DESIGN & ANALYSIS OF ALGORITHMS

UNIT – VI

Branch and Bound

- **Backtracking is effective for subset or permutation problems.**
- **Backtracking is not good for optimization problems. This drawback is rectified by using Branch And Bound technique.**
- **Branch And Bound is applicable for only optimization problems. Branch and Bound also uses bounding function similar to backtracking.**

5.3. 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.
- Both of BFS and D-search generalize to Branch and Bound strategies.
- In Branch and Bound terminology, a BFS-like space search will be called (First in First Out) search as the list of live nodes is a First In First Out List (or Queue)
- A D-Search like state space search will be called LIFO(Last In First Out) search as the list of live nodes is a Last-In- First-Out (or Stack)
- Three common search strategies are FIFO, LIFO and LC.
- The cost function $\hat{c}(.)$ such that $\hat{c}(x) \leq C(x)$ is used to provide lower bounds on solutions obtainable from any node x.
- If u is an upper bound on the cost of minimum cost solution then all live nodes x with $\hat{c}(x) > u$ may be killed as all answer nodes reachable from x have cost $C(x) \geq \hat{c}(x) > u$.
- In case an answer node with cost u has already been reached then all live nodes with $\hat{c}(x) \geq u$ may be killed.

We will use three types of search strategies or Nodes will be expanded in three ways.

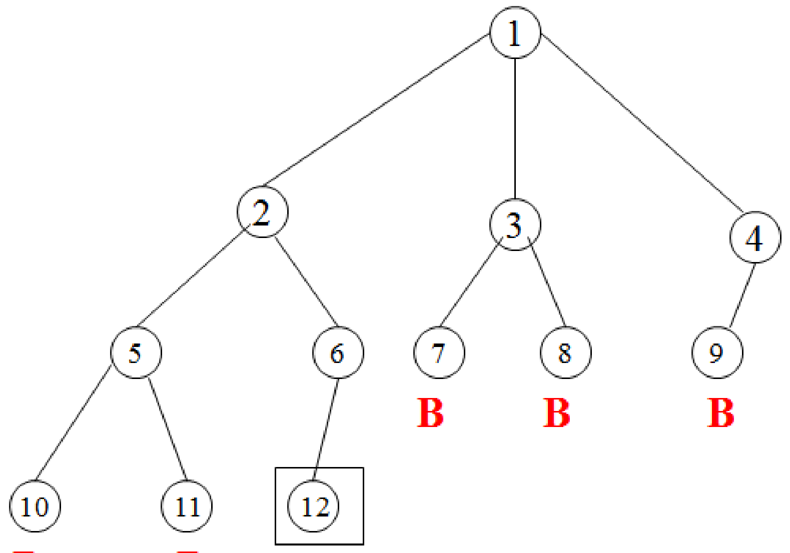
1. FIFO Branch and Bound Search – Queue

2. LIFO Branch and Bound Search or DFS – Stack

Least-Cost (or max priority) Branch and Bound Search — Priority Queue.

Least-cost branch and bound directs the search to the parts which most likely to contain the answer.

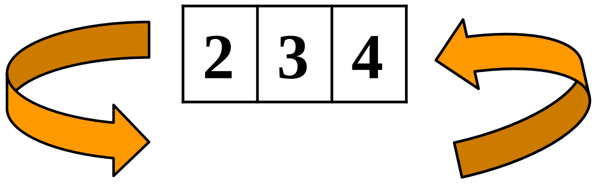
1. FIFO Branch And Bound Algorithm



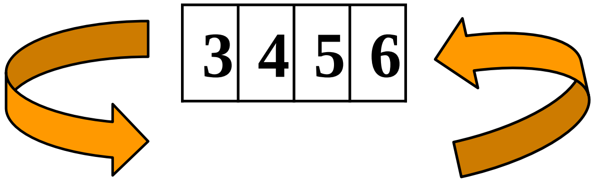
State Space Tree

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

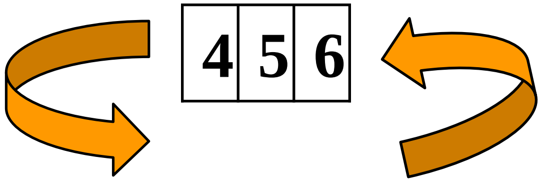
In FIFO search, first we will take E-node as 1. next, we generate the children of node 1. we will place all these live nodes in a Queue.



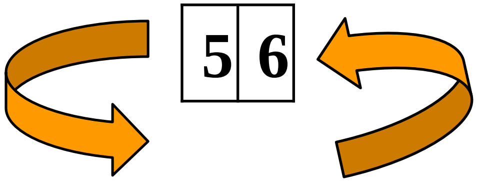
Now we delete an element from Queue, that is node 2. Next generate children of node 2 and place in the Queue



Next delete an element from queue and take it as E-node, generate the children of node 3. 7 and 8 are children of 3 and these live nodes are killed by bounding function. So we will not include in the queue.



Again delete an element from queue. Take as E-node. Generate the children of 4. node 9 is generated and killed by bounding function.



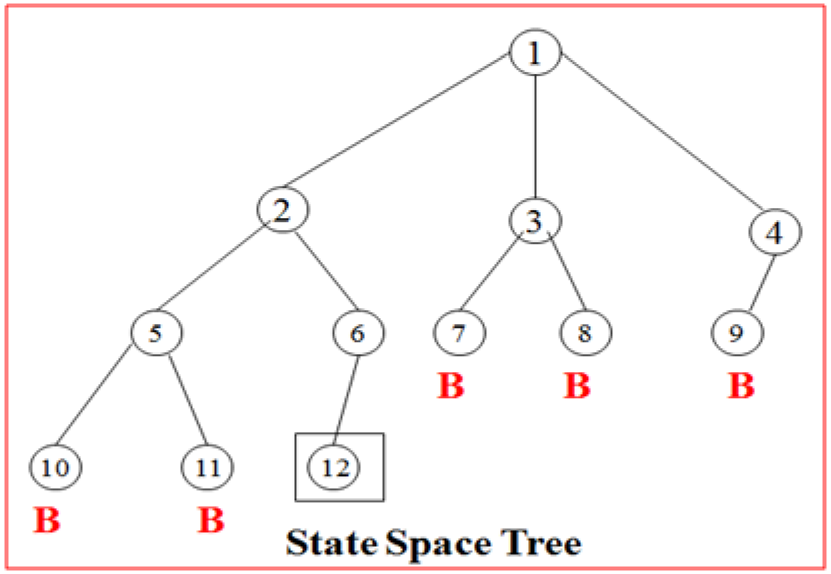
Next delete an element from queue. Generate children of node 5. That is nodes 10 and 11 are generated and killed by bounding function.

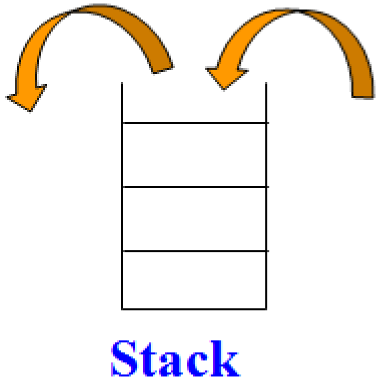
Last node in queue is 6. the child of node 6 is 12 and it satisfies the conditions of the problem. Which is the answer node. So search terminates.

2. LIFO Branch-and-bound:

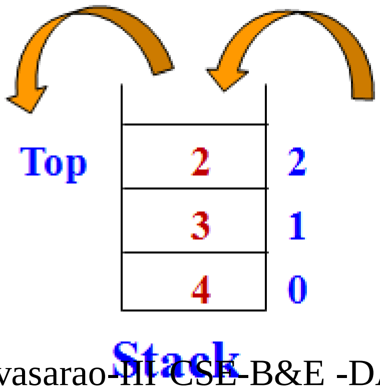
For this we will use a data structure called Stack.

Initially stack is empty.

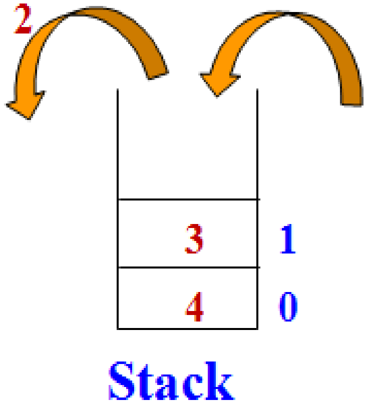




Generate children of node 1 and place these live nodes into stack.

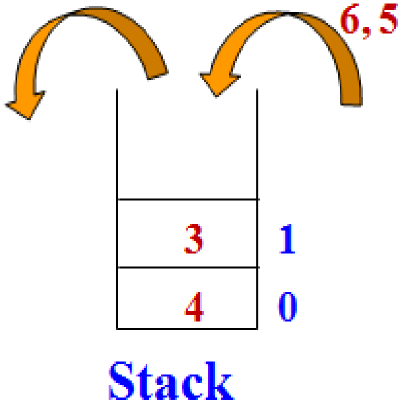


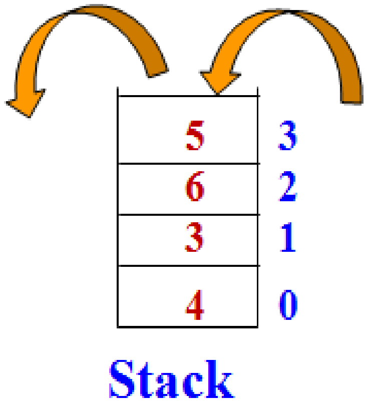
Remove from the stack and generate the children of it. Place these nodes in to stack.



