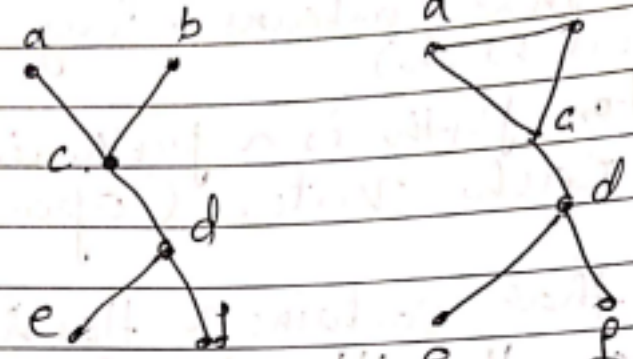
Trees

Let $G = (V, E)$ be a loop-free undirected graph. Graph G is called a tree. If G is connected & contains no cycles.



tree

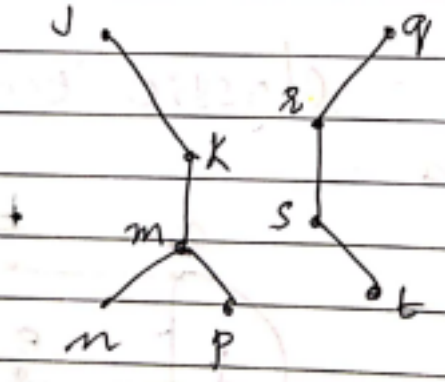
G_1

Not a tree (cycle)

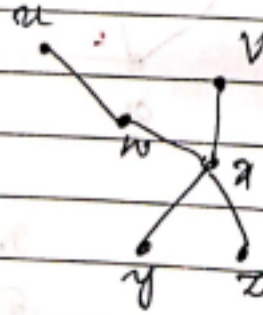
G_2

G_1 is a tree, but G_2 is not a tree as it contains a cycle.

Consider G_3.



This graph G_3 is not connected, so it cannot be a tree, but each component of G_3 is a tree, hence G_3 can be called as a forest.



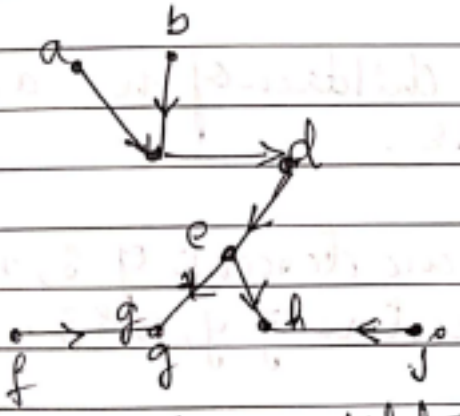
Spanning tree :-

When we consider G_1 & G_2, here G_1 contains all the vertices of G_2 and G_1 is a tree therefore G_1 is is called Spanning tree of G_2

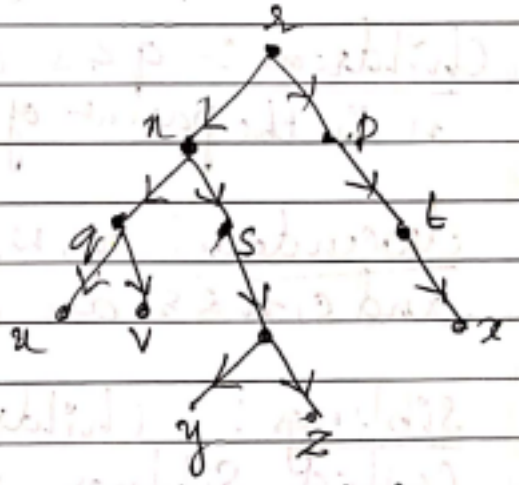
∴ Spanning tree is a Minimally connected Graph that connects all the vertices together.

