of U
 - Two nodes that are children of the same parent are **siblings**
 - A node that has no children is the **leaf** node
 - AKA External Node
 - All **internal** nodes have at least one child

Properties contd...

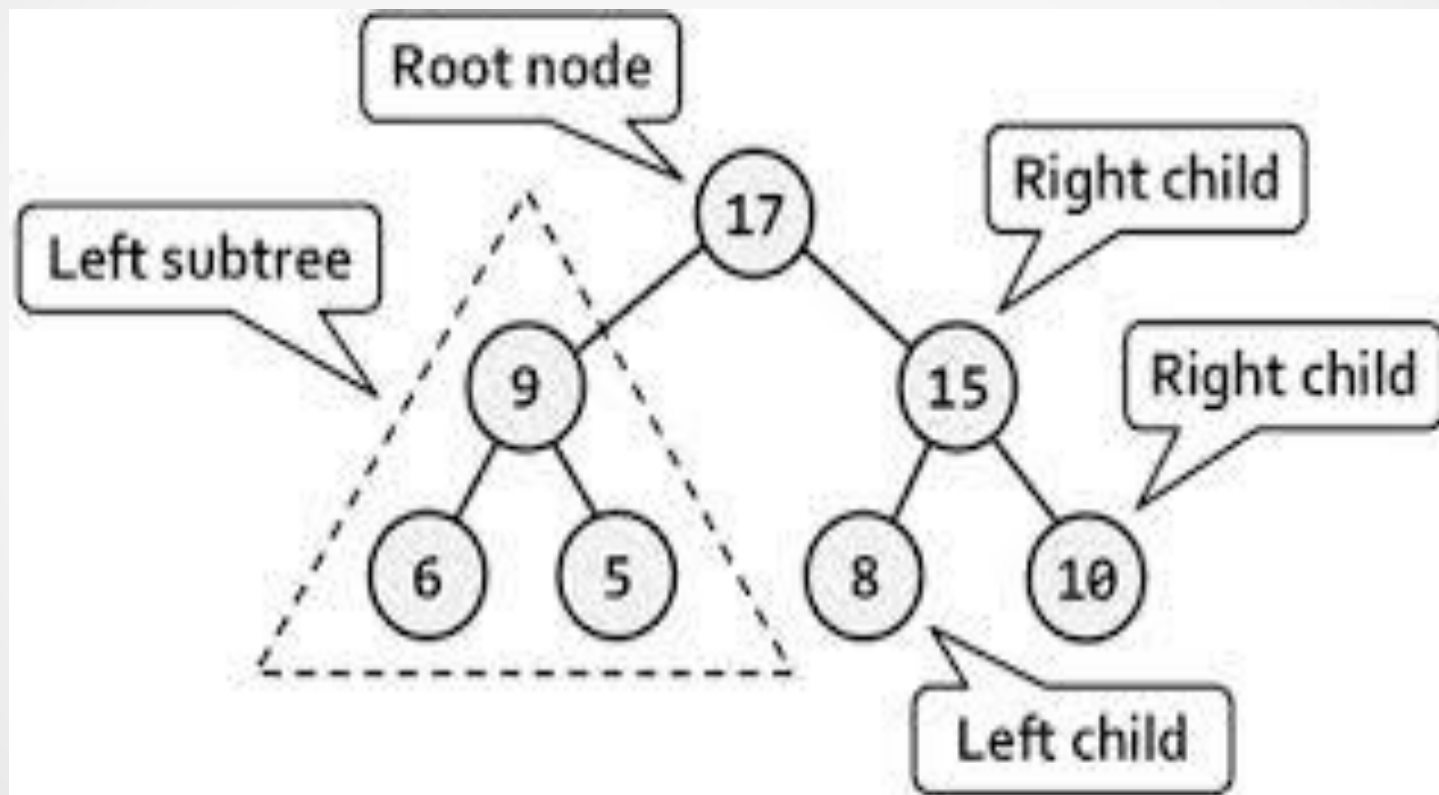
- **Ancestor** of a node is the node itself or all parents of the parent of the node
- A node U is the **descendant** of V if V is the ancestor of U
- A **subtree** S of T rooted at a node V is the tree consisting of V itself and all children and descendants of V in T

Properties Contd...

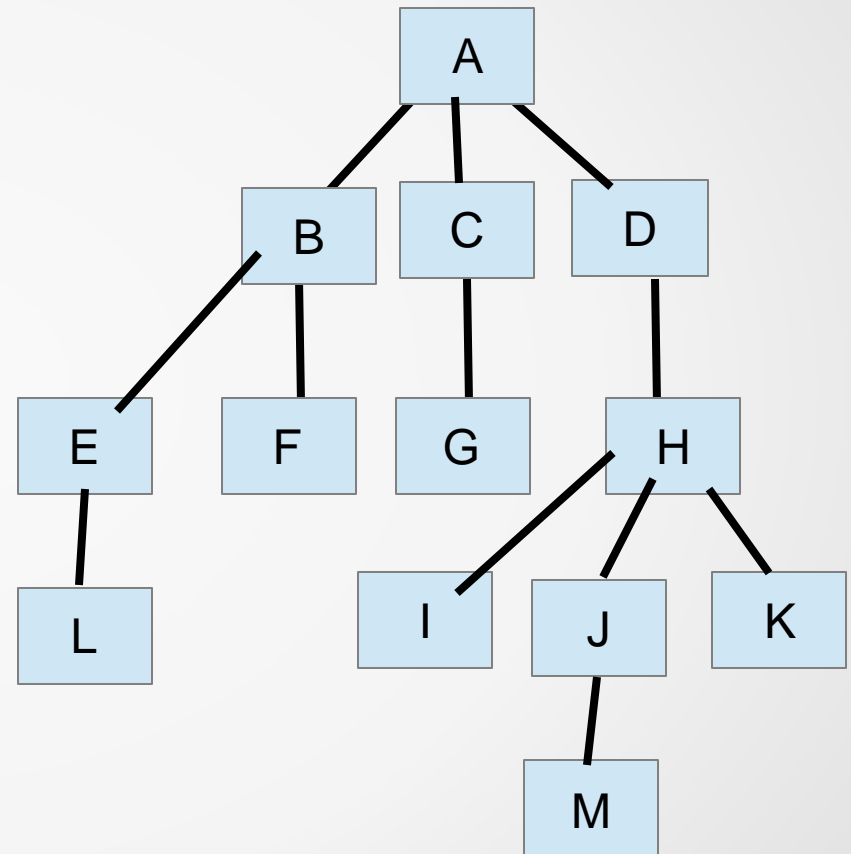
- Depth of a node is the distance from the current to the root node.
 - Also shows the level of the node
 - Depth of root node is 0
- Height of a node is the length of the longest downward path to a leaf from that node
 - Height of the tree is the height of the root
- Note: Height usually refers to the entire tree while depth refers to the distance of the node from the root

Example

- Which are the siblings here?