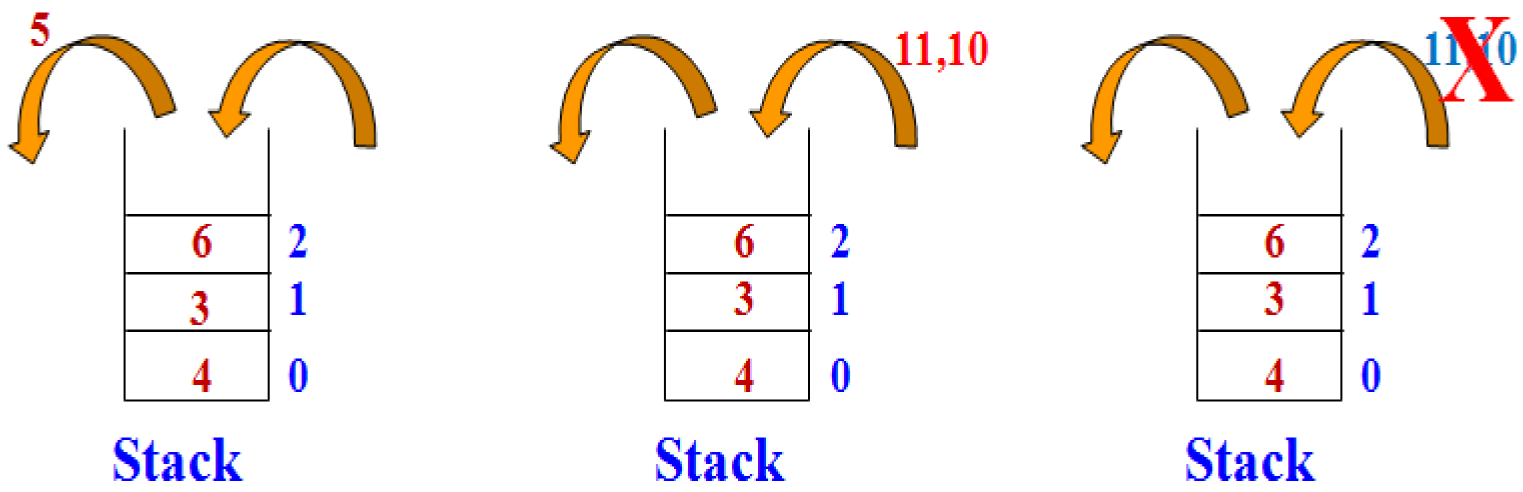
2 is removed from the stack. The children of 2 are 5 and 6.

Now the contents of stack are

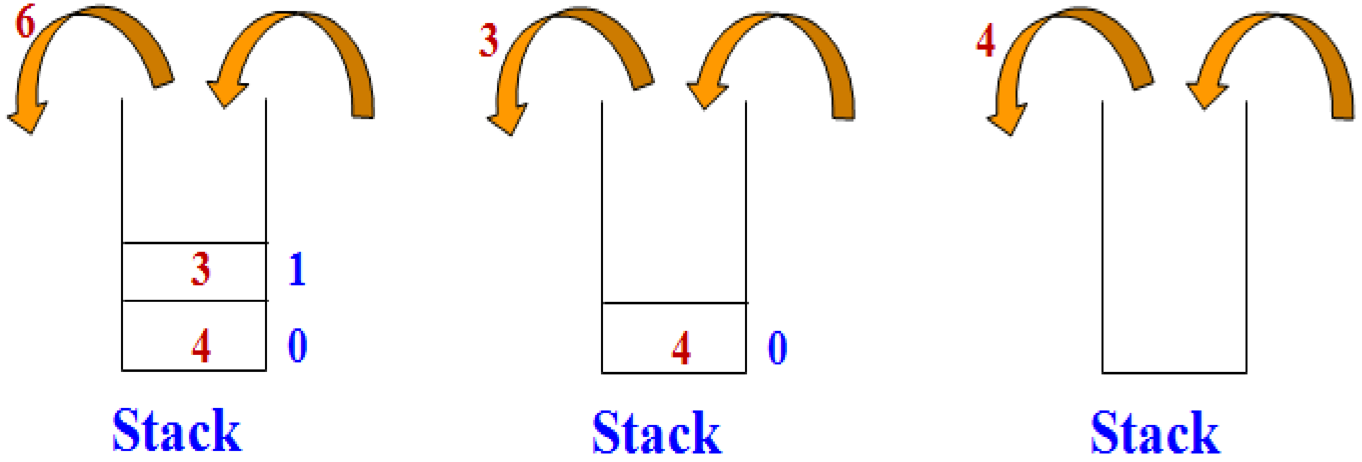




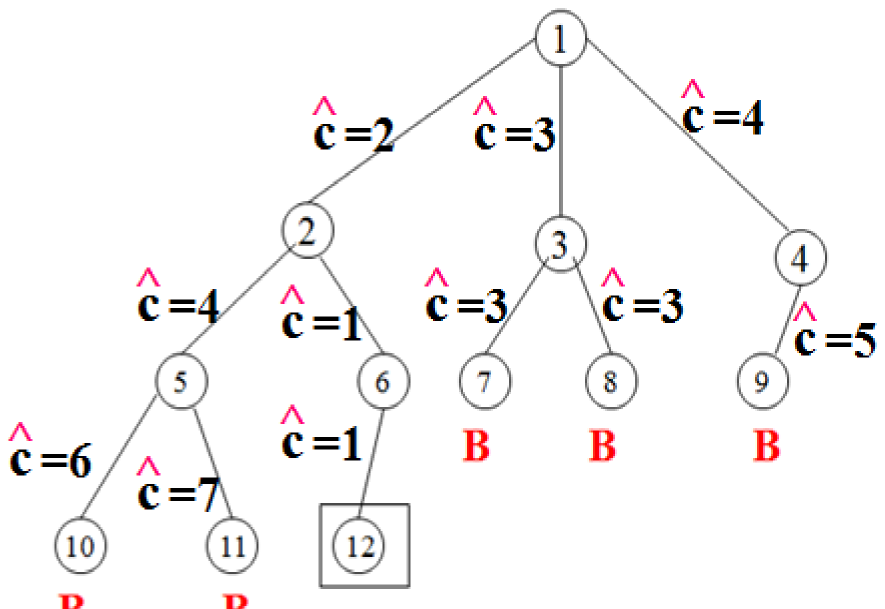
Again remove an element from stack, that is, node 5 is removed and nodes generated by 5 are 10,11 which are killed by bounding function. So, we will not place 10,11 into stack.



Generate node 6, that is 12. it is the answer node. So, search process terminates.



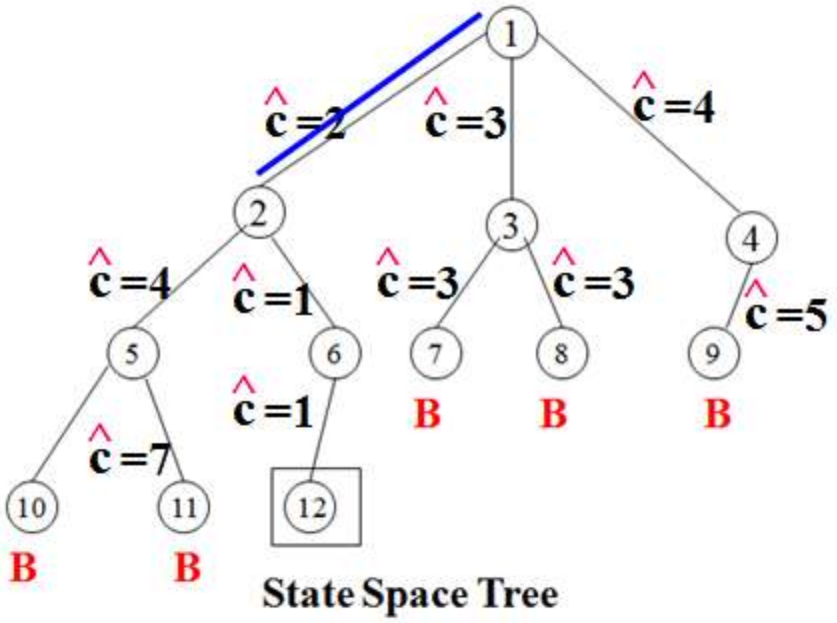
3. Least Cost branch and Bound (LCBB)

