UNIT 5

BRANCH AND BOUND

*Branch and bound:

→ The term 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 can become the E-node.

→ There are two graph search strategies.

1. BFS
2. DFS

→ In this methods, the expansion of a new node cannot begin until the node currently being the exploded is fully exploded.

→ Both of this methods generalise to branch and bound strategies.

→ In Branch and bound terminology, BFS called as FIFO search and DFS called as LIFO search.

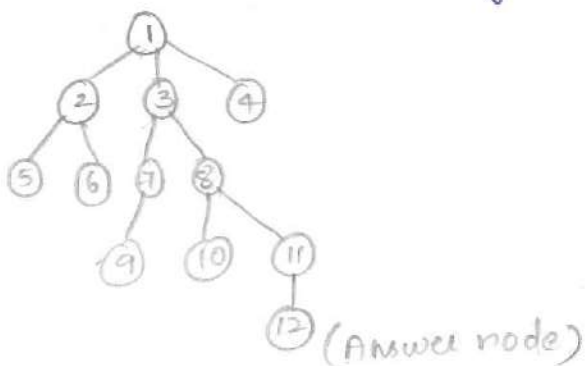
→ A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node.

→ There are three common search strategies.

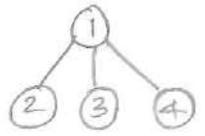
1. First in first out (FIFO)
2. Last in first out (LIFO)
3. Least cost search (LC)

1. First in first out search (FIFO):

→ First in first out search method uses queues data structure. consider the following example.

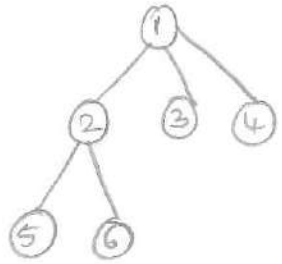


→ Assume that there is only one node. This node becomes Enode. Expand and ~~extent~~ generate all its children 2,3,4 nodes. Insert this nodes into the queue.



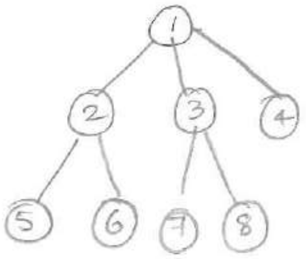
2	3	4

→ Queue follows FIFO. So remove node 2 from the queue and make it as E-node and generate its child nodes. 5,6 and insert 5,6 into queue



3	4	5	6

→ Make node 3 as E-node and generate it child nodes and insert into queue.



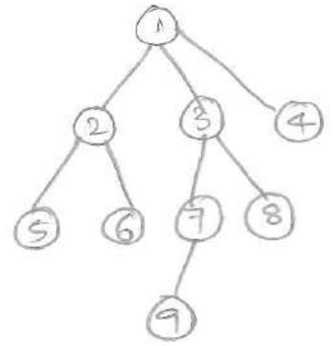
4	5	6	7	8

~~→ Make node 4 as E-node~~

→ Nodes 4,5,6 killed by bounding function because they are not possible to generate child nodes. Now 4,5,6 are removed from queue. Now, queue contains 7,8

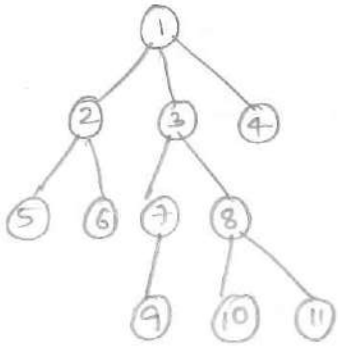
7	8

→ Make 7 as E-node and generate its children and insert into queue.



8	9

→ Make 8 as E-node and generate its children and insert into queue

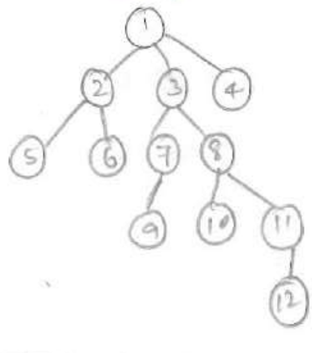


9	10	11

→ Make 9,10 as E-node since it does not generate children. i.e They are dead nodes. Remove 9,10 from queue.

11

→ Make 11 as the E-node and generate its child nodes and insert it into queue. 12 is the answer node and stop the process.