- Find Root Node
- List internal nodes
- Descendants of H
- Ancestors of L
- Identify nodes in subtree D
- What is the height of the tree
- What is the depth at node K



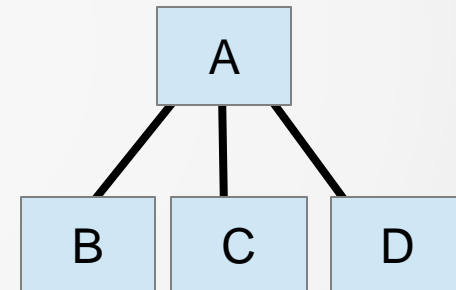
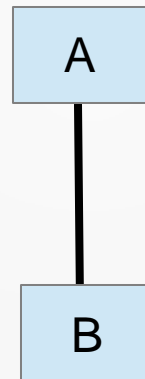
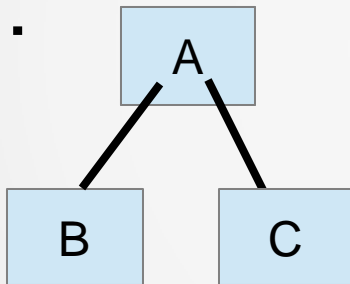
Ordered Trees

- A tree is ordered if there is a linear ordering defined for the children of each node
 - Each child of a node can be identified as the first, second, third etc
 - Usually done by arranging siblings left to right
- Ordered trees represent the linear order relationship between siblings
- Can you come up with any order in the previous example?

Degrees

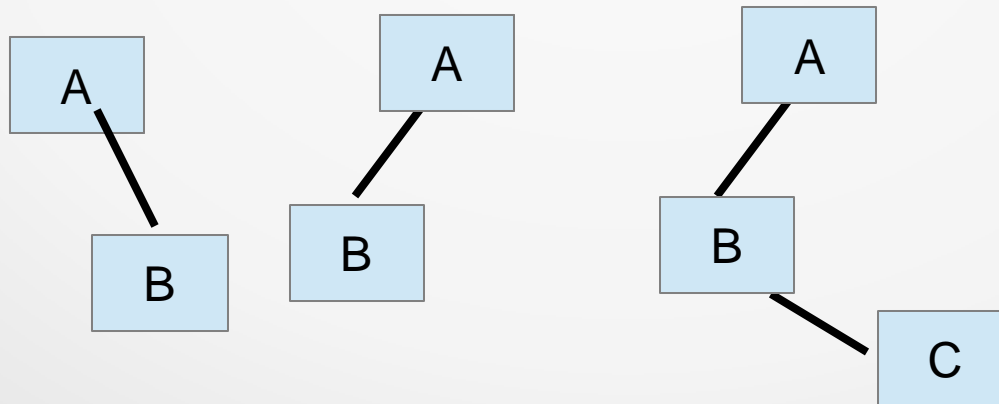
- The degree of a node in a tree is usually the number of child nodes attached to the current node

• Eg.:

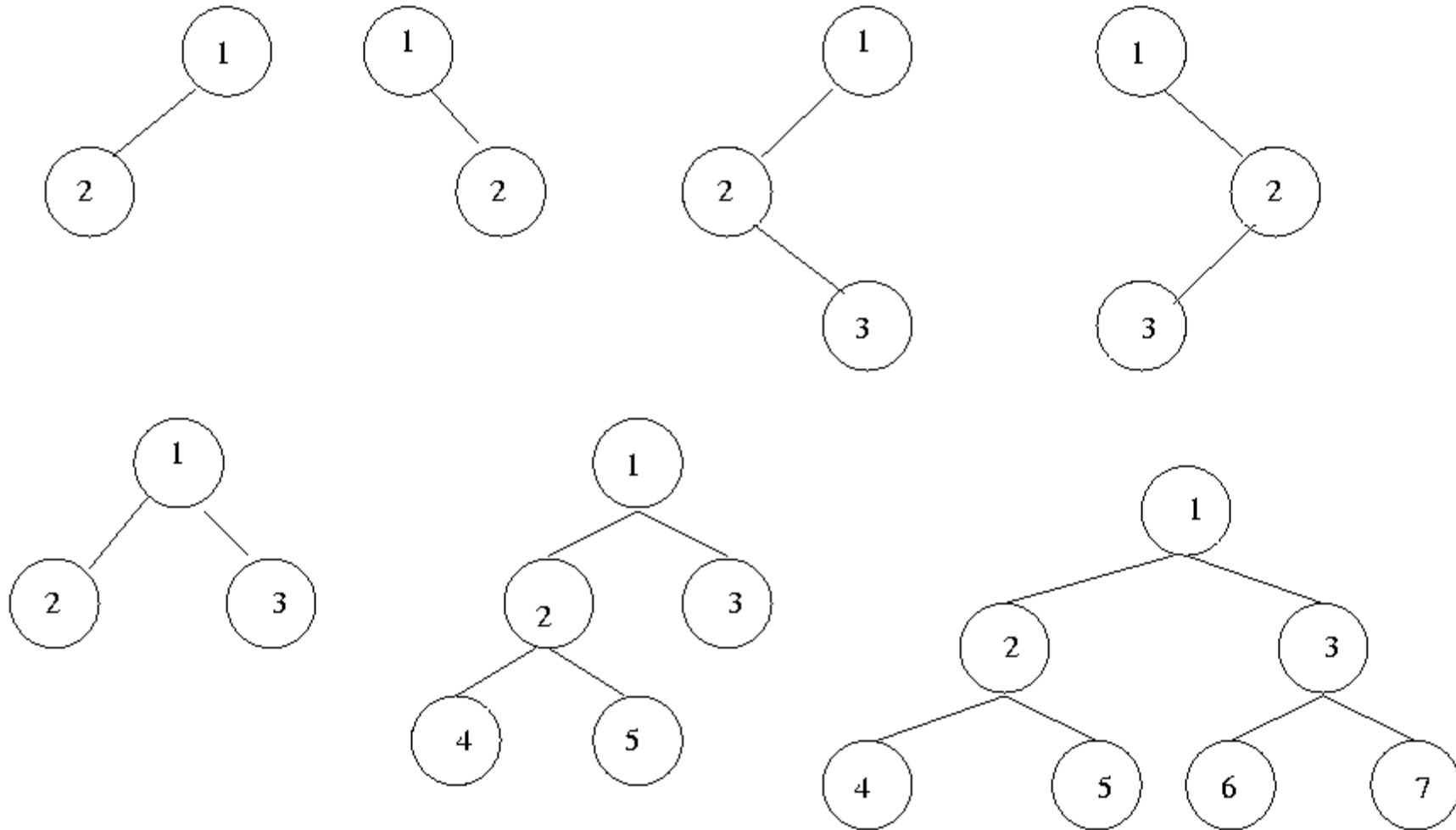


Binary Trees

- A binary tree is an ordered tree structure in which each node has at most 2 children.
- The tree is proper if every internal node has exactly 2 children
- eg.:



