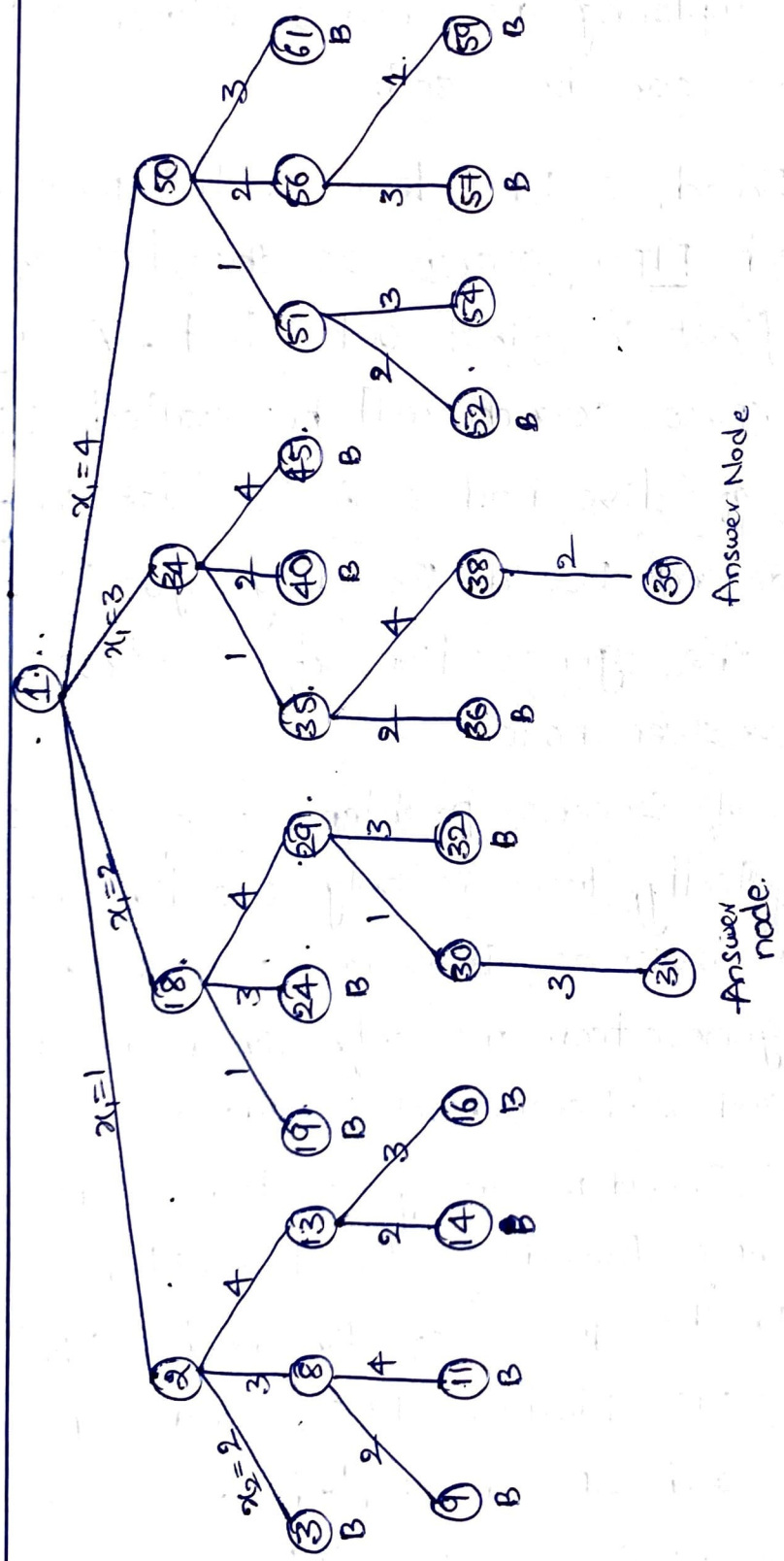
5. Branch and Bound.

General Method: Branch and Bound is general optimization technique that applies where the greedy method and dynamic programming fails. In Branch and Bound, a state space tree is built and all the children of E-nodes are generated before any other live node become E-node. For exploring new nodes either a BFS or D-Search technique can be used.

In Branch and Bound, a BFS like state space search will be called FIFO search as the list of live nodes is a first in first out list. A D-search like state space search will be called LIFO search as the list of live nodes is a last in first out list. In Branch and Bound, Bounding functions are used to avoid the generation of subtrees that do not contain an answer node.

Example: Consider the 4-Queens problem using a FIFO Branch and Bound. Initially, there is only one live node, i.e node 1. this node becomes E-node as its childrens 2, 18, 34 and 50 are generated. the only live nodes now are nodes 2, 18, 34 and 50. Hence, next E-node is 2. It is expanded and nodes 3, 8, and 13 are generated. Node 3 is immediately killed using Bounding function. Nodes 8 and 13 are added to the queue of live nodes. Node 18 becomes the next E-node. Nodes 19, 24 and 29 are generated. Nodes 19 and 24 are killed as a result of Bounding functions. Node 29 is added to the queue

of live nodes. Now the E-node is 34. The following diagram shows the portion of the tree of 4-Queen's problem that is generated by a FIFO Branch and Bound Search.



(*) Least Cost Search (LC): In both FIFO and LIFO branch and bound the selection rule for the next E-node is very complicated and blind. the selection rule for the next E-node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

for speeding up the search process an intelligent ranking function $\hat{c}(\cdot)$ is used for live nodes. the next E-node is selected on the basis of this ranking function. the ideal way to assign ranks would be on the basis of additional computational effort needed to reach an answer node from the live node. for any node x, this cost could be ① the number of nodes in the subtree x that need to be generated before an answer node is generated., ② the number of levels the nearest answer node is from x.

