Module - 5

Backtracking

Nithin Kumar
Asst prof Dept of CSE
VVCE, Mysuru

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

* General Method of Backtracking → In order to applying backtracking to a specific class of problems, one must provide the data "P" for the particular instance of the problem that is to be solved & six procedural parameters. using State-Space Tree.

→ These procedures should take the instance data P as a parameter & should do the following

* Root(p) → return the partial candidate at the root of search tree
* Reject(p,c) → return true only if the partial candidate c is not worth completing.
* Accept(p,c) → return true, if c is soln of problem, & false otherwise
* First(p,c) → generate the first extension of candidate c.
* Next(p,s) → generate the next alternative extension of a candidate, after the extension s.
* Output(p,c) → use the solution c of p, as appropriate to the application.

Here, we will discuss three problems using Backtracking.

* Subset - Sum problem
* N - Queen's problem
* Hamiltonian Cycle problem

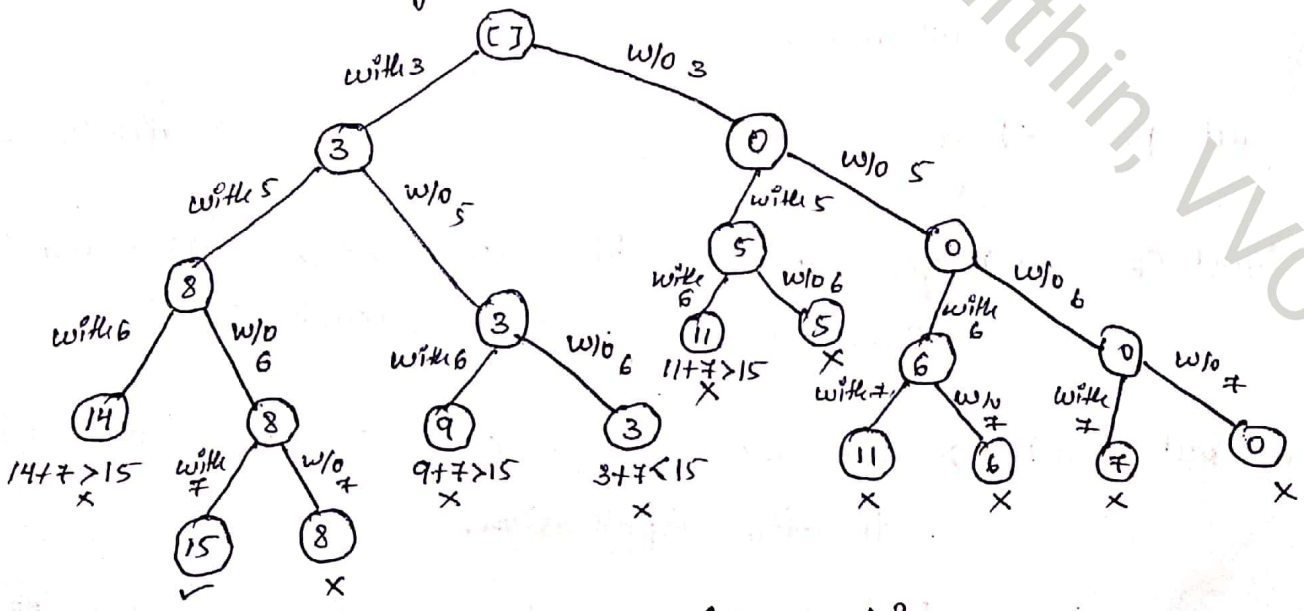
* Subset - Sum problem :- The task is to find a subset of a given set whose elements add-up to a given integer, that is 'd'.

→ Steps to Solve the given Sum-of-Subset problem
* Initial set only contains the Empty set.
* we loop through Each element in the Array & Add it to Every element in power set
* we check to see if the Sum Equals to our goal 'd'.

* Consider the given $S = \{3, 5, 6, 7\}$ & $d = 15$, find the subset for given 'S' using State-Space tree

=> Firstly check the Elements in the given set S Must be Sorted
* Value of first element in S = 3 Must be less than d
* Sum of all elements in S Must be greater than d

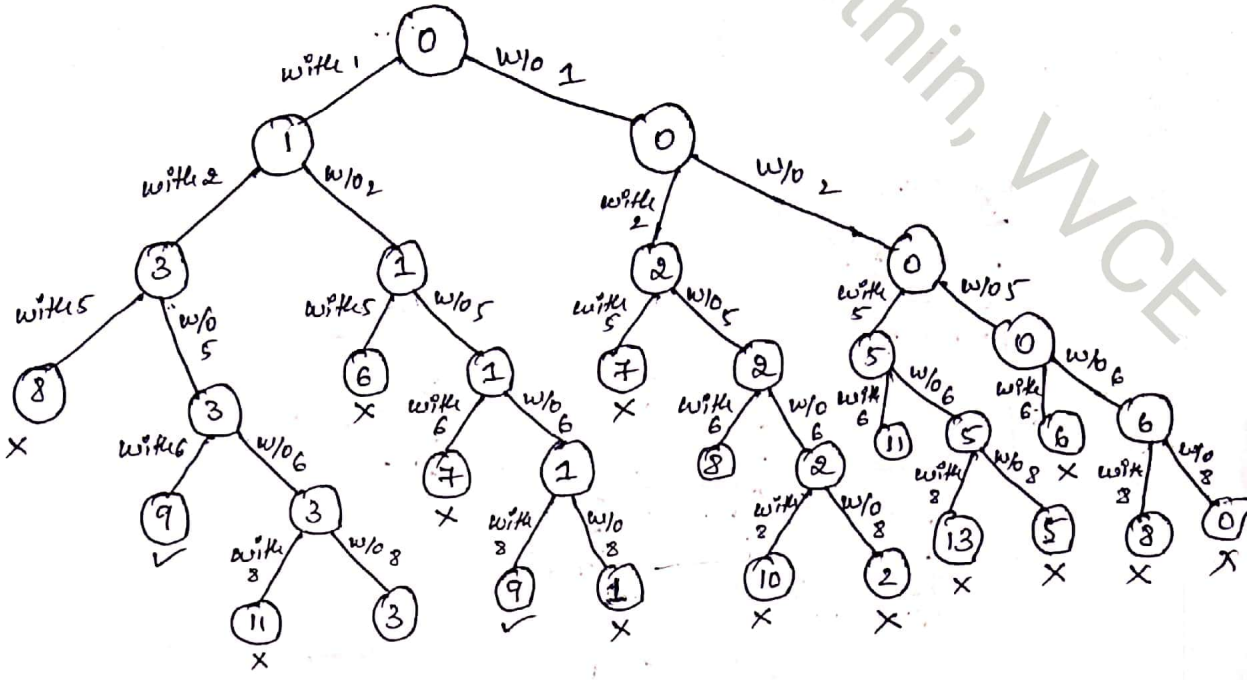
→ Start Constructing the State-Space tree



