Unit-4
Backtracking

General Method:

- It is one of the problem solving strategy.
- Backtracking follows Brute force Approach

↓

Try all possible solutions and pick the desired one.

How Backtracking works:

The backtracking works by recursively exploring all possible solutions to a problem. The algorithm starts by selecting an initial solution and then recursively explores all possible solutions by adding one element at a time. When a dead end is reached, the algorithm backtracks to a previous state and explore another branch. This process continues until a solution is found or all possible solutions have been exhausted.

Algorithm:

Algorithm involves 3 steps:

1. choose: choose a path or a solution to explore.
2. Explore: Explore the choosen path/solution.
3. Backtrack: If the choosen path or solution Leads to dead end, backtrack to the previous state or try another path.

Applications of Backtracking:

* N queen's problem
* Sum of subsets problem
* Graph coloring
* Hamiltonian cycles.

N-Queen's problem:

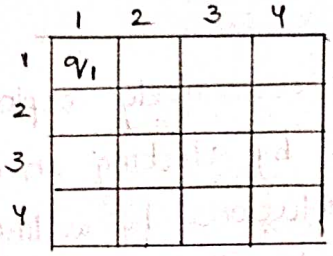
- consider n*n chessboard, our problem is to place 'n' queens on the chessboard such that no two queens attack ie,
  * No two queens should be placed on same row.
  * No two queens should be placed on same columns
  * No two queens should be placed diagonally.

Ex: 4 queens problem:

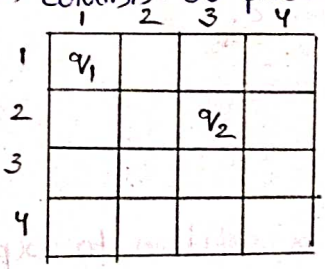
consider a 4×4 chessboard. Let there be 4 queens. The objective is to place the 4 queens on 4×4 chessboard in such a way that no two queens should be placed in the same row, same column or diagonal position.

Let $x = \{x_1, x_2, x_3, x_4\}$ be solution vector where x_i represents the column on which the queen 'i' is placed.

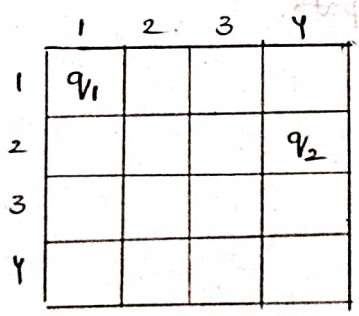
1st queen is placed in 1st row, 1st column.

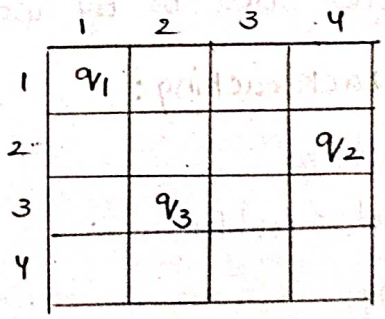


The 2nd queen shouldn't be in 1st row and 2nd column. It should be placed in 2nd row 3rd or 4th column. so place it in 3rd column.

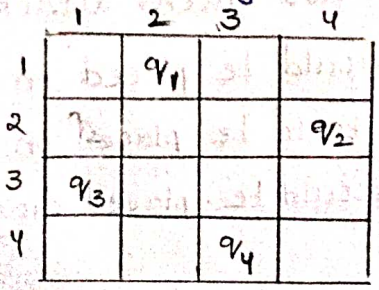


we are unable to place 3rd queen in 3rd row, so go back to queen 2 and place it somewhere.



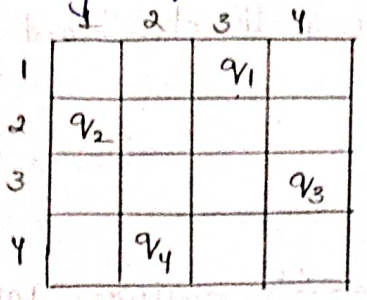


Now 4th queen should be placed in 4th row and 3rd column but there will be a diagonal attack from queen 3. so, go back, remove queen 3 and place it in next column. But it is not possible, so move back to queen 2 and remove it to next column, but it is not possible. so go back to queen 1 and move it to next column.

