mood-book

UNIT-V

BRANCH AND BOUND

GENERAL METHOD

The backtracking algorithm is effective for decision problems, but it is not designed for optimisation problems. This drawback is rectified in case of branch and bound technique. In this also we will use bounding function that is similar to backtracking

→ Bounding functions are used to kill the nodes without generating all their child nodes.

→ 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.

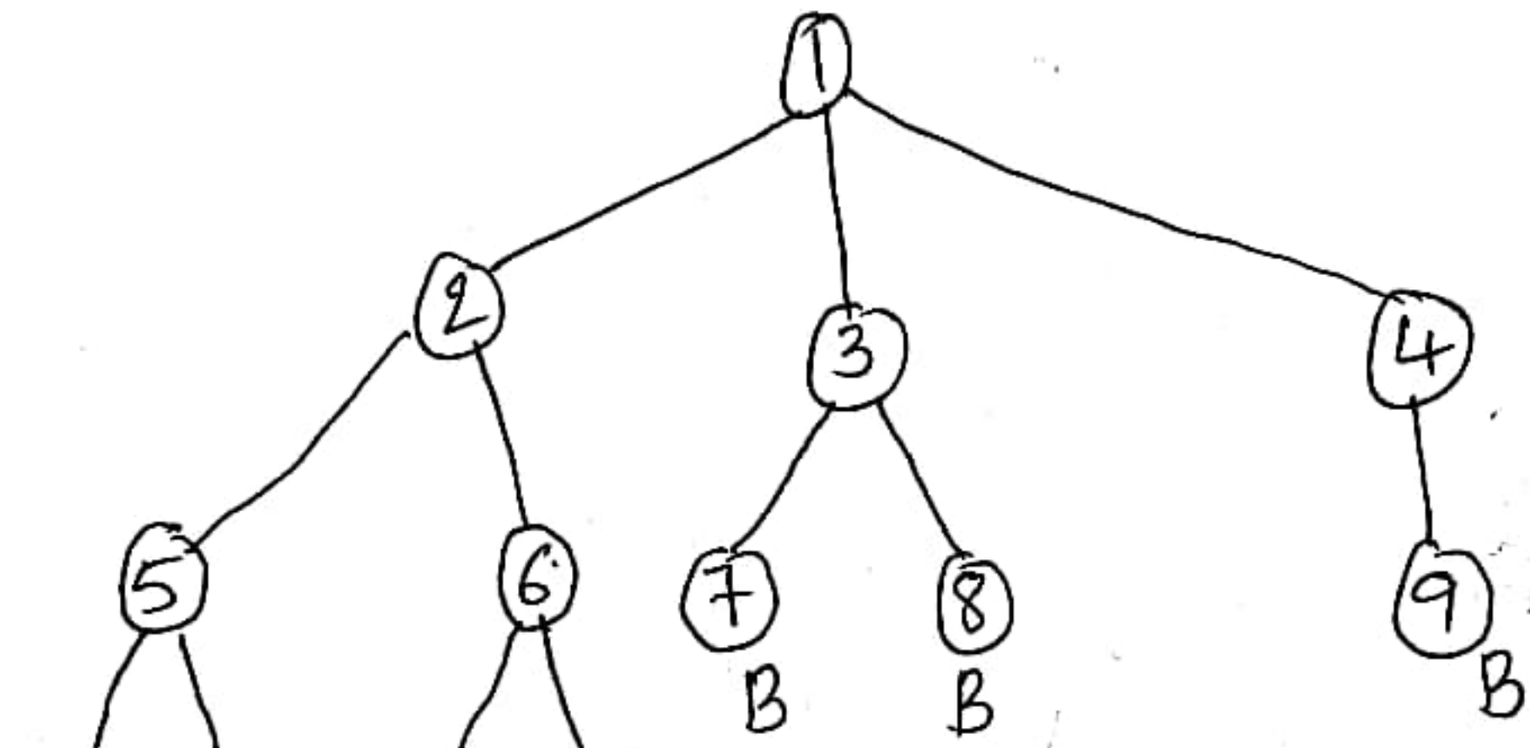
→ A maximization problem can be easily converted into a minimization problem by changing the sign of the objective function.

We will use 3 types of search strategies in branch and bound.

1) FOFIBB search
2) LIFOBB search
3) LCBB. search

FIFOBB (First In First out Branch and Bound)

search or BFS search



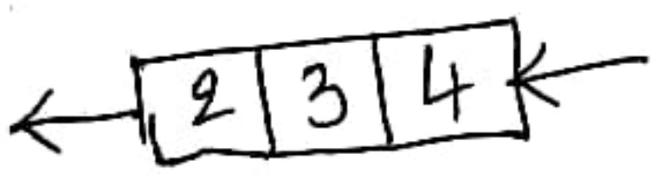
Example state space tree

For this, we will use the data structure queue.

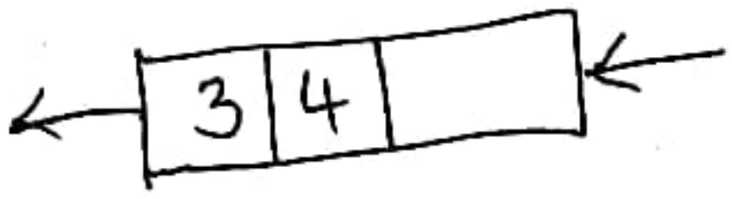
Assume that node 12 is the answer (an) node.

In FIFOBB search, first we will take E-node as node 1. Next we generate the children of node 1. We will place all these live nodes.

in a queue.



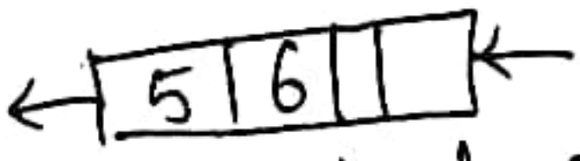
Next we will delete first element from the queue i.e node 2.



Next, generate children of node 2 and place in the queue.

Generate children of node 3. i.e 7,8. If we assume that these nodes are not satisfying bounding function [exceeding bounding function limit], then these nodes will be killed by bounding function. So we will not include 7,8 in the queue

Delete node 4 and generate its child nodes

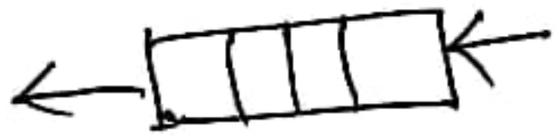


Node 9 is generated from 4, and killed by bounding function.

Delete 5 and generate it child nodes.

Nodes 10 and 11 are generated and killed by bounding function.

Delete 6 from queue and generate its child nodes

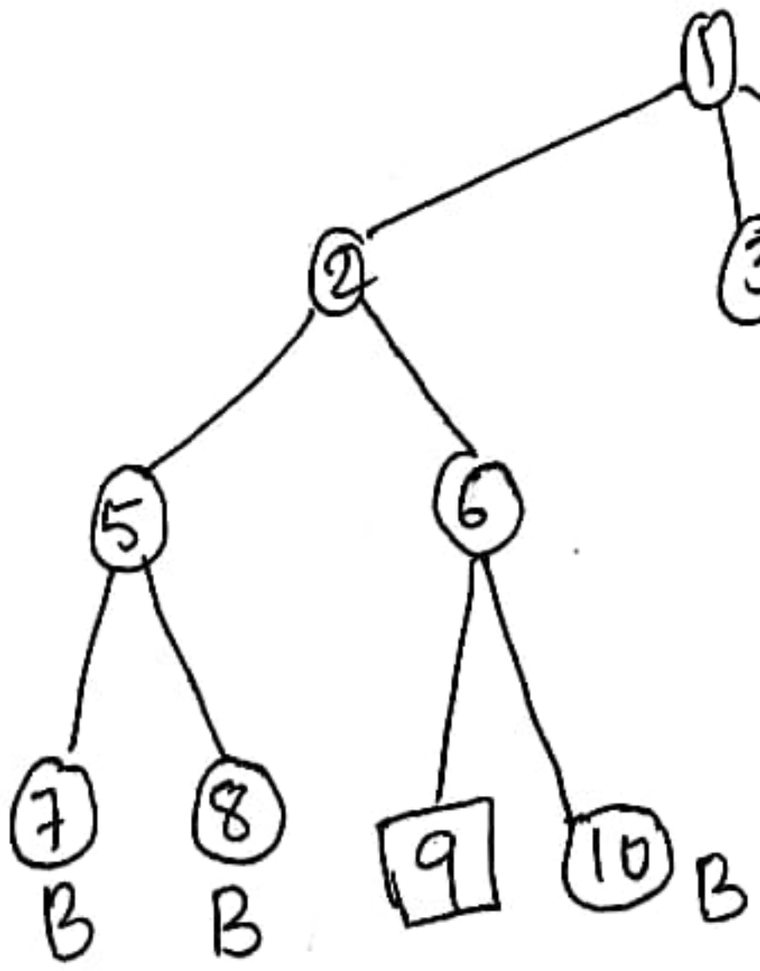


The child of node 6 is 12 and it is satisfying the constraints (conditions) of the problem, which is the answer node. So FIFOBB search terminates.

LIFOBB (Last In First out Branch and Bound)

Search or similar to DFS search.

For this we will use the data structure stack



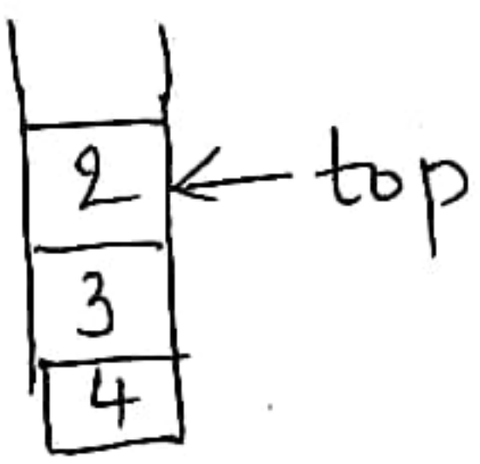
Initially stack is empty

stack.