∴ The Solution is $\{3, 5, 7\}$ //

* Given $S = \{1, 2, 5, 6, 8\}$ & $d = 9$ find the Subsets of S with 9, State-Space tree

⇒ Firstly, Add all elements '22' should be greater than 'd' & first Elemt $S[1] = 1$ should be less than '9'



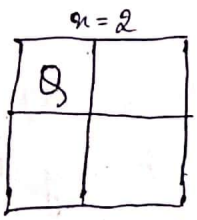
∴ The Solution is $\{1, 2, 6\}$ & $\{1, 8\}$ //

* N-Queen's problem :-

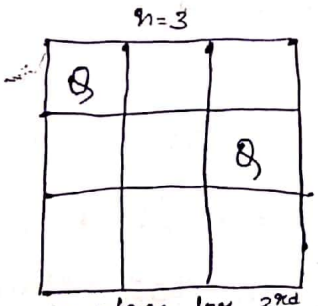
The problem is to place 'n' queens on a $n \times n$ Chessboard so that no two queens attack Each other by being in the Same row, Same Column or on Same diagonal

* For $n = 1$, the Solution would be [Q]

* For $n = 2$ & $n = 3$, there is no Solution because

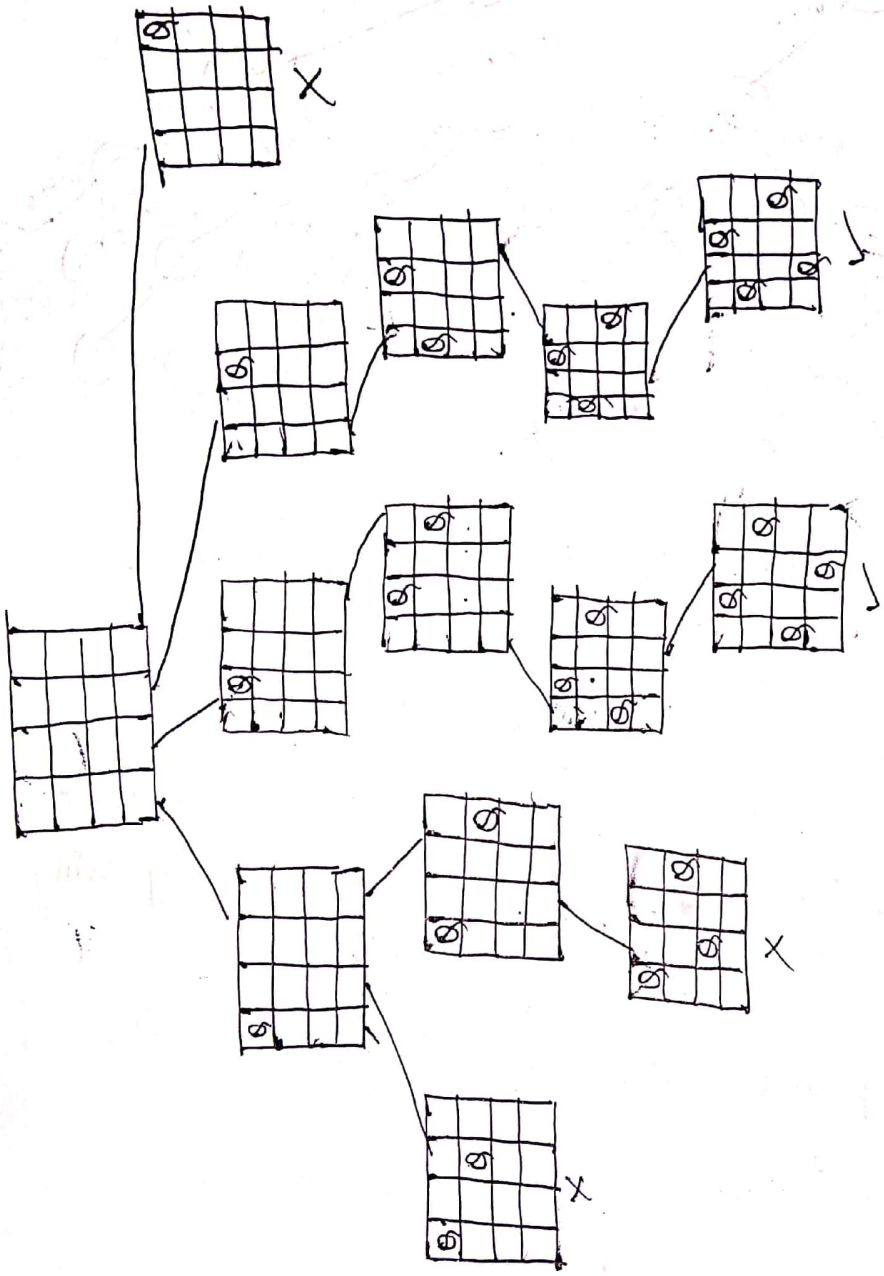


No place for 2nd queen



No place for 3rd queen

* For $n=4$, there are two solutions as shown below

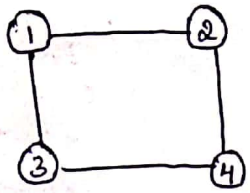


(4 - Queens problem)

* Hamiltonian Cycle :-

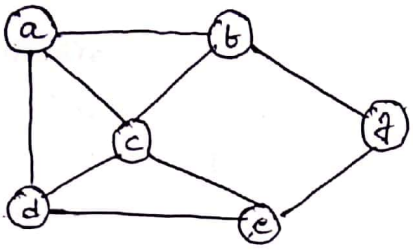
Hamiltonian Cycle is a closed loop on a graph, where Every node (vertex) is visited Exactly once, so a Hamiltonian Cycle is a path traveling from a point back to itself, visiting Every node En route.