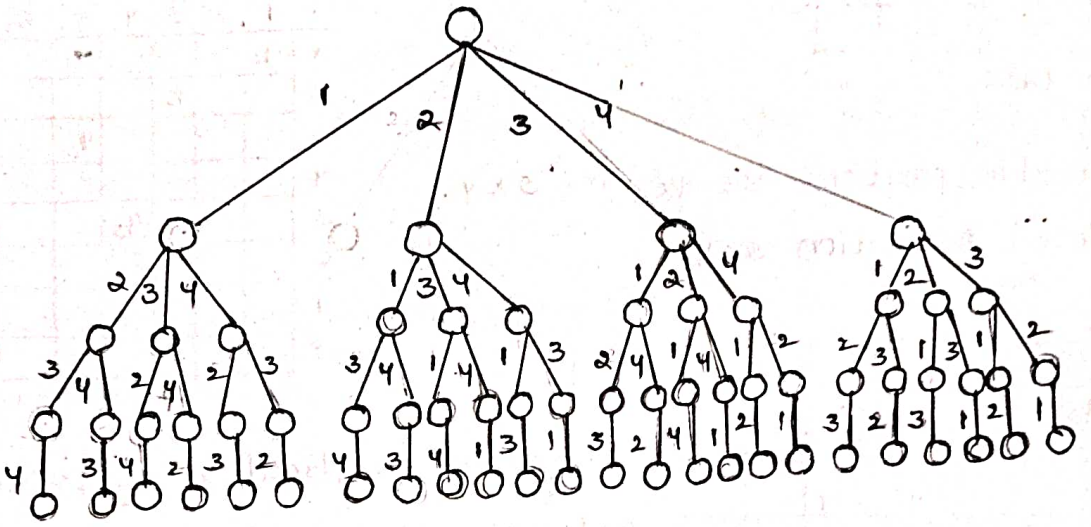


Hence solution for 4 queens problem is $\{x_1=2, x_2=4, x_3=1, x_4=3\}$

Another possibility is mirror image of chessboard.



All possible combinations can be represented, by a state space tree.

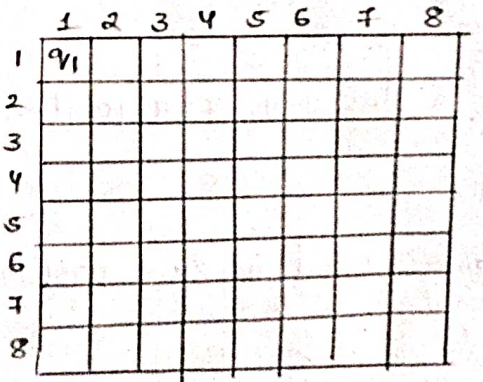


8 queens problem:

- Given a 8x8 chessboard we have to place 8 queens on chessboard in such a way that no two queens should be placed in the same row, same column or diagonal position.

step1: Q_1 is placed in 1x1 position in the chessboard.

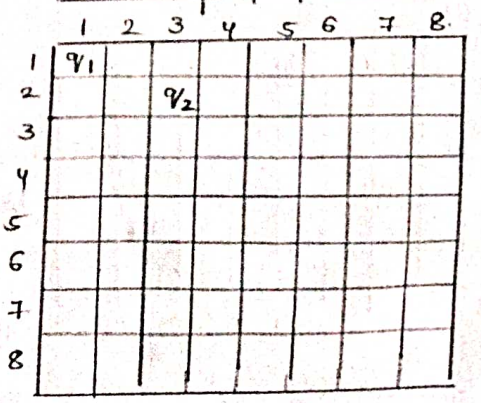
The possible position for q_1 is 1x1, 1x2, 1x3, 1x4, 1x4, 1x5, 1x6, 1x7, 1x8



(a)

step2: Q_2 is placed in 2x3 position in the chessboard.

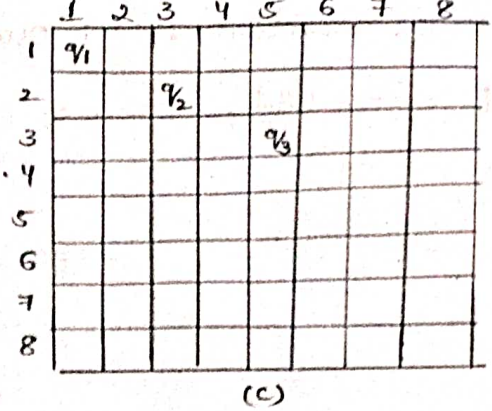
The possible positions for q_2 are 2x3, 2x4, 2x5, 2x6, 2x7, 2x8.



(b)

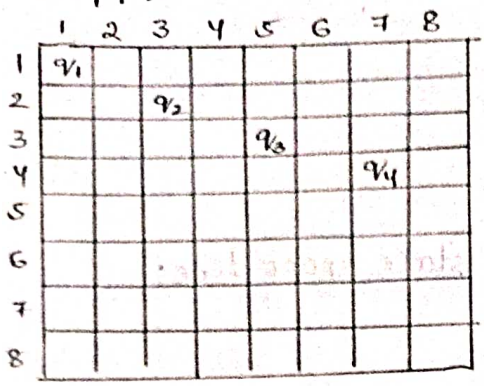
step3: the possible positions for q_3 is 3X5, 3X6, 3X7, 3X8

Q_3 is placed in 3X5 position in the chessboard.



(c)

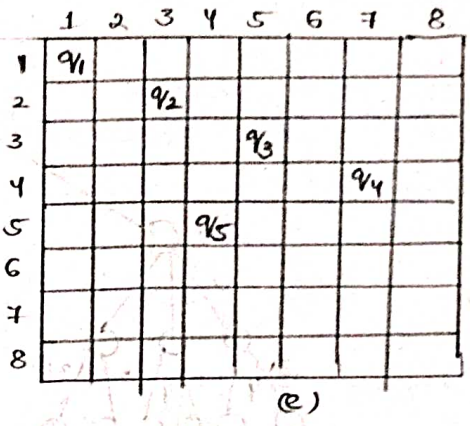
step4:



(d)

the passible positions for q_4 is 4X7, 4X8

Q_4 is placed in 4X7 position in the chessboard.