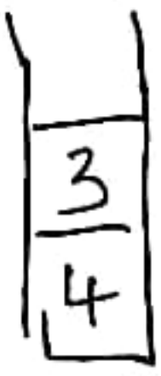
Example state space tree


stack.

Generate children of node1 and place(push) these live nodes into the stack, with left most child should be on top of stack.

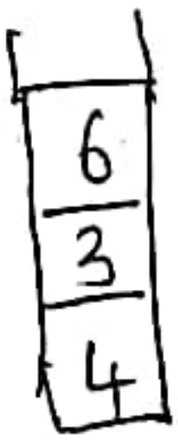


Remove(pop) top element 2 from stack and



generate the children of it, push those live nodes 5, 6 onto top of stack

pop node 5 and generate its children 7, 8



which are killed by bounding function. so we will push 7,8 into stack

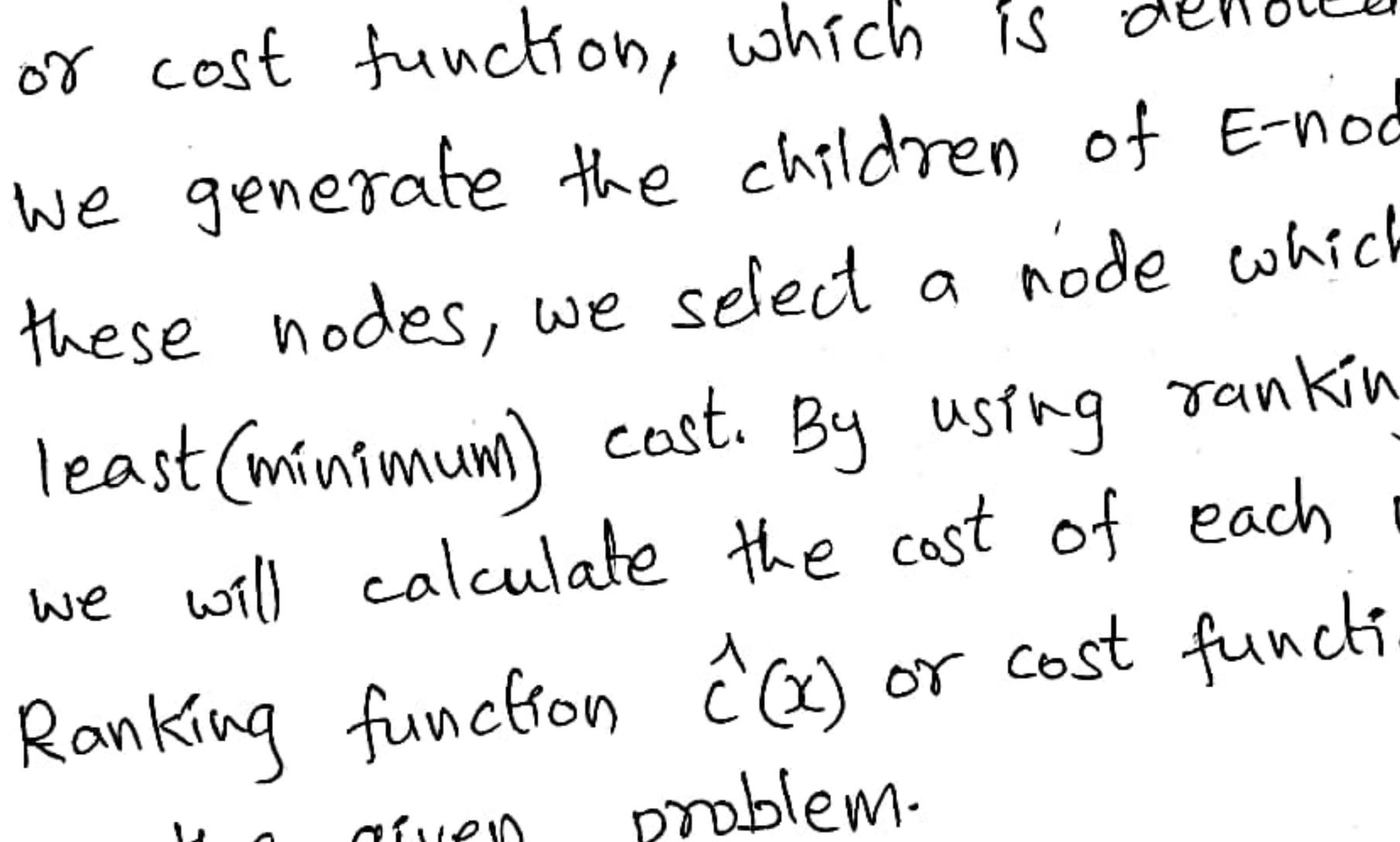
Remove(pop) 6 from stack and generate children 9, 10. 9 is and answer node so search process terminates.

As we have found an answer node, we will not expand next branches 3, 4 of the state space tree.

LCBB (Least (minimum) Cost Branch and Bound)

Search

In this we will use a ranking function or cost function, which is denoted by $\hat{c}(x)$. we generate the children of E-node. Among these nodes, we select a node which has least (minimum) cost. By using ranking function we will calculate the cost of each node. Ranking function $\hat{c}(x)$ or cost function depends on the given problem.



Example state space tree

Generate children of node 1, the children are 2,3,4

By using cost function we will calculate the cost of 2,3,4 nodes. If suppose

$\hat{c}(2)=12$, $\hat{c}(3)=13$, $\hat{c}(4)=16$.

Now, we will select a node which has least (minimum) cost i.e, node 2.

→ Generate children of node 2. They are 5, 6. Between 5 and 6 we will select 6 since its cost is least (minimum).

→ Next generate children of node 6 i.e 7 and 8. we will select node 8 since its cost $\hat{c}(8)=25$ is minimum. than $\hat{c}(7)=30$.

→ moreover if node 8 is the answer node. So, we terminate the search process.

Control Abstraction for LC-search (LCBB)

Let t be a state space tree and c() a cost function for the nodes in t. If x is a node in t, then c(x) is the minimum cost of any answer node in the subtree with root x. Thus, c(t) is the cost of a minimum-cost answer node in t.

LC-Search Algorithm

```
listnode=record
{
  listnode *next, *parent;
  float cost;
}
```

```
1. Algorithm LCsearch(t)
2. // search t for an answer node:
3. {
4.   If *t an answer node then output *t and return;
5.    E:=t; // E-node
6.    Initialize the list of live nodes to be empty;
7.    repeat
8.    {
9.      for each child x of E do
10.     {
11.       if x is an answer node then output the path
12.       from x to t and return;
13.       Add(x); // x is a new live node.
14.       (x->parent):=E; // Pointer for path to root
15.     }
16.     if there are no more live nodes then
17.     {
18.       write("No answer node"); return;
19.     }
20.     E:=Least();
21.   } until(false);
22. }
```

LCSearch algorithm uses $\hat{c}$ to find an answer node.

Least() function is used delete a live node from the list of live nodes. whose cost c() is least

Add(x) function is used to add a new live node x to the list of live nodes.

→ LCsearch algorithm outputs the path from the answer node to root node t.

parent field of node x points to parent node of x.

→ variable E always points to the current E-node. The root node is the first E-node.

→ The for loop of line 9 examines all the children of the E-node. If one of the children is an answer node, then the algorithm outputs the path from x to t. and terminates.

→ If a child of E is not an answer node, then it becomes a live node. It is added to the list of live nodes. and its parent field set to E (line 14).

→ when all the children of E have been generated, E becomes a dead node and line 16 is reached.

This happens only if none of E's children is an answer node. So, the search must continue further.

→ If there are no live nodes left, then the entire state space tree has been searched and no answer nodes found. The algorithm terminates in line 18.

→ Otherwise, Least() function correctly chooses the next E-node and the search continues from here.

Applications of Branch and Bound

1) TSP problem
   LCBB solution and FIFOBB solution
2) 0/1 Knapsack problem
   LCBB solution and FIFOBB solution.

LCBB solution for TSP problem

Given a graph $G_1 = (V, E)$

cost $c_{ij} = \infty$ if $(i, j) \notin E(G)$.

Assume that every tour starts and ends at vertex 1.

cost function $\hat{c}(x)$.

Reduced cost matrix A row or column is said to be reduced if it contains atleast one zero and all remaining entries are non-negative. A matrix is reduced iff every row and column is reduced.

- If a constant t is chosen to be minimum entry in row i or column j then subtracting it from all entries in row i (or column j) will introduce a zero into a row i (col j).
- The total amount subtracted from the columns and rows is lower bound on the length of a minimum cast tour and can be used as the $\hat{c}(x)$ value for the root of state space tree.
- With every node in the state space tree, we associate a reduced cost matrix.

Let A be the reduced cost matrix for node R. Let S be the child node of R such that the edge(R,S) corresponds to including edge(i,j) in the tour.

The reduced cost matrix for node S can be obtained as follows.

1) change all entries in row i, column j of A to ∞.
2) set A(j, 1) to ∞.
3) Apply row reduction and column reduction except for rows and columns containing ∞.
4) The total cost for node S can be calculated as

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

where r is the total amount subtracted in step 3.