Let $\hat{g}(x)$ be an estimate of additional effort needed to reach an answer node from x. Node x is assigned a rank using function $\hat{c}(\cdot)$ such that

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

Where, $h(x)$ is the cost of reaching x from the root and $f(\cdot)$ is any non decreasing function.

In LC Search, a cost function $\hat{c}(\cdot)$ can be defined as follows:

① if x is an answer node, then $c(x)$ is the cost

of reaching x from the root of the state space tree.

(ii) If x is not an answer node, then $c(x) = \infty$.

(iii) Otherwise, $c(x)$ equals the cost of a minimum cost answer node in the subtree x.

(*) Control abstraction for LC-Search:

```
Algorithm LCSearch (t)
{
  if *t is an answer node then output *t and return;
  E := t;
  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();        --> Finds a live node with least ĉ(·).
  } until (false);
}
```

(*) Boundings: the Bounding functions are used to avoid the generation of subtrees that do not contain the answer nodes. In bounding, lower and upper bounds are generated at each node. A cost function $\hat{c}(x)$ is used to provide the lower bounds for any node x. Let upper is an upper bound on cost of minimum cost solution. In that case, all the live nodes with $\hat{c}(x) >$ upper can be killed.

Initially upper is set to ∞. After generating the children of current E-node, upper can be updated by minimum cost answer node.

(*) Applications of Branch and Bound:

(1) 0/1 Knapsack Problem: the 0/1 knapsack problem. states that- there are n objects $i=1,2,\cdots n$ and capacity of knapsack is m. and every object have its corresponding profits and weights. then select some objects to fill the knapsack in such a way that it should not exceed the capacity of knapsack and maximum profit can be earned. i.e the knapsack problem is a maximization problem and this Branch and Bound cannot be directly applied to maximization problem. This can be overcome by replacing the objective function $\sum p_i x_i$ by the function $-\sum p_i x_i$. Hence, the modified 0/1 knapsack problem can be stated as,

minimize $-\sum P_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \le m$, $x_i = 0$ or 1, $1 \le i \le n$.

Algorithm for Computing Upper Bound:

```
Algorithm UBound (cp, cw, k, m)
{
  b := cp; c = cw;
  for i := k+1 to n do
  {
    if (c + w[i] ≤ m) then
    {
      c := c + w[i];
      b := b - P[i];
    }
  }
  return b;
}
```

LC Branch and Bound Solution: the following steps are used to solve 0/1 knapsack using LCBB.

(i) Draw state space tree.
(ii) Compute $\hat{c}(\cdot)$ and $u(\cdot)$ for each node.
(iii) If $\hat{c}(x) >$ upper kill node x.
(iv) Otherwise the minimum cost $\hat{c}(x)$ becomes E-node & generate its children.
(v) Repeat step (iii) and (iv) until all nodes get covered.
(vi) the minimum cost $\hat{c}(x)$ becomes the answer node. Trace the path in backward direction to get solution.

Example: Draw the portion of state space tree generated by LCKNAP for knapsack instance: $n=4, m=15, (P_1,P_2,P_3,P_4) = (10,10,12,18)$ and $(w_1,w_2,w_3,w_4) = (2,4,6,9)$.

Sol: Initially state space tree contains

(1) $\hat{c} = -38$, $u = -32$ — K=0 Level — upper = ~~∞~~ −32

Compute upper bound $u(i)$. using UBound. for $i=1,2,3,4$. ∵K=0

$b = \cancel{0} \cancel{-10} \cancel{-20} -32 \implies u(1) = -32.$

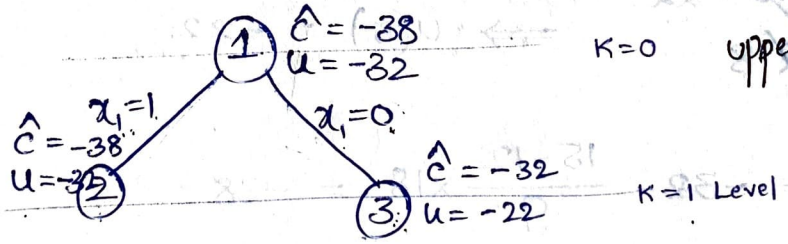
$C = \cancel{0}\ \cancel{2}\ \cancel{6}\ 12$

Compute $\hat{c}(1)$ using the formula

$$\hat{c}(x) = u(x) - \left[\frac{m - \text{Current total wt}}{\text{Actual wt of remaining object}}\right] \times \text{Actual profit of remaining object}$$

$$\therefore \hat{c}(1) = -32 - \frac{15-12}{9} \times 18 = -38.$$

Now generate the childrens of node (1) which are 2,3



Compute $u(2), u(3)$ for $i=2,3,4$ ∵ K=1

u(2)

$b = \cancel{-10} \cancel{-20} -32 \implies u(2) = -32$

$C = \cancel{2}\ \cancel{6}\ 12$

u(3)