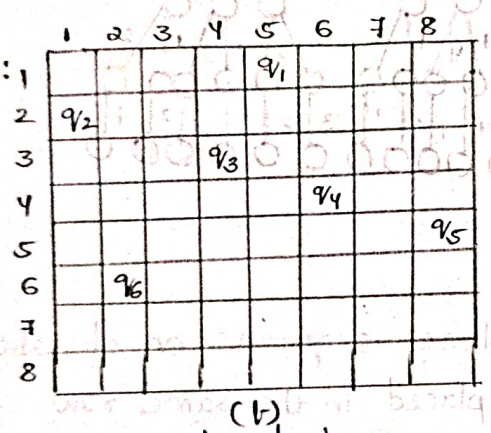
(e)

step5:

the passible positions for q_5 are 5X4.

$\therefore q_5$ is placed in position 5X4

step6:



(f)

Q_6 cannot be placed in any positions of 6X1, 6X2, 6X3, 6X4, 6X5, 6X6, 6X7, 6X8.

Hence apply Backtracking on Q_5, Q_4, Q_3, Q_2, Q_1.

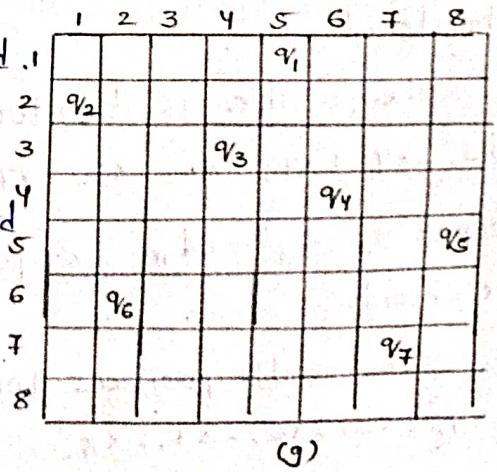
Now Q_1 is placed in 1X5, q_2 is placed in 2X1, q_3 is placed in 3X4 and q_4 is placed in 4X6, q_5 is placed in 5X8 and q_6 is placed in 6X2.

step7:

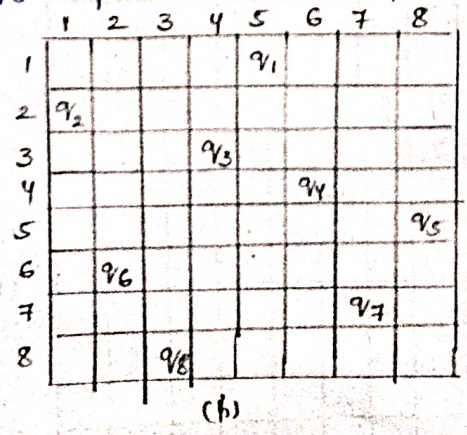
q_7 is placed in 7X7 position in the chess board.

step8:

q_8 is placed in 8X3 position in the chessboard



(g)



(h)

Hence the possible positions are 1X5, 2X1, 3X4, 4X6, 5X8, 6X2, 7X7, 8X3

Hence the feasible solution is (5, 1, 4, 6, 8, 2, 7, 3).

Sum of subsets :

- It is one of the application of Backtracking.
- Let 's' be the set of elements. our goal is to get all the subsets which gives desired sums 'M'.

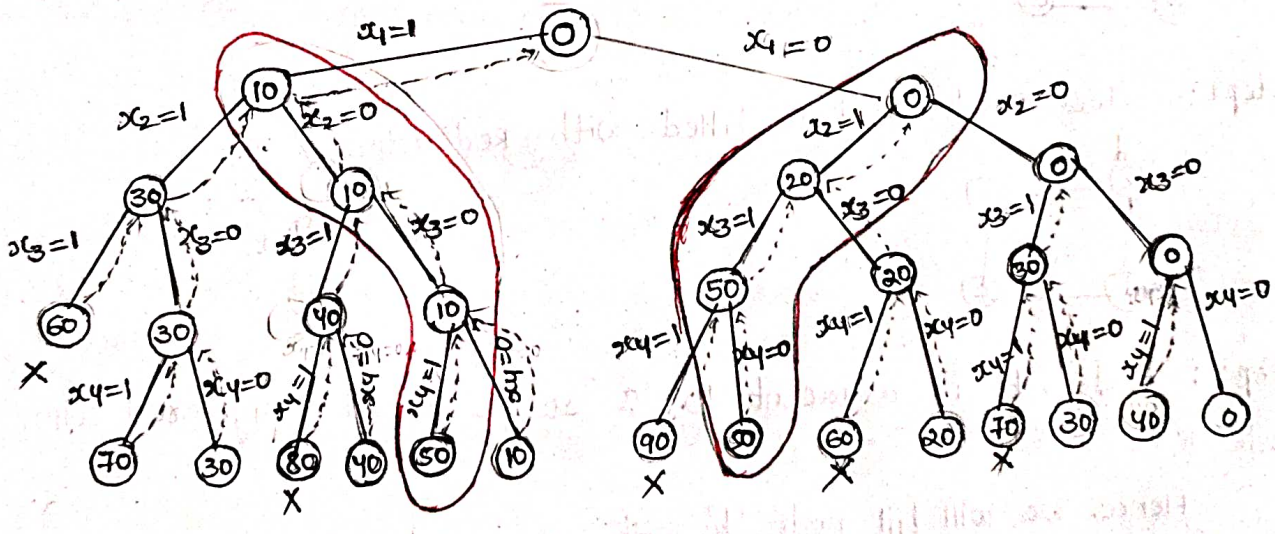
  * If we select the element $x_i = 1$
  * If we don't select the element $x_i = 0$

