4. Backtracking

Back Tracking:

* Back Tracking is a procedure to implement some task recursively with the help of brute force approach which ultimately gives us enumerated sol^n.
* In back tracking, user always need to perform navigations from current level to previous levels, based on the requirement.

Applications of back tracking:

The following applications can be implemented using back tracking procedure:

- Hamiltanion cycle
- Sum of subset problems
- Knapsack problem
- n-Queens problem

Hamiltanion cycle:

Procedure:

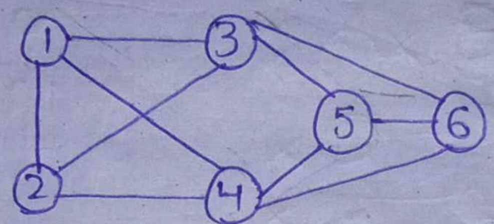
Step-1: Identify no. of vertices & edges in the given graph G.

Step-2: Choose any one of the vertices as the source & traverse every other vertex only once & come back to the source vertex.

Step-3: Identify 'n' no. of sol^ns with the help of state space tree, and pick the final sol^n based on the requirement

$

Eg: Identify no. of Hamiltonian cycles for the given graph G.

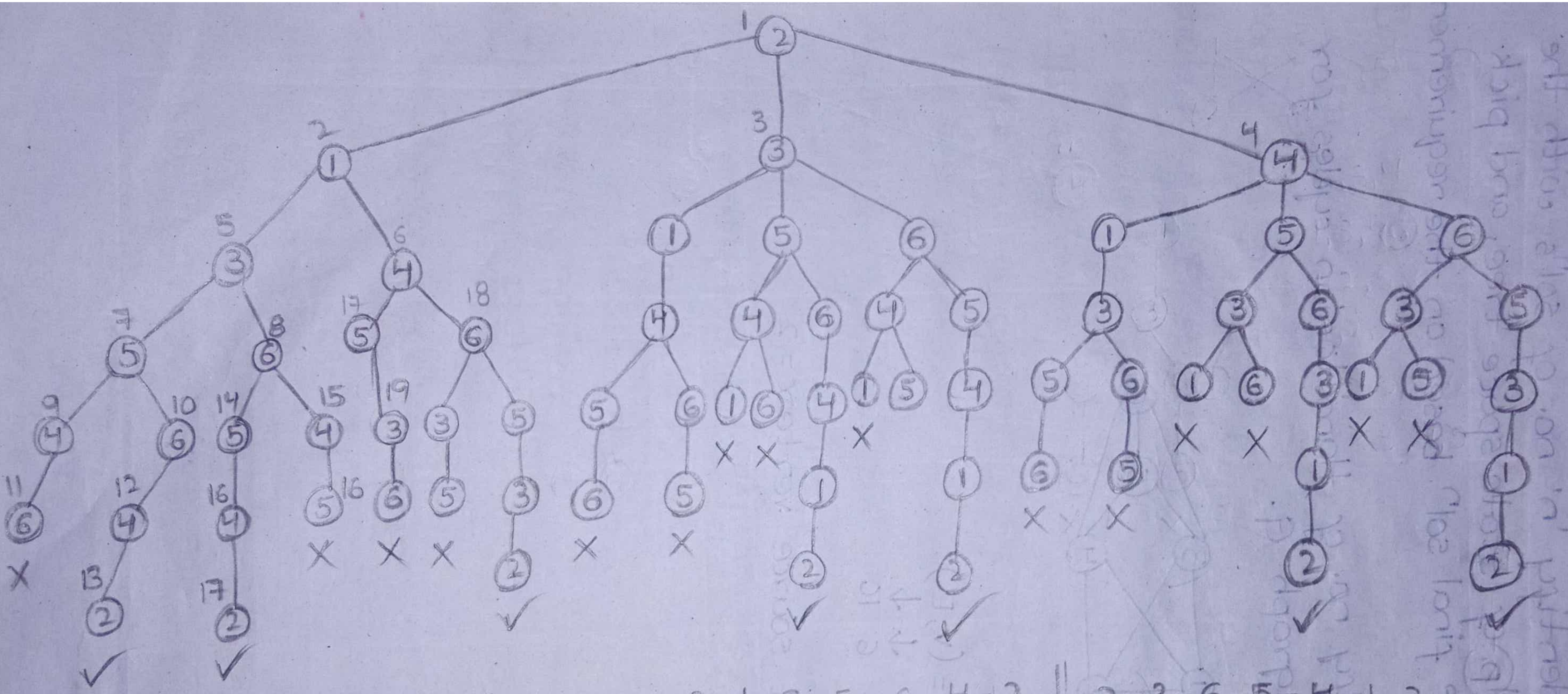


Sol:

Step-1: $G = (V, E)$

$V \to 6$, $E \to 10$

Step-2: Let source vertex = 2



∴ Possible cycles are -

2-1-3-5-6-4-2
2-1-3-6-5-4-2
2-1-4-6-5-3-2
2-3-5-6-4-1-2
2-3-6-5-4-1-2
2-4-5-6-3-1-2
2-4-6-5-3-1-2

Sum of subset problem:

Step-1: Consider given 'n' no. of elements in the set S_1.

Step-2: Identify the summation 'd' by considering 'm' no. of combinations from the given 'n' no. of elements.

Step-3: Use backtracking if summation value of 'n' no. of elements in the given set exceeds.

Step-4: Identify all possible solutions with the help of State Space Tree.

Eg: Identify all possible solutions for the given sum of subset problem. Set = $\{3,8,9,12,4\}$ and $d=12$ where d stands for summation value.

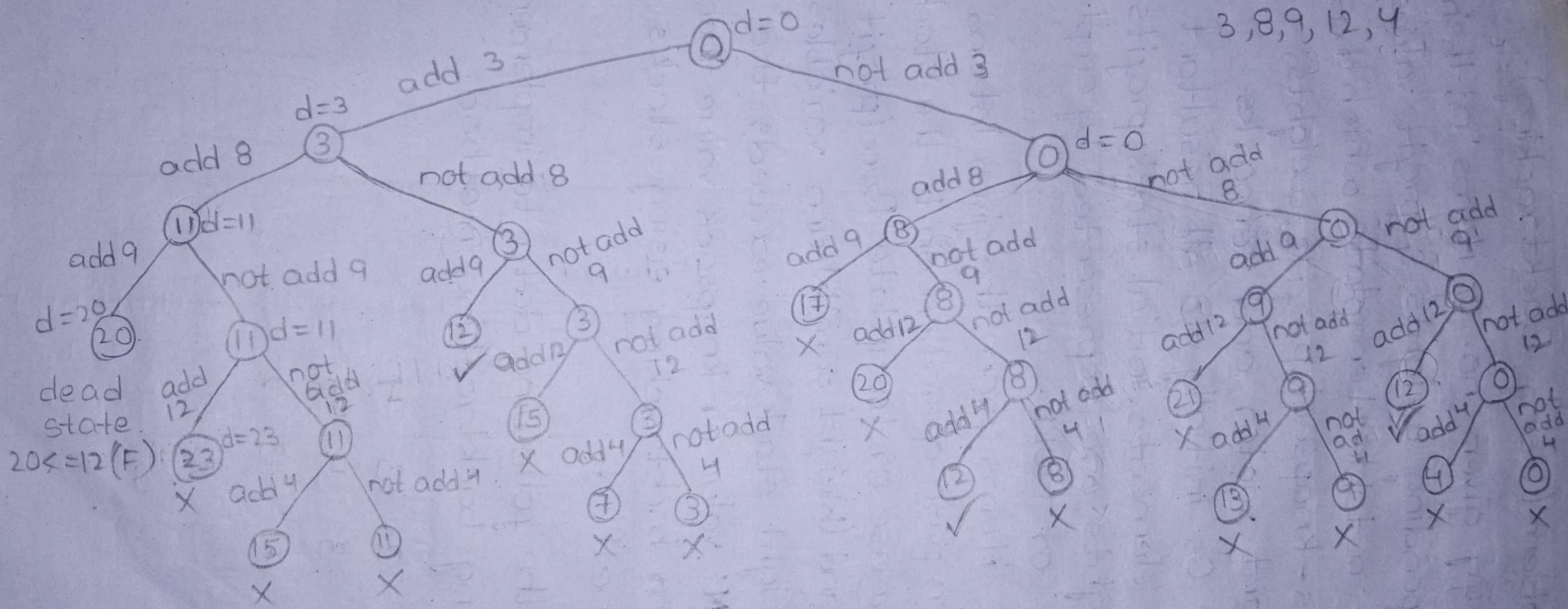
Step-1: No. of elements in the given set