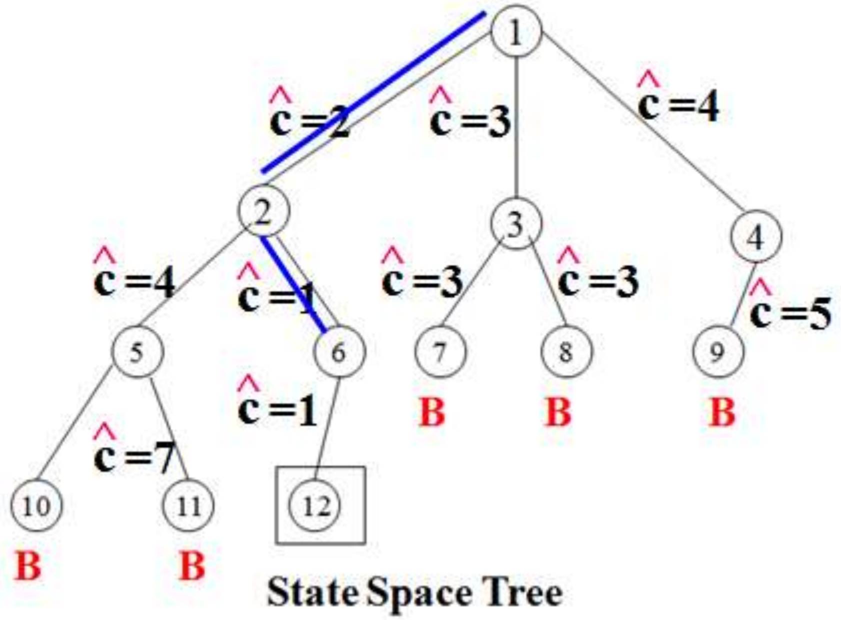


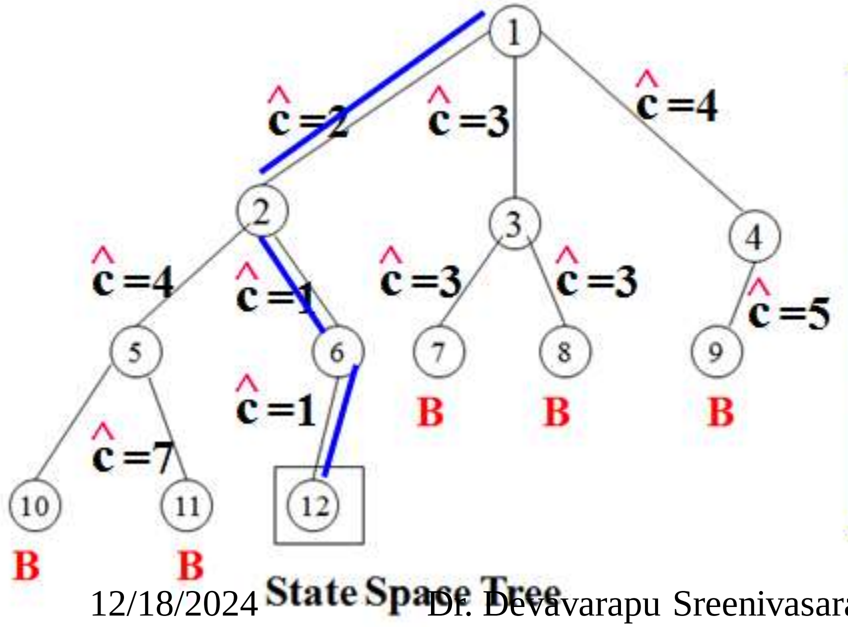
State Space Tree

Generally, children of node 1 are 2,3,4. by using Ranking Function we will calculate the cost of 2,3 and 4 nodes is 2,3 and 4 respectively.

Now we will select a node which has least cost, that is node 2. for node 2, the children are 5 and 6. the least cost among 5 and 6 are 1, that is node 6.







Now we will generate the children of node 6., that is 12. we will select least cost 12 node. So, 1 is the least cost and the node is 12. moreover the node 12 is the answer node. So, terminate the search process.

5.3.1. Branch – and - Bound Control Abstraction:

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

C(t) is the cost of minimum cost answer node in t. usually it is not possible to find an easily computable function c(.). An estimate $\hat{c}$ of c() is used. It is easy to compute $\hat{c}$ has the property that if x either an answer node or leaf node then $c(x) = \hat{c}(x)$.

A LCBB search of tree uses two functions least() and add(). Begin with an upper bound with and node 1 as the first node. When node 1 is expanded, node 2,3,4,…. are generated in the order. Add the live node to the list of live nodes. If the function least() finds least cost node and the node is deleted from the list of live nodes and returned. The output of least cost search is tracing the path from the answer node to root node of the tree.

```
Algorithm LC_Search (t)    // search tree t for an answer node
{
    if *t is an answer node then output *t and return;
     E=t;     // E-node
    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 is a new live node
                   (x->parent):=E;    // pointer for path to root node
             }
              if (there are no more live nodes) then
             {
                  write("No answer node");
                  return;
             }
             E=Least();
      } until (false);
}
```

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

(x ->parent)=E is to print answer path.

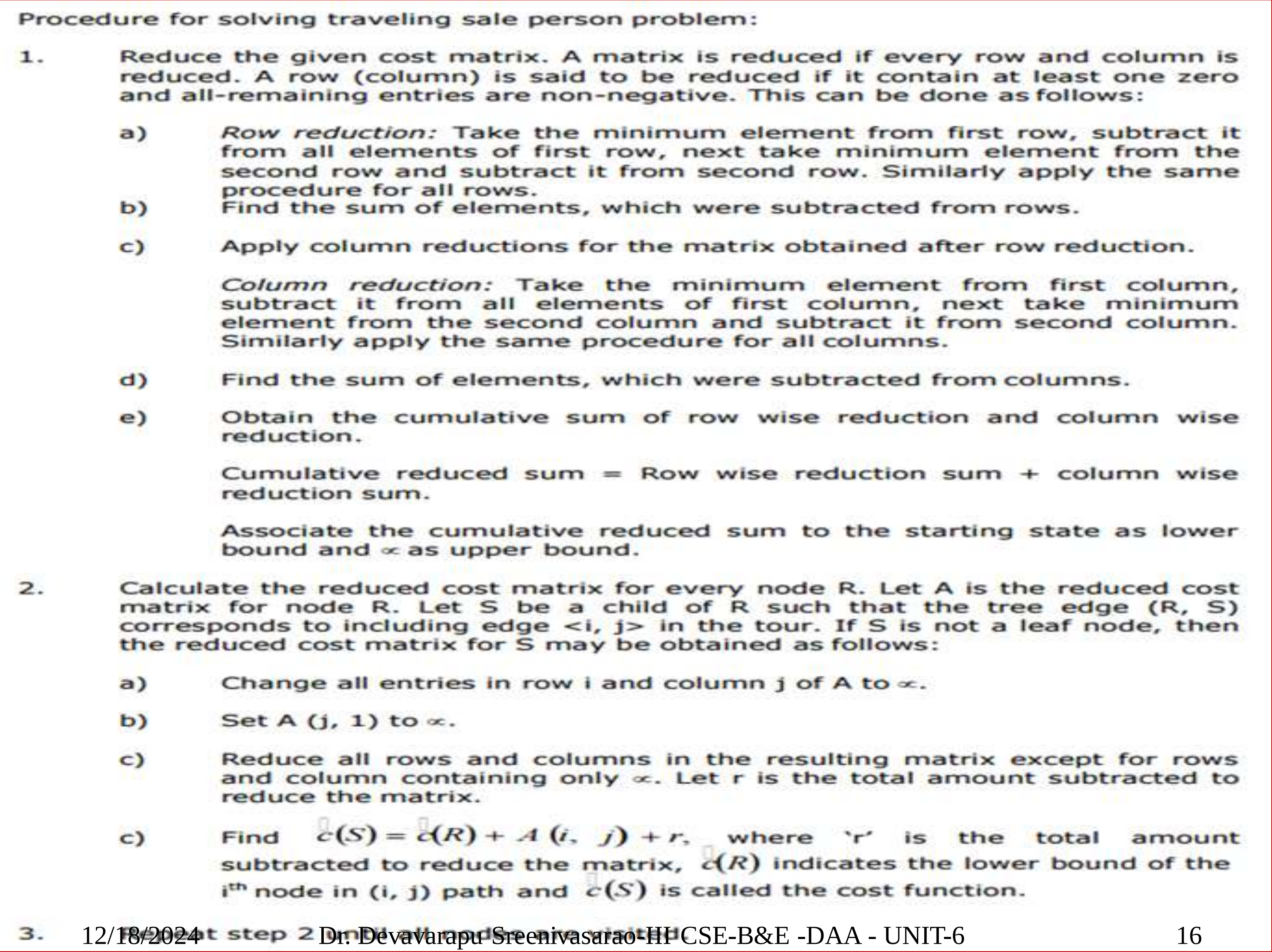
1. It is desirable to find an answer node that has minimum cost among all answer nodes. But LC search can not guarantee to find an answer node G with minimum cost c(G).
2. When there exists two nodes in a graph such that c(x)>c(y) in one branch while c(x) < c(y) in other branch, LC search can not find minimum cost answer node.
3. Even if c(x) < c(y) for every pair of nodes x, y such that c(x)<c(y), LC may not find a minimum cost answer node.
4. When the estimate $\hat{c}()$ for a node is less than the cost c() then a slight modification to the LC search results in search algorithm that terminates when a minimum cost answer node is reached. In this modification, the search continues until an answer node becomes E-node.
5. At the time an E-node is an answer node $\hat{c}(E) \leq \hat{c}(L)$ for every node on a graph L in the list of live nodes. Hence $\hat{c}(E) \leq \hat{c}(L)$ and so E is minimum cost answer node.

5. 4. The Branch and Bound Applications are

- 5.4.1. Travelling Sales Person Problem
- 5.4.2. 0/1 Knapsack Problem-LCBB.
- 5.4.3. 0/1 Knapsack Problem-FIFOBB.

5.4.1. Travelling Sales Person Problem

Problem Statement : If there are n cities and cost of traveling from any city to any other city is given. Then we have to obtain the shortest round-trip such that each city is visited exactly once and then returning to starting city, completes the tour.



Procedure for solving traveling sale person problem:

1. Reduce the given cost matrix. A matrix is reduced if every row and column is reduced. A row (column) is said to be reduced if it contain at least one zero and all-remaining entries are non-negative. This can be done as follows:

   a) *Row reduction:* Take the minimum element from first row, subtract it from all elements of first row, next take minimum element from the second row and subtract it from second row. Similarly apply the same procedure for all rows.

   b) Find the sum of elements, which were subtracted from rows.

   c) Apply column reductions for the matrix obtained after row reduction.

      Column reduction: Take the minimum element from first column, subtract it from all elements of first column, next take minimum element from the second column and subtract it from second column. Similarly apply the same procedure for all columns.

   d) Find the sum of elements, which were subtracted from columns.

   e) Obtain the cumulative sum of row wise reduction and column wise reduction.