Algorithm:

Let 's' be the set of elements and 'M' is the expected sums of subsets.

1. start with an empty set.
2. Add to the subset next element in the List.
3. If the subset is having sum 'M' then stop with the subset as solution.
4. If the subset is not feasible, or if we have reached the end of the set then backtrack through the subset until we find the most suitable value.
5. If the subset is feasible then repeat step2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible, then stop with no solution.

Ex: consider a set $s = \{10, 20, 30, 40\}$ Generate subsets such that $M = 50$



Two possible solutions for the above problem are

(i) $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$

(ii) $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$

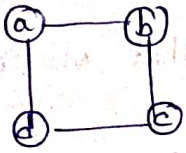
Graph coloring :

- Graph coloring is one of the application of backtracking.
- Graph coloring is a problem of coloring each vertex in the graph in such a way that no two adjacent vertices have same color and yet m-colors can be used. This problem is also called as m-coloring problem.
- If the degree of given graph is 'd' then we can color it with 'd+1' colors.
- The minimum number of colors required to color the graph is called its chromatic number.
- Time complexity is $O(m^n)$

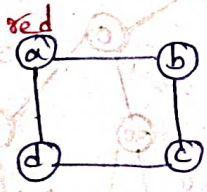
  m = no. of colors
  n = no. of nodes / vertices

Exc: color the graph with n=4 vertices and m=3 colors using Graph coloring problem and draw a state space tree. colors in = {red, green, blue} & red is represented by 1, Green is represented by 2 & Blue is represented by 3

Sol:

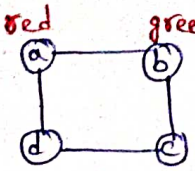


step1: select node 'a' it is filled with red color.



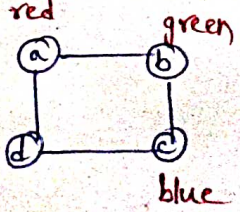
step2: node 'b' is adjacent to 'a' so we cannot assign red color to node 'b'.

Hence we will fill node 'b' with green color



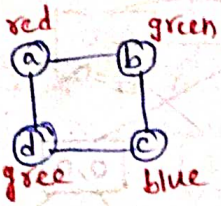
step3: node 'c' is adjacent to 'b' so we cannot assign green color to node 'c'.

Hence we will fill node 'c' with blue color

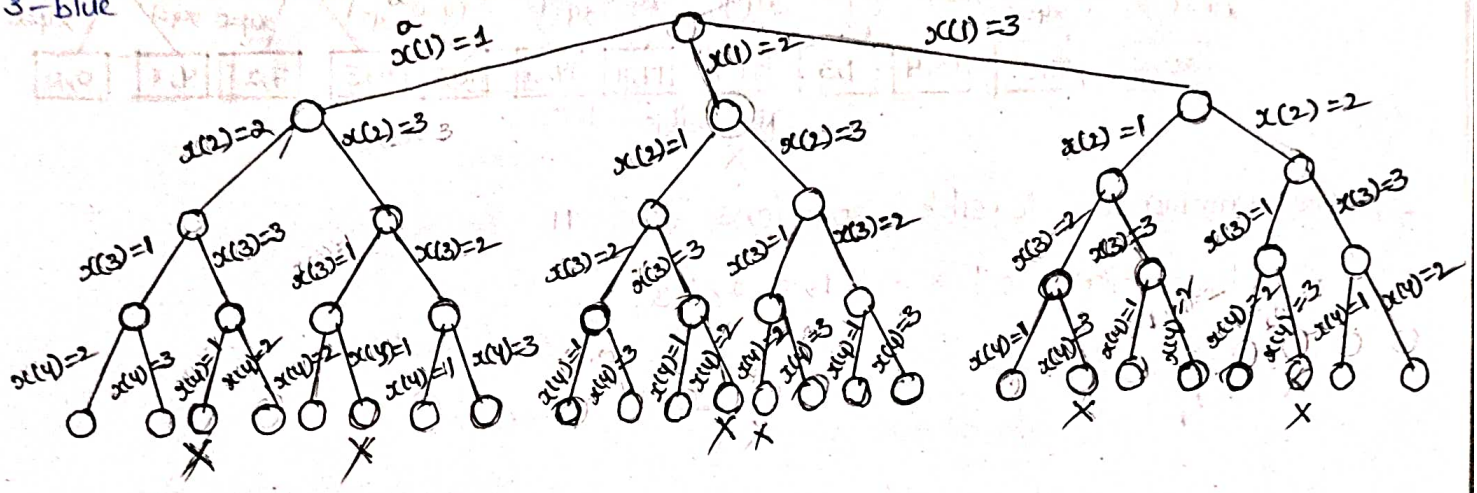


step 4 : Node 'd' is adjacent to node 'a' and node 'c' so we cannot assign either red or blue color to node 'd'.