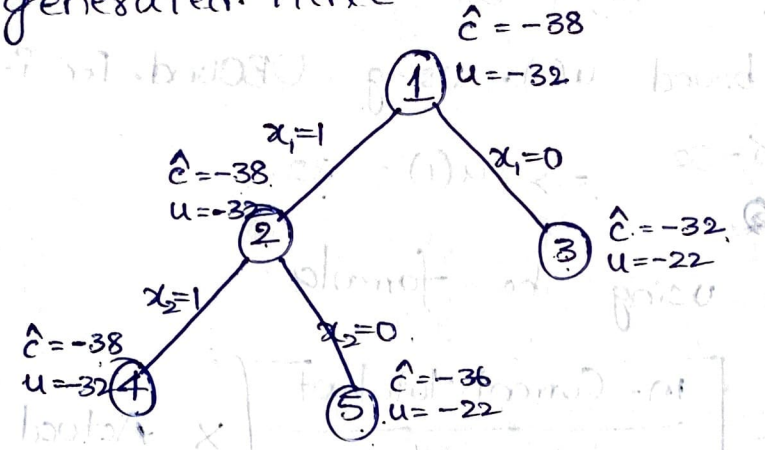
$b = \cancel{0} \cancel{-10} -22 \implies u(3) = -22.$

$C = \cancel{0}\ \cancel{4}\ 10$

$\therefore \hat{c}(2) = u(2) - \frac{15-12}{9} \times 18 = -32 - 6 = -38$

$\therefore \hat{c}(3) = u(3) - \frac{15-10}{9} \times 18 = -22 - 10 = -32$

Since node (2) has minimum ranking function, its children are generated. Here



$\hat{c}(2) > \text{upper}$

$-38 > -32$ is False

upper = -32

Now compute $u(4)$ and $u(5)$ for $i = 3, 4$ $\boxed{\because k = 2}$

$b = -20 - 32$
$c = 8 \quad 12$ $\Longrightarrow u(4) = -32.$

$b = -10 - 22$
$c = 2 \quad 8$ $\Longrightarrow u(5) = -22.$

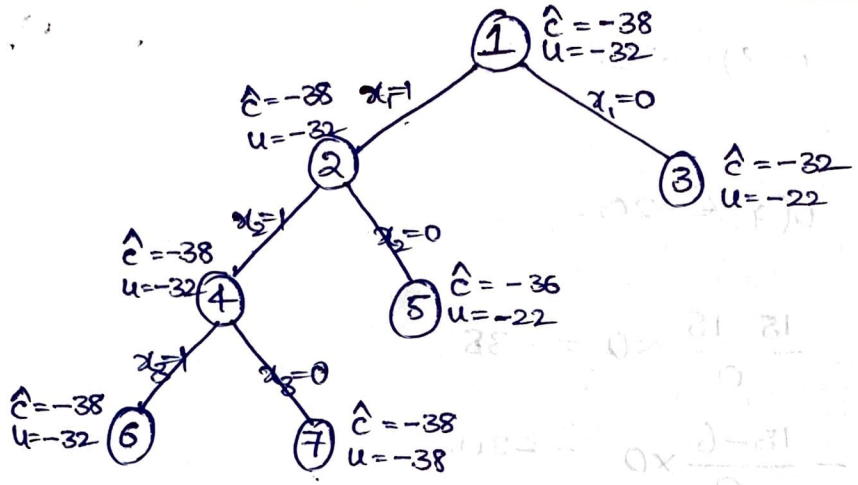
$\therefore \hat{c}(4) = -32 - \frac{15-12}{9} \times 18 = -38$

$\therefore \hat{c}(5) = -22 - \frac{15-8}{9} \times 18 = -36$

Since node (4) has min ranking function, its children are generated.

Compute $u(6), u(7)$ for $i = 4$ $\boxed{\because k = 3}$

$b = -32$
$c = 12$ $\Longrightarrow u(6) = -32.$



ĉ(4) > upper

-38 > -32 is False

upper = ~~-32~~ -38

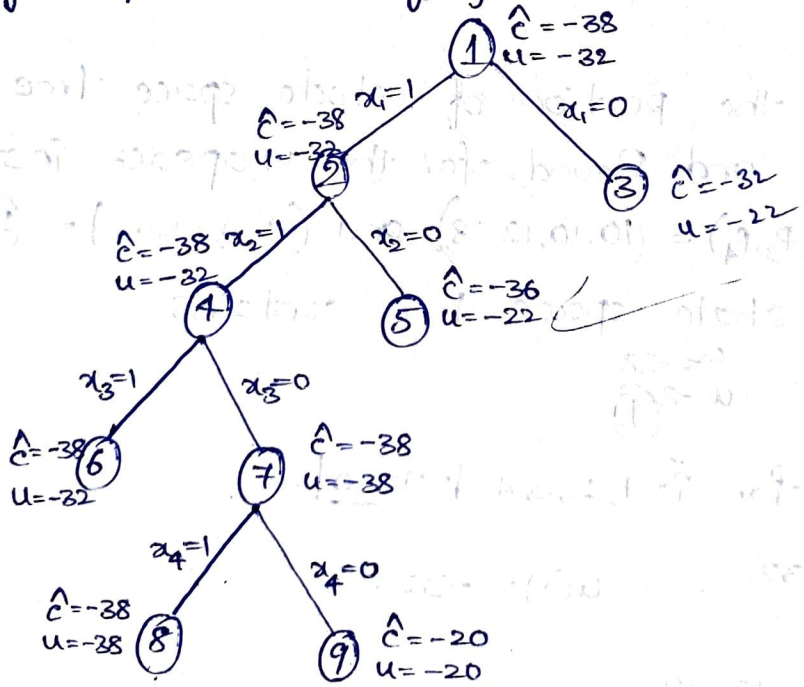
b = ~~-20~~ -38

c = ~~6~~ 15 $\Longrightarrow$ u(7) = -38.

$$\therefore \hat{c}(6) = -32 - \frac{15-12}{9} \times 18 = -38$$

$$\therefore \hat{c}(7) = -38 - \frac{15-15}{0} \times 0 = -38$$

Generate the childrens of node (6) or (7), Since both having equal ranking function. Hence



Now compute u(8), u(9) for i=5 $\boxed{\therefore K=4}$

$b = -38$
$c = 15$ $\Longrightarrow u(8) = -38.$

$b = -20$
$c = 6$ $\Longrightarrow u(9) = -20.$

$\therefore \hat{c}(8) = -38 - \frac{15-15}{0} \times 0 = -38$

$\hat{c}(9) = -20 - \frac{15-6}{0} \times 0 = -20.$

$\therefore \hat{c}(9) >$ upper node (9) is killed. Since node (8) has minimum $\hat{c}(\cdot)$ value it is the answer node. Hence optimal solution is $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$ //

FIFO Branch and Bound Solution :- Similar to LCBB, construct state space tree and compute $\hat{c}(\cdot)$ and upper at each node. If $\hat{c}(x) >$ upper kill the node immediately. Upper bound value is updated with minimum upper value automatically.

