Unit-5

Backtracking

→ Backtracking is used to find the optimal solutions. Greedy & Dynamic Programming also used to find the optimal solution using BruteForce Approach. (Finding the no. of possibilities among all the possibilities we can select the best solutions)

→ Backtracking can be used to finding the optimal solutions with less no. of steps. So it will reduce the time complexity & improve the Efficiency. And can be solved by using two Constraints

1) Implicit Constraint
2) Explicit Constraint

(1) Implicit Constraint: we can apply the conditions within the boundary. Ex Chess Board problem

(2) Explicit Constraint : we can apply the conditions outside the boundaries. Ex Chess Board problem

→ Applications of BT

1, N-Queens problem
2, Graph Coloring problem
3, Hamiltonian Cycle
4, Sum of subsets.

- Technical terms

1, problem state
2, solution state
3, Answer state
4, Live Node
5, Dead Node
6, state space Tree

N-Queens problem

→ We can place that, N×N chessboard totally n^n possibilities are there we place that in N×N Chessboard (N Queen in)

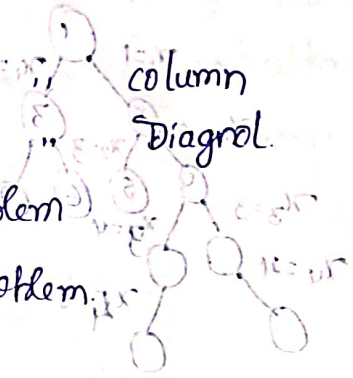
Conditions 1) no Two Queen are placing same Row

2) " " " " column

3) " " " " Diagrol

Suppose n=4 that is 4 Queens problem

∴ Suppose n=8 that is 8 Queens problem



1) 4-Queens problem

Step1 : Initially

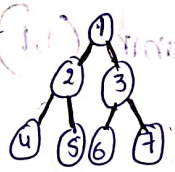
Problem state : Every node in a tree is called problem state

Ex

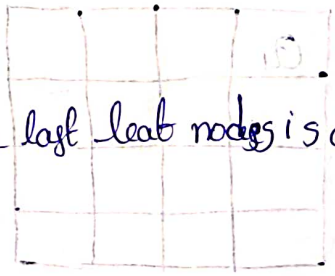
①, ②, ③ are problem states.

Solution state : The path from Root node to leaf node.

Ex

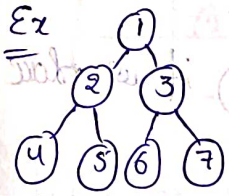


(1,2) (1,3)
(1,2,4) (1,2,5)
(1,3,6) (1,3,7)



Answer state : In a solution state the last leaf nodes is called "Answer State".

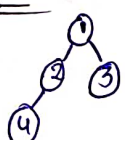
Ex



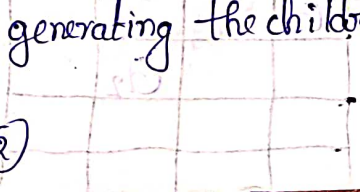
④, ⑤, ⑥, ⑦ Answer states.

Live Node : The node which is generating the children Node

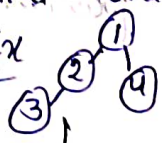
Ex



Live Nodes : ①, ②

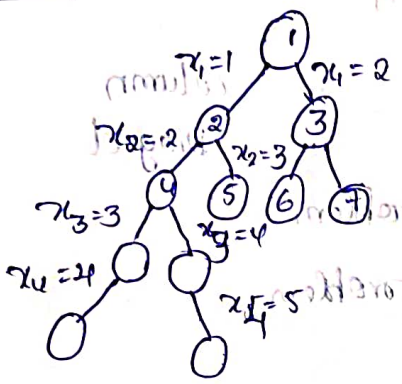


Dead Node: : The node which is not having children node that is called Dead Node.

Ex 

Dead nodes : ③, ④

State spacetree: The solutions are representing in the form of tree

Ex 

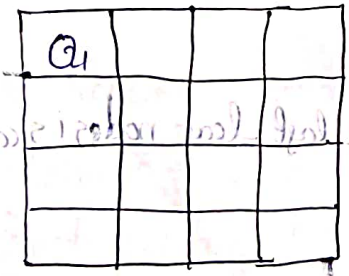
⇒ N-Queen's problem

4-Queens are placing in 4×4 chessboard 4^4 possible ways

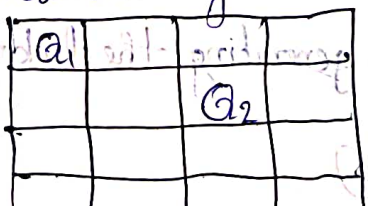
Implicit Constraint 1, No Two Queens are placing same Row
2, " " " " column
3, " " " Diagonal

Explicit Constraint in 4×4 chessboard we can place 4 Queens within the chessboard in 4^4 possible ways.

Step1 Queen is placing in the first row, first column (1,1)

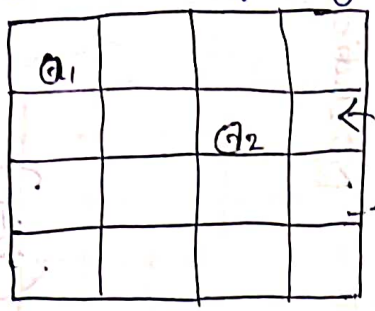


Step2 2nd queen is placing either (2,2) (2,3) (2,4) if we place (2,2) then is a diagonal attack.



So, we can place (2,3)

Step3 3rd Queen is placing in (3,2) (3,4)

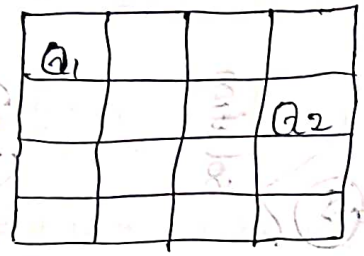


→ Back tracking. if we can place the 3rd Queen in (3,2) There is Diagonal attack

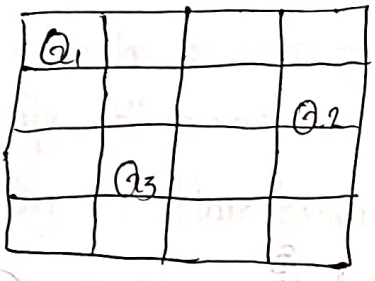
→ if we can place the 3rd Queen in (3,4). There is Diagonal attack.

∴ There is no place for 3rd Queen. BackTrack to its previous

Step4 2nd Queen is placing in (2,4)



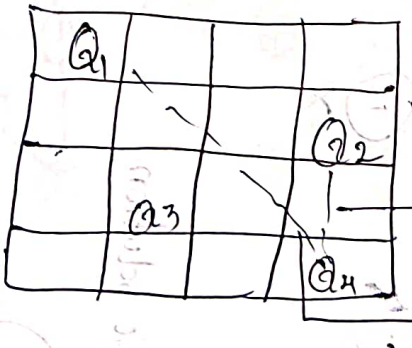
Step5 3rd Queen is placing in (3,2) (3,4)



if we can place 3rd Quee in (3,4) there is a diagonal attack

So we can place (3,2)