→ Consider the following undirected graph, where '1' is source

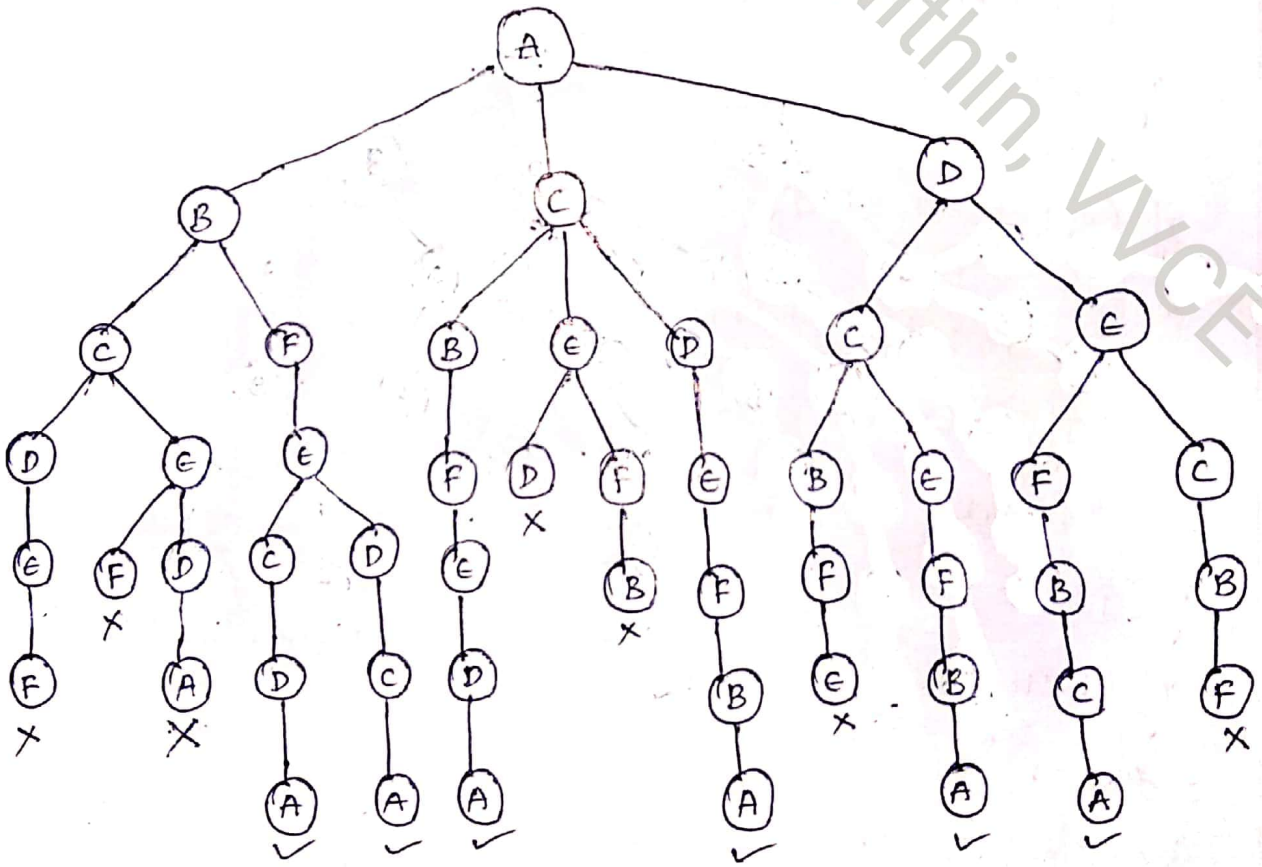


the Hamiltonian cycles are 1-2-4-3-1
1-3-4-2-1

* Apply Backtracking to the problem of finding a Hamiltonian circuit in the graph shown below



⇒ Firstly, Construct a Space-tree for all reachable vertices from each vertex



∴ Hamilton cycles are A-B-F-E-C-D-A, A-C-B-F-E-D-A, A-D-C-E-F-B-A
A-B-F-E-D-C-A, A-C-D-E-F-B-A, A-D-E-F-B-C-A //

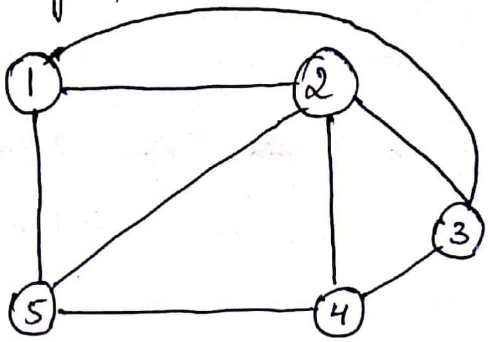
* Algorithm Hamiltonian(k)

```
{
  do
  {
    NextVertex (k);
    if (x[k] == 0)
        return;
    if (k == n)
        print (x[1:n]);
    Else
        Hamiltonian (k+1);
  } while (true);
}
```