$S_1 = n = 5$.

Initial value of $d = 0$.

Step-2: Add elements to an empty set & calculate recursively the summation value, if summation is greater than the given d value then use backtracking to remove recently added elements & repeat the procedure till we reach to given 'd' value.

Action	Set	d	Condition
	Empty	0	
$n_1 = 3$, add to S_1	$\{3\}$	$d = 0+3 = 3$	$3 <= 12$ (T) So, recursive call
$n_2 = 8$, add to S_1	$\{3,8\}$	$d = 3+8 = 11$	$11 <= 12$ (T) So recursive call.
$n_3 = 9$. add to S_1	$\{3,8,9\}$	$d = 11+9 = 20$	$20 <= 12$ (F) Back track
Remove 9	$\{3,8\}$	$d = 11$	$11 <= 12$ (T) Recursive call
$n_4 = 12$ Add to S_1	$\{3,8,12\}$	$d = 11+12 = 23$	$23 <= 12$ (F) Backtrack
Remove 12	$\{3,8\}$	$d = 11$	$11 <= 12$ (T) Recursive call
$n_5 = 4$ Add to S_1	$\{3,8,4\}$	$d = 11+4 = 15$	$15 <= 12$ (F) Backtrack
Remove 4	$\{3,8\}$	$d = 11$	$11 <= 12$ (T)
Remove 8	$\{3\}$	$d = 3$	
$n_3 = 9$ Add to S_1	$\{3,9\}$	$d = 3+9 = 12$	$12 <= 12$ (T)



3,8,9,12,4
d=0
add 3
not add 3
d=3
d=0
add 8
not add 8
add 8
not add 8
d=11
add 9
not add 9
add 9
not add 9
add 9
not add 9
add 9
not add 9
d=20
d=11
dead state
20<=12 (F)
add 12
not add 12
add 12
not add 12
add 12
not add 12
add 12
not add 12
add 12
not add 12
d=23
add 4
not add 4
add 4
not add 4
add 4
not add 4
add 4
not add 4
add 4
not add 4

Knapsack problem (Using Backtracking):

Step-1: Identify given no. of items 'n', their corresponding weights & profit values.

Step-2: Create a table template which has columns - knapsack, profits, weights, resultant vector and remarks.

Step-3: Assign 0s initially to all resultant vector elements, initial stack, profits & weights are represented by empty values or NULL.

Step-4: Select items from the given list and perform push operation on to the stack to increase or to update Psum (profit sum) and (Weight sum) Wsum variables.

Step-5: If weight stack value exceeds the given 'M' value or knapsack value then perform backtracking using stack's pop method.

