Branch and Bound: The General Method, 0/1 Knapsack Problem, Travelling Salesperson problem.

NP Hard and NP Complete Problems: Basic Concepts, Cook's theorem NP Hard Graph Problems: Clique Decision Problem (CDP), Chromatic Number Decision Problem (CNDP), Traveling Salesperson Decision Problem (TSP).

Branch and Bound refers to all state space search methods in which all children of the E- Node are generated before any other live node becomes the E-Node.

Branch and Bound is the generalization of both graph search strategies, BFS and D-search.

- ABFS like state space search is called as FIFO (First in first out) search as the list of live nodes in a first in first out.
- A D-search like state space search is called as LIFO (last in first out) search as the list of live nodes in a last in first out list.

Live node is a node that has been generated but whose children have not yet been generated.

E-node is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.

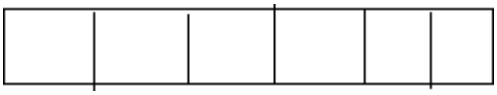
Dead node is a generated a node that is not be expanded or explored any further. All children of a dead node have already been expanded.

Here we will use 3 types of search strategies:

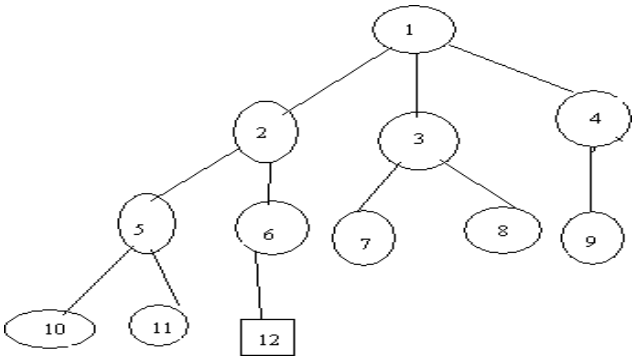
1. FIFO(First In First Out)
2. LIFO(Last In First Out)
3. LC(Least Cost)Search

FIFO Branch and BoundSearch:

For this we will use a data structure called Queue. Initially Queue is empty.



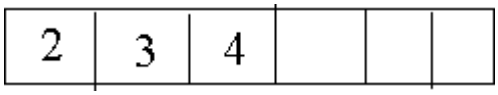
Example:



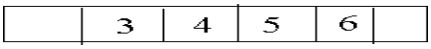
Assume the node12 is an answer node (solution)

In FIFO search, first we will take E-node as a node 1.

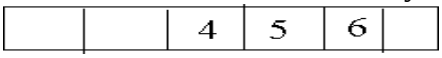
Next we generate the children of node 1.We will place all these live nodes in a queue.



Now we will delete an element from queue, i.e. node 2, next generate children of node 2 and place in this queue.

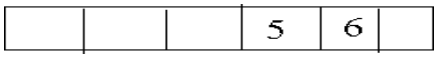


Next, delete an element from queue and take it as E-node, generate the children of node3, 7, 8 are children of 3 and these live nodes are killed by bounding functions. So we



will not include in the queue.

Again delete an element an from queue. Take it as E-node, generate the children of 4. Node 9 is generated and killed by boundary function.

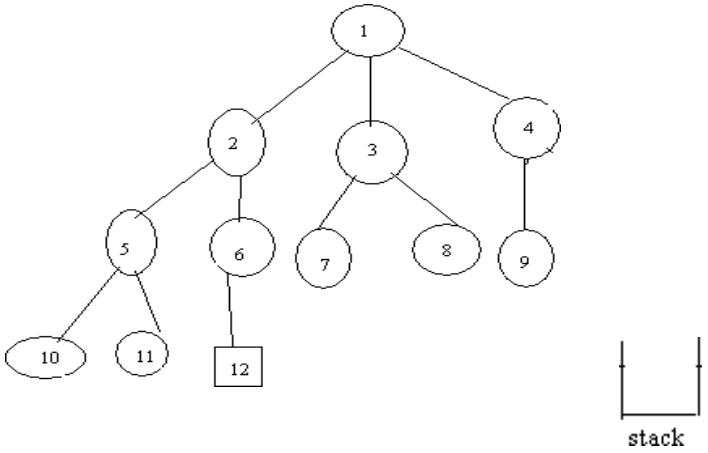


Next, delete an element from queue. Generate children of nodes 5, i.e., nodes 10 and 11 are generated and by boundary function, last node in queue is 6. The child of node 6 is 12 and it satisfies the conditions of the problem, which is the answer node, so search terminates.

LIFO Branch and Bound Search

For this we will use a data structure called stack. Initially stack is empty.

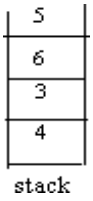
Example:



Generatechildrenofnode1andplacetheselivenodesintostack.

Remove element from stack and generate the children of it, place those nodes into stack.

2 is removed from stack. The children of 2 are 5, 6. The content of stack is,



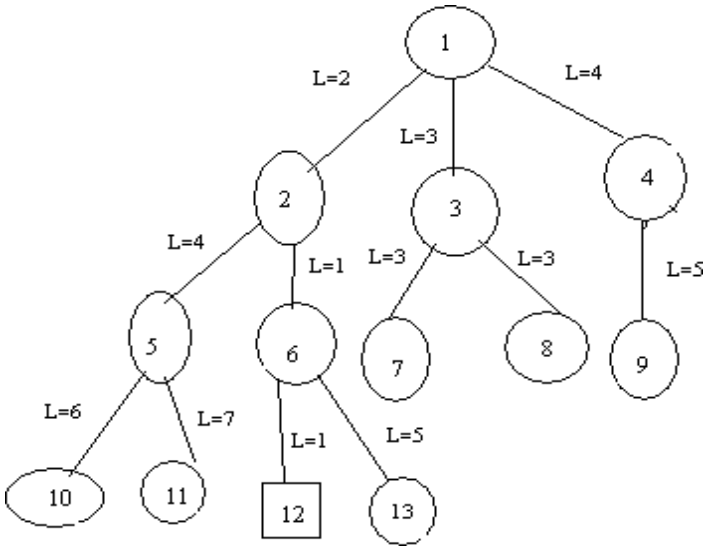
Again remove an element from stack, i.e., node 5 is removed and nodes generated by 5 are 10, 11 which are killed by bounded function, so we will not place 10, 11 into stack.

Delete an element from stack, i.e., node 6. Generate child of node 6, i.e., 12, which is the answer node, so search process terminates.

LC(Least Cost)Branch and Bound Search

In both FIFO and LIFO Branch and Bound the selection rules for the next E-node in rigid and blind. The selection rule for the next E-node does not give any preferences to a node that has a very good chance of getting the search to an answer node quickly.

In this we will use ranking function or cost function. We generate the children of E-node, among these live nodes; we select a node which has minimum cost. By using ranking function we will calculate the cost of each node.



Initially we will take node1as E-node. Generatechildrenofnode1, the children are 2, 3, 4. By using ranking function we will calculate the cost of 2, 3, 4 nodes is ĉ =2, ĉ =3, ĉ =4 respectively. Now we will select a node which has minimum cost i.e., node 2. For node 2, the children are 5, 6. Between 5 and 6 we will select the node 6 since its cost minimum. Generate children of node 6 i.,e 12 and 13. We will select node 12 since its cost (ĉ =1) is minimum. More over 12 is the answer node. So, we terminate search process.

Control Abstraction for LC-search

Let **t** beastatespacetreeand c() acost function forthenodes in t. Ifx isa node int,then c(x) isthe minimumcostofanyanswer node inthesubtreewithroot x.Thus, c(t) isthecostofa minimum-cost answer node in t.