Step6 4th queen is placing (4,3) or (4,4).
either

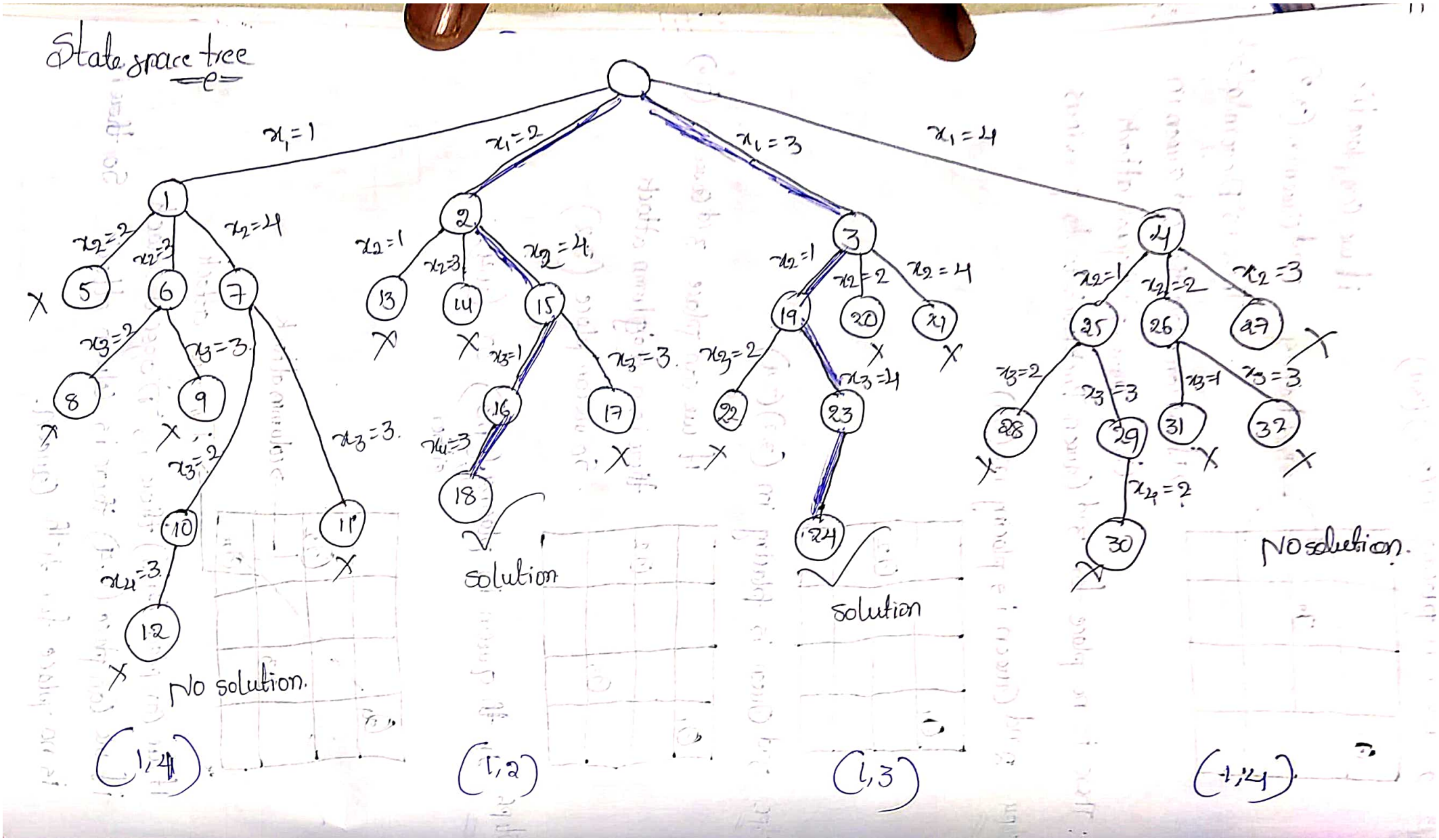


→ column attack

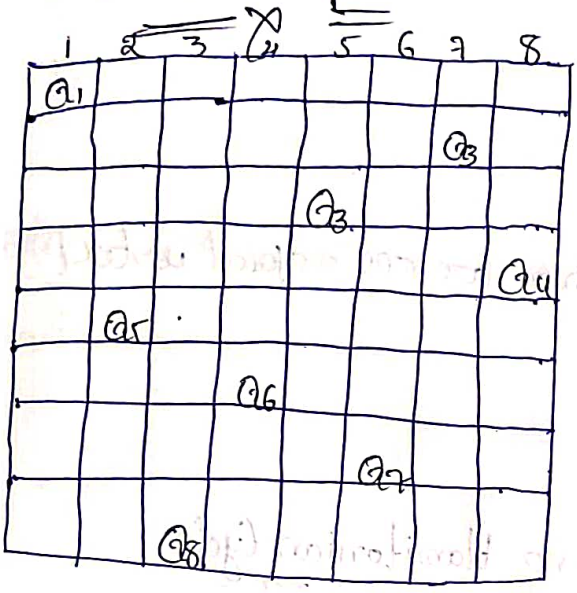
→ Diagonal attack.

If we can place (4,3) there is Diagonal attack.
if we can place (4,4) there is column attack. So there is is no place for 4th Queen.

State space tree



8-Queen problem



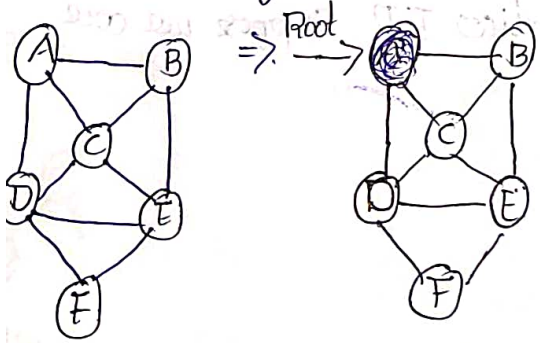
Hamiltonian Cycle

⇒ Starting at one vertex and visiting all the vertices Exactly once and return back to starting point. i.e called "Hamiltonian Cycle"

⇒ This is similar to TSP problem in TSP we can find out the one possibility with minimum cost. But here we can find out no. of possibilities in Hamiltonian cycle

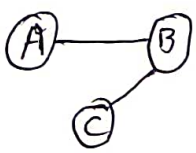
⇒ In this problem we can find out no. of possibilities.

Step1 Starting at vertex A That is the Root Node

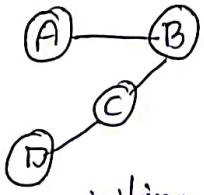


From that we are visiting All the vertices in a graph

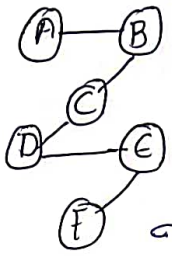
Step2 Suppose we are visiting from A → B. From vertex B. These two adjacent vertices C, E



Step3 There are two adjacent vertices for C. Here are D, E. Suppose we are visiting D vertex



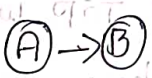
Step4 Suppose we are visiting E. There are one adjacent vertex F.



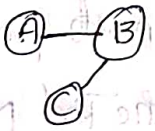
There is no Hamiltonian Cycle

Step1 Suppose we are visiting A→B from vertex A is root node from that we are we are visiting remaing vertices

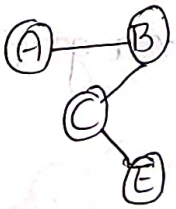
Step2 vertex having 3 adjacent vertices B, C, D. Suppose we are visiting A→B

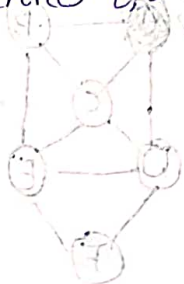


Step3 vertex B having 2 adjacent vertices. C, E. Suppose we are visiting using B→C

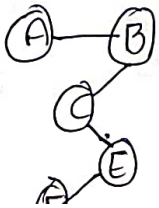


Step4 vertex C having 2adjacent vertices E, D. Suppose we are visiting C→E.

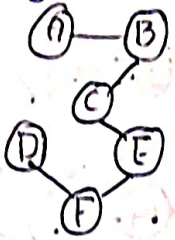




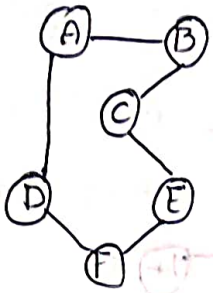
Step5 vertex E. having 2adjacent vertices D, F. Suppose we are visiting E→F



Step6 vertex F having one adjacent vertex D. visit F→D.



Step7 vertex D having 2 adjacent vertices C, A. Suppose we are visiting D - D→A.



⇒ Hamiltonian cycle

⇒ A→B→C→E→F→D→A

⇒ A→B→E→F→D→C→A A→B (2-hamiltoniancycles)

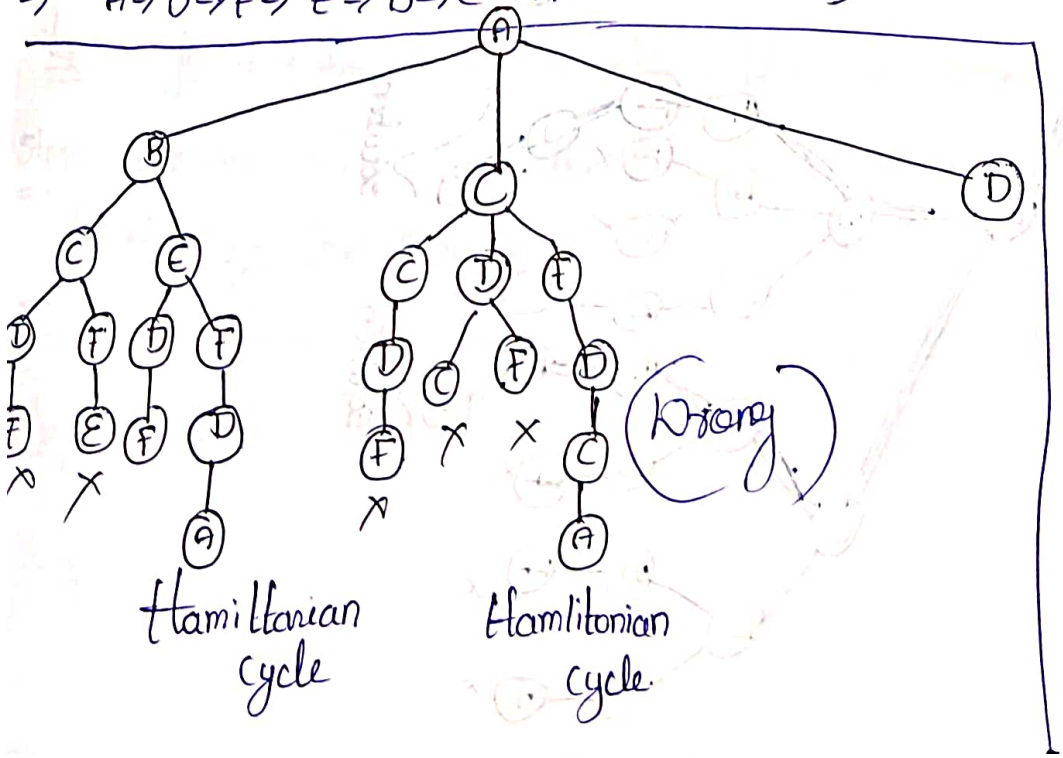
⇒ A→C→D→F→E→B→A

⇒ A→C→B→E→F→D→A A→C (2-hamiltonian cycles)