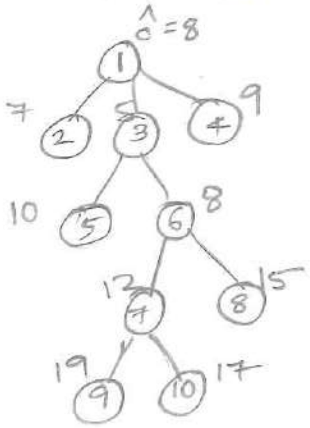
3. Least cost (LC):

→ In both FIFO and LIFO branch and bound. the selection rule for next E-node is not efficient. The selection rule for next E-node does not give any preference to any node that has a very good chance of getting the search to an answer node quickly.

→ In least cost search, the next E-node is selected on the basis of ranking function ($\hat{c}()$).

→ Ranking function is used to find cost of each and every node. The node which has least cost is selected as E-node to generate child nodes. This process continues until the ~~defined~~ to find answer node.

→ Consider the following example.



→ The cost of node 1 is 8 and make this node as E-node to generate child nodes 2, 3, 4. Find the cost of this node using ranking function. The cost of nodes 2, 3, 4 are 7, 5, 9 respectively. The cost of node 3 is least and is selected as E-node to generate child nodes 5 and 6.

→ Find cost of 5 and 6 using ranking function. The node 6 cost is least(8) and select this node as E-node to generate child nodes 7 and 8.

→ Find cost of 7 and 8 using ranking function The costs are 12 and 15 respectively. The cost of node 7 is least and select this node as E-node to generate child nodes 9 & 10. The costs are 19 and 17 respectively. The cost of node 10 is least and it is answer node. Stop the process.

* Least cost algorithm:

→ Control abstraction for branch and bound using least cost search method:

```
Algorithm LcSearch (t)
{
if *t is an answer node then
    output *t and return;
  E:= t, //E-node
```

. The
The
-node
tion.
as
on
of
-node
e
s
s-
sing

Initialize the list of live nodes to be empty

repeat

for each child x of E do

{

if x is an answer node then output the path

from x to t and return

Add(x);

(x→parent) := E;

}

if there are no more live nodes then write

("No answer node"); return;

E = least ( );

} until (false);

}

Algorithm

* 0/1 knapsack problem using least cost search method:

→ Branch and bound technique is used to solve minimization problems where as knapsack problem is a maximization problem. So convert maximization problem into minimization problem by replacing objective function $\sum P_i x_i$ by the function $-\sum P_i x_i$. The modified knapsack problem is stated as

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

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

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

* LC branch and bound solution for knapsack problem:

1) Consider the knapsack instance $n=4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$, $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$ and $m = 15$. Solve this problem using LC branch and bound.

Sol:- The search begins with root node as E-node

Initially root node is node'1'. Find lower bound cost and upper bound cost for node1 using ranking function.

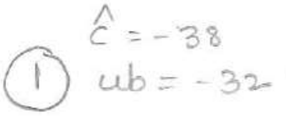
→ lower bound will accept the fractional values and upper bound does not accept fraction values.

Lower bound $(\hat{c}) = -(\sum P_i x_i) = -(P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4)$

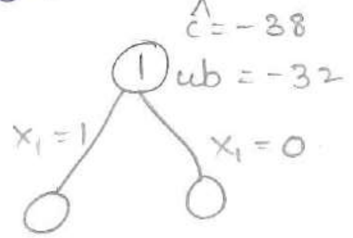
$\hat{c}\,(lb) = (10\cdot 1 + 10\cdot 1 + 12\cdot 1 + 18\cdot 3/9) = 38 = -38$

$ub = (10\cdot 1 + 10\cdot 1 + 12\cdot 1 + 18\cdot 0) = 32 = -32$

upper bound $= -32$



→ Make node1 as E-node to generate child nodes 2 and 3.



$x_1 = 1$ means first item must be placed in the knapsack.

$x_1 = 0$ means first item should not be placed in the knapsack.