LCsearchusesĉto findananswer node.Thealgorithmusestwofunctions

1. Least-cost()
2. Add_node().

Least-cost() finds a live node with least c(). This node is deleted fromthe list of live nodes and returned.

Add_node()todeleteand addalivenodefromortothelistoflivenodes.

Add_node(x)addsthenewlivenodextothelistoflivenodes.Thelistoflivenodesbeimplemented as a min-heap.

BOUNDING

- A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E- node.
- A good bounding helps to prune (reduce) efficiently the tree, leading to a faster exploration of the solution space. Each time a new answer node is found, the value of upper can be updated.
- Branch and bound algorithms are used for optimization problem where we deal directly only with minimization problems. A maximization problem is easily converted to a minimization problem by changing the sign of the objective function.

0/1 KNAP SACK PROBLEM (LCBB)

There are n objects given and capacity of knapsack is M. Select some objects to fill the knapsack in such a way that it should not exceed the capacityof Knapsack and maximum profit can be earned. The Knapsack problem is maximization problem. It means we will always seek for maximum p_1x_1 (where $p_1$represents profit of object x_1).

A branch bound technique is used to find solution to the knapsack problem. But we cannot directly apply the branch and bound technique to the knapsack problem. Because the branch bound deals only the minimization problems. We modify the knapsack problem to the minimization problem. The modifies problem is,

$$\text{minimize} \quad -\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq m$$

$$x_i = 0 \text{ or } 1, \quad 1 \leq i \leq n$$

```
Algorithm Reduce(p, w, n, m, I1, I2)
// Variables are as described in the discussion.
// p[i]/w[i] ≥ p[i + 1]/w[i + 1], 1 ≤ i < n.
{
    I1 := I2 := ∅;
    q := Lbb(∅, ∅);
    k := largest j such that w[1] + ··· + w[j] < m;
    for i := 1 to k do
    {
        if (Ubb(∅, {i}) < q) then I1 := I1 ∪ {i};
        else if (Lbb(∅, {i}) > q) then q := Lbb(∅, {i});
    }
    for i := k + 1 to n do
    {
        if (Ubb({i}, ∅) < q) then I2 := I2 ∪ {i};
        else if (Lbb({i}, ∅) > q) then q := Lbb({i}, ∅);
    }
}
```

Algorithm: KNAP SACK PROBLEM

Example: Consider the instance M=15, n=4, (p_1, p_2, p_3, p_4) = 10, 10, 12, 18 and (w_1, w_2, w_3, w_4)=(2, 4, 6, 9).

Solution: knapsackproblemcanbesolved byusingbranchand boundtechnique. Inthisproblem

we will calculate lower bound and upper bound for each node.

Arrange the item profits and weights with respect of profit by weight ratio. After that, place the first item in the knapsack. Remaining weight of knapsack is 15-2=13. Place next item w_2 inknapsackandtheremainingweight ofknapsack is13-4=9. Place next itemw_3, inknapsack then the remaining weight of knapsack is 9-6=3. No fraction are allowed in calculation of upper bound so w_4, cannot be placed in knapsack.

Profit=$p_1+p_2+p_3$,=10+10+

12 So, Upper bound=32

To calculate Lower bound we can place w_4 in knapsack since fractions are allowed in calculation of lower bound.

Lowerbound=10+10+12+ (3/9*18)=32+6=38

Knapsack is maximization problem but branch bound technique is applicable for only minimizationproblems. Inorderto convert maximizationprobleminto minimizationproblemwe have to take negative sign for upper bound and lower bound.

Therefore,upperbound(U)=-

32 Lower bound (L)=-38

We choosethe path, which has minimized difference ofupper bound and lower bound. If the difference is equalthen we choose the path by comparing upper bounds and we discard node with maximum upper bound.

Nowwewillcalculateupper boundandlower boundfor nodes2,3

Fornode2, x_1=1,meansweshouldplacefirstiteminthe knapsack.

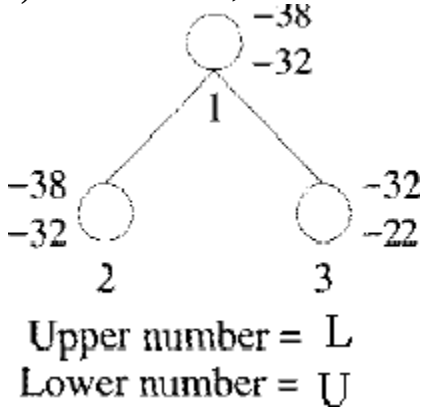
U=10+10+12=32,makeitas-32

L=10+10+12+(3/9*18) =32+6=38, wemakeitas-38

Fornode3,x_1=0,meansweshouldnotplacefirstiteminthe knapsack.

U=10+12=22,make itas-22

L=10+12+(5/9*18)=10+12+10=32,wemakeitas -32

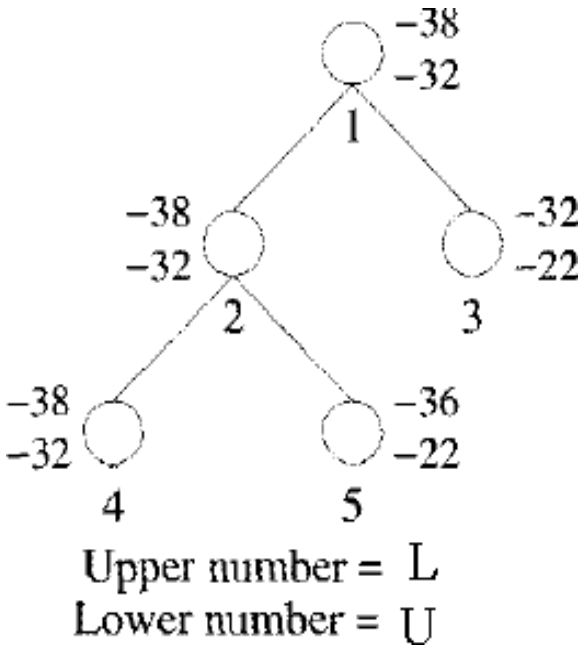


Next wewillcalculatedifferenceofupperboundand lowerbound fornodes2,3 For

node 2, U-L=-32+38=6

Fornode 3,U-L=-22+32=10

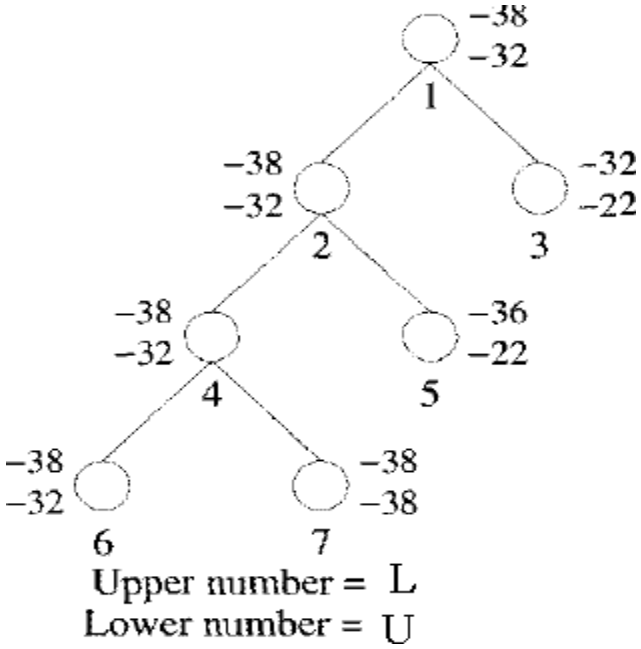
Choosenode2,sinceithas minimumdifferencevalueof6.