Example: Draw the portion of state space tree generated by FIFO Branch and Bound for the knapsack instance: $n=4, m=15, (P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$ and $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$.

Sol: Initially state space tree contains

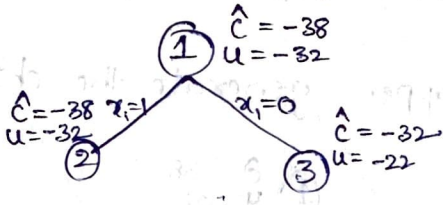
$\hat{c} = -38$
$u = -32$
(1)

Compute $u(1)$ for $i = 1, 2, 3, 4$ $\boxed{\because k=0}$

$b = \cancel{\emptyset} - \cancel{10} - \cancel{26} - 32$
$c = \emptyset \cancel{\times} \emptyset 12$ $\Longrightarrow u(1) = -32.$

$\therefore \hat{c}(1) = -32 - \frac{15-12}{9} \times 18 = -38.$

· Now, generate the childrens of node (1), which are 2, 3



Compute u(2), u(3) for i = 2, 3, 4 ∵ k = 1

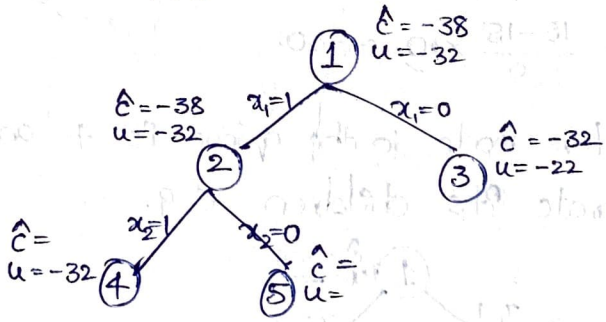
b = -10 -20 -32
c = 7 6 12 ⟹ u(2) = -32.

b = ∅ -10 -22.
c = ∅ 4 10 ⟹ u(3) = -22.

∴ $\hat{c}(2) = -32 - \frac{15-12}{9} \times 18 = -38$

∴ $\hat{c}(3) = -22 - \frac{15-10}{9} \times 18 = -32.$

Since node (2) is next node in Queue and ĉ(2) < upper ~~bound~~ ~~minimum~~ ~~ranking function~~, generate its childrens i.e 4, 5.



Compute u(4), u(5) for i = 3, 4 ∵ k = 2

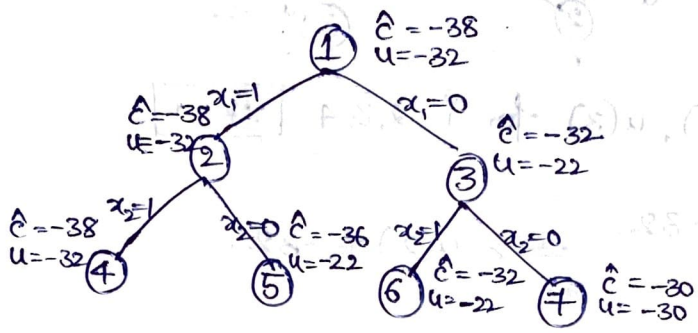
b = -20 -32
c = 8 12 ⟹ u(4) = -32.

b = -10 -22
c = 2 8 ⟹ u(5) = -22

∴ $\hat{c}(4) = -32 - \frac{15-12}{9} \times 18 = -38$

$$\hat{c}(5) = -22 - \frac{15-8}{9} \times 18 = -36$$

Now, $\because \hat{c}(3) <$ upper generate the childrens of 3 i.e 6, 7



Compute $u(6), u(7)$ for $i = 3, 4$ $\boxed{\because k=2}$

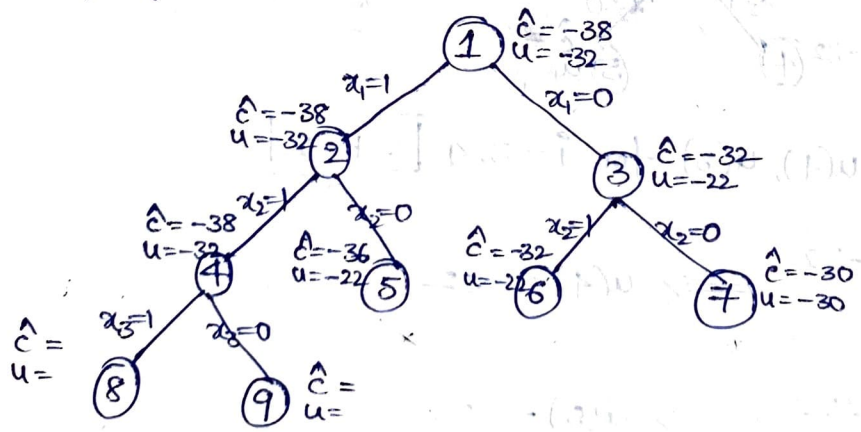
$b = -10 - 22$
$c = \cancel{\infty}\ 10$
$\Longrightarrow u(6) = -22.$

$b = \cancel{\emptyset} - \cancel{12} - 30$
$c = \cancel{\emptyset}\ \cancel{8}\ 15$
$\Longrightarrow u(7) = -30.$

$$\therefore \hat{c}(6) = -22 - \frac{15-10}{9} \times 18 = -32.$$

$$\therefore \hat{c}(7) = -30 - \frac{15-15}{0} \times 10 = -30.$$

Since next live node in the queue is 4 and $\hat{c}(4) <$ upper $(-38 < -32)$, generate its children 8, 9.



