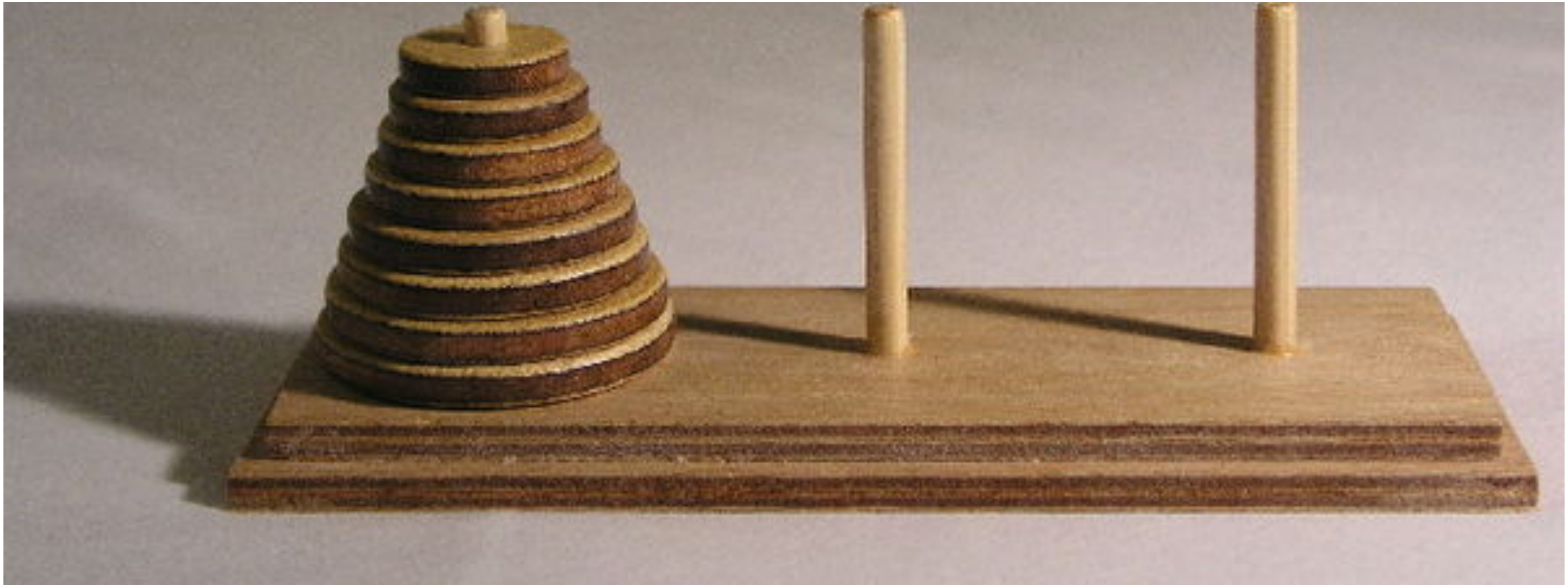
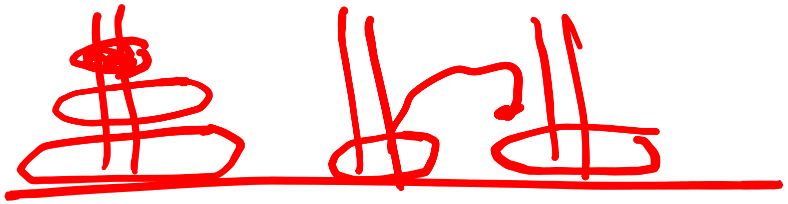
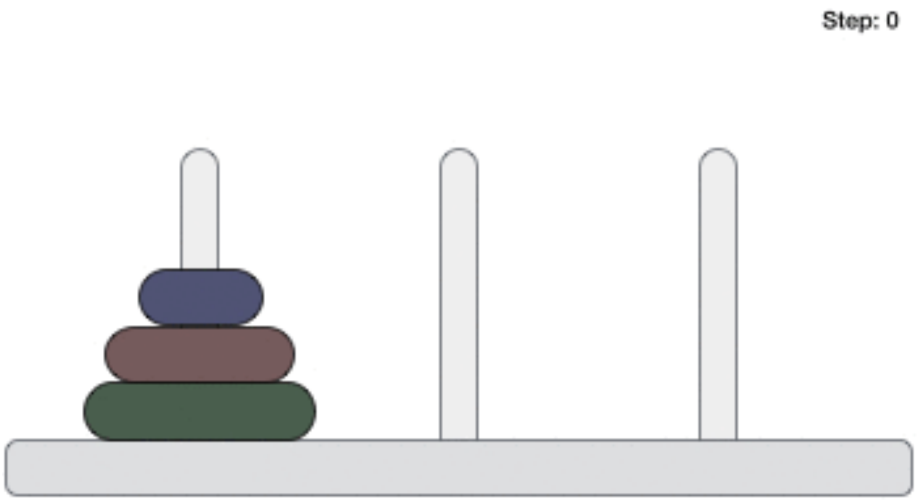
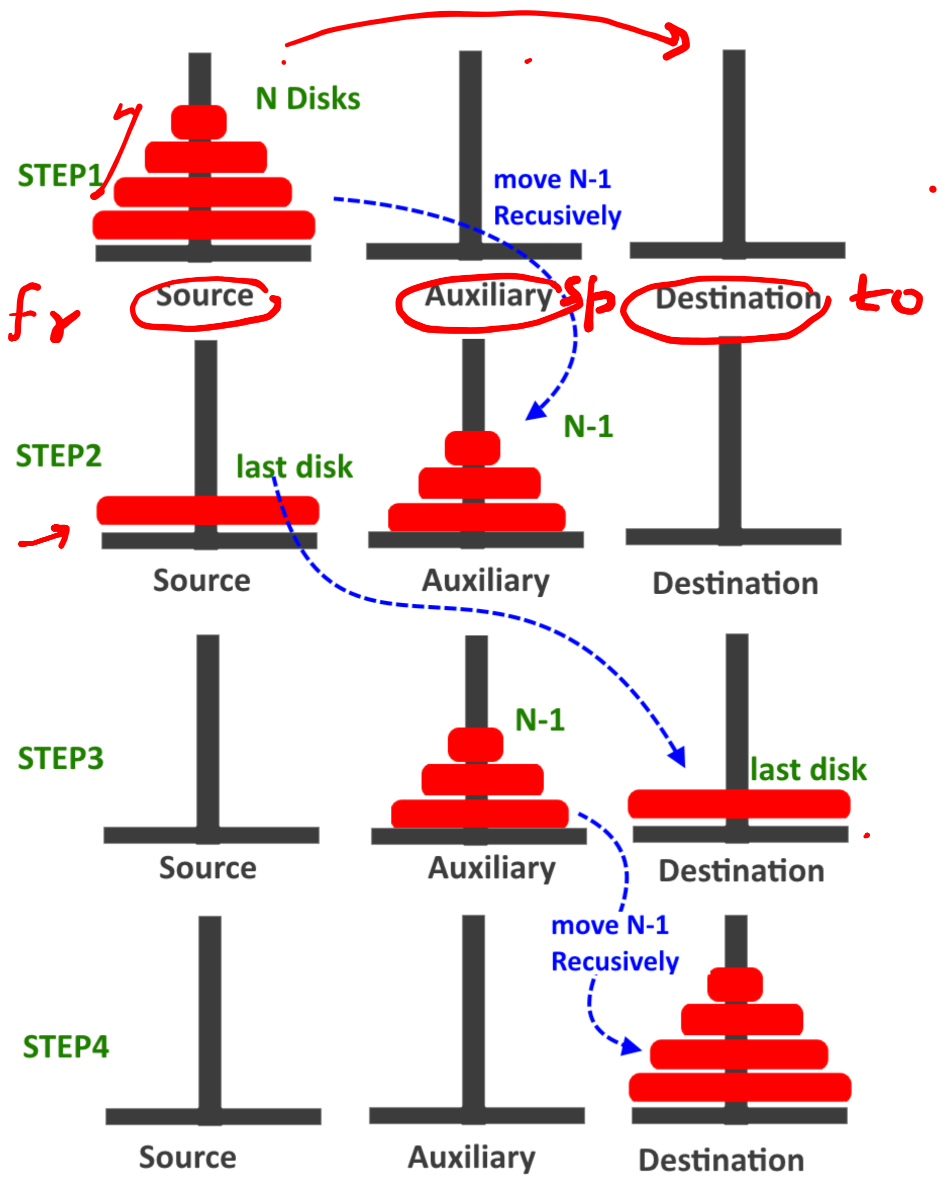




Towers of Hanoi





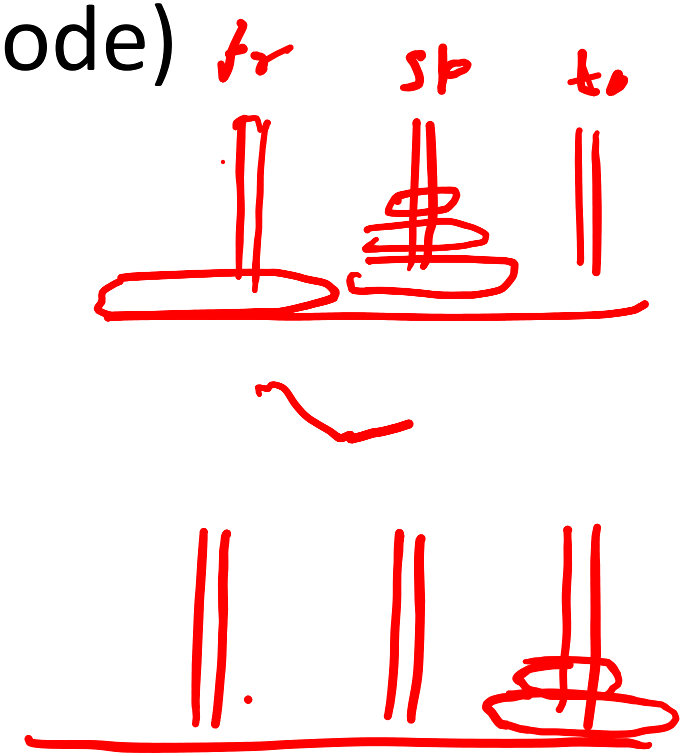
→ n-1 disks
to Aux

→ last disk
to dest

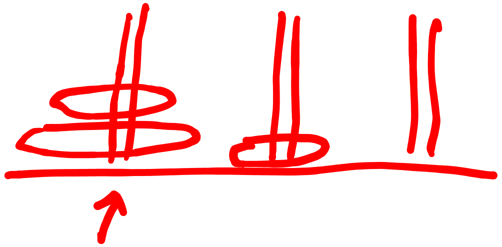
→ n-1
aux → dest

Towers of Hanoi (3 line code)

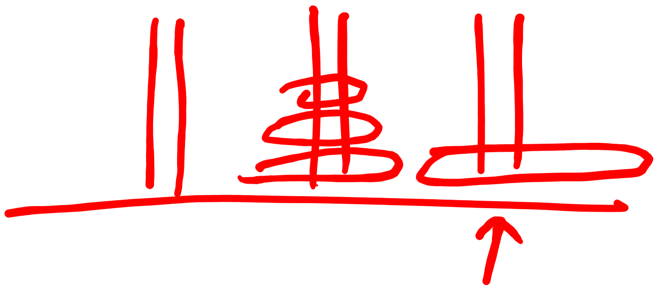
```
Towers(n, fr, to, sp)
    if (n==1)
        printmove(fr, to);
    {
    Towers(n-1, fr, sp, to);
    Towers(1, fr, to, sp);
    Towers(n-1, sp, to, fr);
    }
}
```



Towers of Hanoi



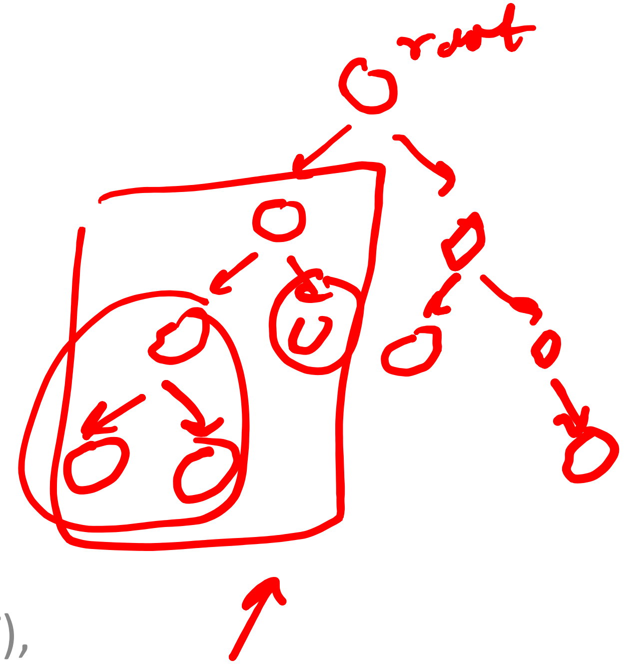
```
printMove(fr, to)
Print('move disk from' + str(fr) + 'to' + str(to));

Towers(n,fr,to,spare)
     if(n==1)
          printMove(fr,to);
else
     Towers(n-1,fr,spare,to);
     Towers(1,fr,to,spare);
     Towers(n-1,spare,to,fr);
```



Advanced Problem Solving (CSE603)