Examples



Source: <http://lcm.csa.iisc.ernet.in/dsa/node87.html#fig:bintree>

Full Binary Tree

Types of Binary Trees

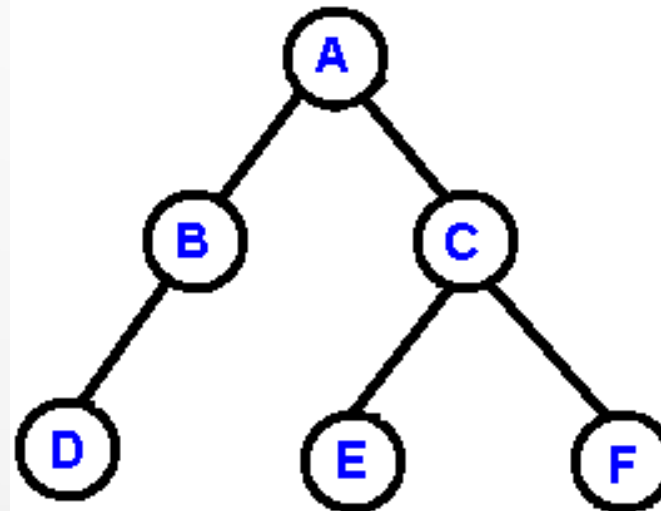
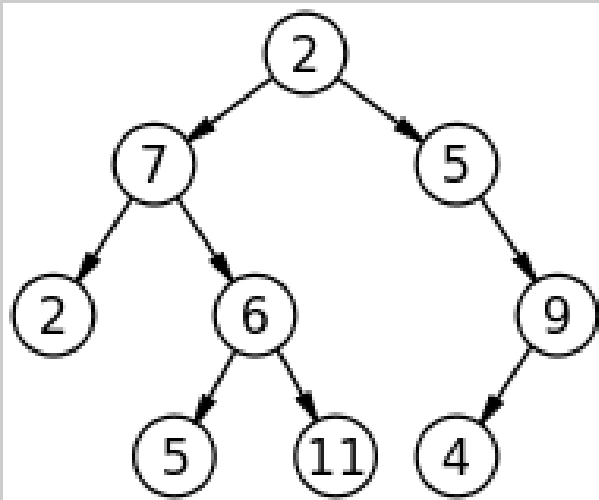
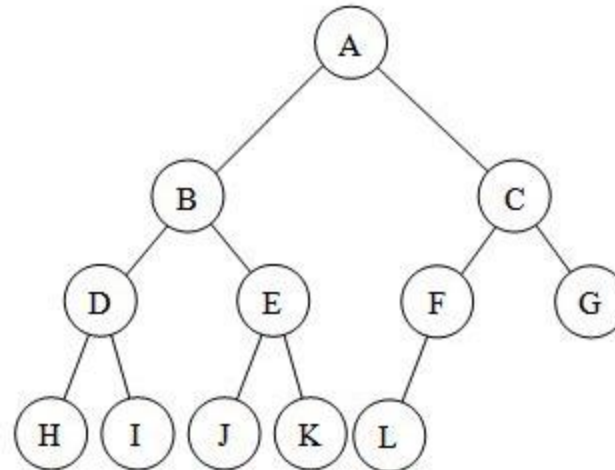
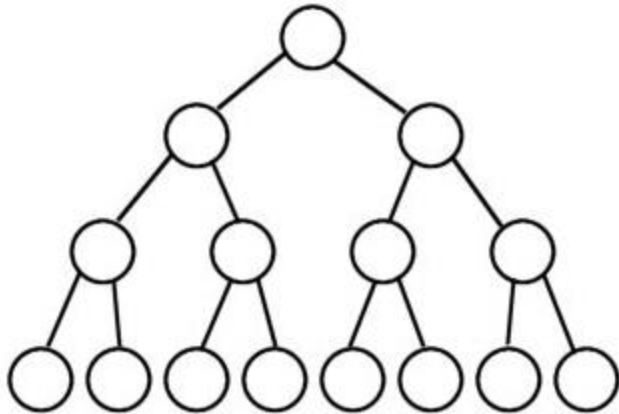
- A **rooted binary tree** is a tree with a root node in which every node has at most two children
- **Full Binary tree** also known as **proper binary tree** is an ordered tree in which each internal node has exactly two children.
- A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible
- A **balanced binary tree** is commonly defined as a binary tree in which the depth of the left and right subtrees of every node differ by 1 or less,

Types Contd...

A **perfect binary tree** is a full binary tree in which all leaves have the same depth or same level, and in which every parent has two children.

What is the difference between complete and perfect binary tree?

Identify the trees



Binary Tree ADT

- Stores values at nodes, and are defined relative to the neighboring node
 - Relationships satisfy the parent child relationship
- Accessor Functions
 - `root()`: returns the root of the tree
 - `parent(v)`: returns the parent of the node `v`; returns error if `v` is root.
 - `children(v)`: returns an iterator of the children of node `v`
 - children can be stored in an array or list
 - For instance left child is at index 0, right at index 1

Binary Tree ADT

- Verification Functions
 - isInternal(x) : Tests if the given node x is an internal node – returns Boolean
 - isExternal(x) : Tests if the given node x is an external node – returns Boolean
 - isRoot(x) : Tests if the given node x is a root node – returns Boolean
 - isEmpty()

Binary Tree ADT

- Update methods
 - `swapElements(u,v)`: swap elements stored at 'u' and 'v'
 - `replacementElement(v,e)` : replace the element at node v with element e
- Other Methods
 - `size()`:
 - `elements()`: returns an iterator of all elements stored at the nodes of the tree
 - `nodes()` : return an iterator of all nodes of the tree

Tree Traversal

A traversal of a tree T is a systematic way of visiting or accessing all the nodes of T

- Types of Traversals

- Preorder traversal

- Node is visited before its descendents

- Postorder traversal

- Node is visited after its descendents

- Inorder Traversal

- node is visited after its left subtree and before its right subtree

Preorder Traversal

First node accessed/visited is the root or parent

- Then nodes of the left subtree are visited (in preorder) before any node of the right subtree
- **Algorithm** preorder(T, v)
 visit(v)
 for each child w of v **do**
 preorder(T, w)

Postorder Traversal

- First node is visited after its descendants
- Used commonly compute space used by files in a directory and its subdirectories
- **Algorithm** $\text{postorder}(T, v)$
 for each child w of v **do**
 $\text{postorder}(T, w)$
 $\text{visit}(v)$

Inorder Traversal

- First visit the left child (including its entire subtree)
- Then visit the node, and finally visit the right child (including its entire subtree)
- **Algorithm** $\text{inorder}(T, v)$
 - if isInternal (v)
 - inOrder (leftChild (v))
 - visit(v)
 - if isInternal (v)
 - inOrder (rightChild (v))

Exercise

Draw a binary tree T such that

- Each internal node of T stores a single character
- A preorder traversal of T yields EXAMFUN
- An inorder traversal of T yields MAFXUEN

Properties of Binary Trees - I

- The number of nodes n in a perfect binary tree can be found using this formula: $n = 2^{h+1}-1$ where h is the depth of the tree.
- The number of nodes n in a binary tree of height h is at least $n = h + 1$ and at most $n = 2^{h+1}-1$ where h is the depth of the tree.
- The number of leaf nodes L in a perfect binary tree can be found using this formula: $L = 2^h$ where h is the depth of the tree.