* Ex Solve the following instance of travelling salesperson problem by using LCBB

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

sol:-

Apply row reduction

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

$\Rightarrow$

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

A-Reduced cost matrix

Apply column reduction

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞
	1	-	3	- - = 4

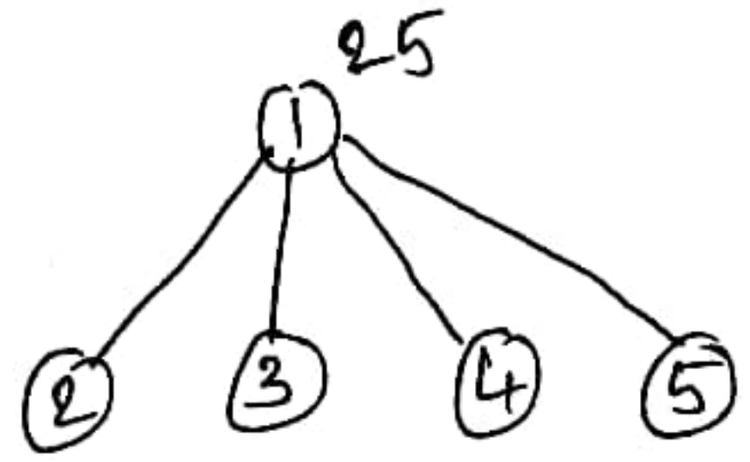
$\Rightarrow$

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

Total amount subtracted

$r = 21 + 4 = 25$

$\hat{c}(1) = 0 + 0 + 25 = 2$



part of state space tree

→ consider the path (1,2): change all entries of 1st row and 2nd column of reduced matrix to ∞ and set A(2,1) to ∞

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

Apply row reduction and column reduction. But all rows and columns are reduced

row reduction = 0+0+0+0 = 0
col reduction = 0+0+0+0 = 0

$$r = 0+0 = 0$$

$$\hat{c}(2) = \hat{c}(1) + A(1,2) + r$$

$$\hat{c}(2) = 25+10+0 = \underline{35}$$

consider the path (1,3): change all entries of 1st row and 3rd column of reduced matrix A to ∞ and set A(3,1) to ∞. we will get the following matrix

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \begin{matrix} - \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$$\underline{11} \quad 0 \quad - \quad 0 \quad 0$$

All rows are reduced
col reduction applied to col 1 $r = 0+11 = 11$.

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

consider the path (1,4): change all entries of 1st row and 4th column to ∞ and set A(4,1) to ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \\ 0 & 0 & 0 & - & 0 \end{bmatrix} \begin{matrix} - \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$ All rows and columns are reduced.

$r = 0 + 0 = 0$

$\hat{C}(4) = \hat{C}(1) + A(1,4) + r$

$\hat{C}(4) = 25 + 0 + 0 = \underline{25}$

Consider the path (1,5):

1st row and 5th column $= \infty$

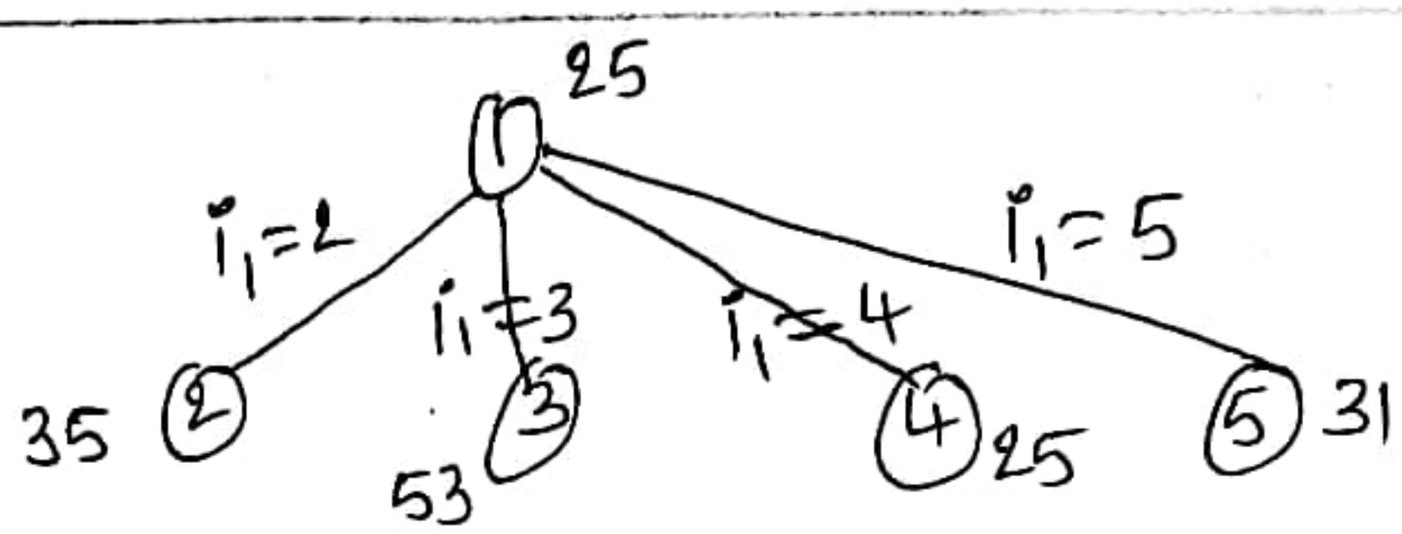
$A(5,1) = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \\ 0 & 0 & 0 & 0 & - \end{bmatrix} \begin{matrix} - \\ 2 \\ 0 \\ 3 \\ 0 \\ \hline 5 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

$r = 5 + 0 = 5$

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

$\hat{C}(5) = 25 + 1 + 5 = \underline{31}$



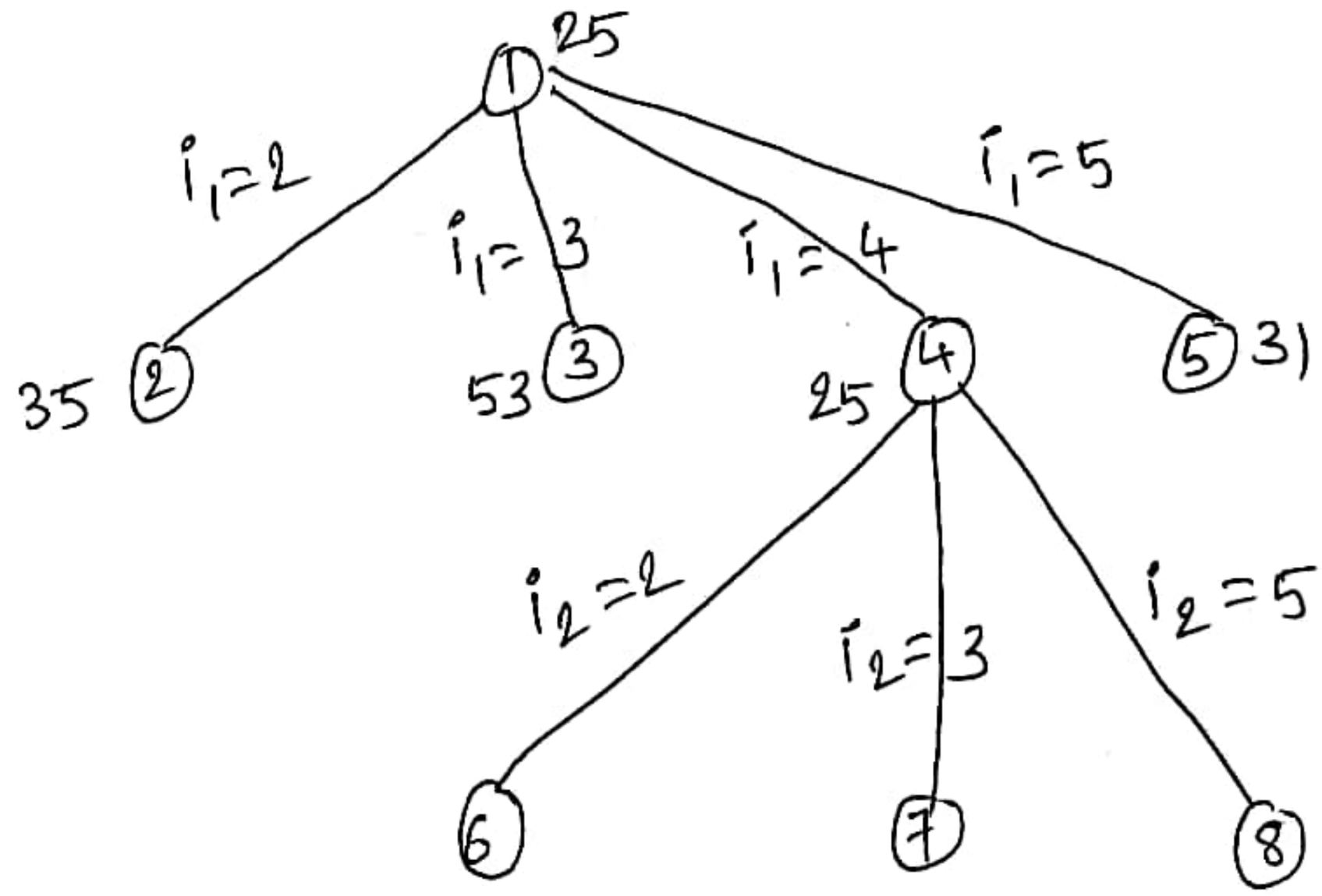
Since the minimum cost = 25

Include edge (1,4) in the tour

The matrix obtained for path (1,4) is considered as reduced cost matrix

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

→ starting from node 1 and after that visiting node 4, next the travelling salesperson can go to node 2 or city 3 or city 5



consider the path (4,2):

set 4th row and 2nd col = ∞

A(2,1) = ∞

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

$r = 0+0$

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

$\hat{C}(2) = 25 + 3 + 0$

$\hat{C}(2) = \underline{28}$

consider the path (4,3)

set 4th row and 3rd col to ∞

A(3,1) = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \\ 11 & 0 & - & - & 0 \end{bmatrix} \begin{matrix} - \\ 0 \\ 2 \\ - \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$r = 2+11 = 13$

