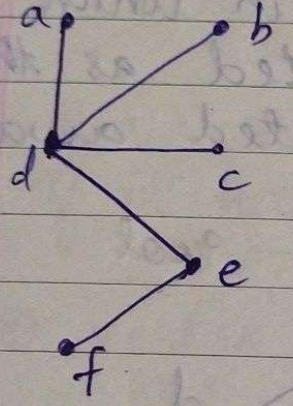
28/02/2020
Friday

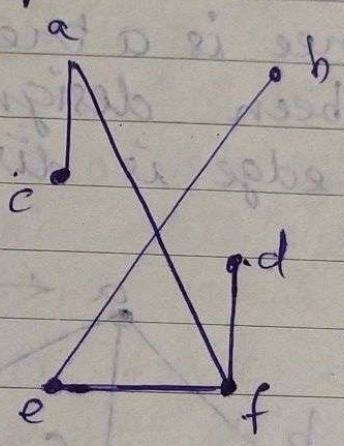
Module - 3

Tree

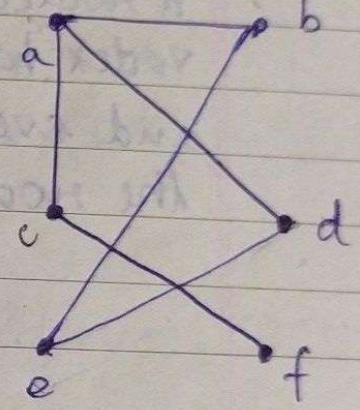
A tree is a connected undirected graph with no simple circuit



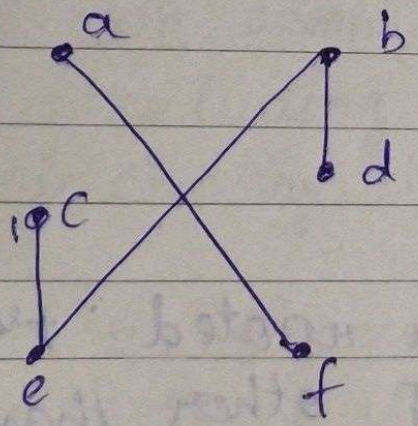
G_1
Tree



G_2
Tree



G_3 (cycle is circuit)
Not a tree

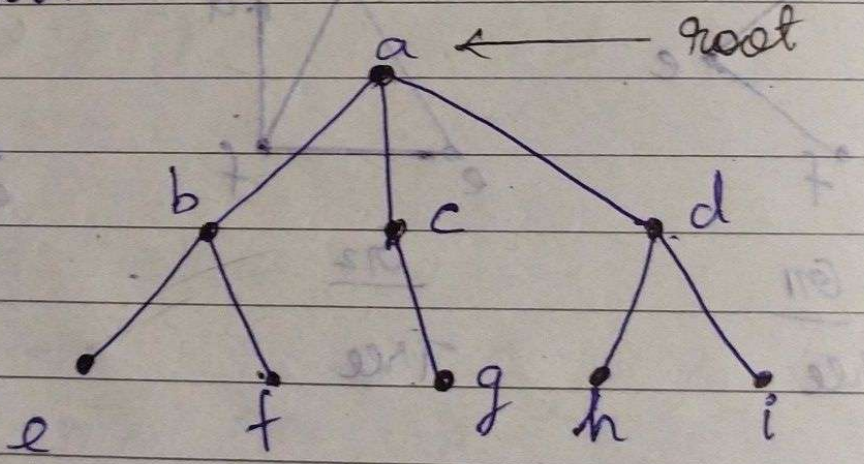


G_4 Forest (trees of one, or two)

An undirected graph is a tree if and only if there is an unique simple path between any 2 of its vertices.

Rooted Trees

A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root



Tree terminology

Suppose that T is a rooted tree

(a) If v is a vertex in T other than root the parent of v is the unique vertex

u such that there is a directed edge from u to v.

(b) When u is the parent of v, v is called the child of u.

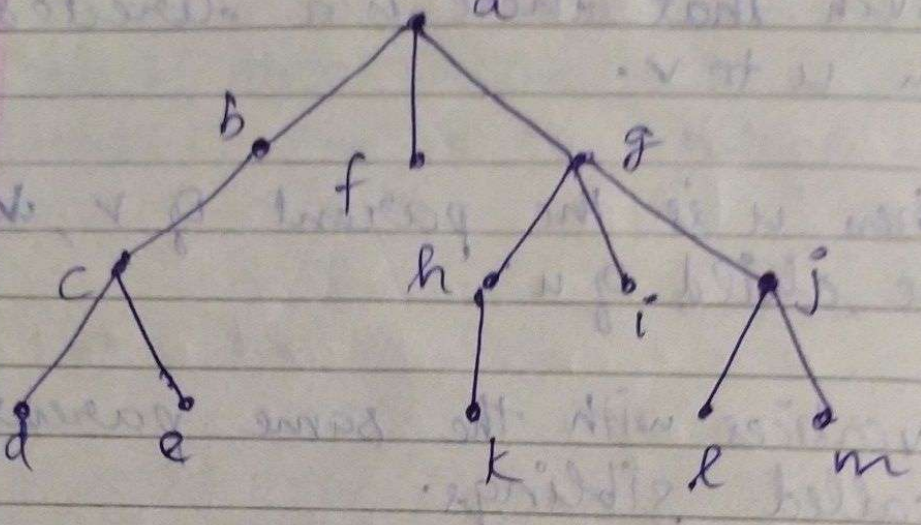
(c) vertices with the same parent are called siblings.

(d) a vertex of a tree is called a leaf if it has no children.

(e) vertex that have children are called internal vertices.

(f) the ancestors of the vertex other than the root are the vertices in the path from the root to its vertex excluding the vertex itself and including the root.

(g) the descendants of the vertex v are those vertices that have v as an ancestor.



Ques (a) find the parent of c :- b

(b) The children of g :- h, i, j

(c) The siblings of h :- i, j

(d) All ancestors of e :- a, b, c

(e) All internal vertices :- a, b, g, c, h, j

(f) All leaf :- f, d, e, i, k, l, m

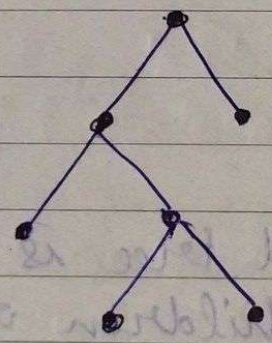
(g) The descendants of b :- d, e, c

descendants of g :- h, i, j, k, l, m

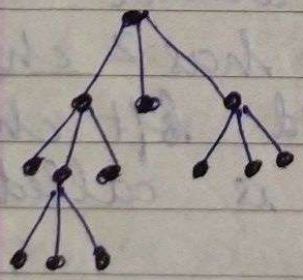
m - array tree

A rooted tree is called an m-array tree if every internal vertex has no more than m children.
The tree is called a full m-array tree if every internal vertex has exactly m children ~~each~~.

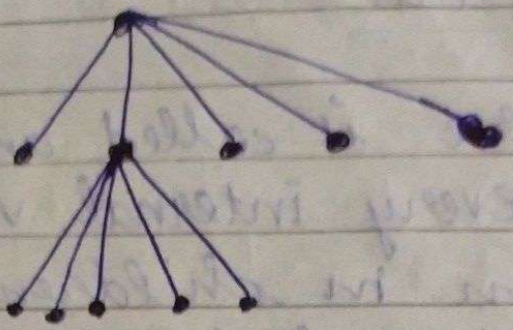
An m-array tree with $m = 2$ is called a binary tree.



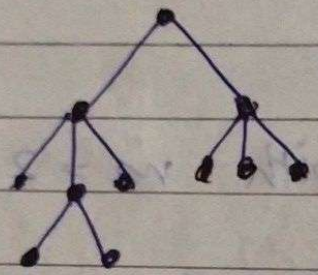
binary 2-array tree



full 3 array tree



full 5 array tree



2, 3 array tree.

Ordered rooted tree

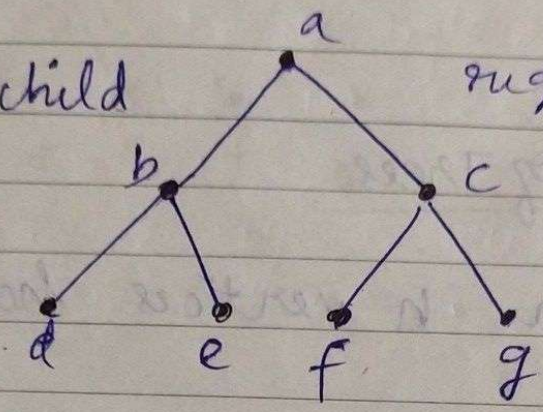
An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.
If an internal vertex has 2 children the first child is called left child and the second child is called right child.
The tree rooted at the left child of a vertex is called left sub-tree

The tree rooted at the right child of a vertex is called the right sub-tree of the vertex

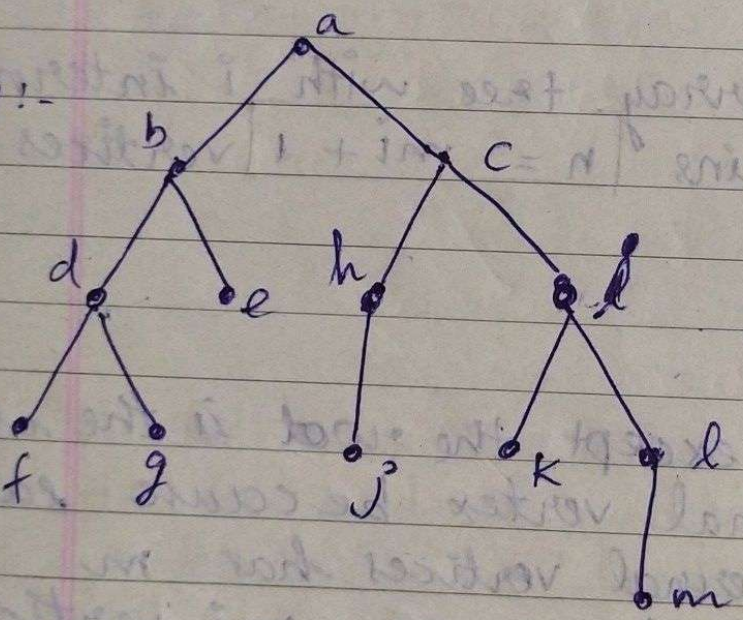
left child

right child

eg:-



Ques:-

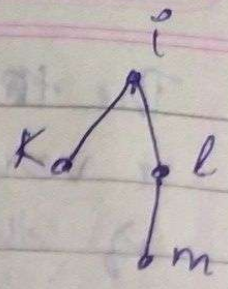


(a) Find left child of d :- f

(b) Right child of c :- i

(c) Draw left subtree of c :-

(d) Draw right subtree of c :-



Properties of Trees

(1) A tree with n vertices has n-1 edges

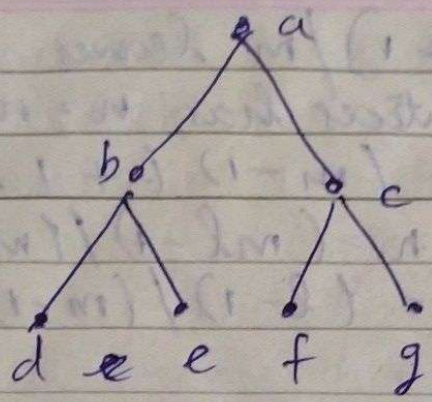
(2) A full m-array tree with i internal vertices contains $n = mi + 1$ vertices

Proof :-

Every vertex except the root is the child of an internal vertex because each of the i internal vertices has m children so there are mi vertices in the tree other than the root.