Rooted trees :-

If G is a directed graph, then G is called a directed tree if the Undirected graph associated with G is a tree. When G is directed tree, G is called rooted tree if there is a Graph Vertex called root in G with indegree = 0 and all other vertices having indegree = 1



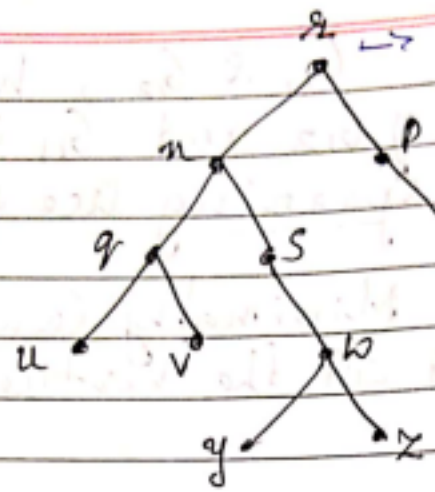
Not a rooted tree



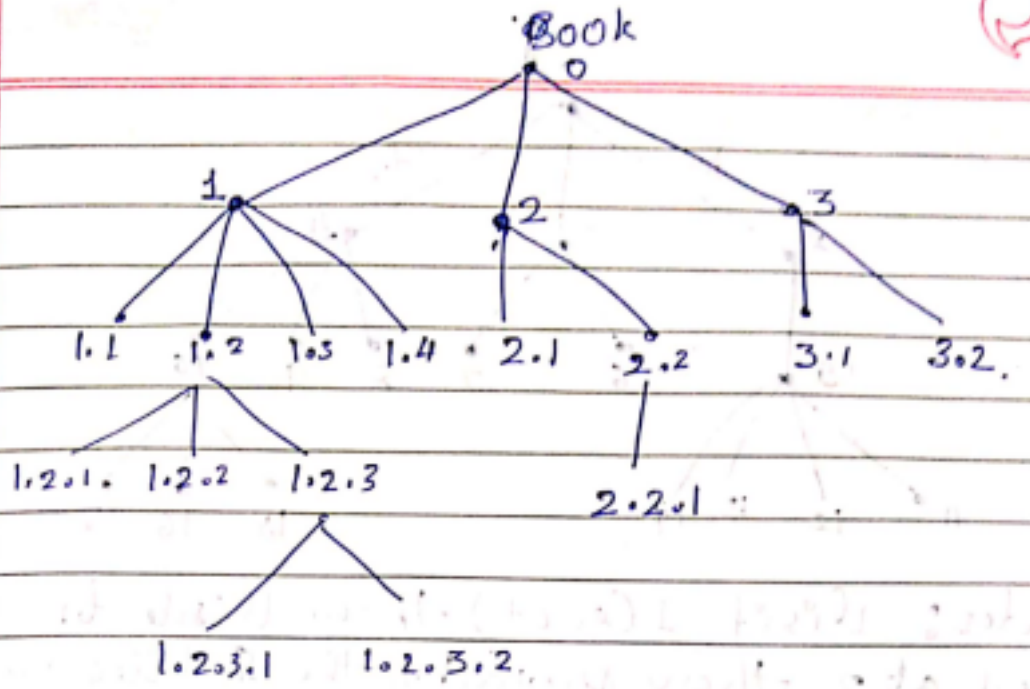
rooted tree.

Rooted tree can be written by omitting direction as it is assumed that direction is from upper level to lower level.

Ex: Used for representing the table of contents of a book.



- leaf nodes: Vertex with outdegree = 0 called leaf eg: u, v, y, z, x. others are called internal vertices or bra node.
- no. of edges in path is called the le of the lete eg:- level of s from r is 2 r is at level 3, y is at 4 etc.
- Children :- q & s are children of n a n is the parent of q & s.
- descendents :- y, w, z are descents of s, r and s, n & r are ancestors of y, w & z.
- Siblings :- Children of same parent a called Siblings eg:- q & s are Siblings u & v are siblings.