$\hat{C}(3) = \hat{C}(4) + A(4,3) + r = 25 + 12 + 13 = \underline{50}$

consider the path (4,5): 4th row & 5th col = ∞

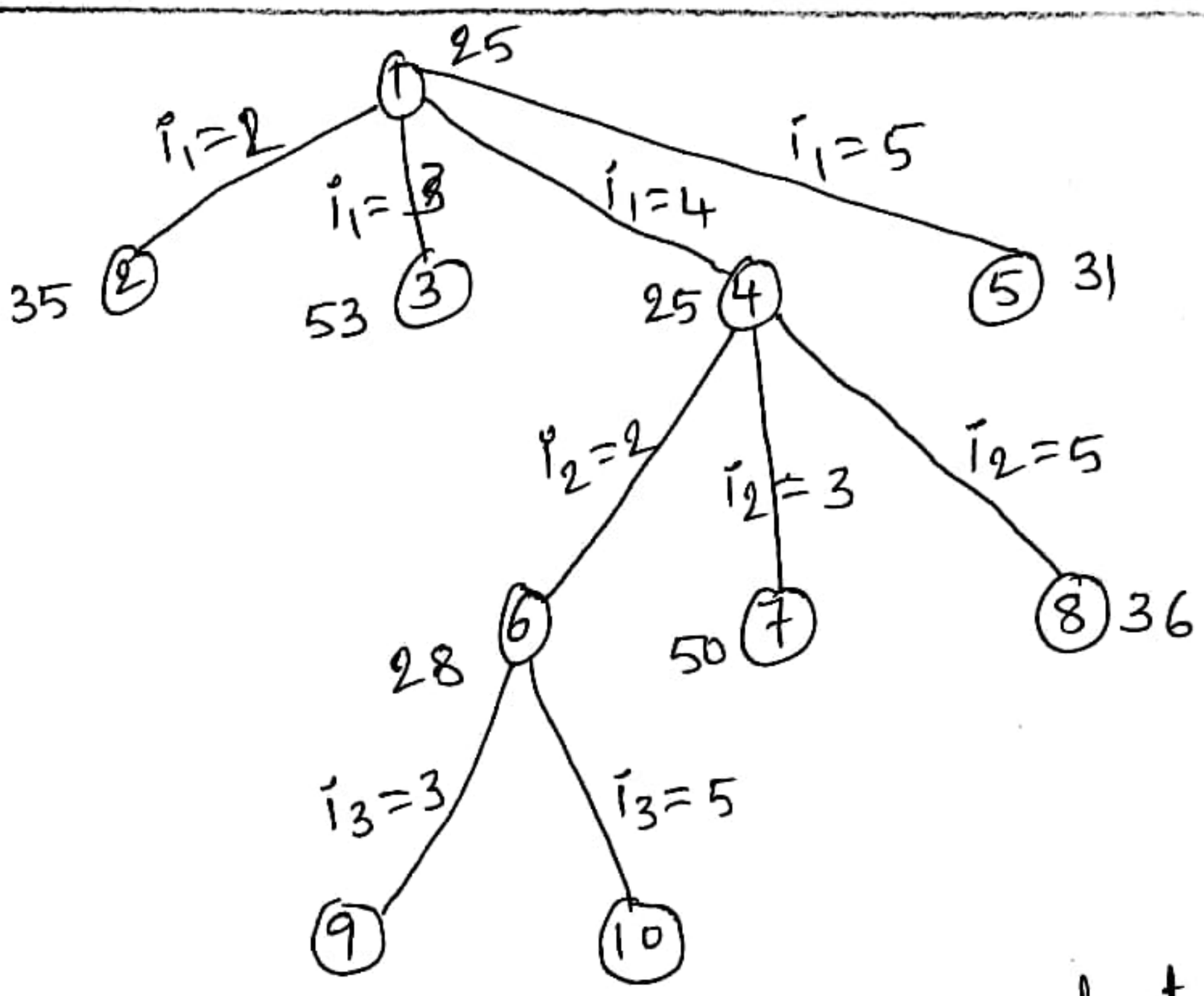
A(5,1) = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \\ 0 & 0 & 0 & - & - \end{bmatrix} \begin{matrix} - \\ 11 \\ 0 \\ - \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

$r = 11+0$

$r = 11$

$\hat{C}(5) = \hat{C}(4) + A(4,5) + r = 25 + 0 + 11 = \underline{36.}$



since the minimum cost is 28. select node 2

Include edge (4,2) in the tour

$1 \rightarrow 4 \rightarrow 2$

The matrix obtained for path (4,2) is considered as reduced cost matrix.

$$A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

→ Next consider the path (2,3) 2^{nd} row, 3^{rd} col $= \infty$

$A(3,1) = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} 2 \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix} \quad r = 2 + 11$$

$r = 13$

11

$$\hat{C}(3) = \hat{C}(2) + A(2,3) + r = 28 + 11 + 13 = \underline{52}$$

consider the path (2,5)

2nd row, 5th col set to ∞ $A(5,1) = \infty$

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

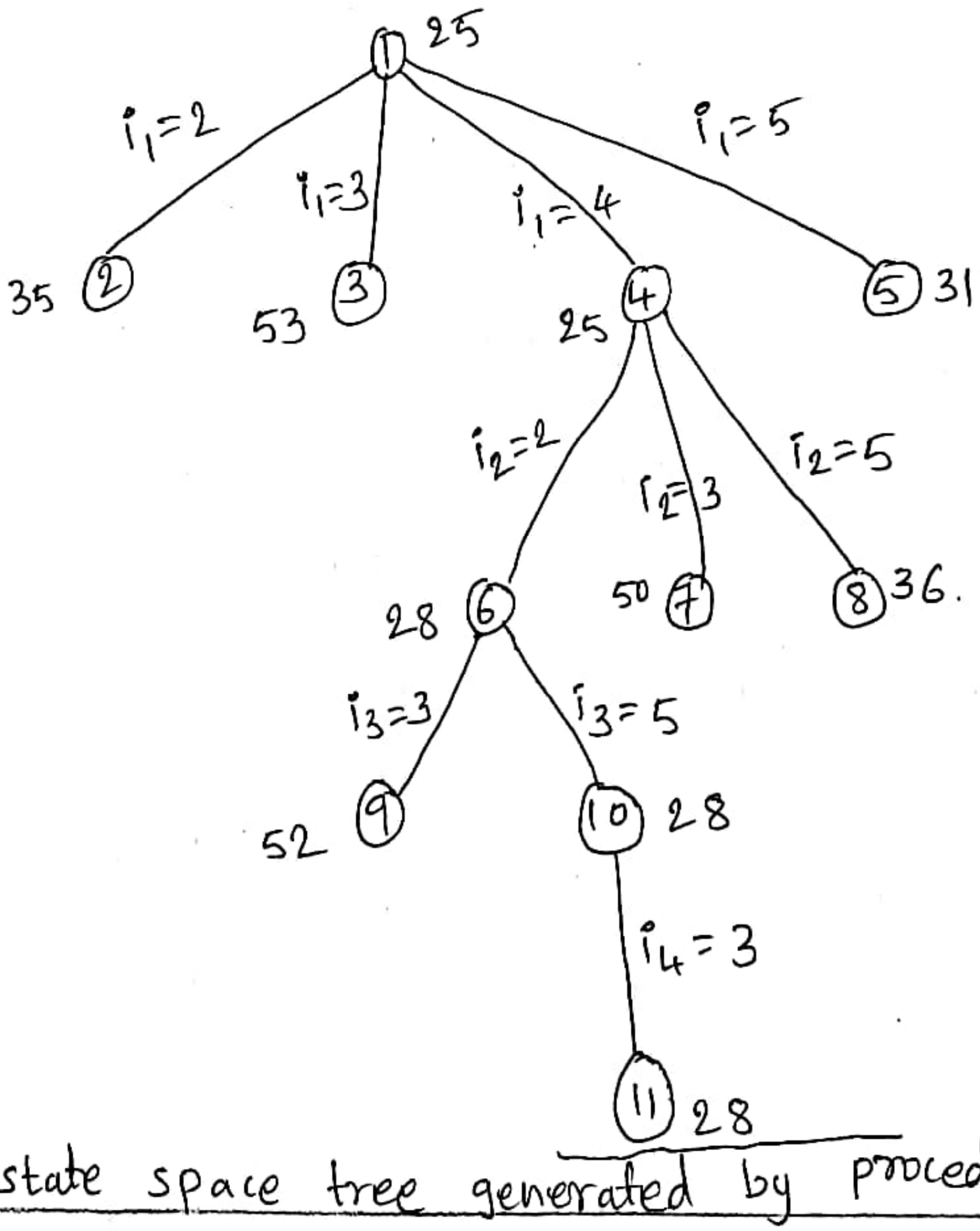
All rows and columns are reduced

$r = 0 + 0 = 0$

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

since the minimum cost is 28, select node 5.

The above matrix obtained for path (2,5) is considered as reduced cost matrix



state space tree generated by procedure LCBB.

consider the path (5,3)

set 5th row = ∞, 3rd col = ∞, A(3,1) = ∞

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

$r = 0 + 0 = 0$

$\hat{c}(3) = \hat{c}(5) + A(5,3) + r = 28 + 0 + 0 = \underline{28}$

∴ The minimum cost tour is

$1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$

cost = 10 + 6 + 2 + 7 + 3 = 28 km

* 0/1 KNAPSACK PROBLEM *

LCBB SOLUTION

0/1 knapsack problem can be stated as

maximize profit $P = P_1x_1 + P_2x_2 + \cdots + P_nx_n$

subject to the constraints

$w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq M.$

$x_i = 0$ or $1.$

→ But the Brand and Bound method deals only with the minimization problems.