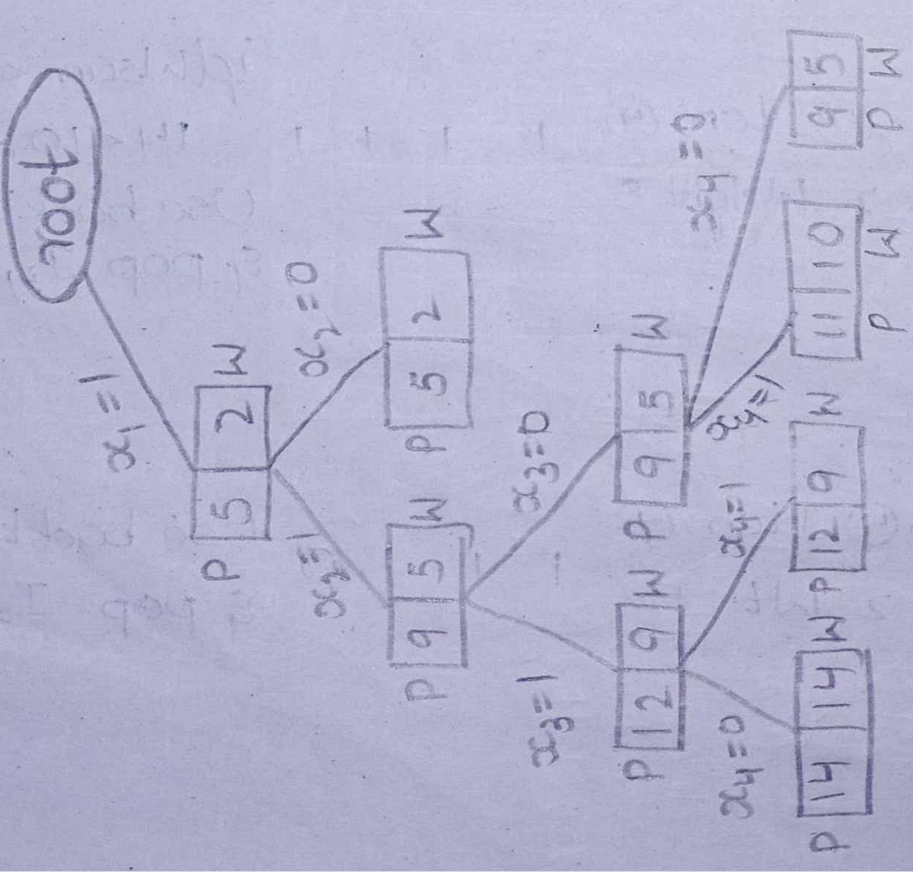
Eg: Consider the following data to implement knapsack using backtracking method.

Item:	1	2	3	4
Weight:	2	3	4	5
Profit:	5	4	3	2

$M = 10$.

Knapsack	Psum	Wsum	Resultant Vector	Remarks
Pr: [ ] Wt: [ ] Psum=0 Wsum=0			0 0 0 0	if(Wsum<M) 0<10 ✓ select item ① Push(P_1, W_1)
Pr: 5 Wt: 2 Psum=5 Wsum=2	Item ① Profit 5	Item ① Weight 2	1 0 0 0	if(Wsum<M) 2<10 ✓ select item ② Push(P_2, W_2)
Pr: 4, 5 Wt: 3, 2 Psum = 9 Wsum = 5	Item ② Profit 4	Item ② Weight 3	1 1 0 0	if(Wsum<M) 5<10 ✓ select item ③ Push(P_3, W_3)
Pr: 3, 4, 5 Wt: 4, 3, 2 Psum = 12 Wsum = 9	Item ③ Profit 3	Item ③ Weight 4	1 1 1 0	if(Wsum<M) 9<10 ✓ select item ④ Push(P_4, W_4)
Pr: 2, 3, 4, 5 Wt: 5, 4, 3, 2 Psum = 14 Wsum = 14	Item ④ Profit 2	Item ④ Weight 5	1 1 1 1	if(Wsum<M) 14<10 ✗ Use backtrack & pop I_4
Pr: 3, 4, 5 Wt: 4, 3, 2 Psum = 12 Wsum = 9	Item ④ Profit 2	Item ④ Wt. 5	1 1 1 0	Use backtrack & pop I_3