Properties of Binary Trees - II

- The number of nodes n in a perfect binary tree can also be found using this formula: $n = 2L - 1$ where L is the number of leaf nodes in the tree.
- The number of null links (i.e., absent children of nodes) in a complete binary tree of n nodes is $(n + 1)$,
- The number of internal nodes (i.e., non-leaf nodes or $n - L$) in a complete binary tree of n nodes is $\lfloor n/2 \rfloor$.
- For any non-empty binary tree with n_0 leaf nodes and n_2 nodes of degree 2, $n_0 = n_2 + 1$.

Properties of Binary Trees - III

- Full Binary Tree Theorem
- Statement:
 - The number of leaves in a non-empty full binary tree is one more than the number of internal nodes
- Proof – Use Induction
 - Base Cases:
 - The non-empty tree with zero internal nodes has one leaf node.
 - A full binary tree with one internal node has two leaf nodes.
 - Thus, the theorem holds for the base cases i.e. for $n = 0$ and $n = 1$

Full Binary Tree Theorem Proof

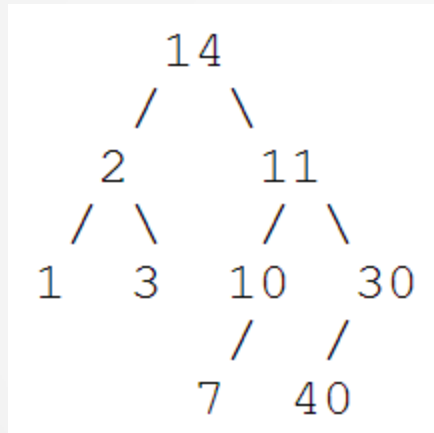
- Induction Hypothesis
 - Assume that, any full binary tree T containing $n - 1$ internal nodes has n leaves
- Induction Step
 - Given T with n internal nodes, select an internal node u whose children are both leaf nodes
 - Remove both children of u , making it a leaf node, and let the resulting tree be T'
 - T' has $n-1$ internal nodes, and by the theorem, n leaves
 - Restore the children of u , hence resulting in T with n internal nodes
 - Since T' has n leaves, adding 2 makes it $n+2$
 - However node u which was earlier counted as leaf, now is an internal node
 - Thus, tree T has $n + 1$ leaf nodes and n internal nodes. Hence proved

Exercise: Properties of Binary trees

- Draw a binary tree with height 9 and maximum number of external nodes
 - What is the minimum number of external nodes for a binary tree with height h . Justify
 - What is the maximum number of external nodes for a binary tree with height h . Justify
 - Let T be a binary tree with height h and n nodes. Show that
 - $\log(n+1)-1 \leq h \leq (n-1)/2$
 - For which values of n and h can the above lower and upper bounds on h be attained with equality

Quiz 2

1. Draw a full binary tree with at least 6 nodes.
2. Write all the nodes visited in all 3 traversals



3. What is the minimum number of nodes in a full binary tree with depth 3? Draw the tree

Quiz contd.

4. Suppose T is a binary tree with 14 nodes. What is the minimum possible depth of T ? Draw the tree
5. Suppose that a binary taxonomy tree includes 8 animals. What is the minimum number of NONLEAF nodes in the tree? Draw the tree

Algorithms on Trees - I

Depth on Trees:

- Depth is recursively defined:
 - Depth of root is 0
 - Otherwise depth of node 'v' is one more than *depth of parent node*

Algorithm depth(T,v):

if T.isRoot(v) then: return 0

else: return 1 + depth(T,T.parent(v))

- Running Time
 - $O(1 + d_v)$ where d_v is depth of node v in T \Rightarrow worst case $O(n)$

Algorithms on Trees - II

Height on trees:

- Also recursively defined
- Except root node, height of v is one more than the *height of child of v*

Algorithm height(T, v):

for each v in $T.nodes()$ do:

if T is External(v) then $h = \max(h, \text{depth}(T, v))$

return h

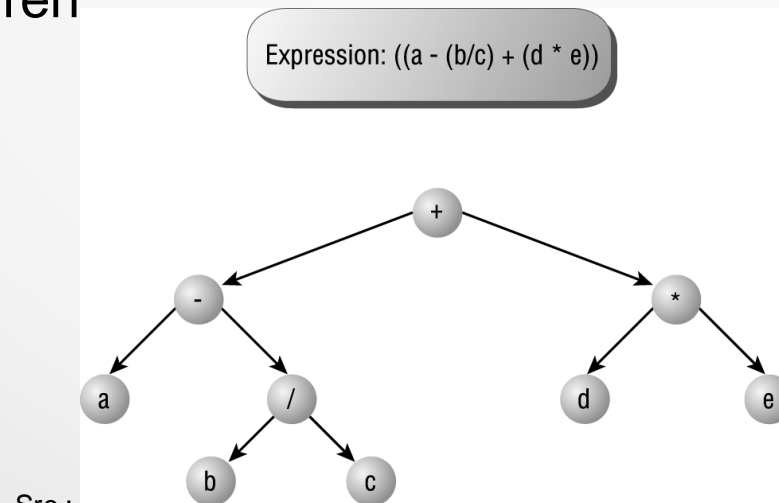
- Running Time

$O(n + \sum_{e \in E} (1 + d_v))$, where d_v is depth of node v in T

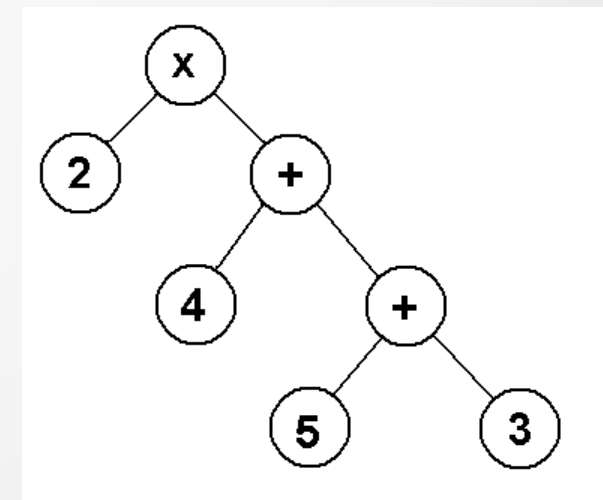
Worst Case is $O(n^2)$

Application: Binary Expression Tree

- Arithmetic expression can be represented by a binary tree
 - External nodes are variables or constants
 - Internal nodes are operators
 - Its value is defined by applying its operation to the value of its children