→ so a maximization problem can be converted to a minimization problem just by changing the sign of profit as follows.

we modify the knapsack problem to the minimization problem as follows

minimize $P = -P_1x_1 - P_2x_2 - \cdots - P_nx_n$

subject to the constraints

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq M.$$

$$x_i = 0 \text{ or } 1.$$

* Ex Draw the state space tree and find an optimal solution to the o/1 knapsack problem $n=4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$, $M=15$. $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$ by using LC BB

Sol:- Given four objects are in decreasing order of profit/weight ration

$$\frac{P_1}{w_1} > \frac{P_2}{w_2} > \frac{P_3}{w_3} \geq \frac{P_4}{w_4}$$

$$\frac{10}{2} > \frac{10}{4} > \frac{12}{6} \geq \frac{18}{9}$$

$$5 > 2.5 > 2 \geq 2$$

upper bound on total profit is obtained by greedy fractional (normal) knapsack where $0 \leq x_i \leq 1$

Lower bound on total profit is obtained by o/1 knapsack where $x_i = 0$ or $x_i = 1$ but not a fraction. objects cannot be divided into fractions.

$M = 15 kg$

$-18 \times \frac{3}{9} = -6$	$9 \times \frac{3}{9}$	$15 kg$
-12	6	
-10	4	
-10	2	

-38 Total profit

(x_1, x_2, x_3, x_4)

$(1, 1, 1, 1/3)$

$w = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$

$w = 2 \times 1 + 4 \times 1 + 6 \times 1 + 9 \times 1/3$

$w = 2 + 4 + 6 + 3 = \underline{15}$

$P = P_1 x_1 + P_2 x_2 + P_3 x_3 + P_4 x_4$

$P = 10 \times 1 + 10 \times 1 + 12 \times 1 + 18 \times 1/3$

$P = -10 - 10 - 12 - 6 = -38$

6	-12
4	-10
2	-10

-32 Total profit.

(x_1, x_2, x_3, x_4)

$(1, 1, 1, 0)$

$w = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$

$w = 2 \times 1 + 4 \times 1 + 6 \times 1 + 9 \times 0$

$w = 2 + 4 + 6 + 0 = \underline{12}$

$P = P_1 x_1 + P_2 x_2 + P_3 x_3 + P_4 x_4$

$P = -10 \times 1 - 10 \times 1 - 12 \times 1 - 18 \times 0$

$P = -10 - 10 - 12 - 0 = -32.$

At node 1

upper bound on profit $= -38$

lower bound on profit $= -32$

we will expand node1 and generate child nodes node 2 and node 3.

At node 2 we will get same profit values, but at node 3 we have to calculate the upper bound and lower bound on total profits.

At node 3

(x_1, x_2, x_3, x_4)

$(0, 1, 1, 5/9)$

$w = 0+4+6+9\times\frac{5}{9} = 15$

$p = -0-10-12-18\times\frac{5}{9} = -32$

(x_1, x_2, x_3, x_4)

$(0, 1, 1, 0)$

$w = 0+4+6+0 = 10$

$p = -0-10-12-0 = -22$

node2	node3
-38	> -32
-32	> -22

so don't expand node 3 only since its cost is least.

Expand node2

At node 5

(x_1, x_2, x_3, x_4)

$(1, 0, 1, 7/9)$

$w = 2+0+6+9\times 7/9 = 15$

$p = -10-0-12-18\times\frac{7}{9} = -36$

(x_1, x_2, x_3, x_4)

$(1, 0, 1, 0)$

$w = 2+0+6+0 = 8$

$p = -10-0-12-0 = -22$

node4	node5
-38	↔ -36
-32	↔ -22

Node 4 is giving more profit. It is LC (least cost) node. So expand node4

Don't expand node5

At node 7

(x_1, x_2, x_3, x_4)	(x_1, x_2, x_3, x_4)
$(1, 1, 0, 1)$	$(1, 1, 0, 1)$
$w = 2+4+0+9 = 15$	$w = 2+4+0+9 = 15$
$p = -10-10-0-18 = -38$	$p = -10-10-0-18 = -38$

node6	node7
-38	$\longleftrightarrow -38$
-32	$\longleftrightarrow -38$ ✓

Don't expand node 6

Expand (explore) node 7.

we will get two child nodes 8 and 9.

At node 8 total profit values are same as node 7. Compute profits at node 9.

At node 9

(x_1, x_2, x_3, x_4)	(x_1, x_2, x_3, x_4)
$(1, 1, 0, 0)$	$(1, 1, 0, 0)$
$w = 2+4+0+0 = 6$	$w = 2+4+0+0 = 6$
$p = -10-10-0-0 = -20$	$p = -10-10-0-0 = -20.$

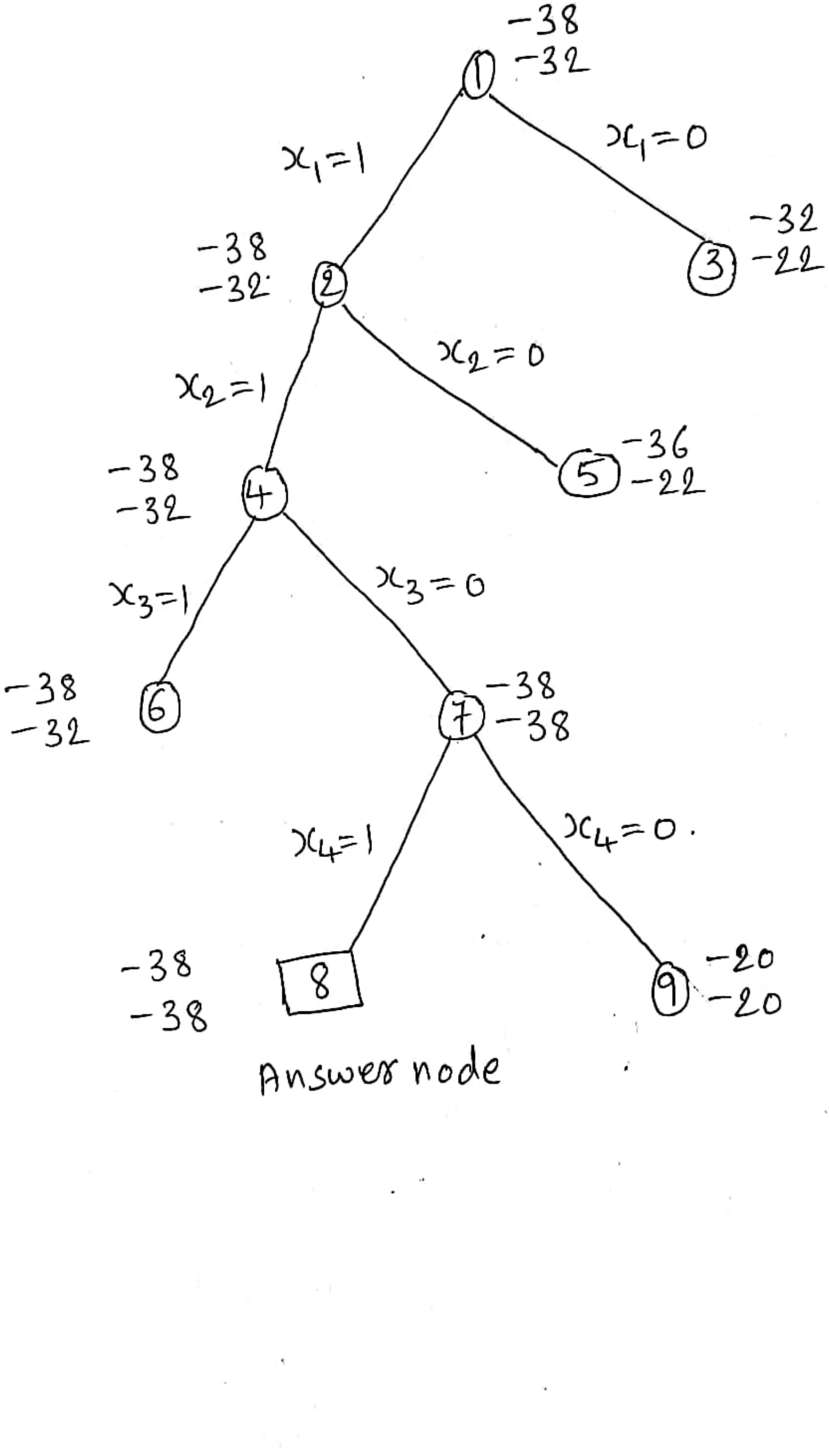
Give $n = 4$ objects. we have generated state space tree upto four levels. and found an answer node 8 which is giving maximum profit. 38

solution tuple $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$

$w = 2+4+6+9 \times 1/3 = 15$

$p = 10+10+12+18 \times 1/3 = 38$

state space tree generated by LCBB.