      Cumulative reduced sum = Row wise reduction sum + column wise reduction sum.

      Associate the cumulative reduced sum to the starting state as lower bound and ∝ as upper bound.

2. Calculate the reduced cost matrix for every node R. Let A is 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 <i, j> in the tour. If S is not a leaf node, then the reduced cost matrix for S may be obtained as follows:

   a) Change all entries in row i and column j of A to ∝.

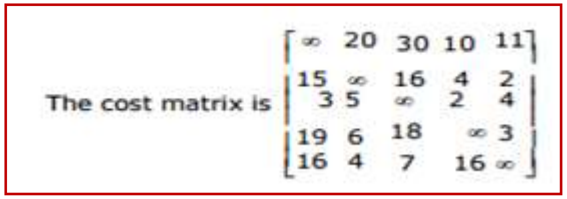
   b) Set A (j, 1) to ∝.

   c) Reduce all rows and columns in the resulting matrix except for rows and column containing only ∝. Let r is the total amount subtracted to reduce the matrix.

   c) Find $\hat{c}(S) = \hat{c}(R) + A(i, j) + r$, where 'r' is the total amount subtracted to reduce the matrix, $\hat{c}(R)$ indicates the lower bound of the ith node in (i, j) path and $\hat{c}(S)$ is called the cost function.

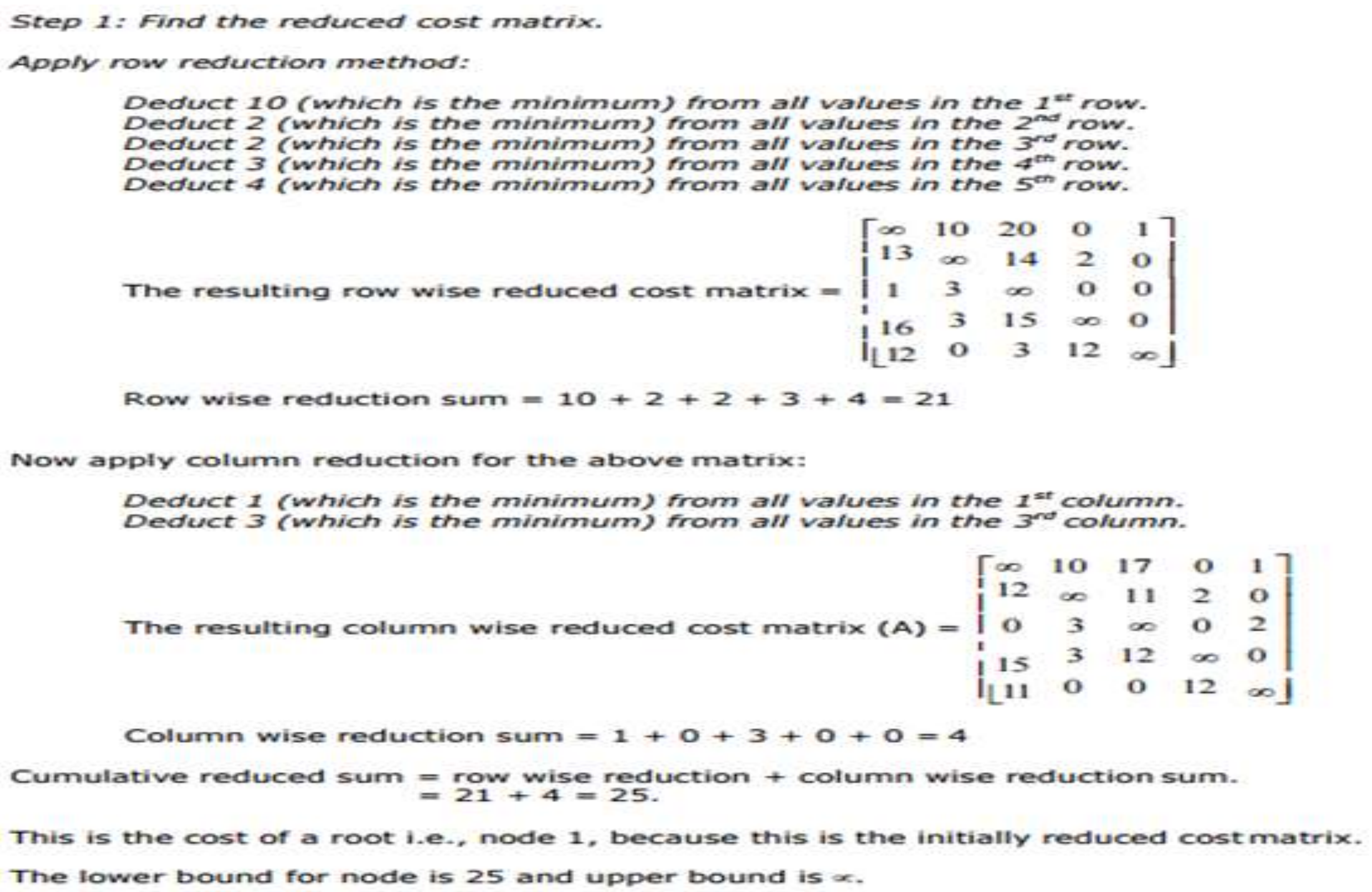
3. Repeat step 2 until all nodes are visited.

Solve the following instance of Travelling Sales Person Problem using LCBB.



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

Solution:



Step 1: Find the reduced cost matrix.

Apply row reduction method:

Deduct 10 (which is the minimum) from all values in the 1st row.
Deduct 2 (which is the minimum) from all values in the 2nd row.
Deduct 2 (which is the minimum) from all values in the 3rd row.
Deduct 3 (which is the minimum) from all values in the 4th row.
Deduct 4 (which is the minimum) from all values in the 5th row.

The resulting row wise reduced cost matrix = $\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 0 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$

Row wise reduction sum = 10 + 2 + 2 + 3 + 4 = 21

Now apply column reduction for the above matrix:

Deduct 1 (which is the minimum) from all values in the 1st column.
Deduct 3 (which is the minimum) from all values in the 3rd column.

The resulting column wise reduced cost matrix (A) = $\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$

Column wise reduction sum = 1 + 0 + 3 + 0 + 0 = 4

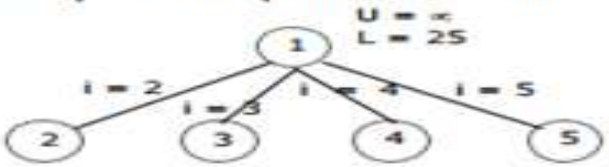
Cumulative reduced sum = row wise reduction + column wise reduction sum.
= 21 + 4 = 25.

This is the cost of a root i.e., node 1, because this is the initially reduced cost matrix.

The lower bound for node is 25 and upper bound is ∞.

Starting from node 1, we can next visit 2, 3, 4 and 5 vertices. So, consider to explore the paths (1, 2), (1, 3), (1, 4) and (1, 5).

The tree organization up to this point is as follows:



Variable 'i' indicates the next node to visit.

Step 2:

Consider the path (1, 2):

Change all entries of row 1 and column 2 of A to ∝ and also set A(2, 1) to ∝.

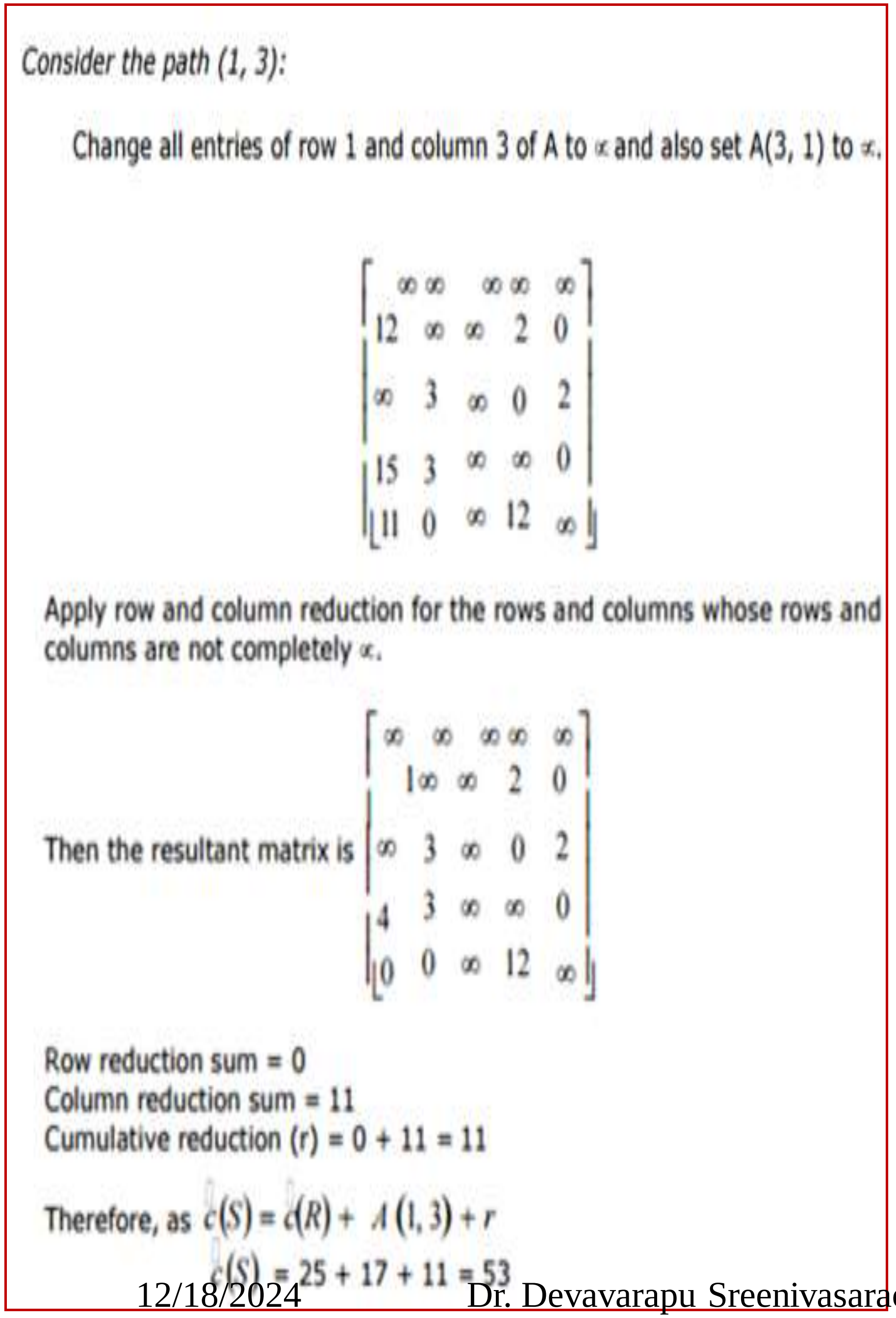
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$