Therefore the tree contains $n = mi + 1$ vertices

eg:-



2-array tree

$m = 2$; $i = 3$

$n = m * i + 1$

$= 2 \times 3 + 1$

$n = 7$ (no of vertices = 7)

(3) A full m-array tree with i internal vertices and l- number of ~~trees~~ leaves gives, $\boxed{n = l + i}$ vertices.

Proof:-

Suppose that T is a full m-array tree with (i) n vertices has $i = (n-1)/m$ internal vertices and

$l = [(m-1)n + 1)/m$ leaves

(ii) i internal vertices has $n = mi + 1$ vertices and $l = (m-1)i + 1$ leaves

(iii) l leaves has $n = (ml - 1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices.

$$n = mi + 1$$
$$n = l + i$$
$$i = (n-1)/m$$
$$l = (n - i)$$

$$l = (n-i)$$
$$(n-i) = [n - (n-1)/m$$
$$= [((m-1)n + 1)/m$$

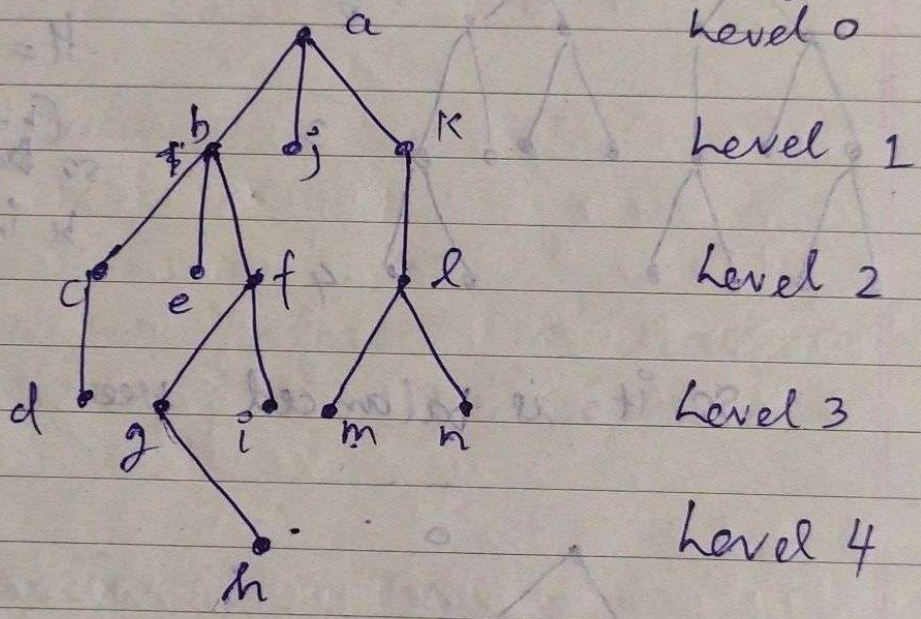
02/03/2020
Monday

Level and Height of a Rooted Tree

The level of a vertex in a rooted tree is the length of the unique path from the root to its vertex.

(6)

The level of the root is defined to be 0.
The height of a rooted tree is the maximum of the levels of its vertices.
In other words, the height of the rooted tree is the length of the longest path from the root to any vertex.



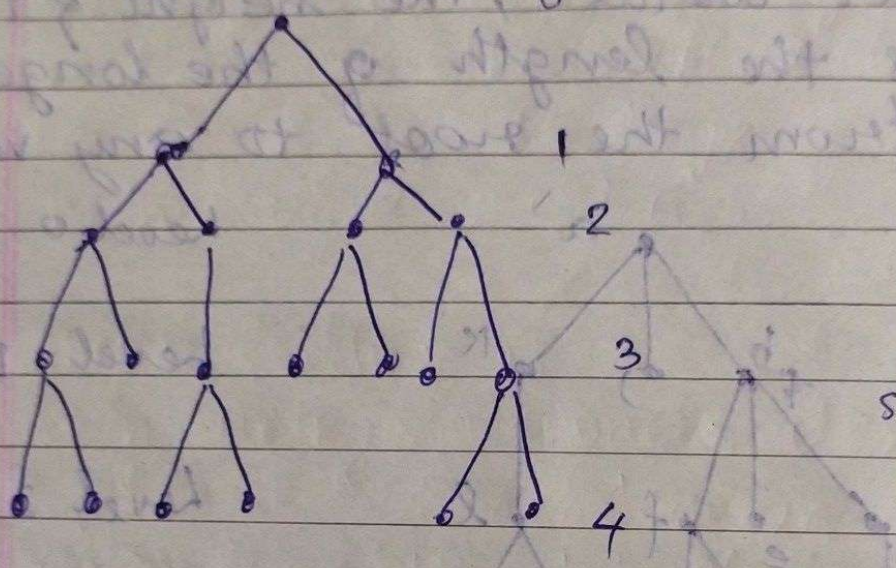
Height = maximum of the level
= 4

Level 0 - a
Level 1 - b, j, k
Level 2 - c, e, f, l
Level 3 - d, g, i, m, n
Level 4 - h

A rooted m-array tree of height h is balanced if all the leaves are at levels h or (h-1).

Ans :-

(a)



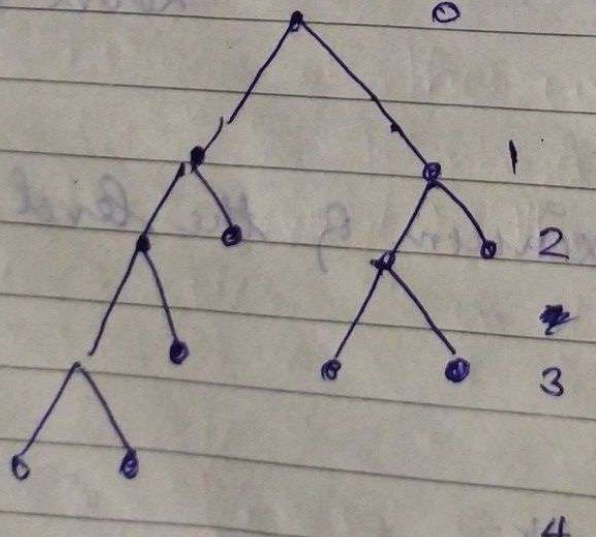
H = 4

(h-1 = 3)

so ~~leaf~~ should be in 4 or 3

So it is balanced tree

(b)

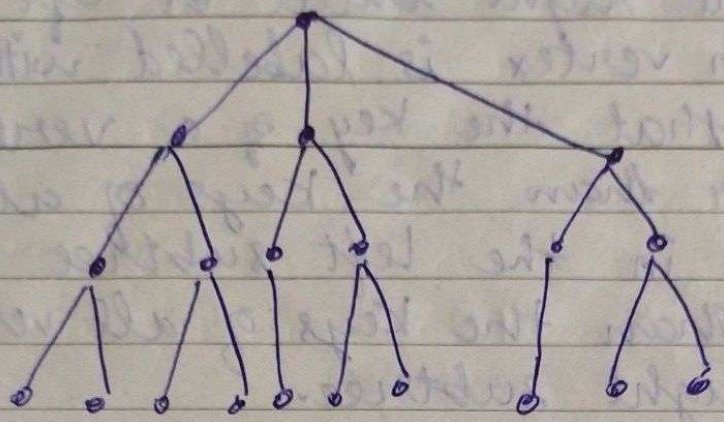


h = 4

(leaf can be seen at level 2 but leaf should only be seen in level 3 or 4)

Not balanced tree.

c)



h = 3
(all the leaf are in level 3)

Balanced tree

- There are atmost m^h leaves in an m-array tree of height h.

Application of Trees

1. Binary Search Tree (BST)

Binary Search Tree is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has ~~no~~ more

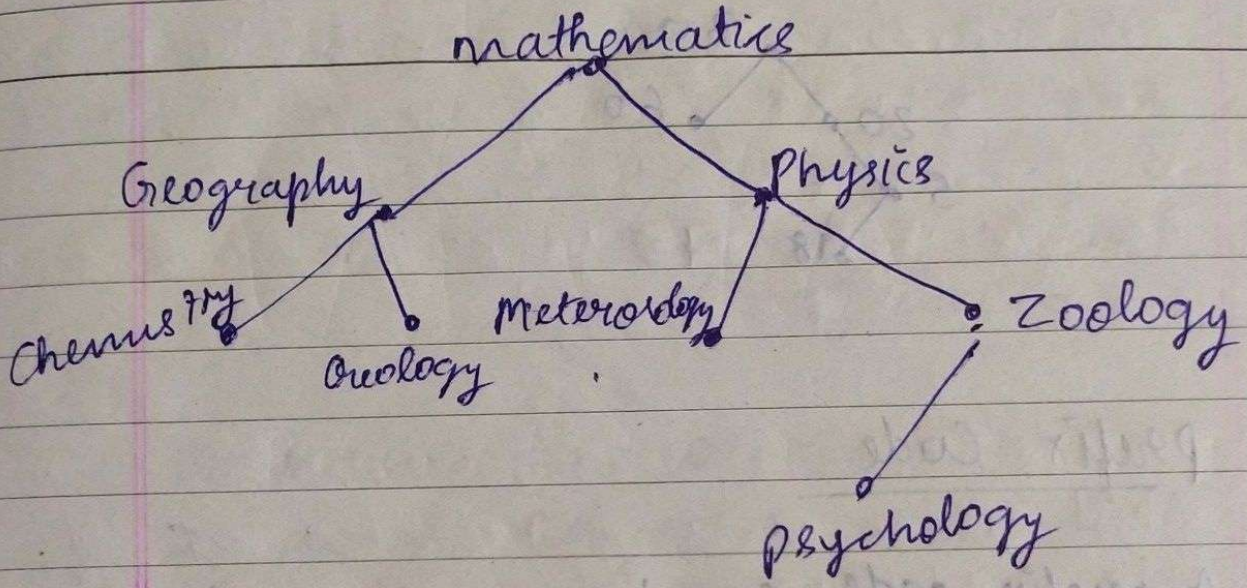
than one right child or left child and each vertex is labelled with a key so that the key of a vertex is larger than the keys of all vertices in the left subtree and smaller than the keys of all vertices in the right subtree.