Answer node

* 0/1 KNAPSACK PROBLEM *

FIFOBB SOLUTION

Ex Draw the state space tree and find an optimal solution to the following 0/1 knapsack problem by using FIFOBB

n = 4 objects, M = 15kg

profits $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$

weights $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$.

sol:-

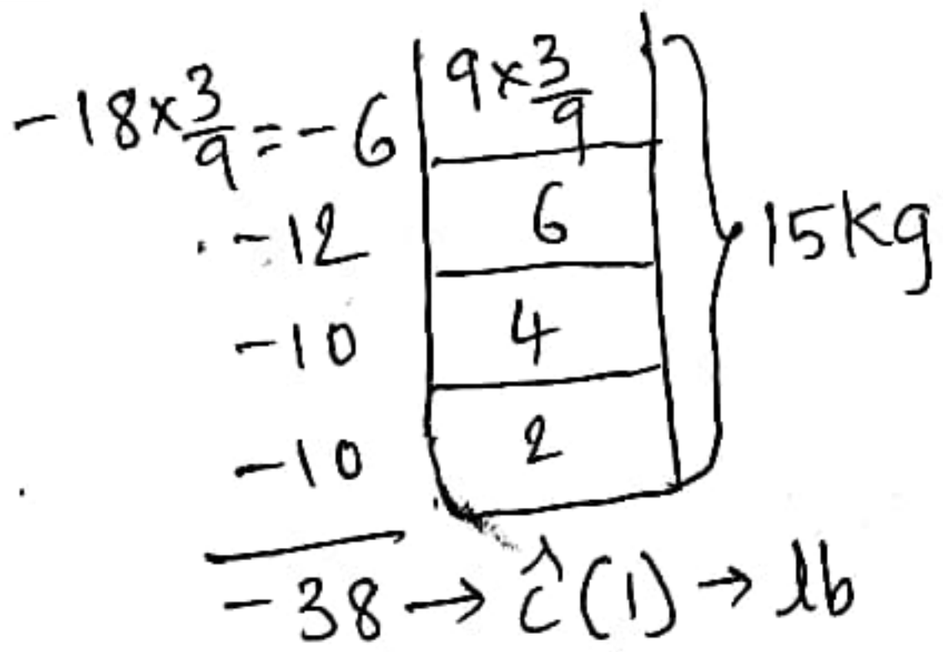
lower bound $\hat{c}()$ → fractional wts are allowed

upper bound $\hat{u}()$ → fractional wts are not allowed

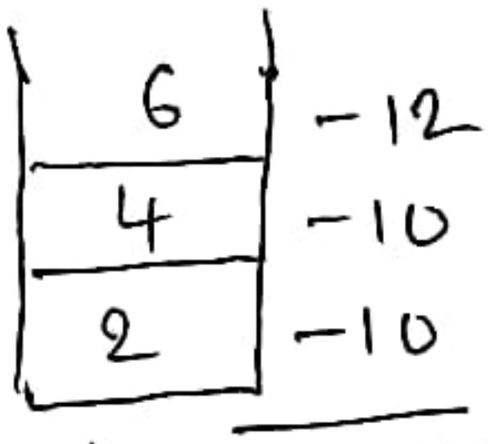
lb → lower bound

ub → upper bound

At node1



$-38 \rightarrow \hat{c}(1) \rightarrow lb$



$\hat{u}(1) = -32$ ↓

Take as global ub.

$\hat{c}(2) = \hat{c}(1) = -38$ $\quad \hat{u}(2) = \hat{u}(1) = -32$

At node3 $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 5/9$

$w = 0 + 4 + 6 + 9 \times 5/9 = 15$

$\hat{c}(3) = -0 - 10 - 12 - 18 \times \frac{5}{9} = -32$

$w = 0 + 4 + 6 = 10$

$\hat{u}(3) = -0 - 10 - 12 = -22$

Explore node2

cost of node4 is same as node2

At node5

(x_1, x_2, x_3, x_4)

$(1, 0, 1, 7/9)$

$w = 2+0+6+9\times 7/9 = 15$

$\hat{c}(5) = -10-0-12-18\times\frac{7}{9}$

$\hat{c}(5) = -36$

(x_1, x_2, x_3, x_4)

$(1, 0, 1, 0)$

$w = 2+0+6+0 = 8$

$\hat{u}(5) = -10-0-12-0 = -22$

At node6

(x_1, x_2, x_3, x_4)

$(0, 1, 1, 5/9)$

$w = 0+4+6+9\times 5/9 = 15$

$\hat{c}(6) = -0-10-12-18\times\frac{5}{9} = -32$

(x_1, x_2, x_3, x_4)

$(0, 1, 1, 0)$

$w = 0+4+6+0 = 10$

$\hat{u}(6) = -0-10-12-0 = -22$

At node7

(x_1, x_2, x_3, x_4)

$(0, 0, 1, 1)$

$w = 0+0+6+9 = 15$

$\hat{c}(7) = -0-0-12-18 = -30$

(x_1, x_2, x_3, x_4)

$(0, 0, 1, 1)$

$w = 0+0+6+9 = 15$

$\hat{u}(7) = -0-0-12-18 = -30$

$\uparrow lb(7)$

$lb(7) > \text{global ub}$

$-30 > -32$

↓ Then kill node7. Don't explore

At node9

(x_1, x_2, x_3, x_4)

$(1, 1, 0, 1)$

$w = 2+4+0+9 = 15$

$\hat{c}(9) = -10-10-0-18 = -38$

(x_1, x_2, x_3, x_4)

$(1, 1, 0, 1)$

$w = 2+4+0+9 = 15$

$\hat{u}(9) = -10-10-0-18 = -38$

$\hat{u}(9) < \text{global ub}$

$-38 < -32$

minimum

update **global ub = -38**

Now

$lb(5) > \text{global ub}$

$-36 > -38$

$lb(6) > \text{global ub}$

$-32 > -38$

$\therefore$ Kill node 5 and node 6.

Don't explore them

At node 10

$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$

we can't place all 4 objects into knapsack

since $w = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$

$w = 2\times1 + 4\times1 + 6\times1 + 9\times1 = 21 > 15$ kg.

greater than knapsack capacity

This is infeasible solution

Not satisfying bounding function

Kill node 10.

At node 11

(x_1, x_2, x_3, x_4)

$(1, 1, 1, 0)$

$w = 2+4+6+0 = 12$

$\hat{c}(11) = -10-10-12-0 = -32$

(x_1, x_2, x_3, x_4)

$(1, 1, 1, 0)$

$w = 2+4+6+0 = 12$

$\hat{u}(11) = -10-10-12-0 = -32$

$lb(11) > \text{global ub}$

$-32 > -38 \Rightarrow$ kill node 11.

At node 12

(x_1, x_2, x_3, x_4)
$(1, 1, 0, 1)$
$w = 2+4+0+9 = 15$
$\hat{c}(12) = -10-10-0-18 = -38$

(x_1, x_2, x_3, x_4)
$(1, 1, 0, 1)$
$w = 2+4+0+9 = 15$
$\hat{u}(12) = -10-10-0-18 = -38$

At node 13

(x_1, x_2, x_3, x_4)
$(1, 1, 0, 0)$
$w = 2+4+0+0 = 6$
$\hat{c}(13) = -10-10-0-0 = -20$

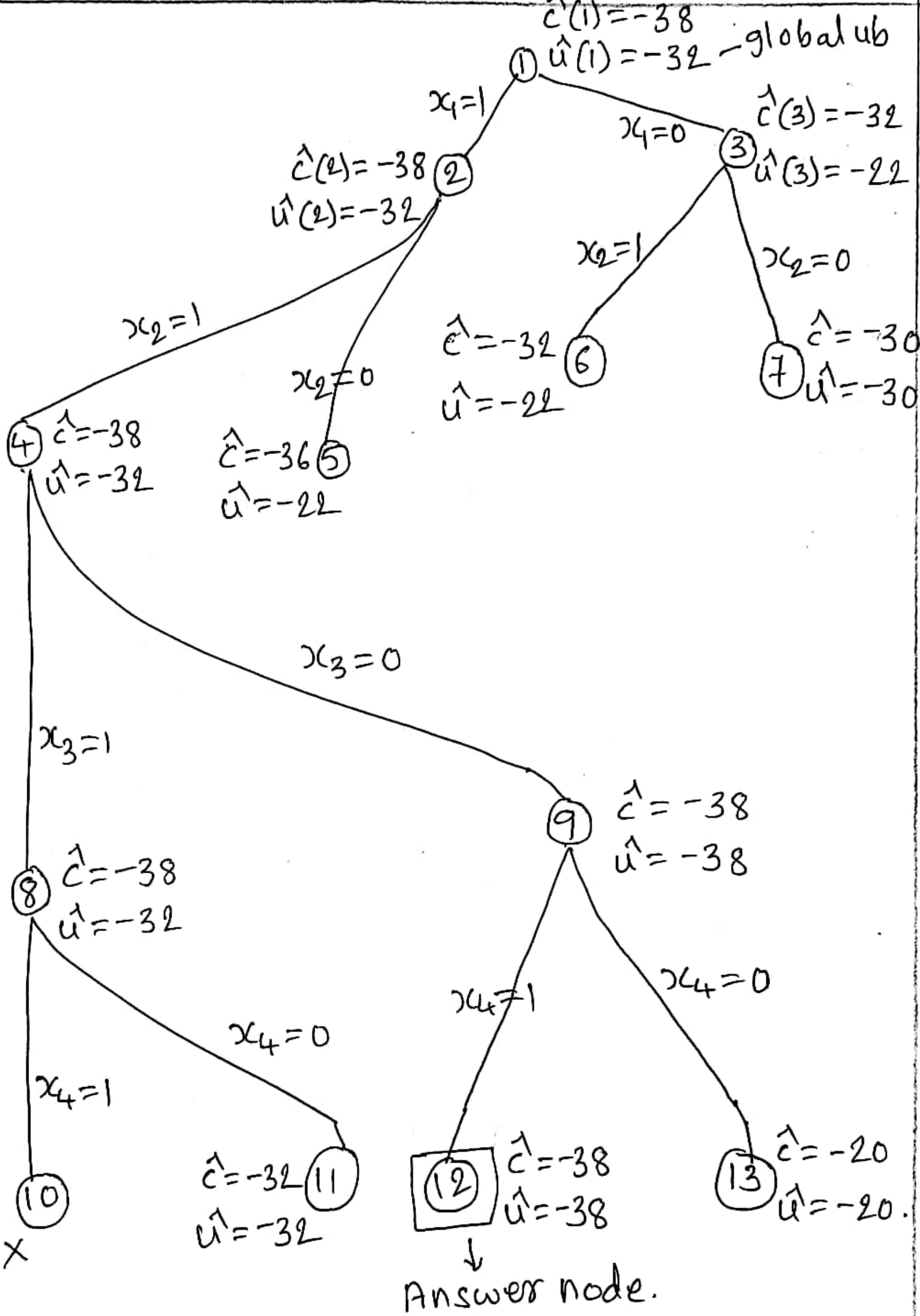
(x_1, x_2, x_3, x_4)
$(1, 1, 0, 0)$
$w = 2+4+0+0 = 6$
$\hat{u}(13) = -10-10-0-0 = -20$

we have generated the state space tree till four levels x_1, x_2, x_3, x_4. Given four objects.. we got the maximum profit at node 12. Therefore it is answer node.

solution tuple (x_1, x_2, x_3, x_4)
$(1, 1, 0, 1)$

$w = 2+4+0+9 = 15$kg

Profit = $10+10+0+18 = 38$.



state space tree generated by FIFOBB.