Nowwewill calculatelowerboundandupperboundof node4and5.Calculatedifferenceof lower and upper bound of nodes 4 and 5.

For node 4, U-L=-32+38=6 Fornode5,U-L=-22+36=14

Choosenode4,sinceithas minimumdifferencevalueof6

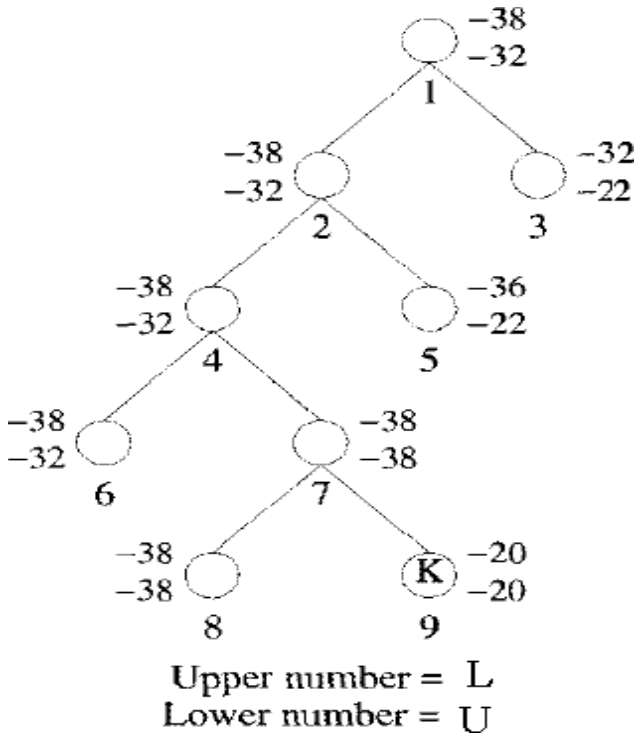


Nowwewill calculatelowerboundandupperboundof node6and7.Calculatedifferenceof lower and upper bound of nodes 6 and 7.

Fornode6,U-L=-32+38=6

Fornode7,U-L=-38+38=0

Choosenode7,sinceithas minimumdifferencevalueof0**.**



Now we will calculate lower bound and upper bound of node 8 and 9. Calculate difference of lower and upper bound of nodes 8 and 9.

Fornode8,U-L=-
38+38=0 Fornode9,U-
L=-20+20=0

Here, the difference is same, so compare upper bounds of nodes 8 and 9. Discard the node, which has maximum upper bound. Choose node 8, discard node 9 since, it has maximum upper bound.

Considerthe pathfrom1□2□4□7□8

X
$_1$=
1
X
$_2$=
1
X
$_3$=
0
X
$_4$=
1

Thesolutionfor0/1knapsackproblemis(x_1,x_2,x_3,x_4)=(1,1,0,1)

Maximum profit is:

∑pixi=10*1+10*1+12*0+18*1
10+10+18=38.

TRAVELLING SALES PERSON PROBLEM

Let G = (V', E) be a directed graph defining an instance of the traveling salesperson problem. Let Cijequalthecost ofedge(i, j), Cij= ∞ if(i, j) !=E, and let IVI =n, without lossof

generality, we can assume that every tour starts and ends at vertex 1.

Procedure

1. Reduce the given cost matrix. A matrix is reduced if every row and column is reduced. This can be done as follows:

   Row Reduction:

   a) Take the minimumelement fromfirst row, subtract it fromallelements offirst row, next take minimum element from the second row and subtract it from second row. Similarly apply the same procedure for all rows.
   b) Findthe sumofelements,whichwere subtractedfromrows.
   c) Applycolumnreductionsforthematrixobtainedafterrowreduction.

   Column Reduction:

   d) Take the minimum element from first column, subtract it from all elements of first column, next take minimumelement fromthe second column and subtract it fromsecond column. Similarly apply the same procedure for all columns.
   e) Findthe sumofelements,whichwere subtractedfrom columns.
   f) Obtain the cumulative sum of row wise reduction and column wise reduction.
   Cumulativereducedsum=Rowwisereductionsum+Columnwisereductionsum.
   Associate the cumulative reduced sumto the starting state as lower bound and α as upper bound.

2. Calculatethereducedcostmatrixfor everynode.
   a) Ifpath(i,j)isconsideredthenchange allentriesinrow**i**and column**j**ofAtoα.
   b) SetA(j,1)to α.
   c) Apply row reduction and column reduction except for rows and columns containing only α. Let **r** is the total amount subtracted to reduce the matrix.
   d) Find ĉ(S)= ĉ(R)+A(i,j)+r.

   Repeatstep2untilallnodesarevisited.

Example: Find the LC branchand bound solution for the travelling sales personproblemwhose cost matrix is as follows.

Thecost matrixis

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

Step1: Find thereducedcostmatrix

Applynowreductionmethod:

Deduct 10(whichisthe minimum) fromallvalues inthe1st row. Deduct 2(which is the minimum) fromall values inthe 2nd row. Deduct 2(whichisthe minimum) fromallvaluesinthe3rdtrow. Deduct 3 (which is the minimum) fromall values in the 4th row. Deduct 4 (which is the minimum) fromall values in the 5th row.

Theresultingrowwisereducedcostmatrix=

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞

Rowwise reductionsum= 10+2+2+3+4=21.

Nowapplycolumnreductionfor theabovematrix:

Deduct 1(which isthe minimum) fromall values inthe 1st column.
Deduct 3(whichisthe minimum) fromallvaluesinthe2ndcolumn.

Theresultingcolumnwisereducedcost matrix(A)=

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	3	12	∞

Column wise reduction sum = 1+0+3+0+0=4.
Cumulativereduced sum=rowwisereduction+columnwisereduction sum.
=21+ 4=25.
Thisisthecostofaroot i.e.node1,becausethisisthe initiallyreducedcost matrix. The lower bound for node is 25 and upper bound is ∞.
Starting fromnode1,wecannext visit 2, 3,4and5vertices. So, considertoexplorethepaths(1, 2), (1,3), (1, 4), (1,5).
Thetreeorganizationuptothisasfollows;
Variable **i** indicate the next node to visit.

Step2:

Nowconsiderthepath(1,2)

Changeallentriesofrow 1andcolumn2ofAto∞andalsosetA(2,1)to∞.

∞	∞	∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
15	∞	12	∞	0
11	∞	0	12	∞

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞. Then the resultant matrix is

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

Row reductionsum= 0+0+0+0=0
Columnreductionsum= 0+ 0+ 0+ 0=0
Cumulativereduction(r)=0+0=0
Therefore, asĉ(S)=ĉ(R)+A(1,2)+r□ ĉ(S)= 25+10+0=35.

Nowconsiderthepath(1,3)

Changeallentriesofrow 1andcolumn3ofAto∞andalsosetA(3,1)to∞.

∞	∞	∞	∞	∞
12	∞	∞	2	0
∞	3	∞	0	2
15	3	∞	∞	0
11	0	∞	12	∞

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

∞ ∞ ∞ ∞ ∞
1 ∞ ∞ 2 0
Thentheresultantmatrix is= ∞ 3 ∞ 0 2

4		3	∞	∞	0
0		0	∞	12	∞

Row reduction sum = 0
Columnreductionsum=11
Cumulativereduction(r)=0+11=1
1 Therefore, as ĉ(S)=
ĉ(R)+A(1,3)+r
ĉ(S)=25+17+11=53.