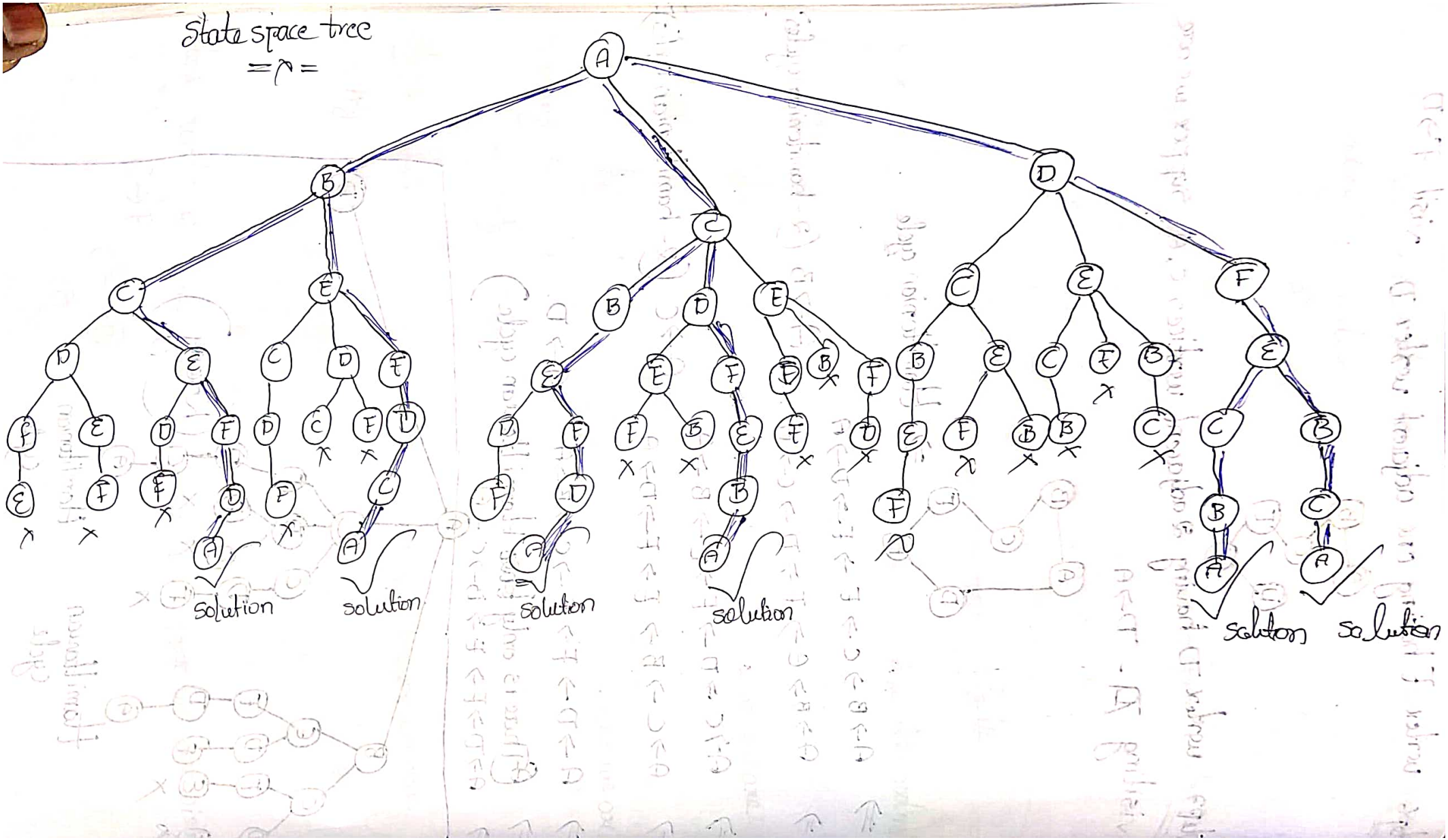
⇒ A→D→F→E→C→B→A A→D.

⇒ (There is only two Hamiltonian cycle)

⇒ A→D→F→E→B→C→A



State space tree



Sum of Subsets problem

→ The set contains the no. of elements $S = \{n_1, n_2, n_3, n_4, n_5\}$. We have to find out the all possible subsets (means feasible solutions) to get / to obtain the required sum

→ This problem is solving using back-tracking approach. The SST contains left & right sub tree.

→ The left sub tree defines adding the values

→ The right sub tree defines not adding the elements

Algorithm

Step1 initially we are not adding any element $\{0\}$ $m=5$ $S=\{5, 10, 15, 20\}$

Step2 adding the element to the subset from the given set $S=\{5\}_5$ (5)

Step3 If the subset is having the solution, then stop the subset, obtaining the solution

Step4 If it is not feasible then backtrack and then find the next feasible solution

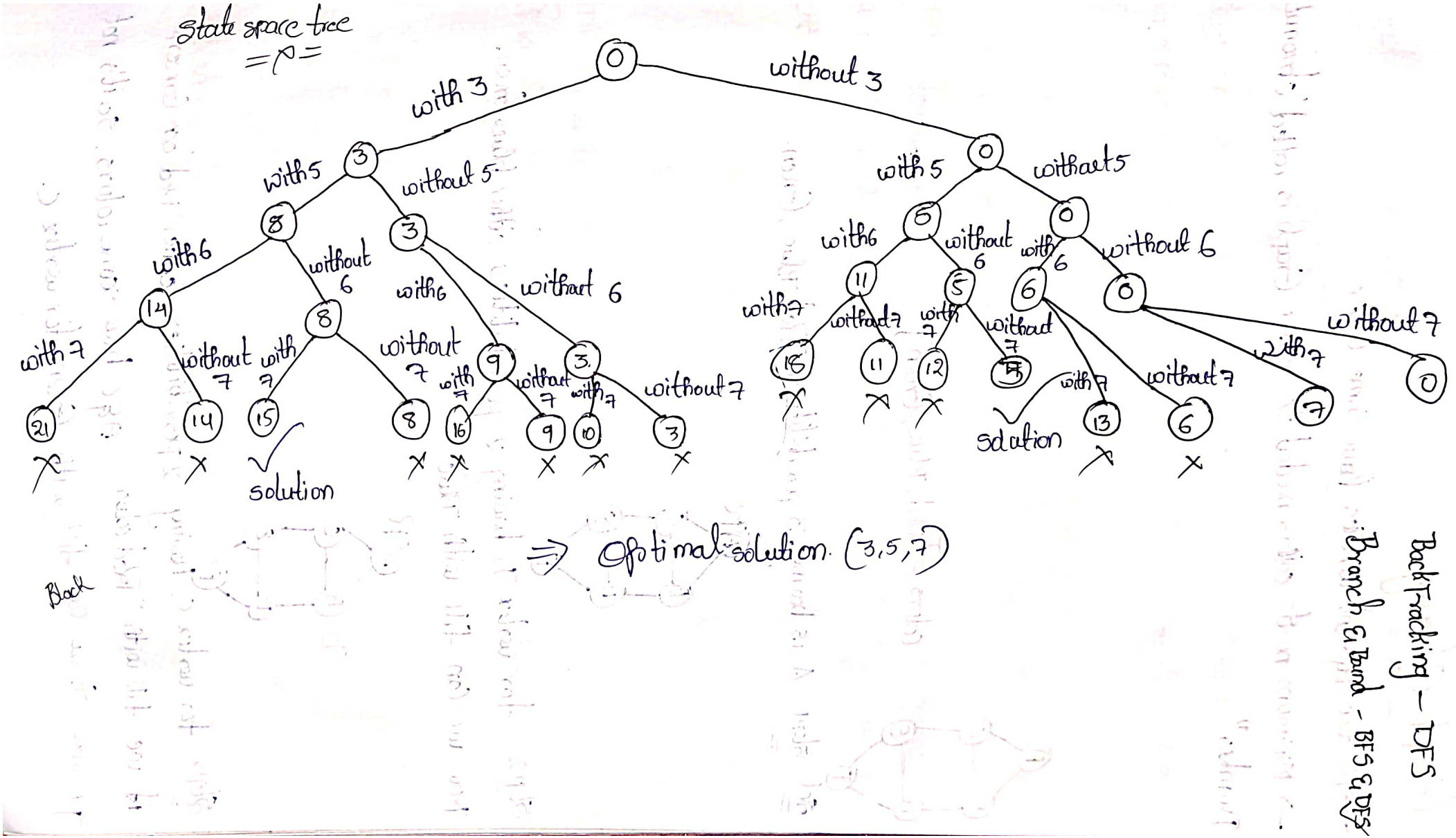
Step5 : If it is feasible solution then add the next element [step2]

Step6 we can visit all the elements without solution and there is no backtracking then there is no solution of the problem

Ex Set contains 4 Elements, $S=\{3, 5, 6, 7\}$ and $m=15$. Solve the problem using Backtracking.