Hence we will fill node 'd' with green color.



The state space tree for the given graph is shown below 1 → red, 2 → green 3 - blue



The 3 color graph (M=3 & n=4) produce 18 feasible solutions.

0/1 knapsack problem using Backtracking :

Given a set of ~~items~~ 'n' items $i_1, i_2, \ldots i_n$ and the weights of the items be $w_1, w_2, \ldots w_n$ and the corresponding profits be $P_1, P_2, P_3 \ldots P_n$. Given the total capacity of knapsack 'M', our objective is to maximize the profit by selecting as many items as we can and placing them in the knapsack without exceeding the knapsack capacity.

ie, $\max$ ie, maximize $\sum_{i=1}^{n} P_i x_i$ w.r.t $\sum_{i=1}^{n} w_i x_i \leq M$

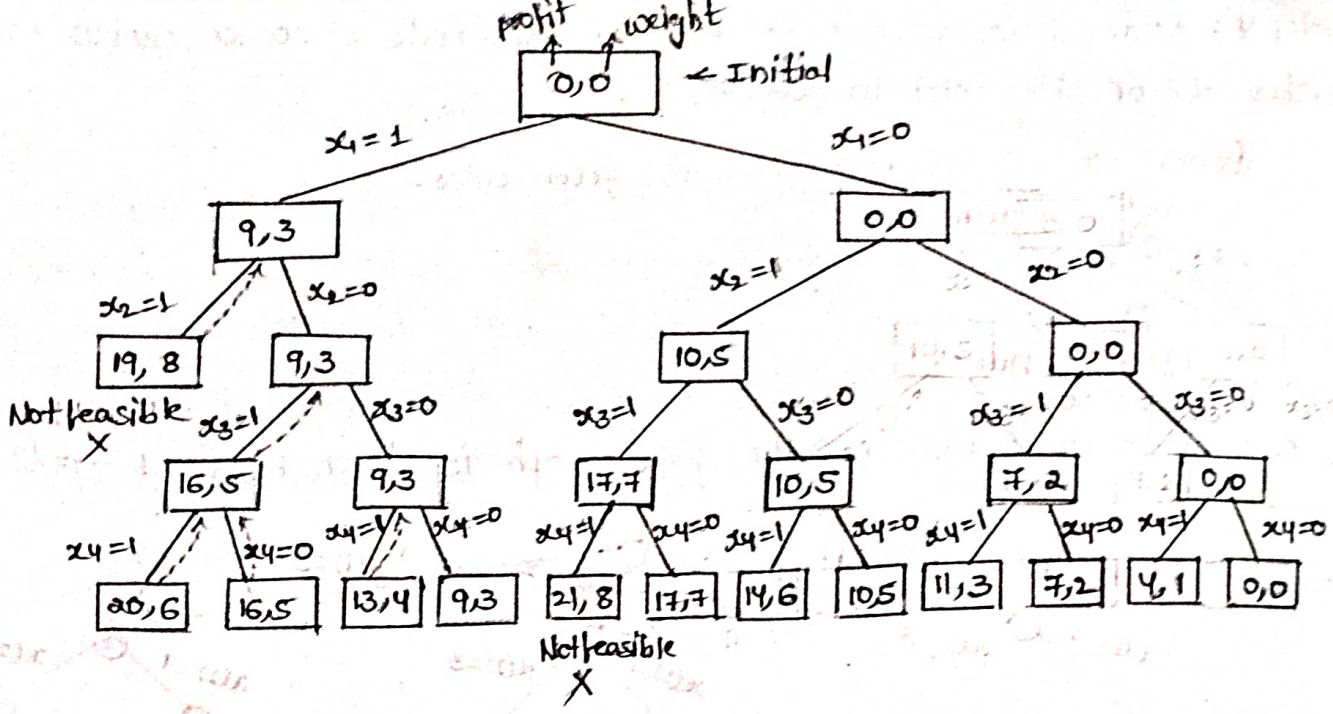
0/1 knapsack is also known as binary knapsack.

The solution vector $X = \{x_1, x_2, \ldots x_n\}$ where,

$x_i = 1 \rightarrow$ An item is placed in knapsack

$x_i = 0 \rightarrow$ An item is not placed in knapsack.

Ex: consider knapsack problem : $n = 4$. $(w_1, w_2, w_3, w_4) = (3, 5, 2, 1)$, $(P_1, P_2, P_3, P_4) = (9, 10, 7, 4)$, $M = 7$. solve the problem by using Backtracking approach.



The Number of feasible solutions are 11

optimal solution is $X = \{1, 0, 1, 1\}$

The General Method:

- Branch & Bound is one of the problem solving approach.
- Branch & bound is a method to solve optimization problems by systematically exploring different possibilities.
- It is similar to backtracking but with key differences.