4, 5 | 3, 2 Pr Wt Psum = 9 Wsum = 5	Item ③ Profit 3	Item ③ Weight 4	1 1 0 0	if(Wsum<M) 5<10 ✓ Select I_4 Push(P_4, W_4)
2, 4, 5 | 5, 3, 2 Pr Wt Psum = 11 Wsum = 10	Item ④ Profit 2	Item ④ Weight 5	1 1 0 1	Wsum = M 10 = 10

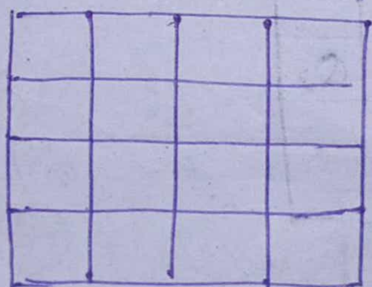


n-Queen's Problem using Backtracking:

* n-Queen's problem is an application of backtracking which uses chess board of size $N \times N$.
* We have 4-queens and 8-queens in consideration to implement n-Queens problem.
* We need to place 'n' no. of queens on the square chess board by following the below conditions-
  (i) No 2 queens to be placed on the same row.
  (ii) No 2 queens to be placed on the same column.
  (iii) No 2 queens to be placed on the same diagonal.

4-Queens problem:

In 4-queens problem, we have a set Q containing $Q = \{Q_1, Q_2, Q_3, Q_4\}$



$$n = 4$$

$$Q = \{Q_1, Q_2, Q_3, Q_4\}$$

$$N \times N = 4 \times 4 = 16.$$

Positions $= 16 \Rightarrow P_1, P_2, P_3 - - - P_{16}$

(i) Place Q_1 in the position P_1.

Q_1			

Now Q_2 can be placed in P_7 or P_8.

(ii) Place Q_2 in the position P_7.

Q_1			
		Q_2	

No place for Q_3.

So, backtrack and remove Q_2.

(iii) Place Q_2 in the position P_8.

Q_1			
			Q_2

(iv) Place Q_3 in the position P_{10}

Q_1			
			Q_2
	Q_3		

No place for Q_4

So, back-track and remove Q_3.

(v) But shifting Q_3 is not possible.

So, again backtrack and remove Q_2

(vi) But shifting Q_2 is also not possible.

So, backtrack and shift Q_1.

