DAA Final Preperation →

Q1 Define asymptotic notations.

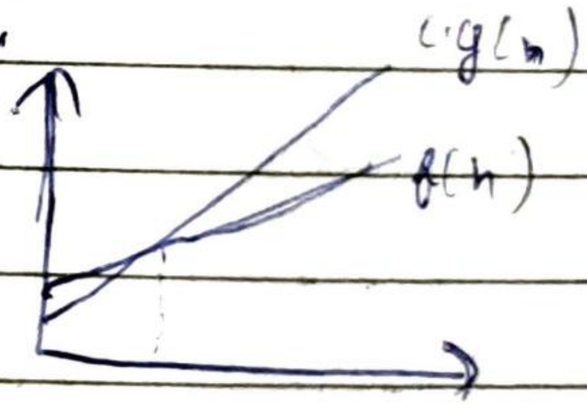
⇒ **Asymptotic notations** → These are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

There are mainly three types of asymptotic notations

2015(2) ① **Big - O notation** →

Big O represents the upper bound of the running time of an algorithm. Thus it gives the worst case complexity of an algorithm.

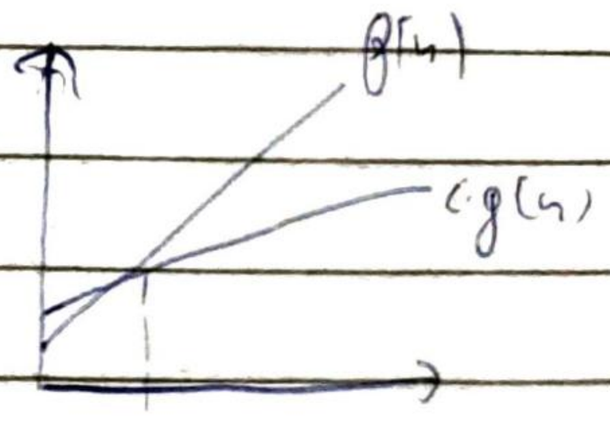
$$\boxed{f(n) \le c.g(n)}$$



② **Omega notation** →

Omega notation represents the lower bound of the running time of an algorithm. This it provides the best case complexity of an algorithm.

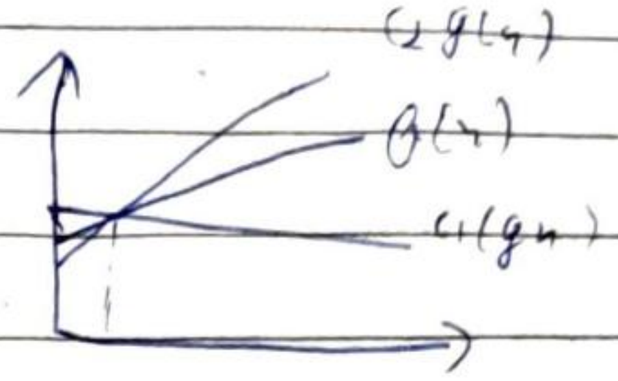
$$\boxed{f(n) \ge c.g(n)}$$



(3) Theta notation:-

Theta notation includes the function from above & below. Since it represents the upper & lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

$$\boxed{f(n) = g(n)}$$



Q2 Illustrate the time complexity of an algorithm.

=>. Time complexity is the amount of time taken by an algorithm to run the compilation

- Time complexity can be broadly explained as compiler + running / execution time

- There are some conditions for the time complexity of an algorithm:-
  1. Drop the non-dominant terms
  2. Drop the constant term

- For example ->

1. For 1 for loop ->

```
for (i = 0; i ≤ n; i++) {        (n+1) time
    // statement                 (n) time
}
```

$T = c_1(n+1) + c_2(n)$

$= c_1 n + c_1 + c_2 n$

$= c_1 n + c_2 n + c_1$

$= n(c_1 + c_2) + c_1$

$\Rightarrow T = an + b$

$\Rightarrow \boxed{T = O(n)}$

② For 2 for loop →

```
for (i = 0 to n) {                    (n+1)
    for (j = 0 to n) {                (n^2 + n)
        // statement                  (n^2 + n - 1)
    }
}
```

$T(n) = c_1(n+1) + c_2(n^2+n) + c_3(n^2+n-1)$

$= c_1 n + c_1 + c_2 n^2 + c_2 n + c_3 n^2 + c_3 n - c_3$

$= c_2 n^2 + c_3 n^2 + c_1 n + c_2 n + c_3 n + c_1 - c_3$

$= n^2(c_1 + c_2) + n(c_1 + c_2 + c_3) + (c_1 - c_3)$

$T = an^2 + bn + c$

$\boxed{T = O(n^2)}$

2015(2)
Q3 Illustrate the principle of optimality.

→ The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state & decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

Q4 Evaluate the efficiency of Binary Search.

=> Suppose there are n elements. In binary search after each iteration, the problem size (array size) reduces by half.

=> $n, n/2, n/4 \ldots 1$

=> $\frac{n}{2^k} = 1$ (Because at last, only 1 element is left)

$n = 2^k$ (2 appears k times)

Taking log on b.s

$\log n = \log 2^k$

$\log n = k \log 2$ ($\log_2 2 = 1$)

$\boxed{\log n = k}$

Best case (when $n=1$)

=> $O(1)$

Worst case (when element is not present)

=> $O(\log n)$

Q5 Examine complexity of (a) Quick Sort, (b) Merge Sort
2015 (5)

=> (a) <u>merge Sort</u> :-

- It is divide & conquer problem therefore it is a recursive algorithm

- Recurance Relation :-

$$T(n) = \begin{cases} 1 & n=1 \\ 2T(n/2) + n & n>1 \end{cases}$$

$$T(n) = 2T(n/2) + n \quad -(1)$$

Substitute $n/2$ in place of n in (1)

$$\Rightarrow T(n/2) = 2T(n/4) + n/2$$

Substitute in (1)

$$\Rightarrow T(n) = 2\left[2T(n/4) + n/2\right] + n$$

$$T(n) = 2^2 T(n/2^2) + 2n \quad -(2)$$

By (1) & (2)

$$\vdots$$

$$T(n) = 2^k T(n/2^k) + kn \quad -(3)$$

Since $T(1) = 1$

$\Rightarrow \frac{n}{2^k} = 1$

$n = 2^k$

Taking log on b.s.

$\log n = \log 2^k$

$\log n = k$

Put values in (3)

$T(n) = nT(1) + (\log n)\, n$

$T(n) = n \log n$

$\Rightarrow \boxed{O(n \log n)}$

2015(2), 2015(10)

(b) <u>Quick sort</u> →

- Quick sort has 2 cases i.e. worst case & average case

- ~~Ans~~
- Average case / Best case →

$T(n) = T(n/2) + T(n/2) + n$

$T(n) = 2T(n/2) + n$ — (1)

Subs. $n/2$ in place of n in ①

$\Rightarrow T(n/2) = 2T(n/4) + n/2$

Substitute in ①

$T(n) = 2[2T(n/4) + n/2] + n$

$T(n) = 2^2 T(n/2^2) + 2n$ — ②

By ① & ②

$T(n) = 2^k T(n/2^k) + kn$ — ③

Since $T(1) = 1$

$\Rightarrow \frac{n}{2^k} = 1$

$n = 2^k$

Taking log on b.s

$\log n = \log 2^k$

$\log n = k$

Put values in ③

$T(n) = nT(1) + (\log n)\, n$

$T(n) = n \log n$

$\Rightarrow \boxed{O(n \log n)}$

- Worst case →

$T(n) = T(n-1) + n$ — ①

Substitute $(n-1)$ in place of n in ①

$T(n-1) = T(n-2) + (n-1)$

Put in ①

$T(n) = T(n-2) + (n-1) + n$

now putting $(n-2)$ in ①, we get

$T(n) = T(n-3) + (n-2) + (n-1) + n$

$\vdots$

$T(n) = T(1) + 2 + 3 + \dots + (n-1) + n$

$\Rightarrow T(n) = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

$\Rightarrow \boxed{O(n^2)}$

2015(2)

Q- What is greedy Technique (Algorithm)?

=>. Algorithms which follow local optimal choice of each stage with intend of finding global optimum are called greedy algorithms.

- Characteristics →

① To construct the solution in an optimal way, this algorithm creates two sets where one set contains all the chosen items & another set contains the rejected items.

② It makes good local choices in the hope that the solution should be either feasible or optimal.

Ex Prim's algorithm, Kruskal's algorithm, Dijkstra's algorithm etc.

graph -> cycle
Tree -> no cycle

Q- what is spanning tree ?

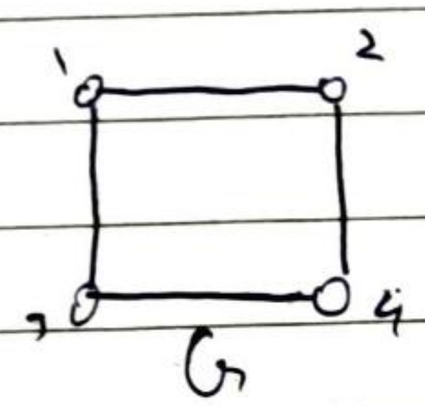
=> . Spanning Tree ->

A connected subgraph 'S' of graph G (V, E) is said to be spanning tree iff

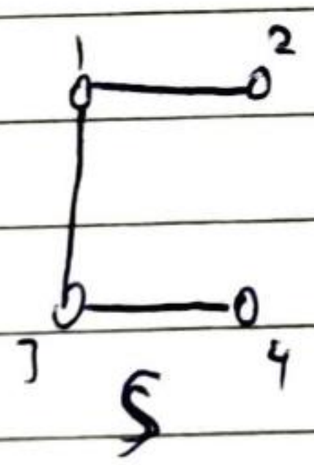
① 'S' should contain all vertices of 'G'
② 'S' should contain (|V| - 1) edges

Ex->

Given graph



& one of its spanning tree is



Q- What is minimum cost spanning Tree or minimum spanning tree.

⇒. The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.

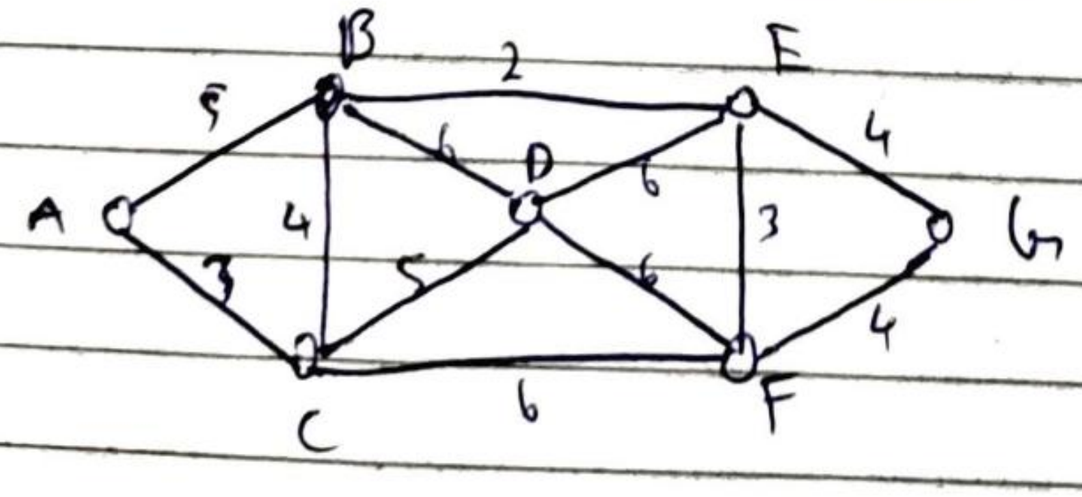
- There are two famous algorithms for finding the minimum spanning tree:

Kruskal's algorithm & Prim's algorithm.

Q- What is Kruskal's algorithm?

⇒. Kruskal's algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at ~~eve~~ every stage instead of focusing on a global optimum.

- Ex



7 vertices,
∴ 6 edges in MST

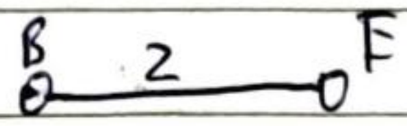
(Always start with lowest wt first)

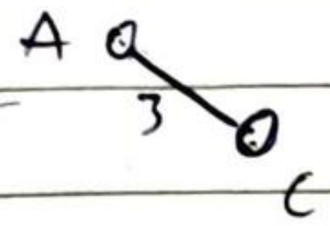
Steps for construction of MST →
Cycle should not be created at any step.