Row reduction sum = 0 + 0 + 0 + 0 = 0
Column reduction sum = 0 + 0 + 0 + 0 = 0
Cumulative reduction (r) = 0 + 0 = 0

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


Consider the path (1, 3):

Change all entries of row 1 and column 3 of A to ∝ and also set A(3, 1) to ∝.

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

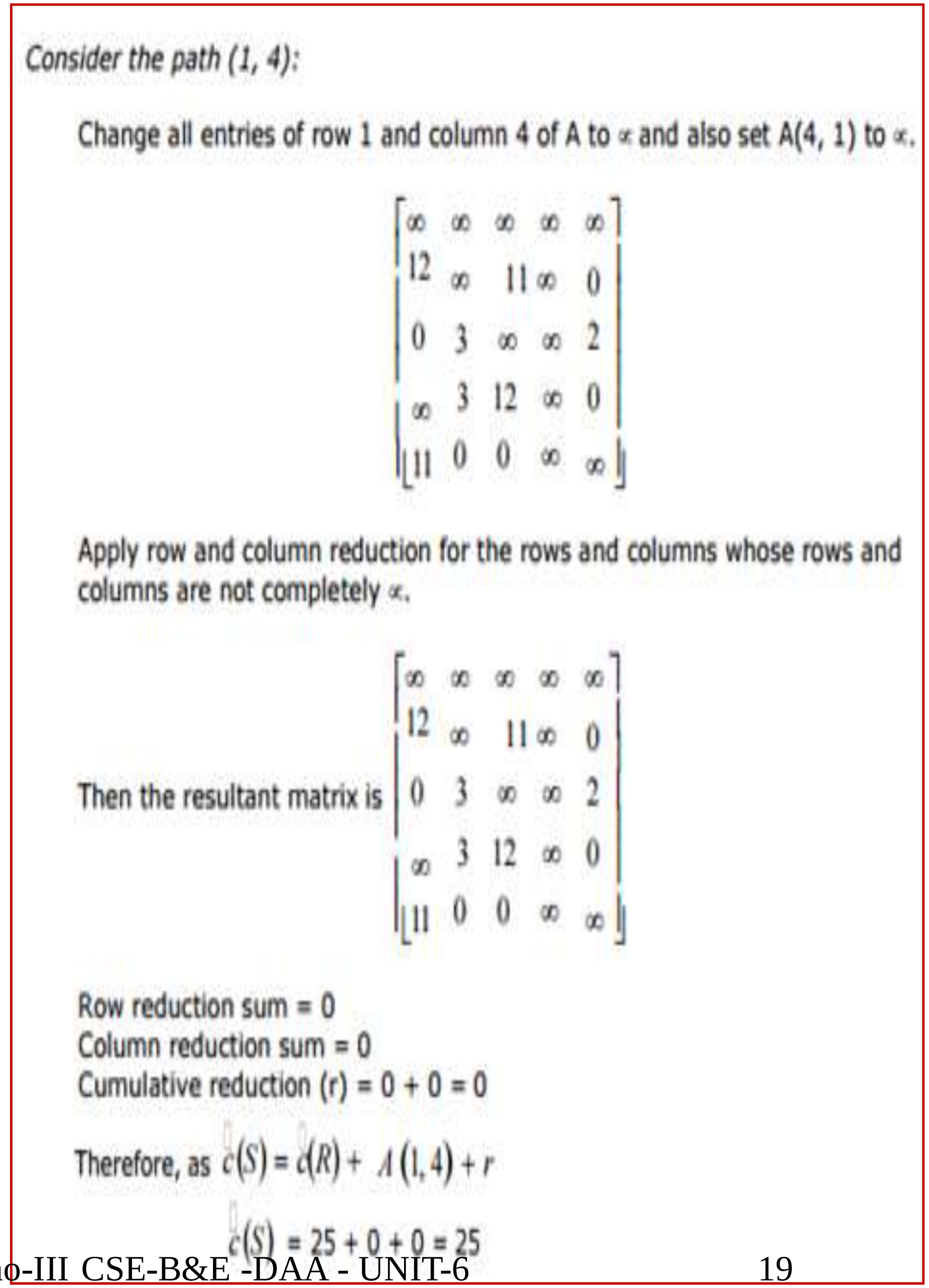
Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$

Row reduction sum = 0
Column reduction sum = 11
Cumulative reduction (r) = 0 + 11 = 11

Therefore, as $c(S) = c(R) + A(1, 3) + r$

$c(S) = 25 + 17 + 11 = 53$


Consider the path (1, 4):

Change all entries of row 1 and column 4 of A to ∝ and also set A(4, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$

Row reduction sum = 0
Column reduction sum = 0
Cumulative reduction (r) = 0 + 0 = 0

Therefore, as $c(S) = c(R) + A(1, 4) + r$

$c(S) = 25 + 0 + 0 = 25$

Consider the path (1, 5):

Change all entries of row 1 and column 5 of A to ∝ and also set A(5, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

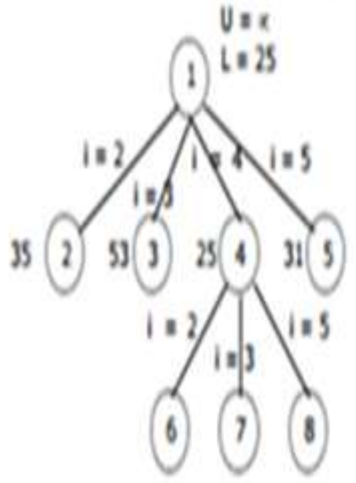
Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$

Row reduction sum = 5
Column reduction sum = 0
Cumulative reduction (r) = 5 + 0 = 0

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(1,5) + r$

$\hat{c}(S) = 25 + 1 + 5 = 31$

The tree organization up to this point is as follows:



The cost of the paths between (1, 2) = 35, (1, 3) = 53, (1, 4) = 25 and (1, 5) = 31. The cost of the path between (1, 4) is minimum. Hence the matrix obtained for path (1, 4) is considered as reduced cost matrix.

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

The new possible paths are (4, 2), (4, 3) and (4, 5).

Consider the path (4, 2):

Change all entries of row 4 and column 2 of A to ∝ and also set A(2, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$

Row reduction sum = 0
Column reduction sum = 0
Cumulative reduction (r) = 0 + 0 = 0

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(4,2) + r$

$\hat{c}(S) = 25 + 3 + 0 = 28$

Consider the path (4, 3):

Change all entries of row 4 and column 3 of A to ∝ and also set A(3, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$

Row reduction sum = 2
Column reduction sum = 11
Cumulative reduction (r) = 2 + 11 = 13

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(4,3) + r$

$\hat{c}(S) = 25 + 12 + 13 = 50$

Consider the path (4, 5):

Change all entries of row 4 and column 5 of A to ∝ and also set A(5, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

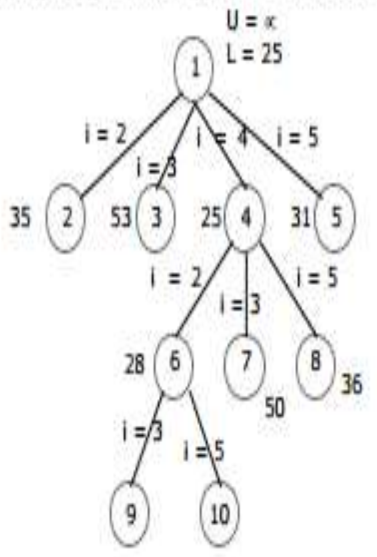
Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$

Row reduction sum = 11
Column reduction sum = 0
Cumulative reduction (r) = 11+0 = 11