The procedure is :- start with a tree containing just one vertex namely root. The first item in the list is assigned as the key of the root.
To add a new item first compare it with the keys of vertices already in the tree.
Starting at the root, moving to the left if the item is less than the key and moving to the right if the item is greater than the tree.

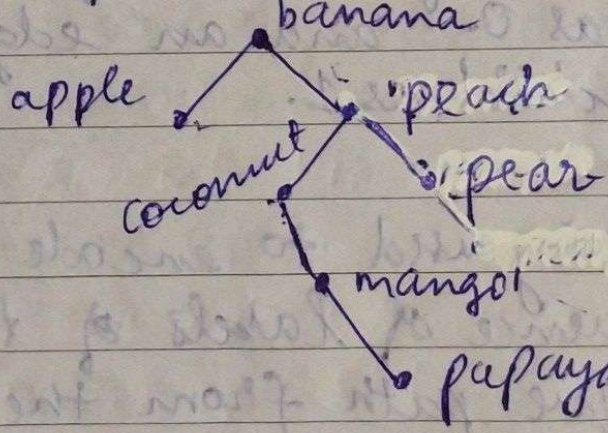
Ques Form a binary search tree for the words mathematics, physics, geography, zoology, meteorology, geology,

greater - right
lesser - left

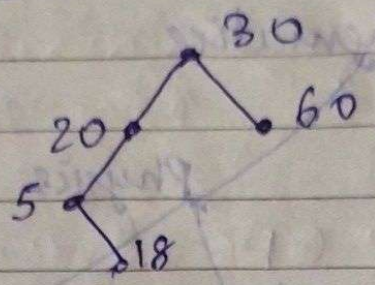
psychology and chemistry



Ques Build a binary search tree for the words banana, peach, apple, pear, coconut, mango, papaya using alphabetical order



Ques Draw the binary search tree of 30, 20, 5, 60 and 18



03/03/2020
Tuesday

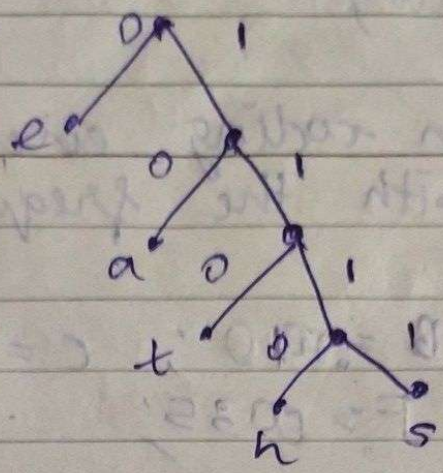
2. Prefix Code

A prefix code can be represented using a binary tree where the characters are the labels of the leaves in the tree.

The edge of the trees are labelled so that an edge leading to a left child is assigned as 0. and an edge leading to the right child as 1.

The bit string used to encode a character is a sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label

eg:-



e = 0
a = 10
t = 110
h = 1110
s = 1111

Bit string 1111 10 1110 0
Ans:→ sane

Ques:- Given the coding scheme
a: 001 ; b: 0001 e: 1 ; h: 0000
s: 0100 ; t: 011 ; x: 01010

find the word represented by ~~(a) 011~~.

(a) 011 | 0100 | 011
t e s t

(b) 0001 | 1 | 1 | 0000
b e e h

(c) 01001 | 01010
s e x

(d) 011 | 001 | 01010
t a x

greater - left
lesser - right

Huffman Coding

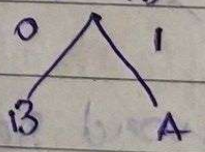
Ques:- Using huffman coding encode the following symbols with the frequencies listed:-

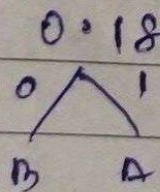
A = 0.08 ; B = 0.10 ; C = 0.12 ; D = 0.15
E = 0.20 ; F = 0.35

What is the average number of bits used to encode a character

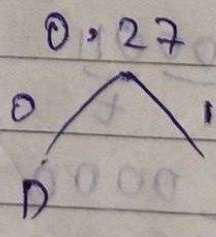
A = 0.08 ; B = 0.10 ; C = 0.12 ; D = 0.15 ; E = 0.20 , F = 0.35

C = 0.12 ; D = 0.15 ; 0.18 E = 0.20 , F = 0.35

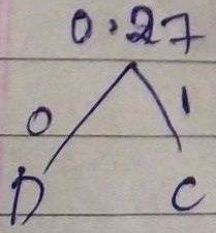




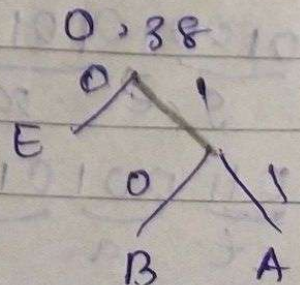
E = 0.20

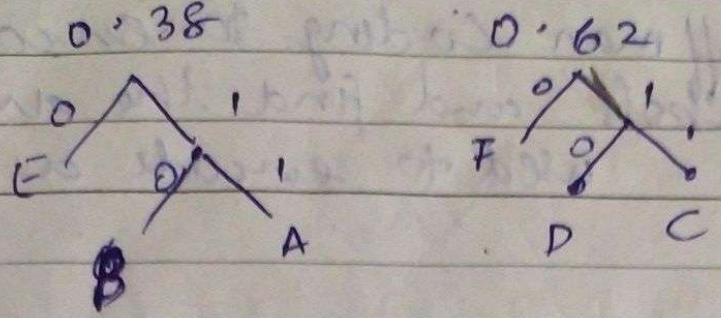


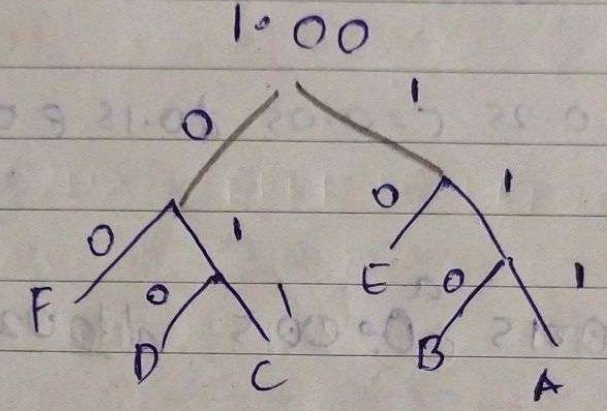
F = 0.35



F = 0.35







A = 111 ; B = 110 ; C = 011 ;
D = 010 ; E = 10 ; F = 00 ;

Average = $3 \times 0.08 + 3 \times 0.10 + 3 \times 0.12 + 3 \times 0.15 + 2 \times 0.30 + 2 \times 0.35$

$= 0.24 + 0.30 + 0.36 + 0.45 + 0.60 + 0.70$

$= 0.90 + 1.55$

$= 2.45$

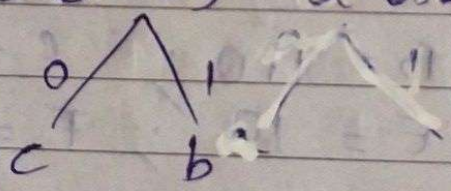
Ques:- Using Huffman coding to encode these symbols and find the average no. of bits used to encode a character.

(a) a: 0.20 b: 0.10 c: 0.15 d: 0.25 e: 0.30

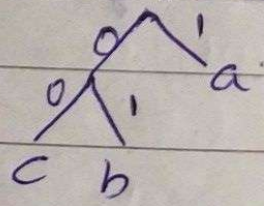
(b) a: 0.10 b = 0.25 c = 0.05 d 0.15 e 0.30 f 0.07
g: 0.08

Ans (a) b 0.10 c 0.15 a 0.20 d 0.25 e 0.30

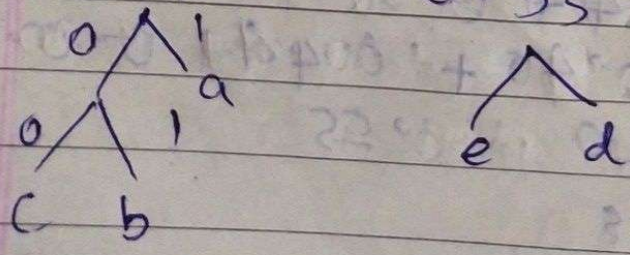
a 0.20 d 0.25 d 0.25 e 0.30

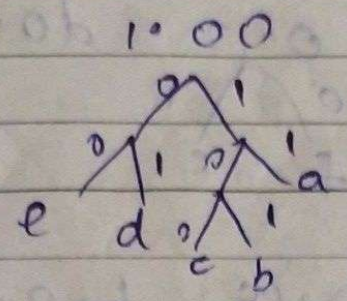


d 0.25 e 0.30 0.45



0.45 0.55





a: 11 b: 101 c: 100
d: 01 e: 00

Average = $2 \times 0.20 + 3 \times 0.10 + 3 \times 0.15 + 2 \times 0.25 + 2 \times 0.30$