Step 1 →

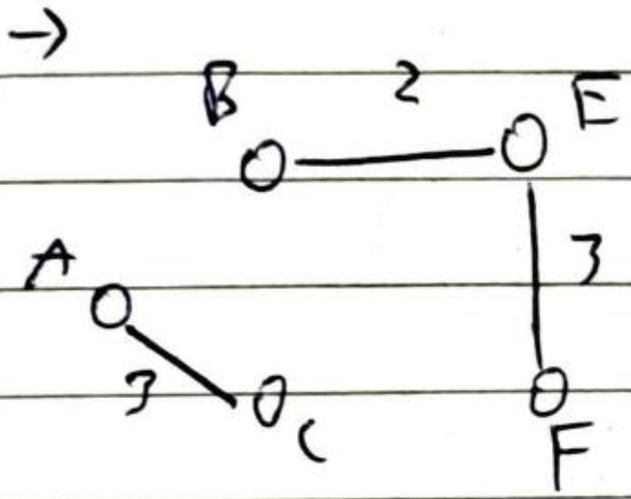
Intermediate vertex can be disconnected also

Step 2 →

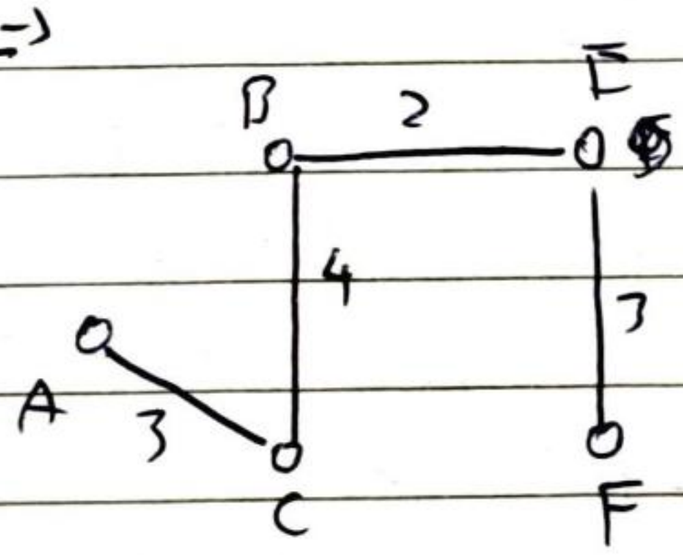




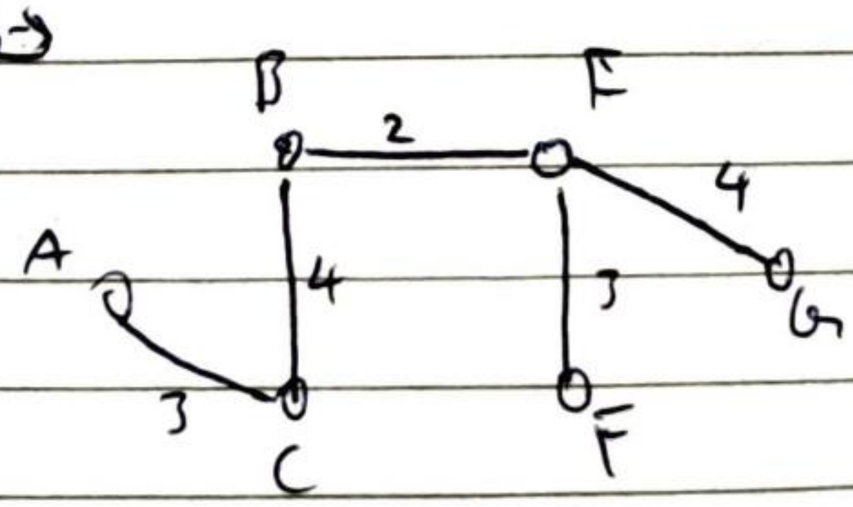
Step 3 →



Step 4 →

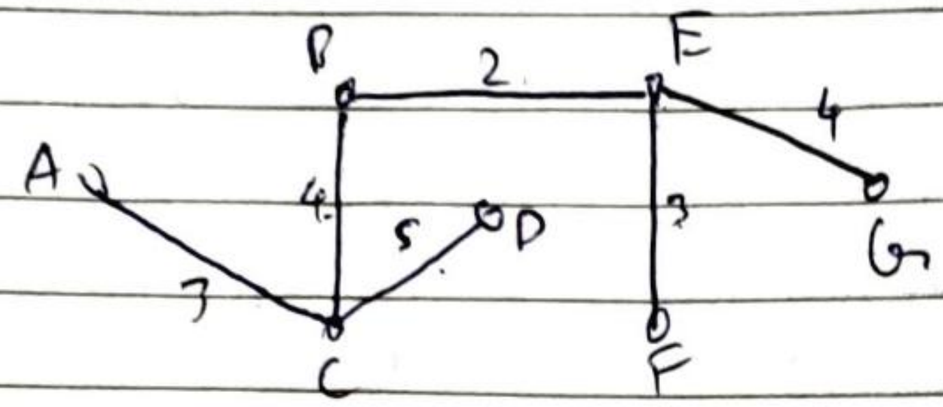


Step 5 →



Step 6 →

(Don't join FG, as cycle formed, ∴ it is graph not tree)



Since 6 edges are formed, ∴ MST received

Total cost = 21 (minimum cost)

- Time complexity → O(E log V)

2015 (5)

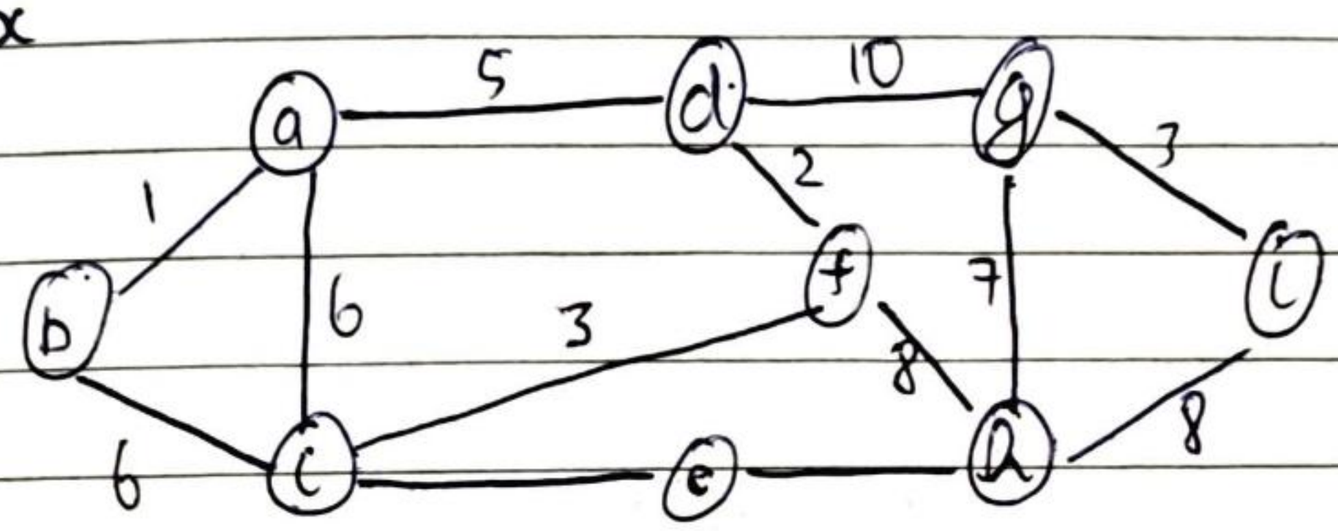
Q What is Prim's algorithm?

⇒ Prim's algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.
Prime algorithm starts with the single node & explores all the adjacent nodes with all connecting edges at every step. The edges with the minimal weights causing no cycles in the graph get selected.

- Points to remember →

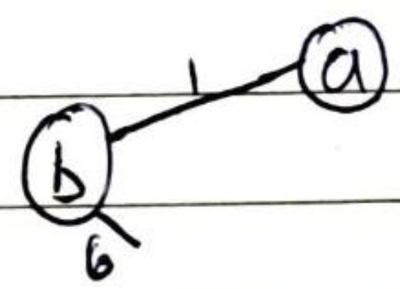
① You can start from any vertex if not given.
② Look the edges which are connected to that vertex, & choose the minimum one.
③ Consider all the open options (previous included) & choose the minimum one.
④ ~~Also~~ The intermediate result is always connected.
⑤ Also consider, cycle should not be formed at any step.