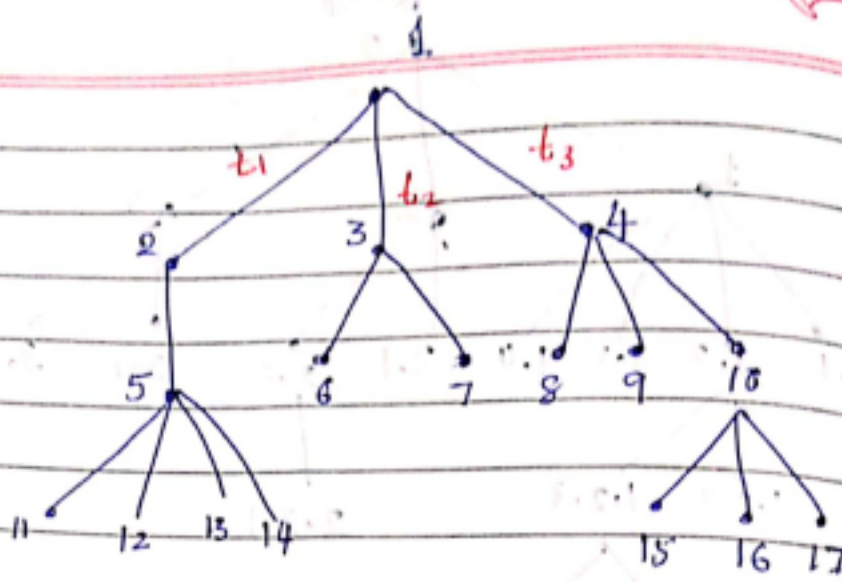


In the tree T shown above, the edges (or branches) leaving each internal vertex are ordered from left to right. Hence T is called an Ordered rooted tree

Tree Traversal. (ordering)

- Inorder – left subtree, root, Right subtree
- preorder – root, left subtree, Right subtree
- postorder – left subtree, Right subtree, Root

Example :-

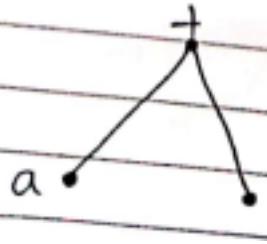


Preorder: Visit 1 (root) then Visite t_1 rooted at 2, then proceed to the subtree rooted at vertex 5. We go to the subtree rooted 11. This subtree has no children, so we visit and backtrack to 5, from 5 we visit (12,13 then backtrack from 5 to 2 then 2 to 1, then Visite t_2 rooted at 3, then 6 and 7 backtrack to 3 then 1, then visite t_3 rooted at 4, then 8, 9, 10, then 15, 16, 17

Therefore preorder: 1,2,5,11,12,13 14,3,6,7,4,8, 10,15,16,17

Similarly postorder: 11,12,13,14,5,2,6,7,3, 8,9 15 16 17 10 4 1.

Expression tree: Is a Binary tree where the internal root nodes represent operators and leaf nodes represent operands.

