National University of Computer & Emerging Sciences

Trees

BINARY TREES

Binary Trees

- A commonly used type of tree is a binary tree
- Each node has *at most* two (bi) children

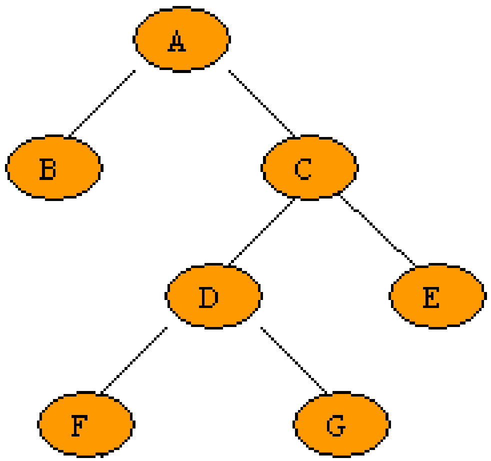
- The "tree" is a conceptual structure
- The data can be stored either in
  - a <u>dynamic linked tree structure</u>, or
  - <u>in contiguous memory cells (array) according to a *set pattern*</u>;

- In other words, implementation can be pointer-based or array-based

Binary Trees -Types

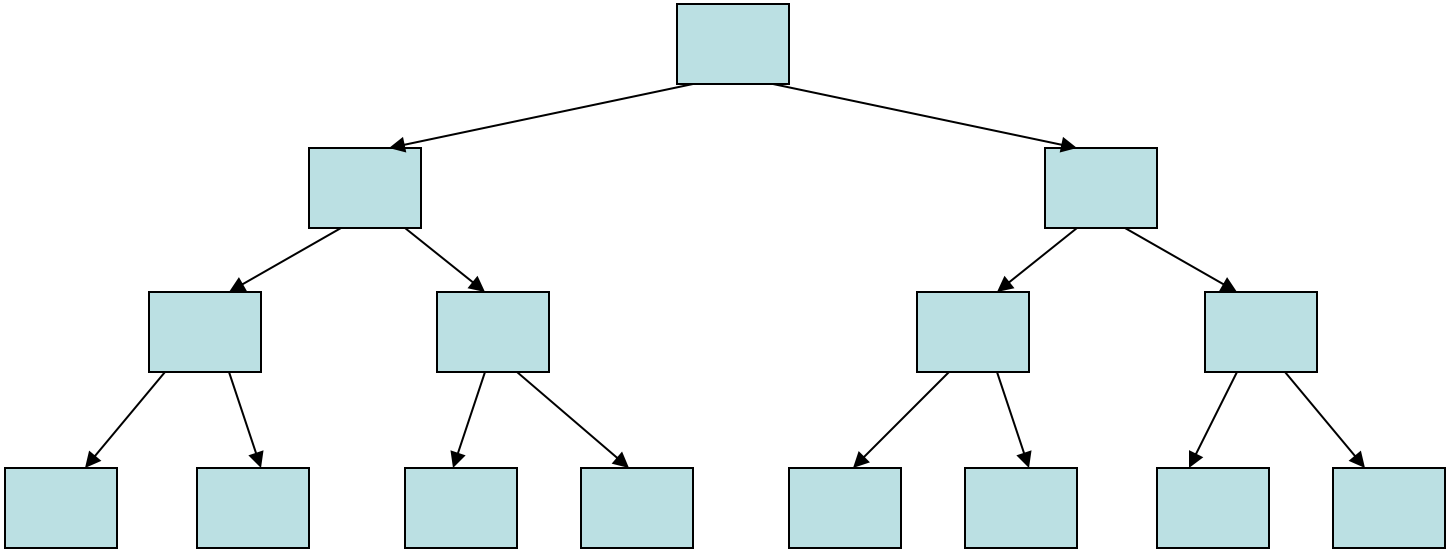
- A **rooted binary tree** is a tree with a root node in which every node has at most two children. There are also un-rooted/free trees (known as graphs)!

- A **full binary tree** (sometimes **proper binary tree** or **2-tree** or **strictly binary tree**) is a tree in which every node other than the leaves has two children. Or, perhaps more clearly, every node in a binary tree has exactly (strictly) 0 or 2 children.



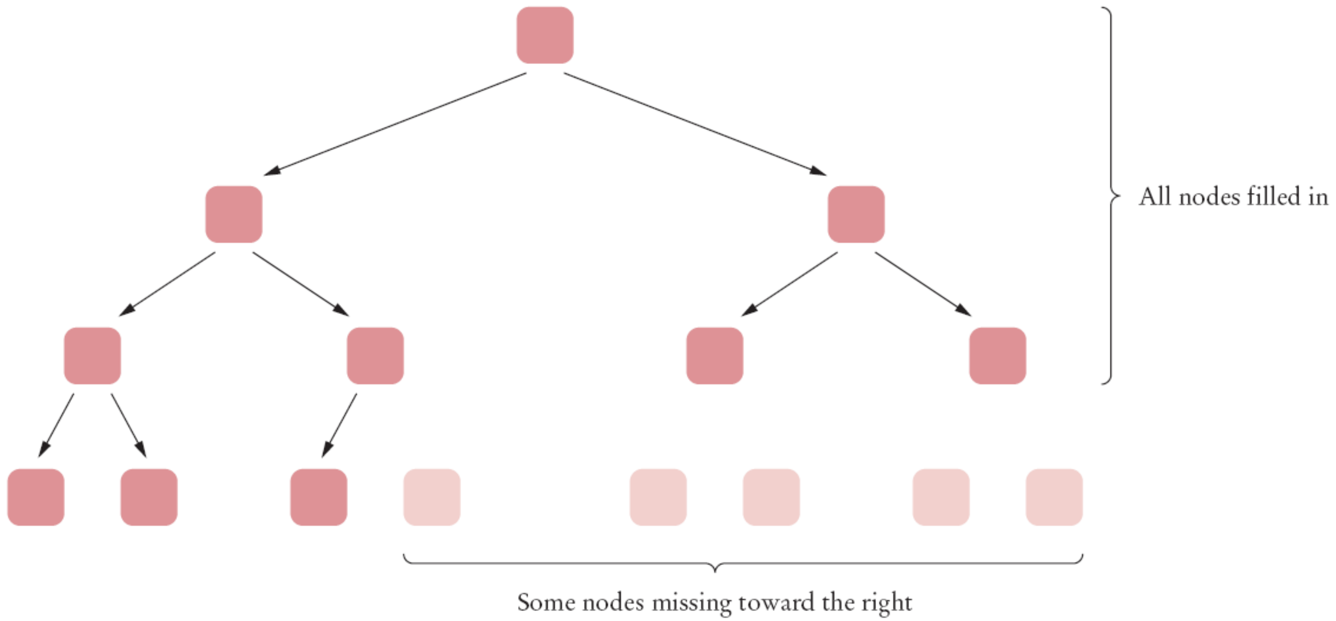
Binary Trees -Types

- A **complete (or perfect) binary tree** is a binary tree in which every level is completely filled.
  - Example?
  - A family tree?