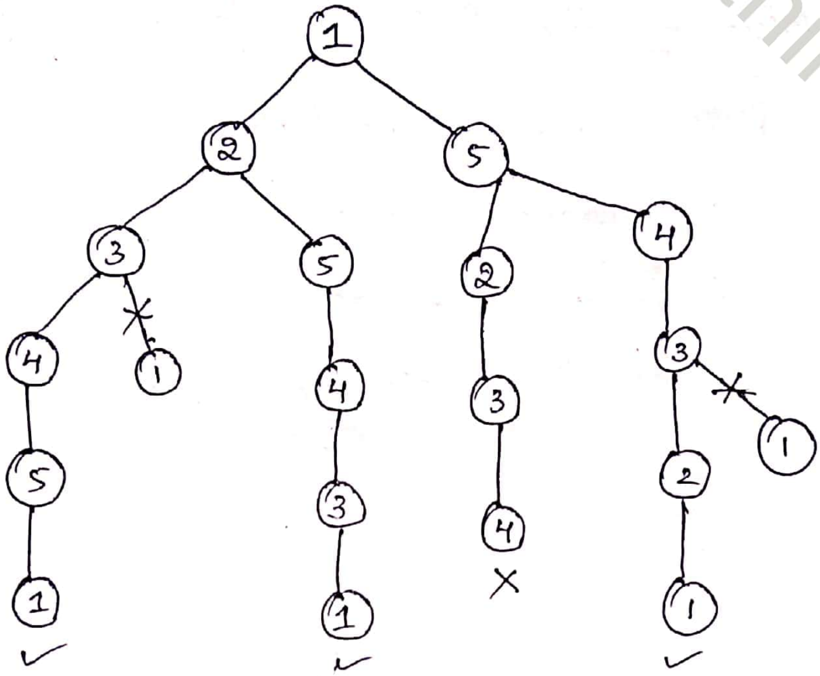
* Next Vertex (k)

```
{
  do
  {
   x[k] = (x[k]+1) % (n+1);
   if (x[k] == 0)
     return;
   if (er[x[k-1], x[k]] != 0)
   {
     for j = 1 to k-1 do
       if (x[j] == x[k])
         Break;
   if (j == k)
   if (k<n or (k==n) && er[x[n], x[1]] != 0)
     return;
  } while (true);
}
```

* Apply Backtracking to the problem of finding a Hamiltonian Circuit in the graph shown below



⇒



∴ The Hamiltonian circuits present are

1 → 2 → 3 → 4 → 5 → 1

1 → 2 → 5 → 4 → 3 → 1

1 → 5 → 4 → 3 → 2 → 1

* Branch & Bound :- is an improved version of Backtracking. here, we deal with optimization (Minimization or Maximization) problem.

→ A "feasible solution" is the one which satisfies the constraints of the problem.
For Ex, a subset of items whose total weight do not exceed the capacity of knapsack.

An "Optimal Solution" is a feasible Solution with the best valu

→ Branch-and-Bound requires two additional items Compared with backtracking

* For Every node of a State Space tree, a way to provide a bound (Either lower or upper) on the best value of the objective function.
* The value of the Best Soln found so far.

→ Here we will discuss 3 problems

* Assignment problem
* Knapsack problem
* Travelling Salesman problem

* Assignment problem :-

Here, the problem is to Assign 'n' Jobs to 'n' people so that the total cost of the Assignment is Minimum.

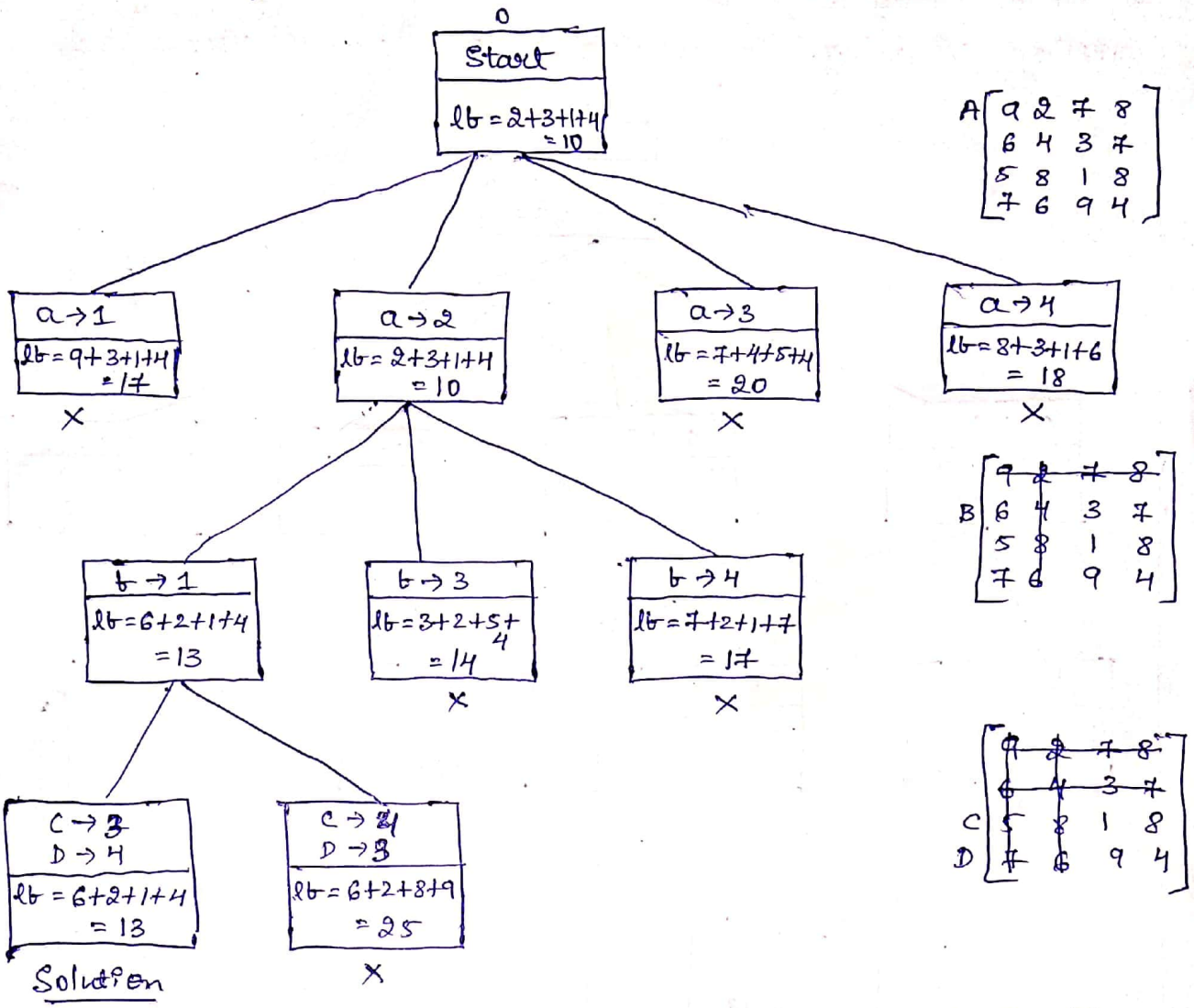
→ Consider the following Cost Matrix

Jobs / person	J_1	J_2	J_3	J_4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

⇒ Firstly, Compute the lower bound by adding the Smallest Elements in Every row.