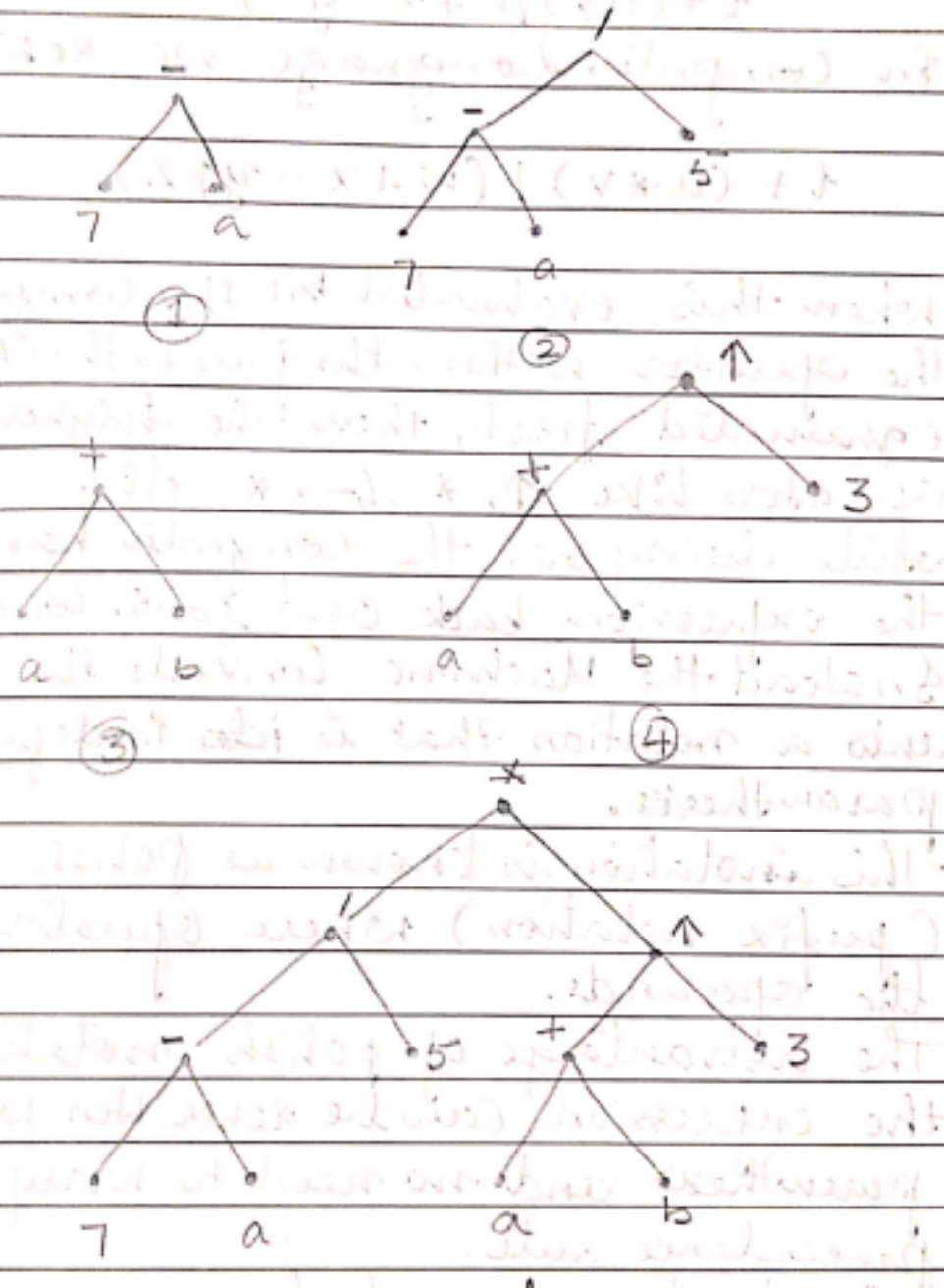


a+b

Construct a Binary tree (expression tree) for the algebraic expression

$((7-a)/5) * ((a+b) \uparrow 3)$

steps :



expression tree

Evaluation of a Expression

Consider an expression.

$$t + (u * v) / (w + x - y \uparrow z)$$

In computer language we represent this

$$t + (u * v) / (w + x - y \uparrow z)$$

- when this evaluated by the computer, the operator within the parantheses is evaluated first, then the higher precedence operators like $\uparrow$, $*$, $/$, $-$, $*$ etc.
while doing so, the computer has to scan the expression back and forth continously. Instead the Machine converts the expression into a notation that is ~~ida~~ independent of paranthesis.

* This notation is known as Polish notation (prefix notation) where operator precede the operands.
* The advantage of polish notation is that the expression can be rewritten without parentheses and no need to worry about precendence rule.
* Evaluation proceeds from right to left.

Eg: convert the expression to polish notation

$t + (u * v) / (w + x - y \uparrow z)$

$t + (*uv) / (w + x - \uparrow yz)$

$t + (*uv) / (w + - x \uparrow yz)$

$t + *uv / +w - x \uparrow yz$

$= + t / * u v + w - x \uparrow y z$

+ t / * qv + w - x yz

Let t = 4, z = 2, v = 3, w = 1, x = 7, y = 2, z = 3

Substituting the values.

+ 4 / * 23 + 1 - 9 23

← 23 = 8

Evaluate proceed from right to left

+ 4 / * 2 3 + 1 - 9 8

9 - 8 = 1

+ 4 / * 2 3 + 1 1

2 + 1 = 2

+ 4 / * 2 3 2

(Use stack)