Nowconsiderthepath(1,4)

Change allentriesofrow 1andcolumn4ofAto∞andalsosetA(4,1)to∞.

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	3	∞	∞	2
∞	3	12	∞	0
11	0	0	∞	∞

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

∞		∞	∞	∞	∞
12		∞	11	∞	0
Thentheresultantmatrixis=	0	3	∞	∞	2
∞		3	12	∞	0
11		0	0	∞	∞

Row reduction sum = 0
Columnreductionsum=0
Cumulative reduction(r) = 0
+0=0
Therefore,asĉ(S)=ĉ(R)+A(1,4)+r
ĉ(S)=25+0+0=25.

NowConsiderthe path(1,5)

Change allentriesofrow 1andcolumn5ofAto∞andalsosetA(5,1)to∞.

∞	∞	∞	∞	∞
12	∞	11	2	∞
0	3	∞	0	∞
15	3	12	∞	∞
∞	0	0	12	∞

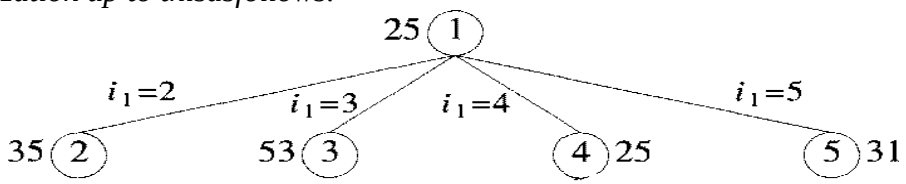
Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

	∞	∞	∞	∞	∞
	10	∞	9	0	∞
Thentheresultantmatrixis=	0	3	∞	0	∞
	12	0	9	∞	∞
	∞	0	0	12	∞

Rowreductionsum=5
Columnreductionsum=0
Cumulativereduction(r)=5 +0=0
Therefore, asĉ(S)=ĉ(R)+A(1,5)+r

ĉ(S)=25+1+5=31.

Thetreeorganization up to thisasfollows:



Numbers outside the node are $\hat{c}$ values

Thecostofthebetween(1, 2) =35,(1,3)=53, (1, 4)=25,(1,5) =31.Thecost ofthe path between (1, 4) is minimum. Hence the matrix obtained for path (1, 4) is considered as reducedcostmatrix.

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

Thenewpossiblepathsare(4,2),(4,3)and (4,5).

Nowconsiderthepath(4,2)

Changeallentriesofrow4andcolumn2ofAto∞andalsosetA(2,1)to∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{matrix}$$

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

Thentheresultantmatrixis=

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{matrix}$$

Row reduction sum = 0

Columnreductionsum=0

Cumulative reduction(r) = 0 +0=0

Therefore,asĉ(S)=ĉ(R)+A(4,2)+r

ĉ(S)=25+3+0=28.

Nowconsiderthepath(4,3)

Changeallentriesofrow 4andcolumn3 ofAto∞and alsosetA(3,1)to∞.

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

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

Thentheresultantmatrixis=

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{matrix}$$

0 0 ∞ ∞ ∞

Rowreductionsum=2
Columnreductionsum=11
Cumulativereduction(r)=2+11=13

Therefore, asĉ(S)=ĉ(R)+A(4,3)+r
ĉ(S)=25+12+13=50.

.***Now considerthepath(4,5)***

Change allentriesofrow 4andcolumn5ofAto∞andalsosetA(5,1)to∞.

∞	∞	∞	∞	∞
12	∞	11	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞

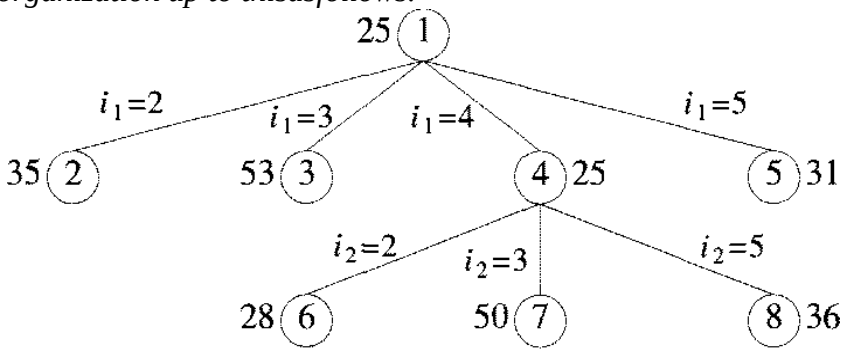
Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

Thentheresultantmatrixis=

∞	∞	∞	∞	∞
1	∞	0	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞

Row reduction sum =11
Columnreductionsum=0
Cumulativereduction(r)=11+0=1
1 Therefore, as ĉ(S)=
ĉ(R)+A(4,5)+r
ĉ(S)=25+0+11=36.

Thetreeorganization up to thisasfollows:



Numbers outside the node are $\hat{c}$ values

The cost ofthe between(4, 2) = 28, (4, 3) = 50,(4, 5) = 36.The cost ofthe path between (4, 2) is minimum. Hence the matrix obtained for path (4, 2) is considered as reduced costmatrix.

	∞	∞	∞	∞	∞
	∞	∞	11	∞	0
A=	0	∞	∞	∞	2
	∞	∞	∞	∞	∞
	11	∞	0	∞	∞

Thenewpossiblepathsare(2,3)and (2,5).

NowConsiderthe path(2,3):

Changeallentriesofrow2andcolumn3ofAto∞andalsosetA(3,1)to∞.

∞ ∞ ∞ ∞ ∞

∞	∞	∞	∞	∞
∞	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	∞	∞	∞

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

Thentheresultantmatrixis=

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
0	∞	∞	∞	∞

Rowreductionsum=13
Columnreductionsum=0
Cumulativereduction(r) =13+0=13
Therefore, asĉ(S)=ĉ(R)+A(2,3)+r
ĉ(S)=28+11+13=52.

NowConsiderthe path(2,5):
Changeallentriesofrow 2andcolumn5 ofAto∞and alsosetA(5,1)to∞.

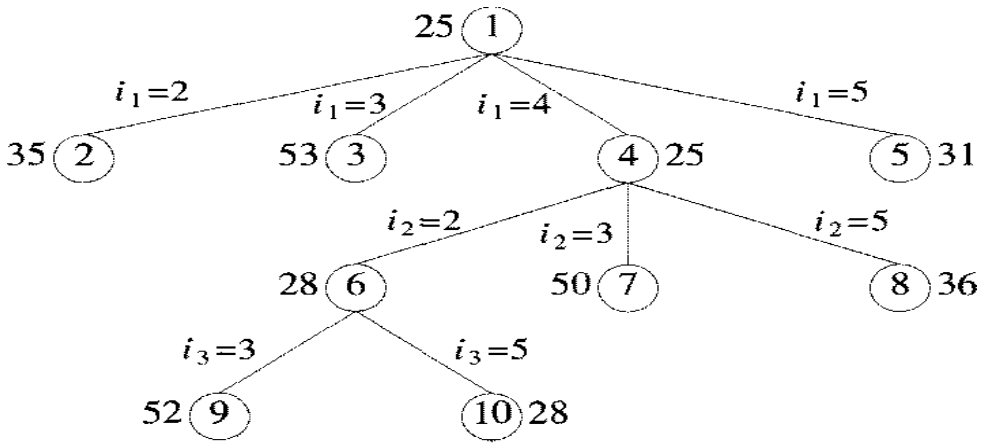
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	0	∞	∞

Applyrowandcolumnreductionfortherowsandcolumnswhoserowsandcolumnarenot completely ∞

Thentheresultantmatrixis=

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	0	∞	∞

Row reduction sum =0
Columnreductionsum=0
Cumulativereduction(r)=0 +0=0
Therefore,asĉ(S)=ĉ(R)+A(2,5)+r ĉ(S)= 28+0+0=28.
Thetreeorganization up to thisasfollows:



Numbers outside the node are $\hat{c}$ values

Thecostofthebetween(2,3)=52and(2,5)=28.Thecostofthepathbetween(2,5)is minimum. Hence the matrix obtained for path (2, 5) is considered as reduced cost matrix.

$$A=\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

Thenewpossiblepathis(5,3).