Compute $u(8), u(9)$ for $i = 4 \because k = 3$

$b = -32$
$c = 12$ $\Longrightarrow u(8) = -32.$

$b = -38$
$c = 15$ $\Longrightarrow u(9) = -38.$ $u = -32 \; -38.$

$\therefore \hat{c}(8) = -32 - \frac{15-12}{9} \times 18 = -38$

$\hat{c}(9) = -38 - \frac{15-15}{0} \times 0 = -38.$

Here upper is updated to -38.

Next live node in the queue is 5 and

$\because \hat{c}(5) >$ upper $\; -36 > -38.$ kill the node 5

Next live node in the queue is 6 and.

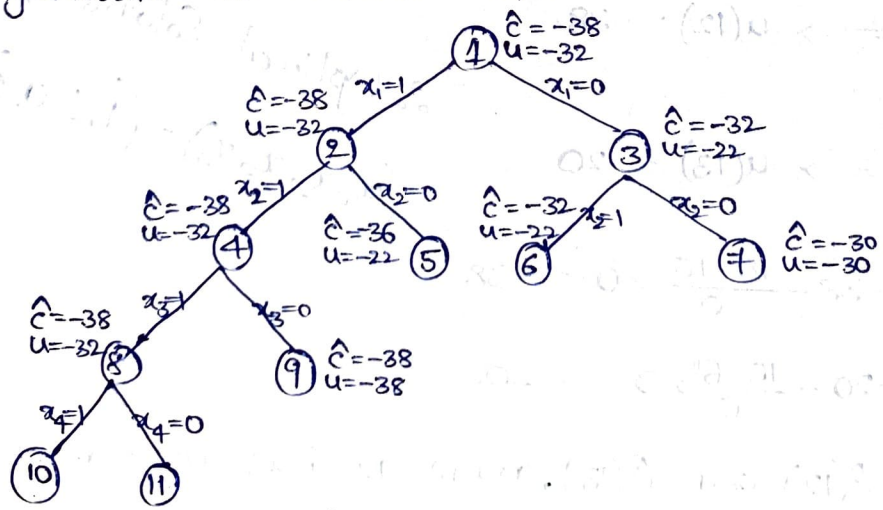
$\because \hat{c}(6) >$ upper $\; -32 > -38$ kill the node 6.

Next live node in the queue is 7 and

$\because \hat{c}(7) >$ upper $\; -30 > -38$ kill the node 7.

Next live node in the queue is 8 and $\hat{c}(8) <$ upper $(-38 < -38$

generate its children 10, 11.



Node 10 is not answer node, hence compute $u(11)$

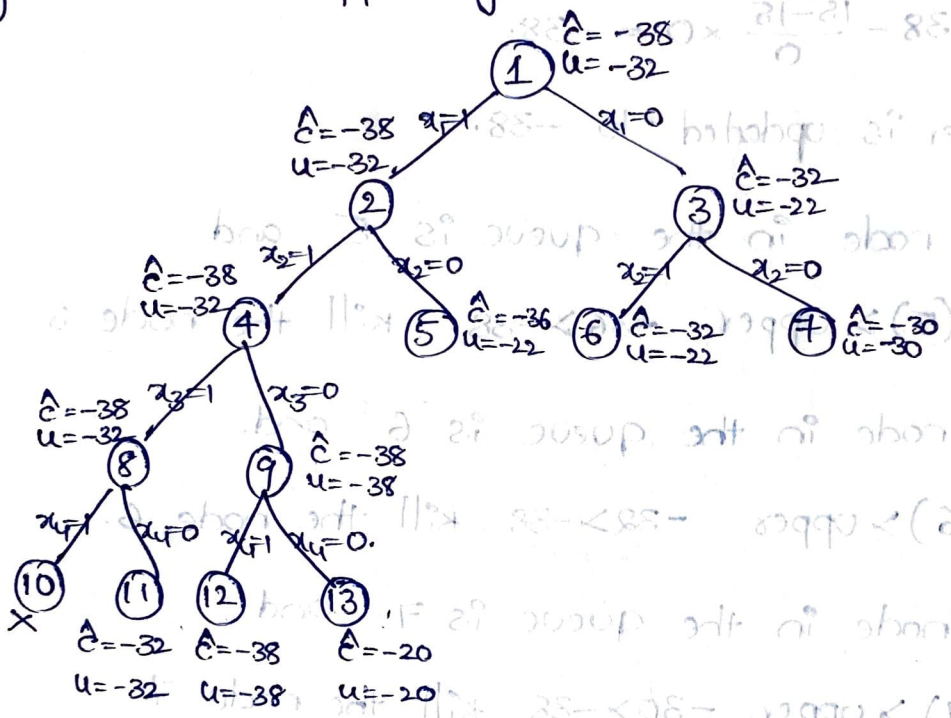
for $i=5$ $\boxed{\because k=4}$

$b=-32 \implies u(11)=-32.$

$c=12$

$\therefore \hat{c}(11) = -32 - \frac{15-12}{8} \times 8 = -32.$

Next live node in the queue is 9 and $\hat{c}(9)$ is not greater than upper generate its children. 12, 13.