Lecture # 07

Trees



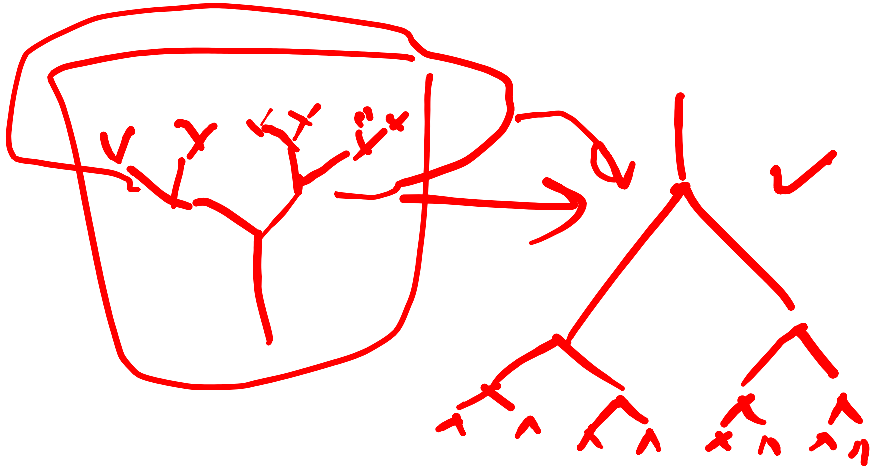
Avinash Sharma

Center for Visual Information Technology (CVIT),

IIIT Hyderabad

Introduction

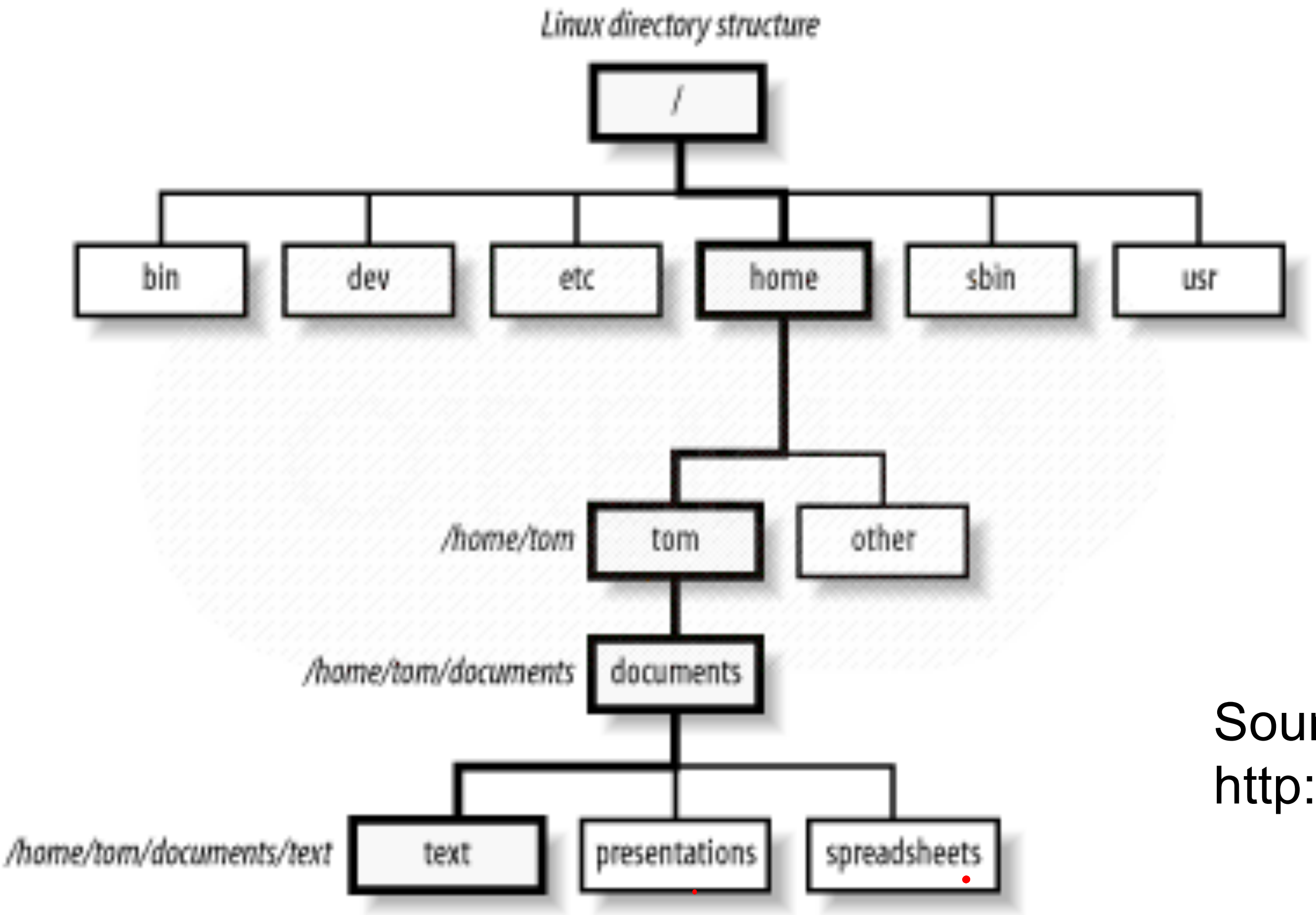
- The story so far
  - Saw some fundamental operations as well as advanced operations on arrays, stacks, and queues
  - Saw a dynamic data structure, the linked list, and its applications.
- This week we will
  - Study data structures for hierarchical data
  - Operations on such data.
  - Leading to efficient insert/delete/find.



Motivation

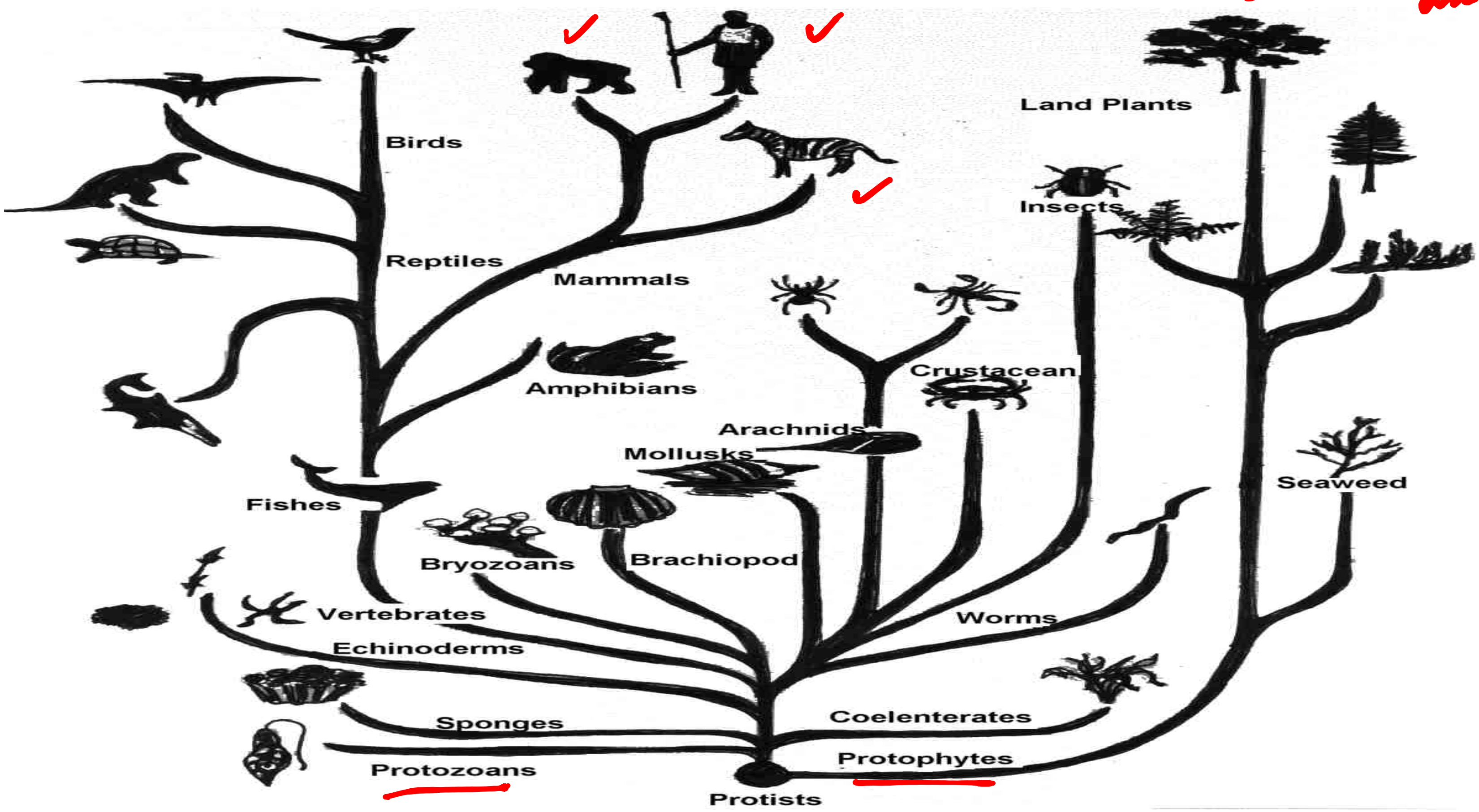
- Consider your home directory.
- /home/user is a directory, which can contain sub-directories such as work/, misc/, songs/, and the like.
- Each of these sub-directories can contain further sub-directories such as ds/, maths/, and the like.
- An extended hierarchy is possible, until we reach a file.

Motivation



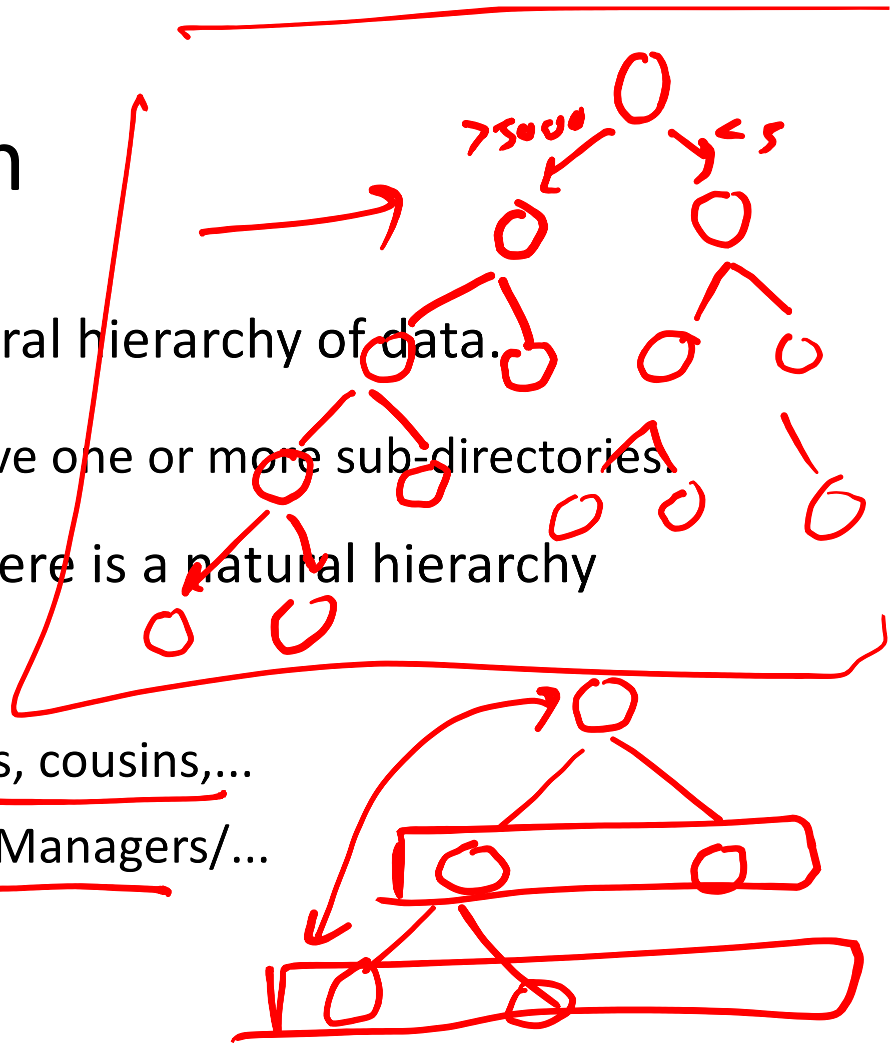
Source: http://khalihari.blogspot.in/

Motivation


Birds
Reptiles
Mammals
Amphibians
Fishes
Land Plants
Insects
Crustacean
Arachnids
Mollusks
Seaweed
Bryozoans
Brachiopod
Vertebrates
Echinoderms
Worms
Sponges
Coelenterates
Protozoans
Protophytes
Protists

Motivation

- In all of the above examples, there is a natural hierarchy of data.
  - In the first example, a (sub)directory can have one or more sub-directories.
- Similarly, there are several setting where there is a natural hierarchy among data items.
  - Family trees with parents, ancestors, siblings, cousins,...
  - Hierarchy in an organization with CEO/CTO/Managers/...

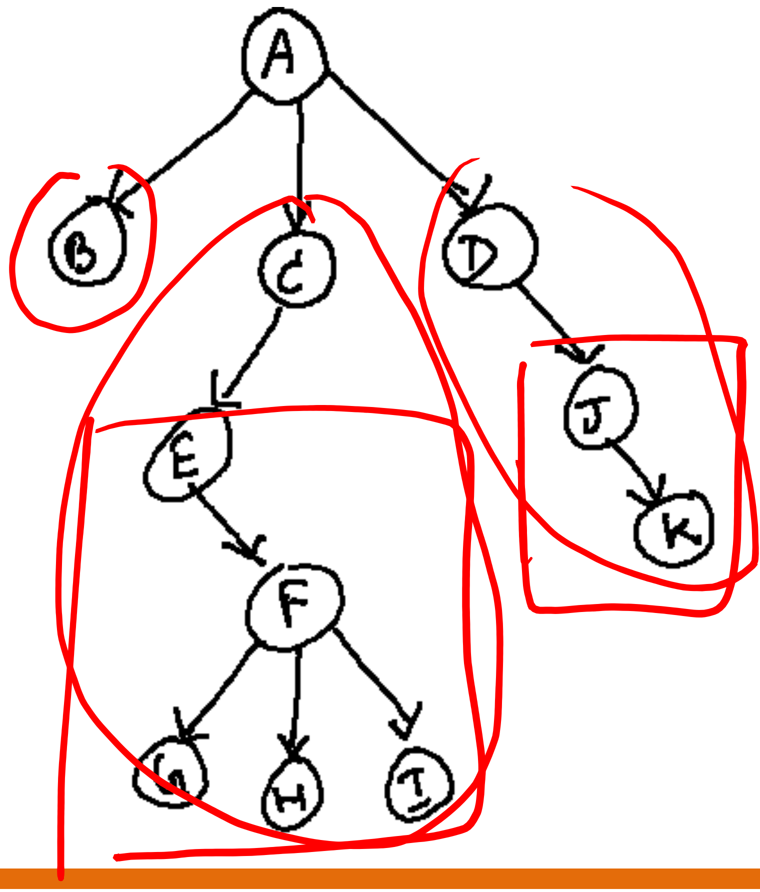


The Tree Data Structure

- A tree on n nodes always has n-1 edges.
- Why?
  - One parent for every one, except the root.
- Before going in to how a tree can be represented, let us know more about the tree.