→ for node 2 $[x_1=1]$: First item must be placed in bag

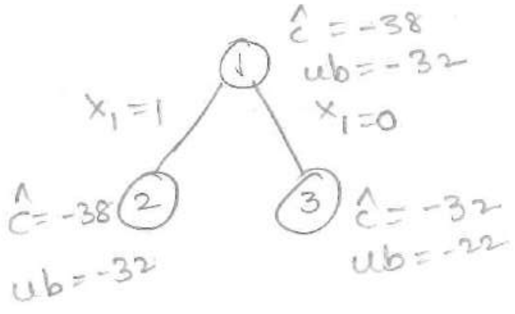
$$\hat{c} = 10\cdot1+10\cdot1+12\cdot1+18\cdot3/9 = 38 = -38$$

$$ub = 10\cdot0+10\cdot1+12\cdot1+18\cdot0 = 32 = -32$$

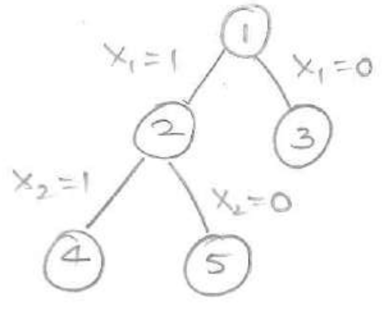
→ for node 3 $[x_1=0]$: first item should not be placed in bag.

$$\hat{c} = 10\cdot0+10\cdot1+12\cdot1+\cancel{18}^{2}\cdot5/\cancel{9} = -32$$

$$ub = 10\cdot0+10\cdot1+12\cdot1+18\cdot0 = -22$$



→ upper bound cost of node 2 is -32 which is least compared to node 3. So select node 2 as E-node to generate child nodes. 4 and 5



→ node 4 $[x_2=1]$: Second item must be placed in bag

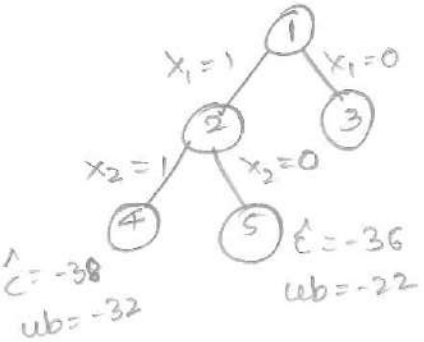
$$\hat{c} = lb = 10\cdot1+10\cdot1+12\cdot1+18\cdot3/9 = -38$$

$$ub = 10\cdot0+10\cdot1+12\cdot1+18\cdot0 = 32 = -32$$

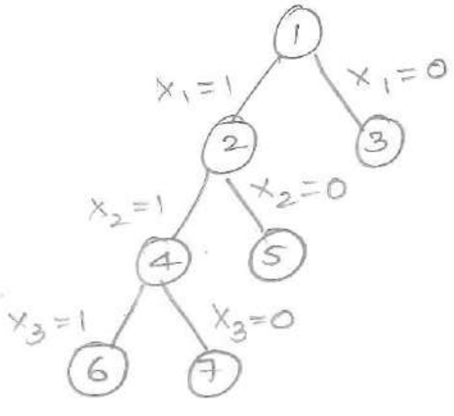
→ node 5 $[x_2=0]$: Second item should not be placed in bag

$$\hat{c} = lb = -(10\cdot 1 + 10\cdot 0 + 12\cdot 1 + 18\cdot 7/9) = -36$$

$$ub = -(10\cdot 1 + 10\cdot 0 + 12\cdot 1 + 18\cdot 0) = -22$$



→ upper bound cost of node 4 is -32 which is least compared to node 5. So select node 4 as E-node to generate its child nodes 6 & 7.



node 6 $[x_3=1]$:

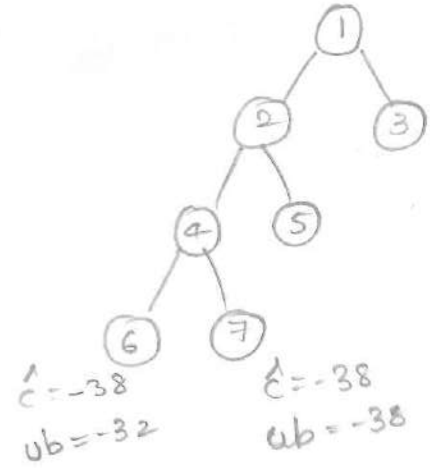
$$\hat{c} = lb = 10\cdot 1 + 10\cdot 1 + 12\cdot 1 + 18\cdot 3/9 = -38$$