- Ex

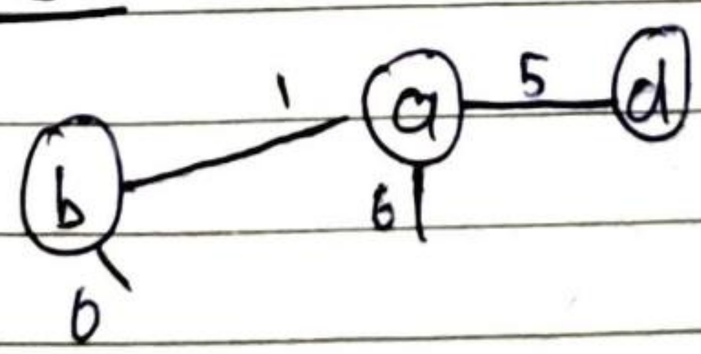


Vertices = 9
∴ Edges in MST = 8

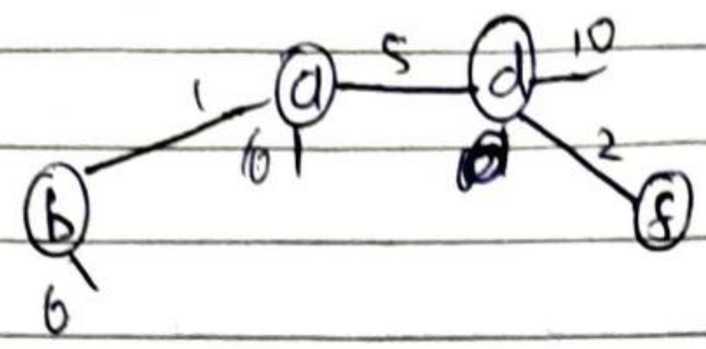
Step 1 →



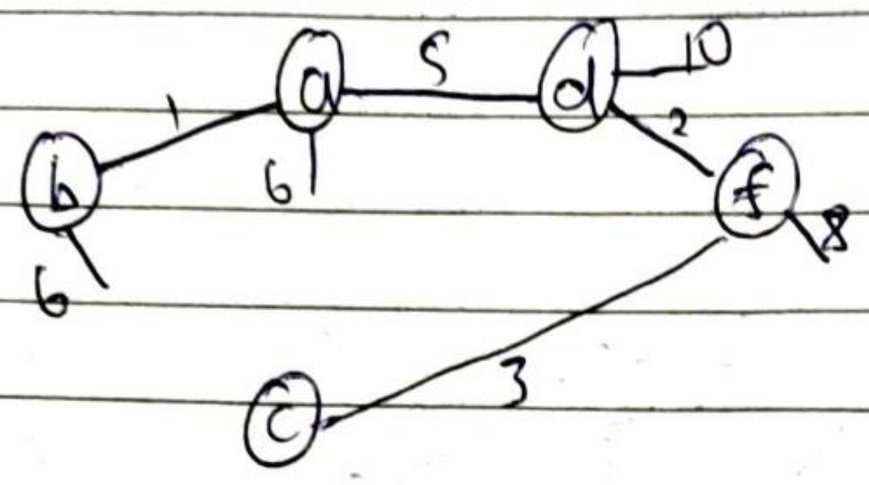
Step 2 →



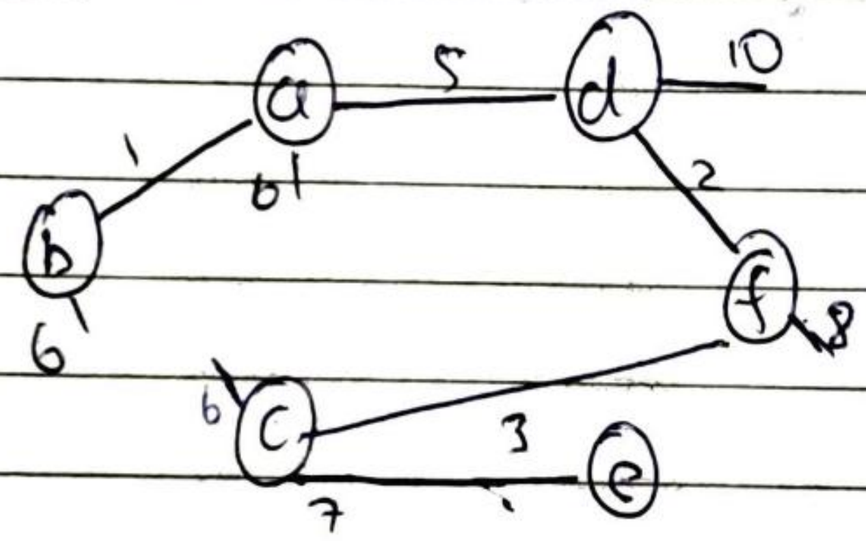
Step 3 →



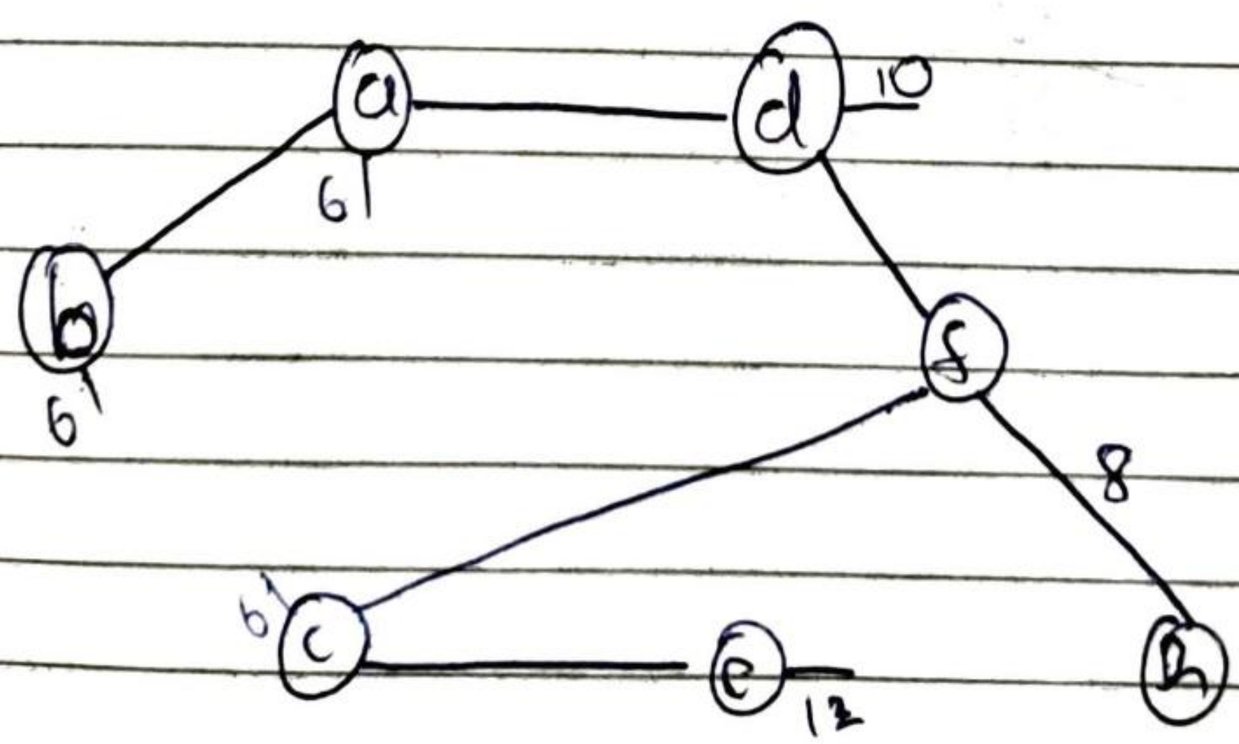
Step 4 →



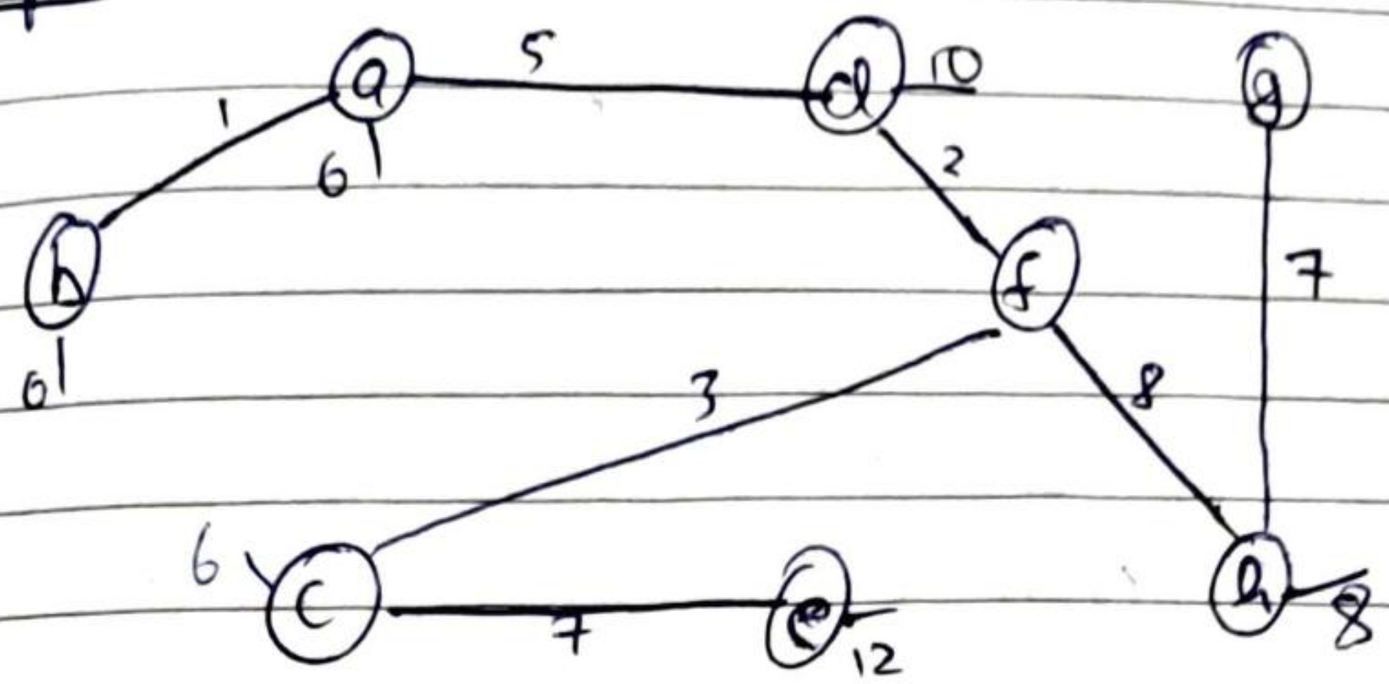
Step 5 →



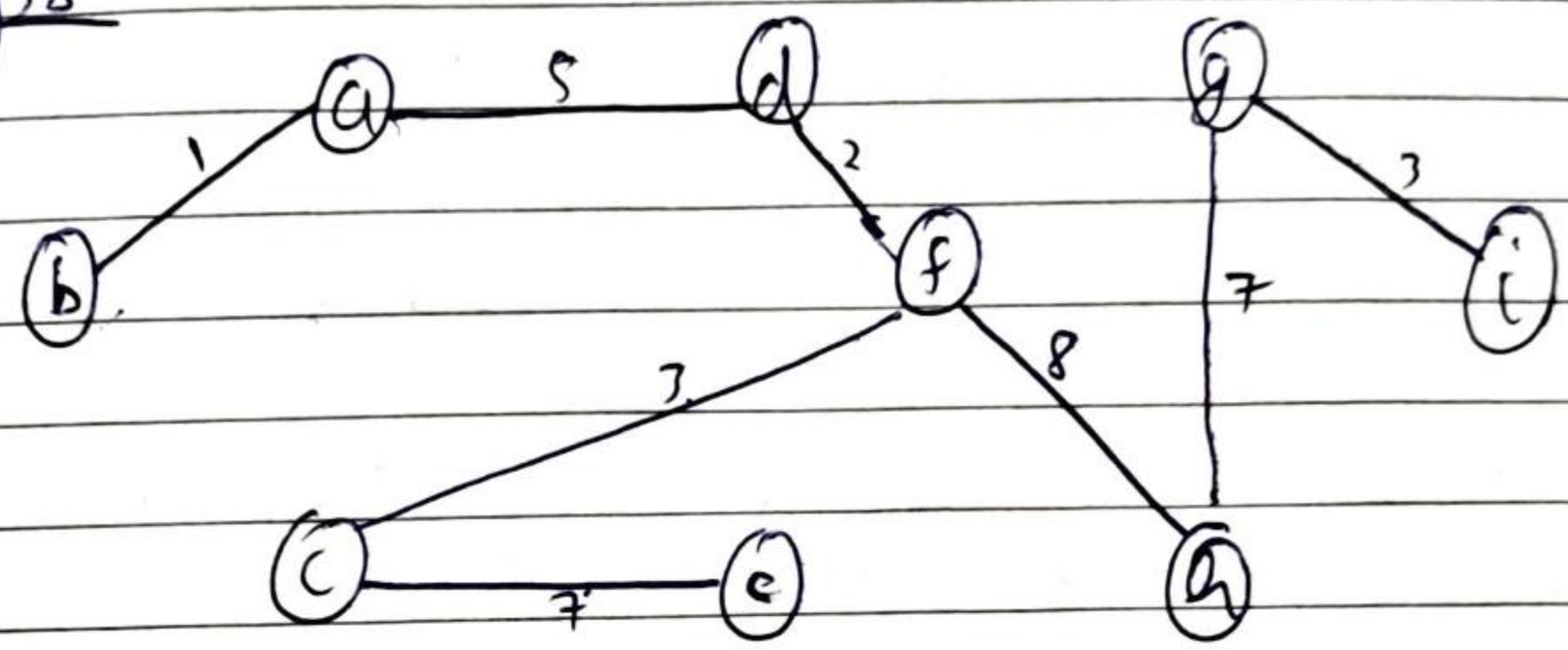
Step 6 →



Step 7 ->



Step 8 ->



Final Answer

Total wt = 36

Q7 Compare Kruskal's Algorithm & Prim's Algorithm.

=>

Kruskal's Algorithm →	Prim's Algorithm →
① It starts to build the minimum spanning tree from the vertex carrying minimum weight in the graph.	① It starts to build the minimum spanning tree from any vertex in the graph.
② It traverses one node only once.	② It traverses one node more than one time to get the minimum distance.
③ Time complexity is $O(E \log V)$, where V is no. of vertices.	③ Time complexity using heaps can be $O(E \log V)$
④ It can generate disconnected tree at any instance.	④ It has connected tree at every instance.
⑤ It runs faster in sparse graphs.	⑤ It runs faster in dense graphs.

Q- What is single source shortest path?

=> • The single source shortest path problem consists of finding the shortest paths between a given vertex v and all other vertices in the graph.

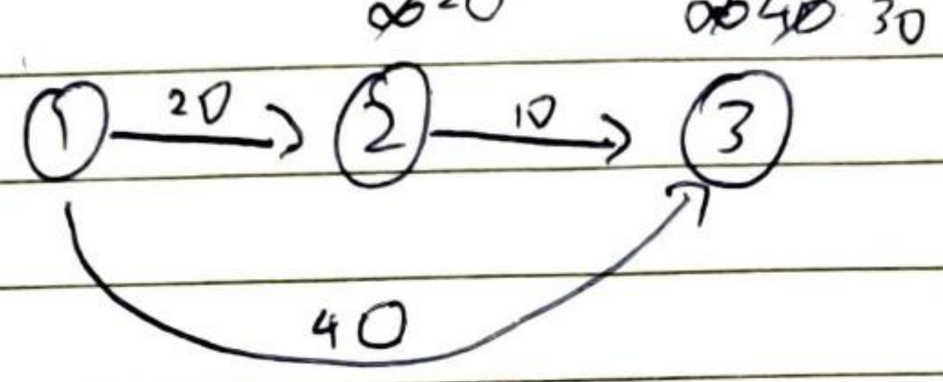
• Path may be direct or via other vertices
• Dijkstra's & Bellman Ford's algo are used.

Q What is Dijkstra's algorithm?

=> Dijkstra algorithm is a single source shortest path algorithm. Here, the single source means that only one source is given, & we have to find the shortest path from the source to all the nodes.

- The Dijkstra algorithm performs relaxation which is defined as →

if $d(u) + c(u,v) < d(v)$
then $d(v) = d(u) + c(u,v)$

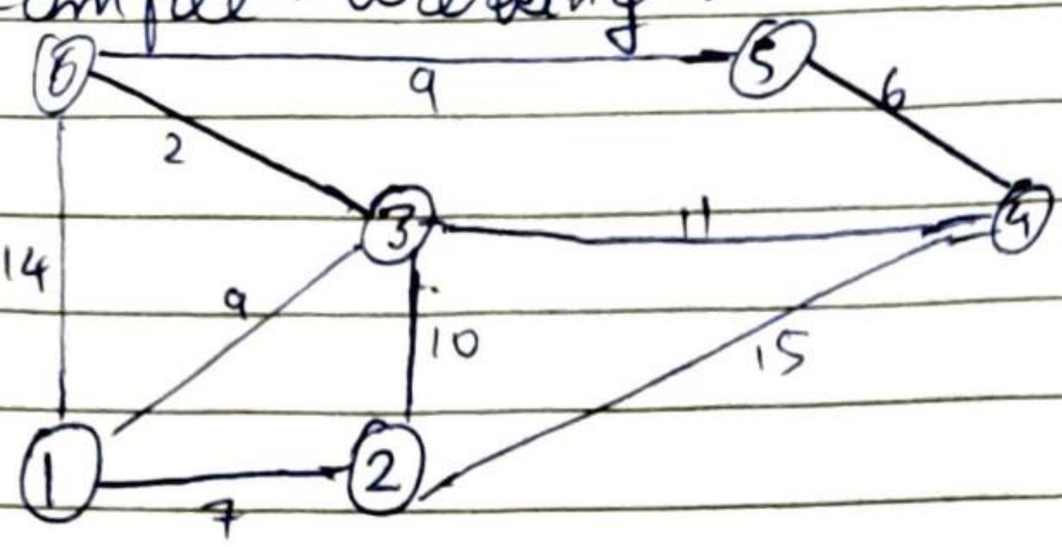


- relaxation I → (① vertex selected) (u = ①, v = ②)
  $d(u) = 0$, $c(u,v) = 20$, $d(v) = \infty$
  $d(u) + c(u,v) = 20 + 0 = 20$
  $20 < \infty$
  $\therefore d(v) = 20$