The Tree Data Structure

- Consider the tree shown to the right.
- The node A is the root of the tree.
- It has three subtrees whose roots are B, C, and D.
- Node C has one subtree with node E as the root.



The Tree Data Structure (terms)

A ← root

→ D, E, F

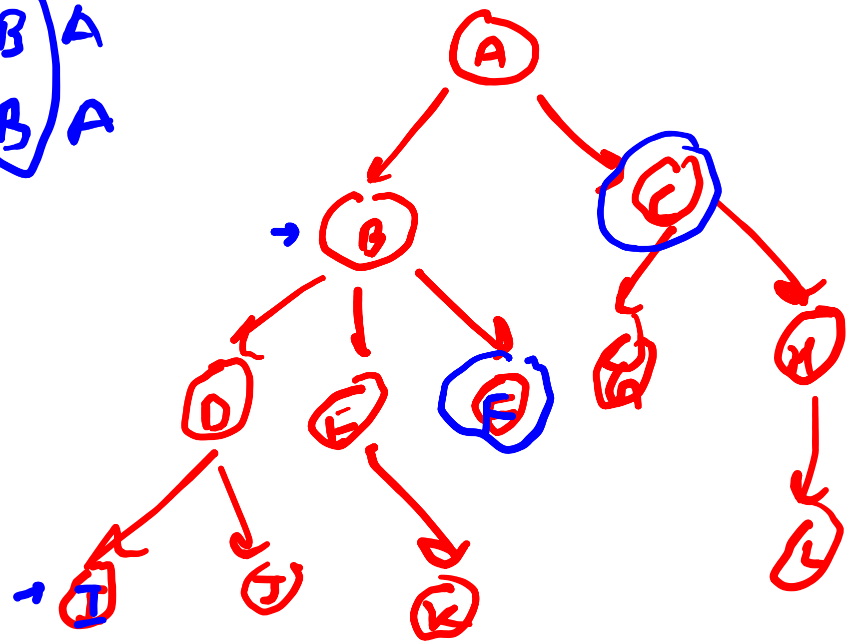
→ if we can go from P to Q

P is an ancestor of Q

→ F, G ~~k~~

B A

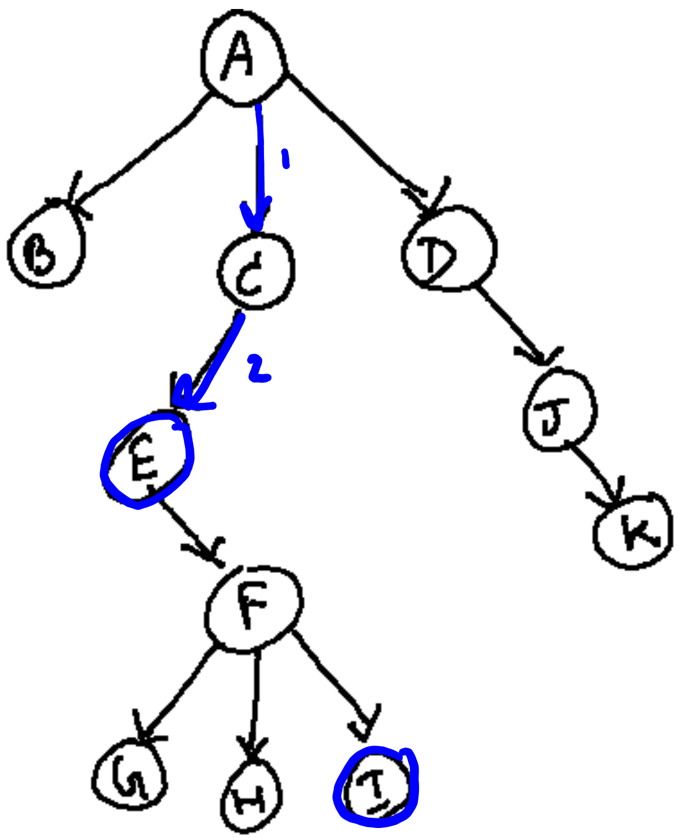
D B A



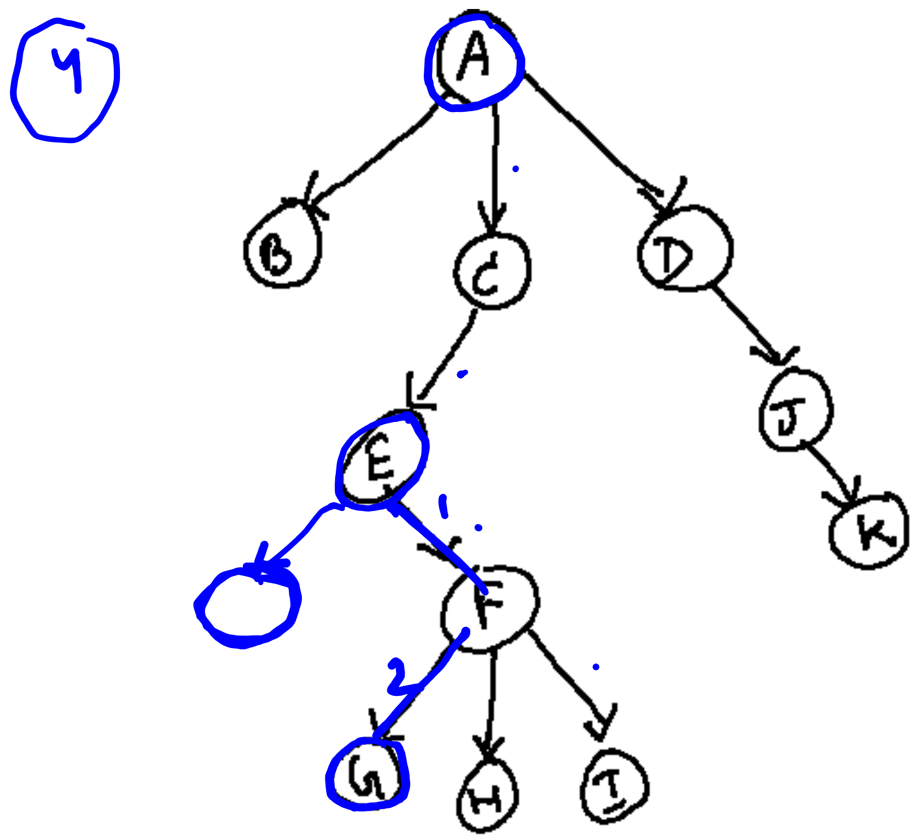
Depth

length

$\boxed{\text{E is 2}}$

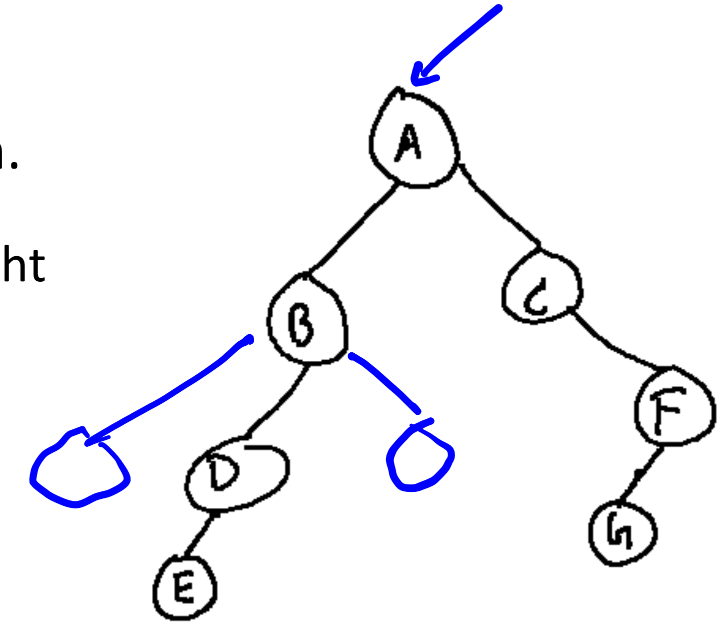


Height



Binary Trees

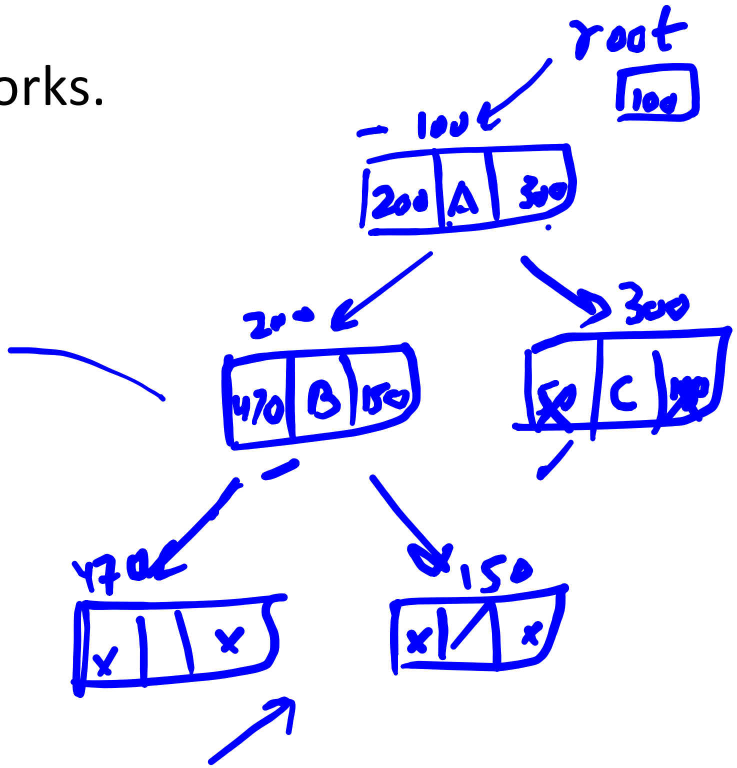
- A special class of the general trees.
- Restrict each node to have at most two children.
  - These two children are called the left and the right child of the node.
  - Easy to implement and program.
  - Still, several applications.



Binary Trees (implementation)

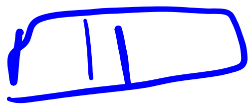
- Briefly, we also mention how to implement the tree data structure.
- The following node declaration as a structure works.

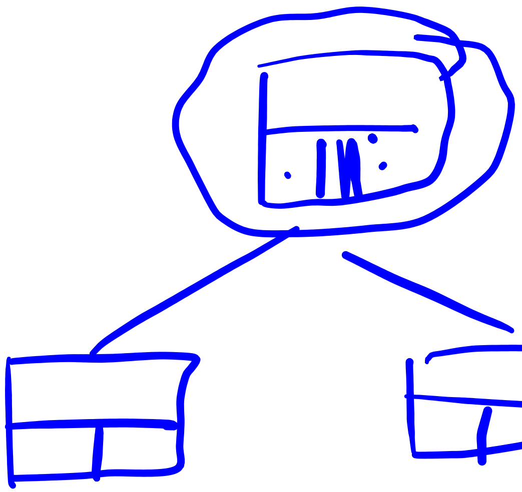
```
struct node {
    int data;
    node *left;
    node *right
};
```



Binary Trees

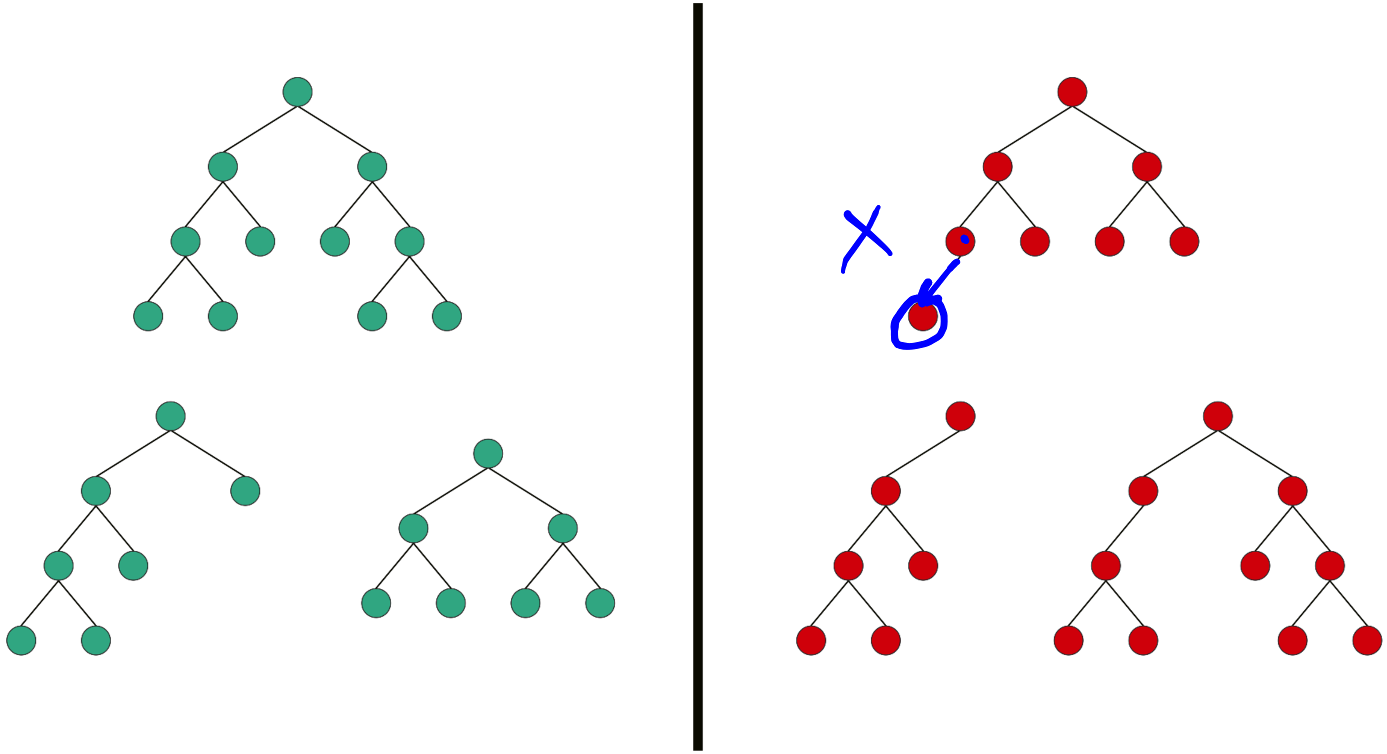
```
struct node {
    int data;
    node *left;
    node *right
};
```