$$ub = 10\cdot 1 + 10\cdot 1 + 12\cdot 1 + 18\cdot 0 = -32$$

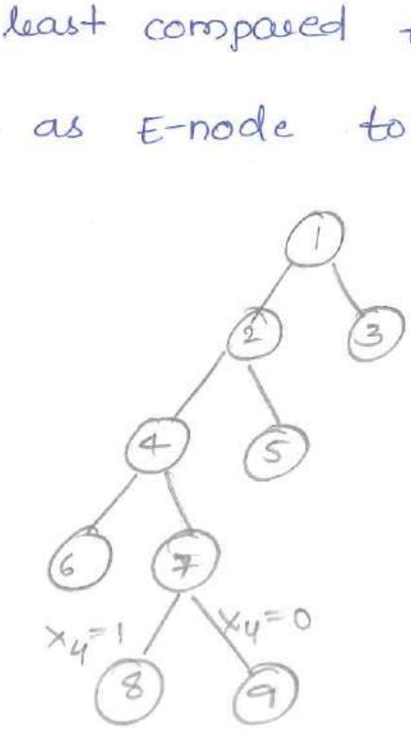
node 7 $[x_3=0]$:

$$\hat{c} = lb = -(10\cdot 1 + 10\cdot 1 + 18\cdot 1) = -38$$

$$ub = -(10\cdot 1 + 10\cdot 1 + 18\cdot 1) = -38$$



→ upper bound cost of node 7 is -38 which is least compared to node 6. So select node 7 as E-node to generate its child nodes 8 & 9



node 8 $[x_4=1]$:

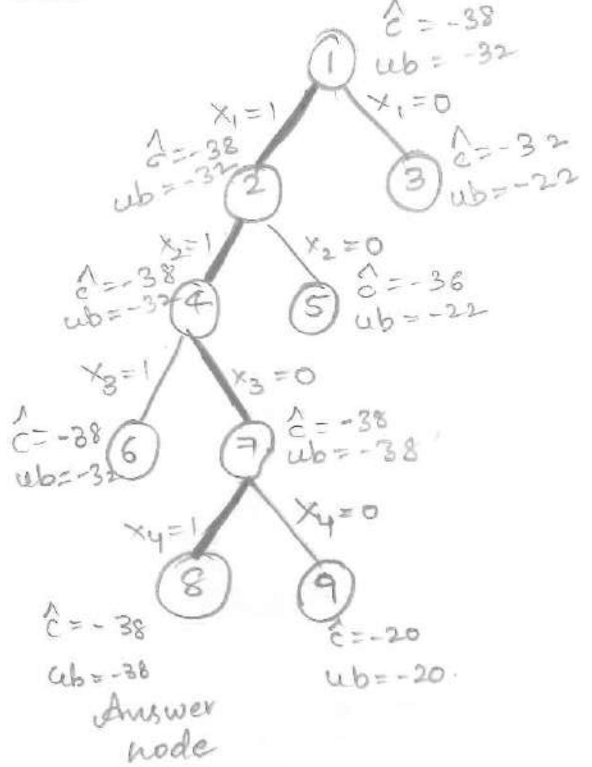
$$\hat{c} = 10\cdot1 + 10\cdot1 + 12\cdot0 + 18\cdot 9/9 = -38$$

$$ub = 10\cdot1 + 10\cdot1 + 12\cdot0 + 18\cdot 9/9 = -38$$

node 9 $[x_4=0]$:

$$\hat{c} = 10\cdot1 + 10\cdot1 + 12\cdot0 + 18\cdot0 = -20$$

$$ub = 10\cdot1 + 10\cdot1 + 12\cdot0 + 18\cdot0 = -20$$



→ ubcost of node 8 is −38 which is least then compared to node 9 and select this node as answer node.

→ So optimal solution is $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$.

Maximum profit

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

$$= 10 \cdot 1 + 10 \cdot 1 + 12 \cdot 0 + 18 \cdot 1$$

$$\Sigma P_i x_i = 38$$

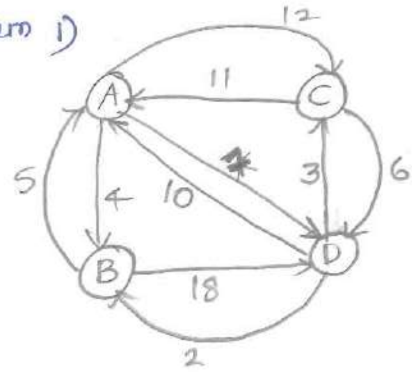
* Travelling salesperson problem:

→ Travelling salesperson problem consists of salesman and set of cities. A salesman has to visit every city, from starting a certain city (home city) and

Come back to same city. The objective of this problem is that salesman wants to reduce cost of the trip.