Sol

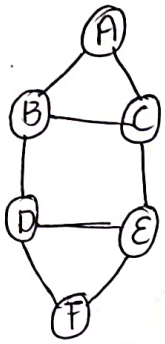
Subset	Sum	Action
{ 0	0	Initially
3	3	
3,5	3+5=8	
3,5,6	3+5+6=14	
3,5,6,7	3+5+6+7=21	Exceeded the required solution the Back-Track optimal solution
3,5,7	3+5+7=15	
3,6,7	3+6+7=16	Exceeded the Backtrack
3,7	3+7=10	
0	0	
5	5	
5,6	5+6=11	Less than required solution.
5,6,7	5+6+7=18	
5,7	5+7=12	→ Initially less than required solution
5	5	
6	6	
6,7	6+7=13	Less than then Back Track.
7	7	Less than then BT But there is no Elements


State space tree
0
with 3
without 3
3
0
with 5
without 5
with 5
without 5
8
3
5
0
with 6
without 6
with 6
without 6
with 6
without 6
14
8
3
11
5
6
0
with 7
without 7
with 7
without 7
with 7
without 7
with 7
without 7
with 7
without 7
21
14
15
8
16
9
10
3
16
11
12
15
13
6
7
0
X
X
solution
X
X
X
X
X
X
solution
X
X
Back
⇒ Optimal solution (3,5,7)
Backtracking - DFS
Branch & Bound - BFS & DFS

Graph Coloring

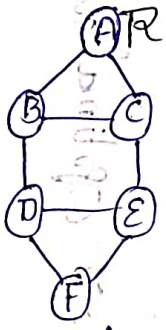
→ No two adjacent vertices have same color.

→ Minimum no. of colors needed to color a Graph is called "chromatic Number"

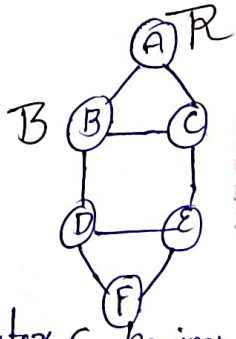


Colors = [Red, Blue, Green]

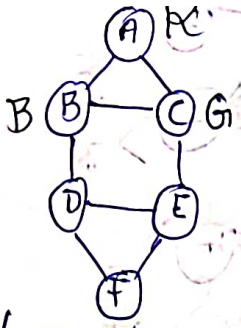
Sol Step 1 A is having 3 possibilities Red, Blue, Green.



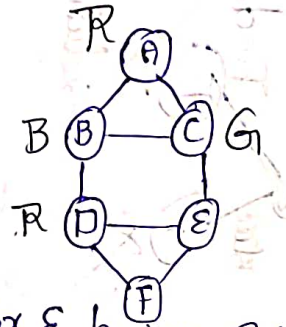
Step 2 for vertex B having 2 possibilities either blue or Green.
Now we can fill with Blue.



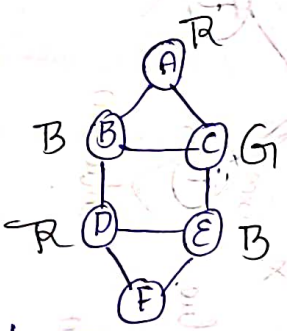
Step 3 for vertex C, having 2 possibilities either Red or Green.
if we fill with Red color, A & C have same colors. so it's not possible. So we can fill with Green for vertex C.



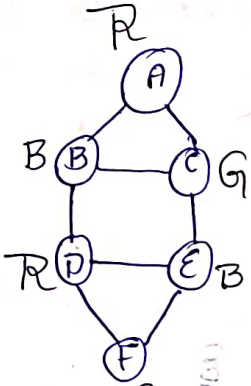
Step4 for vertex D, having 2 possibilities either Red or Blue. Supposithwe fill with Blue, it's not possible so we can fill with red



Steps for vertex E having 2 possibilities either Blue or Green. if we will fill with Green, it's not possibible so Now we can fill with Blue.

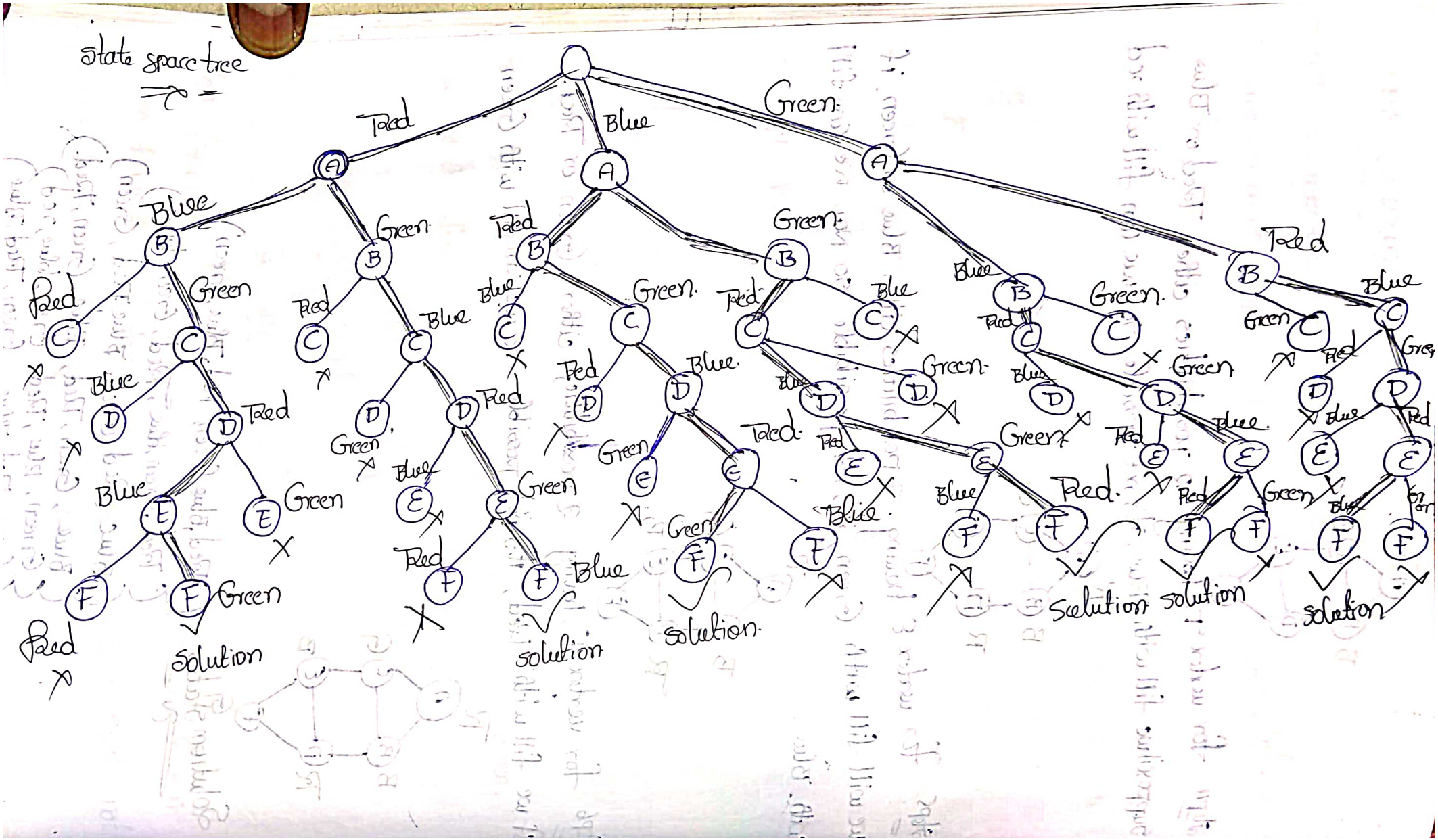


Step6 for vertex F, having 2 possibilities either Green or Red. If we fill with Red, it's not possible. So we can fill with Green



Solution States

(Red, Blue, Green, Red, Blue, Green)
(Red, Green, blue, Red, Green, blue)
(Blue, Red, Green, Blue Red, Green)
(Blue, Green, Red, Blue, Green, Red)
(Green, Blue, Red, Green, blue, red)
(Green, Red, Blue Green, Red, Blue)


State space tree
A
B
C
D
E
F
Red
Blue
Green
Solution
Solution
Solution
Solution
Solution
Solution

Branch & Bound

D/B Backtracking & Branch & Bound

Back Tracking	Branch & Bound
1. It can be solved using DFS	1. It can be solved using DFS, BFS
2. It is solved using Decission Problem	2. solved for optimization problems
3, state space tree is not completed When we get the solution	3. State space is Continued even though we will get the solution.
4. Applications Graph coloring, Hamiltonian Cycle, N-Queens problem Sum of subsets problem.	4. Applications TSP, 0/1 Knapsack problems

* Travelling Sales person problem.