Binary Trees -Types

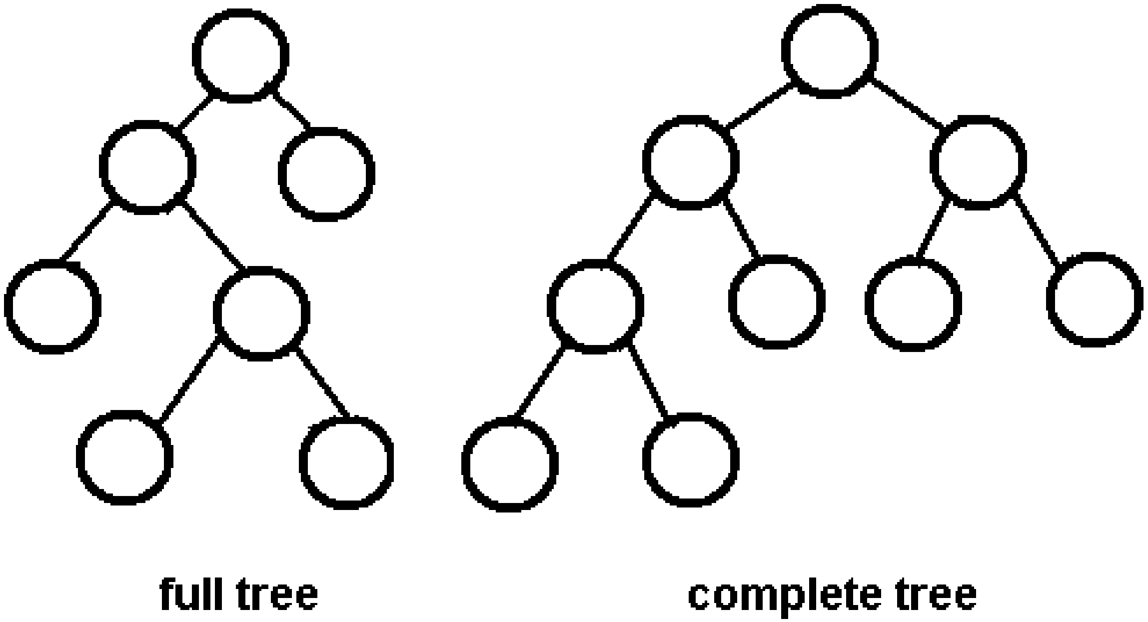
- A tree is called an **almost complete binary tree** or nearly complete binary tree if the last level is not completely filled and all nodes are as far left as possible.



An Almost Complete Tree

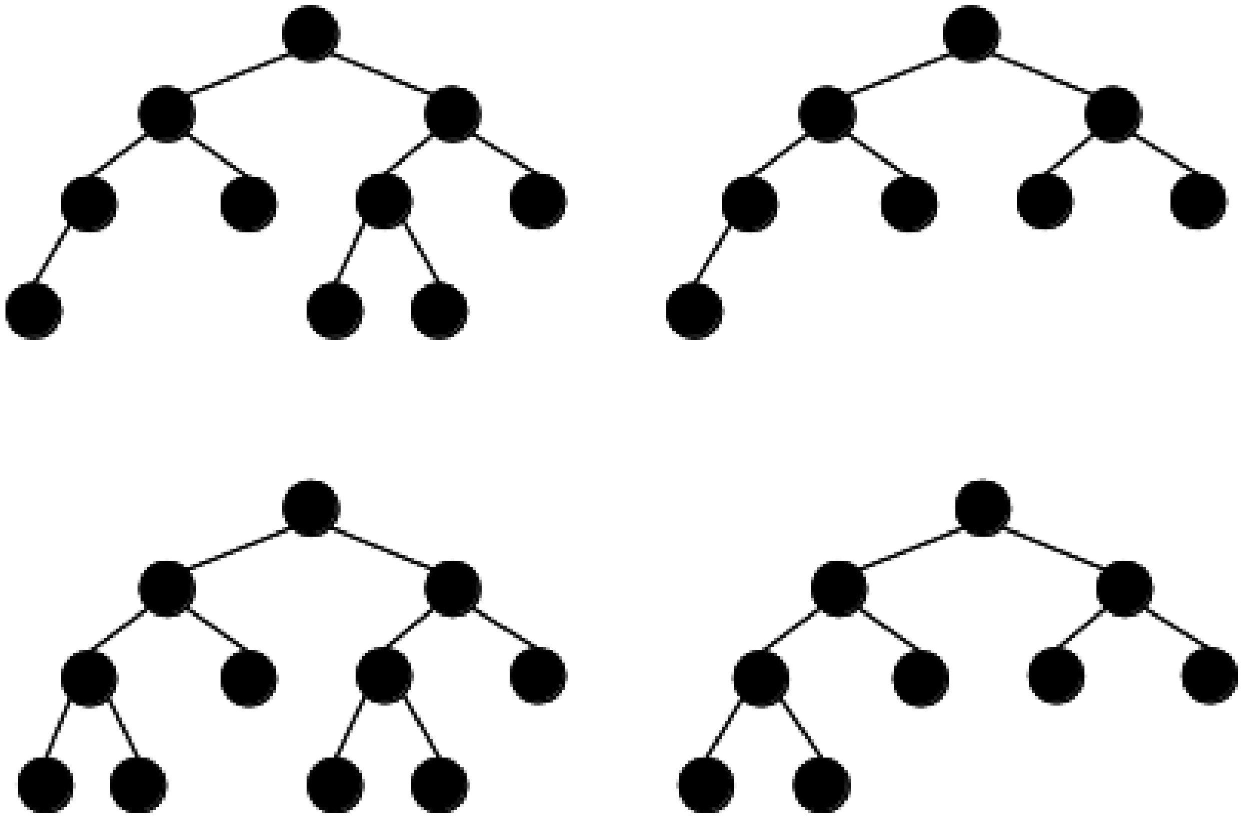
Binary Trees -Types

- What is the difference between full and complete binary tree?



Binary Trees -Types

- What is the difference between full and complete binary tree?

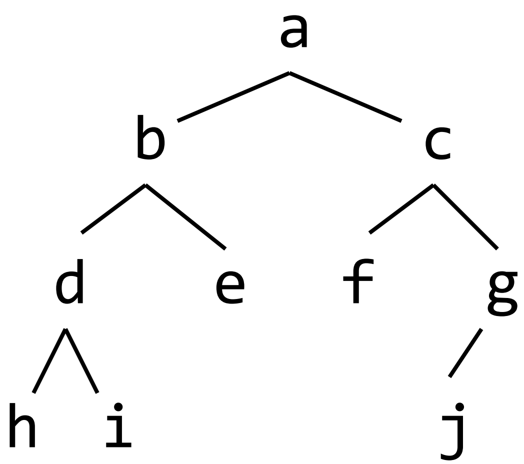


Binary Trees -Types

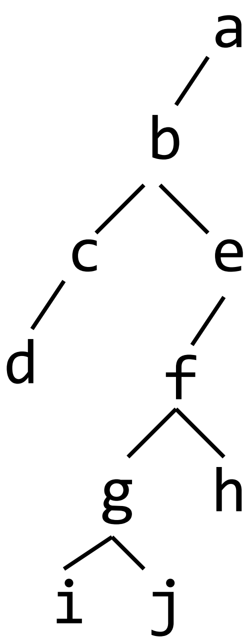
- **Balanced binary tree**: for each node, the difference in height of the right and left sub-trees is no more than one

- **Completely balanced tree**: left and right sub-trees of every node have the same height

Balanced Binary Tree



A balanced binary tree



An unbalanced binary tree

- A binary tree is balanced if every level above the lowest is "complete" (contains 2^n nodes)

Properties of binary trees

- A *complete binary tree* of level *d* is the strictly binary tree *(full)* all of whose leaves are at level *d*
- A complete binary tree of level *d* has 2^l nodes at each level *l* where $0<=l<=d$
- Total number of nodes (*tn*) in a complete binary tree of depth d will be:

$$tn = 2^0 + 2^1 + 2^2 + ---+ 2^d = \sum_{j=0}^{d} 2^j = 2^{d+1} - 1$$

Properties of binary trees

- The number n of nodes 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 height (or depth) of the tree.