Full Binary tree

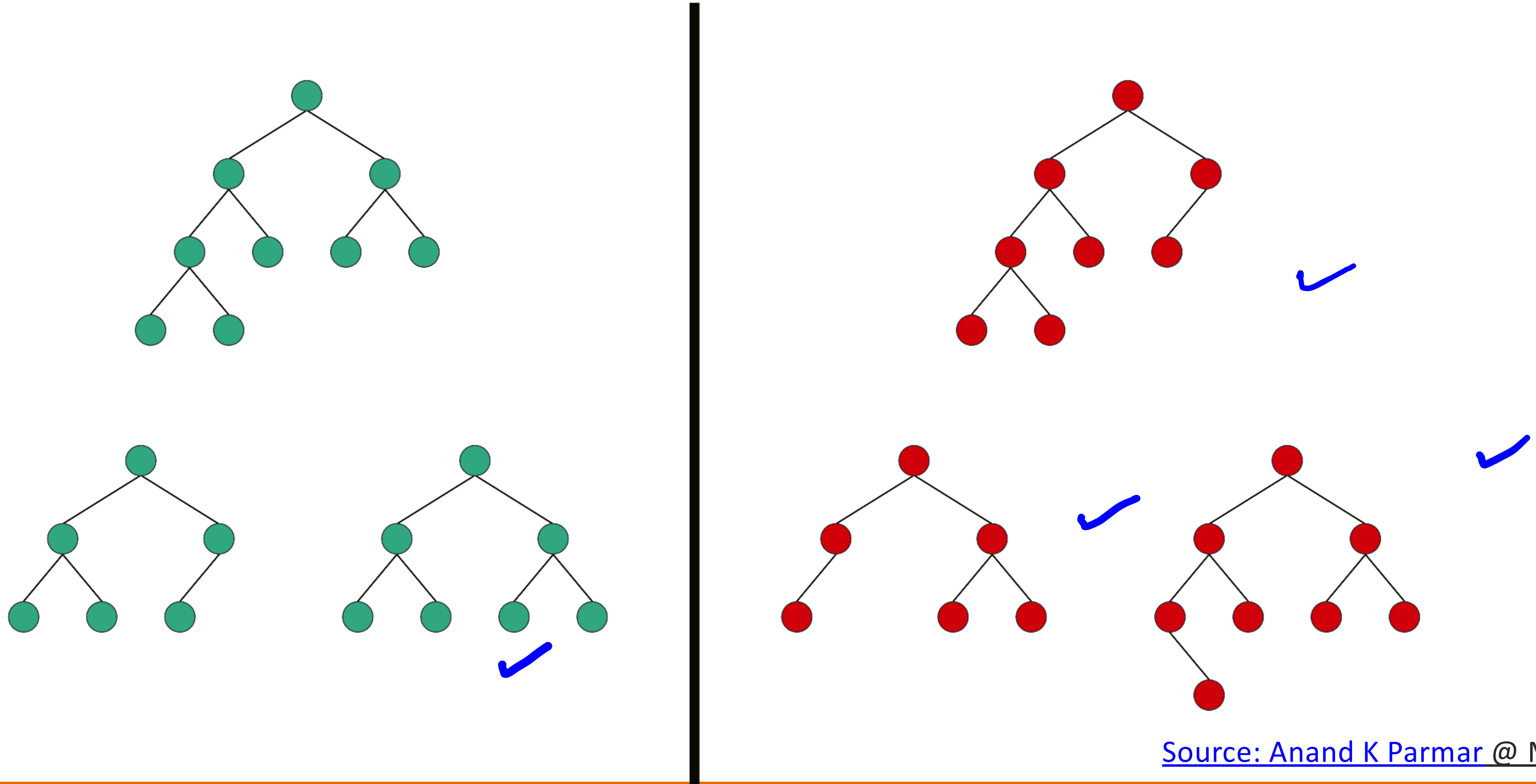
Full Binary Tree is a Binary Tree in which every node has 0 or 2 children.



Source: Anand K Parmar @ Medium

Complete Binary Tree

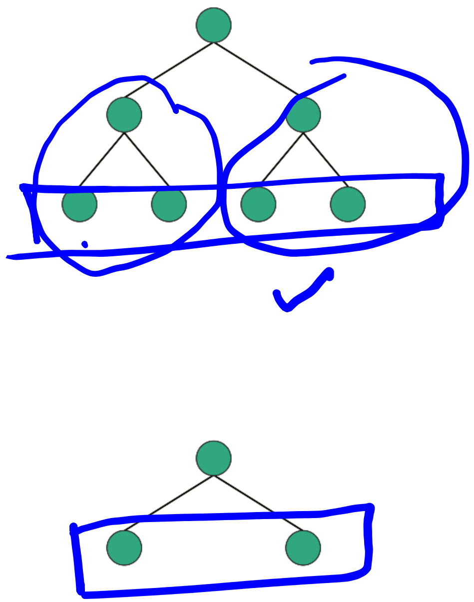
Complete Binary Tree has all levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.

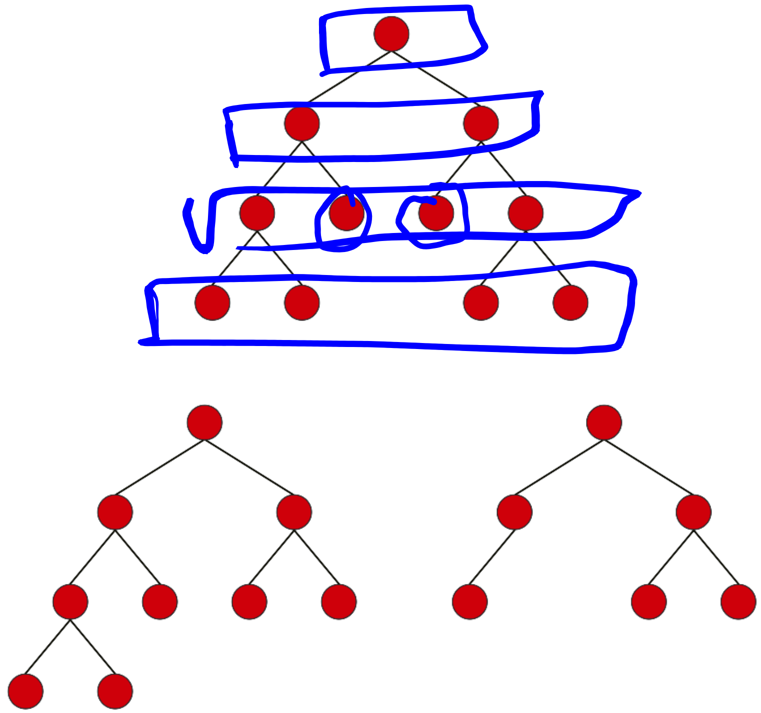


[Source: Anand K Parmar](#) @ Medium

Perfect Binary Tree

Perfect Binary Tree is a Binary Tree in which all internal nodes have 2 children and all the leaf nodes are at the same depth or same level.

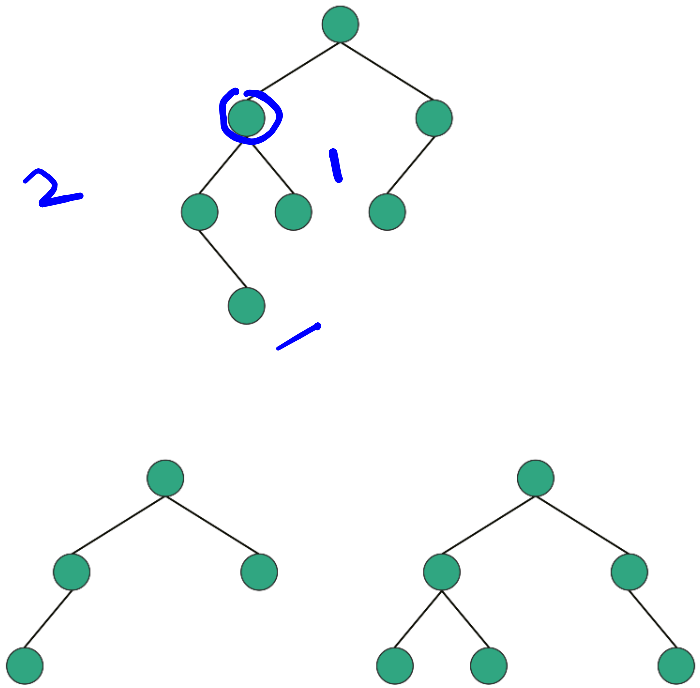


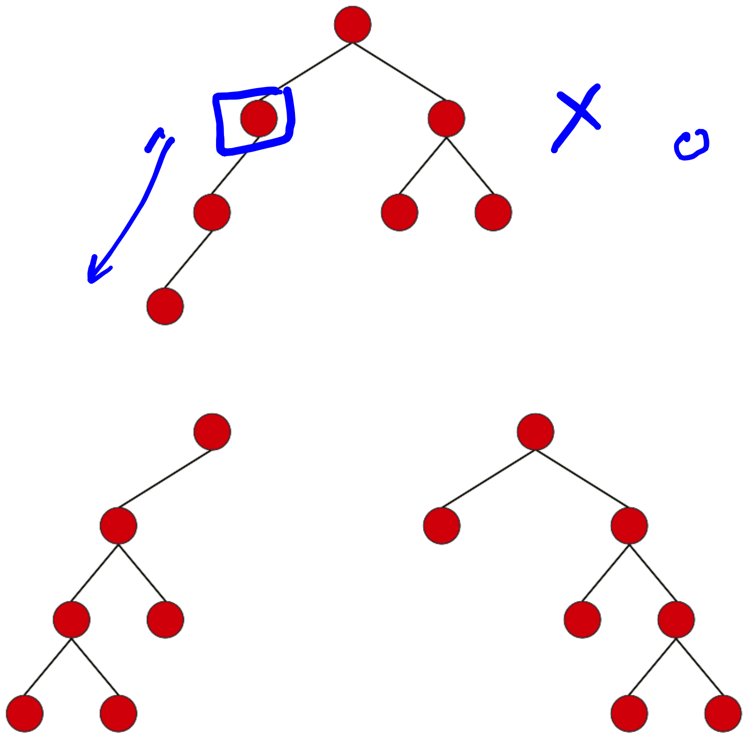


Source: Anand K Parmar @ Medium

Balanced Binary Tree

Balanced Binary Tree is a Binary tree in which height of the left and the right sub-trees of every node may differ by at most 1.

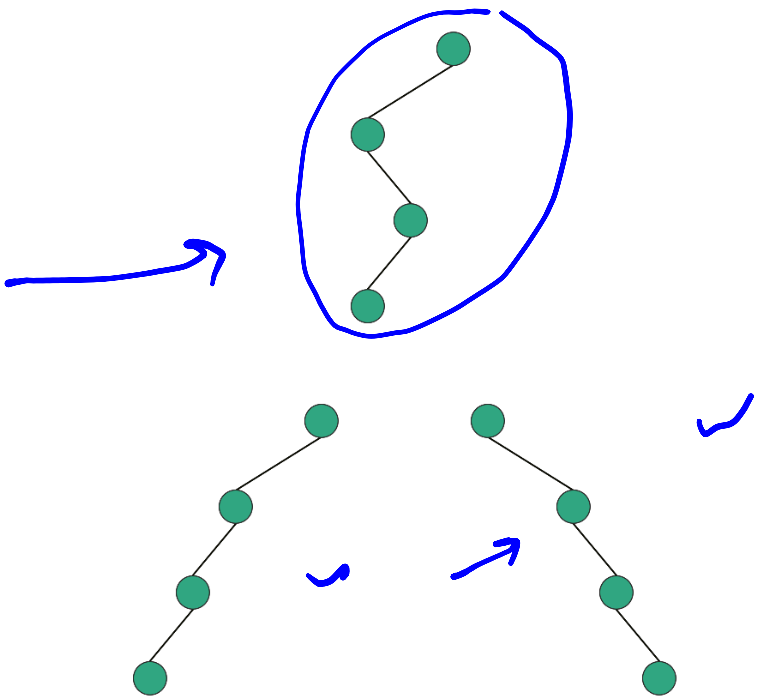


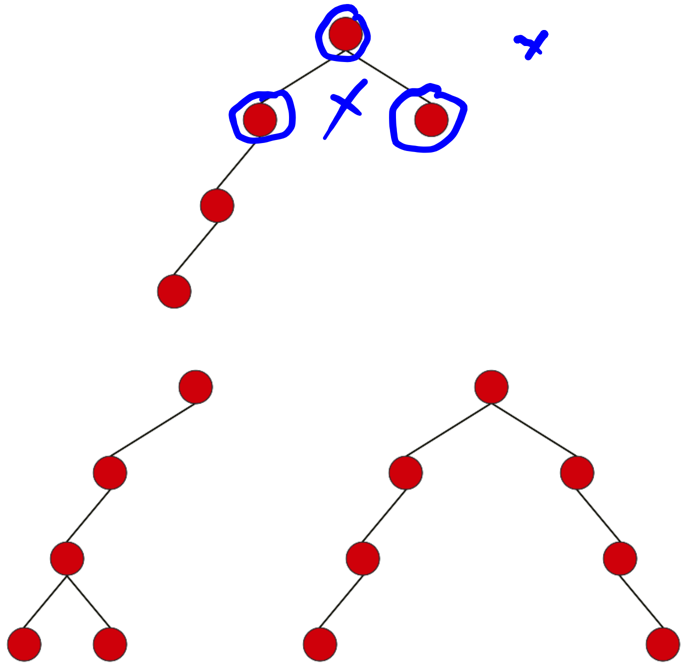


Source: Anand K Parmar @ Medium

Degenerate Binary Tree

Degenerate Binary Tree is a Binary Tree where every parent node has only one child node.