TSP problem can be solving using Branch & Bound Technique Using some following steps.

1. Initially we can apply matrix Reduction
   i, row reduction
   ii, column reduction
2. Cost of root = Row Reduction + column reduction
   = 1
3. we are travelling from i to j A[i,j] then changing all the values of i and j to infinity
4. A[j,1] to ∞ (infinity)
5. Again we can apply row & column reductions
6. root of node = row + column

8. Total cost $\hat{c}[s] = \hat{c}[r] + A[i,j] + r$

where $\hat{c}[s]$ = destination cost

$\hat{c}[r]$ = cost to root node

$A[i,j]$ = we are travelling from i^{th} node to j^{th} node

r = row reduction value + column reductional value

Ex Solve the TSP problem using LCBB (Least cost Branch & Bound)

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Sol 1: Apply Matrix Reduction

Step1 Row Reduction:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \begin{matrix} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{matrix} \Rightarrow \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

$= 21$

Step2 Column Reduction

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \\ 1 & 0 & 3 & 0 & 0 \end{bmatrix} = 4 \Rightarrow \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Step3 Cost of Root r = Row Reduction + Column Reduction

$= 21 + 4$

$= 25$ ✓

(4) travelling from i to j, A[i,j] mean. 1 → 2, changes all the value of A[1,2] to ∞

$$\begin{array}{|c|cccc|}\hline \infty & 10 & 17 & 0 & 1 \\ \hline 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \\ \hline\end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

Steps A[j,1] to ∞, ⇒ A[2,1] to ∞ Steps Row & column Reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \\ 0 & 0 & 0 & 0 & \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}_{(1,2)}$$

Step $r = r + c$
$= 0 + 0$
$= 0$

Steps Total cost $\hat{c}(s) = \hat{c}(r) + A[i,j] + r$ | A[i,j] set to ∞

$\hat{c}[2] = 25 + A[1,2] + 0$
$= 25 + 20 + 0$
$= 45$

∴ Travelling from 1 → 2 the cost is 45

Step1 path (1,3) Travelling from 1-3 Then changes all the value of A(1,3) to ∞

$$\begin{array}{|c|ccc|}\hline \infty & \infty & \infty & \infty & \infty \\ \hline 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \\ \hline\end{array} \Leftarrow \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Step2 A[j,1] to ∞ ⇒ A[3,1] to ∞ Step3 Row & column reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 20 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 20 & \\ \infty & 3 & \infty & 02 & \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \begin{matrix} \\ \\ \\ \\ =11 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 20 & \\ \infty & 3 & \infty & 02 & \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

Step4 $r = r + c$

$r = 0 + 11 = 11$

Step6 Total cost $\hat{c}(3) = \hat{c}[1] + A[1,3] + 11$

$= 25 + 30 + 11$

$= 66$

The travelling from (1&3) the cost is 66

Step5 Path (1,4) Travelling from 1→4 Then changes all the values of A(1,4) to ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ 15 & 0 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Step2 A[j,1] ⇒ A(1,4) to ∞ Step3 Row & column Reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ 0 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ \\ \\ 0=0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

10

0 0 0

Step4 $r = r + c = 0 + 0 = 0$

Step5 Total cost $\hat{c}(4) = \hat{c}(1) + A[1,4] + 0$

$\hat{c}(4) = 25 + 10 + 0 = 35$

Step1 path (1, 5) Travelling from 1→5 Then changes all the values of A(1,5) to ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Step2 A[j,1] to ∞, ⇒ A(5,1) to ∞ Step3 Row & column Reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 2 \\ 0 \\ 3 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

2+3 = 5

0 0 0 0 = 0

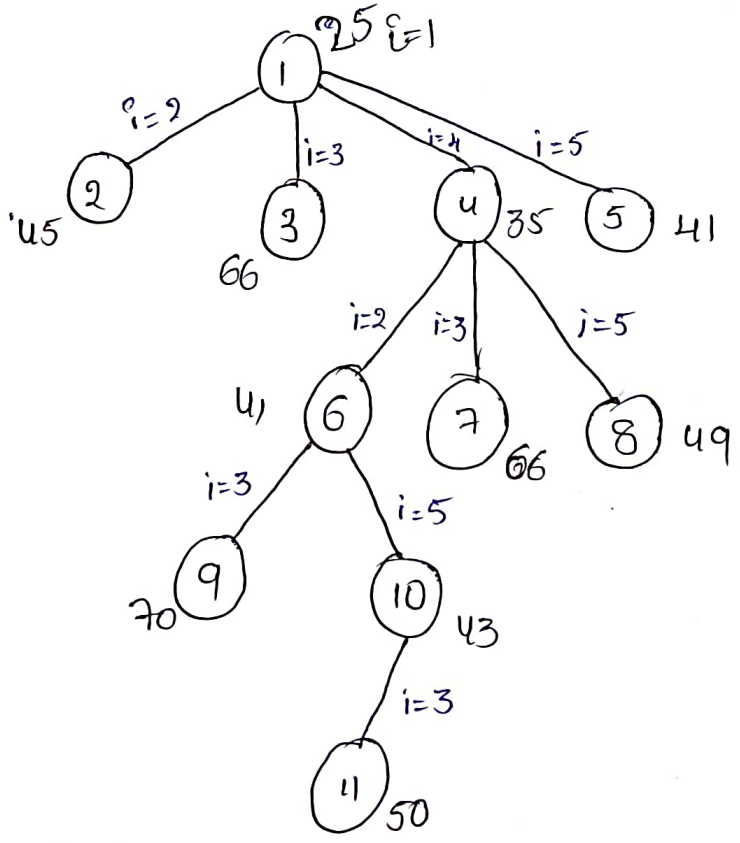
Step4 $r = r + C$

$= 5 + 0 = 5$

Step6 : Total cost $\hat{c}(5) = \hat{c}(1) + A(1,5) + 5$

$= 25 + 11 + 5$

$= 41$



Shortest path : 1 — 4 — 2 — 5 — 3 — 1

: 10 + 6 + 2 + 7 + 3 = 28 ✓

0/1 Knapsack problem

Thief wants to robbery the bank so that he will brought the empty bag that Empty bag is called knapsack.

→ Store contains 4 no. of items $\{n_1, n_2, n_3, n_4\}$ and profit of each item is $(P_1 P_2 P_3 P_4)$ the weight of each & Every item is $(w_1 w_2 w_3 w_4)$

→ In what way he hay to robbery the bank so that he will get maximum profit. This can be calculated by using Equation

$$\sum P_i x_i = P_1 x_1 + P_2 P x_2 + P_3 x_3 + P_4 x_4$$

→ Knapsack problem is maximization problem. Branch & Bound minimization problem. Inorder to solve the knapsack problem into Branch & Bound technique the profits are Converting into negative values.

→ When we are calculating K.S into B & B problem we are calculating lower bound & Upper bound for Each and Every node.

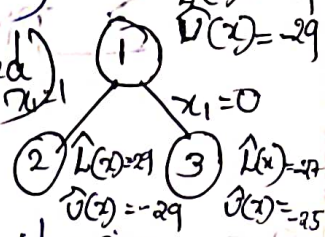
→ Lower Bound: We can place the fraction of items.

→ Upper Bound: We can't place the fraction of items

Ex: Draw the state space tree generated By LCBB. by the following Knapsack problem $n=5$, $m=12$ $(P_1 P_2 P_3 P_4 P_5) = (10, 15, 8, 6, 8, 4)$

items $(w_1 w_2 w_3 w_4 w_5) = (4, 6, 3, 4, 2)$

Sol Convert the profits into negative so that becomes minimizing problem $(P_1 P_2 P_3 P_4 P_5) = (-10, -15, -6, -8, -4)$



For node: 1 Upper bound (fractions are not allowed)

→ first we can place 1st item into the bag

Remaining Bag size = Bag size - item size

$= 12 - 4 = 8$

→ place 2nd item into the Bag.

Remaining Bag size $= 8 - 6 = 2$

→ we can't place 3rd, 4th items into the bag. Why Because fractions are not allowed.