Properties of binary trees

- The number L of leaf nodes in a perfect (or complete) binary tree can be found using this formula: $L = 2^h$ where h is the depth of the tree

- The number n of nodes in a perfect (or complete) binary tree can also be found using this formula: $n = 2L - 1$ where L is the number of leaf nodes in the tree.

Questions?

"He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever"
Chinese Proverb

"The wise man doesn't give the right answers, he poses the right questions."
Claude Levi-Strauss

"A wise man can learn more from a foolish question than a fool can learn from a wise answer."
Bruce Lee

Tree ADT

- Objects: any type of objects can be stored in a tree
- accessor methods
  - root() – return the root of the tree
  - parent(p) – return the parent of a node
  - children(p) – returns the children of a node
- query methods
  - size() – returns the number of nodes in the tree
  - isEmpty() - returns true if the tree is empty
  - elements() – returns all elements
  - isRoot(p)
- other methods
  - Tree traversal, Node addition/deletion, create/destroy

Binary Trees Storage

- Array based implementation
- Linked List based implementation

Binary Tree: contiguous storage

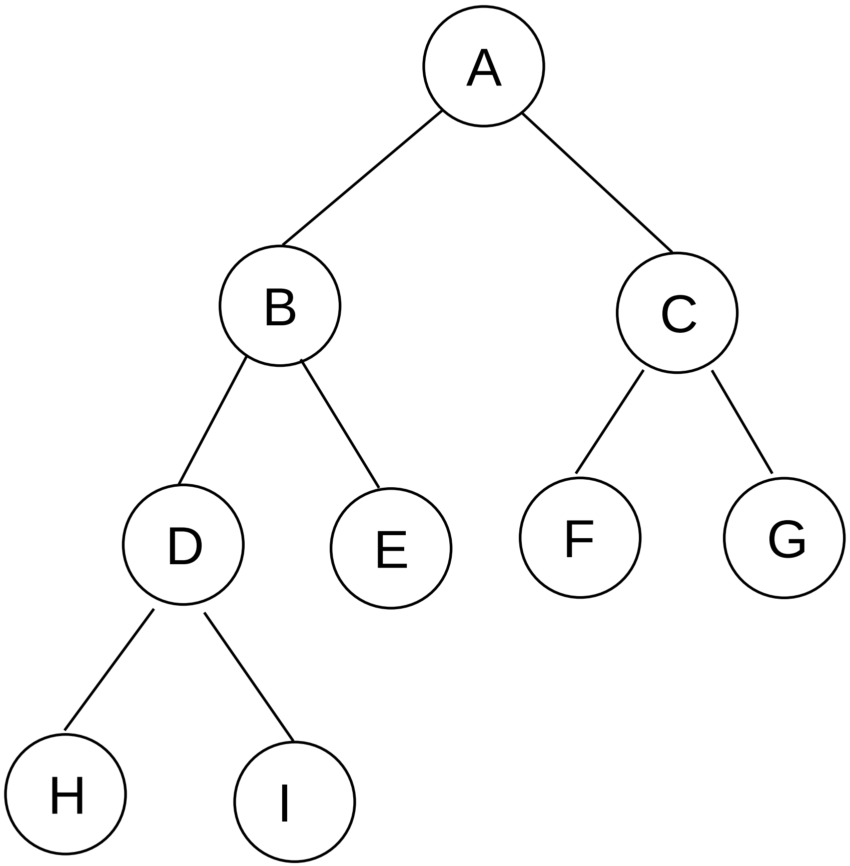
- Value in root node stored first, followed by left child, then right child
- Each successive level in the tree stored left to right; unused nodes in tree represented by a bit pattern to indicate nothing stored there

- Children of any given node *n* is stored in cells $2n$ and $2n + 1$ (If array index starts at 1)
  - What if it starts at 0?

- Storage allocated for full tree, even if many nodes empty
- What size array is required for a tree of depth *h?*

Trees

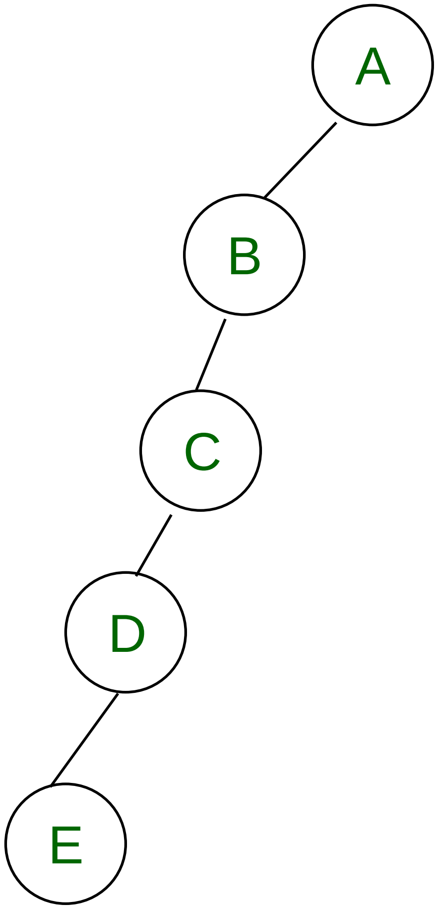
A complete (almost) binary tree in array