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(4,5) + r$

$$\hat{c}(S) = 25 + 0 + 11 = 36$$

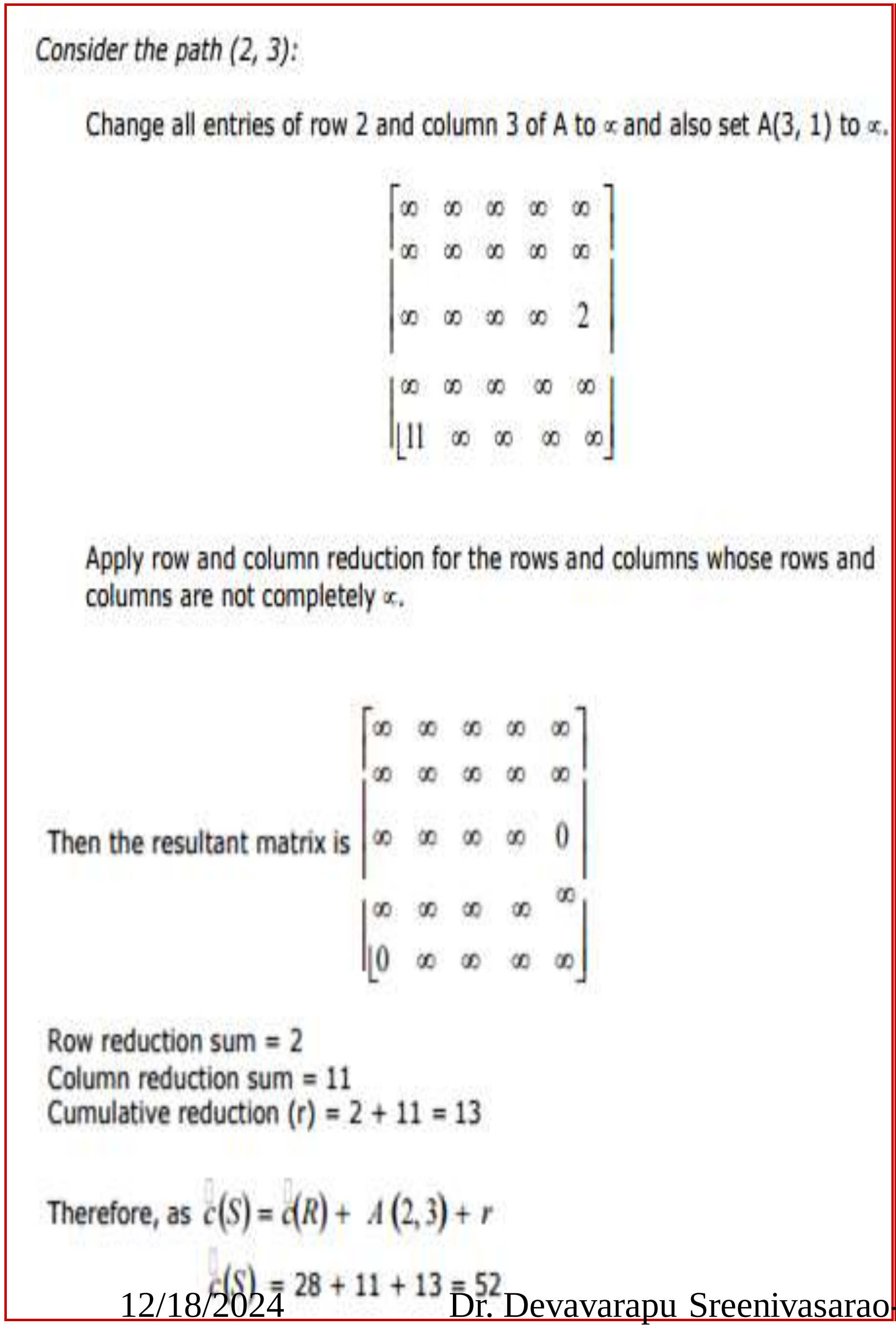
The tree organization up to this point is as follows:



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

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

The new possible paths are (2, 3) and (2, 5).



Consider the path (2, 3):

Change all entries of row 2 and column 3 of A to ∝ and also set A(3, 1) to ∝.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix}$$

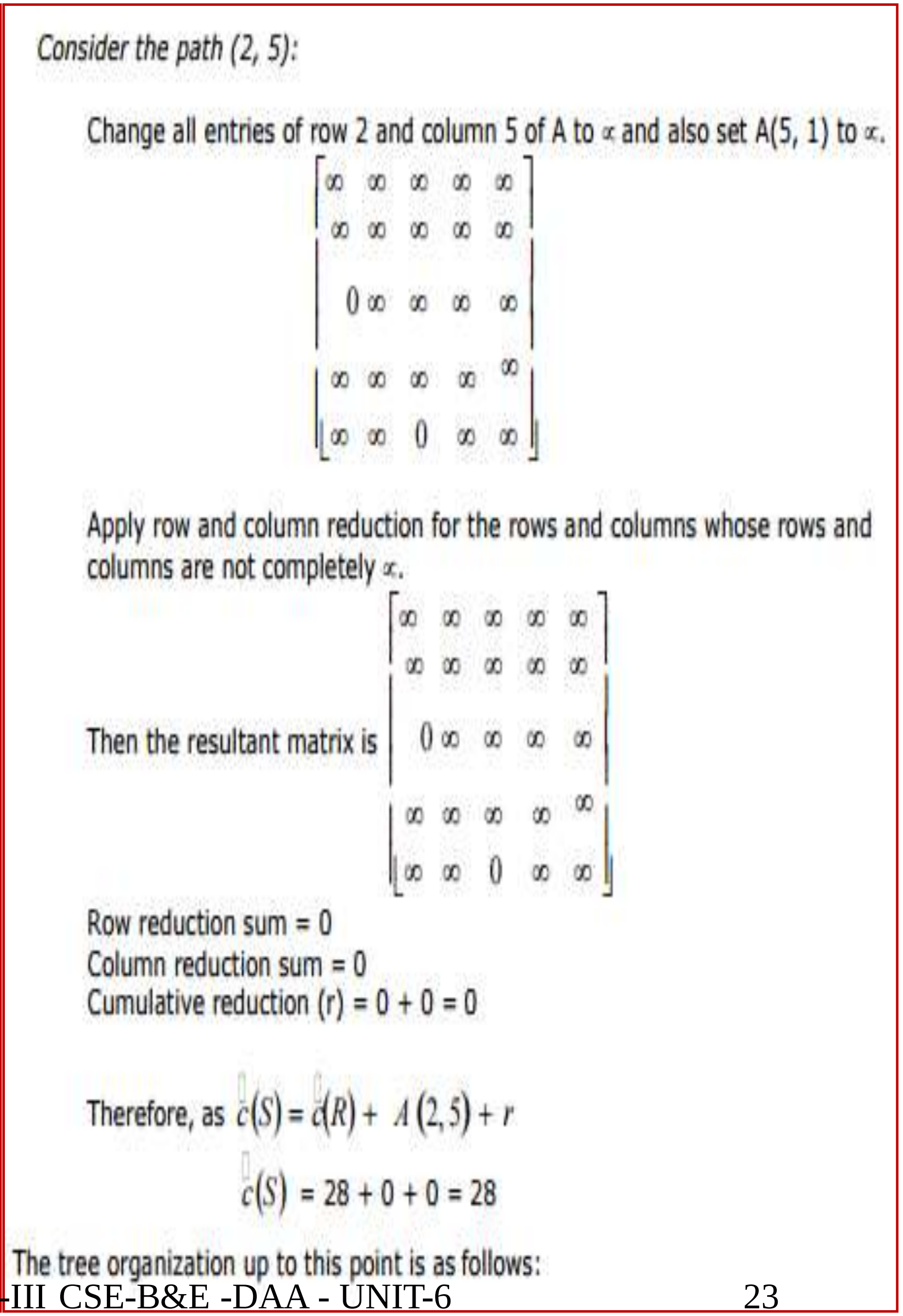
Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$

Row reduction sum = 2
Column reduction sum = 11
Cumulative reduction (r) = 2 + 11 = 13

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(2,3) + r$

$\hat{c}(S) = 28 + 11 + 13 = 52$



Consider the path (2, 5):

Change all entries of row 2 and column 5 of A to ∝ and also set A(5, 1) to ∝.

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

Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

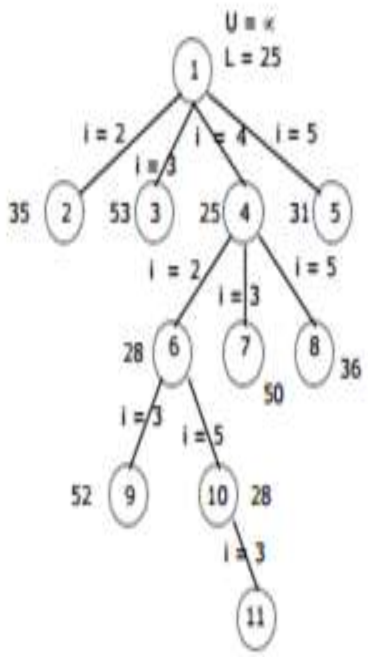
Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$

Row reduction sum = 0
Column reduction sum = 0
Cumulative reduction (r) = 0 + 0 = 0

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(2,5) + r$

$\hat{c}(S) = 28 + 0 + 0 = 28$

The tree organization up to this point is as follows:



The cost of the paths between (2, 3) = 52 and (2, 5) = 28. The cost of the path between (2, 5) is minimum. Hence the matrix obtained for path (2, 5) is considered as reduced cost matrix.

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

The new possible paths is (5, 3).

Consider the path (5, 3):

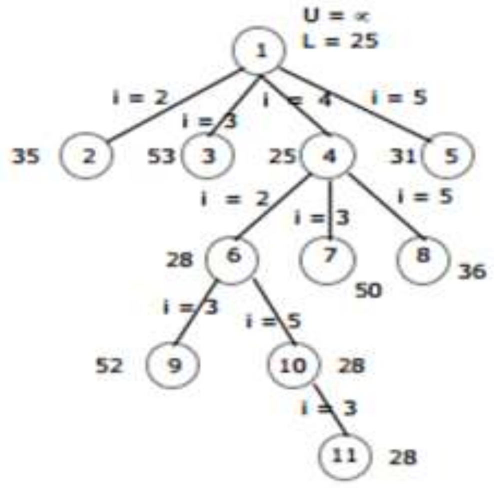
Change all entries of row 5 and column 3 of A to ∝ and also set A(3, 1) to ∝. Apply row and column reduction for the rows and columns whose rows and columns are not completely ∝.