→ Consider the following graph.

problem 1)



Sol:- LC branch and bound solution.

<u>Step-1</u>: Represent the given graph in the form of adjacent cost matrix.

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & 4 & 12 & 7 \\ B & 5 & \infty & \infty & 18 \\ C & 11 & \infty & \infty & 6 \\ D & 10 & 2 & 3 & \infty \end{array}$$

Now reduce the cost matrix by using row reduction and column reduction.

Row reduction:

→ Subtract least element from all elements of the selected row

By row reduction

$$\begin{bmatrix} \infty & 4 & 12 & 7 \\ 5 & \infty & \infty & 18 \\ 11 & \infty & \infty & 6 \\ 10 & 2 & 3 & \infty \end{bmatrix} \begin{array}{c} 4 \\ 5 \\ 6 \\ \underline{2} \\ 17 \end{array} \Rightarrow \begin{bmatrix} \infty & 0 & 8 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 1 & \infty \end{bmatrix}$$

Column reduction:

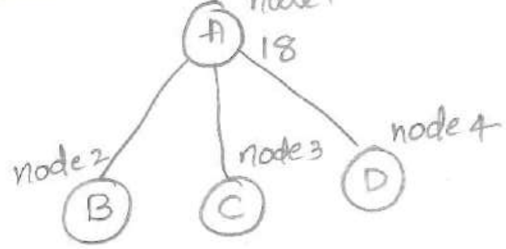
→ Subtract least element from all elements of selected row

$$\begin{array}{c} \begin{bmatrix} \infty & 0 & 8 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 1 & \infty \end{bmatrix} \\ \begin{array}{cccc} 0 & 0 & 1 & 0 \end{array} \end{array} \Rightarrow \boxed{\begin{bmatrix} \infty & 0 & 7 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 0 & \infty \end{bmatrix}}$$

→ So cost of node 1 is cost of row reduction + cost of column reduction

$$\text{node1 cost} = 17 + 1$$
$$= 18$$

→ Draw a state space tree



Step-2:

→ Choose vertex 'B' [node 2] to visit (path A → B)

1. From the reduced matrix of step-1 cost of [A,B] = 0

2. Set elements of row A and column B to infinite

3. Set cost of [B, A] = ∞.

4. The resulting cost matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix}$

Find the reduced cost matrix for above matrix.

row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix} \begin{matrix} \\ 13 \\ 0 \\ 0 \\ \hline 13 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix}$$

column reduction

$$\begin{array}{c} \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix} \\ \begin{matrix} 5 & & 0 & 0 \end{matrix} \end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & 0 \\ 3 & \infty & 0 & \infty \end{bmatrix}$$

reduced cost matrix

→ cost of node 2 = cost of node 1 + reduction cost + cost of [A, B]

= 18 + (13 + 5) + 0 = 36.

→ Now determine cost of node 3. choose vertex 3 (node 3) path A → C

1. From reduced matrix of step 1, cost $[A, C] = 7$

2. Set all elements of row A and column C to infinite

3. Set cost of $[C, A] = \infty$

4. Now resulting cost matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix}$

→ This matrix is already reduced the cost of nodes row reduction and column reduction is already done in above matrix.

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix}$$

reduced matrix

cost of node 3 = cost of node 1 + reduction cost + cost $[A, C]$

$= 18 + 0 + 7 = 25.$

→ choose vertex D [node 4] path $[A \to D]$ 1. From the reduced matrix of step '1' cost of (A, D) is 3

2. Set all elements of row A and column D to ∞

3. Set cost $[D, A] = \infty$

4. Resulting cost matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 5 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix}$

row reduction

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 5 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix} \begin{array}{c} \infty \\ 0 \\ 5 \\ 0 \end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix}$$

This matrix is already column reduced.