[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Trees

A binary tree (incomplete) in array



[1]	A
[2]	B
[3]	--
[4]	C
[5]	--
[6]	--
[7]	--
[8]	D
[9]	--
.	.
[16]	E

Binary Tree: as a linked structure

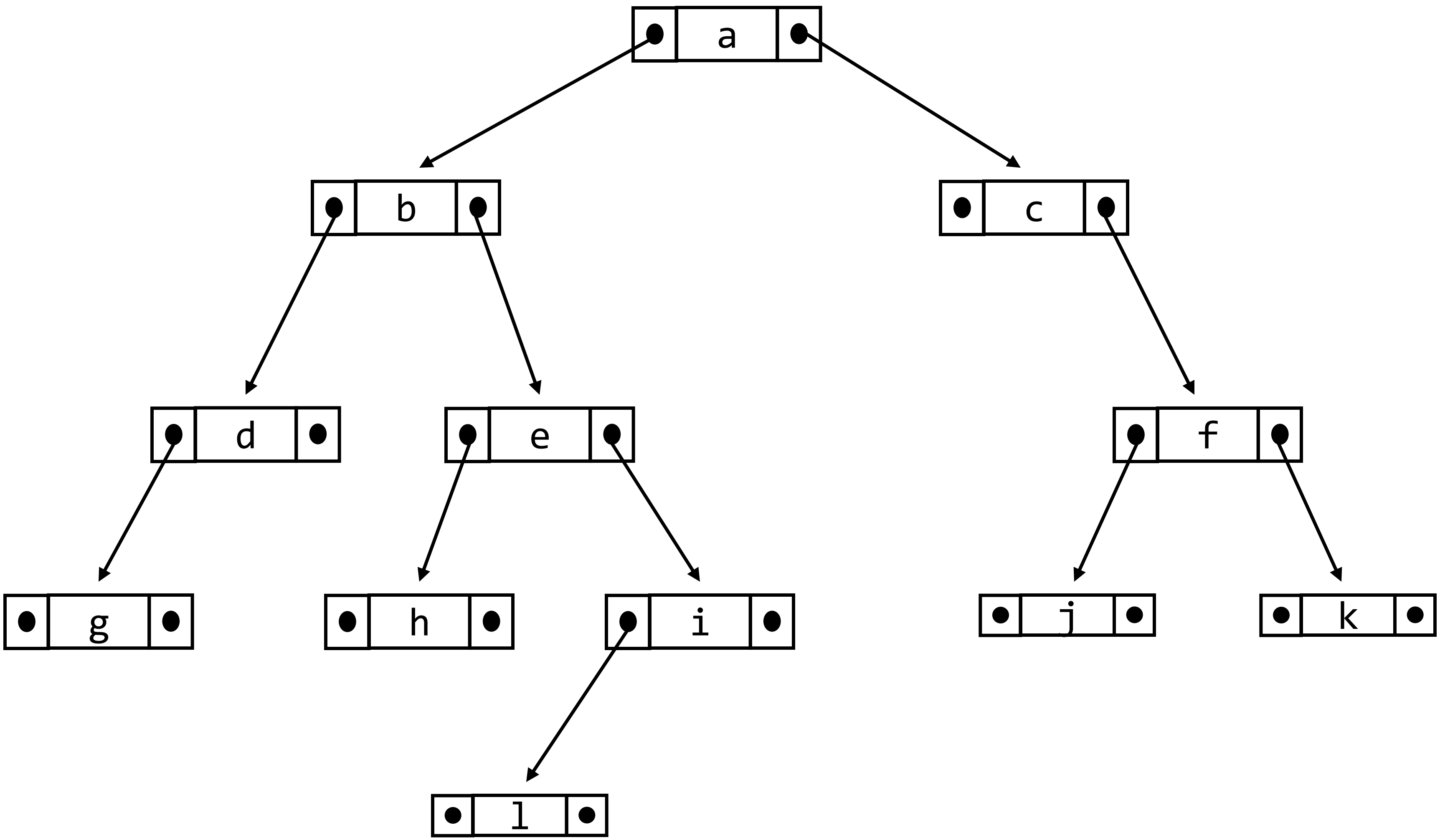
- Each node in the tree consists of:
  - The data, or value contained in the element
  - A left child pointer (pointer to first child)
  - A right child pointer (pointer to second child)

Binary Tree: as a linked structure

- A root pointer points to the root node
  - Follow pointers to find every other element in the tree
- Add and remove nodes by manipulating pointers
- Leaf nodes have child pointers set to null

```
class CBinTree
{
    struct Node
    {
        int value;
        Node *LeftChild,*RightChild;
    }*Root;
    /***Operations*********/
    /***********************/
};
```

Trees



Questions?

"He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever"
Chinese Proverb

"The wise man doesn't give the right answers, he poses the right questions."
Claude Levi-Strauss

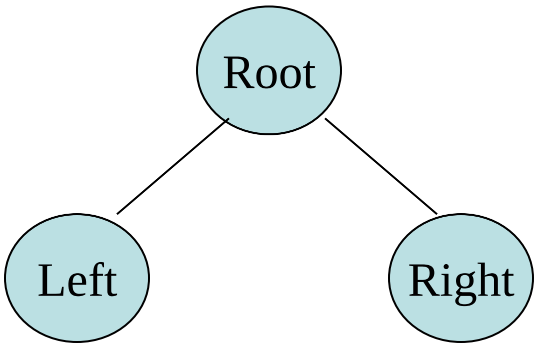
"A wise man can learn more from a foolish question than a fool can learn from a wise answer."
Bruce Lee

Traversal of Binary Trees

- Pass through all nodes of tree
  - Inorder (symmetric traversal)
  - Preorder
  - Postorder

Trees Traversal

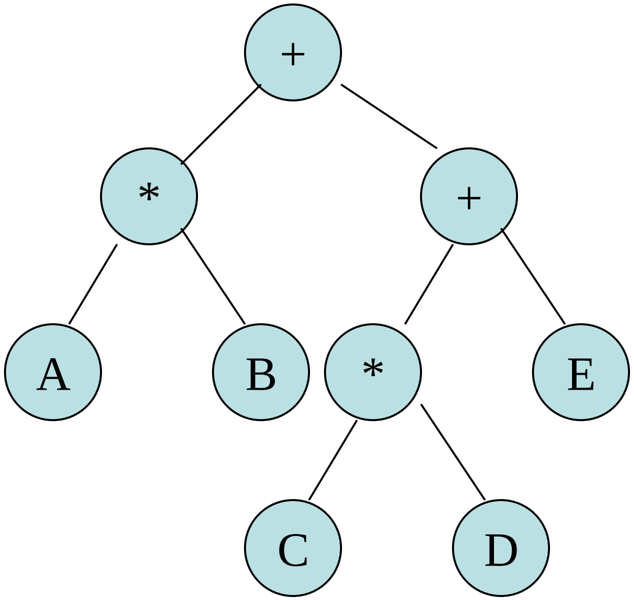
- Inorder
  - (Left) Root (Right)
- Preorder
  - Root (Left) (Right)
- Postorder
  - (Left) (Right) Root



Inorder Traversal

Left Root Right manner

- Left + Right
- [Left*Right]+[Left+Right]
- (A*B)+[(Left*Right)+E)
- (A*B)+[(C*D)+E]

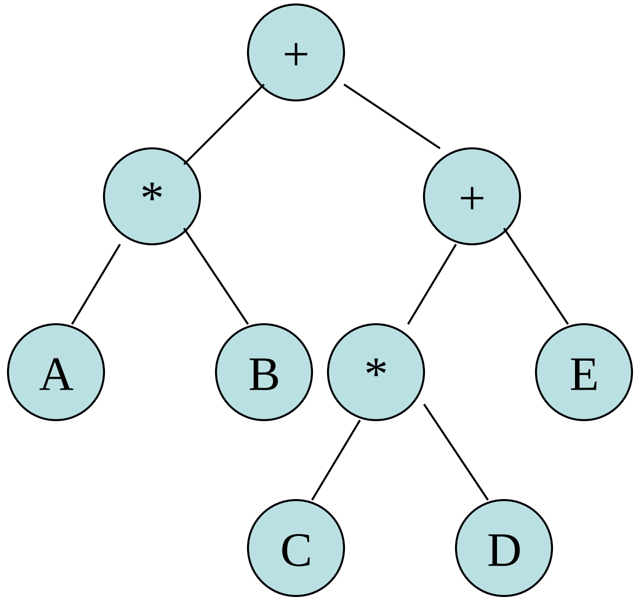


(A*B)+(C*D+E)