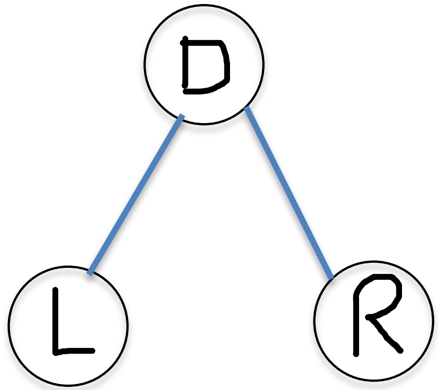




Source: Anand K Parmar @ Medium

Our First Operation

- To print the nodes in a (binary) tree
- This is also called as a traversal.
- Need a systematic approach
  - ensure that every node is indeed printed
  - and printed only once.
- Several methods possible. Attempt a categorization.
- Consider a tree with a root D and L, R being its left and right sub-trees respectively.



T→ left

Tree Traversal

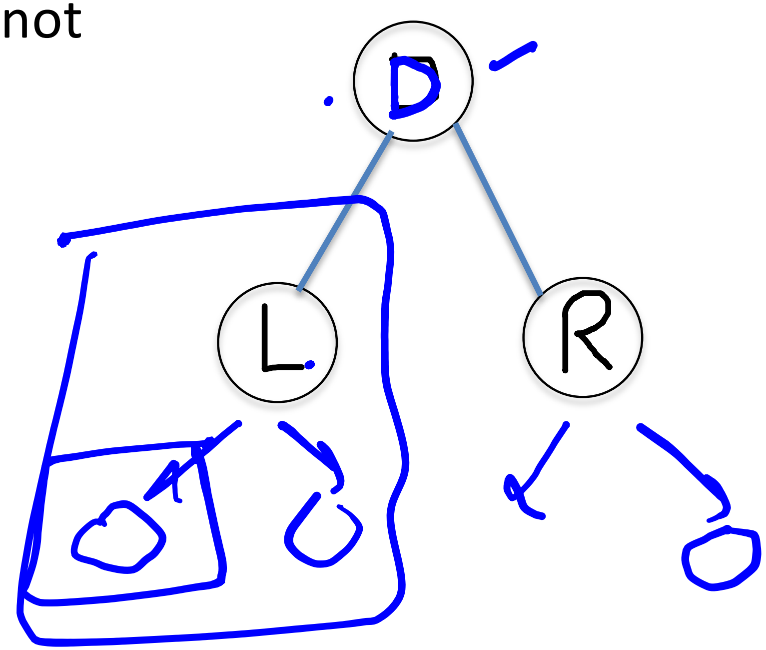
→ <left> <right> <root>

→ $\boxed{\text{<root> <left> <right>}}$

→ <left> <root> <right>

- Of these, let us make a convention that R can not precede L in any traversal.
- We are left with three:
  - L R D
  - L D R
  - D L R
- We will study each of the three. Each has its own name.

D

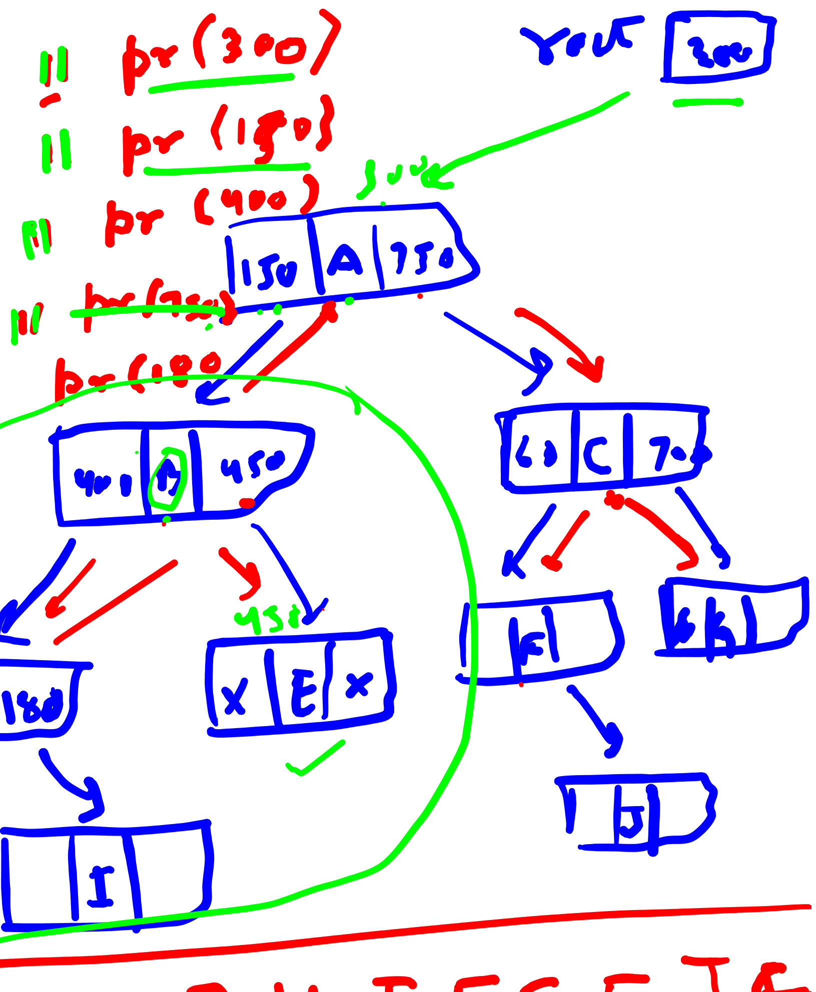


Preorder Traversal

Tree Traversal

<root><left><right>

```
void Preorder(Node *root) {
    if (root == Null) return;
    print("%c", root->data);
    Preorder(root->left);
    Preorder(root->right);
}
```

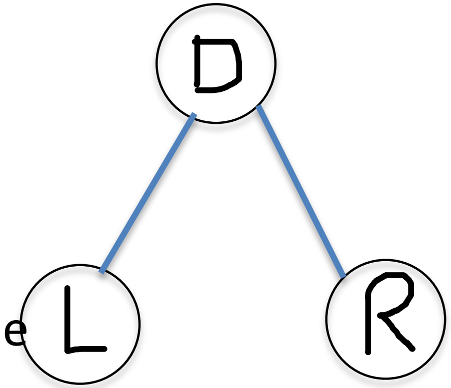


A B D H I E C F J G

Tree Traversal

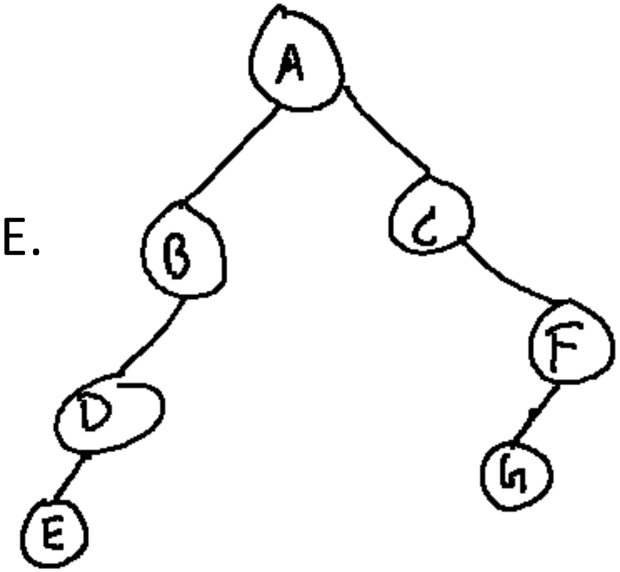
The Inorder Traversal (LDR)

- The traversal that first completes L, then prints D, and then traverses R.
- To traverse L, use the same order.
  - First the left subtree of L, then the root of L, and then the right subtree of L.



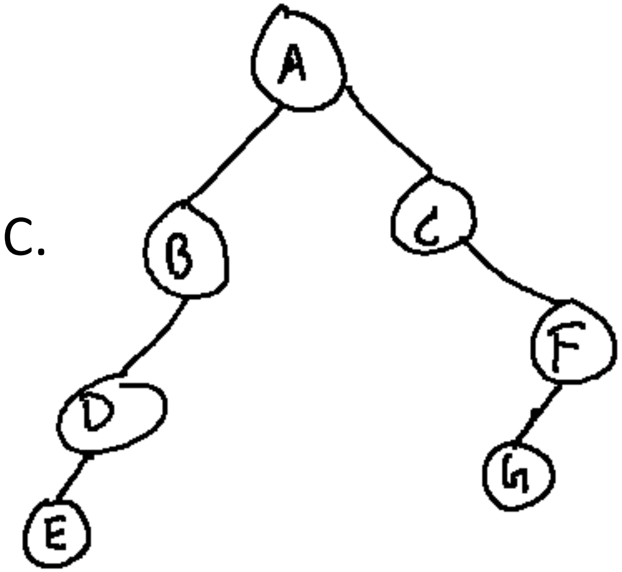
The Inorder Traversal

- Start from the root node A.
- We first should process the left subtree of A.
- Continuing further, we first should process the node E.
- Then come D and B.
- The L part of the traversal is thus E D B.



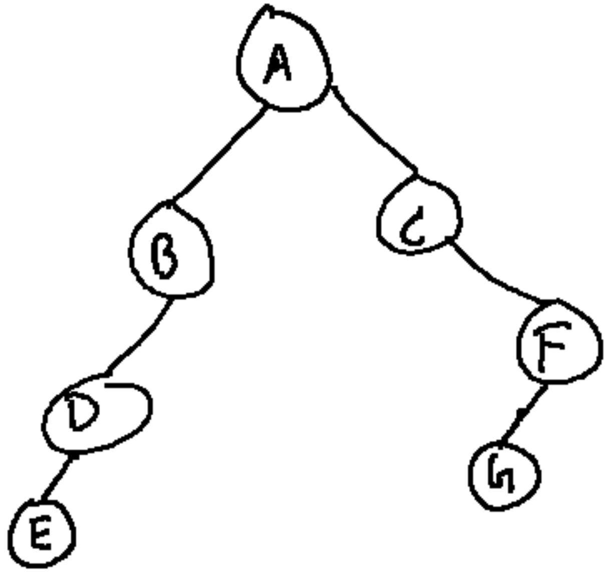
The Inorder Traversal

- Then comes the root node A.
- We next process the right subtree of A.
- Continuing further, we first should process the node C.
- Then come G and F.
- The R part of the traversal is thus C G F.



The Inorder Traversal

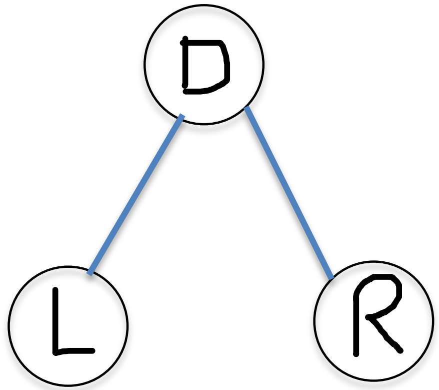
```
Procedure Inorder(T)
begin
      if T == NULL return;
      Inorder(T->left);
      print(T->data);
      Inorder(T->right);
end
```



Inorder: E D B A C G F

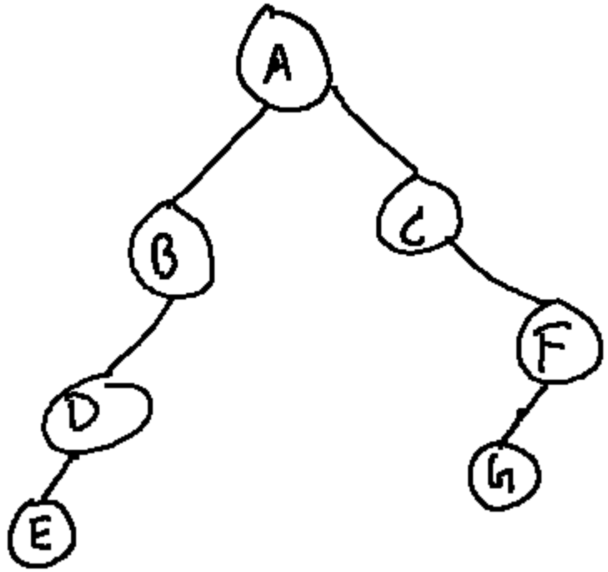
The Postorder Traversal (LRD)

- The traversal that first completes L, then traverses R, and then prints D.
- To traverse L, use the same order.
  - First the left subtree of L, then the right subtree of R, and then the root of L.