* Lower Bound would be (lb) = 2 + 3 + 1 + 4 = 10 //

→ Now at Every Step, we have to keep finding Minimal Soln without Violating the Constraints of the problem



A
9	2	7	8
6	4	3	7
5	8	1	8
7	6	9	4

B
~~9~~	~~2~~	~~7~~	~~8~~
6	4	3	7
5	8	1	8
7	6	9	4

C
~~9~~	~~2~~	~~7~~	~~8~~
~~6~~	~~4~~	~~3~~	~~7~~
5	8	1	8
7	6	9	4

D
~~9~~	~~2~~	~~7~~	~~8~~
~~6~~	~~4~~	~~3~~	~~7~~
~~5~~	~~8~~	1	8
7	6	9	4

→ ∴ the cost of given Assignment problem is '13' & the Job Allocation would be

person A → Job 2
person B → Job 1
person C → Job 3
person D → Job 4 //

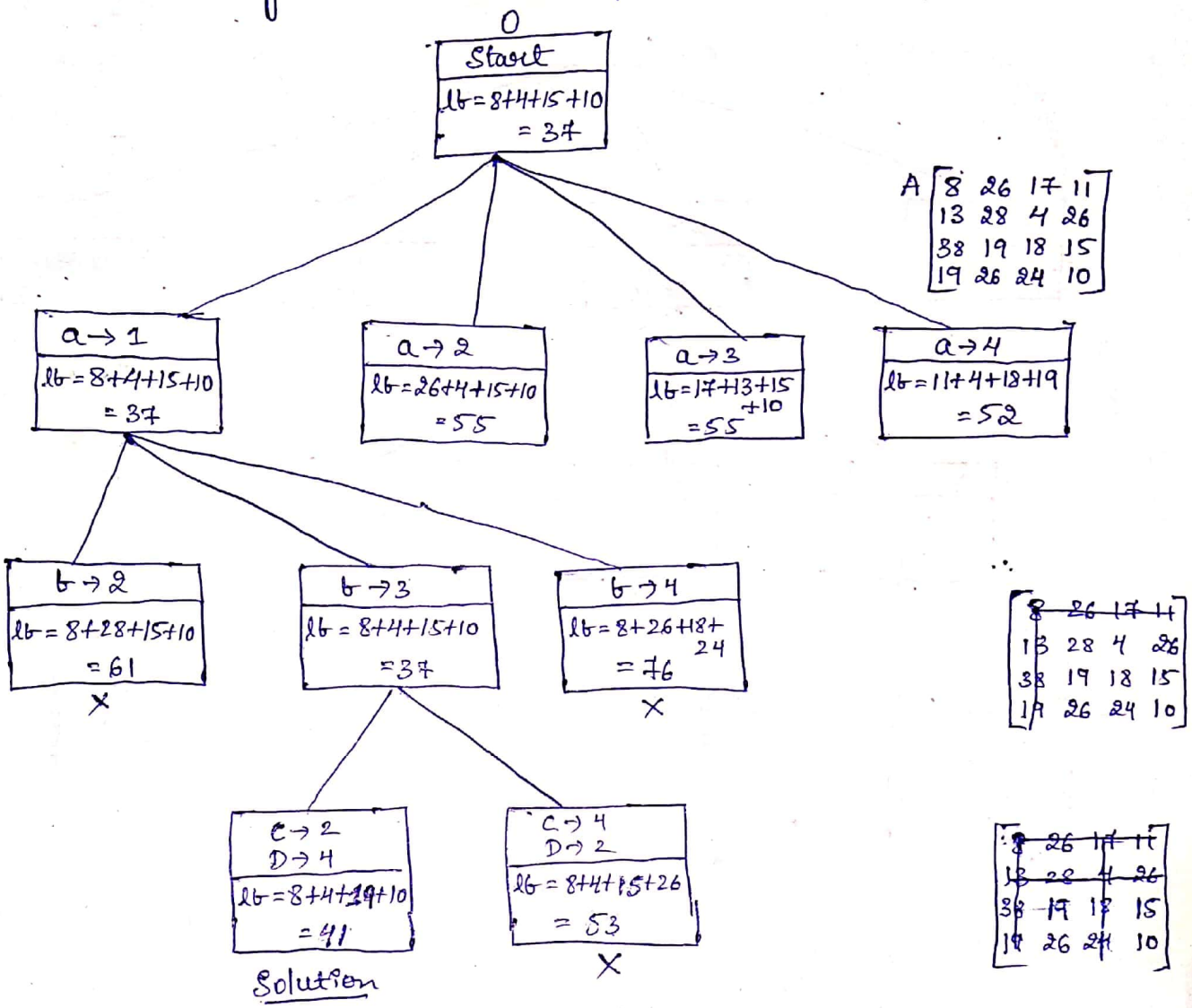
* Solve the given Assignment problem

8	26	17	11
13	28	4	26
38	19	18	15
19	26	24	10

⇒ Firstly, Compute the lower bound by adding the Smallest Elements in Every row

∴ lower bound (lb) = 8+4+15+10 = 37 //

→ Now at Every Step, we have to keep finding Minimal Soln without Violating the Constraints of the problem.



→ ∴ The cost of given Assignment problem is "41" & the Job Allocation would be

Person A → Job 1

Person B → Job 3

Person C → Job 2

Person D → Job 4 //

* 0/1 knapsack using Branch & Bound :-

There are n items with weights $w_1, w_2 \ldots w_n$ & values $v_1, v_2, \ldots v_n$. The total capacity of knapsack is W.

The problem is to maximize the total profit such that the total weight of the items is less than or equal to W.