Nowc***onsiderthe path(5,3):***

Changeallentriesof row5andcolumn3of Ato∞andalsosetA(3,1)to∞.Applyrowand column reduction for the rows and columns whose rows and column are not completely ∞

Thentheresultantmatrixis=

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Rowreductionsum=0

Columnreductionsum=0

Cumulativereduction(r) =0+0=0

Therefore, asĉ(S)=ĉ(R)+A(5,3)+r

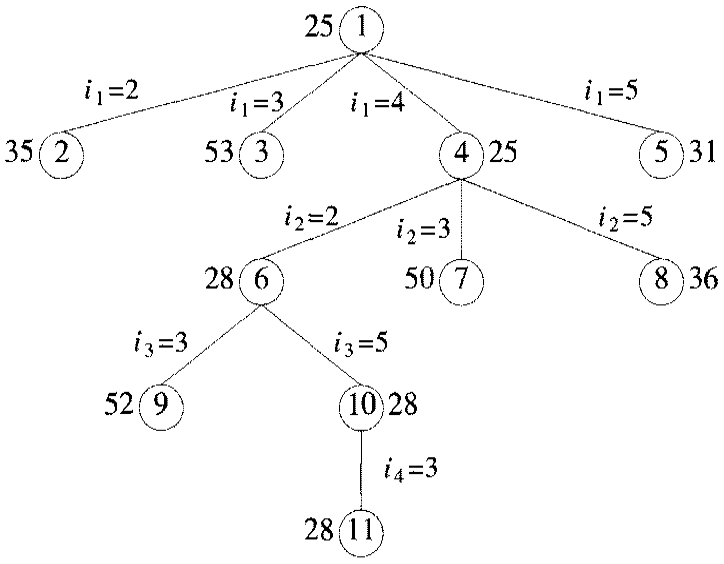
ĉ(S)=28+0+0=28.

Thepathtravellingsalespersonproblemis:

1□4□2□5□3□1**:**

Theminimumcost ofthepathis:10+2+6+7+3=28.

Theoverall treeorganization isasfollows:



Numbers outside the node are $\hat{c}$ values

NP Hard and NP Complete Problems

Basic concepts:

NP-) Nondeterministic Polynomial time

The problems has best algorithms for their solutions have "Computing times", that cluster into two groups

Group1	Group2
> Problems with solution time bound by a polynomial of a small degree. > It also called "Tractable Algorithms" > Most Searching & Sorting algorithms are polynomial time algorithms > **Ex:** Ordered Search **O(log n)),** Polynomial evaluation **O(n)** Sorting **O(n.log n)**	> Problems with solution times not bound by polynomial (simply non polynomial) > These are hard or intractable problems > None of the problems in this group has been solved by any polynomial Time algorithm > **Ex:** Traveling Sales Person **$O(n^2 2^n)$** Knapsack **$O(2^{n/2})$**

No one has been able to develop a polynomial time algorithm for any problem in the 2nd group (i.e., group 2)

So, it is compulsory and finding algorithms whose computing times are greater than polynomialveryquicklybecausesuchvastamountsoftimetoexecutethatevenmoderate size problems cannot be solved.

Theory of NP-Completeness:

Show that many of the problems with no polynomial time algorithms are computational time algorithms are computationally related.

There are two classes of non-polynomial time problems

1. NP-Hard
2. NP-Complete

NP Complete Problem: A problem that is NP-Complete can solved in polynomial time if and only if (iff) all other NP-Complete problems can also be solved in polynomial time.

NP-Hard: Problem can be solved in polynomial time then all NP-Complete problems can be solved in polynomial time.

All NP-Complete problems are NP-Hard but some NP-Hard problems are not known to be NP-Complete.

Non deterministic Algorithms:

Algorithms with the property that the result of every operation is uniquely defined are termed as deterministic algorithms. Such algorithms agree with the way programs are executed on a computer.

Algorithms which contain operations whose outcomes are not uniquely defined but are limited to specified set of possibilities. Such algorithms are called nondeterministic algorithms.

The machine executing such operations is allowed to choose anyone of these outcomes subject to a termination condition to be defined later.

To specify nondeterministic algorithms, there are 3 new functions.

Choice(S)-) arbitrarily chooses one of the elements of sets S

Failure ()-) Signals an Unsuccessful completion

Success()-)Signals a successful completion.

Example for NonDeterministic algorithms:

Algorithm Search(x){
//Problem is to search an element x
//output J, such that A[J]=x; or J=0 if x is not inA J:=Choice(1,n);
if(A[J]:=x)then{
Write(J);
Success();
}
else{
write(0);
failure();
}
	Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.
	A Nondeterministic algorithm terminates unsuccessfully if and only if (iff) there exists no set of choices leading to a successful signal.

Non deterministic Knapsack algorithm	

Algorithm DKP(p,w,n,m,r,x){
W:=0;
P:=0;
for i:=1 to n do{
x[i]:=choice(0,1);
w:=w+x[i]*w[i];
p:=p+x[i]*p[i];
}
if((w>m) or (p<r)) then
	p-) given Profits
	w-) given Weights
	n-) Number of elements (number of p or w)
	m-)Weight of bag limit
	p-) Final Profit
	w-) Final weight

Failure();
Else
Success();
}	

The Classes NP-Hard & NP-Complete:

For measuring the complexity of an algorithm, we use the input length as the parameter. For example, An algorithm A is of polynomial complexity p() such that the computing time of A is O(p(n)) for every input of size n.

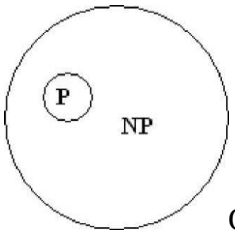
Decision problem/Decision algorithm: Any problem for which the answer is either zero or one is decision problem. Any algorithm for a decision problem is termed a decision algorithm.

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. An optimization algorithm is used to solve an optimization problem.

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

NP-)is the set of all decision problems solvable by nondeterministic algorithms in polynomial time.

Since deterministic algorithms are just a special case of nondeterministic, by this we concluded that **P ⊆ NP**



Commonly believed relationship between P&NP

The most famous unsolvable problems in Computer Science is Whether P=NP or P≠NP In considering this problem, s.cook formulated the following question.

If there any single problem in NP, such that if we showed it to be in 'P' then that would imply that P=NP.

Cook answered this question with

Theorem: Satisfiability is in P if and only if(iff)P=NP

-)Notation of Reducibility

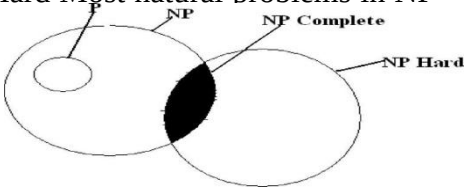
Let L1 and L2be problems, Problem L1 reduces to L2 (written **L1 α L2**) iff there is a way to solve L1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L2 in polynomial time

This implies that, if we have a polynomial time algorithm for L2, Then we can solve L1 in polynomial time.