Cost of node 4 = cost of node 1 + reduction cost + cost of [A, D]

$= 18 + 5 + 3 = 26.$

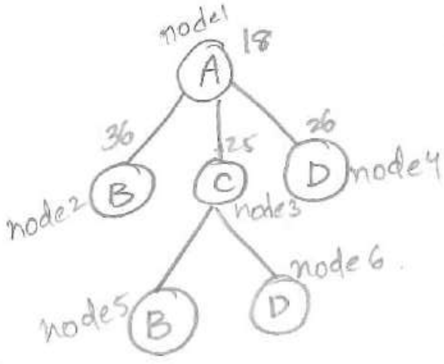
cost of node 2 = 36 $[A \to B]$

cost of node 3 = 25 $[A \to C]$

cost of node 4 = 26 $[A \to D]$

→ we have to choose the node with least cost

Therefore we choose node 3.



Step-3:

Note that we have to use cost matrix of node 3

(i.e) $\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix}$

→ choose vertex B to visit from vertex C [A→C→B] is path.

1. From reduced matrix of step 2 cost of [C, B] = ∞

2. Set all elements of row C and column B to infinite.

3. Set cost of [B, A] to infinite.

4. Now resulting cost matrix is : $\begin{bmatrix} \infty & \infty & 40 & 30 \\ \infty & \infty & \infty & 13 \\ \infty & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty \end{bmatrix}$

row reduction

$$\left[\begin{array}{cccc} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ \infty & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty \end{array}\right] \begin{array}{c} \bullet \\ 13 \\ \\ \underline{8} \\ 21 \end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \end{bmatrix}$$

This matrix is already column reduced.

cost of node 5 = cost of node 3 + reduction cost + cost(C,B)

= 25+21+∞

= ∞.

→ choose vertex D (node 6) to visit (A → C → D)

1. From reduced matrix of step 2 cost of [C, D] = 0

2. Set all elements of row C and column D to infinite

3. Set cost of [D, A] = ∞

4. The resulting cost matrix is

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \end{bmatrix}.$$

This matrix is already reduced so that

cost of node 6 = cost of node 3 + reduction cost + cost of [C, D]

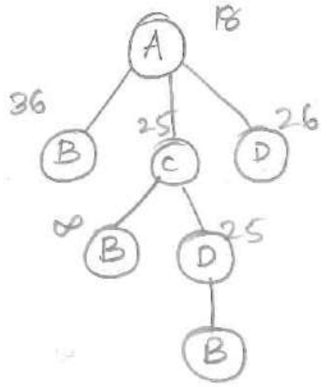
$= 25 + 0 + 0$

$= 25.$

→ Thus we have cost(5) = ∞, cost(6) = 25

path [A → C → D]

→ Now LC node is 6 select this node as E-node to generate child nodes.



→ choose vertex B [node 7] path [A → C → D → B]

1. From reduced matrix of step-3 cost of [D, B] = 0

2. Set all elements of row D and column B to infinity

3. Set cost of [B, A] = ∞

4. The resultant matrix is

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

→ This matrix is already reduced. So

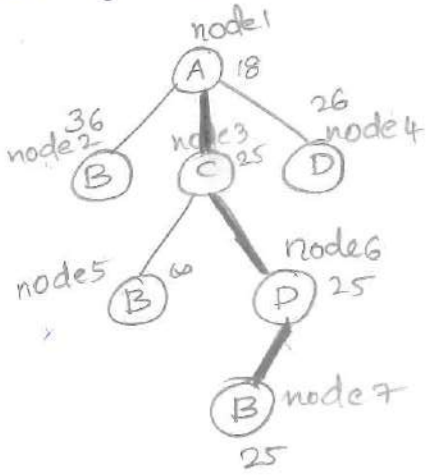
cost of node 7 = cost of node 6 + reduction cost + cost [D, B]

$= 25 + 0 + 0$

$= 25$

∴ cost of node 7 = 25

→ Thus optimal path is A → C → D → B → A with the cost of 25.



Solve travelling Salesperson problem using FIFO.

NP HARD AND NP-COMPLETE PROBLEMS.

→ Various problems have different time complexities such as

Time complexity	Description
$O(1)$	Constant
$O(n)$	Linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(\log n)$	logarithm
$O(n \log n)$	n log n
$O(2^n)$	exponent

Order of time complexity:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

* P-problems:

→ The problems that can be solvable in polynomial time using deterministic algorithm is called as P-problems.

→ Polynomial time can be expressed in Big Oh notation that is $O(n^k)$ where $k = 0, 1, 2, \ldots$

Ex: Searching of elements, sorting of elements.