Src :
msdn.microsoft.com

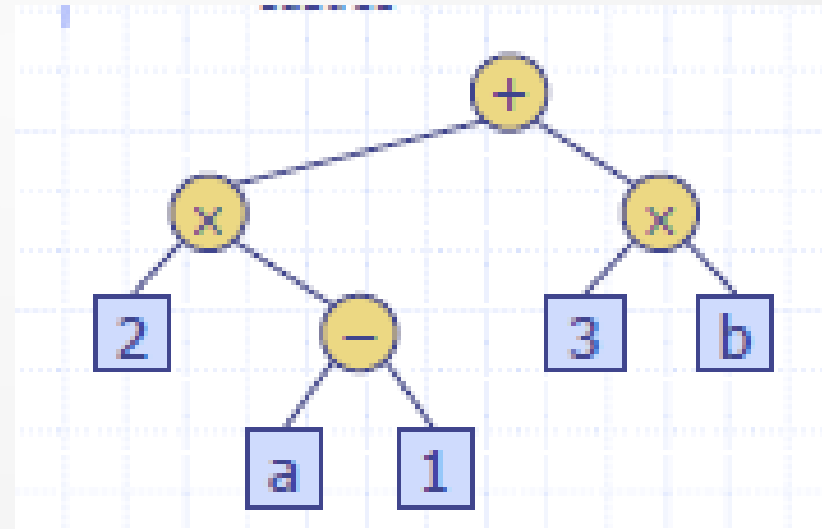


Src : www.andrew.cmu.edu

Printing Arithmetic Expressions

- Uses specialization of inorder traversal
 - print operand or operator when visiting node
 - print “(“ before traversing left subtree
 - print “)” after traversing right subtree
- Algorithm printExpression(v)

```
if isInternal (v)
    print("(")
    inOrder (leftChild (v))
print(v.element ())
if isInternal (v)
    inOrder (rightChild (v))
print(")")
```



Src: Goodrich Notes

Evaluate Arithmetic Expressions

- Specialization of a postorder traversal
 - recursive method returning the value of a subtree
 - if an internal node is visited, combine the values of the subtrees

Algorithm evalExpr(v)

if isExternal (v) return v.element ()

else

$x \leftarrow \text{evalExpr}(\text{leftChild}(v))$

$y \leftarrow \text{evalExpr}(\text{rightChild}(v))$

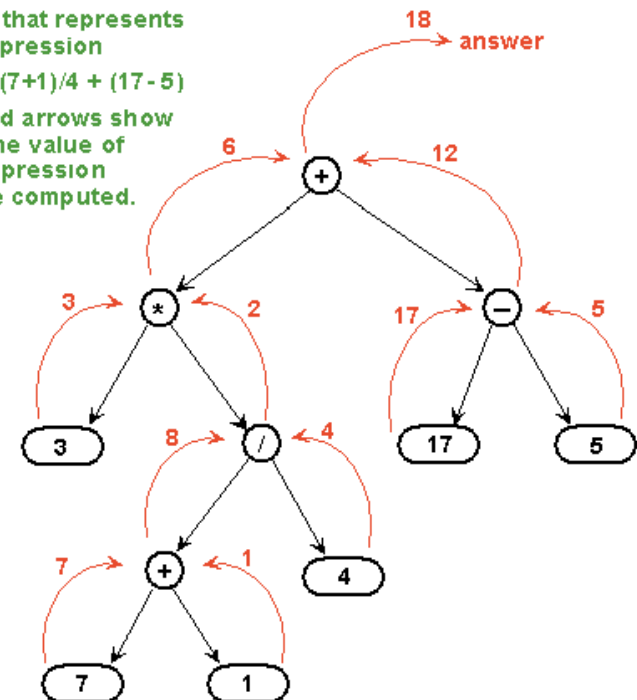
$O \leftarrow \text{operator stored at } v$

return $x \ O \ y$

A tree that represents the expression

$3 * (7 + 1) / 4 + (17 - 5)$

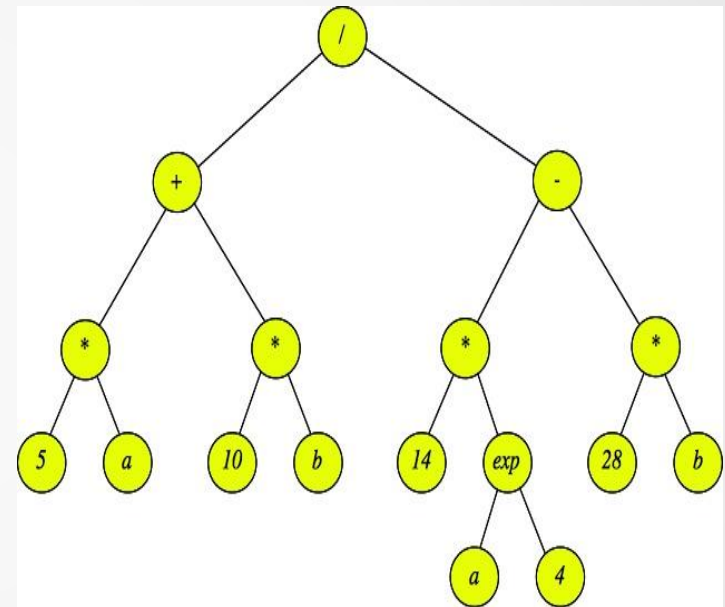
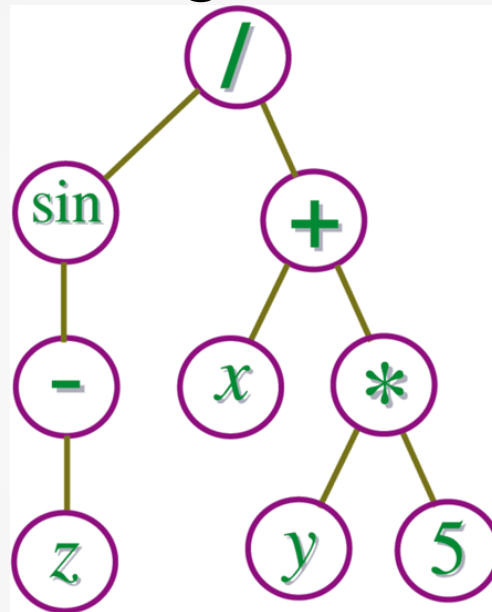
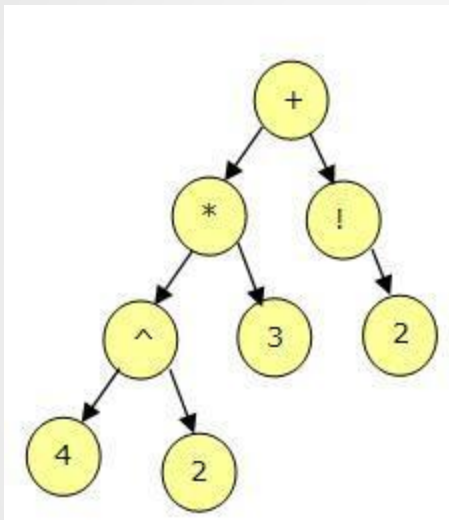
The red arrows show how the value of the expression can be computed.