Here α-)is a transitive relation i.e.,**L1αL2**and**L2αL3**then**L1αL3**

A problem L is NP-Hard if and only if (iff) satisfiability reduces to L ie., **Statisfiability α L**

A problem L is NP-Complete if and only if (iff) L is NP-Hard and **L€ NP** Commonly believed relationship among P, NP, NP-Complete and NP-Hard Most natural problems in NP are either in P or NP-complete.



Examples of NP-complete problems:

- Packing problems: SET-PACKING,INDEPENDENT-SET.
- Covering problems: SET-COVER,VERTEX-COVER.
- Sequencing problems: HAMILTONIAN-CYCLE,TSP.
- Partitioning problems: 3-COLOR, CLIQUE.
- Constraint satisfaction problems: SAT,3-SAT.
- Numerical problems: SUBSET-SUM, PARTITION, KNAPSACK.

Cook's Theorem: States that satisfiability is in P if and only if P=NP If P=NP then satisfiability is in P
If satisfiability is in P, then P=NP
To do this

- A-)Any polynomial time nondeterministic decision algorithm.
  I-)Input of that algorithm

Then formula Q(A,I), Such that Q is satisfiable iff '**A'** has a successful termination with Input **I**.

- If the length of '**I**' is '**n'** and the time complexity of **A** is **p(n)** for some polynomial **p()** then length of Q is $\mathbf{O(p^3(n) \log n)=O(p^4(n))}$

  The time needed to construct Q is also $\mathbf{O(p^3(n)\log n).}$

- A deterministic algorithm '**Z'** to determine the outcome of '**A**' on any input '**I**'

  Algorithm **Z** computes '**Q**' and then uses a deterministic algorithm for the satisfiability problem to determine whether '**Q**' is satisfiable.

- If **O(q(m))** is the time needed to determine whether a formula of length '**m**' is satisfiable then the complexity of '**Z**' is $\mathbf{O(p^3(n) \log n + q(p^3(n)\log n)).}$

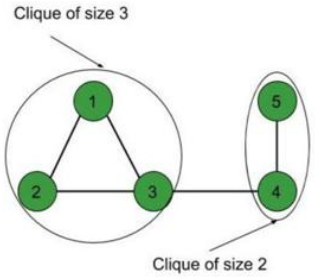
- If satisfiability is '**p**', then '**q(m)**' is a polynomial function of '**m**' and the complexity of '**Z**' becomes '**O(r(n))'** for some polynomial '**r()'.**

- Hence, if satisfiability is in **p**, then for every nondeterministic algorithm **A** in **NP**, we can obtain a deterministic **Z** in **p.**

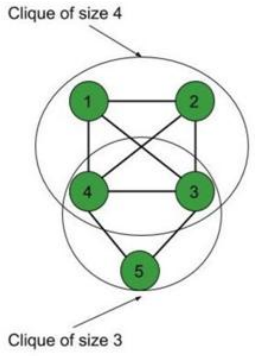
By this we shows that satisfiability is in **p** then **P=NP**

Clique Decision Problem (CDP)

A clique is a subgraph of a graph such that all the vertices in this subgraph are connected with each other that is the subgraph is a complete graph. The Maximal Clique Problem is to find the maximum sized clique of a given graph G, that is a complete graph which is a subgraph of G and contains the maximum number of vertices. This is an optimization problem. Correspondingly, the Clique Decision Problem is to find if a clique of size k exists in the given graph or not.



The above graph contains a maximum clique of size 3



The above graph contains a maximum clique of size 4

To prove that a problem is NP-Complete, we have to show that it belongs to both NP and NP-Hard Classes. (Since NP-Complete problems are NP-Hard problems which also belong to NP)

The Clique Decision Problem belongs to NP – If a problem belongs to the NP class, then it should have polynomial-time verifiability, that is given a certificate, we should be able to verify in polynomial time if it is a solution to the problem.

Proof:

1. Certificate – Let the certificate be a set S consisting of nodes in the clique and S is a subgraph of G.
2. Verification – We have to check if there exists a clique of size k in the graph.
   Hence, verifying if number of nodes in S equals k, takes O(1) time.
   Verifying whether each vertex has an out-degree of (k-1) takes $O(k^2)$ time.
   (Since in a complete graph, each vertex is connected to every other vertex through an edge. Hence the total number of edges in a complete graph = ${}^kC_2 = k*(k-1)/2$).

Therefore, to check if the graph formed by the k nodes in S is complete or not, it takes $O(k^2) = O(n^2)$ time (since k<=n, where n is number of vertices in G).

Therefore, the Clique Decision Problem has polynomial time verifiability and hence belongs to the NP Class.

The Clique Decision Problem belongs to NP-Hard –

A problem L belongs to NP-Hard if every NP problem is reducible to L in polynomial time. Now, let the Clique Decision Problem by C. To prove that C is NP-Hard, we take an already known NP-Hard problem, say S, and reduce it to C for a particular instance. If this reduction can be done in polynomial time, then C is also an NP-Hard problem.

The Boolean Satisfiability Problem (S) is an NP-Complete problem as proved by the Cook's theorem. Therefore, every problem in NP can be reduced to S in polynomial time. Thus, if S is reducible to C in polynomial time, every NP problem can be reduced to C in polynomial time, thereby proving C to be NP-Hard.

Proof that the Boolean Satisfiability problem reduces to the Clique Decision Problem

Let the boolean expression be – $F = (x_1 \text{ v } x_2) \text{ ^ } (x_1\text{' v } x_2\text{'}) \text{ ^ } (x_1 \text{ v } x_3)$

where x_1, x_2, x_3 are the variables,

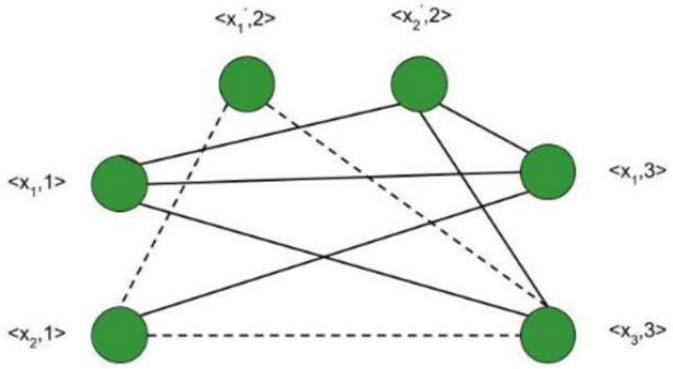
'^' denotes logical 'and', 'v' denotes logical 'or' and x' denotes the complement of x.

Let the expression within each parentheses be a clause.

Hence we have three clauses – C_1, C_2 and C_3.

Consider the vertices as – <x_1, 1>; <x_2, 1>; <x_1', 2>; <x_2', 2>; <x_1, 3>; <x_3, 3>
where the second term in each vertex denotes the clause number they belong to.
We connect these vertices such that –

1. No two vertices belonging to the same clause are connected.
2. No variable is connected to its complement.



Thus, the graph G (V, E) is constructed such that –
V = { <a, i> | a belongs to C_i } and
E = { (<a, i>, <b, j>) | i is not equal to j ; b is not equal to a' }
Consider the subgraph of G with the vertices <x_2, 1>; <x_1', 2>; <x_3, 3>.
It forms a clique of size 3 (Depicted by dotted line in above figure) .
Corresponding to this, for the assignment – <x_1, x_2, x_3> = <0, 1, 1> F evaluates to true.

- Therefore, if we have k clauses in our satisfiability expression, we get a max clique of size k and for the corresponding assignment of values, the satisfiability expression evaluates to true.
- Hence, for a particular instance, the satisfiability problem is reduced to the clique decision problem.