NP-HARD AND NP-COMPLETE PROBLEMS

BASIC CONCEPTS

Polinomial The sum of several terms that contain different powers of the same variable(s)

Ex $x^3+2x^2yz^2+yz+1 = F(x,y,z)$

order of this polynomial function is 3 i.e highest power.

Problems are of two types

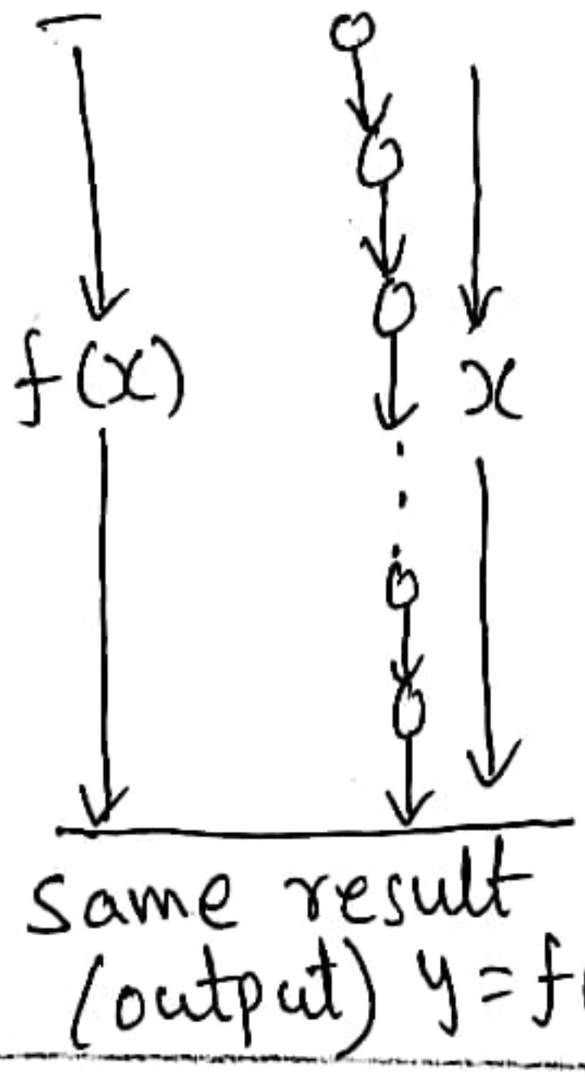
Tractable problems	Intractable problems
1) These are the problems that can be solved by a polynomial time algorithm. Time complexity function is a polynomial	2) Problems for which no polynomial time algorithm is known Time complexity function is nonpolynomial (means exponential function)
2) Ex binary search $O(\log n)$ merge sort $O(n \log n)$ Quick sort $O(n^2)$ matrix multiplication $O(n^3)$	2) Ex TSP problem $O(2^n n^2)$ Exponential time $\rightarrow 2^n$
These are simple problems which are called tractable problems	These are hard or intractable problems. No one has been able to develop a polynomial time algorithm for any any problem in this group.

DETERMINISTIC AND NONDETERMINISTIC ALGORITHMS

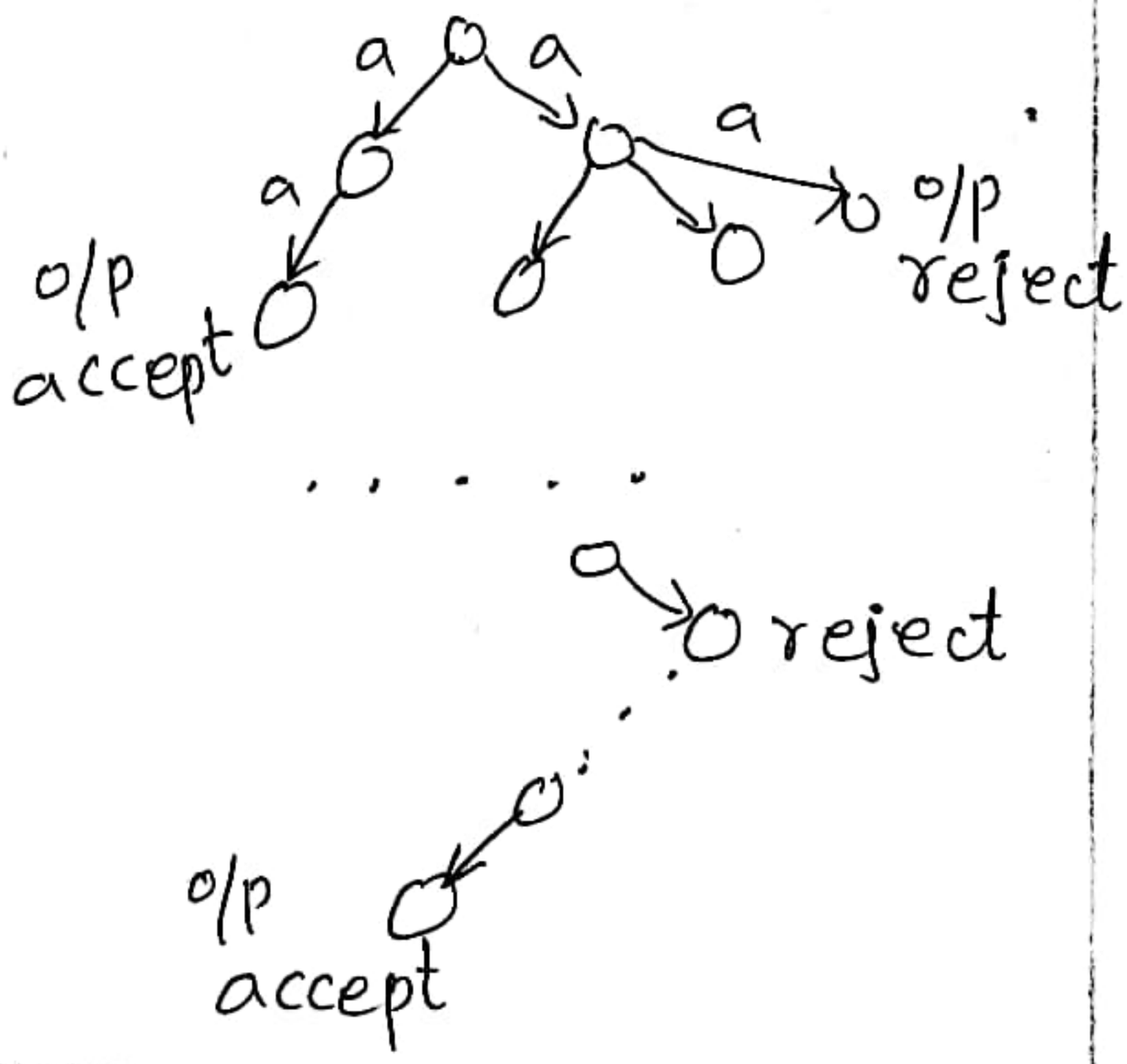
In computer science, a deterministic algorithm is an algorithm that produces (gives) same output (Ex y) for the same input (Ex x) in different runs

A nondeterministic algorithm is an algorithm that, even for the same input (x) can (may) exhibit different behaviors [gives output y,z,...] on different runs, as opposed to deterministic algorithm. There are several ways a nondeterministic algorithm may behave differently from one run to another run

Deterministic Algorithm



Nondeterministic algorithm



All the algorithms that we have studied in the first 4 units are examples of determinristic algorithms

Non-Determinristic Algorithms

Nondeterministic algorithms are written by using three functrons.

1) select(s) or choice(s): arbitrarily selects or chooses one element from set S.
2) Success(): signals successful completron of algorithm.
3) Farlure(): signals unsuccessful completon of algorithm

Nondeterminristic algorithm for searching

```
Algorithm nd_search(a,n,x)
{
// a is array of size n
// x is element to be searched
for i=1 to n do
                    // Each call of select() selects a
{
   j=select(a,n); // different location j. select a
   if(a[j]==x) // location j from array a[n]
   Success(); // x found at location j. algorithm
              // terminates successfully
}
Farlure(); // element x not found in array a[]
           // even after n choice selectrons.
}          // Algorithm terminates unsuccessfully.
```

Nondeterministic algorithm for sorting

```
Algorithm nd_sort(a,b,n)
{
// a is given array
// b is array for auxiliary (additional) storage

for i=1 to n do
{
  j=select(a,n);
  b[i]=a[j];  // create array b[] by selecting
              // n elements from array a[] in
}             // increasing order

for i=1 to n do
{
 if(b[i]>b[i+1])  // b[i], and b[i+1] are not
                  // in increasing order
  Failure();
}
 Success();  // All the n elements are in
             // increasing order.
}
```

NP-HARD AND NP-COMPLETE CLASSES

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

Ex $P(n) = n^3 + 2n^2 + 4n + 10$
polynomial function of order 3
$T(n) = O(n^3)$.