The Postorder Traversal

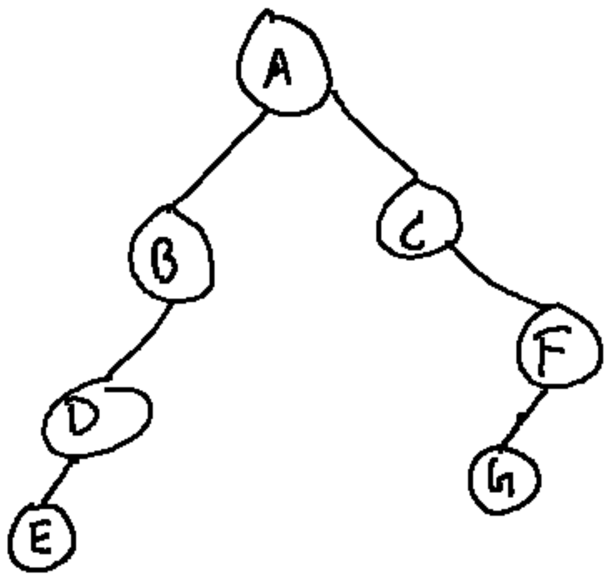
- We next process the right subtree of A.
- Continuing further, we first should process the node C.
- Then come G and F.
- The R part of the traversal is thus G F C.
- Then comes the root node A.



postorder: E D B G F C A

The Postorder Traversal

```
Procedure postorder(T)
begin
    if T == NULL return;
    Postorder(T->left);
    Postorder(T->right);
    print(T->data);
end
```



postorder: E D B G F C A

Another Kind of Traversal

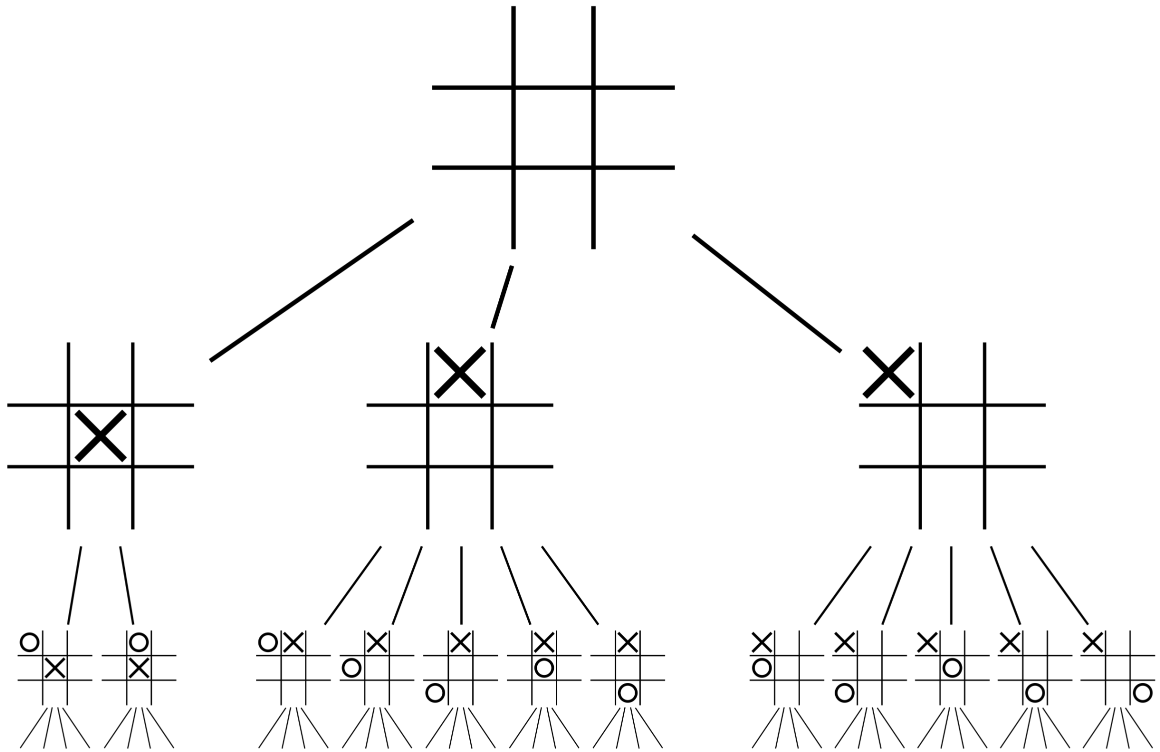
- When left and right subtree nodes can be intermixed.
- One useful traversal in this mode is the level order traversal.
- The idea is to print the nodes in a tree according to their level starting from the root.

Another Kind of Traversal

- Why would any one want to do that?
- One example:
  - Think of printing the organization chart.
  - Start with the CEO, there are CTO, CFO, and COO, say.
  - Then, five managers under the CTO, 2 managers under the CFO, and so on,
  - Each manager has more Assistant Managers who work with a team.
  - Want to list this in that order.
- There are other such examples too
  - Game trees

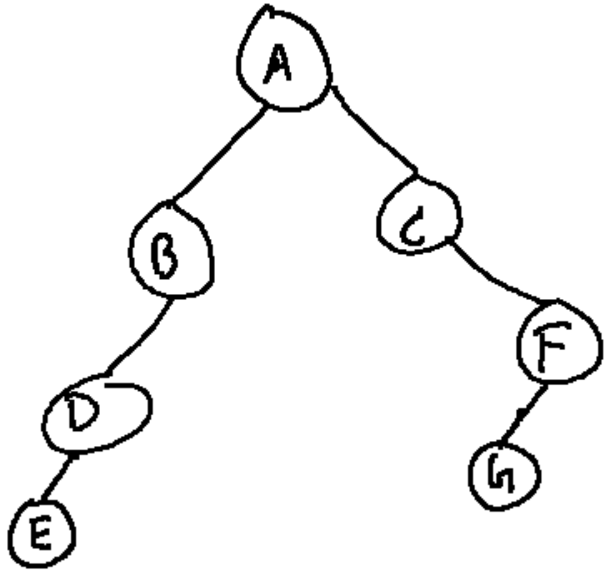
How to Perform a Depth Order Traversal

- Game Tree Example



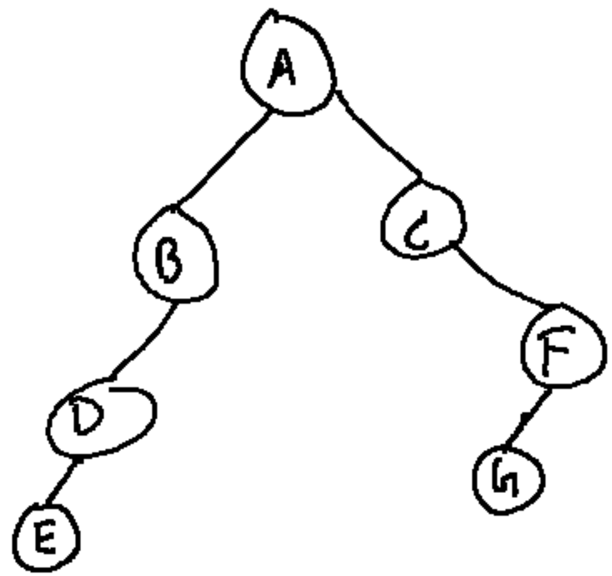
How to Perform a Depth Order Traversal

- Consider the same example tree.
- Starting from the root, so A is printed first.
- What should be printed next?
- Assume that we use the left before right convention.
- So, we have to print B next.
- How to remember that C follows B.
- And then D should follow C?



How to Perform a Depth Order Traversal

- Indeed, can remember that B and C are children of A.
- But, have to get back to children of B after C is printed.
- For this, one can use a queue.
  - Queue is a first-in-first-out data structure.



How to Perform a Depth Order Traversal

- The idea is to queue-up children of a parent node that is visited recently.
- The node to be visited recently will be the one that is at the front of the queue.
  - That node is ready to be printed.
- How to initialize the queue?
  - The root node is ready!

How to Perform a Depth Order Traversal

```
Procedure DepthOrder(T)
begin
    Q = queue;
    insert root into the queue;
    while Q is not empty do
        v = delete();
        print v->data;
        if v->left is not NULL insert v->left into Q;
        if v->right is not NULL insert v->right into Q;
    end-while
end
```

How to Perform a Depth Order Traversal

- Queue and output are shown at every stage.

Queue

A

B C

C D

D F

F E

E G

G

EMPTY

Output

A

B

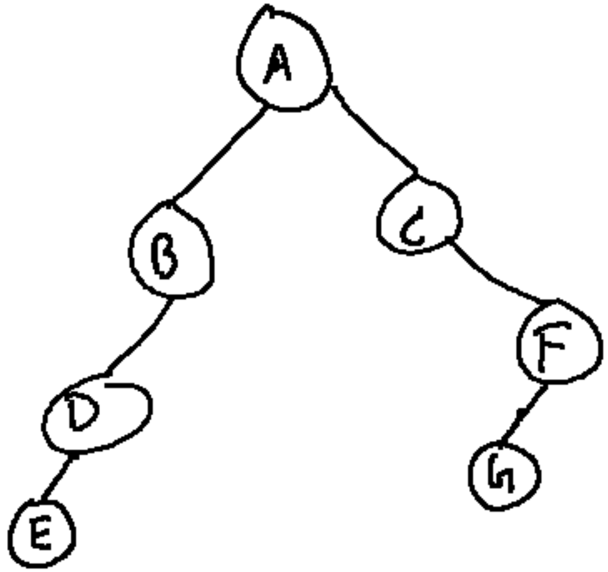
C

D

F

E

G



Analysis of Traversal Techniques

- For inorder, preorder, and postorder traversal, let the tree have n nodes of which n_1 are in the left subtree and the rest in the right subtree.
- Recurrence relation:

$$T(n) = T(n_1) + T(n-n_1-1) + O(1)$$

- Can solve by guessing that $T(n) \leq cn$ for constant c.
- Verify.

$T(n) \leq cn_1 + c(n-n_1-1) + O(1) \leq cn$, provided c is large enough.

Analysis – Depth Order Traversal

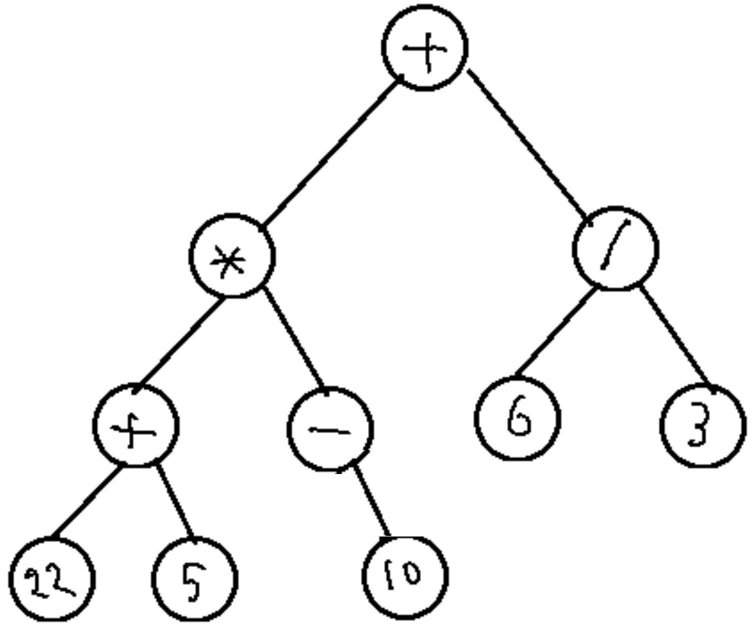
- How to analyze this traversal?
- Assume that the tree has n nodes.
- Each node is placed in the queue exactly once.
- The rest of the operations are all O(1) for every node.
- So the total time is O(n).
- This traversal can be seen as forming the basis for a graph traversal.

Application to Expression Evaluation

- We know what expression evaluation is.
- We deal with binary operators.
- An expression tree for a expression with only unary or binary operators is a binary tree where the leaf nodes are the operands and the internal nodes are the operators.

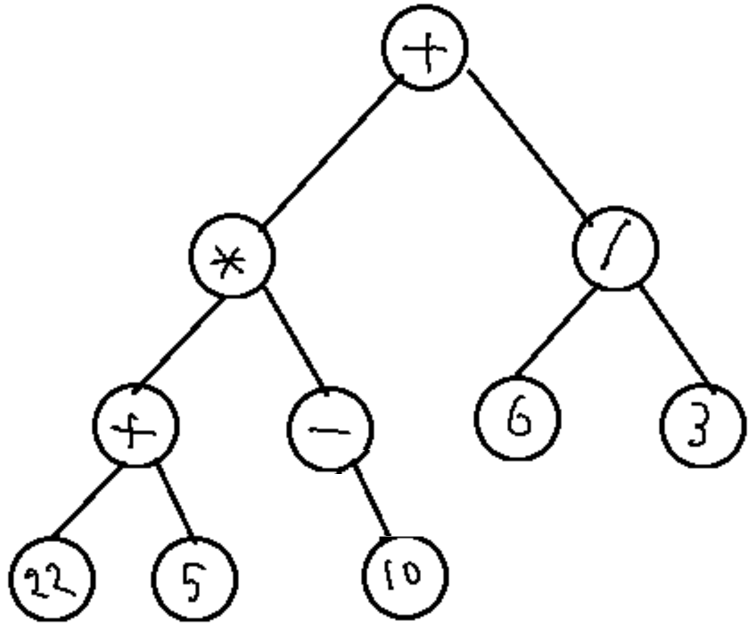
Example Expression Tree

- See the example to the right.
- The operands are 22, 5, 10, 6, and 3.
- These are also leaf nodes.



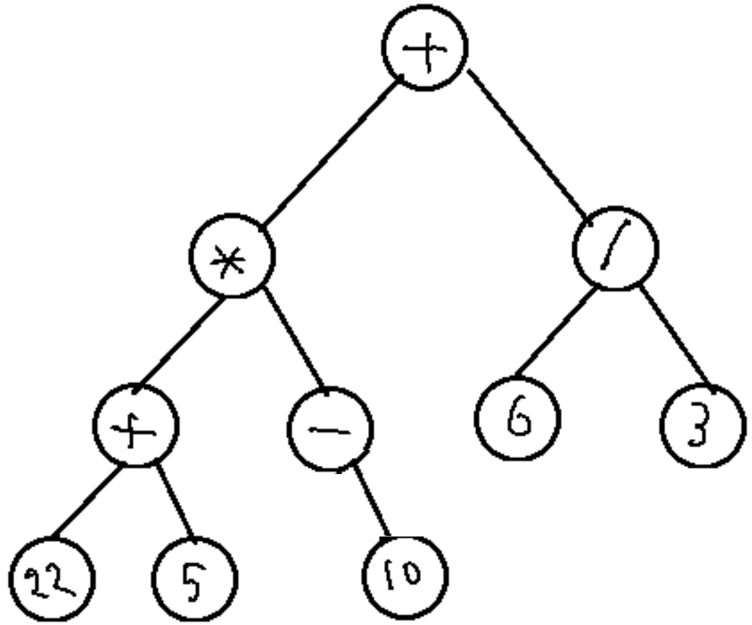
Questions wrt Expression Tree

- How to evaluate an expression tree?
  - Meaning, how to apply the operators to the right operands.
- How to build an expression tree?
  - Given an expression, how to build an equival expression tree?