Preorder Traversal

Root Left Right manner

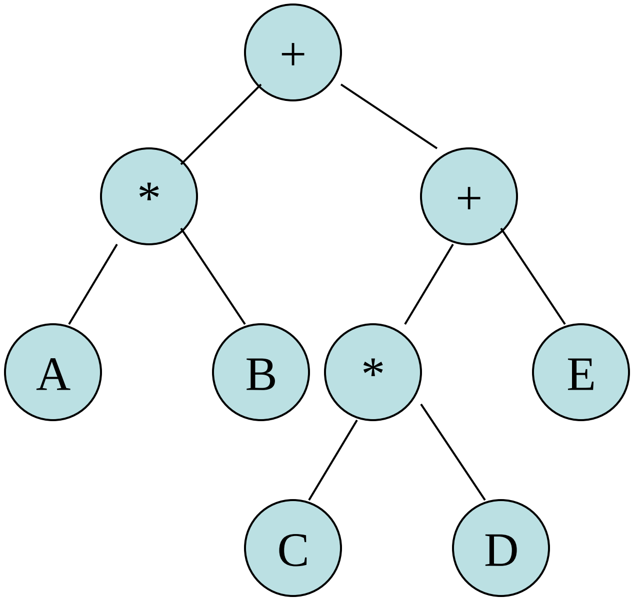
- + Left Right
- + [*Left Right] [+Left Right]
- +(*AB) [+ *Left Right E]
- +*AB + *C D E



Postorder Traversal

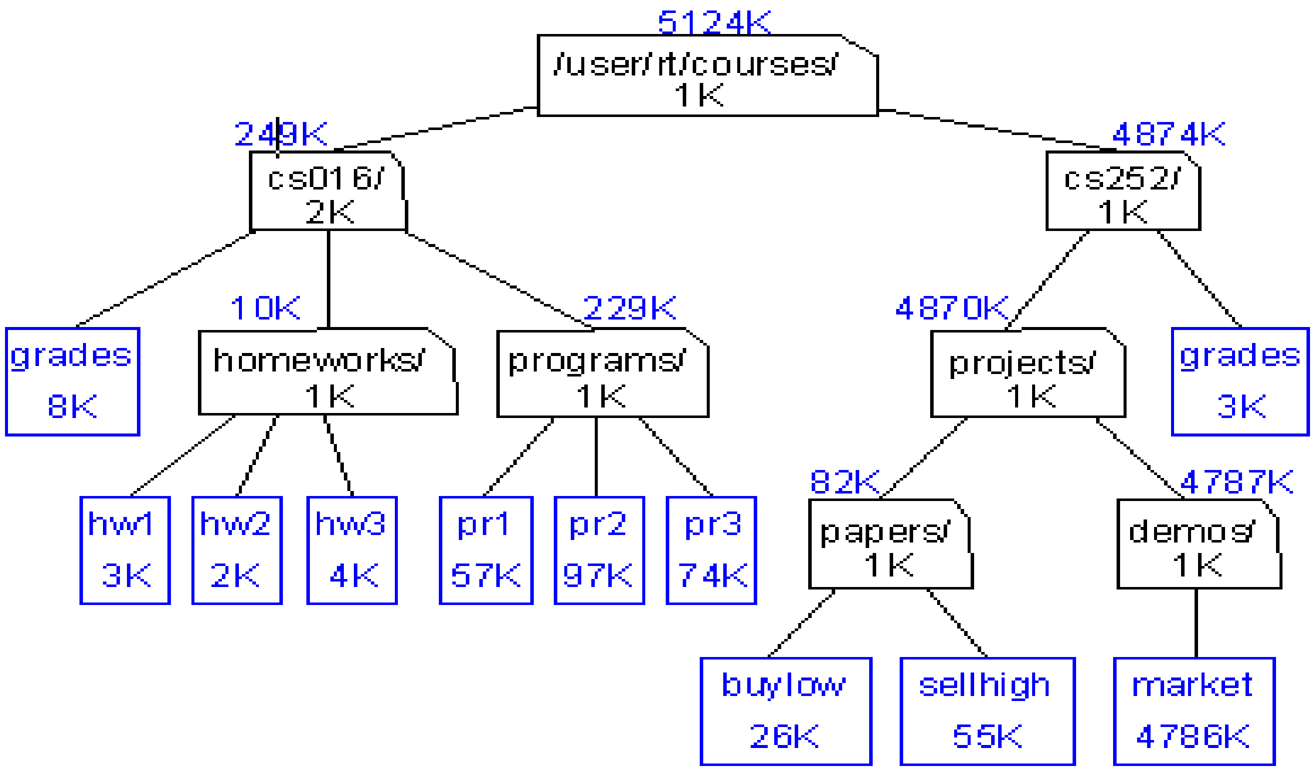
Left Right Root manner

- Left Right +
- [Left Right *] [Left Right+] +
- (AB*) [Left Right * E +]+
- (AB*) [C D * E +]+
- AB* C D * E + +



Example

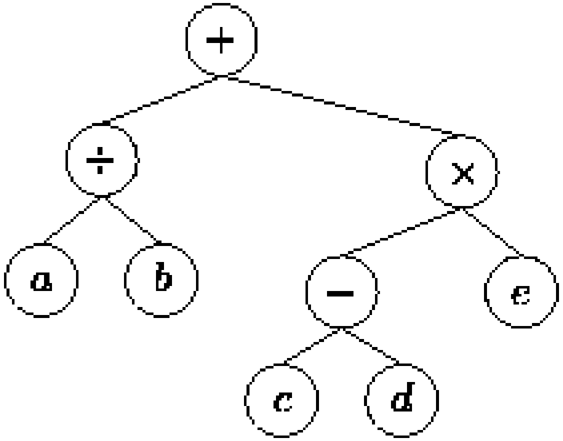
- du (**disk usage) command in Unix**
  - **which traversal should be used?**



Expression tress

- What is an algebraic expression?
- Each algebraic expressions has an inherent tree-like structure