$= 0.40 + 0.30 + 0.45 + 0.50 + 0.60$

$= 0.70 + 1.10 + 0.45$

$= 1.15 + 1.10$

$= 2.25$

(b) c0.05 f0.07 g0.08 a0.10 d0.15 b0.25 e0.30

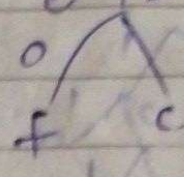
~~0.12~~ g0.08 a0.10 d0.15 b0.25 e0.30

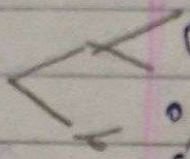
~~a~~ ~~c~~

f

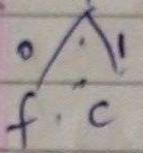
e 0.30

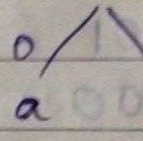
g 0.08 a 0.10 0.12 d 0.15 b 0.25



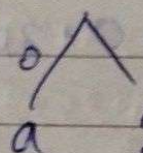


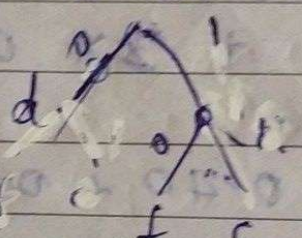
0.12 d 0.15 0.18 b 0.25 e 0.30



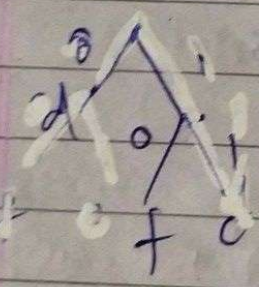


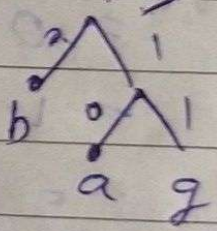
0.18 b 0.25 0.27 e 0.30



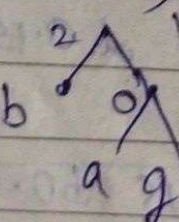


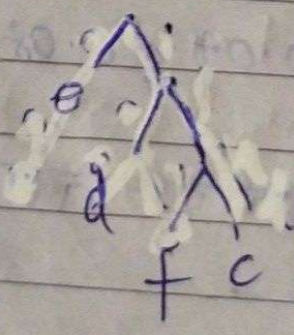
0.27 e 0.30 0.43

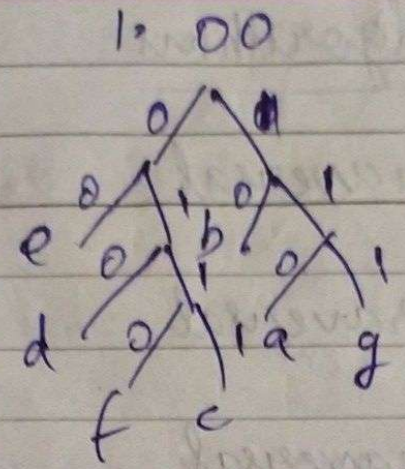




0.43 0.57







15
×3
45

a: 110 b:- 10 c:- 0111 d:- 010
e: 00 f 0110 g 111

11
23
75
25
23
5

Average = $3 \times 0.10 + 2 \times 0.25 + 4 \times 0.05 + 3 \times 0.15 + 2 \times 0.30 + 4 \times 0.07 + 3 \times 0.08$

$= 0.30 + 0.50 + 0.20 + 0.45 + 0.60 + 0.28 + 0.24$

$= 0.80 + 0.65 + 0.81 + 0.24$

$= 2.57$

80
81
1.61

1.61
0.81
2.26
0.24
2.50

visiting each node

visiting each vertex of a graph - traversal

Traversal Algorithms

① Pre-order traversal : Root LR

② Inorder traversal : LRoot R

③ Postorder traversal : LR Root

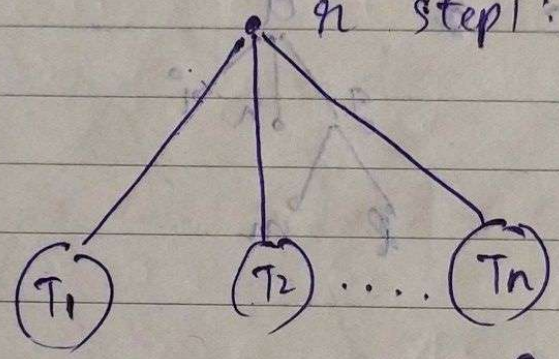
Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms

There are 3 types of such algorithms as ~~defi~~ listed above:-

① Pre-order Traversal

Let T be an ordered rooted tree with root r. If T consists only of r then r is the preorder traversal of T.

Otherwise suppose that $T_1, T_2 \ldots T_n$ are the subtrees at r from left to right in T. The tree order traversal begins by visiting R. It continues by traversing T_i in preorder then T_2 in preorder and so on until T_n is traversed in preorder.

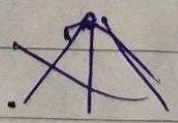


Step 2: visit T_1 in preorder

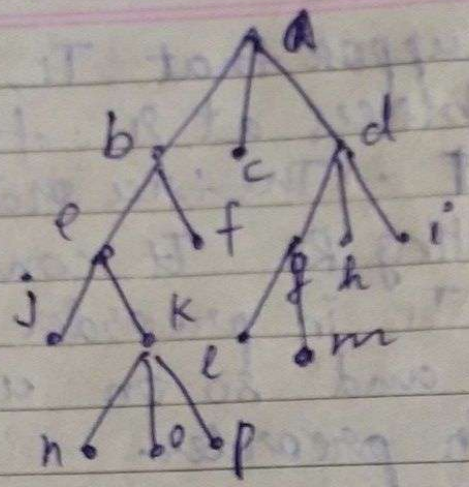
Step 3: visit T_2 in preorder

Step n+1: visit T_n in preorder

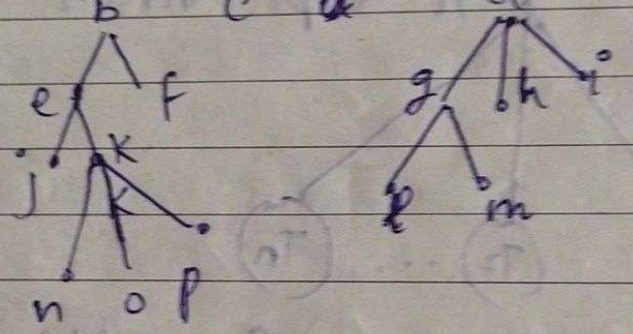
eg:-



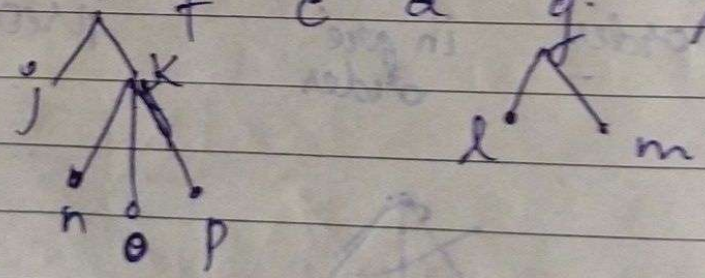
eg:-



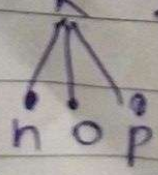
→ ① a b c d



→ ② a b e f c d g h i



→ ③ a b e j k f c d g l m h i



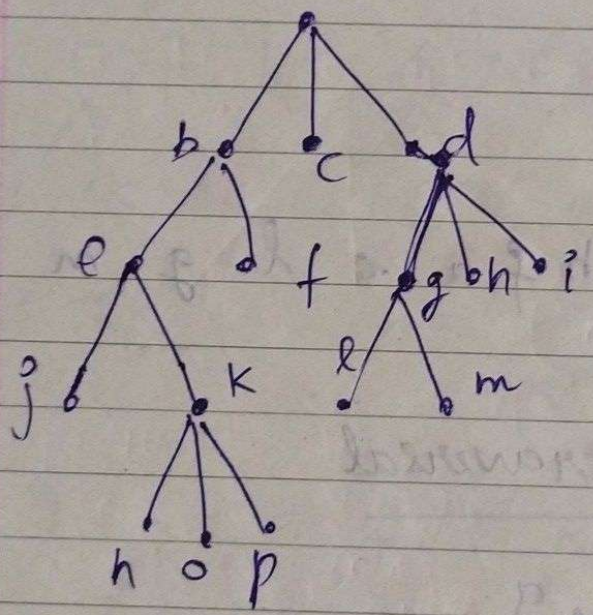
(6)

→ (4) a b e j k n o p f c d g l m h i

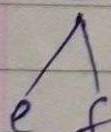
The preorder traversal order.