1. Branching function:

This decides how the search will proceed. It can use

* Depth first search – exploring one branch deeply before other.
* Breadth first search – exploring all options level by level.
* It can be ~~bounded~~ guided by a bounding function to choose which branch to explore first.

2. Bounding function:

It helps in cutting off branches that cannot lead to the best solution. By using this function, we avoid exploring unpromising branches of the search tree.

0/1 Knapsack problem using Branch & Bound:

- Given the capacity of knapsack 'M', and number of objects/items 'n' and corresponding weights of objects $w_1, w_2, \ldots w_n$ and respective profits of objects $P_1, P_2, \ldots P_n$, our objective is to maximize the profit by placing as many items as we can inside knapsack in such a way that the sum of weights of all the items inside knapsack shouldn't exceed the capacity of the knapsack.

This is a maximization problem so Hence we have to calculate upper bound (ub).

- Here, we should always order the items in the descending order of their profits to weights ratio

ie, $\frac{P_1}{w_1} \geq \frac{P_2}{w_2} \geq \ldots$ (or) $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots$

- The upperbound of the knapsack problem can be calculated as

$$ub = V + (M - W)\frac{V_{i+1}}{w_{i+1}}$$

where,

ub = upperbound

V = profit/value of object

M = capacity of knapsack

W = weight of object

$\frac{V_{i+1}}{w_{i+1}}$ = profit per weight ratio of next object in sequence.

Ex: consider knapsack problem $n=4$, $(w_1, w_2, w_3, w_4) = (4, 7, 5, 3)$, $(P_1, P_2, P_3, P_4) = (40, 42, 25, 12)$, $M=10$. solve the problem by using Branch and bound.

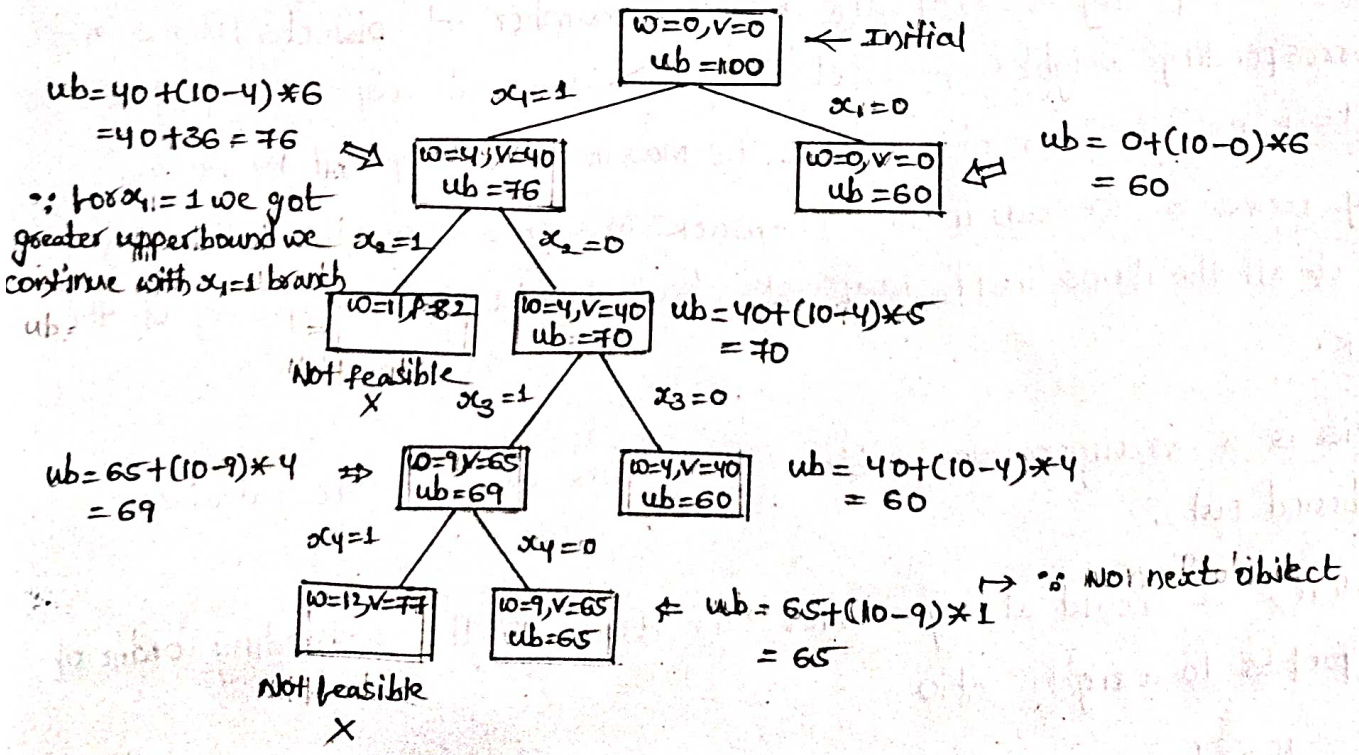
Sol: $M=10$, $n=4$

	1	2	3	4
Value (V_i)	40	42	25	12
weight (w_i)	4	7	5	3
V_i/w_i	10	6	5	4

The v_i/w_i's are already in descending order.