- Example: a/b + (c-d) * e can be represented as

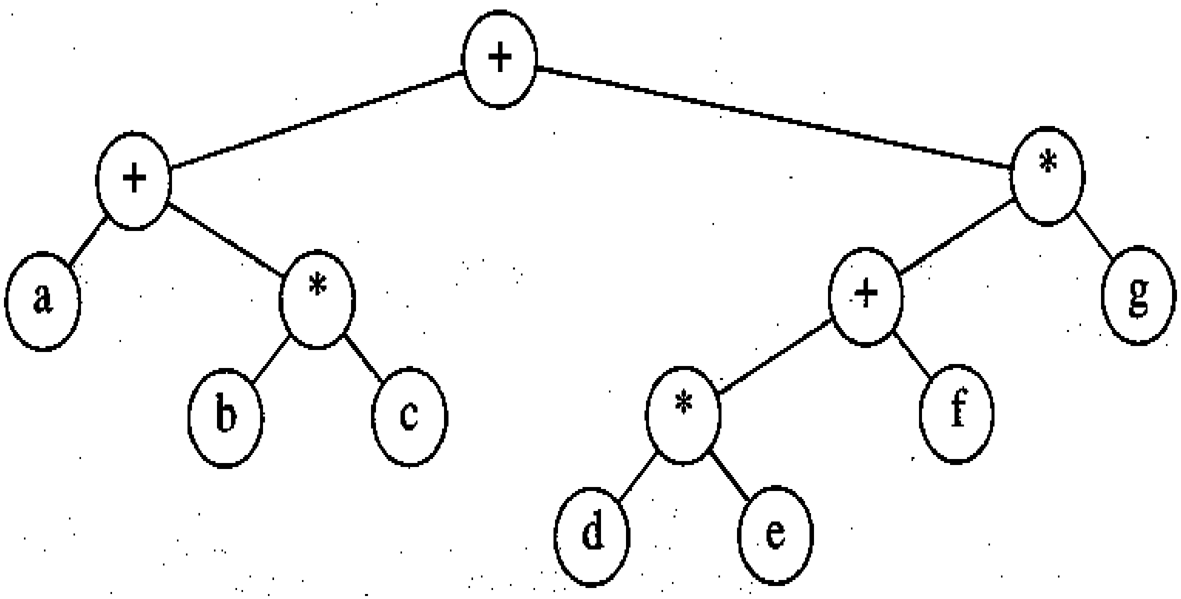


Expression Trees

- The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (*a*, *b*, *c*, *d*, and *e*).
- The non-terminal nodes of an expression tree are the operators (+, -,*, and /).
- No parentheses but the tree representation has captured the intent of the parentheses since the subtraction is lower in the tree than the multiplication.

Expression Trees

(a+b*c) +((d*e+f)*g)

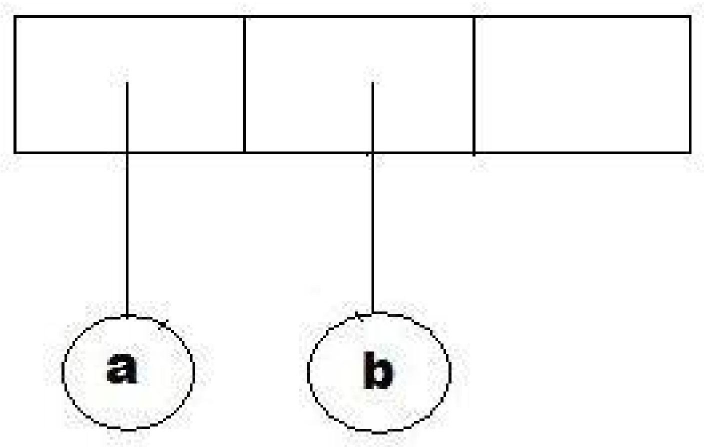


Algorithm: convert postfix into expression tree

- Read one symbol at a time from the expression.
  - If the symbol is an operand, one-node tree is created and a pointer is pushed onto a Stack
  - If the symbol is an operator, the pointers are popped to two trees *T1* and *T2* from the stack and a new tree whose root is the operator and whose left and right children point to *T2* and *T1* respectively is formed . A pointer to this new tree is then pushed to the Stack.

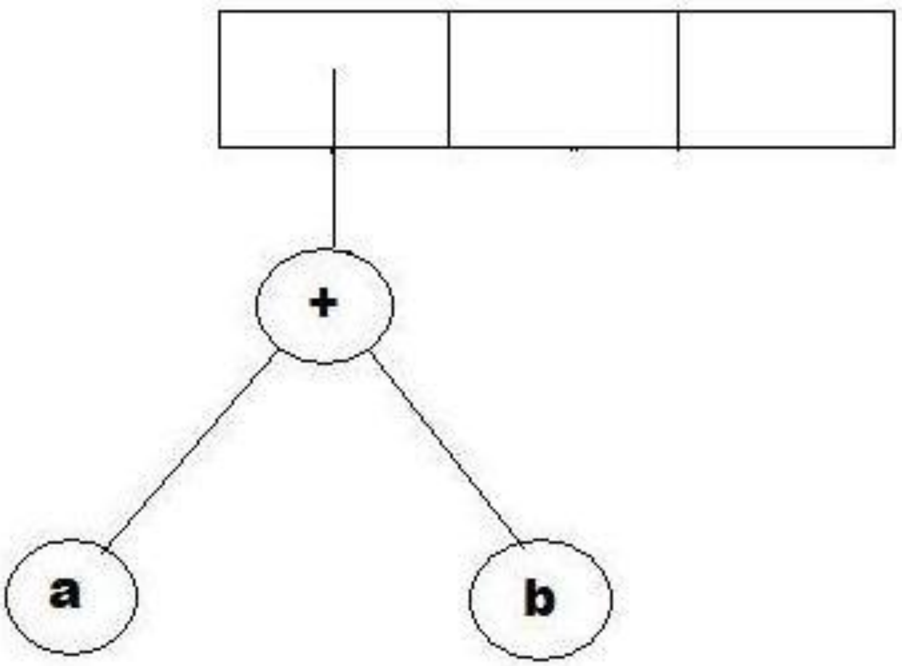
Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



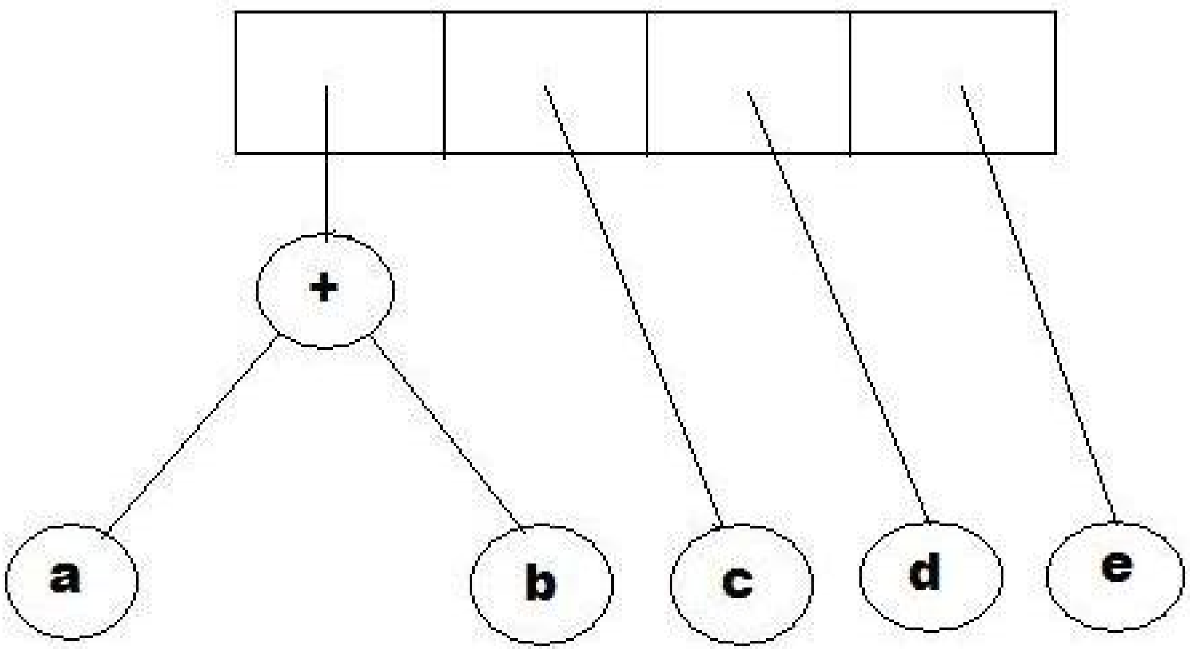
Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