Then the resultant matrix is $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$

Row reduction sum = 0
Column reduction sum = 0
Cumulative reduction (r) = 0 + 0 = 0

Therefore, as $\hat{c}(S) = \hat{c}(R) + A(5, 3) + r$
$\hat{c}(S) = 28 + 0 + 0 = 28$

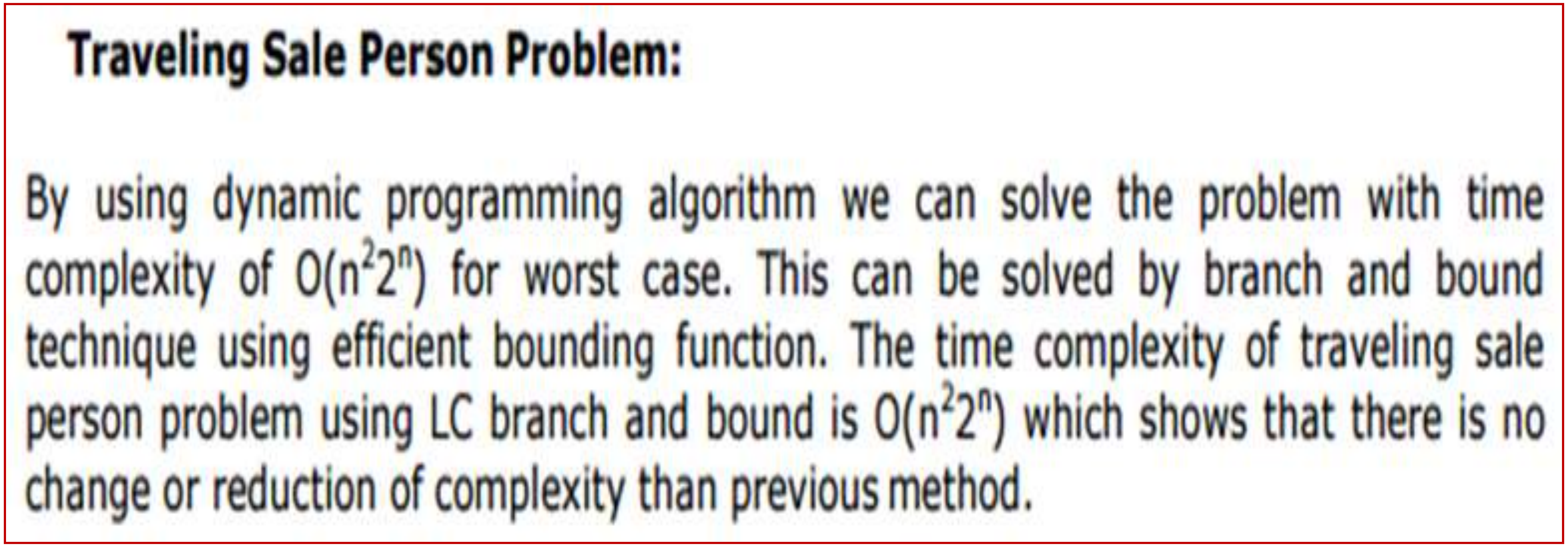
The overall tree organization is as follows:



The path of traveling sale person problem is:

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

The minimum cost of the path is: 10 + 6 +2+ 7 + 3 = 28.



Traveling Sale Person Problem:

By using dynamic programming algorithm we can solve the problem with time complexity of $O(n^2 2^n)$ for worst case. This can be solved by branch and bound technique using efficient bounding function. The time complexity of traveling sale person problem using LC branch and bound is $O(n^2 2^n)$ which shows that there is no change or reduction of complexity than previous method.

0/1 Knapsack Problem

- **Generally Branch and Bound will minimize the objective function.**
- **The 0/1 knapsack problem is a maximization problem.**
- **This difficulty can be avoided by replacing the objective function $\sum p_i x_i$ by $-\sum p_i x_i$.**

Note: All live nodes with $c(x) > upper$ can be killed when they are about to become E-nodes.

Steps to be followed in LCBB

Step 1:Compute $\hat{c}(.)$ and u(.) for a generated node, then if $\hat{c}$>upper(initially upper=u(root node)) immediately kill the node otherwise add the node to the list of live nodes.

Step 2: Update upper = u(.), if upper is greater than u(.).

Step 3: The Least Cost node will be the next E-node.

Step 4: Whenever a node about become an E-node, check if $\hat{c}(.)$ is greater than upper, if yes kill the node otherwise generate its children. Also kill the nodes which are not feasible.

Step 5: Continue step 1 to 4 till all the live nodes are expanded.

Step 6 : Finally the highest U value (absolute value) node will be the answer node. Trace the path in backward direction from answer node to root node for solution subset.

Example:- Draw the portion of state space tree generated by LCBB by the following Knapsack Problem. 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

Solution: First convert the profits to negative. $(p_1,p_2,p_3,p_4)=(-10,-10,-12,-18)$

The total profits must be maximization. So, it is maximization problem.

Calculate the lower bound and upper bound for each node. Lower bound and upper bound are calculated as sum of $P_i.X_i$. The sum is always <=M.

But in lower bound calculation the fractions are allowed. In upper bond calculations fractions are not allowed.

May be solution in the form of S ={x1,x3} or S= {1,0,1,0}

Place the first item in the bag. That is $w_1 = 2$

Remaining weight is 15 – 2 = 13.

Now, Place the 2nd item in the bag. That is $w_2 = 4$

Remaining weight is 13 – 4 = 09.

Now, Place the 3rd item in the bag. That is $w_3 = 6$

Remaining weight is 09 – 6 = 03.

Initially, the upper bound is ∞.

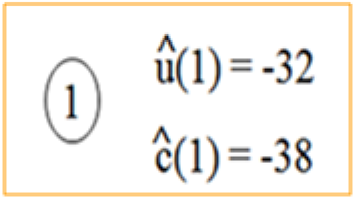
Now we are calculating the upper bound of node 1, fractions are not allowed.

$\hat{u}(1) = -10 + -10 + -12 = -32$

Therefore, $\infty > -32$, the upper bound is -32

To calculate the lower bound, place the 4th item in the bag. Since fractions are allowed.

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



For Node 2: (x_1 item included. That is x_1=1)

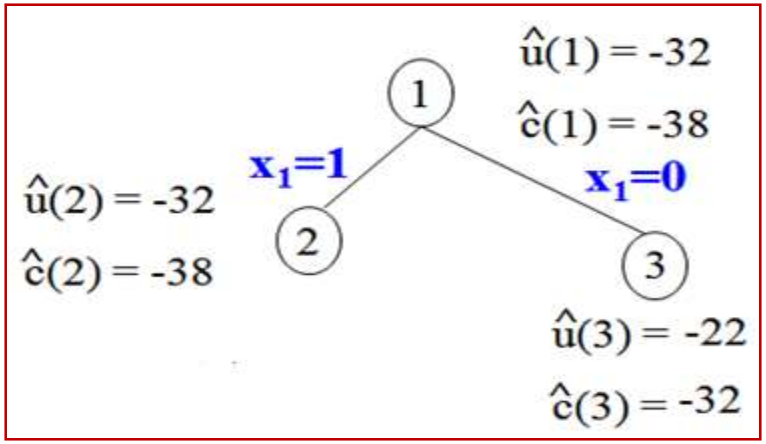
Lower bound (2) = $\hat{c}(2)$ = -10 -10-12-(3/9) X 18 = -38

Upper bound(2) = $\hat{u}(2)$ = -10 -10-12 = -32

For Node 3: (x_1 item not included. That is x_1=0)

Lower bound (3) = $\hat{c}(3)$ = -10-12-(5/9) X 18 = -32

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



The cost of node 3 is -22. -22 > - 38. so, node 3 is killed.

Select the minimum cost. That is

Min{$\hat{c}(2)$, $\hat{c}(3)$} = {-38, -32} = -38, that is c(2)

Therefore choose the node 2.

Therefore, select the first item, i.e x_1=1

Consider the 2nd variable to take the decision at 2nd level

For Node 4: (x_2 item included. That is x_2=1)

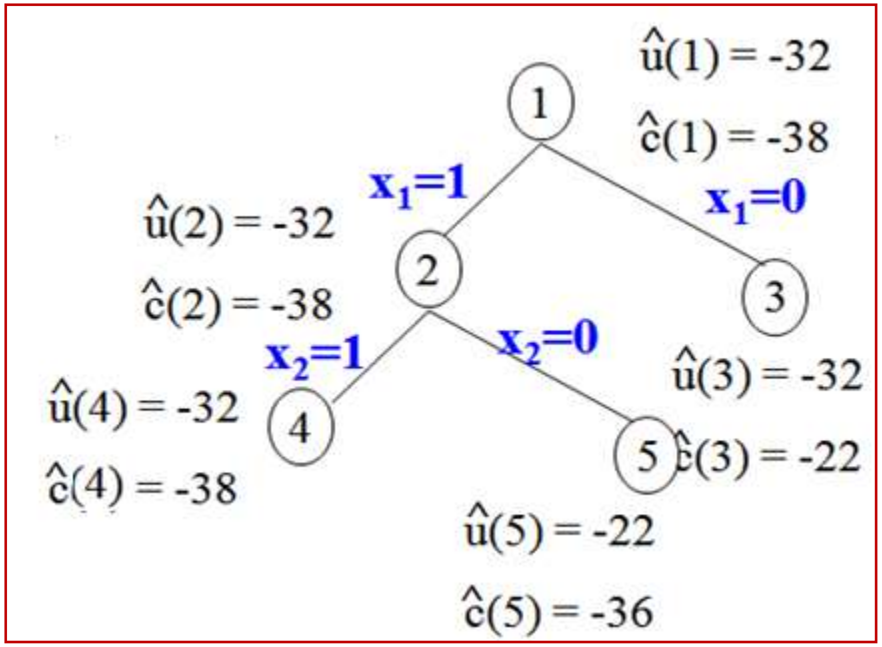
Lower bound (4) = $\hat{c}(4)$ = -10 -10-12-(3/9) X 18 = -38

Upper bound(4) = $\hat{u}(4)$ = -10 -10-12 = -32

For Node 5: (x_2 item not included. That is x_2=0)

Lower bound(5)=$\hat{c}(5)$=-10-12-(7/9)X18=-36

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



Select the minimum cost. That is

Min{$\hat{c}(4)$, $\hat{c}(5)$} = {-38, -36} = -38, that is c(4)

Therefore choose the node 4.