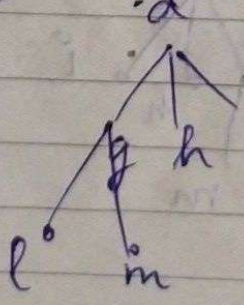
(2) Inorder traversal

eg:-

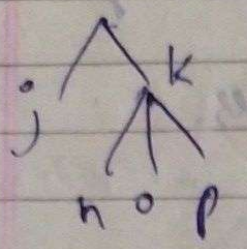


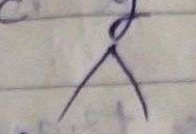
(1) ⇒ b a c d



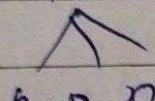


② → e b f a c g i d h i





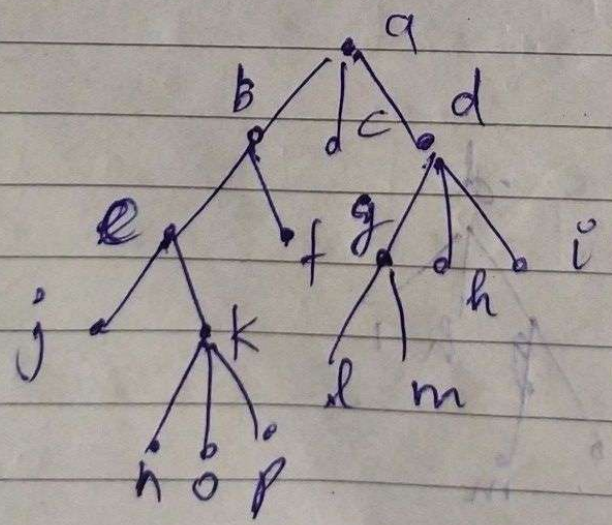
③ → j e k f a c l g m d h i



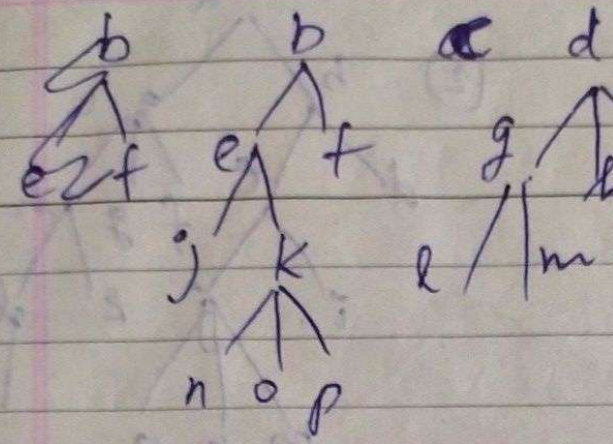
④ → j e n k o p b f a c l g m d h i

③ Post order traversal

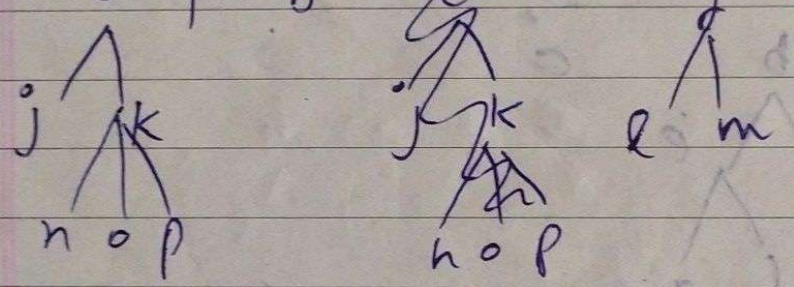
eg:-



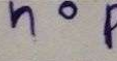
→ ① b b c d a



⇒ ② e f b e c g h i d a

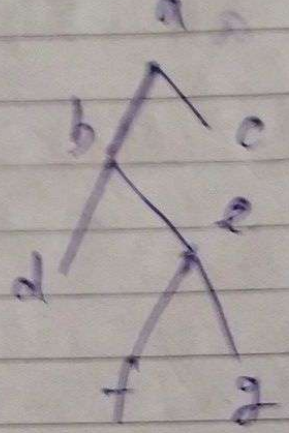


⇒ ③ j k e f b c l m g h i d a

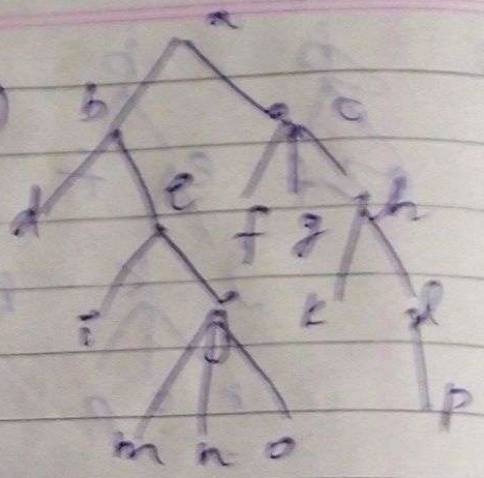


⇒ ④ j n o p k e f b c l m g h i d a

(1)

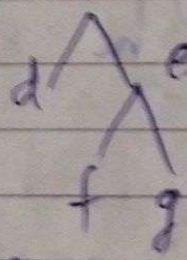


(2)