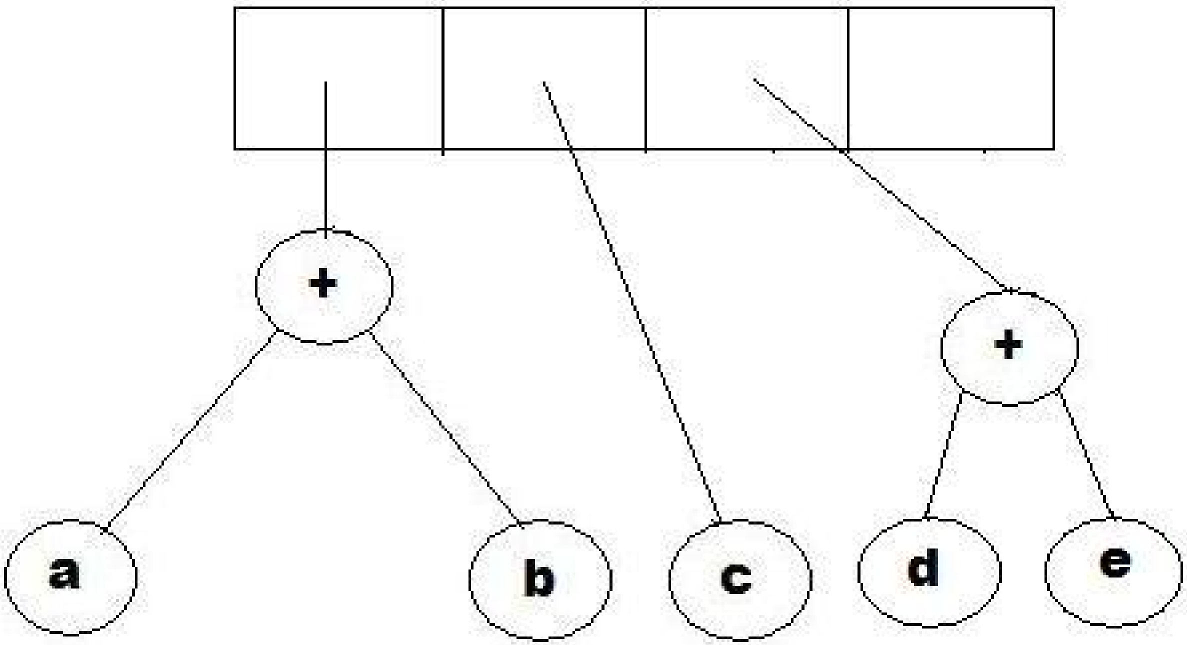
Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



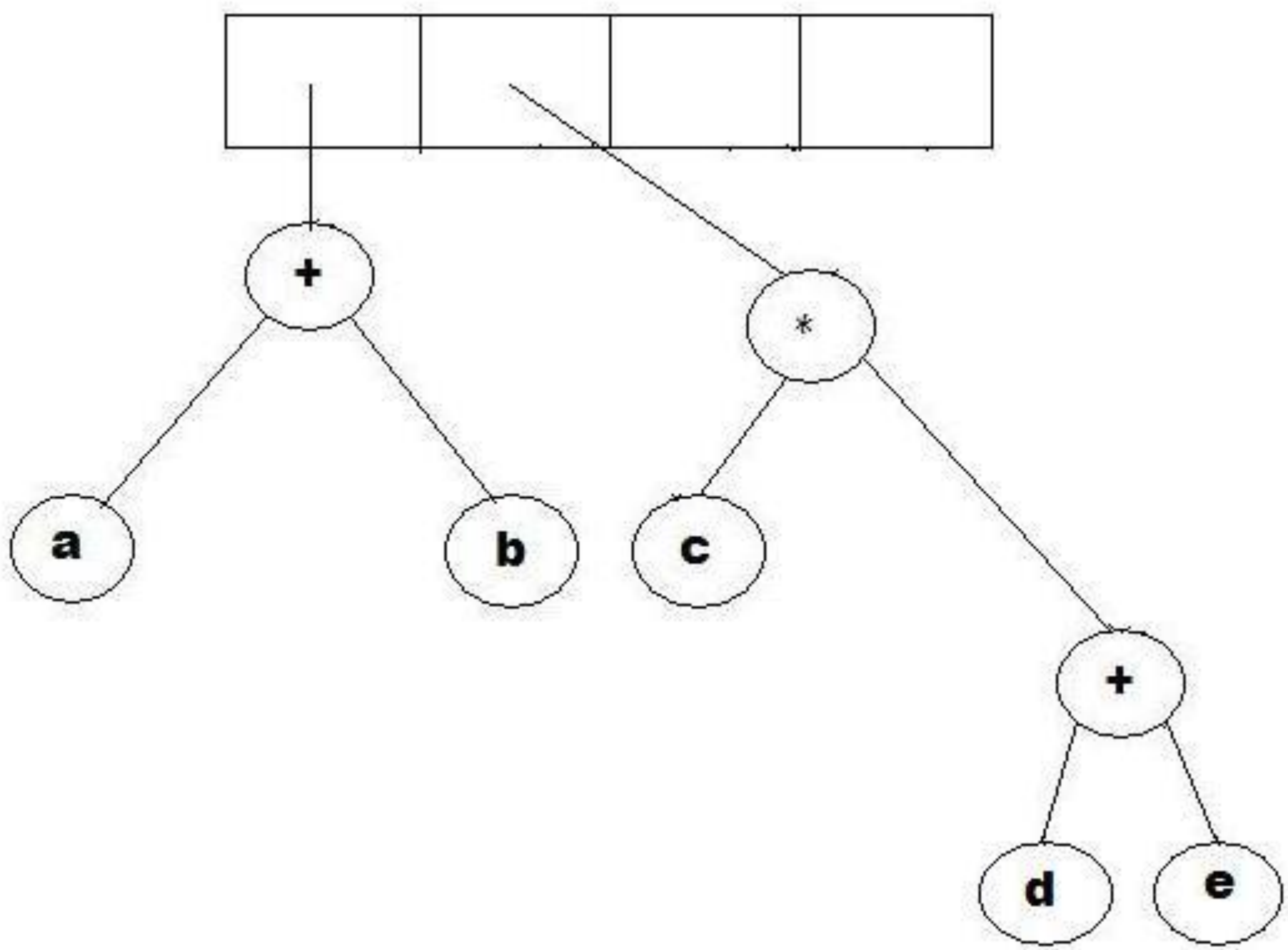
Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