- relaxation 2 → (u = ①, v = ③)
  $d(u) = 0$, $c(u,v) = 40$, $d(v = \infty)$
  $d(u) + c(u,v) = 0 + 40$
  $40 < \infty$
  $\therefore d(v) = 40$

- relaxation 3 → (u = ②, v = ③)
  $d(u) = 20$, $c(u,v) = 10$, $d(v) = 40$
  $30 < 40$
  $\therefore d(v) = 30$

- Example - Working ->

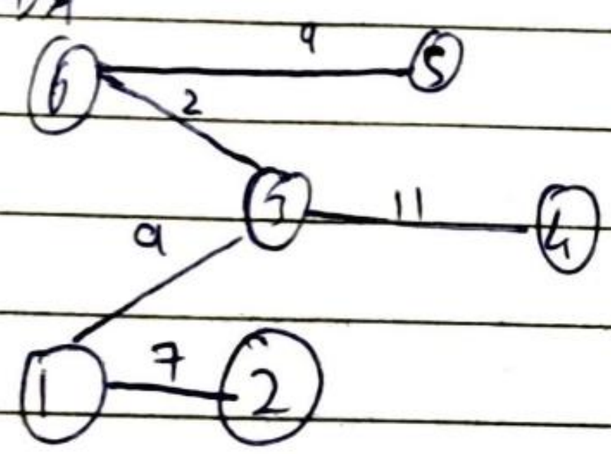


I can go from 6 -> 3 -> 5 but cost is 17

Source	Destination				
1	2	3	4	5	6
1	∞	∞	∞	∞	∞
	7	9	∞	∞	14
1, 2	7	9	22	∞	14
1, 2, 3	7	9	20	∞	11
1, 2, 3, 6	7	9	20	20	11
1, 2, 3, 6, 4	7	9	20	20	11
1, 2, 3, 6, 4, 5					

added in source as it is fixed minimum selected

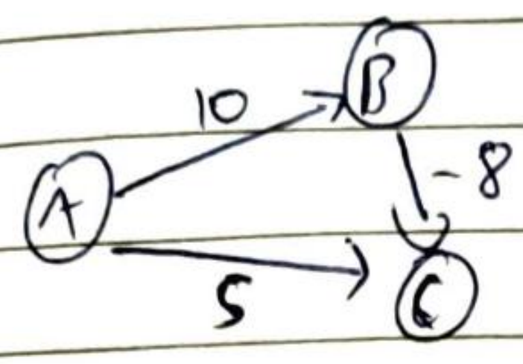
mark all the direct costs, the infinite relaxed (choose the minimum)



- Time complexity -> $O((V+E)\log V)$

Q- Why does Dijkstra fail on negative weights?

=>. Dijkstra algorithm does not necessarily give the correct results on the digraph [directed graph] when some or all the edges have negative length.



	A	B	C
		∞	∞
		10	5
A, C		10	5
A, C, B			

- Since A, C, B all are relaxed, but as we can see after relaxation, path A → B → C is also available to us whose wt is 2 (10 - 8).
- But the Dijkstra algorithm states that once the vertex is relaxed, we cannot change it.
- Due to this, the result is incorrect as we are not getting single source shortest path.

- Its solution is to use Bellman Ford algorithm

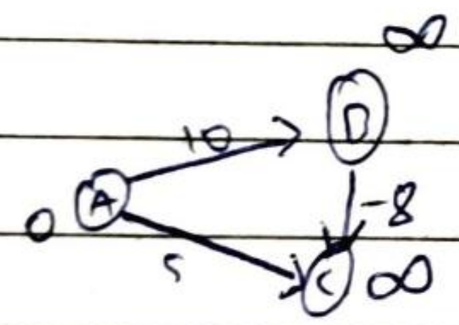
Q- What is Bellman Ford's Algorithm?

⇒. Bellman Ford Algorithm is a single source shortest path algorithm. This algorithm is used to find the shortest distance from the single vertex to all the other vertices of a weighted graph.

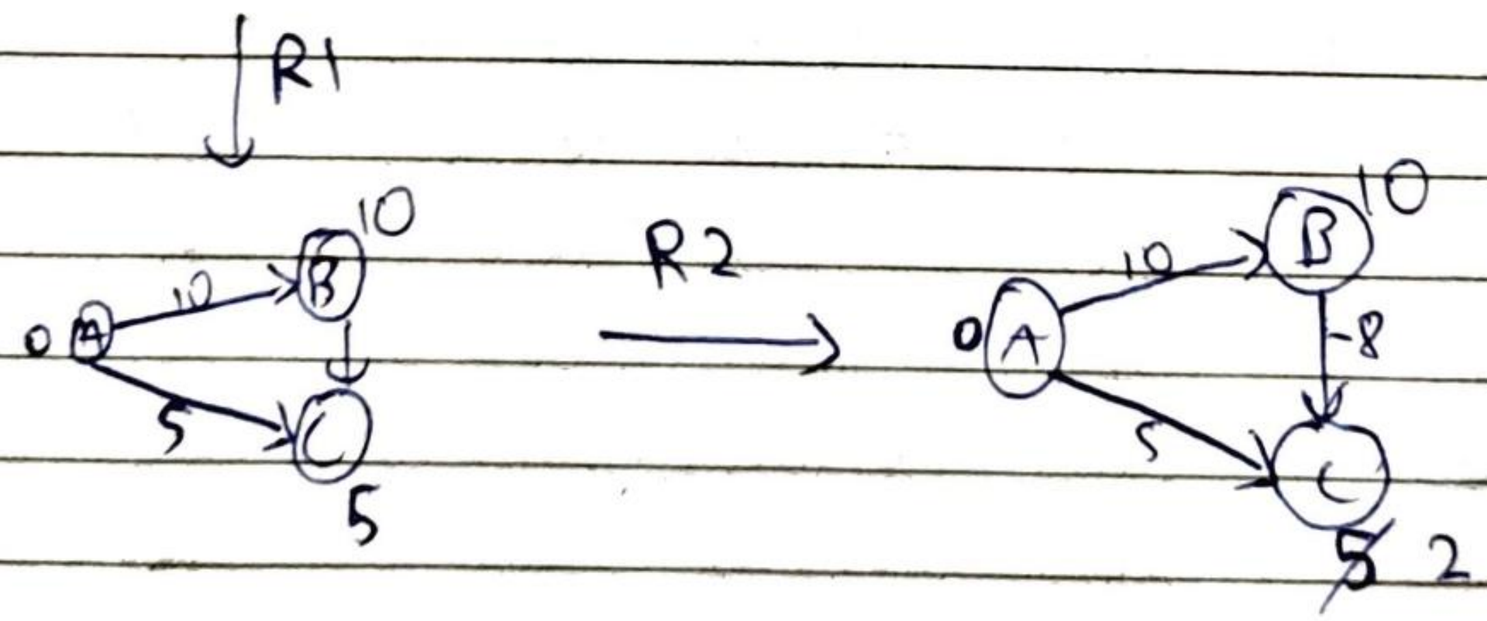
- Bellman Ford algorithm guarantees the correct answer even if the weighted graph contains the negative weight values.

- We have to relax every edge $(v-1)$ times.

- Bellman Ford also tells if there is any negative edge cycle by performing relaxation v times. If on the vth relaxation, the distance still decreases, then there is negative edge cycle present.

- 

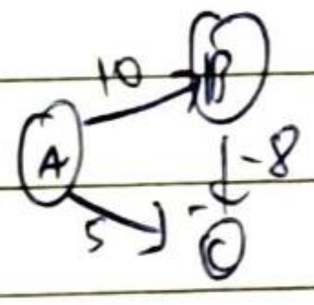
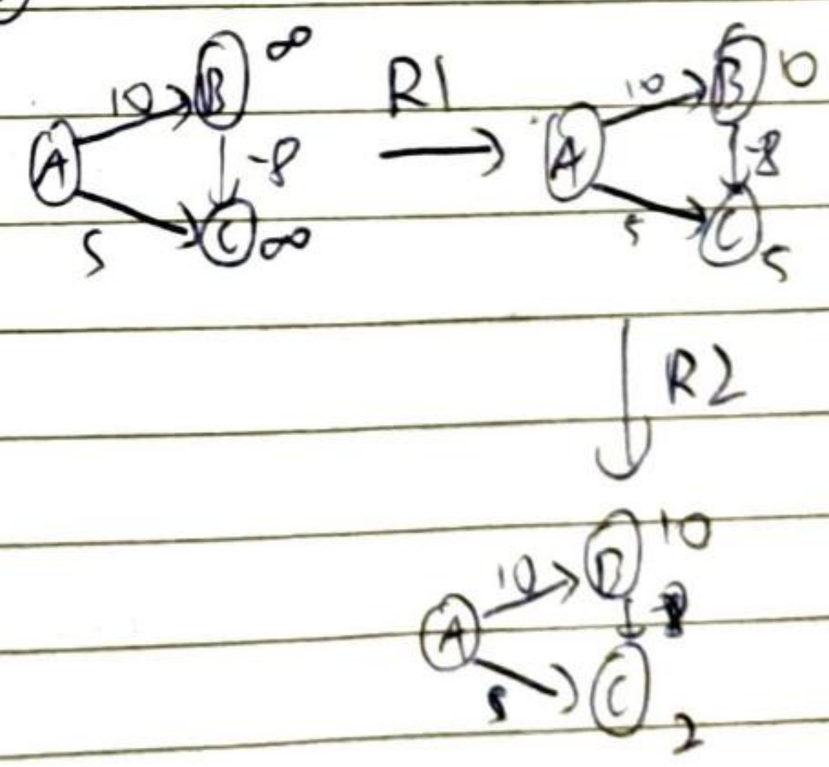
Relaxation →
if $d(u) + c(u,v) < d(v)$
then $d(v) = d(u) + c(u,v)$



- Time complexity → $O(VE)$

Q6 Compare Dijkstra's Algorithm & Bellman-Ford's algorithm.

=>

Dijkstra's Algorithm →	Bellman-Ford Algorithm →
① Dijkstra's algorithm does not works on negative weight cycle.	① Bellman Ford algorithm works on negative weight ~~edge~~ cycle. ~~& it also detects the~~ negative
② It cannot detect negative edge cycle.	② It can detect negative edge cycle.
③ Greedy approach is taken to implement the algorithm.	③ Dynamic programming approach is taken to implement the algorithm.
④ Time complexity → $O((V+E)\log V)$	④ Time complexity → $O(VE)$
⑤ Ex 	⑤ Ex 
⑥ Answer is not always optimal.	⑥ Answer is always optimal.

A	B	C
	∞	∞
A	10	5
AC	10	5
ACB		

· Dijkstra will stop here

Q What is Knapsack problem?

=> The Knapsack problem is an optimization problem used to illustrate both problem & solution. It derives its name from a scenario where one is constrained in the number of items that can be placed inside a fixed-size Knapsack. Given a set of items with specific weights & values, the aim is to get as much value into the Knapsack as possible given the weight constraint of the Knapsack.

Q 8 Discuss & give example of greedy method to solve Knapsack problem

=> To solve the Knapsack problem, we can take greedy approach for the optimal solution (i.e. maximum profit & least weight)

- Let us take an example →

Objects	Obj 1	Obj 2	Obj 3
Profit	25	24	15
weight	18	15	10

$M = 20$

① ~~Lets take greedy about profit approach → Take maximum profit first~~

~~Total profit $= 25 + \frac{2}{15} \times 24 = 28.2$~~

~~② Lets take greedy about weight approach,~~
~~Take minimum weight first~~

~~Total profit = $15 + \frac{10}{15} \times 24$~~

~~$= 31$~~

③ Lets take greedy approach for both profit & weight,
Take ratio of profit & weight

Objects	Obj 1	Obj 2	Obj 3
Profit	25	24	15
Weight	18	15	10
p/w	1.3	1.6	1.5

$M = 20$

Obj 1
Obj 2

$\frac{5}{20}$, $\frac{15}{20}$

Take maximum p/w ratio first

Total profit $= 24 + \left(\frac{5}{10} \times 15\right)$

$= \boxed{31.5}$

maximum profit

Time complexity → $\boxed{O(n \log n)}$

2015(2)

Q What is Dynamic Programming?

=> Dynamic programming is a technique that breaks the problems into sub-problems & saves the result for future purposes so that we do not need to compute the result again. Such problems involve repeatedly calculating the value of the same subproblems to find the optimum solution.