Questions wrt Expression Tree

- Notice that an inorder traversal of the expression tree gives an expression in the infix notation.
  - The above tree is equivalent to the expression $((22 + 5) \times (-10)) + (6/3)$
- What does a postorder and preorder traversal of the tree give?
  - Answer: ??



Why Expression Trees?

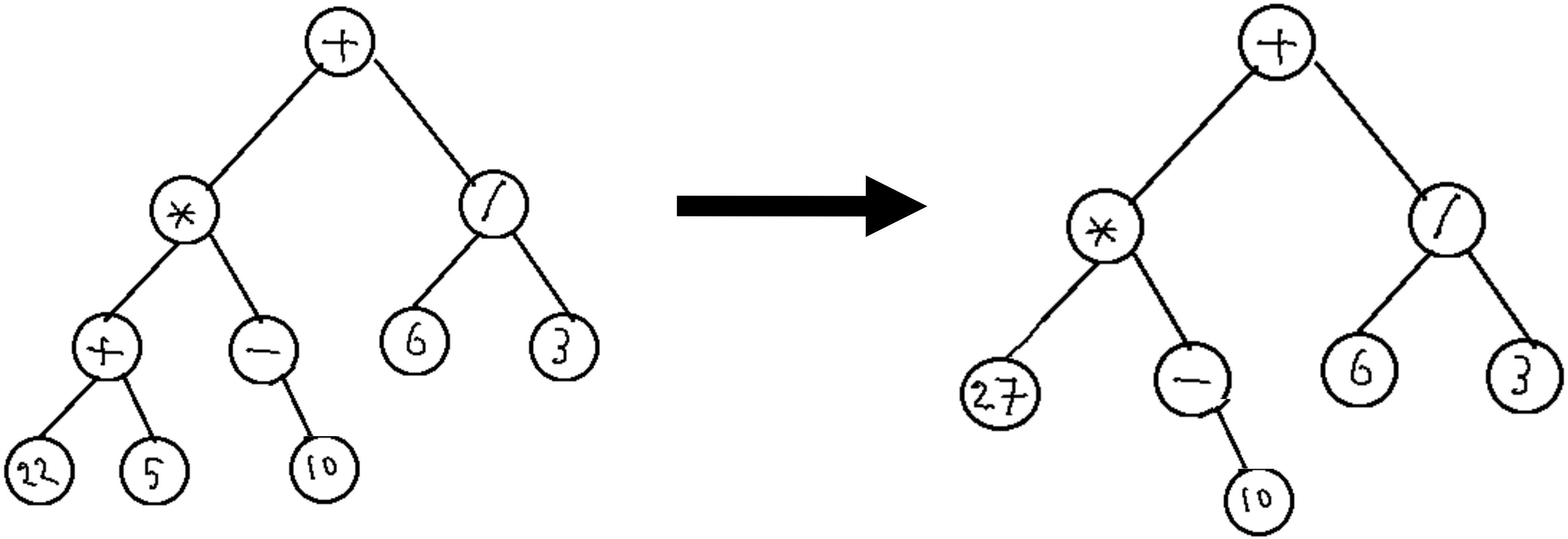
- Useful in several settings such as
  - compliers
  - can verify if the expression is well formed.

How to Evaluate using an Expression Tree

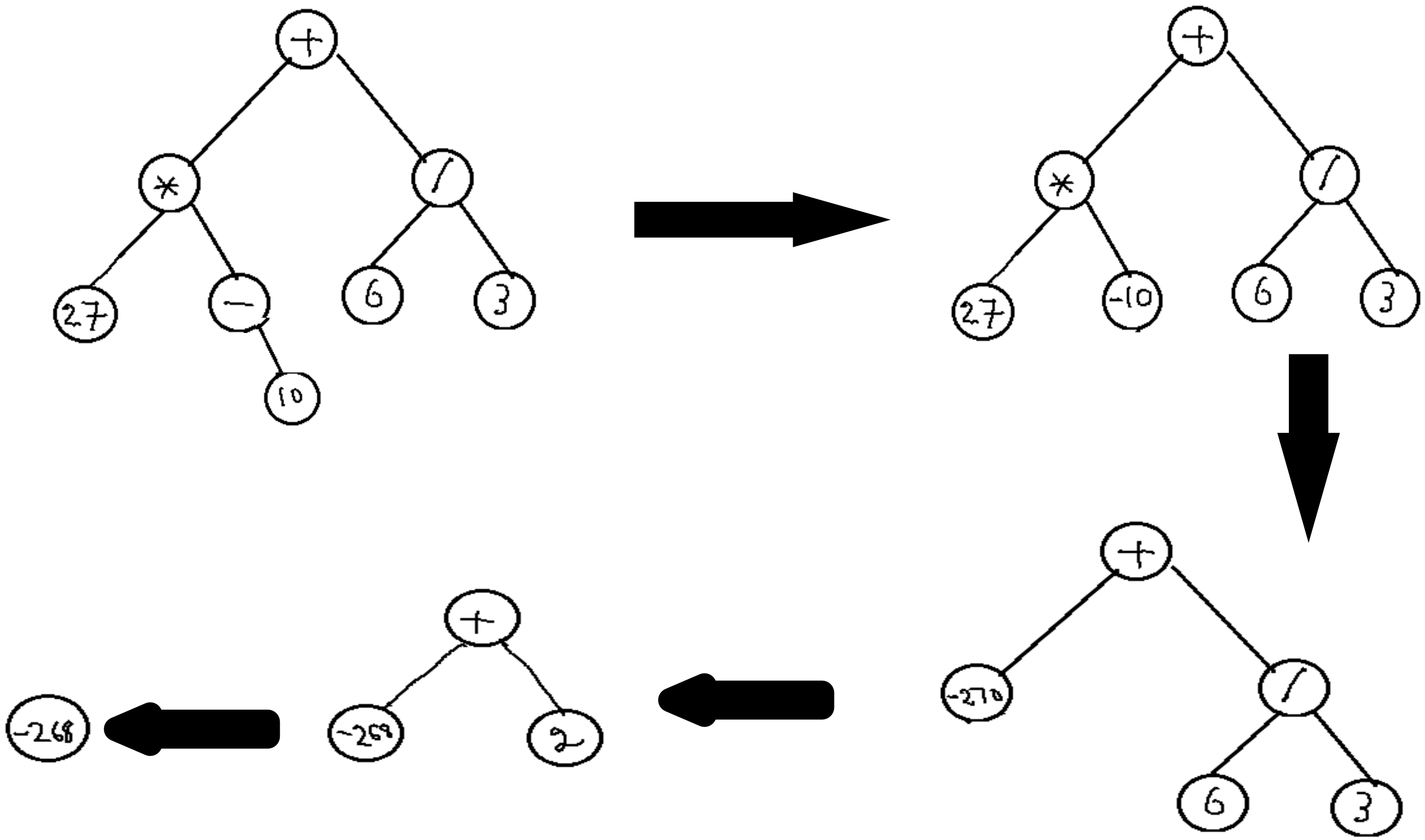
- Essentially, have to evaluate the root.
- Notice that to evaluate a node, its left subtree and its right subtree need to be operands.
- For this, may have to evaluate these subtrees first, if they are not operands.
- So, Evaluate(root) should be equivalent to:
  - Evaluate the left subtree
  - Evaluate the right subtree
  - Apply the operator at the root to the operands.

How to Evaluate using an Expression Tree

- This suggests a recursive procedure that has the above three steps.
- Recursion stops at a node if it is already an operand.



How to Evaluate using an Expression Tree



Pending Question

- How to build an expression tree?
- Start with an expression in the infix notation.
- Recall how we converted an infix expression to a postfix expression.
- The idea is that operators have to wait to be sent to the output.
  - A similar approach works now.

Building an Expression Tree

- Let us start with a postfix expression.
- The question is how to link up operands as (sub)trees.
- As in the case of evaluating a postfix expression, have to remember operators seen so far.
  - need to see the correct operands.
- A stack helps again.
- But instead of evaluating subexpression, we have to grow them as trees.
  - Details follow.

Building an Expression Tree

- When we see an operand :
  - That could be a leaf node...Or a tree with no children.
  - What is its parent?
  - Some operator.
  - In our case, operands can be trees also.
- The above observations suggest that operands should wait on the stack.
  - Wait as trees.

Building an Expression Tree

- What about operators?
- Recall that in the postfix notation, the operands for an operator are available in the immediate preceding positions.
- Similar rules apply here too.
- So, pop two operands (trees) from the stack.
- Need not evaluate, but create a bigger (sub)tree.

Building an Expression Tree

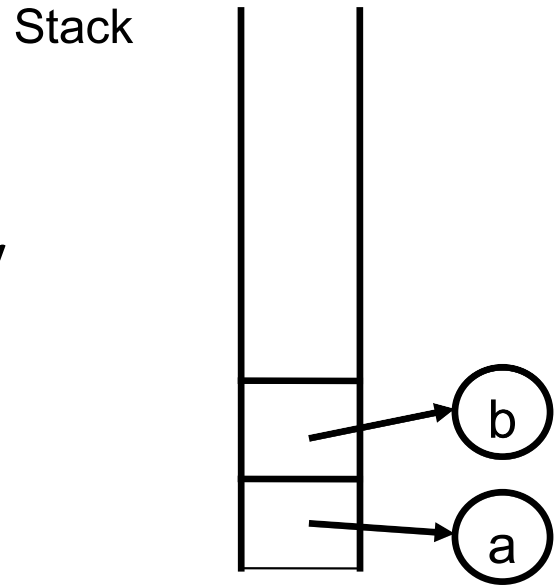
```
Procedure ExpressionTree(E)
//E is an expression in postfix notation.
begin
    for i=1 to |E| do
        if E[i] is an operand then
            create a tree with the operand as the only node;
            add it to the stack
        else if E[i] is an operator then
            pop two trees from the stack
            create a new tree with E[i] as the root and the two trees popped as its children;
            push the tree to the stack
    end-for
end
```

Building an Expression Tree

- Consider the expression $(a + b - f) / (c \times d + e)$
- The postfix of the expression is $a\ b + f - c\ d \times e + /$
- Let us follow the above algorithm.