→ place 5th item into the Bag

Remaining Bag size $= 2 - 2 = 0$

We know $x_1 = 1, x_2 = 1, x_3 = 0$

$x_4 = 0, x_5 = 1$

$$\sum P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3 + P_4 x_4 + P_5 x_5$$

$$= -10 \times 1 + -15 \times 1 + -6 \times 0 + -8 \times 0 + -4 \times 1$$

$$= -29$$

For node 1 Lower Bound (fractions are allowed)

→ first we can place 1st item into the Bag

R.B.S = Bag size - item size

$= 12 - 4 = 8$

→ place 2nd item into the Bag

R.B.S $= 8 - 6 = 2$

→ place 3rd item into the Bag, Bag size = 2, item size = 3.

$$= \frac{\text{Remaining Bag size}}{\text{item size}} = 2/3$$

→ we have $x_1=1, x_2=1, x_3=2/3, x_4=0, x_5=0$

$$\sum p_i x_i = p_1x_1 + p_2x_2 + p_3x_3 + p_4x_4 + p_5x_5$$
$$= -10\times1 - 15\times1 - 6\times\tfrac{2}{3} + -8\times0 - 4\times0$$
$$= -29$$

For node 2 $x_1=1$ means we should place 1st item into the Bag

Lower Bound → first we can place 1st item into the Bag.

$$R.B.S = \text{Bag size} - \text{item size}$$
$$= 12-4 = 8$$

→ place 2nd item into the Bag

$$R.B.S = 8-6 = 2$$

→ place 3rd item into the bag, Bag size = 2, item size = 3

$$= \frac{R.B.S}{\text{item size}} = 2/3$$

→ we have $x_1=1, x_2=1, x_3=2/3, x_4=0, x_5=0$

$$\sum p_i x_i = p_1x_1 + p_2x_2 + p_3x_3 + p_4x_4 + p_5x_5$$
$$= -10\times1 - 15\times1 - 6\times\tfrac{2}{3} - 8\times0 - 4\times0$$
$$= -10 - 15 - 4$$
$$= -29$$

Upper Bound: (fractions are not allowed)

→ first we can place 1st item into the Bag

$$R.B.S = \text{Bag size} - \text{item size}$$
$$= 12-4 = 8$$

→ place 2nd item into the Bag

$$R.B.S = 8-6 = 2$$

→ we can't place 3rd item and 4th item into the Bag · fractions are not allowed.

→ place 5th item into the Bag

R.B.S = 2-2 = 0 ✓

we have $x_1=1, x_2=1, x_3=0, x_4=0, x_5=1$

$$\sum p_i x_i = -10\times1 - 15\times1 - 6\times0 - 8\times0 - 4\times1$$
$$= -29$$

for node 3 $x_1=0$ we can't place first item into the Bag

Upper Bound (fractions are not allowed)

→ place 2nd item into the Bag

R.B.S = Bag size - item size
= 12 - 6 = 6

→ we can place 3rd item into the Bag

R.B.S = 6-3 = 3

→ we can't place 4th item Bcoz fractions are not allowed

→ Now we can place 5th item into the bag

R.B.S = 3-2 = 1

→ we have $x_2=1, x_3=1, x_4=0, x_5=1, x_1=0$ (we can't place 1st item)

$$\sum p_i x_i = -10\times0 - 15\times1 - 6\times1 - 8\times0 - 4\times1$$
$$= -25$$

Lower Bound (fractions are allowed)

→ We can place 2nd item into the Bag

R.B.S = 12-6 = 6

→ place 3rd item into the Bag = R.B.S = 6-3 = 3

→ place 4th item into the Bag = $\frac{\text{R.B.S}}{\text{item size}}$ $l = \frac{3}{4}$

→ we have $x_1=0$, $x_2=1$, $x_3=1$, $x_4=3/4$ $x_5=0$

$$\sum p_i x_i = -10\times 0 + 15\times 1 + 6\times 1 - 8\times 3/4 + 4\times 0$$

$$= -15 - 6 - 6 = -27$$

Now we can decide whether we are placing first item into the Bag or not

$$= \min\{L\hat{c}(x), U\hat{c}(x)\}$$

$$= \min\{-29, -27\}$$

$$= -29$$

We can place first item into the Bag $x_1=1$

Unit-6

String Matching

Part-1

* Naive String Matching ⟹ possible placements of pattern P[1...m] relative to text T[1...n]

Ex Text A = AABA ACAA DAA BAA BAA.

Pattern P = AABA

Sol Step1 Text T = AABA ACAA DAA BAA BAAA

pattern AABA Pattern is found at index 1.

Step2 A AABAA CAA DAA BAA BAA

ABBA Pattern is not matching

Step3 AA BAACAA DAA BAA BAA

AABA Not matching

then shifting one step to right side

Step4 AAB AACAA DAA BAA BAA

AABA Not matching

then shifting one step to right side

Step5 AABA ACAA DAA BAA BAA

AABA Not matching

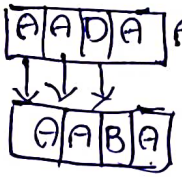
then shifting one step to right side

Step6 AABAA CAAD AABAA BAA

AABA Not matching

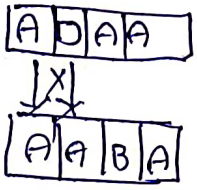
then shifting one step to right side

Step7 AABAACAADAABAABAA



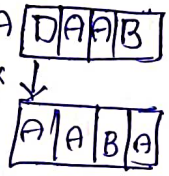
Not matching
Then shifting one step to right side.

Step8 AABAACAABAABAA



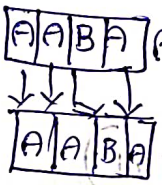
Not matching
Then shifting one step to right side.

Step9 AABAACAADAABAABAA



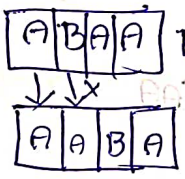
Not matching
Then shifting one step to right side.

Step10 AABAACAADABABAABAA



String is matching at index 10

Step11 AABAACAADABABAABAA

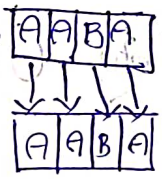


Not matching
Then shifting one step to right side

Step12 AABAACAADABABAABAA

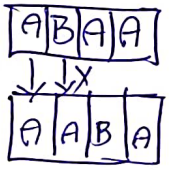
Not matching
Then shifting one step to right side.

Step13 AABAACAADABABAABAA



String is matching at index 13.
Then window is shifting one step to right side.

Step14 AABAACAADABABAABAA



Not matching
Then window is shifting to one step to right side. There is no text.

∴ Pattern is found at index 1

" " " 10

" " " 13

Algorithm

Algorithm Naive()

{

n ← length (text)

m ← length (pattern)

for s ← 1 to n-m do

do p(1 --- m) = T(s+1 --- s+m)

then pattern " pattern occurs with shift "

}

Time Complexity = O(n-m+1)

Best Case: Text : AABAACAA

Patter : FAA

Best case time complexity = O(m)

Worst case Text : AAAAAAAAA

Patterr : AAAA

Worst case time complexity = O(m × (n-m+1))

→ no. of windows

→ length of the pattern

window

Tries

Tries is Information Retrieval Data structure

Trie is one type of tree

Each and Every node stores the alphabets / strings in alphabetical order

1. Standard trie 2. Compressed trie 3. Suffix trie.

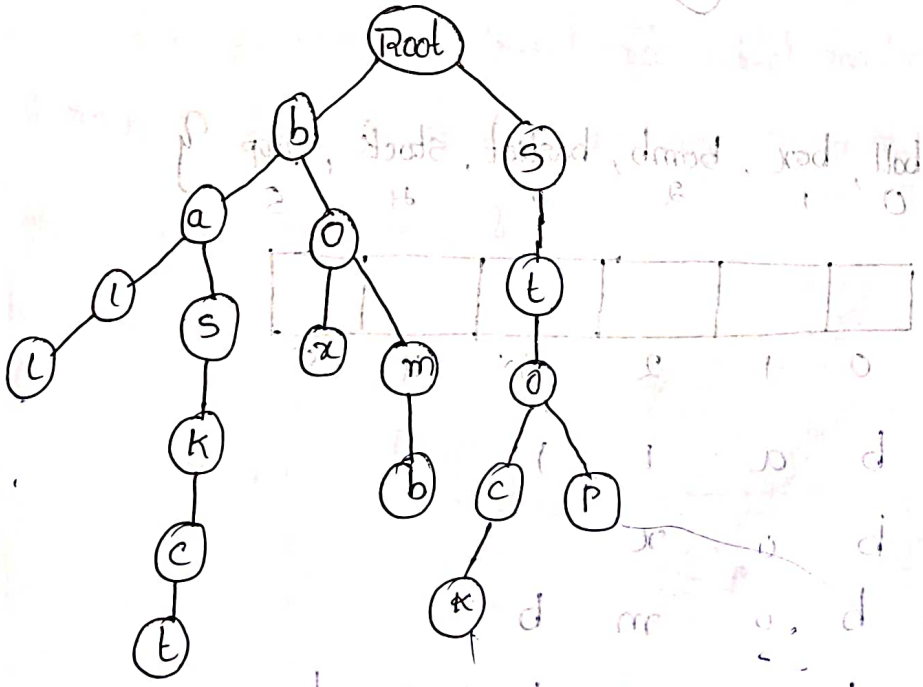
→ Tries also called as Radix trie (or) Digital trie (or) prefix trie.

(1) Standard trie :

Standard trie is a tree it is stores the strings.