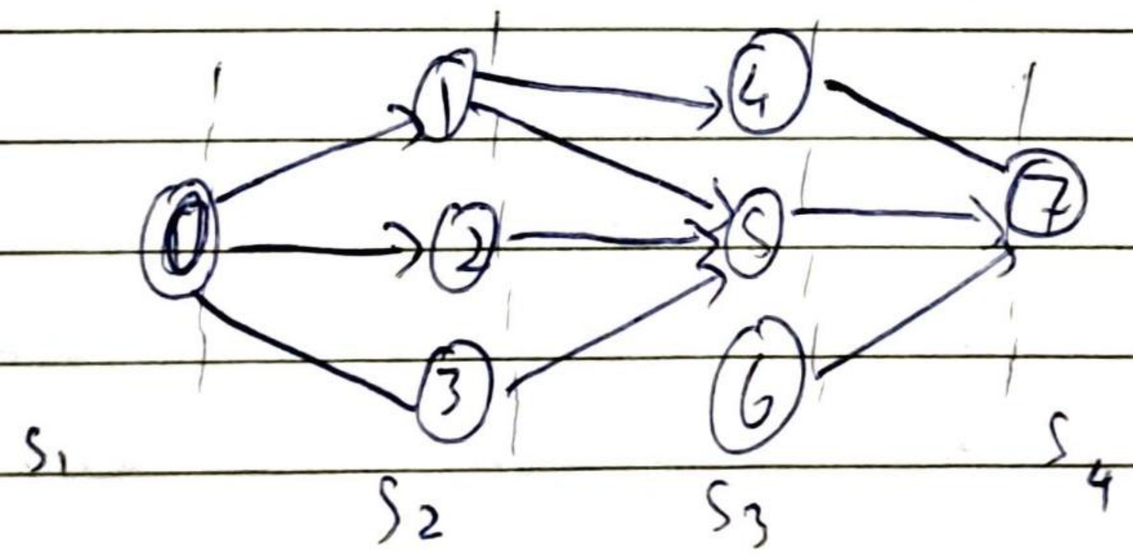
- Dynamic programming always give the optimum solution.

Q Partial knapsack vs 01 knapsack

=> Partial -> ratio of elements is added
01 -> Element is present (0) or not present (1)

Q11 What are forward & backward approaches to solve multistage graph problem

=> • <u>Multistage Graph</u> :- A multistage graph is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only.



• There are two approaches of dynamic programming to solve this problem

- Forward approach →

$$cost(i,j) = \min \{ \underbrace{c(j, l)}_{\text{edge}} + cost(\underbrace{i+1}_{\text{next stage}}, l) \}$$

- Backward approach →

$$cost(i,j) = \min \{ cost(\underbrace{i-1}_{\text{previous stage}}, l) + c(\underbrace{l, j}_{\text{edge}}) \}$$

- Time complexity → $\boxed{O(V+E)}$

vertices ← V, E → edges

Q - What is All Pair Shortest Path?

⇒ The All Pair Shortest Path is used to find the minimum distance from any node to all other nodes in the graph.

- This will generate a matrix, which represents the minimum distance from any node to all other nodes in the graph.

- Floyd Warshall Algorithm is used to find All Pair Shortest Path.

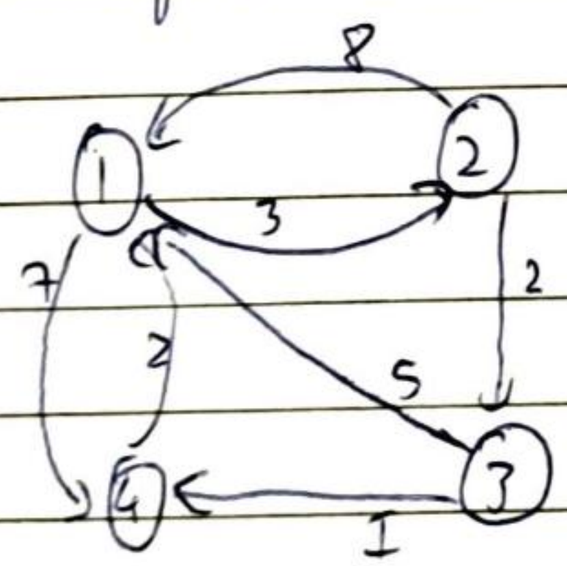
Q 10 Elaborate how dynamic programming is used by Floyd Warshal Algorithm. Evaluate the efficiency of Floyd-Warshall Algorithm.

=> • Floyd Warshall algorithm is used for finding shortest path b/w all pairs of vertices in a weighted graph. This algorithm follows dynamic programming approach to find the shortest path. By using dynamic programming, the previous results are stored in a matrix & the new result is derived from the previous result.

• The result is derived from the formula →

$$A^{k}[i,j] = \min\{A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j]\}$$

• Example →



A^0 i.e. all the given paths directly, if not then ∞

$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & \infty \\ 3 & 5 & \infty & 0 & 1 \\ 4 & 2 & \infty & \infty & 0 \end{array}$$

(diagonals always 0 as forming self loop)

A^1 i.e. find path via vertex 1 using A^0

$$A^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

(all the paths that belongs to vertex 1 will remain unchanged & 1 row & 1 column) same as previous

Find like

$A^0[2,3]$ $\quad$ $A^0[2,1] + A^0[1,3]$

$2 < 8 + \infty$

$\therefore$ previous value will be stayed

$A^0[2,4]$ $\quad$ $A^0[2,1] + A^0[1,4]$

$\infty > 8 + 7 = 15$

$\therefore$ now update $A^1[2,4]$ to 15

//merely find all others

A^2 i.e. find path via vertex 2 using A^1

$$A^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

(2 row & column same as previous)

$A^1[1,3]$ $\quad$ $A^1[1,2] + A^1[2,3]$

$\infty > 3 + 2 = 5$

$\therefore$ update $A^1[1,3]$ to 5

//merely

A^3 i.e. find path via vertex 3 using A^2

$$A_3^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

Finally

$$A^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 3 & 6 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

A^4 gives the shortest path i.e. δ_{ij}

- Efficiency →

```
for (k=1; k<=n; k++) {
    for (i=1; i<=n; i++) {
        for (j=1; j<=n; j++) {

            A[i,j] = min(A[i,j], A[i,k] + A[k,j]);
        }
    }
}
```

$\boxed{O(n^3)}$

Q12 Classes of complexity?

⇒ ① Constant complexity: $O(1)$

② Logarithmic complexity: $O(\log(n))$

③ Quadratic complexity: $O(n^2)$

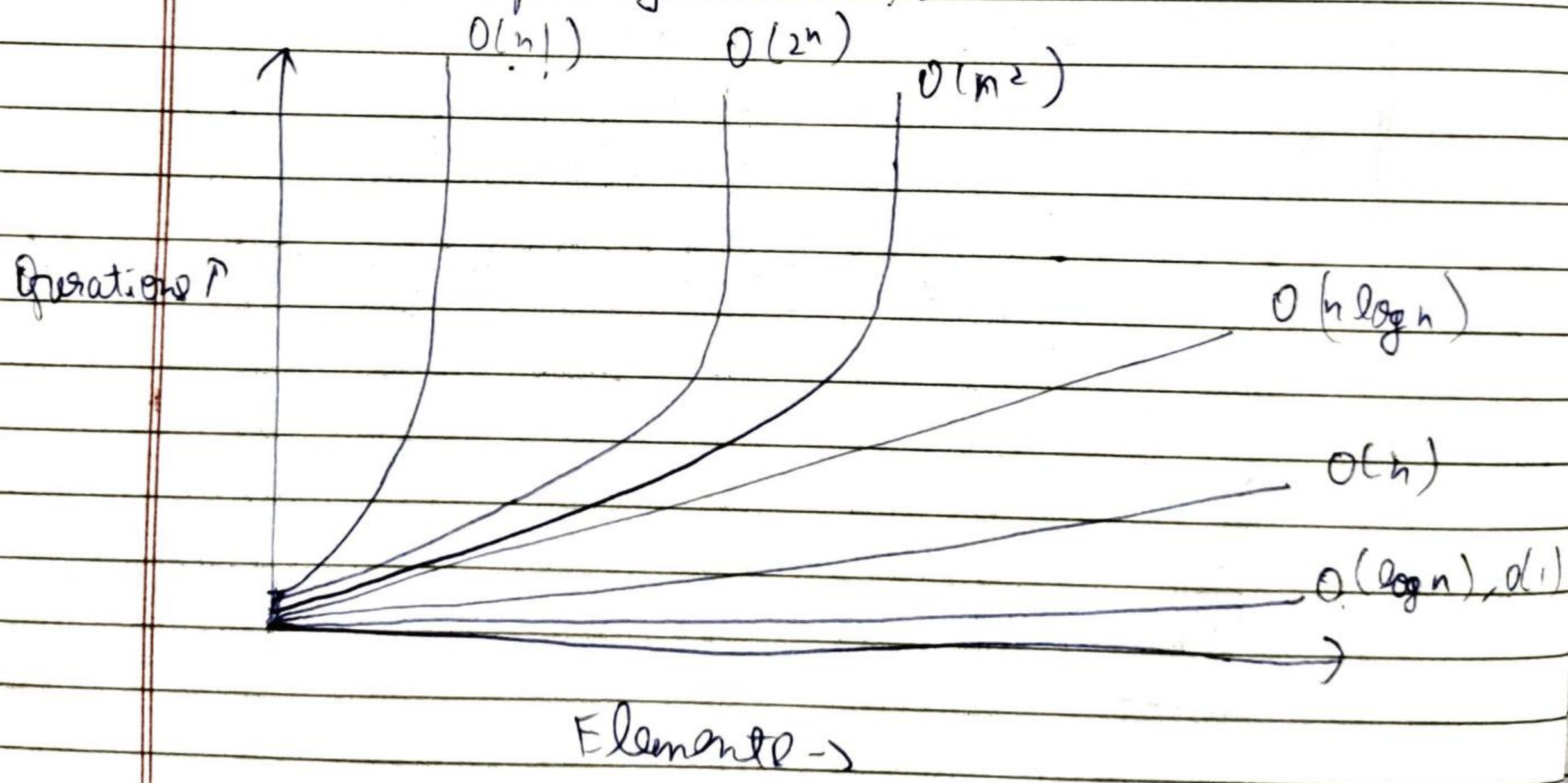
④ Cubic complexity: $O(n^3)$

⑤ Polynomial complexity: $O(n^i)$, $i = 1, 2, 3 \ldots$

⑥ Linear complexity: $c \geq 1, 2, 3 \ldots$

⑦ Exponential complexity: $O(c^n)$ where $n = 2, 3, \ldots$

⑧ Factorial complexity: $O(n!)$

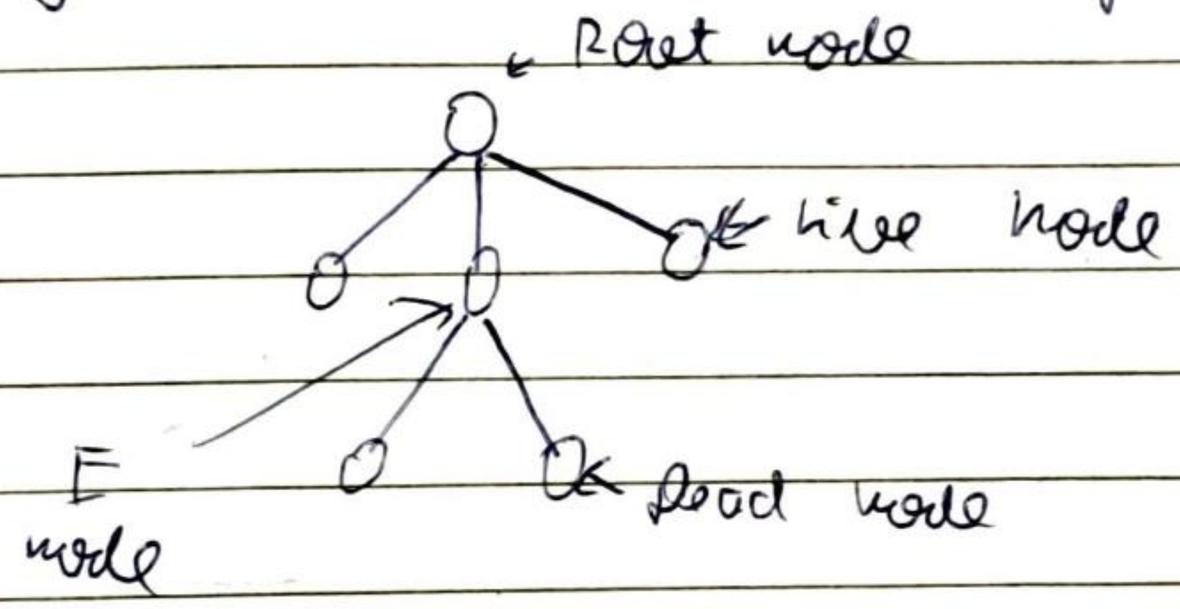


2015(S)
Q I Describe in detail general backtracking algorithm.

=) • Backtracking is an algorithm technique for solving problems recursively by trying to build a solution incrementally, one piece at a time removing those solutions that fails to satisfy the constraints of the problem at any point of time.

• The basic idea of the backtracking algorithm is to build up the solution vector one component at a time & to use Bounding functions to test whether the vector being formed has any chance of success.

• Generally a state space tree is generated as →



• Some problems which can be solved using backtracking are →
n queen problem, sum of subsets, graph colouring, knapsack problem.

1-queen → solvable
2, 3 → unsolvable
4, 5 ... → solvable

2015(8)

Q-2 Demonstrate how backtracking can be used to solve n-queen's problem.

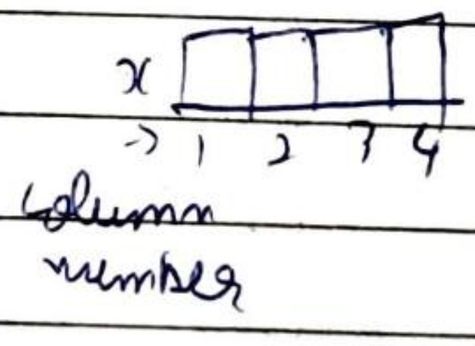
⇒ The problem is to place n queens on a n×n chessboard so that no two queens "attack".