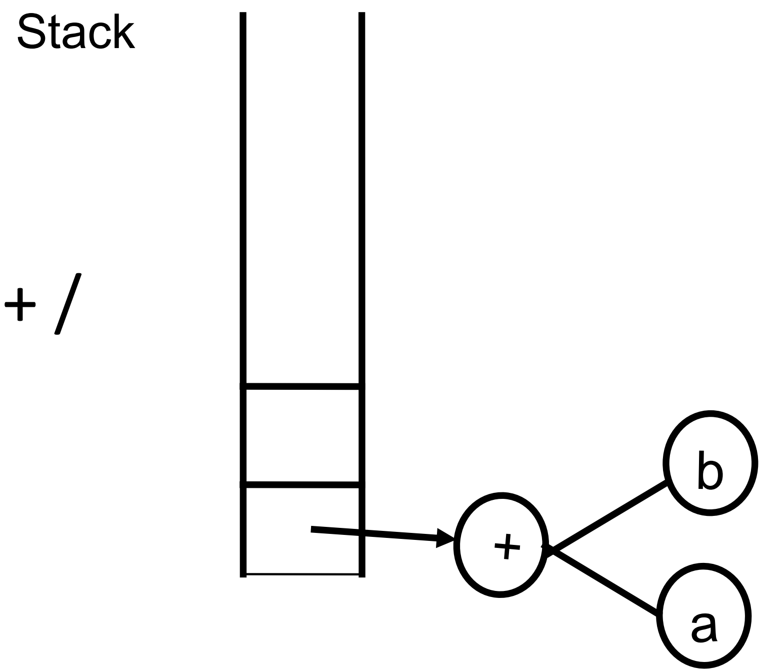
Building an Expression Tree

+ f − c d × e + /



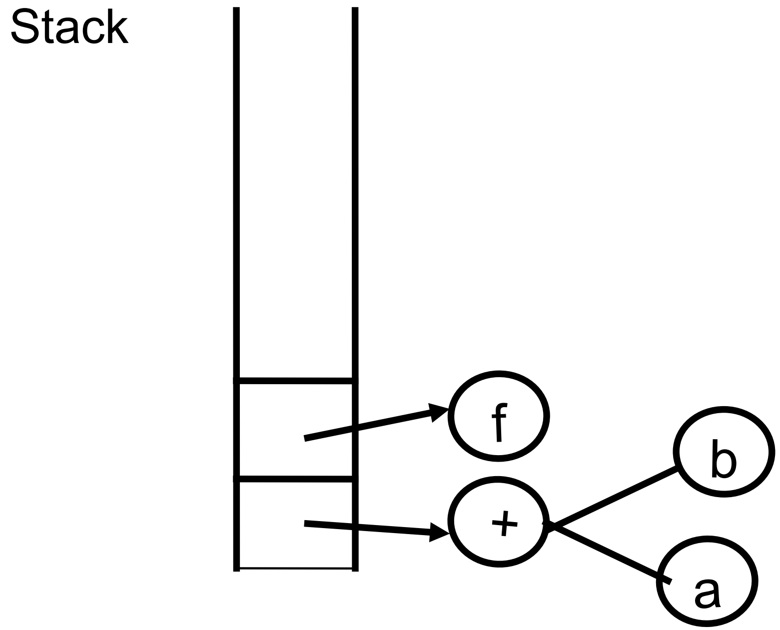
Building an Expression Tree

$+ \; f \; - \; c \; d \; \times \; e \; + \; /$



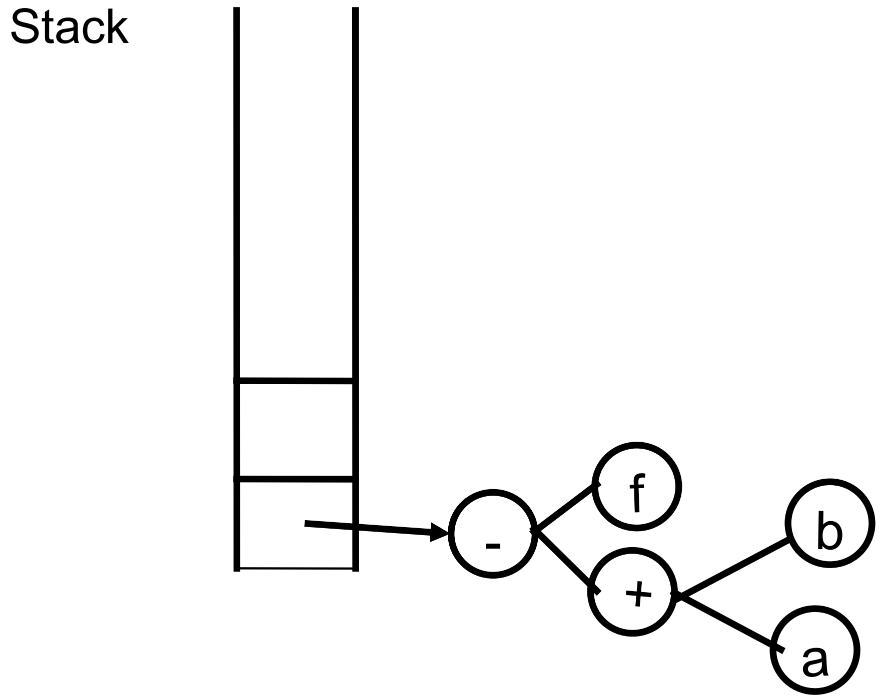
Building an Expression Tree

$- \; c \; d \; \times \; e \; + \; /$



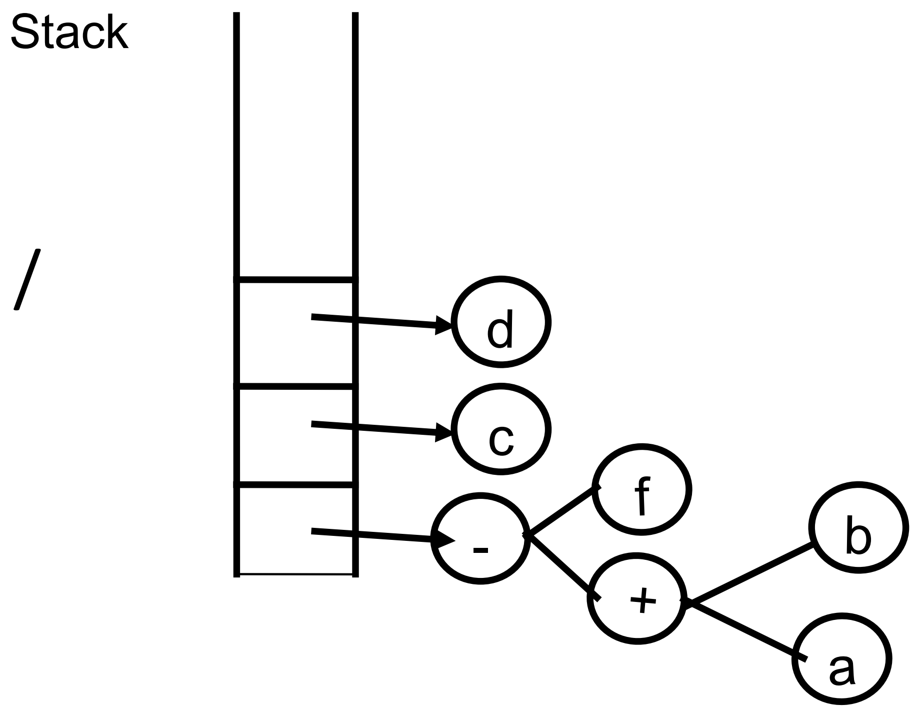
Building an Expression Tree

c d × e + /



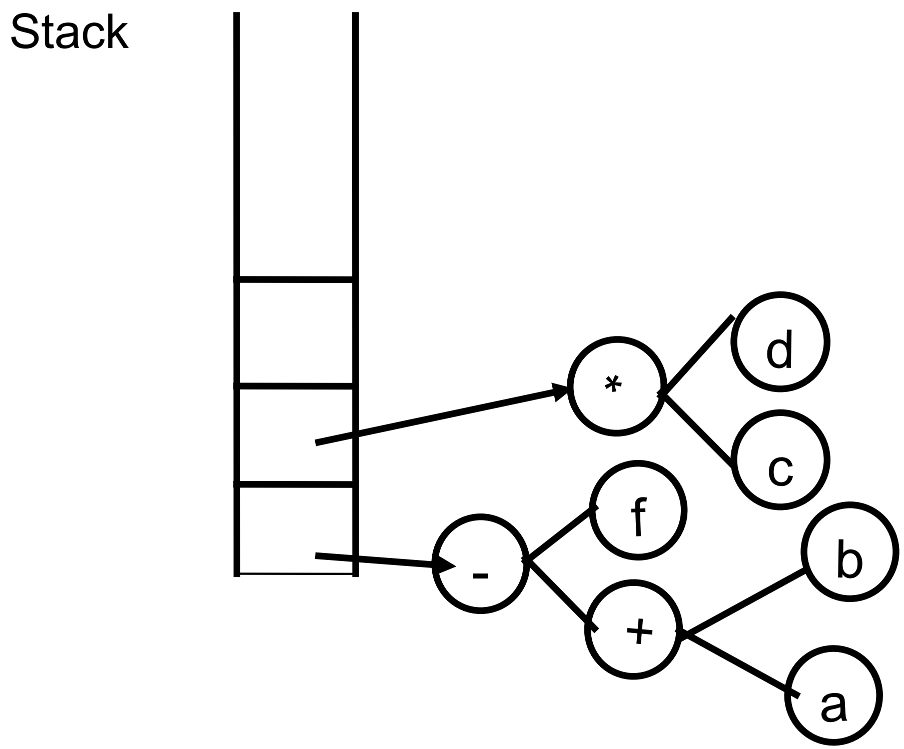
Building an Expression Tree

× e + /

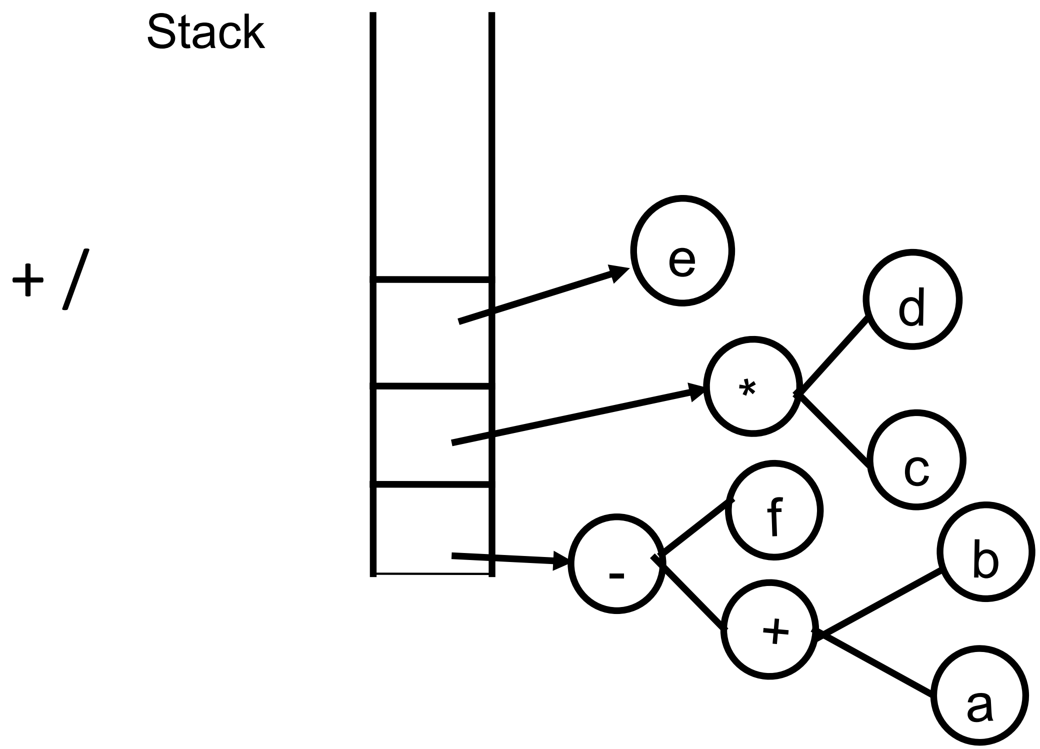


Building an Expression Tree

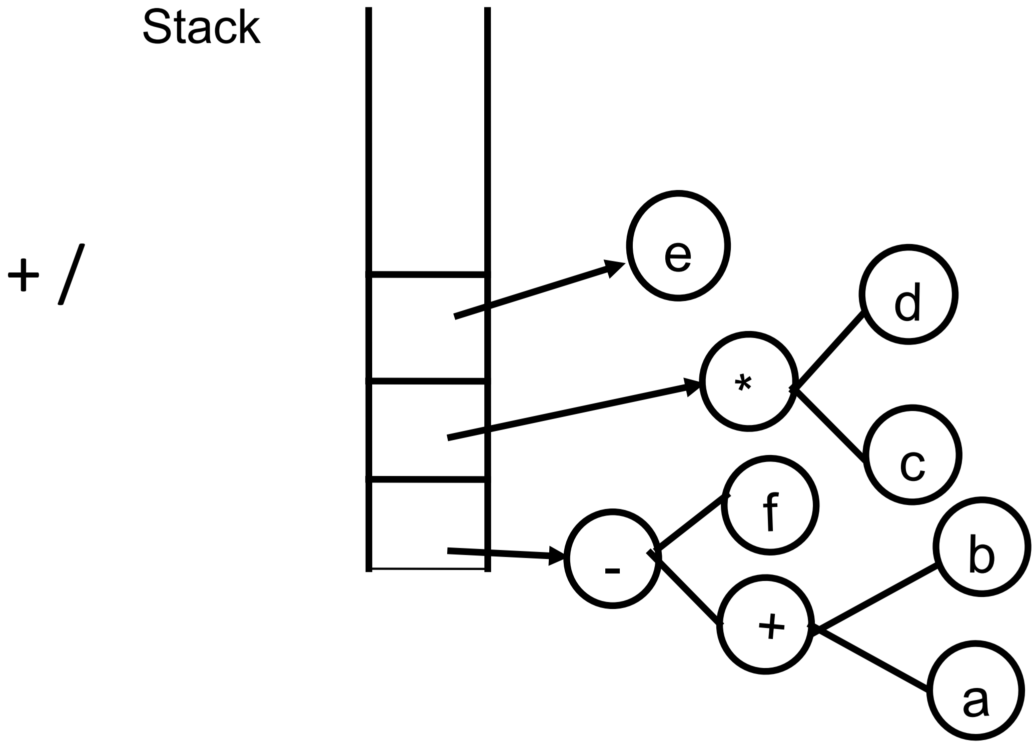
e + /



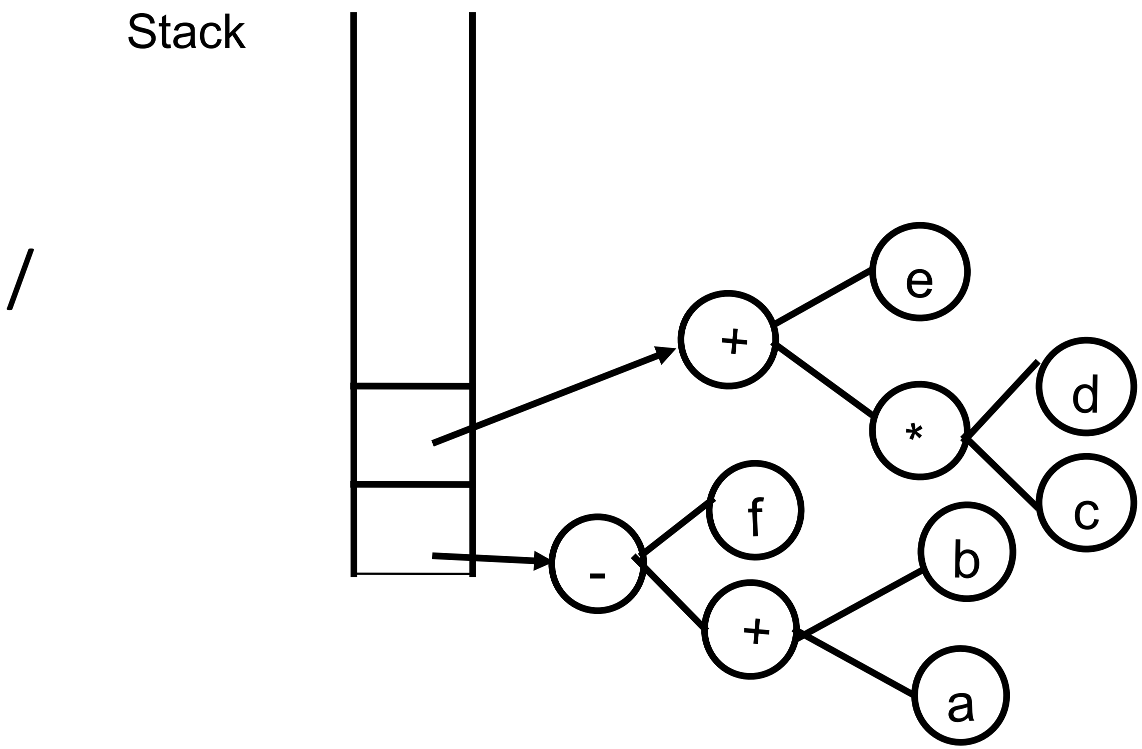
Building an Expression Tree



Building an Expression Tree



Building an Expression Tree



Building an Expression Tree