Src: math.hws.edu

Exercises

Evaluate the following

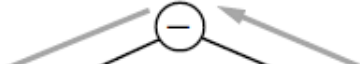


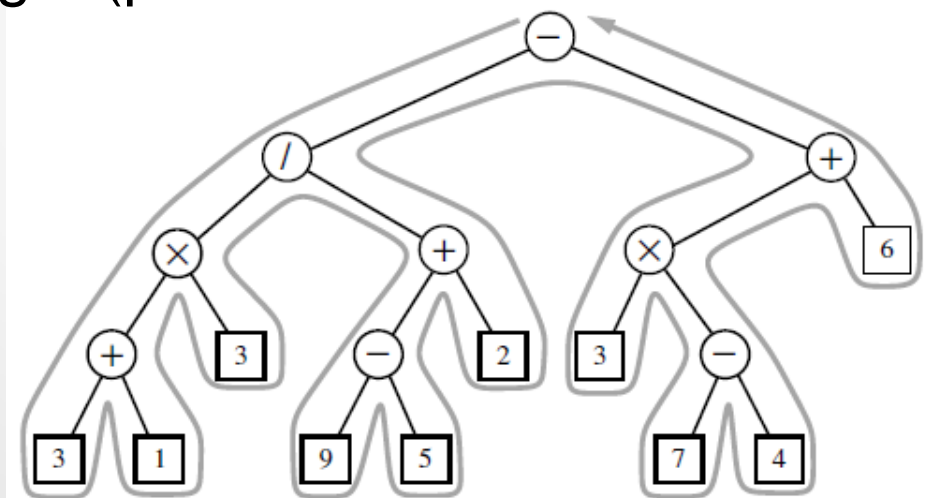
Draw expression trees for: $a + (b * c) - d + (e * f) / (g^h)$

Euler Tour Traversal

- An Euler tour is a trail in a tree which visits every edge exactly once
 - If it is undirected, it visits once in each direction
 - The Euler tour traversal of a general tree T can be informally defined as a “walk” around T , where we start by going from the root toward its leftmost child, viewing the edges of T as being “walls” that we always keep to our left
 - Euler tour of is a way of drawing each edge exactly once without taking your pen off the paper
 - The Euler tour traversal would draw each edge twice but if you add in the parent pointers, each edge is drawn once

Euler Tour Traversal

- Walk around the tree and visit each node three times:
 - on the left (preorder ie before Euler tour of v's left subtree)
 - from below (inorder ie between Euler's tour of both subtrees)
 - on the right (postorder ie after Euler tour of v's right subtree)
- 
- A diagram showing a node, represented by a circle with a minus sign inside. Three arrows point towards this node from below and to the sides, representing the three visits described in the list: from the left, from below, and from the right.



Euler Tour Traversal

- A “pre visit” occurs when first reaching the position, that is, when the walk passes immediately left of the node in our visualization.
- • A “post visit” occurs when the walk later proceeds upward from that position, that is, when the walk passes to the right of the node in our visualization.

Euler Tour Traversal

Algorithm EulerTour(T, v)

 visitLeft(T, v)

 if $T.\text{hasLeft}(v)$

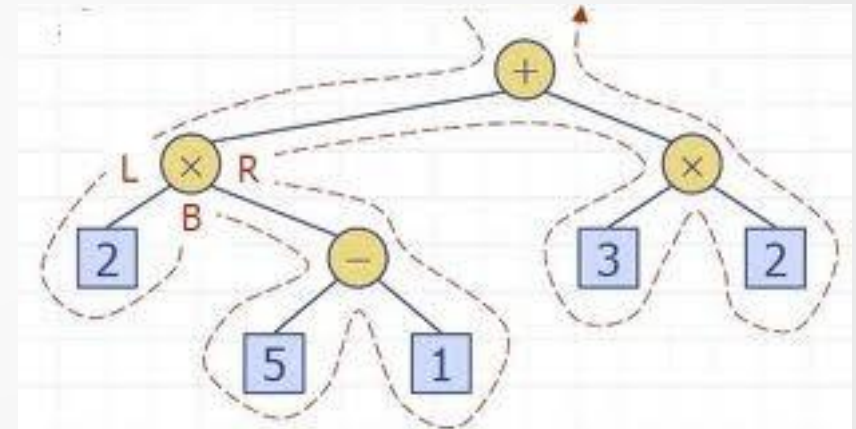
 eulerTour($T, \text{leftChild}(v)$)

 visitBelow(T, v)

 if $T.\text{hasRight}(v)$

 eulerTour($T, \text{rightChild}(v)$)

 visitRight(T, v)

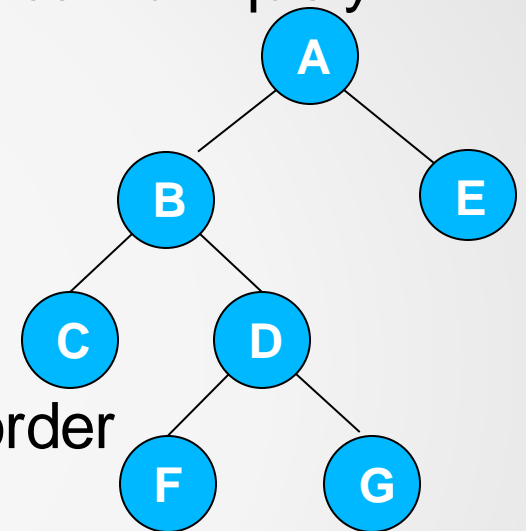


Did you notice

- While evaluating the expression tree is a specialization of the post order traversal, how can you print a tree?
 1. A level order representation
 2. The Euler Tour Traversal
 - Start from root and return to root

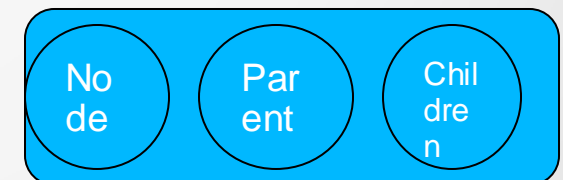
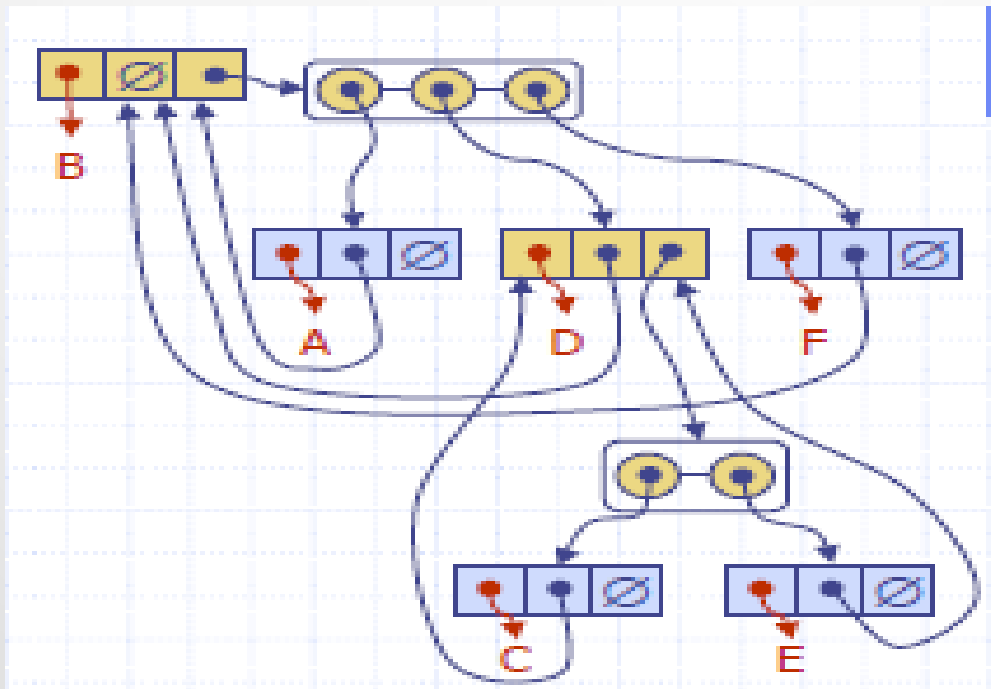
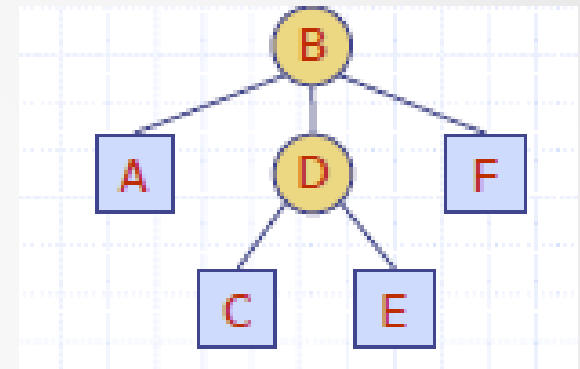
Application of Traversals