- By "attack" it means that no two of them are on the same row, column or diagonal.

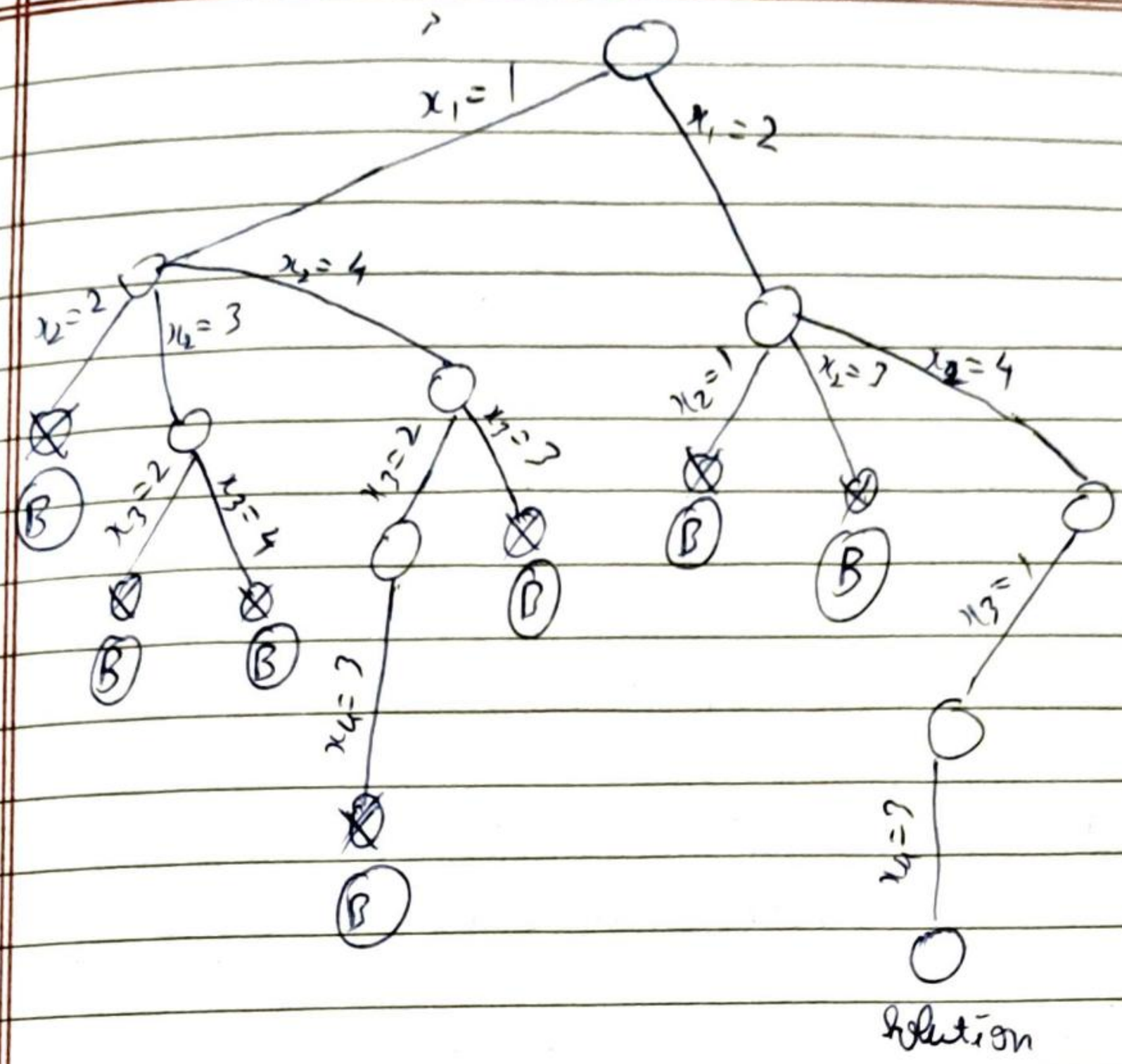
- We will use state space tree to find the solution for the n queen problem.
- Example for 4-queen
- There are some conditions need to be applied →
(i) We will place all 4 queens in different row.

- Deciding functions →
(i) no two queens can be placed in same row
(ii) no two queens can be placed in same column
(iii) no two queens can be placed diagonally.

	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	



queen | inlet column



x_s	2	4	1	3

Solution

column number

// merely we can find the other answers.

- Time complexity → $O(q(n)\, n!)$

  where $q(n)$ is polynomial in n.

Q. What is sum of subsets problem?

=>. Suppose we are given n distinct positive numbers & we have to find all combinations of these numbers whose sum are m.

The worst case time complexity →

$$O(p(n)\ 2^n)$$

where $p(n)$ is polynomial in n.

Q 3 Illustrate the bounding functions involved in solving sum of subsets problem using backtracking technique.

=> . The bounding functions for the sum of subset problem are →

$$B_k(x_1, \ldots, x_k) = true$$

iff

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \le m$$

&

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \ge m$$

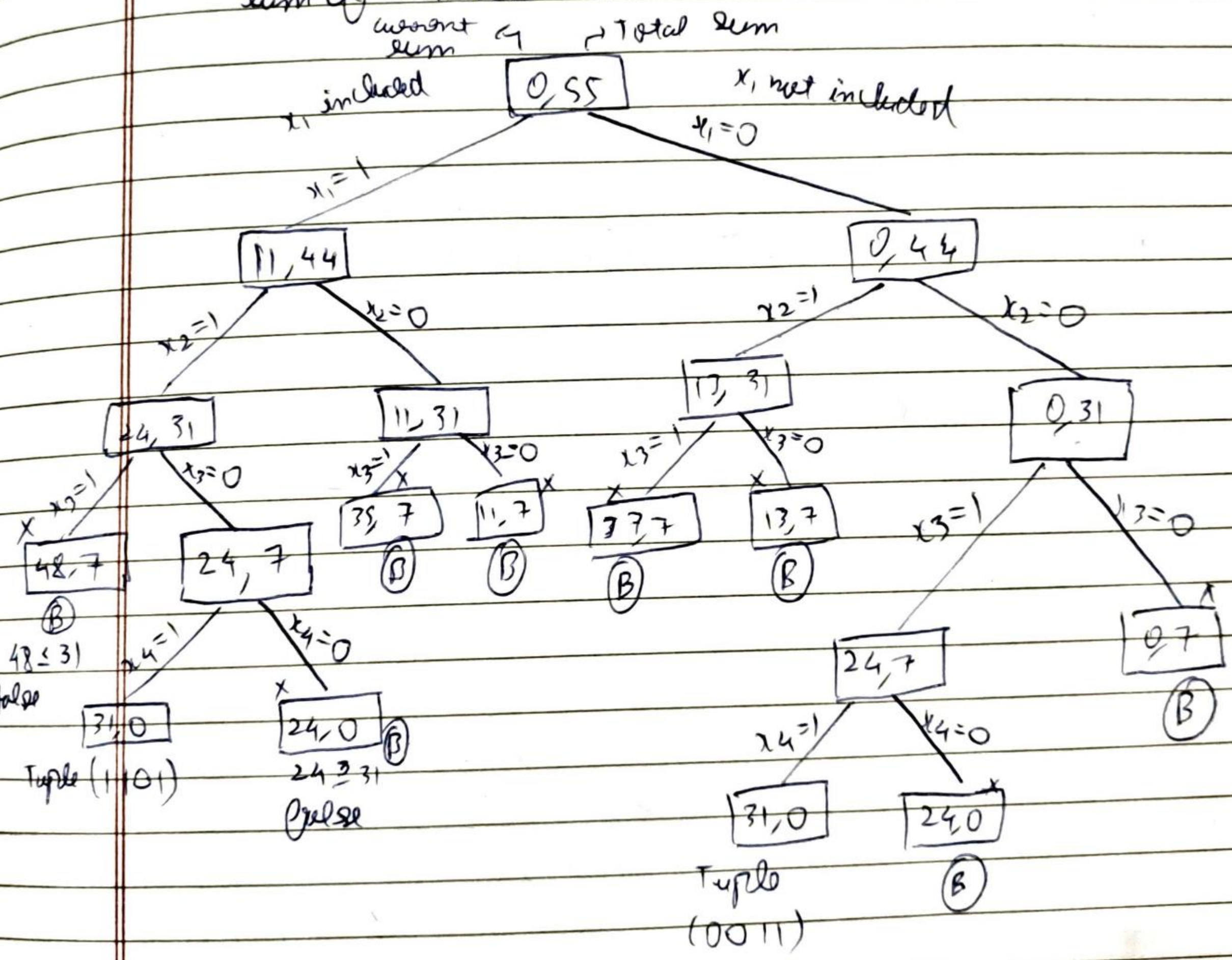
The first condition →

that when ~~now~~ next weight i.e. w_{k+1} is added to the already computed weights i.e. $\sum_{i=1}^{k} w_i x_i$ should be less than or equal to ~~m~~ m. It means it should not exceed m because m is the required sum.

The second condition →
that the sum computed till k i.e. $\sum_{i=1}^{k} w_i x_i$ when is added to the sum of remaining weights i.e. from (k+1) onwards i.e. $\sum_{i=k+1}^{n} w_i$ should be greater than or equal to m.

Q4 Solve sum of subsets problem using backtracking if the elements n = 4 are {11, 13, 24, 7} & the required sum is m = 31.

=> Elements = { x_1 11, x_2 13, x_3 24, x_4 7 } , m = 31
sum of elements = 55



Answer Tuples →

$$\boxed{\begin{array}{l} \text{Tuple 1} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ 1 & 1 & 0 & 1 \end{pmatrix} \\ \text{Tuple 2} = \begin{pmatrix} x_1 & x_2 & x_3 & x_5 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}}$$

Q- What is graph colouring problem?

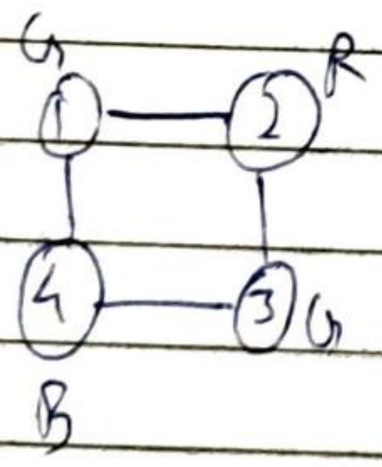
⇒. Let G be a graph & m be given positive integer.

- We want to discover whether the nodes of G can be coloured in such a way that no two adjacent nodes have the same colour yet only m ~~colo~~ colours are used. This is called m- colorability decision problem.

- The m- colorability optimization problem asks for the smallest integer m for which the graph G can be coloured.

- Worst case time complexity → $\boxed{O(n m^n)}$

- For example

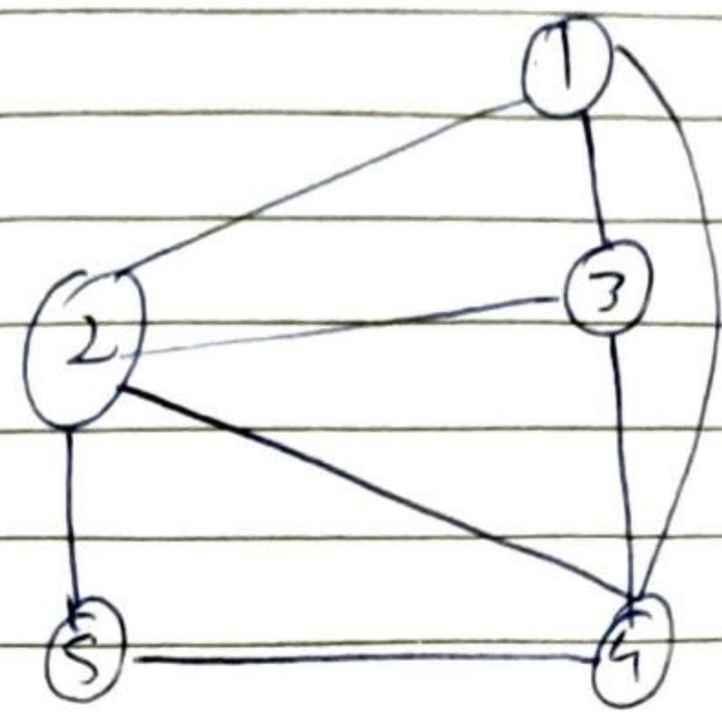


$n = 4$

$m = 3$ (R, G, B)

∴ 3 colours are required in this graph to colour it.