Therefore, the Clique Decision Problem is NP-Hard.

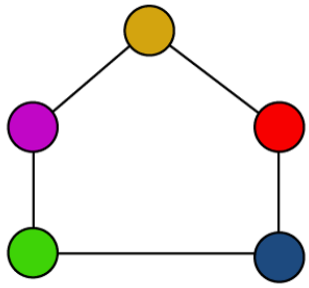
Chromatic Number Decision Problem (CNDP)

Graph coloring

Graph coloring can be described as a process of assigning colors to the vertices of a graph. In this, the same color should not be used to fill the two adjacent vertices. We can also call graph coloring as Vertex Coloring. In graph coloring, we have to take care that a graph must not contain any edge whose end vertices are colored by the same color. This type of graph is known as the Properly colored graph.

Example of Graph coloring

In this graph, we are showing the properly colored graph, which is described as follows:



The above graph contains some points, which are described as follows:

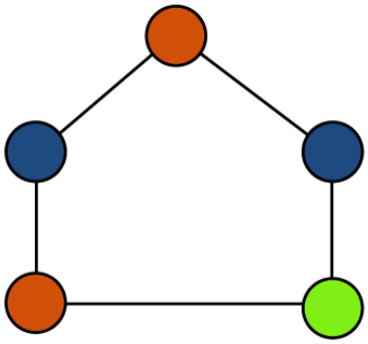
- The same color cannot be used to color the two adjacent vertices.
- Hence, we can call it as a properly colored graph.

Chromatic Number

The chromatic number can be described as the minimum number of colors required to properly color any graph. In other words, the chromatic number can be described as a minimum number of colors that are needed to color any graph in such a way that no two adjacent vertices of a graph will be assigned the same color.

Example of Chromatic number:

To understand the chromatic number, we will consider a graph, which is described as follows:



The above graph contains some points, which are described as follows:

- The same color is not used to color the two adjacent vertices.
- The minimum number of colors of this graph is 3, which is needed to properly color the vertices.
- Hence, in this graph, the chromatic number = 3
- If we want to properly color this graph, in this case, we are required at least 3 colors.

Types of Chromatic Number of Graphs:

There are various types of chromatic number of graphs, which are described as follows:

Cycle Graph:

A graph will be known as a cycle graph if it contains 'n' edges and 'n' vertices (n >= 3), which form a cycle of length 'n'. There can be only 2 or 3 number of degrees of all the vertices in the cycle graph.

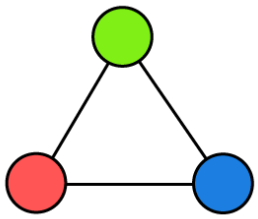
Chromatic number:

1. The chromatic number in a cycle graph will be 2 if the number of vertices in that graph is even.
2. The chromatic number in a cycle graph will be 3 if the number of vertices in that graph is odd.

Examples of Cycle graph:

There are various examples of cycle graphs. Some of them are described as follows:

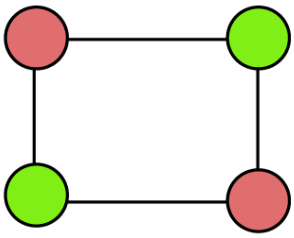
Example 1: In the following graph, we have to determine the chromatic number.



Solution: In the above cycle graph, there are 3 different colors for three vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is odd. So

Chromatic number = 3

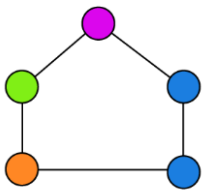
Example 2: In the following graph, we have to determine the chromatic number.



Solution: In the above cycle graph, there are 2 colors for four vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is even. So

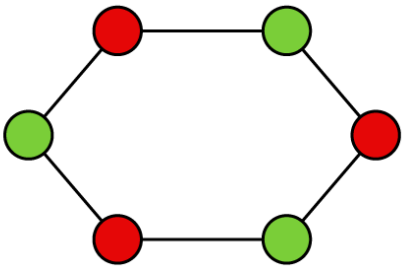
Chromatic number = 2

Example 3: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 4 different colors for five vertices, and two adjacent vertices are colored with the same color (blue). So this graph is not a cycle graph and does not contain a chromatic number.

Example 4: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 2 different colors for six vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is even. So Chromatic number = 2

Planner Graph

A graph will be known as a planner graph if it is drawn in a plane. The edges of the planner graph must not cross each other.

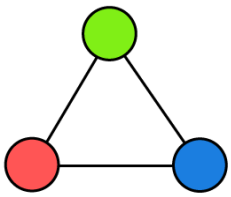
Chromatic Number:

1. In a planner graph, the chromatic Number must be Less than or equal to 4.
2. The planner graph can also be shown by all the above cycle graphs except example 3.

Examples of Planer graph:

There are various examples of planer graphs. Some of them are described as follows:

Example 1: In the following graph, we have to determine the chromatic number.

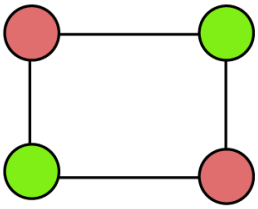


Solution: In the above graph, there are 3 different colors for three vertices, and none of the edges of this graph cross each other. So

Chromatic number = 3

Here, the chromatic number is less than 4, so this graph is a plane graph.

Example 2: In the following graph, we have to determine the chromatic number.

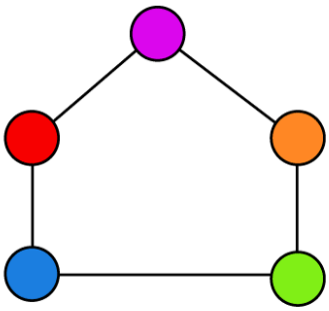


Solution: In the above graph, there are 2 different colors for four vertices, and none of the edges of this graph cross each other. So

Chromatic number = 2

Here, the chromatic number is less than 4, so this graph is a plane graph.

Example 3: In the following graph, we have to determine the chromatic number.

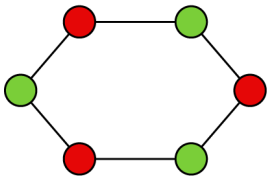


Solution: In the above graph, there are 5 different colors for five vertices, and none of the edges of this graph cross each other. So

Chromatic number = 5

Here, the chromatic number is greater than 4, so this graph is not a plane graph.

Example 4: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 2 different colors for six vertices, and none of the edges of this graph cross each other. So

Chromatic number = 2

Here, the chromatic number is less than 4, so this graph is a plane graph.

Complete Graph

A graph will be known as a complete graph if only one edge is used to join every two distinct vertices. Every vertex in a complete graph is connected with every other vertex. In this graph, every vertex will be colored with a different color. That means in the complete graph, two vertices do not contain the same color.

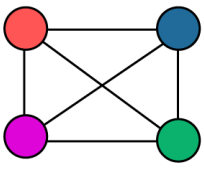
Chromatic Number

In a complete graph, the chromatic number will be equal to the number of vertices in that graph.

Examples of Complete graph:

There are various examples of complete graphs. Some of them are described as follows:

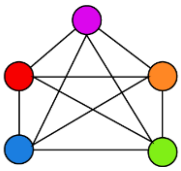
Example 1: In the following graph, we have to determine the chromatic number.



Solution: There are 4 different colors for 4 different vertices, and none of the colors are the same in the above graph. According to the definition, a chromatic number is the number of vertices. So,

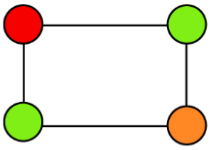
Chromatic number = 4

Example 2: In the following graph, we have to determine the chromatic number.