Compute $u(12), u(13)$ for $i=5$ $\boxed{\because k=4}$

$b=-38 \implies u(12)=-38.$

$c=15$

$b=-20 \implies u(13)=-20$

$c=6$

$\therefore \hat{c}(12) = -38 - \frac{15-15}{0} \times 0 = -38$

$\hat{c}(13) = -20 - \frac{15-6}{0} \times 0 = -20.$

Among $\hat{c}(12)$ and $\hat{c}(13)$, node 12 has less ranking function which becomes answer node.

$\therefore$ Optimal Solution is

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

(2) Travelling Sales Person Problem: Let $G(V, E)$ be a directed or undirected graph with V vertices and E edges. Let c_{ij} = cost of the edge $\langle i, j \rangle$, $c_{ij} = \infty$ if there is no edge between i and j.

In Branch and Bound, define a cost function $\hat{c}(\cdot)$ to search the traveling salesperson state space tree. the cost $\hat{c}(\cdot)$ is such that the solution node with least $\hat{c}(\cdot)$ corresponds to a shortest tour in G.

A better $\hat{c}(\cdot)$ can be obtained by using reduced cost matrix corresponding to G. A row or column is said to be reduced iff it contains atleast one zero and all remaining entries are non-negative. A matrix is reduced if every row and column is reduced. For example consider the following matrix with vertices.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \begin{matrix} \to 10 \\ \to 2 \\ \to 2 \\ \to 3 \\ \to 4 \end{matrix} \quad 21.$$

$$\Longrightarrow \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 3 & 0 & 0 \end{bmatrix} = 4.$$

Reduced Cost matrix is $\Longrightarrow$

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Let A be the reduced cost matrix for node R. Let S be a child of R such that the tree edge (R,S) corresponds to including edge $\langle i, j \rangle$ in the tour. If S is not a leaf, then the reduced cost matrix for S is obtained as follows:

(i) change all entries in row i and column j of A to ∞.

(ii) Set $A(j,1)$ to ∞.

(iii) Reduce all rows and columns in the resulting matrix except for rows and colums containing only ∞.

then $\hat{c}(S) = \hat{c}(R) + r + A(i,j)$

where r is the reduced cost. = row Reduction + Column Reduction

Example: Apply the branch and bound algorithm to solve TSP for the following cost matrix.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Sol: Given that

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \begin{matrix} -10 \\ -2 \\ -2 \\ -3 \\ -4 \end{matrix}$$

Row Reduction = 21

$$\Longrightarrow \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

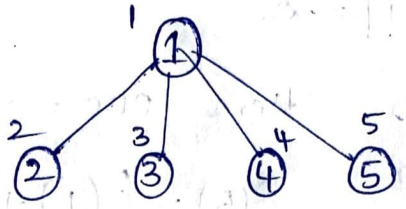
$1 \quad 0 \quad 3 \quad 0 \quad 0$ = Column Reduction

$$\Longrightarrow \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

$\therefore$ Optimum Cost = Row Reduction + Column Reduction

$\hat{c}(\cdot) = 21 + 4 = 25$

Now Construct the state space tree as follows.



To Compute $\hat{c}(2)$, make 1st row and 2nd column as ∞ and make $a[2,1]=\infty$.

∞	∞	∞	∞	∞	0
∞	∞	11	2	0	0
0	∞	∞	0	2	0
15	∞	12	∞	0	0
11	∞	0	12	∞	0
1	1	1	1	1	
0	0	0	0	0	

Here RRC = 0, CRC = 0 $\therefore r = 0$

$$\hat{c}(2) = \hat{c}(1) + r + A(1,2)$$
$$= 25 + 0 + 10$$
$$= 35$$

To Compute $\hat{c}(3)$, make 1st row and 3rd column as ∞. and make $a[3,1]=\infty$.

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

Here RRC = 0, CRC = 11, $\therefore r = 11$

$$\hat{c}(3) = \hat{c}(1) + r + A(1,3)$$
$$= 25 + 11 + 17$$
$$= 53$$

To Compute $\hat{c}(4)$, make 1st row and 4th column as ∞ and make $A[4,1]=\infty$

∞	∞	∞	∞	∞	0
12	∞	11	∞	0	0
0	3	∞	∞	2	0
∞	3	12	∞	0	0
11	0	0	∞	∞	0
1	1	1	1	1	
0	0	0	0	0	

Here RRC = CRC = 0, $\therefore r = 0$

$$\hat{c}(4) = \hat{c}(1) + r + A(1,4)$$
$$= 25 + 0 + 0$$
$$= 25$$

To compute $\hat{c}(5)$, make 1st row and 5th column as ∞ and make $A[5,1] = \infty$.

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

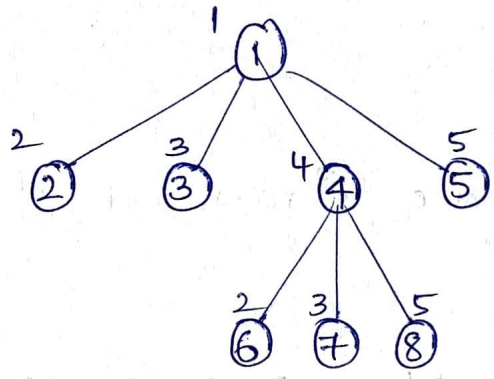
Here RRC = 2+5 = 5, CRC = 0 $\therefore r = 5$

$\hat{c}(5) = \hat{c}(1) + r + A(1,5)$

$= 25 + 5 + 1$

$= 31$

Since, cost of node 4 is optimum, Generate its children nodes 6, 7, 8 as shown below.