+ 4 / 6 2

2 × 3 = 6

+ 4 3

6 / 2 = 3

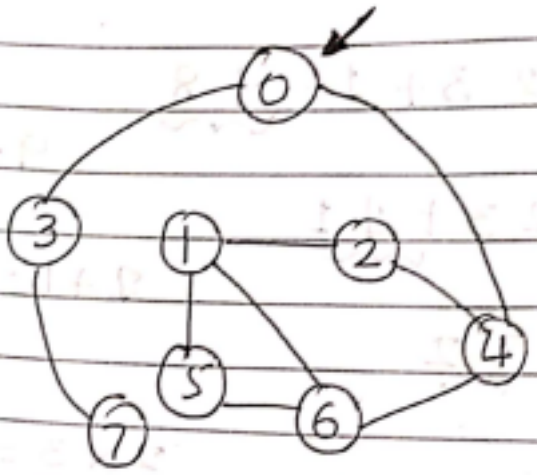
4 + 3 = 7

= 7

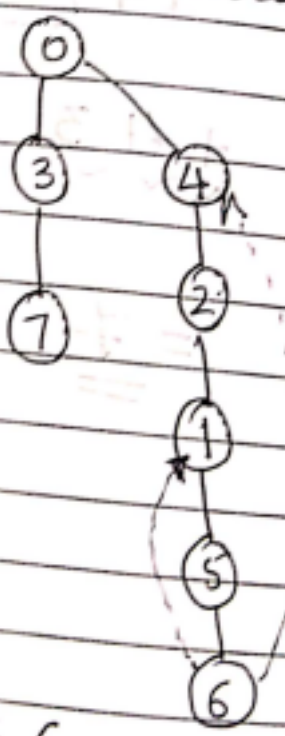
Graph Traversal Techniques

- Depth-first Search. } finds the spanning tree from the Graph if it is connected or forest if it is not connected
- Breadth-first Search. }

Consider a Undirected, connected Graph.



Spanning tree



0	Ø 1
1	Ø 1
2	Ø 1
3	Ø 1
4	Ø 1
5	Ø 1
6	Ø 1
7	Ø 1

Visited array

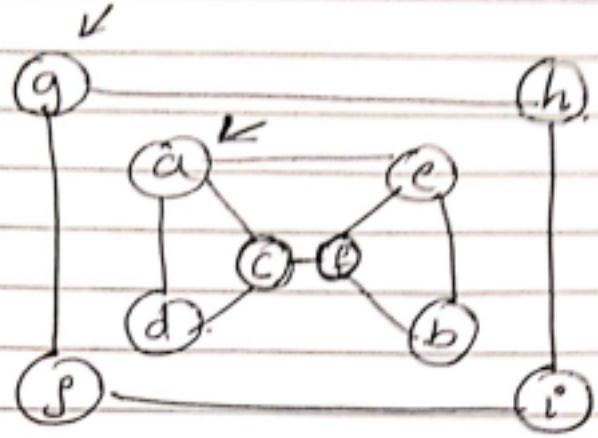
3	dfs(6)
4	dfs(5)
5	dfs(1)
6	dfs(2)
7	dfs(4)
1	dfs(7)
2	dfs(3)
8	dfs(0)

Stack

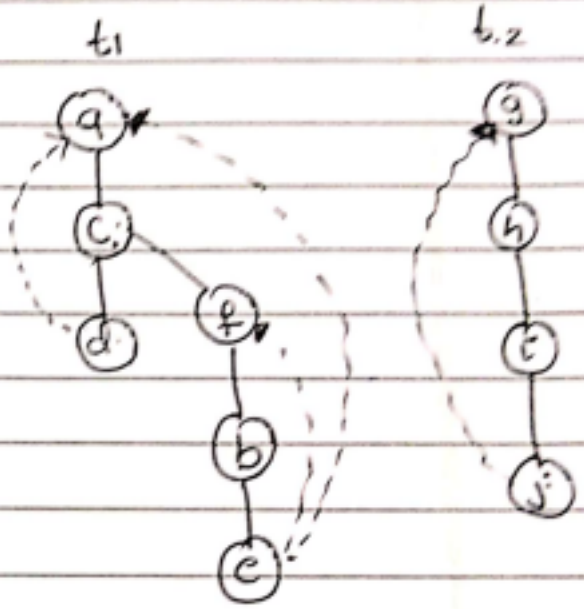
Order of Visit - 0 3 7 4 2 1 5 6

Order in which they became deadend 7 3 6 5 1 2 4 0

Consider a Undirected, Disconnected Graph.



g	1		dfs(j)	7
b	5		dfs(i)	8
c	2	t2	dfs(h)	9
d	3		dfs(g)	10
e	6		dfs(e)	1
f	4		dfs(b)	2
g	7	t1	dfs(f)	3
h	8		dfs(d)	4
i	9		dfs(c)	5
j	10		dfs(a)	6



Visited array

Stack.