* NP-problems:

→ NP stands for non-deterministic polynomial time.

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

→ Note that deterministic algorithms are just a special case of non-deterministic algorithms so we conclude that $P \subseteq NP$

→ NP problems catagorised into two classes.

1. NP complete
2. NP Hard.

1. NP complete:

→ A problem that is NP complete has the property that it can be solved in polynomial time if and only if all other NP-complete problems can also be solved in polynomial time.

Ex: travelling salesperson problem ($O(n^2 2^n)$), knapsack problem ($O(2^{n/2})$), Graph colouring, Sums of subsets, Hamiltonian cycle

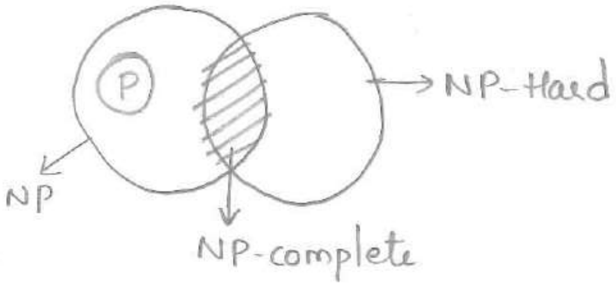
2. NP-Hard problem:

→ If any NP-Hard problem can be solved in polynomial time then all NP complete problems can be solved in polynomial time

→ Note that all NP-complete problems are NP-hard problems but not vice versa.

Ex: Halting problem.

* Relationship among P, NP, NP-complete and NP-Hard:



* Decision problem:

→ Any problem for which the answer is either yes or no is called a decision problem.

* Decision Algorithms:

→ The algorithm used for a decision problem is known as decision algorithm.

* Optimization problem:

→ Any problem for which the solution is optimal [minimum or maximum] value is called optimization problem

* Optimization algorithms:

→ The algorithm used to solve a optimization problem is known as optimization algorithm.

* Deterministic algorithms:

→ Deterministic algorithm has the property that the result of every operation is uniquely defined.

* Non-deterministic algorithms:

→ The algorithm has the property that the result of every operation is not uniquely defined but are limited to specified sets of possibilities.

→ Non-deterministic algorithm consists of 3 functions

1. Choice(s):

· This function randomly chooses one of element of set 's'

2- Failure ( ):

Signals and unsuccessful completion of algorithm.

3- Success ( ):

Signals and successful completion of algorithm

Ex: 1. Searching of element using non-deterministic algorithm.

```
Algorithm search [A[ ], n, x)
{
j := choice [i, n];
if (A[j] = x) then
{
write(j);
Success( );
}
else
{
write (0);
failure( );
}
}
```

2. Sorting of elements using non-deterministic algorithm.

```
Algorithm Nsort (A, n)
{
for i := 1 to n do B[i] = 0;
for i := 1 to n do
```

```
{
j:= choice (i,n);
if (B[j] ≠ 0) then, failure(),
B[j] = A[i];
}
for i:=1 to n-1 do
if B[i] > B[i+1] then Failure();
write (B[1:n]);
Success();
}
```

3. knapsack problem algorithm using NDA:

```
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
failure ();
else
success();
}
```

*clique problem:

Consider undirected graph $G=(V,E)$.

Clique:

A subset of vertices in V all connected to each other by edges in E.[i.e forming a complete graph]

Size of clique:

No. of vertices it contains

Max clique:

A maximal complete subgraph of graph 'G' is called a clique. whose size is no. of vertices in it.

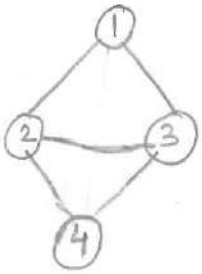
Optimization problem:

Find the maximal clique in a given graph 'G'.

Decision problem:

Find whether graph 'G' has a clique of size atleast k for some given 'k'.