Now construct state space tree

Initial ub = $0 + (10-0) * 10 = 100$



∴ solution = $\{x_1=1, x_2=0, x_3=1, x_4=0\}$ & profit earned = 65.

Travelling salesman problem using Branch & Bound:

Problem:

- Given 'n' cities, a salesman starts at a specified city (source) visit all 'n-1' cities only once and return to the city from where he has started.
- Objective is to find a tour that minimize the distance / cost.
- Vertices of the graph represent various cities
- The weights associated with edges represent the distance between two cities or the cost involved from one city to another city during travelling.

Rules to remember when constructing state space tree:

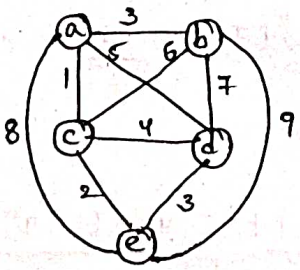
- Tour always starts at vertex 'a' (first vertex in graph)
- In the tour the first and last city remain same whereas other intermediate nodes be distinct.
- Node 'b' should be visited before node 'c'.

The Lower bound of the travelling salesman problem can be calculated as (∵ we have to use minimum distance, it is Lower bound)

$$Lb = \frac{s}{2} \text{ where,}$$

s = sum of two Lower distance edges of all vertices.

Ex: Apply the branch & bound algorithm to solve the travelling salesman problem for the following graph.



sol: compute the initial lower bound value

$$Lb = \frac{s}{2}$$

$$lb = \frac{\overset{A}{(1+3)} + \overset{B}{(3+6)} + \overset{C}{(1+2)} + \overset{D}{(3+4)} + \overset{E}{(2+3)}}{2} = \frac{28}{2} = 14$$

initial
lb = 14

compute lowerbound from vertex 'a' to other vertices

a,b ⇒ lb = (3+1) + (3+6) + (1+2) + (4+3) + (2+3) = $\frac{28}{2}$ = 14

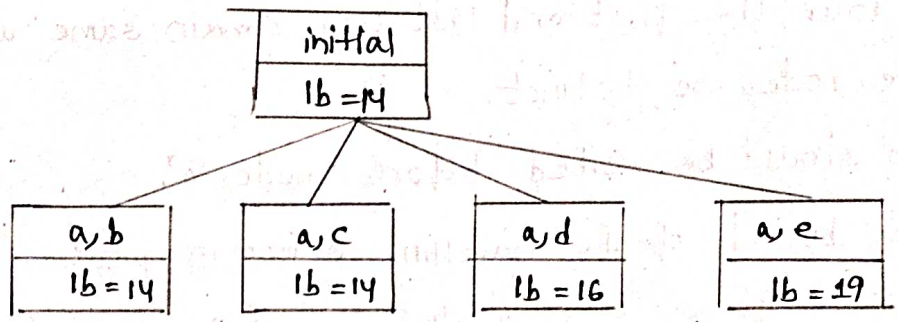
a-b edge weights must be included in both 'a' & 'b'

a,c ⇒ lb = (1+3) + (3+6) + (1+2) (4+3) + (2+3) = $\frac{28}{2}$ = 14

a-c edge weight must be included in both 'a' & 'c'

a,d ⇒ lb = (5+1) + (3+6) + (1+2) + (5+3) + (2+3) = $\frac{31}{2}$ ≃ 16

a,e ⇒ lb = (8+1) + (3+6) + (1+2) + (4+3) + (8+2) = $\frac{38}{2}$ = 19



∵ a,b & a,c are having least lower bound we consider a,b because Node 'b' should be visited before 'c' so a,c is not feasible.