→ we can either choose or not choose an item, we have $x_1, x_2 \ldots x_n$, where $x_i = \{1, 0\}$

* $x_i = 1$ i.e item is selected
* $x_i = 0$ i.e item is not selected

* Solve the following knapsack problem: capacity W = 10 & weights are {4, 7, 5, 3} & values are {40, 42, 25, 12}

⇒ Firstly, compute v_i/w_i (ratio) & place it in decreasing order

Item	Weight	Value	v_i/w_i
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

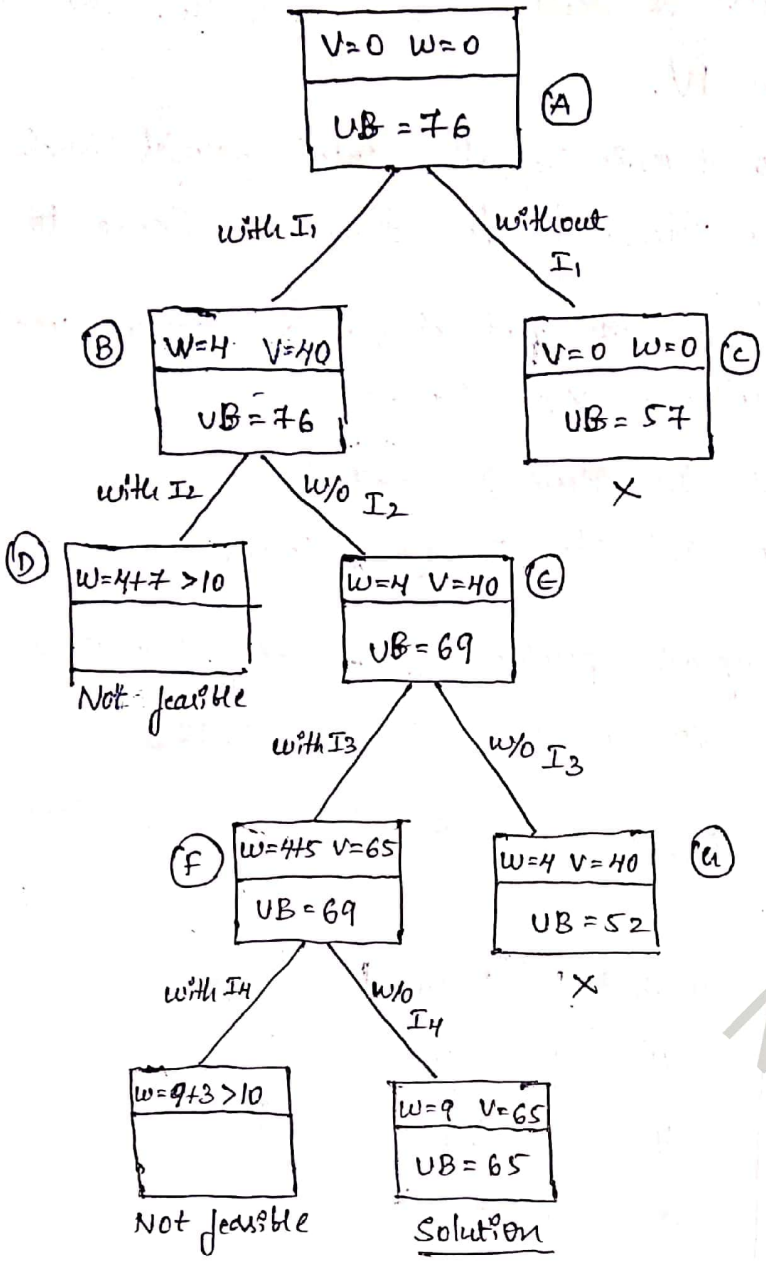
→ Next, Apply Greedy Approach to calculate Maximum profit i.e Upper Bound (UB)

Items $I_1 \rightarrow I_2$
W $4 + 6$

∴ Upper Bound (UB) = $40 + 6 * 6$

= 76 //

→ Next, Draw a State-Space tree with V=0 & W=0 & Ub = 76 as root node



*Node(C) = 42 + 3*5
Ub = 57

* Node (E) = 40 + 25 + 1*4
Ub = 69

* Node (G) = 40 + 12
Ub = 52

∴ The total profit gained is '65' //
Items added are {I_1, I_3} //

* Solve the given knapsack problem using Branch & Bound V = {10, 10, 12, 18} W = {2, 4, 6, 9} & the Maximum capacity of knapsack is W = 15.

⇒ Firstly, Compute v_i/w_i (ratio) & place it in decreasing order

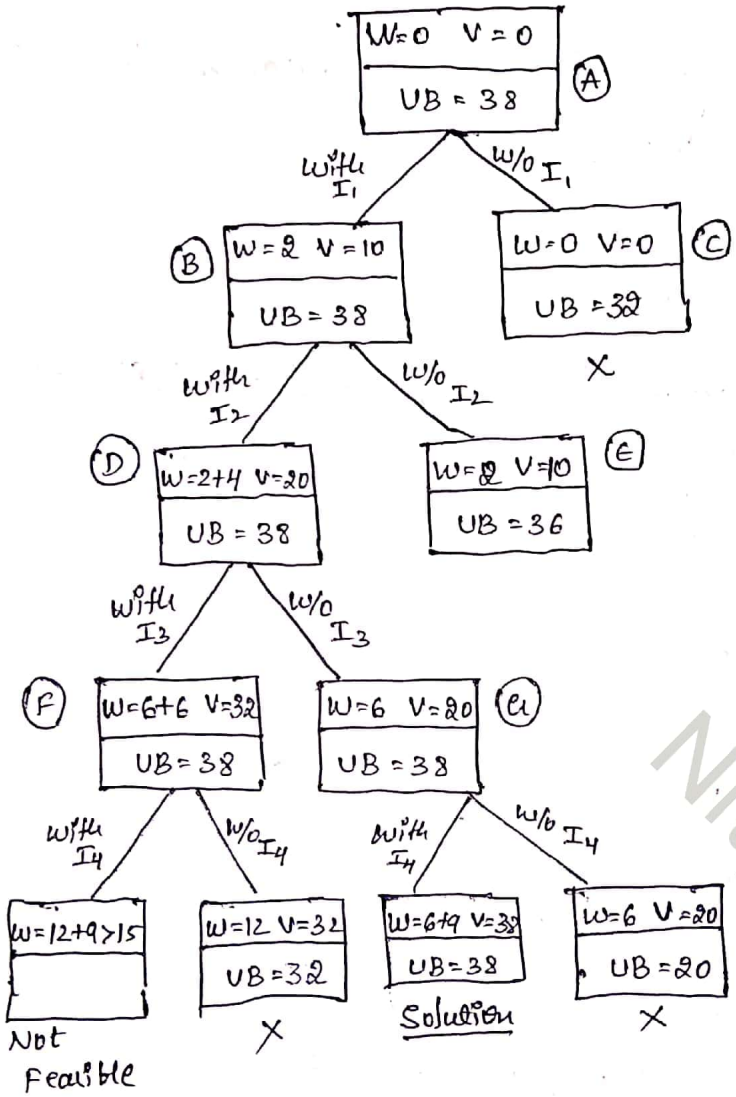
Item	Weight	Value	v_i/w_i
1	2	10	5
2	4	10	2.5
3	6	12	2
4	9	18	2