→ Each and Every node in a tree stores the letter from the given string

Ex string S = { ball, box, bomb, basket, stock, stop }



(2) Compressed trie

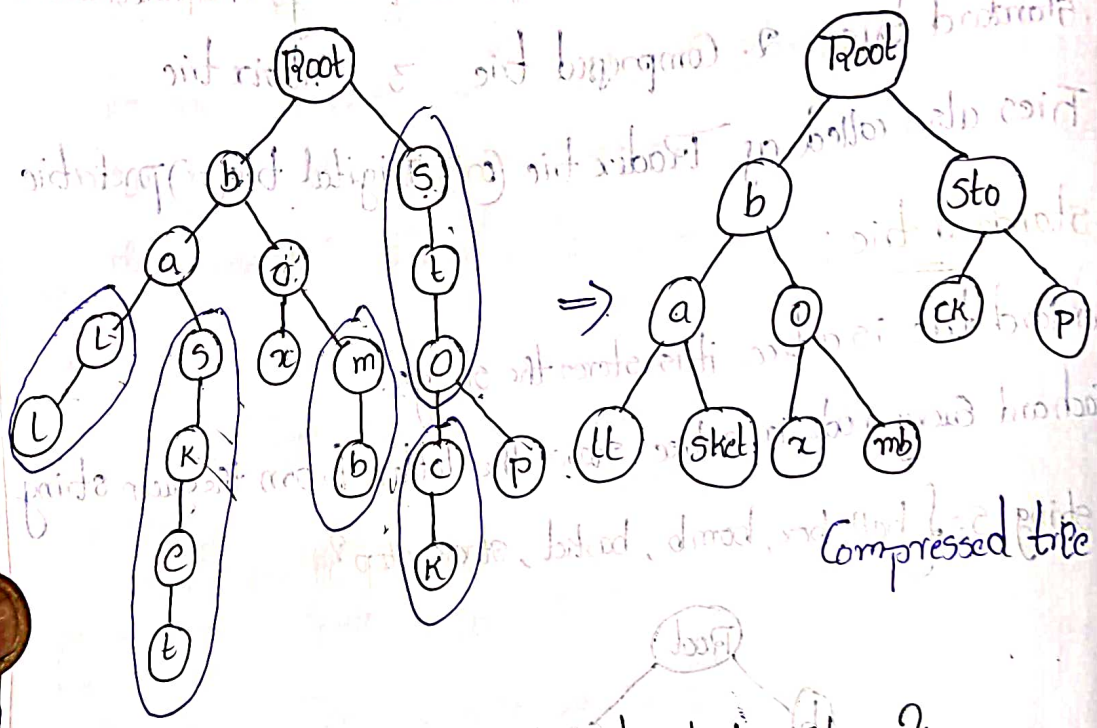
Compressed trie is a tree.

It is advanced version of Standard trie which stores that strings in the form of tree structure. Each and Every node stores the

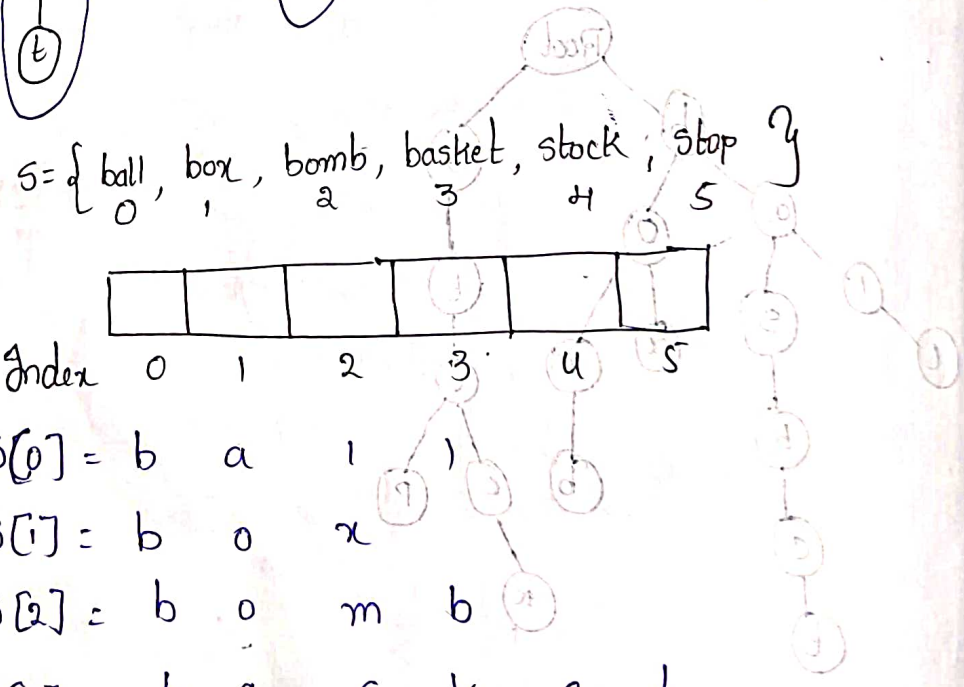
letters from the given string.

→ we are merging the leaf nodes which contains one node or g
of nodes.

Ex String = { ball, box, bomb, basket, stock, stop }



Compressed trie



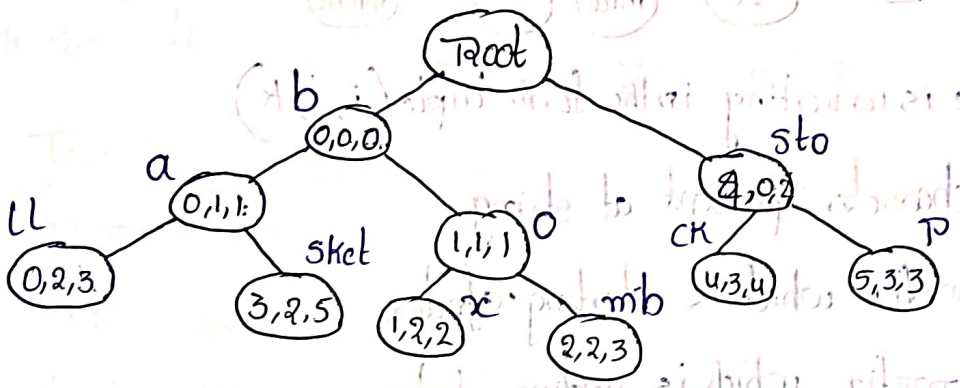
Converting into Tuple (i, j, k)

i is the (string present in the index). index present in the string

j is the prefix which is starting at index

k is the prefix which is Ending at index.

Compressed Index tree



(3) Suffix Trie

→ Suffix trie is a Compressed trie which can be used to stores the strings in the form of trees by using Suffixes.

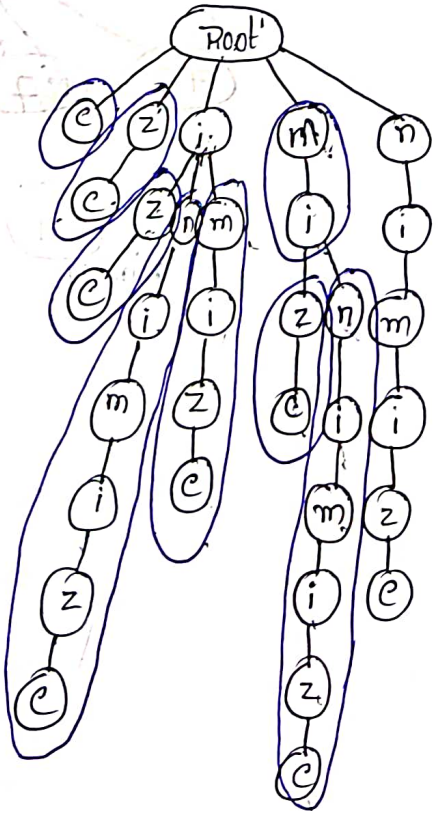
Ex string = {minimize$}

Sol

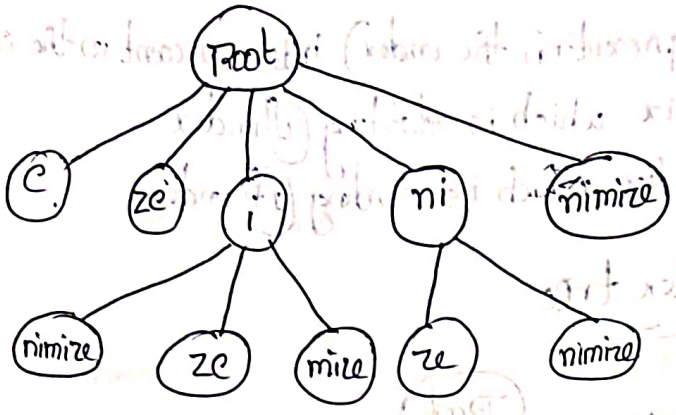
e$
ze$
ize$
mize$
imize$
nimize$
inimize$
minimize$

Construct into standard trie.