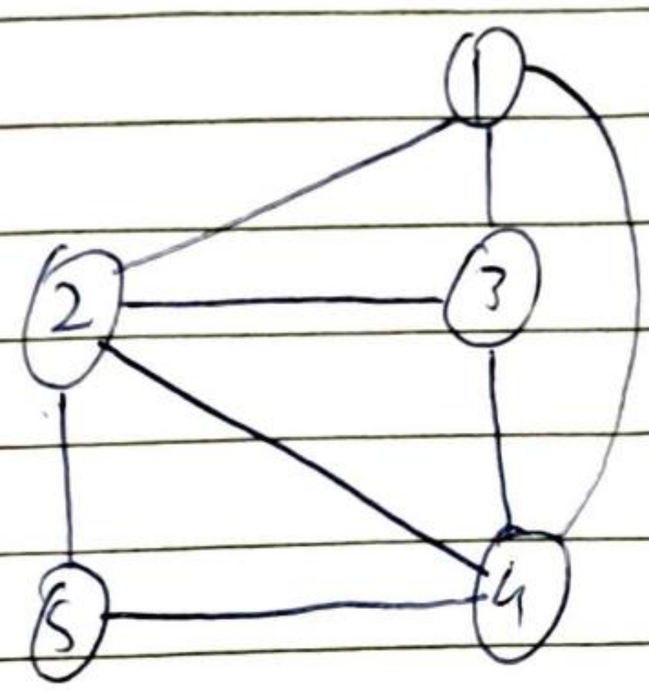
Q5 Demonstrate graph coloring problem for →



$n = 5$ (no. of vertices of graph G)

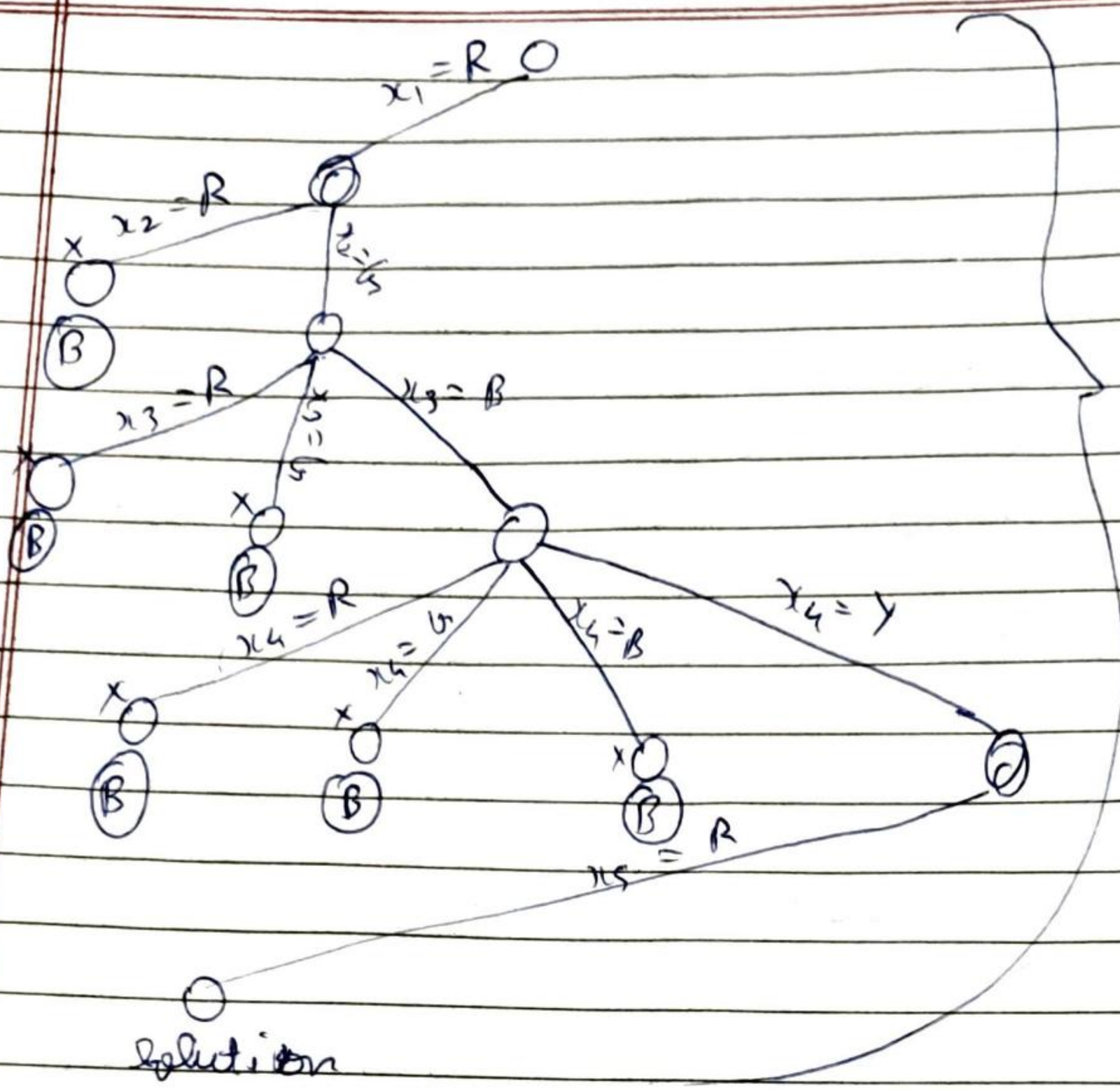
Compute the req. no of colors

=>



$m = 4$ are required to color this graph.

- We will use backtracking to generate state space tree.
- Bounding function which we will use is that no two adjacent nodes should have same color.
- Let color 1 = R, color 2 = G, color 3 = B, color 4 = Y



// only all other nodes are also generated.

· Solution I →

① → color 1, ② → color 2, ③ → color 3

④ → color 4, ⑤ → color 1

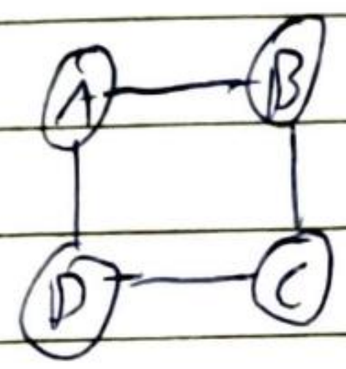
Q. What is Hamiltonian cycle?

=> Let G (V,E) be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once & returns to its starting point.

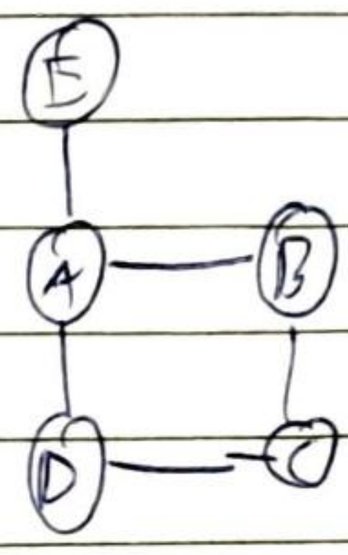
- Time complexity is O(n!)

- The graph may be directed or undirected

- Ex



It is a Hamiltonian cycle



It is not a Hamiltonian cycle

Q7 Appraise the importance of using greedy method & relaxing the condition of $x_i = 0$ or 1 to $0 \le x_i \le 1$ when computing optimal solution for 0/1 knapsack problem in a recursive backtracking algorithm.

=> The knapsack problem computes optimal solution of

$\sum_{1 \le i \le n} w_i x_i \le m$ & $\sum_{1 \le i \le n} p_i x_i$ is maximized

x_i is either 0 or 1

The recursive backtracking algorithm runs on the basis of bounding function (B) as recursive function.

Bound function (int k, float cp, float cw)

where k is index of last element
cp is current profit total
cw is current weight total

Steps for solving 0/1 knapsack using recursive backtracking:

Step 1:- Arrange the items in decreasing order of their p_i / w_i

Step 2:- Find the upper bound with the help of greedy approach.

Step 3:- Generate state space tree

Step 4: Expand the state space tree to upper bound calculated

For example :-

	x_1	x_2	x_3	x_4
p	45	30	45	10
w	3	5	9	5
p/w	15	6	5	2

$\boxed{m = 16}$

Upper Bound →

Step I: Arrange all to p_i/w_i decreasing order:

$p_1/w_1 \Rightarrow p_2/w_2 > p_3/w_3 > p_4/w_4$

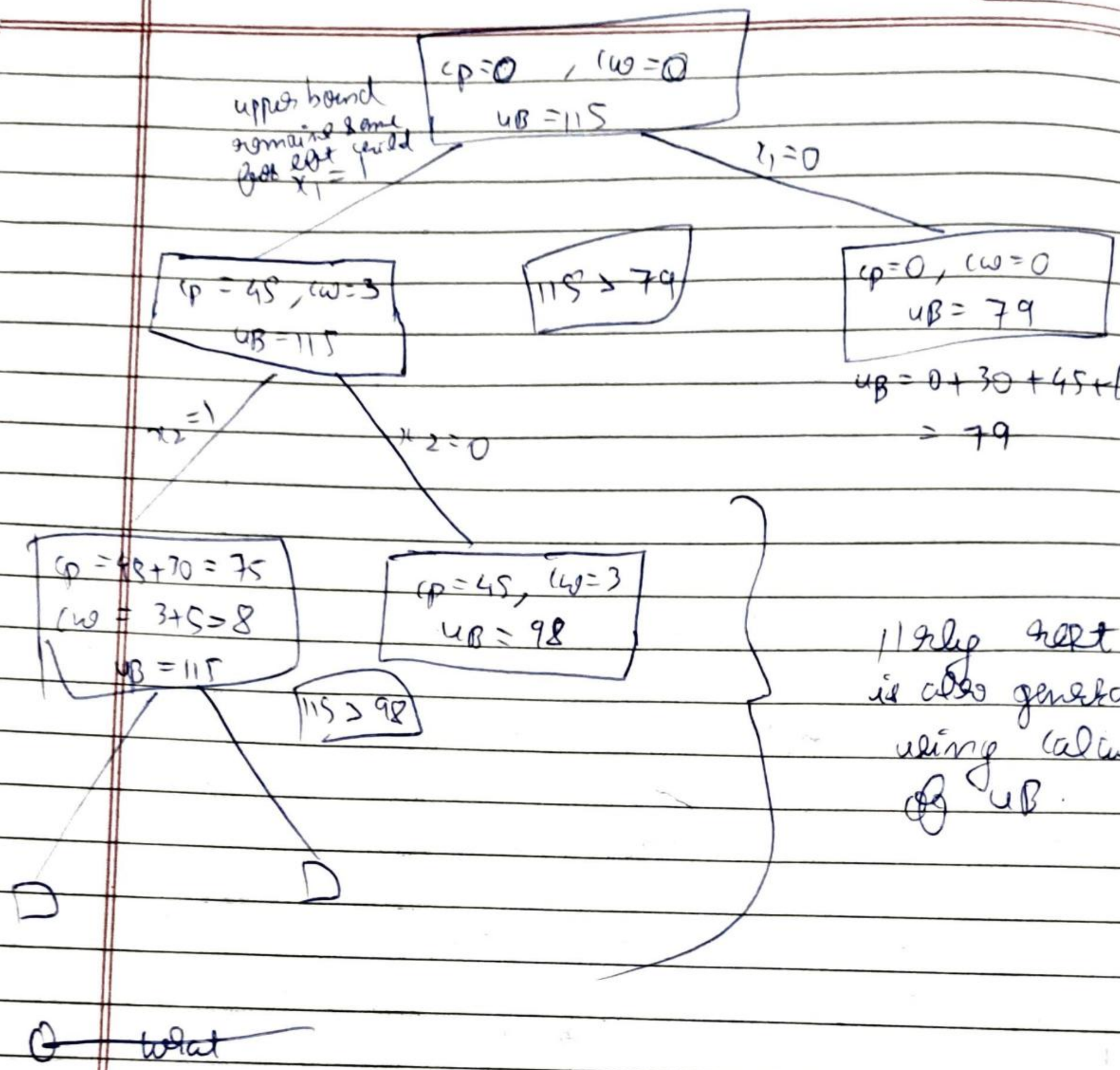
Step 2:- Calculate upper bound :-

	x_1	x_2	x_3	
p	45	30	$8/9 \times 45 = 40$	$= 115$
w	3	5	$8/9 \times 9 = 8$	$= 16$

Step 3:-

Generate State Space Tree :-

PTO


cp = 0 , cw = 0
uB = 115
upper bound remains same, but left child $x_1 = 1$
$x_1 = 0$
cp = 45 , cw = 3
uB = 115
115 > 79
cp = 0 , cw = 0
uB = 79
uB = 0 + 30 + 45 + (10 × 2/5)
= 79
$x_2 = 1$
$x_2 = 0$
cp = 45 + 30 = 75
cw = 3 + 5 = 8
uB = 115
cp = 45 , cw = 3
uB = 98
115 > 98
// right child is also generated using calculation of uB.

what

2015 (5) (any one algo)

Q What is string matching algorithm?

=> . The string matching problem →

It is the problem of detecting the occurance of a particular substring called pattern, in another string called the text.

The algorithms used are →

1. Naive / Brute Force Pattern matching.
2. Rabin Karp Algorithm
3. Boyer-Moore Algorithm
4. Knuth-Morris-Pratt (KMP) Algorithm

Text T [1 ... n]

P → pattern to be found

P [1 ... m]

Q- Explain ~~naive~~ naive string / Brute Force Pattern matching

=> . In this comparisons are done (in left to right order) on the ~~par~~ pairs of characters in array P & T.

When first mismatch occurs, the pattern is moved one position forward with respect to text, & comparisons start again at the left end of pattern.