$W = 15$

→ Next, Apply Greedy Approach to calculate Upper Bound (UB)

$\therefore$ UpperBound $= 10 + 10 + 12 + 3 * 2 = 38$ //

→ Next, Draw a State-Space tree with $V=0$ & $W=0$ & $UB = 38$ as root



* Node (C) = $10 + 12 + 5 * 2$
UB = 32

* Node (E) = $10 + 12 + 7 * 2$
UB = 36

* Node (G) = $10 + 10 + 18$
UB = 38

$\therefore$ The total profit gained is "38" //
Items added are $\{I_1, I_2, I_4\}$ //

* Travelling Salesman problem using Branch-&-Bound (TSP)

TSP is repre

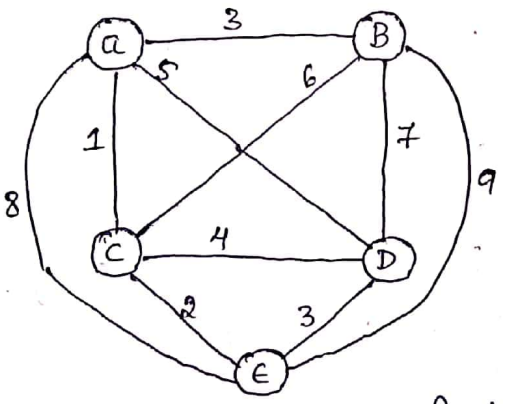
-sented by a Complete graph of N nodes. Here 'N' cities are Connected to Each other.

A Solution Should Start from one city (Home Town) & Visit Every City Exactly once to Comeback to his hometown. The travel path of the Salesman Should be of Minimum distance.

→ Computation of Lower-Bound for TSP using Branch-and-Bound technique Involves foll Steps

* For Each City i, find Sum S_i of distances from city i to two nearest cities.
* Compute Sum S for all cities
* Divide the result by 2
* If in case of fraction (result), Round-up (Ceil) the result to the Nearest Integer

* Solve the foll TSP using Branch-and-Bound



Src = A

⇒ Compute the Lower Bound (LB), Initially by choosing Min cost Edge from Each vertex & total Sum should be divided by 2

$$LB = [\underbrace{(1+3)}_{A} + \underbrace{(3+6)}_{B} + \underbrace{(1+2)}_{C} + \underbrace{(4+3)}_{D} + \underbrace{(2+3)}_{E}]/2 = 28/2 = 14$$

→ Next, Make calculated Lower Bound (LB) as root node & Start check -ing all possibilities.

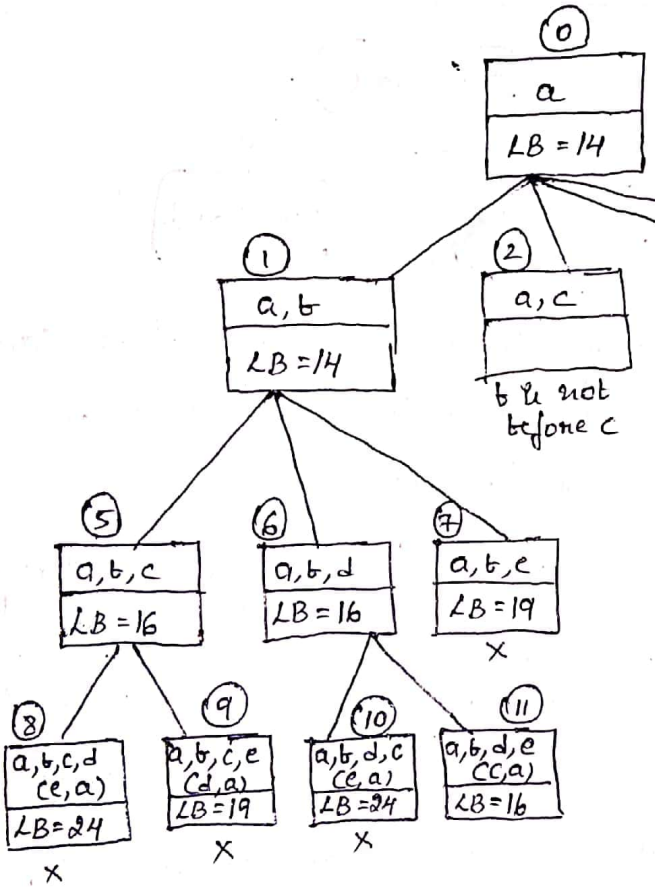
While checking the lower Bound (LB) of paths, keep the value of path, that you've considered same & for remaining vertices, you can choose Min cost Edges.