(vii) $Q_1 \longrightarrow P_2$

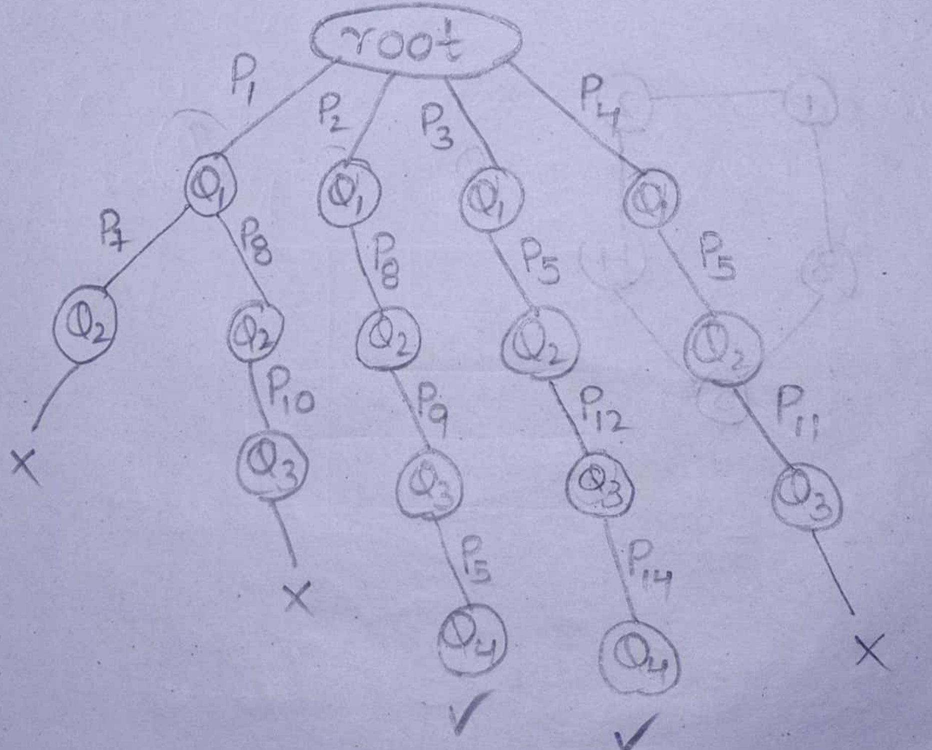
	Q_1		

(viii) Place Q_2 in P_8, Q_3 in P_9, Q_4 in P_{15}

	Q_1		
			Q_2
Q_3			
		Q_4	

$$\left.\begin{array}{l} Q \longrightarrow P_2 \\ Q_2 \longrightarrow P_8 \\ Q_3 \longrightarrow P_9 \\ Q_4 \longrightarrow P_{15} \end{array}\right\} \text{final solution.}$$

(ix) <u>State</u> <u>Space</u> <u>Tree</u>:



∴ The no. of possible solⁿs = 2.

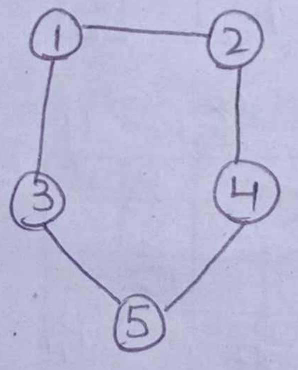
Graph Coloring Problem:

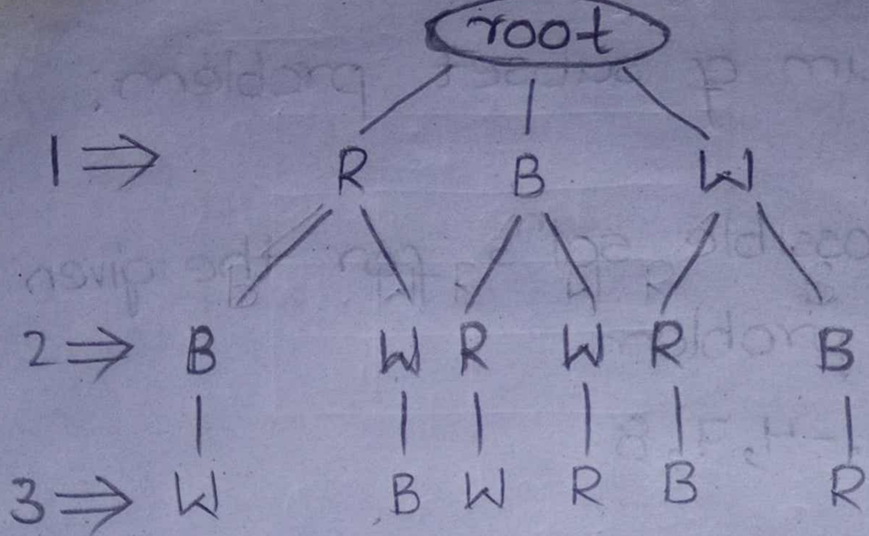
* Graph coloring problem is an application of backtracking method.
* Identify chromatic number (minimum) of colors N to color or identify adjacent vertices by satisfying the condition that "no two adjacent vertices should contain same color."

Applications:

* To implement clustering position.
* To implement adhoc networks.
* Device identification in a network.

Eg: Consider the following graph and identify chromatic no. of colors to color the given graph.




root
1 ⇒
R B W
2 ⇒
B W R W R B
3 ⇒
W B W R B R