Solution: There are 5 different colors for 5 different vertices, and none of the colors are the same in the above graph. According to the definition, a chromatic number is the number of vertices. So, Chromatic number = 5

Example 3: In the following graph, we have to determine the chromatic number.



Solution: There are 3 different colors for 4 different vertices, and one color is repeated in two vertices in the above graph. So this graph is not a complete graph and does not contain a chromatic number.

Bipartite Graph

A graph will be known as a bipartite graph if it contains two sets of vertices, A and B. The vertex of A can only join with the vertices of B. That means the edges cannot join the vertices with a set.

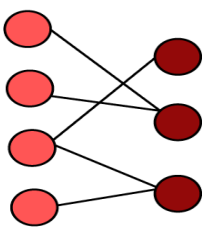
Chromatic Number

In any bipartite graph, the chromatic number is always equal to 2.

Examples of Bipartite graph:

There are various examples of bipartite graphs. Some of them are described as follows:

Example 1: In the following graph, we have to determine the chromatic number.



Solution: There are 2 different sets of vertices in the above graph. So the chromatic number of all bipartite graphs will always be 2. So

Chromatic number = 2

Tree:

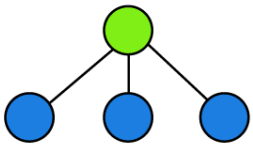
A connected graph will be known as a tree if there are no circuits in that graph. In a tree, the chromatic number will equal to 2 no matter how many vertices are in the tree. Every bipartite graph is also a tree.

In any tree, the chromatic number is equal to 2.

Examples of Tree:

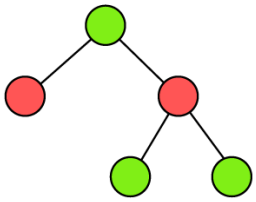
There are various examples of a tree. Some of them are described as follows:

Example 1: In the following tree, we have to determine the chromatic number.



Solution: There are 2 different colors for four vertices. A tree with any number of vertices must contain the chromatic number as 2 in the above tree. So, Chromatic number = 2

Example 2: In the following tree, we have to determine the chromatic number.



Solution: There are 2 different colors for five vertices. A tree with any number of vertices must contain the chromatic number as 2 in the above tree. So, Chromatic number = 2

Traveling Salesperson Decision Problem (TSP)

The travelling salesman problem(TSP) is a solution where a salesman has to start from one place and go to all other cities just once and then come back to their own place. TSP is all about finding the minimum distance path. The polynomial-time hardness is called NP Hard which defines the property of a class of problems. The subset sum is a simple example of NP hard problem.

NP-hard − The class of problem which cannot be solved within a polynomial time is called NP-hard.

Let’s take an example of five cities to understand how salesmen travel to each city by taking the minimum distance.

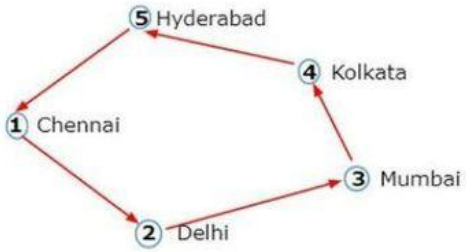


Figure I

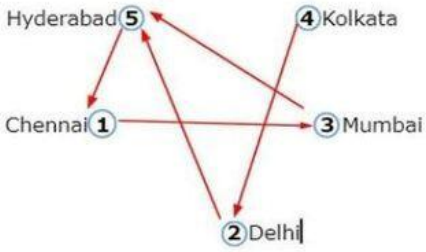


Figure II

Which one we will prefer among these two possibilities of figure?

In these two figures, figure II represents the least/minimum distance path. In figure I, when salesmen travel from Chennai(point 1) to Hyderabad(point 5) it takes a lot of time to reach its own location and total distance covered i.e.(1 -> 2 -> 3 -> 4 -> 5 -> 1) but in the case of figure II the salesman directly reach the point at 3(Mumbai) which saves the time and also cover the total minimum distance i.e. (1 -> 3 -> 5 -> 1).

Can we find the possibilities of 32 cities problem?

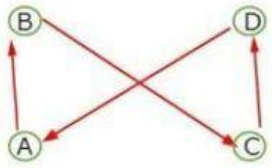
The possibility is given by **(n-1)!/2** solution.

- In the case of 4 cities problem,

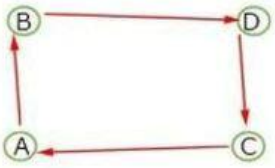
we have (4-1)!/2 = 3!/2

So there is a total of 3 possibilities for covering the distance.

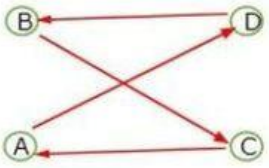
Let’s look at the picture and understand the possibilities of 4 cities.



A->B->C->D->A

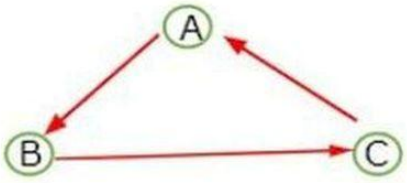


A->B->D->C->A



A->D->B->C->A

- In the case of 3 cities problem, there is only 1 possibility for covering the distance.



A->B->C

Now, let takes the problem of 41 cities where we can understand the NP-hard problem.

In the case of 41 cities, we will have 40!/2 solutions for a salesman to travel to each city.

If the salesman reaches each city and finds the best possible route it will take

- 40! / (2*106) seconds

Or 40! / (2*106*24*60*60) days

Or 40! / 365*(2*106*24*60*60) years = 6.8 * 1013 times which is more than the life of earth. Therefore, it is impossible to find all possible routes and get the optimal solution and this type of problem is called NP-hard.

Conclusion

We explored the concept of a traveling salesman problem in NP Hard. We saw many figures which simply give the meaning of minimum distance and find the best possible route but when we come to know the NP Hard problem we observe that this type of problem cannot be solved within a polynomial time. So, this is NP Hard.

TSP is NP-Complete

The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip

Proof

To prove ***TSP is NP-Complete***, first we have to prove that ***TSP belongs to NP***. In TSP, we find a tour and check that the tour contains each vertex once. Then the total cost of the edges of the tour is calculated. Finally, we check if the cost is minimum. This can be completed in polynomial time. Thus ***TSP belongs to NP***.

Secondly, we have to prove that ***TSP is NP-hard***. To prove this, one way is to show that ***Hamiltonian cycle*** $\leq_p$ ***TSP*** (as we know that the Hamiltonian cycle problem is NPcomplete).

Assume ***G = (V, E)*** to be an instance of Hamiltonian cycle.

Hence, an instance of TSP is constructed. We create the complete graph $G' = (V, E')$, where

$$E'=\{(i,j):i,j\in V and i\neq jE'=\{(i,j):i,j\in V and i\neq j$$

Thus, the cost function is defined as follows −

$$t(i,j)=\{01if(i,j)\in Eotherwiset(i,j)=\{0if(i,j)\in E1otherwise$$

Now, suppose that a Hamiltonian cycle h exists in G. It is clear that the cost of each edge in h is **0** in G' as each edge belongs to E. Therefore, h has a cost of **0** in G'. Thus, if graph G has a Hamiltonian cycle, then graph G' has a tour of **0** cost.

Conversely, we assume that G' has a tour h' of cost at most **0**. The cost of edges in E' are **0** and **1** by definition. Hence, each edge must have a cost of **0** as the cost of h' is **0**. We therefore conclude that h' contains only edges in E.

We have thus proven that G has a Hamiltonian cycle, if and only if G' has a tour of cost at most **0**. TSP is NP-complete.