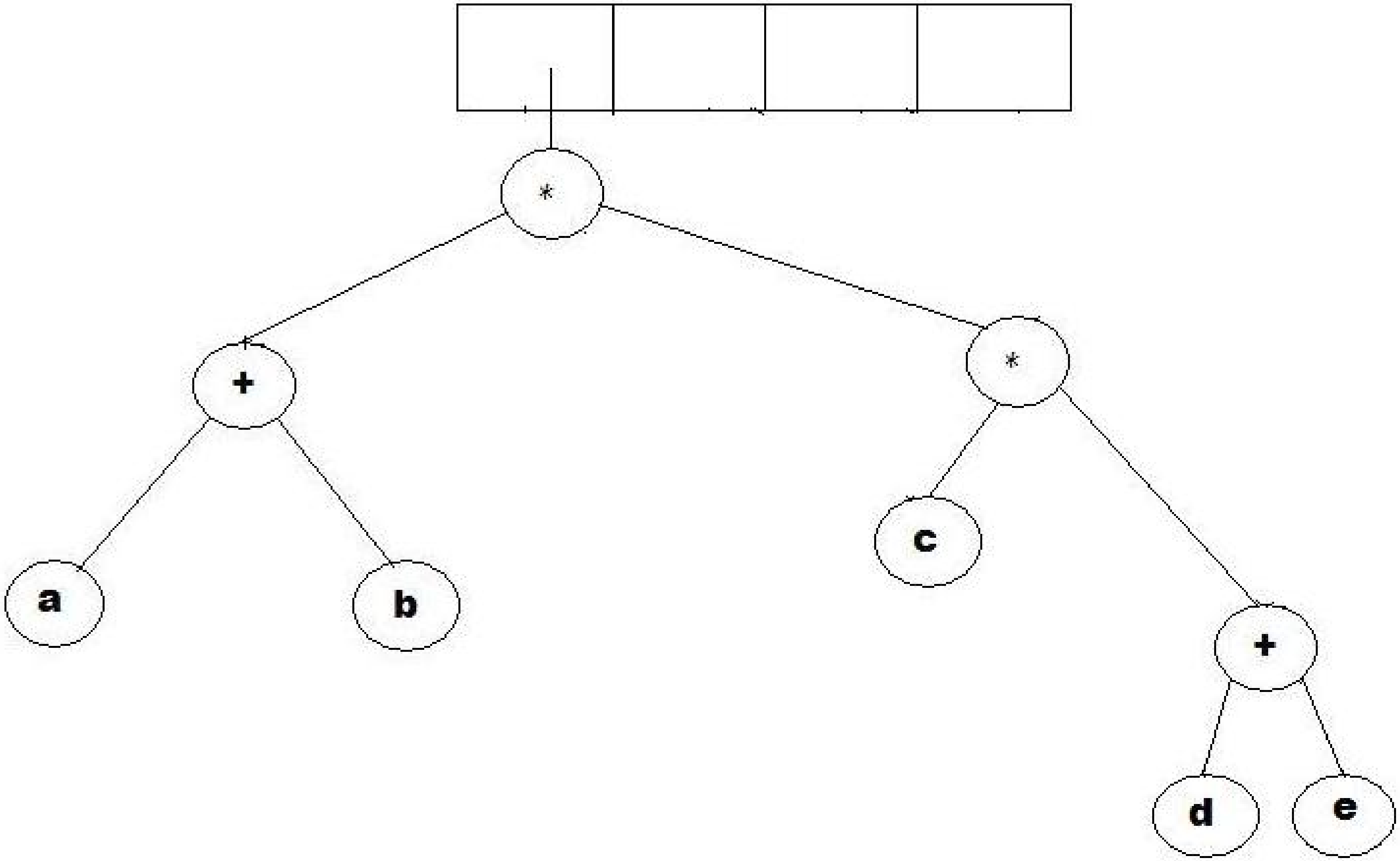
Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



Algorithm: convert postfix into expression tree

Example: a b + c d e + * *



Why use expression trees?

- Perhaps the simplest thing to do is to print the expression represented by the tree.
- But it would be better if we can somehow evaluate the tree to get results.
- Any suggestions on how this can be done?

How to evaluate an expression tree?

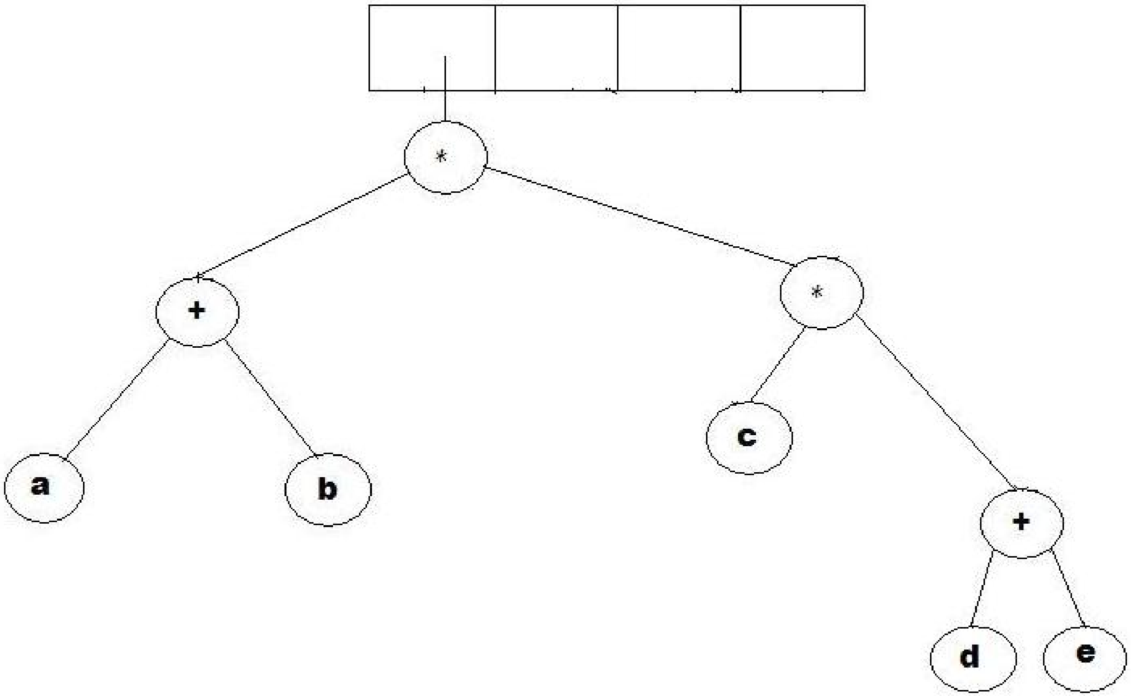
- To evaluate a binary expression tree, we just need to do a post-order traversal of the tree, and ask each node to evaluate itself.

- An operand node evaluates itself by just returning its value. An operator node has to apply the operator for that node to the result of evaluating its left sub-tree and its right sub-tree.

How to evaluate an expression tree?

- Example: a b + c d e + * *

  1 2 + 3 4 5 + * *



Questions?

“He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever”

Chinese Proverb

“The wise man doesn't give the right answers, he poses the right questions.”

Claude Levi-Strauss

“A wise man can learn more from a foolish question than a fool can learn from a wise answer.”

Bruce Lee