pre order

(1) ⇒ a b c



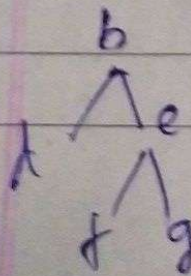
⇒ a b d e c

f g

⇒ a b d e f g c

Inorder

⇒



a c

⇒ d b e a c

f g

⇒ d b f e g a c

Postorder

b b c a

⇒

d e

f g

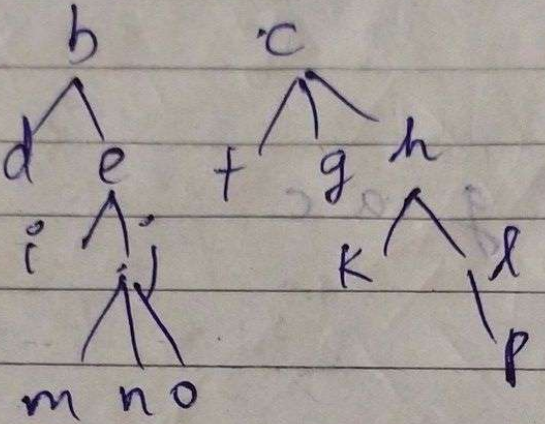
⇒ d e b c a

f g

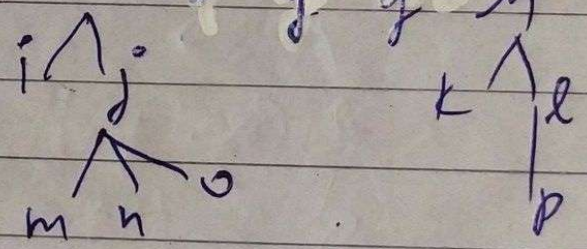
⇒ d f g e b c a

① ⇒ Preorder

⇒

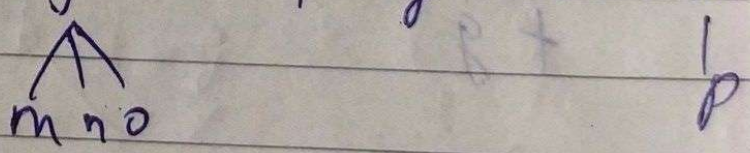


⇒ a b d e



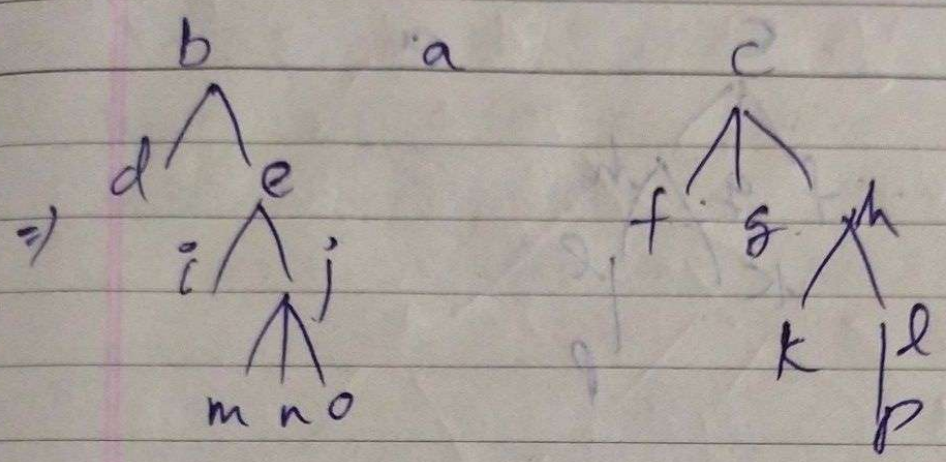
f g h

→ a b d e i j c f g h k l

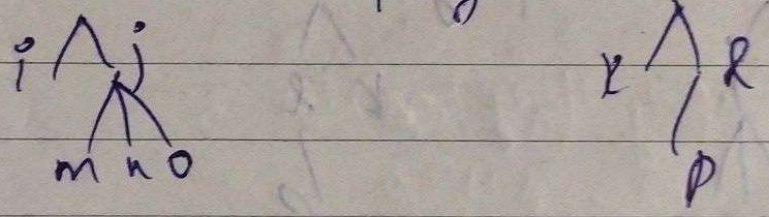


⇒ a b d e i j m n o c f g h k l p

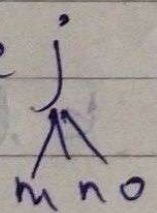
Inorder



⇒ d b e a f g c h



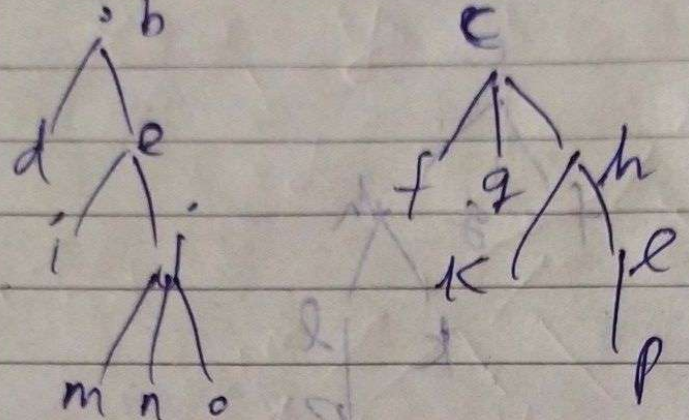
⇒ d b i e j a f g c k h l



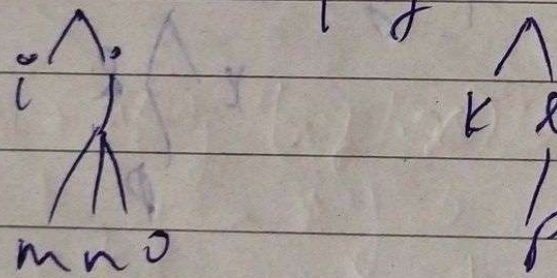
⇒ d b i e m n j o a f g c k h p l

Postorder

⇒



⇒ d e b f g h c a



⇒ d i j e b f g k l h c a

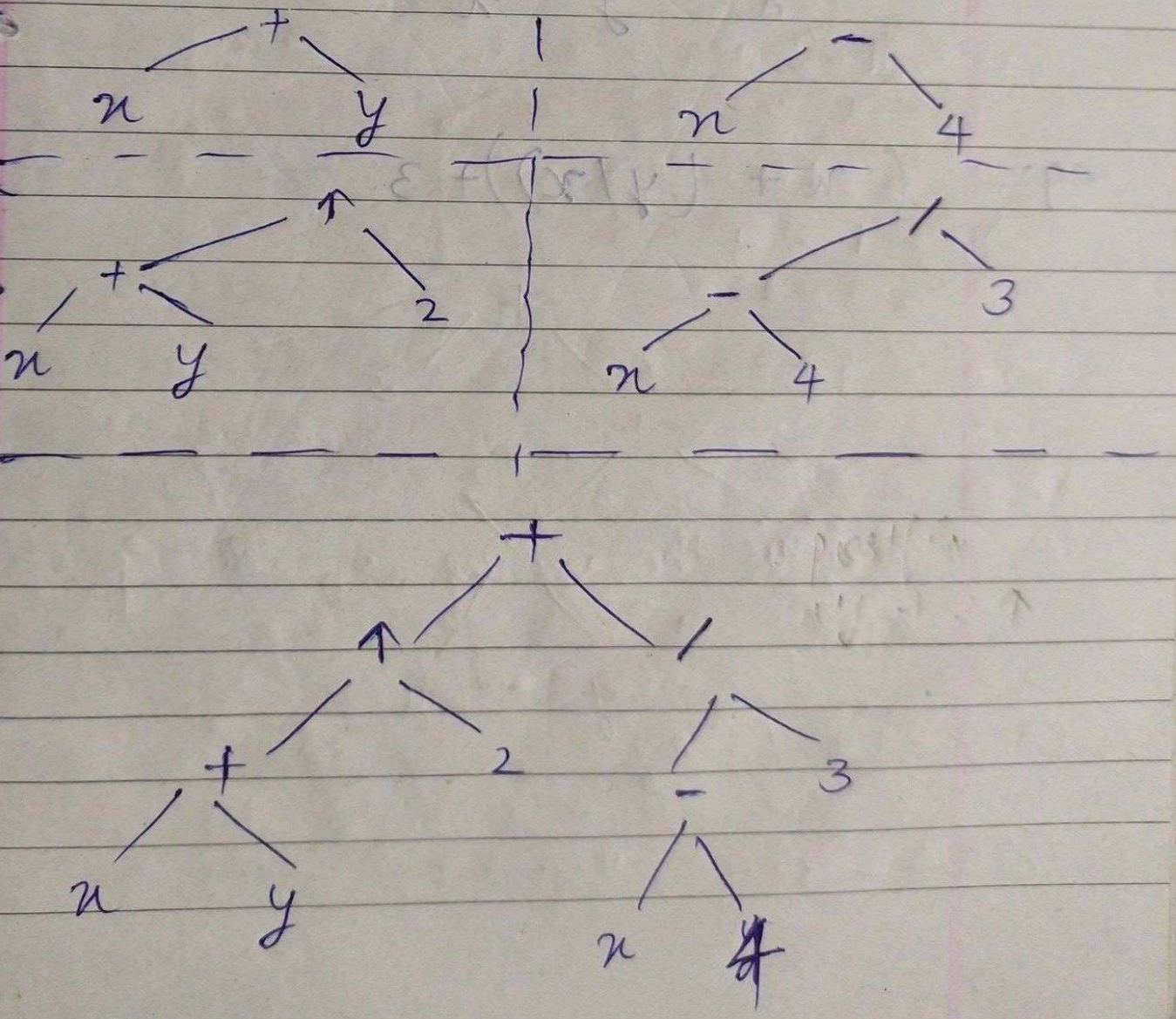
⇒ d i m n o j e b f g k p l h c a

2 June 2020

Ordered Rooted tree

An Ordered rooted tree
such expressions, where the internal vertices represents operations and leaves represents variables or numbers.

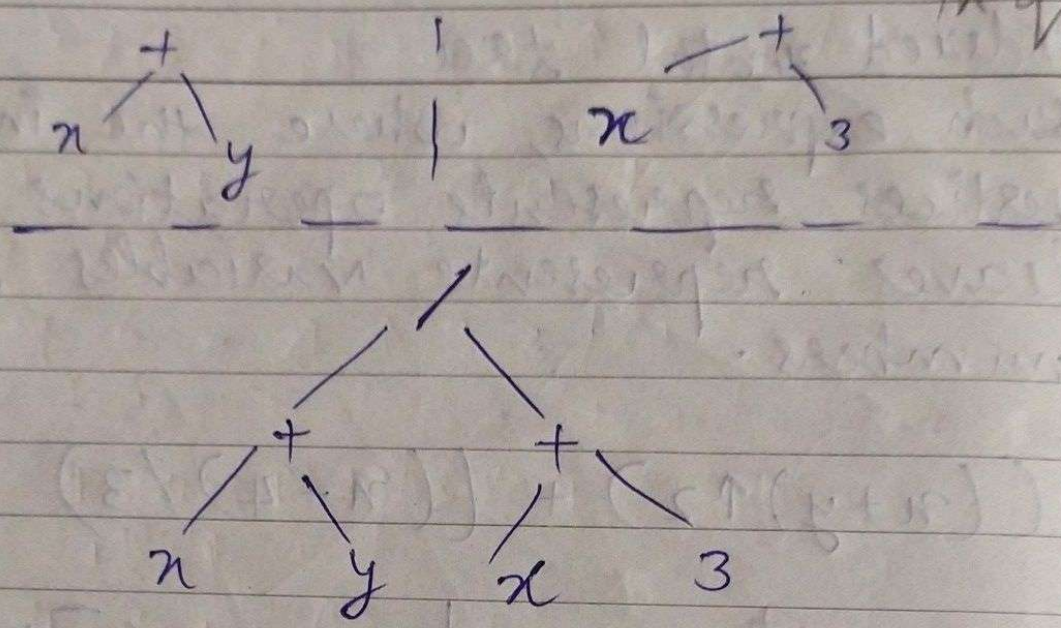
eg:- $((x+y)\uparrow 2) + ((x-4)/3)$



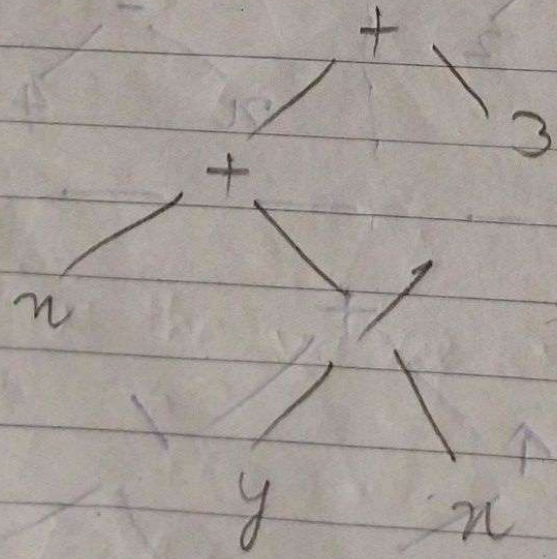
postfix / post order → $x\ y\ +\ 2\ \uparrow\ x\ 4\ -\ 3\ /\ +$

prefix / preorder → $+\ \uparrow\ x\ y\ 2\ /\ -\ x\ 4\ 3$

eg:- $(x+y) / (x+3)$ infix - usually given in this form in question



eg:- $(x + (y/x)) + 3$



right side

Q:- What is the value of the prefix expression

→ + - * 235 / ↑ 234 ?

→ + - * 235 / ↑ 2 3 4

2 ↑ 3 = 2^3 = 8

→ + - * 235 / 8 4

8/4 = 2

→ + - * 2352

* 2*3

→ + - 2*3 52

→ + - 6 5 2

6 - 5 = 2

→ + 1 2

1 + 2

→ = 3

traverse from left

Ques What is the value of the postfix expression?

→ 7 2 3 * - 4 ↑ 9 3 / +

→ 7 2*3 - 4 ↑ 9 3 / +

→ 7 6 - 4 ↑ 9 3 / +

7-6

→ 1 4 ↑ 9 3 / +

$1^4 = 1$

→ 1 9 3 / +

9/3 = 3

→ 1 3 +

1+3

→ 1+3 = 4

HW

Ques What is the value of the prefix expression?

(a) - * 2 / 8 4 3

(b) ↑ - * 3 3 * 4 2 5

(a) $- * 2 / 8 4 3$

$\rightarrow - * 2 / 8 4 3$

$8/4 = 2$

$\rightarrow - * 2 2 3$

$2 * 2 = 4$

$\rightarrow - 4 3$

$\rightarrow 4 - 3 = 1$

(b) $\uparrow - * 3 3 * 4 2 5$

$\rightarrow \uparrow - * 3 3 * 4 2 5$

$4 * 2 = 8$

$\rightarrow \uparrow - * 3 3 8 5$

$3 * 3 = 9$

$\rightarrow \uparrow - 9 8 5$

$9 - 8 = 1$

$\rightarrow \uparrow 1 5$

$1 \uparrow 5 = 1^5 = 1$

$\rightarrow 1$

HW

Ques What is the value of the postfix expression?

(a) 521 - - 314 + + *

(b) 93/5 + 72 - *

(a) 521 - - 314 + + *

2 - 1 = 1

→ 51 - 314 + + *

5 - 1 = 4

→ 4314 + + *

1 + 4 = 5

→ 435 + *

3 + 5 = 8

→ 48 *

→ 4 * 8 = 32

→ 32

=

(b) 93/5 + 72 - *

9/3 = 3

→ 35 + 72 - *

3 + 5 = 8

→ 8 7 2 − *

7 − 2 = 5

→ 8 5 *

→ 8 * 5 = 40

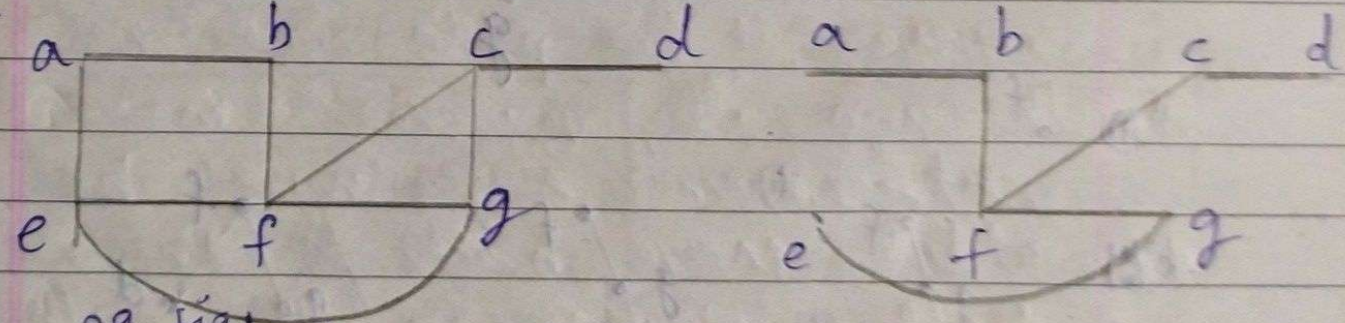
→ 40

4 June 2020

SPANNING TREES

Let G be a simple graph, a spanning tree of G is a subgraph of G, that is a tree containing every vertex of G

SPANNING TREE
- no circuits
- all vertices must be present



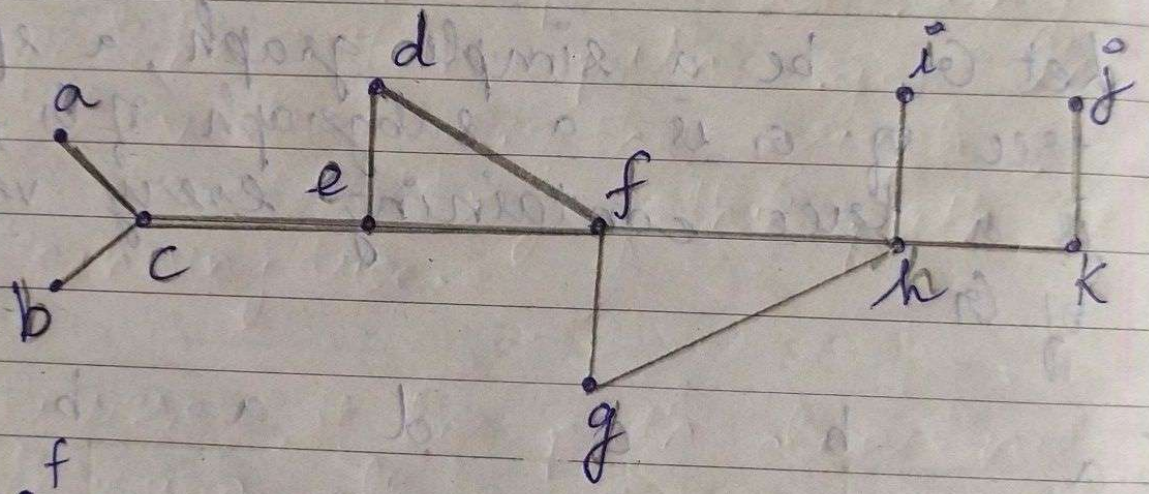
eg fig1-

Theorem

A simple graph is connected if and only if it has a spanning tree.