- Given the inorder and preorder traversals, we can uniquely determine a tree
- Eg:
 - Preorder: a,b,c,d,f,g,e
 - Inorder: c,b,f,d,g,a,e
- We can also use postorder in place of preorder



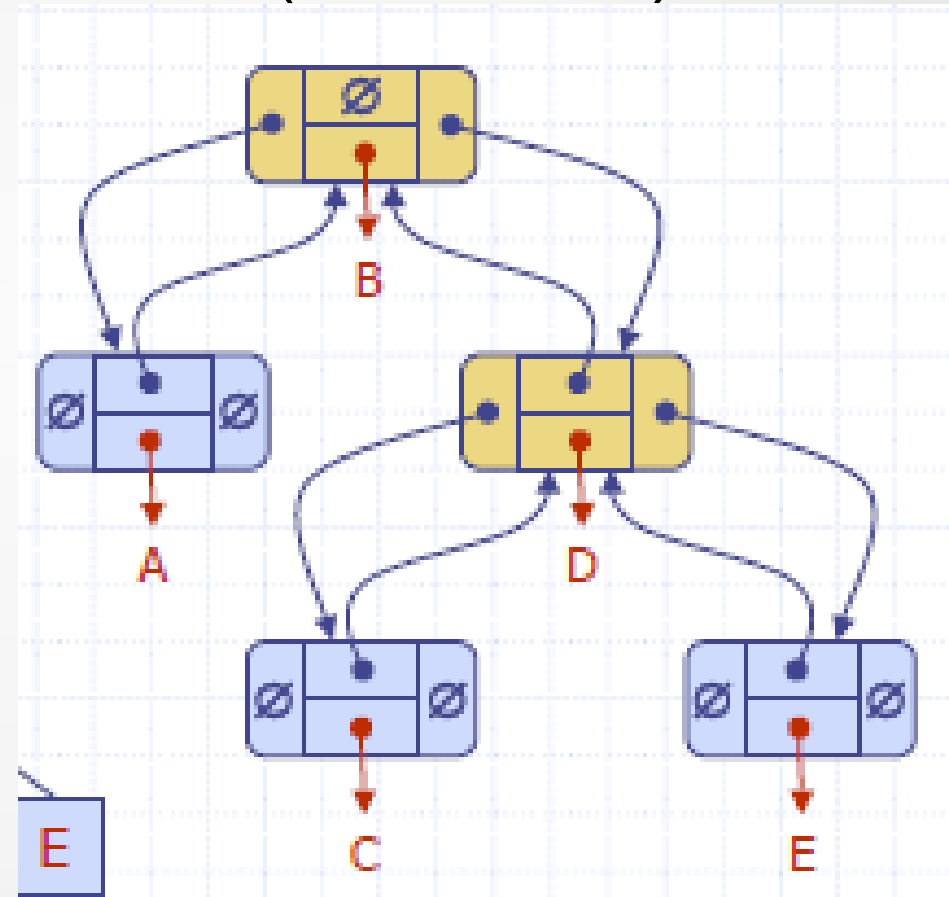
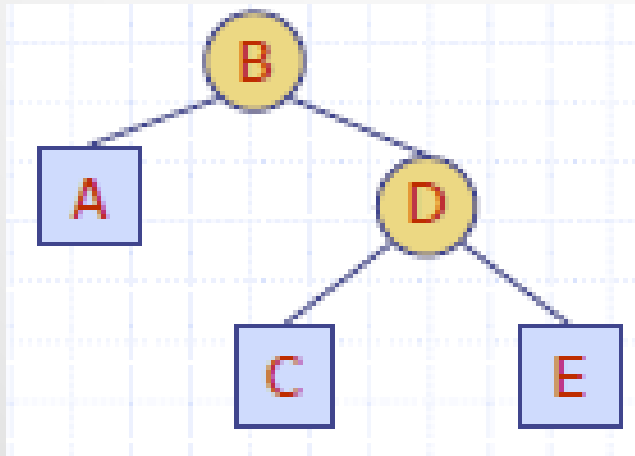
Data Structure for Trees: Linked Representation

- Node represented as follows (Method 1)
 - Element
 - Parent node
 - Sequence of children nodes



Linked Representation

- Node represented as follows (Method 2)
 - Element
 - Parent
 - Left Child Node
 - Right Child Node