Steps:

1. Arrange P from start of T & start comparison left to right.
2. When a mismatch occurs, shift P ~~for~~ by one position & start again. Do this till the pattern P is matched.

- Time complexity → $O(n^2)$

Q9 Illustrate the working of Rabin Karp algorithm for string matching.

⇒ Suppose there is →
T is text T [1...n]
P is pattern [1...m]

- In this algo, first a calculation is done before doing exact comparisons of characters. ~~This A function~~
- A function called hash function is calculated.
- The hash function is calculated for the pattern.

Ex

P = B A A

$2 \times 26^2 \quad 1 \times 26^1 \quad 1 \times 26^0$

- Hash function = $\boxed{p \bmod q}$
- Strong hash function

q is any prime +ve integer. Higher the number, less chance of collision i.e. called superior hit or false / false hit.

- After calculating the hash function, find the submission of p mod q from the text T of first m characters
- If the value matches, then compare the two strings. And if the string matches, then the substring is found.

- If the value doesn't match then roll to the next window.

- Ex → T : 2 3 5 9 0 4 2
  p = 42
  q = 5

First compute
$h = p \bmod q = 42 \bmod 5 = 2$

$S_1 = 2\ 3$
$(S_1 \bmod q)$
$\Rightarrow 23 \bmod 5$
$\Rightarrow 3$

not same as h
∴ roll to next window
Similarly we will apply the same till the pattern is matched.

- Worst case Time complexity $\boxed{O(mn)}$

Q10 Examine how Boyer moore algorithm is working efficiently as string matching algorithm.

=>. Boyer moore algorithm is a string searching or matching algorithm. It is widely used & the most efficient string matching algorithm.

- This algorithm takes a backward approach i.e. the pattern string (P) is aligned with the start of the text string (T) & then compares the characters of a pattern from right to left, beginning with right most character.

- Boyer moore algorithm is the combination of 2 approaches i.e. bad character heuristic & good suffix heuristic.

* ① Bad character Heuristic ->

The bad character rules says upon mismatch, skip the alignments until :-

(a) mismatch becomes a match (align the character of text T & pattern P that has caused mismatch)

OR

(b) P moves past mismatched character.

(i.e. crosses the character ~~all~~ altogether)

Ex

Bad character

I-> T: G C T T C T G C T A C T T T T

P: C C T T T T G C

∴ check if there is another C in P & align them. If not then move past the mismatch

II → T: GCTTCTGCTACCTTTT
P: CCTTTTGG

// money other steps are performed.

(2) good suffix heuristic →

The good suffix heuristic says that for the characters which did match, we want to shift P such that we don't turn any three matches into a mismatch at end of that shift.

~~Let t be substring matched between P & t~~

Let t be substring matched by inner loop.

Skip until :

(a) there are no mismatches between P & t
(i.e. old matches are still maintained to main extend)

or

(b) P moves past t
(moves altogether)

Ex

I → T: CGTGCCTACTTACTTACTTAC
P: CTTACTTAC
mismatch

II → T: CGTGCCTACTTACTTACTTAC
P: CTTACCTTAC
matched

// merely all other steps are performed

- In practice, we use both bad character heuristic & good suffix heuristic, & apply that method that gives maximum number of alignments / placements.

- Time complexity by using both heuristics

$$\boxed{O(n+m)}$$

Q. What is Boyer Moore Horspool Algorithm?

⇒ • The Boyer Moore Horspool algorithm were able to give the average time complexity $= O(n)$ which is linear & is independent of the size of the pattern m.

- But its worst case is $O(nm)$ which is same as that of brute force algorithm

- It includes the making of 'Bad match Table', compare pattern to text, starting from right most character in the pattern. If mismatch, move pattern forward corresponding to value in the bad match table

- Construction of Bad match Table →
  Value = length − index − 1
  and keep last letter's value = length
  and every other letter's value = length

· EX

T: tarutarad toothbrusher
P: tooth

TOOTH
Index 0 1 2 3 4

Letter	T	O	H	* (Any other)
Value	~~4~~ 1	~~3~~ 2	5	5

For T (0)
value = length - index - 1
= 5 - 0 - 1
= 4

For O (1)
value = length - index - 1
= 5 - 1 - 1
= 3

For O (2)
value = length - index - 1
= 5 - 2 - 1
= 2

For T (3)
value = length - index - 1
= 5 - 3 - 1
= 1

For H (4)
as last letter
∴ keep the value as length

For any other
keep the value as length

Table->

Letter	T	O	H	T
Value	1	2	5	5

Step I ->

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

mismatch

Since T is compared and its value is 1, ∴ move forward by 1

Step 2->

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

mismatch

Since S is mismatch, but the first character compared was H, ∴ move forward by value of H i.e. 5

// Similarly we will perform other steps.

Q11 Demonstrate the working of Knuth-morris Prath algorithm (KMP).

=> • The KMP algorithm avoids the waste of information in case a pattern character does not match in the text.

• Thus it achieves running time of O(m+n)

• The main idea of KMP algo is to ~~prepare~~ preprocess the pattern string P so as to compute a failure function that indicate the proper shift of P.

• This helps to the largest extent possible, we can reuse previously performed comparisons

• We compute a π table of the P.

• Ex :

T: abacaabaccabacabaabb

P: abacab

• Prefix function or π table

i =	1	2	3	4	5	6
a	b	a	c	a	b	P
0	0	1	0	1	2	

• Let we consider i in start of T & j at 0th position of P.

- Steps to perform →
(a) compare T[i] with P[j+1]
(b) If the comparision is same, we increment both i & j.
(c) If the comparison is not same, then move j to position that is the value of the π table.
(d) now compare T[i] with the ~~element~~ current element
(e) If j reaches at 0, then increment i.

- By following the steps, we can get the solution

Q- What is Polynomial Time Problems?

⇒ There are some problems where best-known algorithms are Polynomial.
eg. Sorting algorithm ⇒ $O(n \log n)$

Q- Exponential problems / non-polynomial problems?

⇒ There are some problems where best-known algorithms are non polynomial i.e. exponential in nature.
eg. sum of subset using backtracking
⇒ $O(q(n)\, 2^n)$

Q What is ~~[illegible]~~ deterministic algo?

⇒ The algorithm which has the property that the result of every operation is uniquely defined is called as deterministic algorithm.

2015(2)

Q What is non deterministic algorithm?

⇒ If an algorithm contains operations whose outcomes are not uniquely defined but are limited to specific sets of possibilities is called as non-deterministic algorithm.

- In such algorithms, a choice is made out of a set of elements. This choice is arbitrary & has no specific rule. The choice may lead to a successful completion or a failure of the algorithm. Thus its output is not unique.

- For example →

```
Algorithm Nsearch (A, n, key) {

    int j = choice (1,n)
    if (A[j] == x)
    {
        cout << j;
        success();          // display successful
    }                          completion

    else
    {
        cout << '0';
        failure();          // to declare
    }                          unsuccessful
                               completion
}
```

Q What is Decision Problem / Decision Algorithm?

=> Any problem for which the answer is either yes or one (or true / false or yes / no) is called as a decision problem.
An algorithm for a decision problem is known as a decision algorithm.

Ex n-queen problem.

Q What is Optimization Problem / Optimization Algorithm?

=> Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem.

Ex- Knapsack problem.

Q What is Conjunctive Normal Form (CNF)?

=> A formula is in conjunctive normal form (CNF) if it is represented as $\bigwedge_{i=1}^{k} C_i$

(where C_i are clauses each represented as $\vee$ of boolean variables x_i or their negations $\bar{x}_i$)

Example->

$(x_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_4)$ is in CNF

Q What is Disjunctive normal Form (DNF)?

=> A formula is in disjunctive normal Form (DNF) iff it is represented as $\bigvee_{i=1}^{n} C_i$

(where C_i are clauses each represented as $\wedge$ of boolean variables x_i or their negations $\bar{x}_i$)

Example->

$(x_2 \wedge \bar{x}_1) \vee (x_3 \wedge x_4)$ is in DNF

Q What is satisfiability problem?

=> Looking truth values of boolean variables & verifying that the formula is true or not for that assignment is called as satisfiability problem.

Q What is P class problem?

=> P is the set of all decision problems solvable by deterministic algorithms in polynomial time.

Q What is NP class problem?

=> NP is the set of all decision problems solvable by non-deterministic algorithms in polynomial time.

Q16 Examine the relationship b/w P & NP problem classes.

=> • P is subset of NP (any problem that can be solved by deterministic machine in polynomial time can also be solved by non-deterministic machine in polynomial time). But we have to note that $P \neq NP$

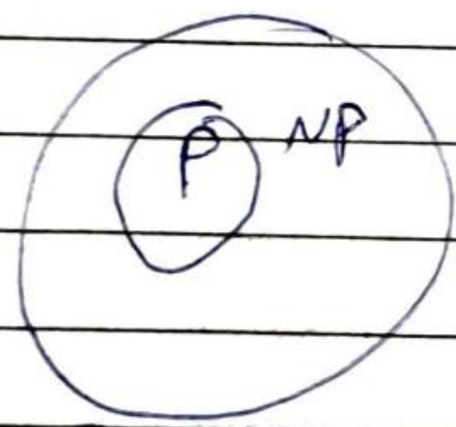
• NP is set of problems that can be solved by a non-deterministic ~~tes~~ machine in polynomial time.

• Since deterministic algorithms are just a special case of non-deterministic ones, we conclude that

$$P \subseteq NP$$

$\therefore$ P is proper subset of NP

•



$$P \subseteq NP$$

2015(2)

Q 20 Explain satisfiability.

=). Satisfiability is in P iff P = NP

P = NP

satisfiability is in P

- The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables

- It is having 2^n possible assignments of truth values of variables. True is exponential in nature.

- But it can be solved by a non-deterministic algorithm in polynomial time by choosing the values (& not trying all possibilities)

- It is a decision problem (True / False), can be solved by non-deterministic algorithm in polynomial time, so we can say that satisfiability is in NP problem class.

- But if the non-deterministic algo has only one choice, producing a uniquely defined output, then it become a deterministic algo.

i.e. P = NP

- Then, satisfiability problem (a decision problem) is being solved by a deterministic algo in polynomial time.

- This implies that, now
  satisfiability is in P problem class

Thus proved cook's th^m.

2015(2)

Q- Rel^n b/w NP, NP hard, NP complete.

PAGE :
DATE: / /

Q. 0 1 Knapsack problem using Dynamic Programming.

⇒. In this problem, there is knapsack like a container or a bag.

- Suppose we have given some items which have some weights or profits.
- We have to put some items in knapsack in such a way total value produce a maximum profit.
- The 0 1 knapsack problem means that the items are either completely or no items are filled in a knapsack.

- While using dynamic programming we have to :

$$f_n(m) = \max \{ \underbrace{f_{n-1}(m)}_{term\ 1}, \underbrace{f_{n-1}(m - w_n) + P_n}_{term\ 2} \}$$

where term 1 represents not including element, term 2 represents including element.
We have to find the maximum of the both.

- We will generate ordered set

PAGE :
DATE: / /

Ex $P = \{1, 2, 5, 6\}$ $m = 8$

$w = \{2, 3, 4, 5\}$ $n = 4$

(P, w)

1st object

$S^0 = \{(0,0)\}$

$S_1^0 = \{(1,2)\}$

$S^1 = \{(0,0)\ (1,2)\}$ → 1st included

1st not included

Add (2,3) from second obj

$S_1^1 = \{(2,3)\ (3,5)\}$

$S^2 = \{(0,0)\ (1,2)\ (2,3)\ (3,5)\}$

no obj included ← 1st obj included ↲ second obj is included ↲ all objects are included

capacity of knapsack

$S_1^2 = \{(5,4)\ (6,6)\ (7,7)\ \cancel{(8,9)}\}$

$S^3 = \{(0,0)\ (1,2)\ (2,3)\ \cancel{(3,5)}\ (5,4)\ (6,6)\ (7,7)\}$

↓ because profit inc, weight also inc, but after 5 is 4. This discarded because it had less profit

$S_1^3 = \{(6,5)\ (7,7)\ (8,8)\ (4,9)\ (12,11)\ (13,12)\}$

max

$S^4 = \{(0,0)\ (1,2)\ (2,3)\ (5,4)\ \cancel{(6,6)}\ (6,5)\ (7,7)\ (8,8)\ \cancel{(11,9)}\ \cancel{(12,11)}\ \cancel{(13,12)}\}$

PAGE :
DATE: / /

max^m is $(8,8)$

- $(8,8) \in S^4$

(1) but $(8,8) \notin S^3 \therefore x_4 = 1$

- now sub profit & weight of 4th obj

$(8-6, 8-5) = (2,3)$

(2) $(2,3) \in S^3$ $x_3 = 0$
& $(2,3) \in S^2 \therefore$

(3) $(2,3) \in S^2$
but $(2,3) \notin S^1 \therefore x_2 = 1$

$(2-2, 3-3) = (0,0)$

(4) $(0,0) \notin S^1$
& $(0,0) \notin S^0 \therefore x_1 = 0$

$\boxed{\text{Solution} \Rightarrow \{0, 1, 0, 1\}}$

- Time complexity $\rightarrow$ $\boxed{O(2^n)}$