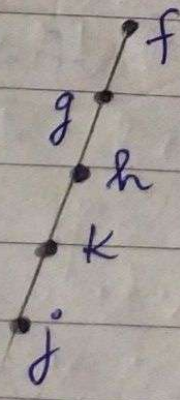
example of Depth-First Search

eg:-

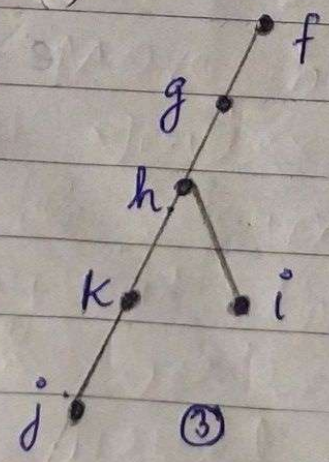


(1) f

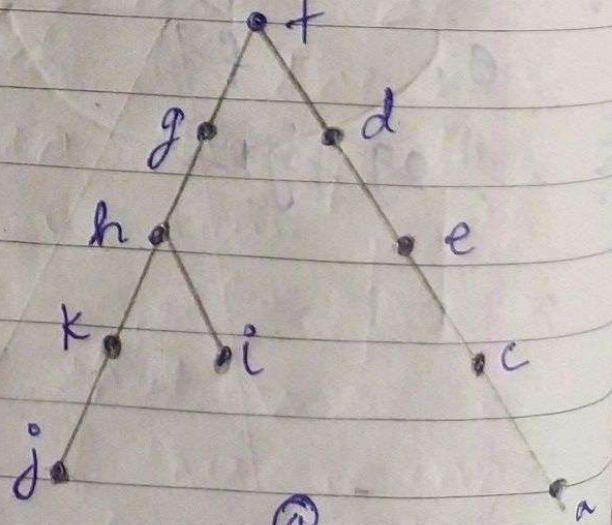
(2)



(i)



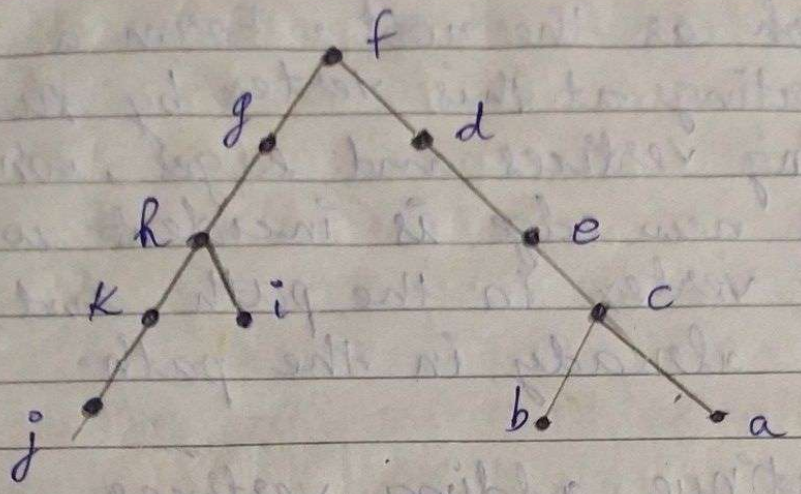
(3)



(4)

(6)

(5)



* The proof of theorem 1 gives an algorithm for finding spanning trees by removing edges from simple circuits

Depth-First Search / Backtracking

We can build a spanning tree for a connected simple graph using depth-first search. We will form a rooted tree, and the spanning tree will be the underlying undirected graph of this rooted tree.

Arbitrarily choose a vertex of the

graph as the root. Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.

Continue adding vertices and edges to this path as long as possible. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.

However, if the path does not go through all vertices, more vertices and edges must be added. Move back to the next to last vertex in the path, and if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another

vertex in the path, that is, two vertices back in the path and try again.

* Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new paths that are as long as possible until no more edges can be added.

Because the graph has a finite number of edges and is connected, this process ends with the production of a spanning tree. Each vertex that ends a path at a stage of the algorithm will be a leaf in the rooted tree, and each vertex where a path is constructed starting at this vertex will be an internal vertex.

Depth-first search is also called back-tracking, because the algorithm returns to vertices previously visited to add paths.

Breadth-First Search

We can also produce a spanning tree of a simple graph by the use of breadth-first search.
Again, a rooted tree will be ~~cons~~ constructed, and the underlying undirected graph of this rooted tree forms the spanning tree.

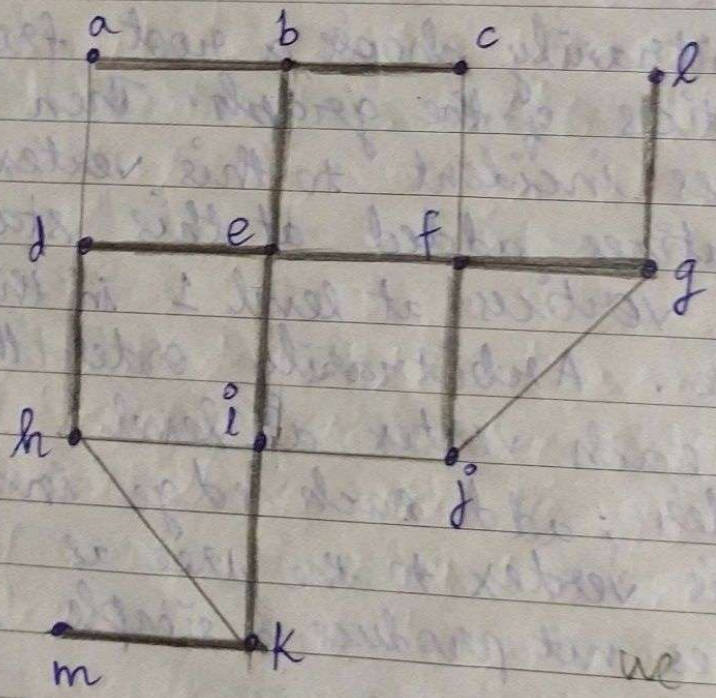
Arbitrarily choose a root from the vertices of the graph. Then add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit.

Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree. Follow

the same procedure until all the vertices in the tree have been added. The procedure ends because there are only a finite number of edges in the graph.

A spanning tree is produced because we have produced a tree containing every vertex of the graph.

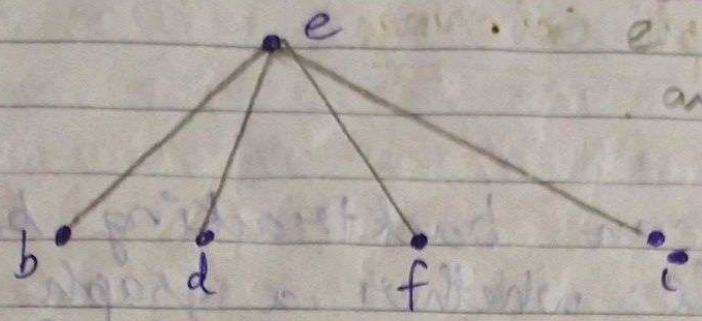
eg:-



① 

we choose e to be the root. Then we add edges incident with all vertices adjacent to e.

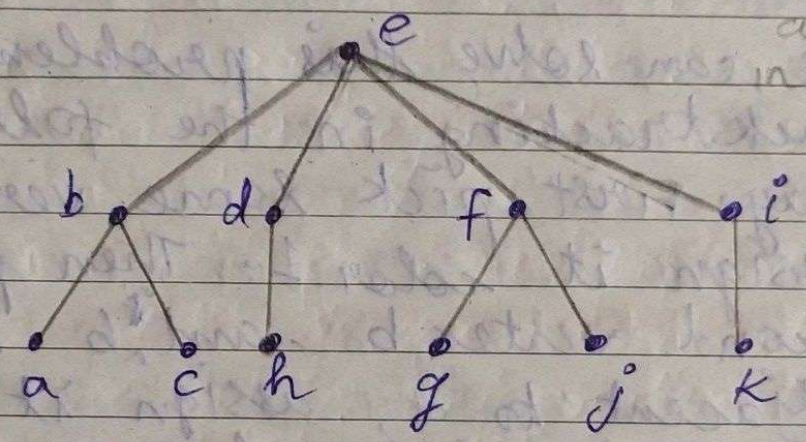
(2)



So edges from e to b, d, f and i are added.

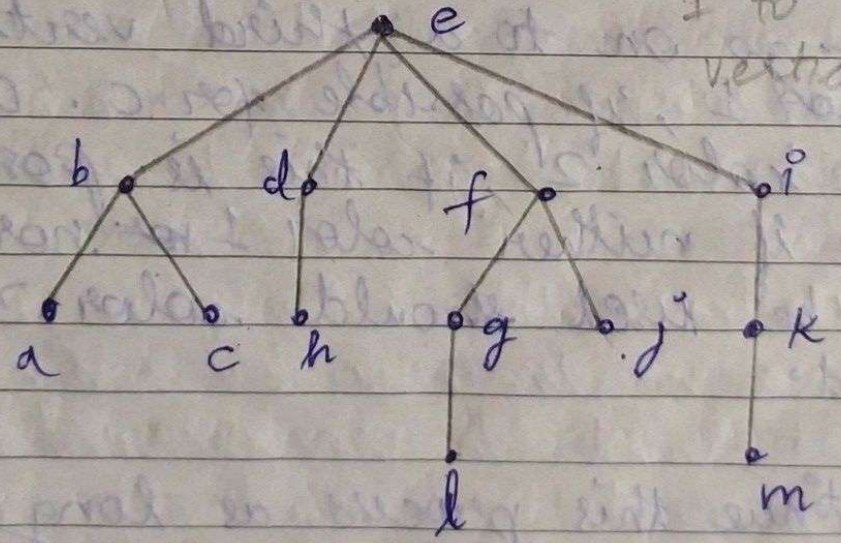
These 4 vertices are at level 1 in the tree.