compute lower bound from 'B' to other vertices

a,b,c ⇒ lb = (1+3) + (3+6) + (1+6) + (3+4) + (2+3) = 16

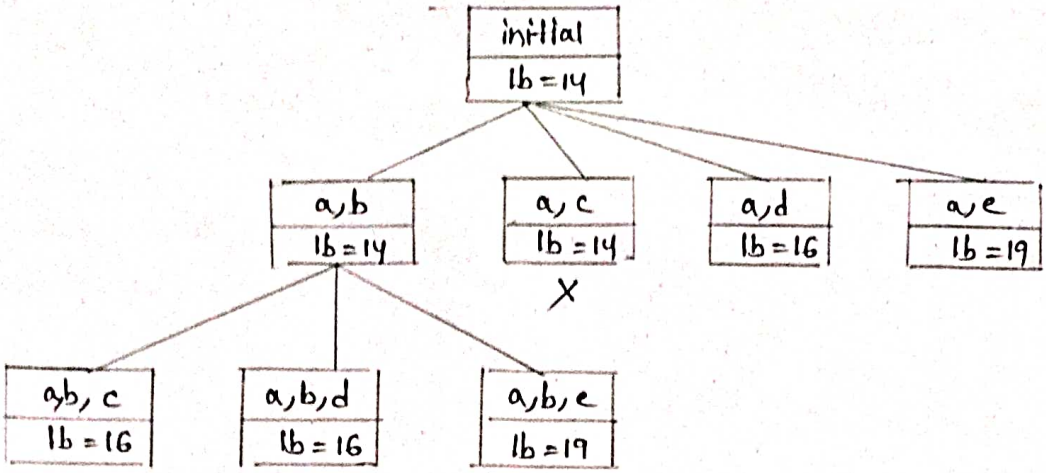
a to b, b to c

a,b,d ⇒ lb = (1+3) + (3+4) + (1+2) + (3+4) + (2+3) = 16

a to b edge, b to do edge

a,b,e ⇒ lb = (1+3) + (3+9) + (1+2) + (3+4) + (2+9) = 19

a to b, b to e



compute lower bound from a,b,c,d (e,a)

$a,b,c,d,e,a \Rightarrow lb = \frac{(3+8)+(3+6)+(6+4)+(4+3)+(3+8)}{2} = \frac{48}{2} = 24.$

(A, B, C, D, E)

compute lower bound for a,b,c,e (d,a)

$a,b,c,e,d,a \Rightarrow lb = \frac{(3+5)+(3+6)+(6+2)+(3+5)+(2+3)}{2} = \frac{38}{2} = 19$

(A, B, C, D, E)

compute lower bound for a,b,d,c (e,a)

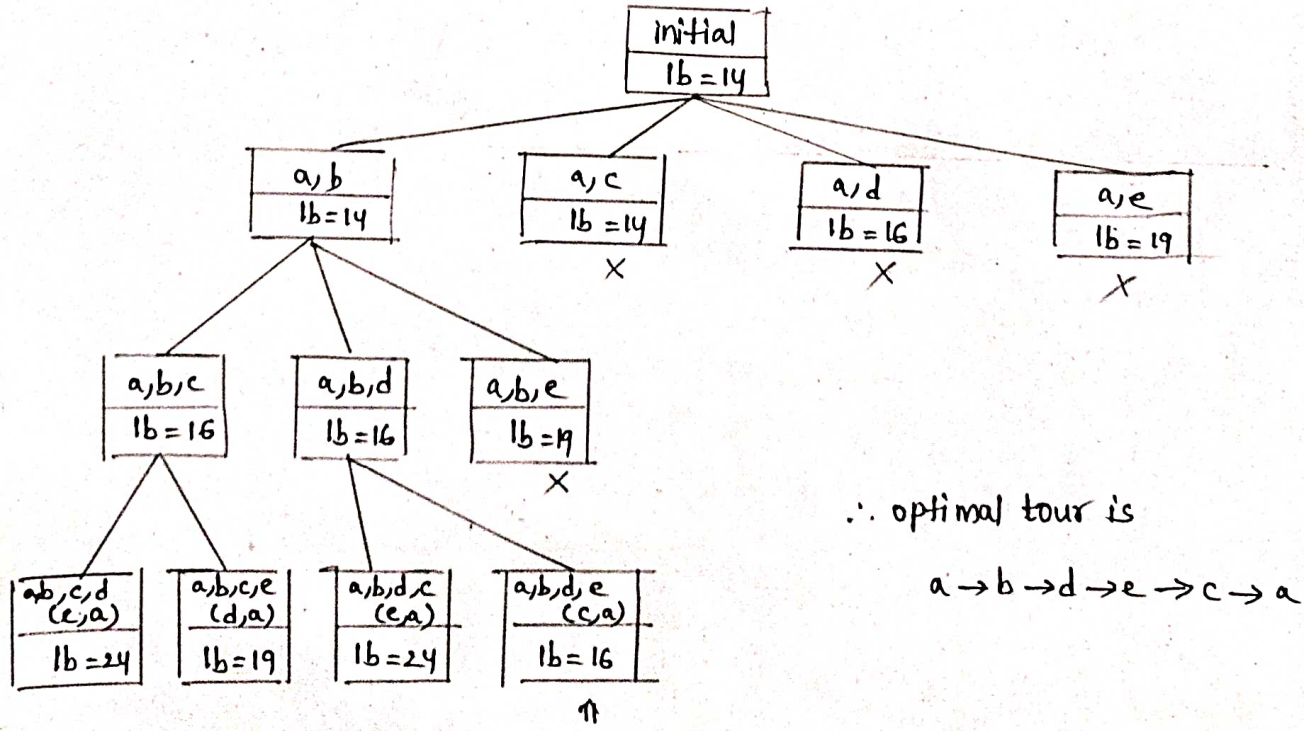
$a,b,d,c,e,a \Rightarrow lb = \frac{(3+8)+(3+7)+(4+2)+(7+4)+(2+8)}{2} = \frac{48}{2} = 24$

(A, B, C, D, E)

compute lower bound for a,b,d,e (c,a)

$a,b,d,e,c,a \Rightarrow lb = \frac{(3+1)+(3+7)+(2+1)+(7+3)+(3+2)}{2} = \frac{32}{2} = 16$

(A, B, C, D, E)



∴ optimal tour is

$a \to b \to d \to e \to c \to a$