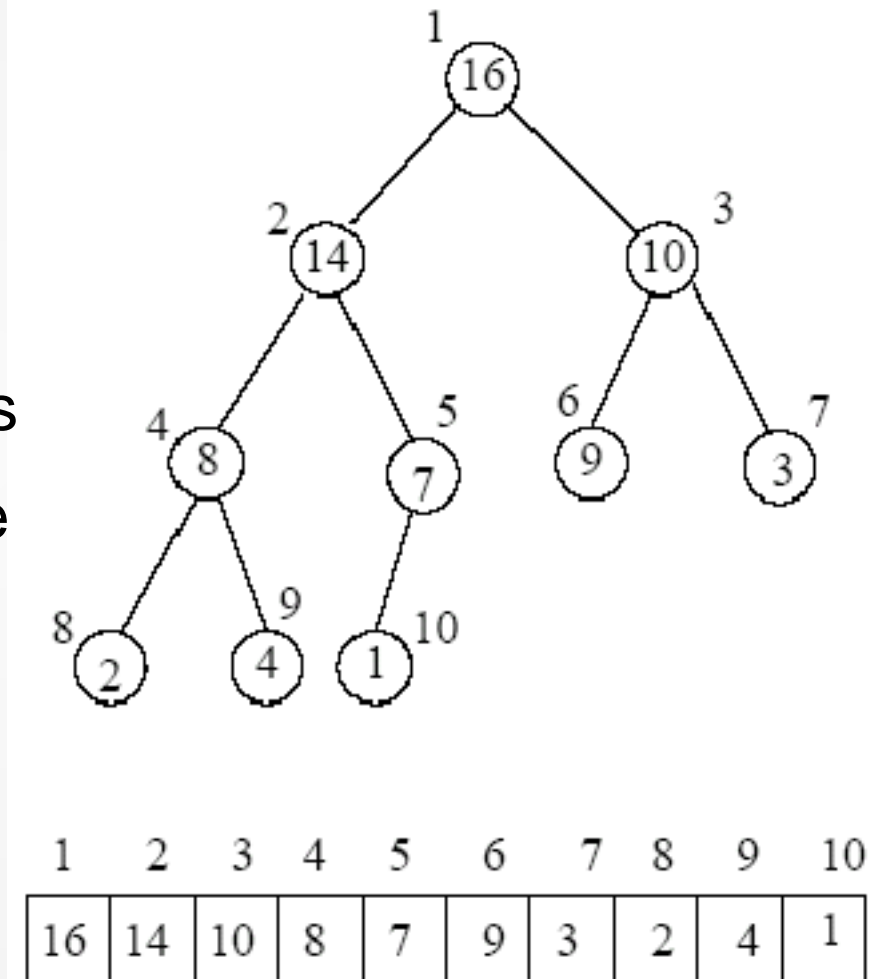


Vector based representation

- Based on number nodes
- For each node v , $p(v)$ defined as follows
 - If v is the root, $p(v) = 1$
 - If v is left child of node u , then $p(v) = 2p(u)$
 - If v is right child of node u , then $p(v) = 2p(u) + 1$
- Numbering function p is the “level numbering” of the nodes in a binary T
 - Numbers in each level increase from left to right
 - Numbers can be skipped

Vector based representation

- The different functions can be computed using arithmetic operations
- Extendible array representations can be used to manage the size



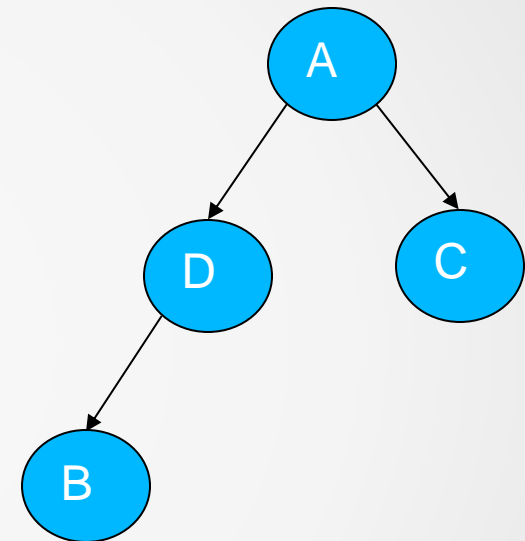
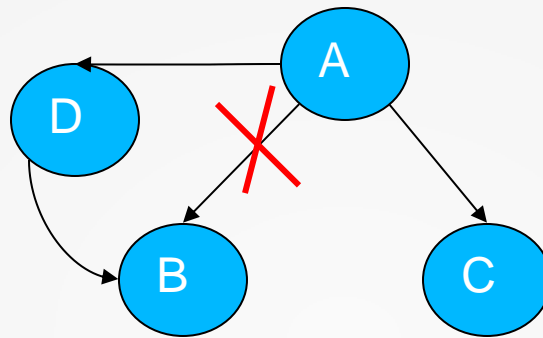
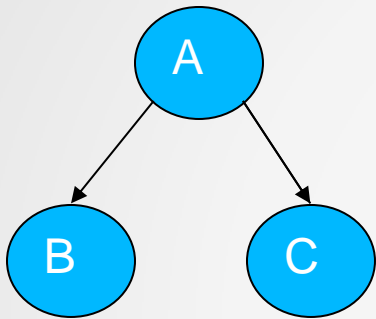
Binary Tree ADT

- leftChild(v):
 - Return left child of v
 - Error if v is external node
- rightChild(v)
 - Return right child of v
 - Error if v is external node
- Sibling(v)
 - Return sibling of a node v
 - Error if v is Root

Insertion

- Insertion: InsertAfter(node A)
 - A is an internal node
 - Let node B be child of A
 - A assigns its child to the new node and the new node assigns its parent to A.
 - New node assigns its child to B and B assigns its parent as the new node
 - A is an external node
 - A assigns the new node as one of its children
 - The new node assigns node A as its parent

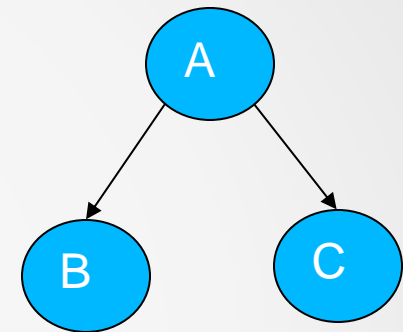
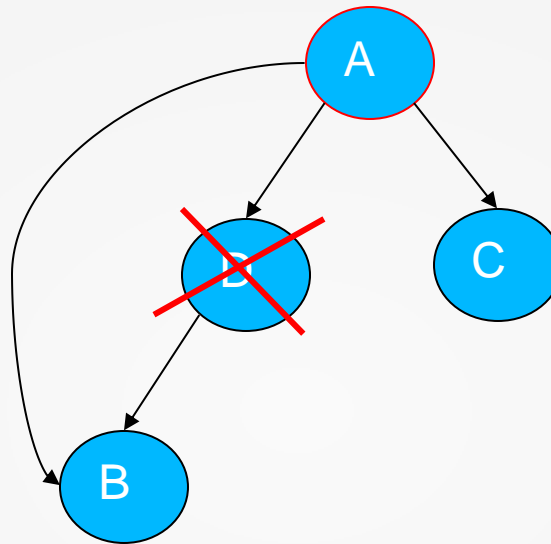
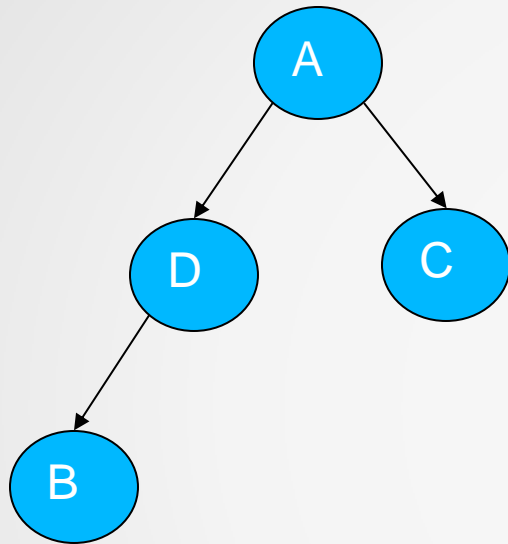
Example



Deletion

- Delete(node A)
 - A is an external node
 - set the corresponding child of A's parent to null.
 - A has one child
 - set the parent of A's child to A's parent
 - set the child of A's parent to A's child
 - A has two children
 - Usually not recommended
 - Involves more complex operations

Example



End of Lecture 9