(3)



Next, add the edges from these vertices at level 1 to adjacent vertices not already in the tree

(4)



Backtracking Applications

① Graph Colorings

⇒ How can backtracking be used to decide whether a graph can be coloured using n colors?

Sol. We can solve this problem by backtracking in the following way. First pick some vertex a and assign it color 1. Then pick a second vertex b and, if b is not adjacent to a, assign it color 1. Otherwise assign color 2 to b.

Then go on to a third vertex c. Use color 1, if possible for c. Otherwise use color 2, if this is possible. Only if neither color 1 nor color 2 can be used should color 3 be used.

Continue this process as long as it is possible to assign one of the n

colors to each additional vertex, always using the first allowable color in the list.

If a vertex is reached that cannot be colored by any of the n colors, backtrack to the last assignment made and change the coloring of the last vertex colored, if possible, using the next allowable color in the list.

If it is not possible to change this coloring, backtrack farther to previous assignments, one step back at a time, until it is possible to change a coloring of a vertex. Then continue assigning colors of additional vertices as long as possible.

If a coloring using n colors exists, backtracking will produce it. (unfortunately this procedure can be extremely efficient.)

In particular, consider the problem of coloring the graph shown in fig with three colors.
The tree shown in fig illustrates how backtracking can be used to construct a 3-coloring. In this procedure, red is used first, then blue, and finally green. This simple example can obviously be done without backtracking, but this is a good illustration of the technique.

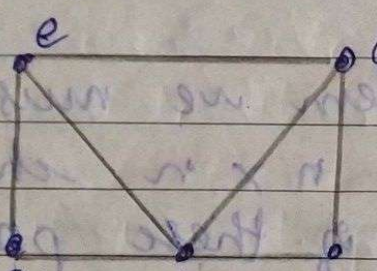
In this tree, the initial path from the root, which represents the assignment of red to a, leads to a coloring with a red, b blue, c red and d green.

It is impossible to color e using any of the three colors when a, b and c and d are colored in this way.

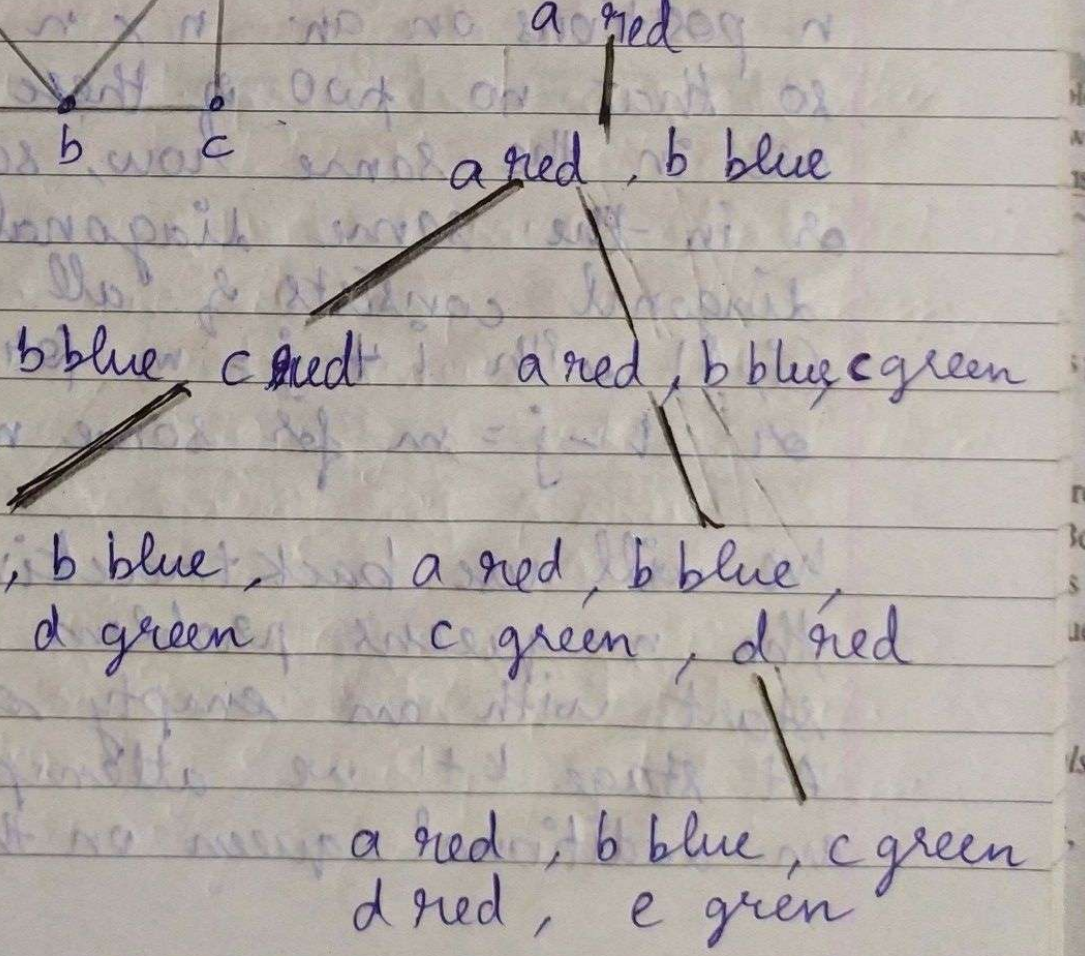
coloring a graph using

So, backtrack to the parent of the vertex representing this coloring. Because no other color can be used for d, backtrack one more level. Then change the color of c to green. We obtain a coloring of the graph by then assigning red to d and green to e.

Fig:-



Coloring a graph using Back-tracking



② The n-Queens Problem

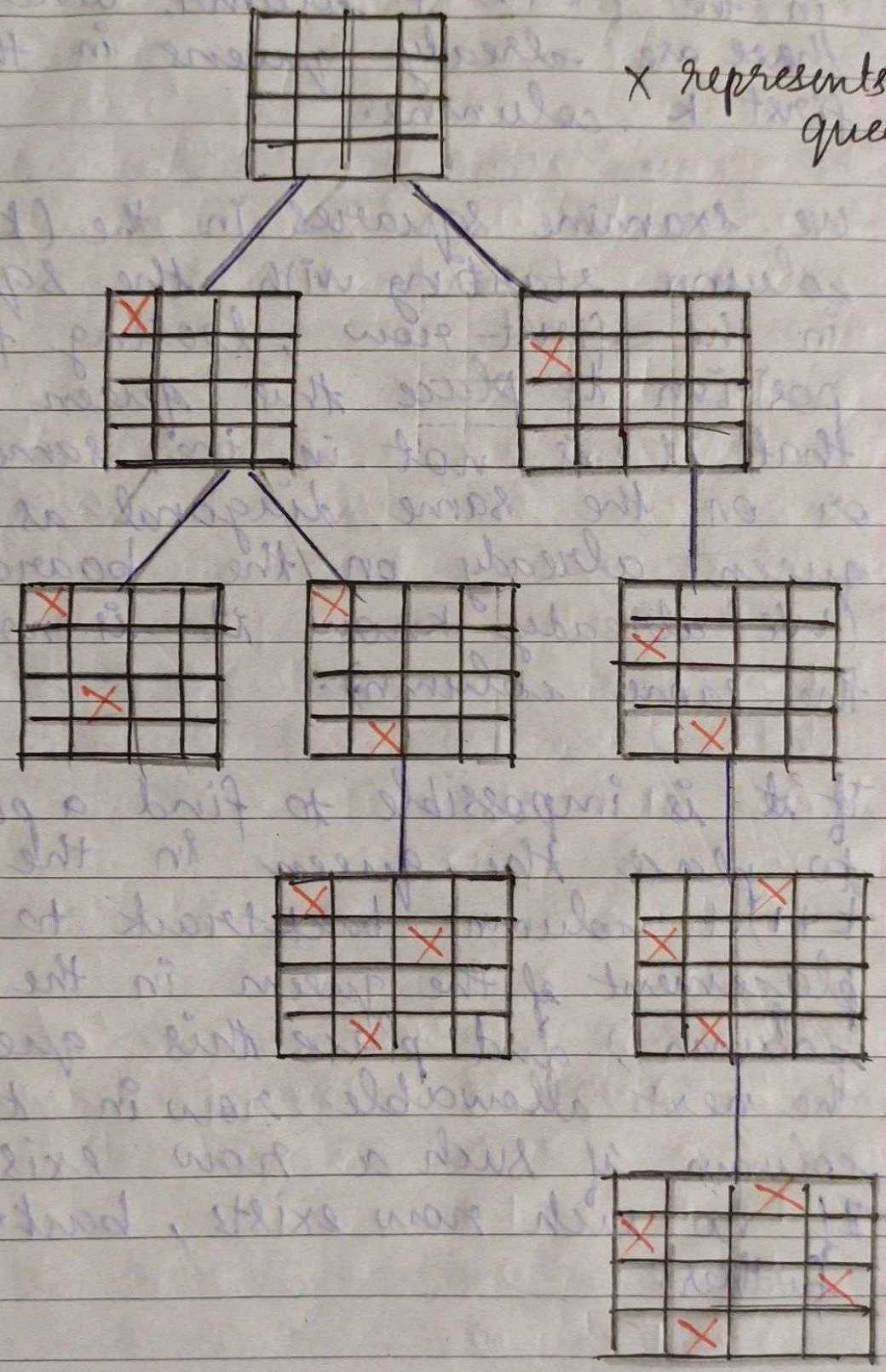
→ The n-queens problem asks how n queens can be placed on an n x n chessboard so that no two queens can attack one another. How can backtracking be used to solve the n-queens problem?

Sol. To solve this problem we must find n positions on an n x n chessboard so that no two of these positions are in the same row, same column, or in the same diagonal [a diagonal consists of all positions (i, j) with i + j = m for some m, or i - j = m for some m]

We will use backtracking to solve the n-queens problem. We start with an empty chessboard. At stage k+1 we attempt putting an additional queen on the board

Backtracking Solution of the Four-Queens Problem

fig:-



x represents a queen

in the (k+1) st column, where there are already queens in the first k columns.

We examine squares in the (k+1)st column starting with the square in the first row, looking for a position to place this queen so that it is not in the same row or on the same diagonal as a queen already on the board. (We already know it is not in the same column):

If it is impossible to find a position to place the queen in the (k+1)st column, backtrack to the placement of the queen in the kth column, and place this queen in the next allowable row in this column, if such a row exists. If no such row exists, backtrack further.