⇒



Now construct standard trie into Subtrie trie



→ The trie is converting in the form tuple (i, j, k)

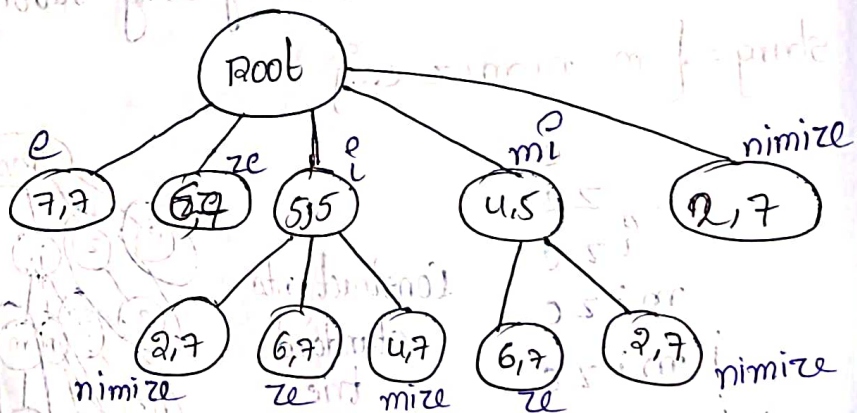
i is character present at string

j is prefix which is starting index

k is prefix which is ending index

String = { m i n i m i z e . }
0 1 2 3 4 5 6 7

s[0]



Rabin-Karp Algorithm (1987)

Rabin-karp algorithm is finding or matching the pattern in a given text by using hash value

→ first pattern characters are converting into hash values that is matching with text window hash value. if both the hash values are same then only we can check the characters

Eg: Text T = "AB CC DD AE F G"
Pattern P = "CCD"

Sol: Each window size 3.
n is the size of pattern.
m is the size of Text.
n=3, m=10, d=10 here d is the no of character in a given string.

Step 1
→ Find the hash value of pattern $h(P) = \sum(v \times d^{n-1}) \bmod 13$

A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10

"String pattern CCD"

$= (3 \times 10^{3-1} + 3 \times 10^{3-2} + 4 \times 10^{3-3}) \bmod 13$

$= (3 \times 10^2 + 3 \times 10^1 + 4 \times 10^0) \bmod 13$

$= (300 + 30 + 4) \bmod 13 = 334 \bmod 13$

$= 9$

Step 1
→ Find the hash value of Text window hash(text) = ABC

Sol: $hash(T) = (1 \times 10^{3-1} + 2 \times 10^{3-2} + 3 \times 10^{3-3}) \bmod 13$

$= 10^2 + 2 \times 10 + 3) \bmod 13$

$= 123 \bmod 13 = 6$

∴ pattern hash value and Text window hash value not matching

Step2

⇒ find the hash value of Text window. hash(text) = BCC

$$hash(T) = (2\times10^{3-1} + 3\times10^{3-2} + 3\times10^{3-3}) \bmod 13$$

$$= (2\times10^{2} + 3\times10^{1} + 3) \bmod 13$$

$$= (233 - 0) \bmod 13$$

$$= 11$$

∴ pattern hash value and Text window hash value not matching

Then text window is moving one step to right hand side.

Step3

⇒ find the hash value of text window hash(text) = CCD

$$hash(T) = (3\times10^{3-1} + 3\times10^{3-2} + 4\times10^{3-3}) \bmod$$

$$= (3\times10^{2} + 3\times10^{1} + 4\times10^{0}) \bmod 13$$

$$= (334 - 0) \bmod 13$$

$$= 9$$

∴ Pattern hash value and Text window hash value both are same. Then Compare characters.

Now we can Compare Characters

Text T = A B C C D D A E F G

pattern p = C C D ∴ pattern is found at index 3

Now the window is moving one step to right hand side.

Step4

find the hash value of text window hash(text) = CDD

$$hash(T) = (3\times10^{3-1} + 4\times10^{3-2} + 4\times10^{3-3}) \bmod 13$$

$$= (3\times10^{2} + 4\times10^{1} + 4\times10^{0}) \bmod 13$$

$= (3001401) \bmod$

$= (349) \bmod 13 = 6$

∴ pattern hash value and Text window hash value not matching then text window is moving one step to right window side.

Steps

Knauth - Morris pratt Algorithm (KMP)

We can check the matching the pattern in a given Text by using prefix table. The main idea KMP algorithm is reducing the no. of Comparisions is there any prefix value is same as or matching with Suffix values.

Ex: Text T = a b a b a b d

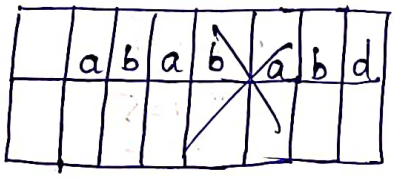
pattern p = a b a b d

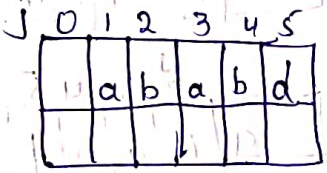
Sol first prepare the prefix table for the given pattern,

j	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

Step1 T =

i	1	2	3	4	5	6	7
	a	b	a	b	a	b	d





j	0	1	2	3	4	5
	a	b	a	b	d	

$\therefore j+1 = i$

$0+1 = 1$

$1 = 1$

$a = a$ both are matching i, j values incremented.

Step2 T =

i	1	2	3	4	5	6	7
	a	b	a	b	a	b	d

j	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

$\therefore j+1 = i$

$1+1 = 2$

$2 = 2$

$b = b$

both are matching then i, j value are incremented

Step 3

	1	2	3	4	5	6	7
T=	a	b	a	b	a	b	d

(i = 3)

	0	1	2	3	4	5
		a	b	a	b	d
	1	0	0	1	2	0

(j = 2)

$\therefore j+1=1$

$a=1$

Both are matching i, j values are incremented.

Step 4

	1	2	3	4	5	6	7
T=	a	b	a	b	a	b	d

(i = 4)

	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

(j = 3)

$\therefore j+1=i$

b = b matching

Step 5

	1	2	3	4	5	6	7
T=	a	b	a	b	a	b	d

(i = 5)

	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

(j = 4)

$\therefore j+1=i$

$4+1=5$

$5=5$

d = a not matching

j is moving to 2nd index

Step 6

	1	2	3	4	5	6	7
T=	a	b	a	b	a	b	d

(i = 5)

	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

(j = 2)

$j+1=i$

$2+1=5$

$3=5$

a = a matching now i, j values are incremented

Step 7

	1	2	3	4	5	6	7
T=	a	b	a	b	a	b	d

(i = 6)

	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

(j = 3)

$j+1=i$

$3+1=6$

$4=6$

b = b matching

Step 8

1	2	3	4	5	6	7
a	b	a	b	a	b	d

(i)

	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

(j = 4)

$j+1=7$

$4+1=7$

$5=7$

d = d matching

Boyer Moore Algorithm

Boyer moore Algorithm is the fastest string finding or matching the pattern in a string

→ If the characters are not matching then we can check the Bad Match table

→ The Bad Match table is deciding how many jumps the pattern is moving from current position to the next matching position depends on values of the Bad Match table.

Ex Text T = THIS IS A TEST

pattern p = "TEST". Find the pattern using Boyermoore Algorithm.

Sol Step1 First create Bad Match table for the pattern "TEST"

Letter	T	E	S	*
value	3	2	1	4

Find the value of each and every letter in the pattern

Value - length of the pattern - Index of letter - 1

Value (T) = 4 - 0 - 1 = 3

Value (E) = 4 - 1 - 1 = 2

Value (S) = 4 - 2 - 1 = 1

Step2 Text = | T | H | I | S | | I | S | | A | | T | E | S | T |

pattern: | T | E | S | T | X Not matching

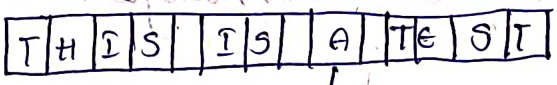
The letter 'S' is available in Bad Match Table. The value of S is = 1 The pattern is jumping one position to right side.

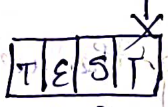
Step3 Text = | T | H | I | S | | I | S | | A | | T | E | S | T |

Pattern = | T | E | S | T | X Empty space Not matching

The Empty letter is not Available in the Bad Match table.

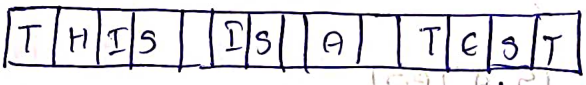
Then Consider * the value of * is 4.

Step4 Text T = 

Patter P = 

Not matching

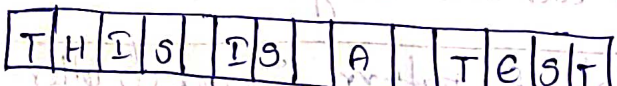
The letter A is not Available in the Bad Match Table, then Consider * the value of * is 4. That means the pattern is jumping 4 positions to the right

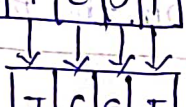
Step5 Text T = 

Pattern P = 

Not matching

Text Text window letter "S" is Available in the Bad Match table. Then the value of S is 1. That means the pattern is jumping one position to the right

Step6 Text = 



Matching.

Ex