Ex:

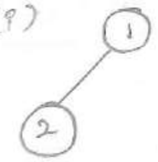


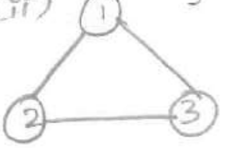
$G=(V,E)$

$V=\{1,2,3,4\}$

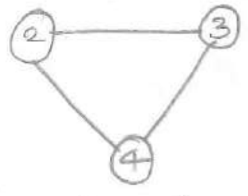
$E=\{(1,2),(1,3),(2,3),(2,4),(3,4)\}$

→ The above graph consisting of two cliques

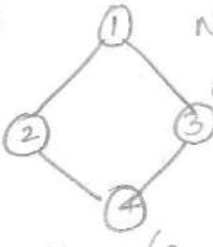
(i) 

(ii) $V=\{1,2,3\}$  It is complete graph and also max clique.

clique$(G,2)$ = Yes   clique$(G,3)$ = Yes.

(iii)  $V=\{2,3,4\}$

clique $(G,3)$ = Yes

(iv)  Not a complete graph, not clique.

clique $(G,4)$ = No

→ Complete graph is a graph in which every vertex is connected to remaining vertices.

* Satisfiability:

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

→ Let $x_1, x_2, \ldots x_n$ denote boolean variables (there value is either true or false)

Ex: Conjunctive normal form (CNF), Disjunctive normal form (DNF)

* Conjunctive normal formal form:

→ A formula is in CNF iff it is represented as

$\wedge_{i=1}^{k} C_i$ where C_i are clauses each represented as

$\vee l_{ij}$ where l_{ij} are literals.

Ex: 2-CNF $= (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2)$

$$3\text{-CNF} = \underbrace{(x_1 \vee x_2 \vee x_4)}_{\text{clause-1}} \wedge \underbrace{(x_3 \vee \bar{x}_4 \vee x_5)}_{\text{clause-2}}$$

Here each clause consisting of three literals. So this formula is said to be 3CNF.

* Disjunctive normal form (DNF):

A formula is in DNF iff it is represented as $\vee_{i=1}^{K} C_i$ and clause C_i is represented as $\wedge l_{ij}$ are literals.

Ex: 2-DNF $= (x_3 \wedge \bar{x}_4) \vee (x_1 \wedge \bar{x}_2)$

3-DNF $= (x_1 \wedge x_2 \wedge x_4) \vee (x_3 \wedge \bar{x}_4 \wedge x_5)$

Ex: 1. This formula is said to be ~~for~~ satisfiability

$p \wedge q$ where $p = T$

$T \wedge T$ $\quad q = T$

T

2. $p \wedge \sim p$

$T \wedge F$ $\quad P = T$

F

This formula is said to be not satisfiability.

* Halting problem:

→ Best example of NP- hard decision problem is halting problem.

→ The halting problem is to determine for an arbitary deterministic algorithm A and an input i whether algorithm A with input 'i' ever terminates or enters an infinite loop.

→ This problem is undecidable. Hence there exists no algorithm to solve this problem. So it clearly cannot be NP.

→ To show satisfiability of halting problem simply construct an algorithm 'A' whose input is a proportional formula 'x'. If x has n variables then algorithm 'A' tries out all 2^n possible truth assignments and verifies whether x is satisfiable.

→ If it is satisfiable then algorithm 'A' terminates successfully if it is not then algorithm 'A' enters infinite loop. Hence algorithm 'A' halts on input 'x' iff x is satisfiable.

* Cook's theorem:

→ Cook's theorem states that satisfiability is in P iff $P = NP$.

→ What is satisfiability:

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

- for ex: 1. $p \wedge q$ is satisfiable if $p = T, q = T$

  2. $p \wedge \sim p$ is not satisfiable if $p = T$ or $p = F$

<u>Theorem</u>:

Satisfiability problem is NP-complete. The proof consists of two steps.

1. Convert the execution of polynomial time non-deterministic machine to a bunch of well-formed formula such that formula satisfies iff the machine accept input.
2. Show the sum of lengths of formula is polynomial in the size of problem

Step-1: NP-Hard (L) - can polynomially reduce any NP problem to L.

Step-2: NP-complete - $L \in NP$

Step-3: $L \in NP \rightarrow$ Non deterministic machine for L that runs in polynomial time.

→ A problem that is NP-complete has the property that it can be solved in polynomial time iff all other NP-complete problems can also be solved in polynomial time.

→ If an NP-Hard problem can be solved in polynomial time then all NP-complete problems can be solved in polynomial time.

→ Satisfiability (SAT) problem $SAT \in NP$.

Therefore if we can polynomially reduce and an arbitary polynomial non-deterministic machine to satisfiability problem [SAT]. It means we have proved that satisfiability problem is NP-complete.