To compute $\hat{c}(6)$ make 1st row, 4th row, 4th column and 2nd column as ∞ and also make $A[4,1] = A[2,1] = \infty$.

∞	∞	∞	∞	∞	0
∞	∞	11	∞	0	0
0	∞	∞	∞	2	0
∞	∞	∞	∞	∞	0
11	∞	0	∞	∞	0
1	1	1	1	1	
0	0	0	0	0	

Here RRC = CRC = 0, $\therefore r = 0$

$\hat{c}(6) = \hat{c}(4) + r + A(4,2)$

$= 25 + 0 + 3$

$= 28.$

To compute $\hat{c}(7)$ make 1st row, 4th row, 4th column and 3rd column entries a ∞ and also make $A[4,1] = A[3,1] = \infty$.

∞	∞	∞	∞	∞	0
12	∞	∞	∞	0	0
∞	3	∞	∞	2	2
∞	∞	∞	∞	∞	0
11	0	∞	∞	∞	0
1	1	1	1	1	
11	0	0	0	0	

Here RRC = 2, CRC = 11 $\therefore r = 13$

$\hat{c}(7) = \hat{c}(4) + r + A(4,3)$

$= 25 + 13 + 12$

$= 50$

To compute $\hat{c}(8)$ make 1st row, 4th row, 4th column and 5th column entries as ∞ and also make $A[4,1] = A[5,1] = \infty$.

∞	∞	∞	∞	∞	0
12	∞	11	∞	∞	11
0	3	∞	∞	∞	0
∞	∞	∞	∞	∞	0
∞	0	0	∞	∞	0
1	1	1	1	1	
0	0	0	0	0	

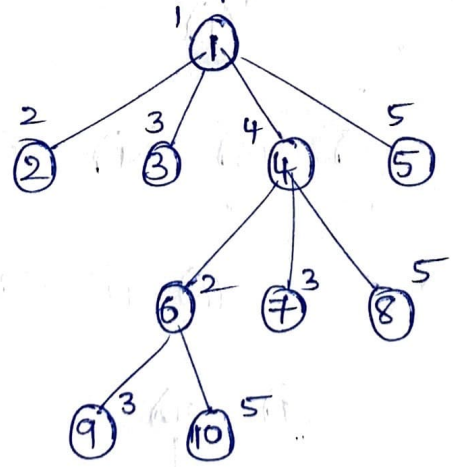
Here RRC = 11, CRC = 0 $\therefore r = 11$

$\hat{c}(8) = \hat{c}(4) + r + A(4,5)$

$= 25 + 11 + 0$

$= 36$

Since node 6 has optimum cost expand its children



To compute $\hat{c}(9)$ make 1^{st}, 4^{th}, 2^{nd} rows, 4^{th}, 2^{nd}, 3^{rd} columns as ∞ and make $A[4,1] = A[2,1] = A[3,1] = \infty$.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 2 \\ 0 \\ 11 \end{matrix}$$

Here RRC = 13, CRC = 0 $\therefore r = 13$

$$\hat{c}(9) = \hat{c}(6) + r + A(2,3)$$
$$= 28 + 13 + 11$$
$$= 52$$

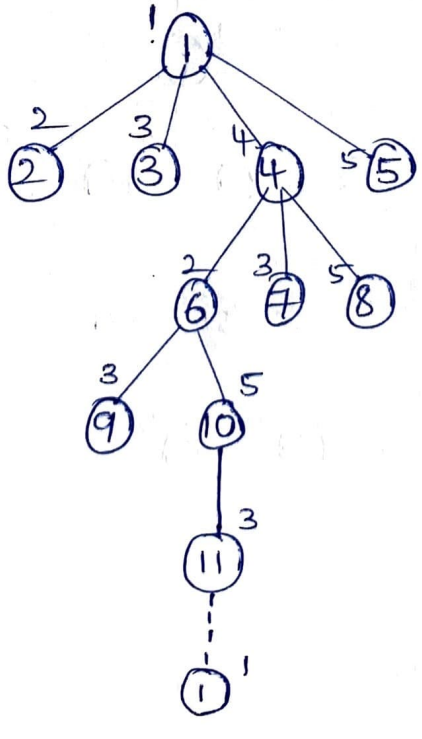
To compute $\hat{c}(10)$, make 1^{st}, 4^{th}, 2^{nd} rows, 4^{th}, 2^{nd}, 5^{th} columns as ∞ and make $A[4,1] = A[2,1] = A[5,1] = \infty$.

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

Here RRC = CRC = 0 $\therefore r = 0$

$$\hat{c}(10) = \hat{c}(6) + r + A(2,5)$$
$$= 28 + 0 + 0$$
$$= 28$$

Since node (10) has optimum cost, its only children (11) is generated



Optimum path is

$1 \to 4 \to 2 \to 5 \to 3 \to 1$

⊛ NP-Hard and NP-Complete Problems:

Basic Concepts: Depending on computing times, algorithms are clustered in 2 groups.

→ Problems whose solution times are bounded by polynomials of small degree. for example,

Linear Search - $O(n)$

Binary Search - $O(\log n)$

Insertion Sort - $O(n^2)$

Merge Sort, Quick Sort - $O(n \log n)$

Matrix Multiplication - $O(n^3)$.

→ Problems whose solution times are bounded by Non-Polynomial / Exponential. for example,

0/1 Knapsack - $O(2^n)$

Sum of Subsets - $O(2^n)$.

Graph Coloring - $O(2^n)$.

Hamiltonian Cycle - $O(2^n)$.

Travelling Sales Person - $O(2^n)$.