Ex: Consider path "ABC", Here the cost of vertices are

$\underbrace{(AB+min)}_{A} + \underbrace{(AB+AC)}_{B} + \underbrace{(BC+min)}_{C} + \underbrace{(min+min)}_{D} + \underbrace{(min+min)}_{E}$

Where "min" → min cost Edge from that vertex

→ The State-Space-tree is constructed with the calculated LowerBound = 14 & nodes are numbered Sequentially



* Node(1) = (3+1) + (3+6) + (1+2) + (4+3) + (2+3)/2
= 14

* Node(3) = (5+1) + (3+6) + (1+2) + (5+3) + (2+3)/2
= 15.5 (16)

* Node(4) = (8+1) + (3+6) + (1+2) + (4+3) + (8+2)/2
= 19

* Node(5) = (3+1) + (3+6) + (6+1) + (4+3) + (2+3)/2
= 16

* Node(6) = (3+1) + (3+7) + (1+2) + (7+3) + (2+3)/2
= 16

* Node(7) = (3+1) + (3+9) + (1+2) + (4+3) + (9+2)/2
= 19

* Node(8) = (8+3) + (3+6) + (6+4) + (4+3) + (3+8)/2
= 24

* Node(9) = (5+3) + (3+6) + (6+2) + (2+3) + (3+5)/2
= 19

* Node(10) = (8+3) + (3+7) + (7+4) + (4+2) + (2+8)/2
= 24

* Node(11) = (1+3) + (3+7) + (7+3) + (3+2) + (2+1)/2
= 16

∴ the path is

a → b → d → e → c → a

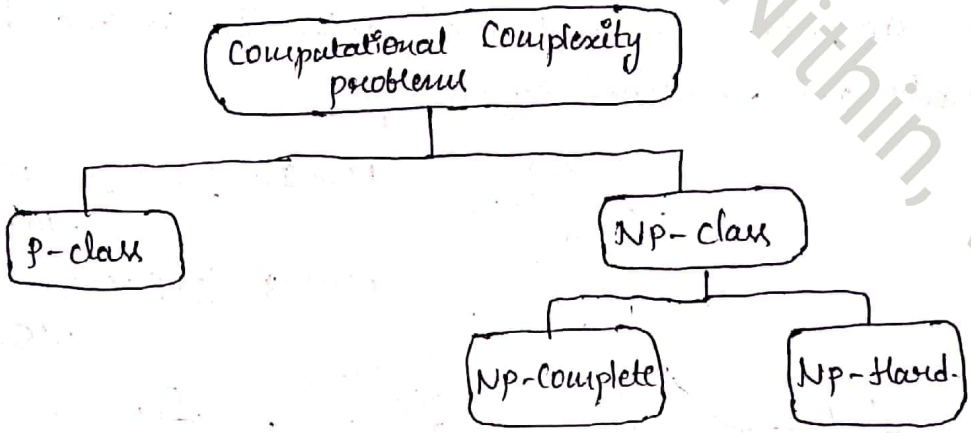
The cost of path is 16 //

* P, NP, NP-complete & NP-Hard classes :-

The Algorithm in which Every operation is uniquely defined is called "Deterministic Algo".

→ The Algorithm in which Every operation may not have unique result, rather there can be specified set of possibilities for Every operation such an Algorithm is called "Non-deterministic Algo".

→ There are two groups in which a problem can be classified



* P-class :- class-p is a class of decision problems that can be solved in "polynomial" time by Deterministic Algorithms.
This class of problems is called "polynomial".

→ Some of the class-p Algorithms include Sorting, Searching, GCD, Multiplication of two Integers Etc.
There are Algorithms for which no polynomial time Algo has been found & hence cannot be categorized as class-p are Tsp problem, knapsack problem, Hamiltonian Circuit Etc.

* NP-class :- class-NP is a class of decision problems that can be solved by Non-deterministic polynomial Algo.
This class of problems is called "Non-deterministic polynomial (NP)" class.

→ All the problems which are of class-p are also included under the class-NP.

However, Np also Contain Knapsack problem, Tsp problem, Hamilt-onian cycles Etc.

* Np-Complete problems :- A decision problem 'D' is said to be Np-Complete if
  * It belongs to class - Np
  * Every problem in Np is polynomially reducible to D.

* Np-Hard problems :- A problem is Np-Hard, if an Algorithm for Solving it can be translated into one for Solving any Np-problem.

  Np-Hard therefore Means "atleast as hard as any Np-problem

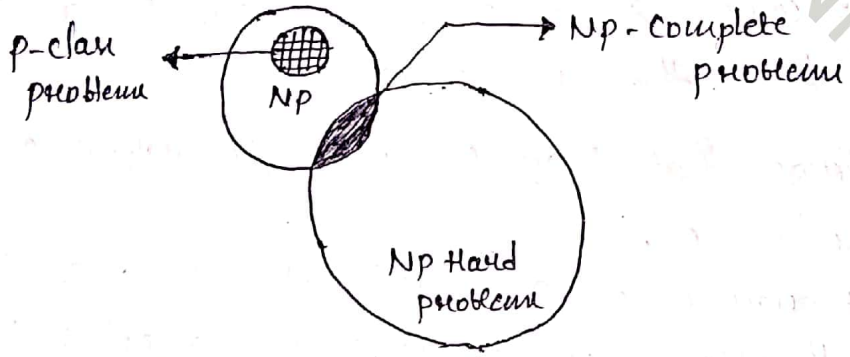


Fig: Relationship b/w P, NP, Np-Complete & Np-Hard

* Challenges of Numerical Algorithms :-

  Numerical Algorithms refers to Such Algorithms that are used for Solving Mathematical problems Such as

  * Evaluating Sin x, log x Etc
  * Evaluating Integrals Etc

However, numerous challenges are Encountered while Solving Mathematical problems Such as

* Most Numerical problems cannot be Solved "Exactly", they

have to be Solved "Approximately". This is usually done by replacing an Infinite object by a finite Approximation.

Ex: $e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \frac{x^n}{n!}$

* Due to Such Approximations, "Truncation Errors" would occur. One of the Major challenges in Numerical Analysis is to Estimate the Magnitude of Truncation Error.
This is done using Calculus tools.

* Other type of Error that could occur is the "round-off-Error". This is caused due to limited Accuracy while representing real nos in digital Computer.
Most Computers permit 3 levels of precision namely Single precision, Double precision & Extended precision. Using Extended precision Slows down the Computation.

* Another challenge that May occur is "overflow" & the "underflow" phenomenon.
An overflow occurs when an Arithmetic operation yields a result outside the range of Computer floating point No.
Underflow occurs when an Arithmetic operation yields a result of Such a Small Magnitude that cannot be represented.