Decision problem / Decision algorithm Any problem for which the answer is either yes or no [0 or 1] is called a decision problem.

An algorithm for a decision problem is termed as a decision algorithm

COMPLEXITY CLASSES

P, NP, NP-HARD, NP-COMPLETE

P. class

P – Polynomial time
P is a set of all problems that can be solved by deterministic algorithms in polynomial time

Ex Qurck sort $O(n^2)$

matrix multiplrcation problem $O(n^3)$.

NP class

NP- Nondeterminlstic Polynomial time

NP is the set of all problems that can be solved by nondeterminlstic algorithms in polynomial time

Ex Graph coloring problem $O(m^n)$

m-coloring problem

coloring n vertices of a graph with given m colors

Since determinlstic algorithms are just a special case of nondeterminlstic algorithms, by this we can conclude that

$$P \subseteq NP$$

P is a subset of NP.

commonly belieed relatronship between P and NP problems. (class of problems).

There are two classes of non-polynomial (or nondeterministic polynomial) time problems

1) NP-Hard problems
2) NP-Complete problems.

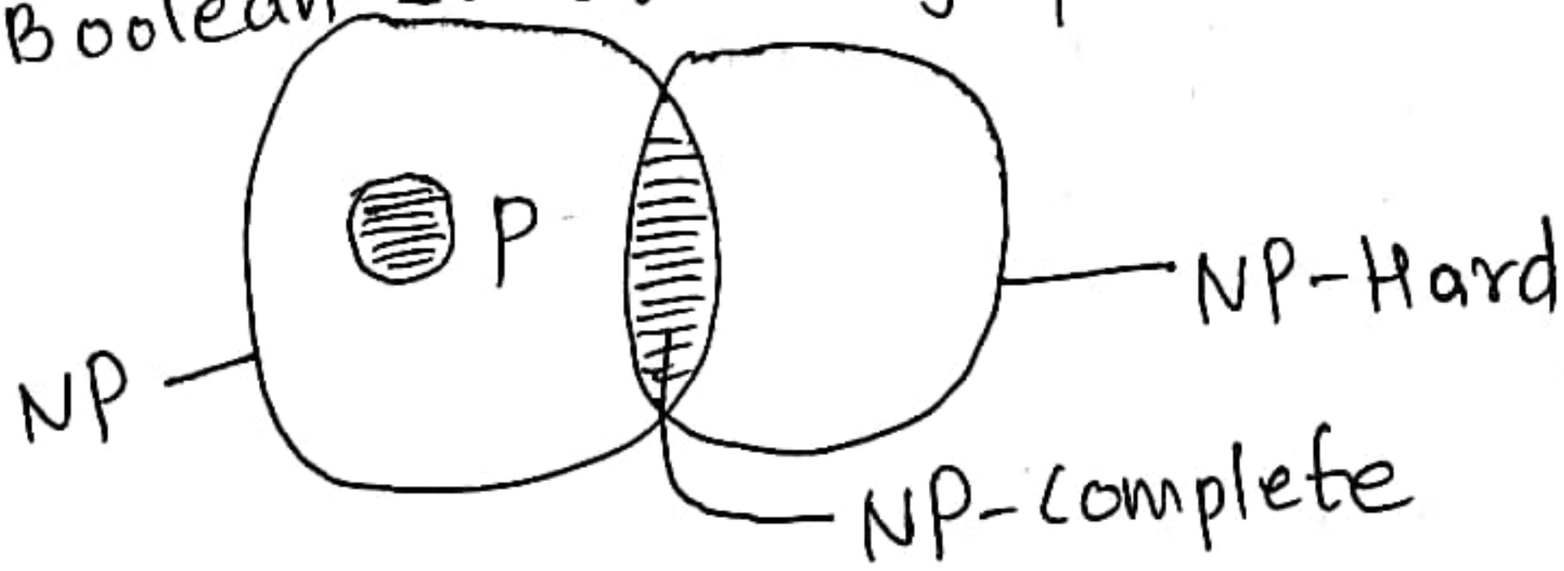
NP-Hard

Nondeterministic Polynomial time Hard.
Is a set/class of problems that are "atleast as hard as the hardest problems in NP class".

Ex Travelling salesperson Problem $O(2^n n^2) = O(2^n)$.

NP-Complete

Nondeterministic Polynomial time Complete
A problem L is NP-complete iff
L is NP-Hard and $L \in NP$
Is intersection of NP and NP-Hard
Ex Boolean satisfiability problem $O(2^n)$.



commonly believed relationship between P, NP, NP-Hard and NP-Complete classes.

Boolean Satisfiability (SAT) Problem

If there is atleast one assignment of truth values (true/false 0/1) to the boolean variables (literals) possible such that the boolean formula (evaluates) becomes or turns out to be TRUE, then we say that the boolean formula is satisfiable, otherwise unsatisfiable.

Ex1 $F = A \wedge \bar{B}$ is a satisfiable formula.

A	B	$\bar{B}$	$F = A \wedge \bar{B}$
F	F	T	F
F	T	F	F
T	F	T	T ✓
T	T	F	F

check for $2^2 = 4$ truth value combinations of A and B.

Ex2 $G = A \wedge \bar{A}$ is an unsatisfiable formula

A	$\bar{A}$	$G = A \wedge \bar{A}$
F	T	F
T	F	F

$2^1 = 2$ combinations.

SAT problem is NP-complete
Its time complexity is $O(2^n)$
where n is no. of variables in the boolean formula.

A	B	C	$F = A \wedge B \wedge C$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	F
T	F	F	F
T	F	T	F
T	T	F	F
T	T	T	T ✓

$2^3 = 8$ combinations

we need to check the truth value of boolean formula F for 2^n truth value combinations of n variables.

DNF Disjunctive Normal Form

Ex $F = (A \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge B \wedge \bar{C})$

CNF Conjunctive Normal Form

Ex $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2)$

3CNF

A 3CNF formula is a boolean formula in CNF with the added restriction that each clause has exactly three literals (variables)

Ex Following is a 3CNF formula (expression)

$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

3-SAT problem is a restriction of SAT problem where each clause is required to have exactly 3 literals (variables).

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

x_1	x_2	x_3	F
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

$2^3 = 8$ combinations

Nondeterministic algorithm for SAT problem

```
Algorithm Eval(E,n)
// Assign truth values to each literal x_i
{
for i=1 to n do // guessing
{
    x_i ← choice select (true, false);
}
// checking. check whether boolean formula
// or Expression E() evaluates to TRUE for
// the values of x_1, x_2, ... x_n
if E(x_1, x_2, ..., x_n) is true then
    Success(); // E() is satisfiable formula
else
    Failure(); // E() is unsatisfiable formula.
}
```

E- Boolean formula or Expression
Propositional formula.

COOK'S THEOREM *

state and prove cook's theorem

cook's theorem states that

satisfiability is in P if and only if P=NP

Proof: We have to prove

i) If P=NP, then satisfiability (~~O(2^n)~~) is in P

ii) If satisfiability is in P, then P=NP

i) It is known [from the previous topic] that satisfiability ($O(2^n)$) is in NP & if P=NP ⇒ satisfiability is in P

i) is proved

ii) we have to prove vice versa

i.e if satisfiability is in P, then P=NP

To prove this, a boolean formula Q(A,I) has to be formulated from polynomial time nondeterministic algorithm A and input I.

Q is satisfiable iff algorithm A has successful termination with input I.

When the length of input 'I' is n and the time complexity of algorithm 'A' is P(n)

Then the length of Q is $O(P^3(n)\log n) = O(P^4(n))$.

- A deterministic algorithm 'D' computes Q and determines whether Q is satisfiable (its value is TRUE or not).

If $O(q(m))$ is the time required to determine whether a formula of length 'm' is satisfiable then the time complexity of deterministic algorith 'D' is $O(p^3(n)\log n + q(p^3(n)\log n))$.

If satisfiability is in P then the time complexity of deterministic algorithm 'D' 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 algorithm D in P.

Thus, if satisfiability is in P, then P=NP.

ii) is proved

Hence the proof.

* Differentiate Greedy method and Dynamic programming (DP)

Greedy method	Dynamic programming(DP)
1) A single sequence of decisions is generated at a time	1) Various nos of decision sequences are generated at a time
2) we make our decision based on the best current situation	2) we make our decision based on the solution of previous step. The solution may relay on the solution of subproblems
3) Greedy algorithms make locally optimal choices at each step with the hope of finding a global optimum	3) Dynamic programming is an optimization algorithm (technique) that breaks down a complex problem into simpler subproblems and stores the solutions to these subproblems in a table to avoid redundant (repeated) computations.
4) They are simple and easy to implement, as the follow a step-by-step approach	4) It is more complex than the greedy method, as it requires additional memory and computational resources to store and retrieve solutions of

	subproblems from the table
5) The greedy method never alters the earlier choices, thus making it more efficient interms of memory	5) This technique prefers memorization due to which the memory complexity increases, making it less efficient.
6) Greedy techniques are faster than DP	6) DP is comparatively slower
7) Locally optimal choices are made at each step	7) Uses already produced solutions of the previous steps.
8) In this, there is no assurance of obtaining the optimal solution, as they may get stuck in a local minimum	8) In DP technique, you can get the assurance of an optimal solution.
9) Follows serial forward approach	9) Follows bottom-up or top-down approach.
10) An optimal solution may not be achieved	10) The optimal solution is achieved every time
11) Example problems - fractional knapsack - Job sequencing with deadlines problem - minimum-cost spanning trees - single source shortest paths problem	11) Applications - 0/1 knapsack problem - OBST problem - TSP problem - Reliability design problem - All pairs shortest paths problem.