Problems in the second group requires vast amounts of time to execute that even moderate size problems can not be solved.

P Class: A class of problems that are solvable in Polynomial Time by deterministic algorithms.

Examples: Sorting and Searching.

An Algorithm is said to be solvable in Polynomial time if the no: of steps required to complete the algorithm for a given input is $O(n^k)$ $\forall$ $k \geq 1$, where n is the input size.

In deterministic algorithm all statements are clear. the Algorithm in which the result of every algorithm is uniquely defined are known as the deterministic algorithm.

NP Class: A class of problems that are solvable in Polynomial time by non-deterministic algorithms. Examples are 0/1 knapsack Problem, Travelling sales Person problem, Su-do-ku etc..

NP problems are checkable/verified in Polynomial time. It means that for given solution of problem, we can check whether the solution is correct or not in Polynomial time.

For example, Su-do-ku game needs exponential time. to find solution. But checking whether the

the solution is correct or not in polynomial time.

So, we always try to solve exponential time algorithm in Polynomial time. If Polynomial time algorithm cannot write exponential time algorithm, try to write non-deterministic algorithm which can be solved in polynomial time.

In Non-Deterministic algorithm, most of the statements are cleared but some statements are not clear about how they work. In future, somebody may find about how they work. this algorithm is known as Non-Deterministic algorithm.

for example, Currently we have Binary search algorithm which requires $O(\log n)$ time. We are trying to find algorithm which require $< \log n$ time or Constant time. But we can't find algorithm which require constant time. So, finally write non-deterministic algorithm for Binary Search.

To specify Non-deterministic algorithm, we introduce three functions.

(i) Choice (S): Abitarily chooses one of the elements of the set S.

(ii) Failure (): Signals an Unsuccessful completion

(iii) Success (): Signals a successful completion.

Ex: Non-deterministic algorithm for Searching

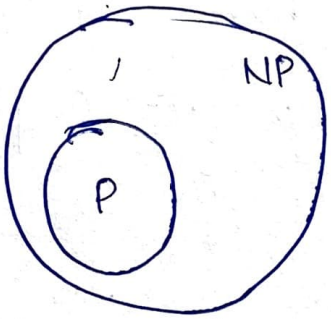
```
Algorithm my Search (A, x, n)
{
    j := Choice (1,n);
    if A[j] = x then
    {
        write j
        Success ()
    }
    write 0;
    Failure ()
}
```

So, the time complexity of this algorithm is O(1). But it is Non-deterministic algorithm. In future, if someone finds solution for this in O(1) time, algorithm becomes Deterministic.

the Assignment statement j := Choice (1, n) could result in j being assigned any one of the integers in the range (1, n). There is no rule how this choice is be made. the Failure () and Success() signals are used to define a computation of the algorithm.

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 successfully.

A Non-deterministic algorithm terminates unsuccessfully iff there exists no set of choices leading to a success signal. the computing times for Choice(), Success() and failure() are taken to be O(1).



Commonly believed relationship between P & NP

NP-Hard: A Problem is NP-Hard if all problems in NP class are polynomial time reducible to it, even though it may not be in NP itself. NP Hard problems are as hard as NP Complete problems. NP Hard problem need not to be in NP class.

Reduction: Suppose we have two decision problems P1 and P2

P1	P2
Input: I_1	Input: I_2
Algorithm: ?	Algorithm: B.

Suppose if P1 problem can be solved by using algorithm of problem P2, then there is noneed to write algorithm for P1. It is denoted as

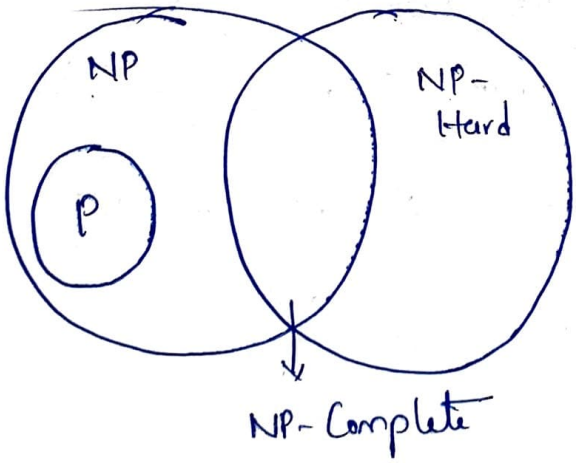
$$P1 \leq_p P_2 \quad \text{or} \quad P1 \propto P_2$$

NP- Complete : Group of problems which are both in NP and NP-Hard are known as NP- Complete problem. All NP- Complete problems are NP- Hard but not all NP Hard problems are NP- Complete.

A problem L_1 is NP- Complete iff it satisfies two conditions

i) L_1 is in NP ie $L_1 \in NP$

ii) Every problem L_2 in NP is Polynomial reducible to L_1 i.e $L_2 \propto L_1$



Commonly believed relation between P, NP, NP-Hard & NP- Complete

Examples for NP-Hard class problems: Travelling Sales Person problem, Vertex Cover, Set Cover and Circuit Satisfiability problem etc..

Examples for NP-Complete class problems: Boolean Satisfiability problem, Hamiltonian cycle, Sub of Subset, Knapsack problem, Graph Coloring problem etc..

(*) Cook's Theorem: Cook's theorem states that satisfiability is in P if and only if P = NP. Cook's theorem states that the boolean SAT is NP-Complete.

By definition SAT is in NP

Assume NP = P, then SAT is in P

Note: Boolean Satisfiability or Simply SAT is the problem of determining if a boolean formula is satisfiable or not.

If the boolean variables can be assigned values such that the formula turns out to be True, then we say that the formula is satisfiable.