Therefore, select the second item, i.e x_2=1

The cost of node 5 is -36. -36 > - 38. so, node 5 is killed.

Consider the 3rd variable to take the decision at 3rd level

For Node 6: (x_3 item included. That is $x_3=1$)

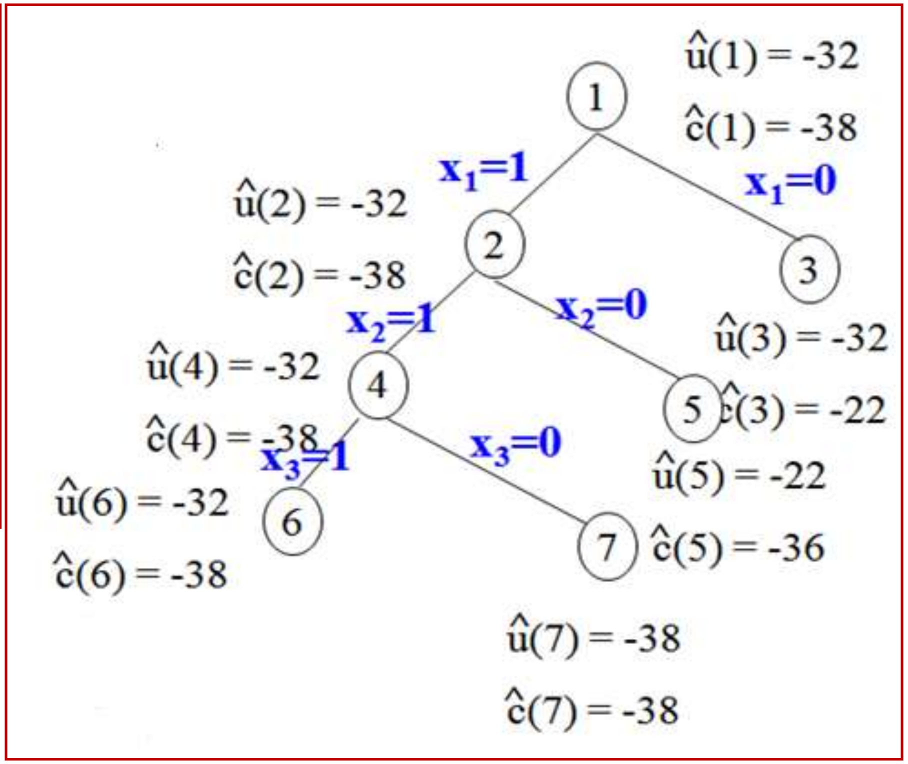
Lower bound (6) = $\hat{c}(6)$ = -10 -10-12-(3/9) X 18 = -38

Upper bound(6) = $\hat{u}(6)$ = -10 -10-12 = -32

For Node 7: (x_3 item not included. That is $x_3=0$)

Lower bound(7)=$\hat{c}(7)$=-10-10-18=-38

Upper bound(7) = $\hat{u}(7)$ = -10-10-18 = -38

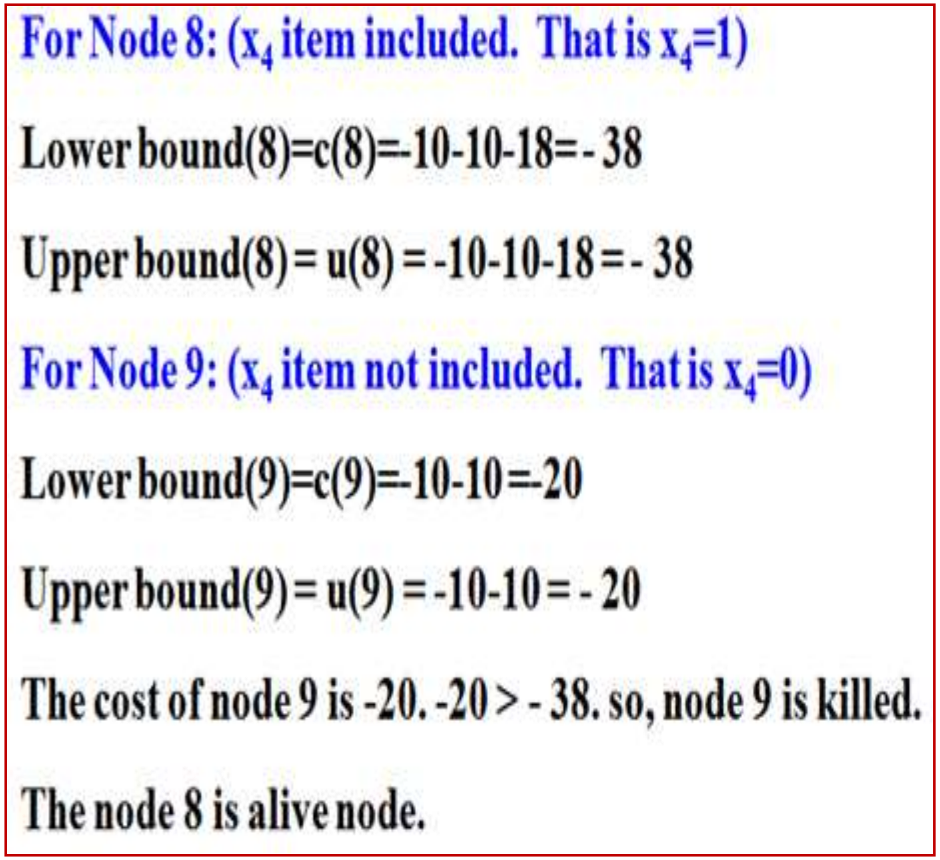


If we include first, second, third and fourth items. The weight is 2 + 4 + 6 + 9 = 21. so, 21 > 15. therefore, Node 6 not produce the solution. It is infeasible.

Therefore, expand the node 7.

Therefore, 3rd item is not selected i.e $x_3=0$

Consider the 4th variable to take the decision at 4th level



For Node 8: (x_4 item included. That is x_4=1)

Lower bound(8)=c(8)=-10-10-18=- 38

Upper bound(8) = u(8) = -10-10-18 = - 38

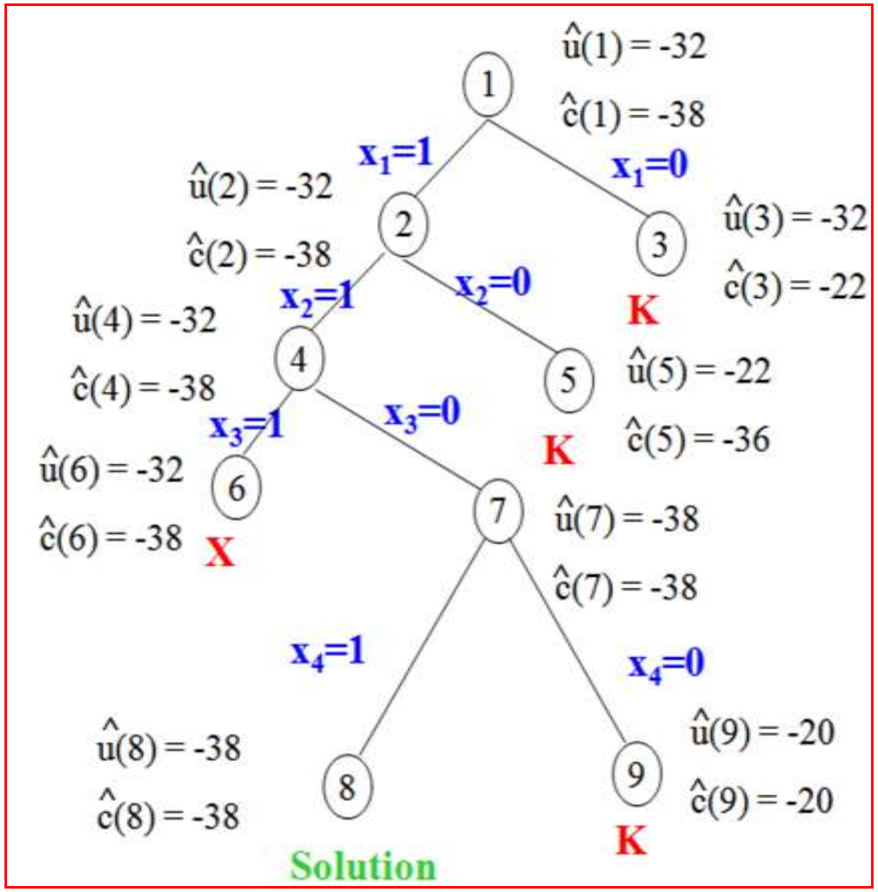
For Node 9: (x_4 item not included. That is x_4=0)

Lower bound(9)=c(9)=-10-10=-20

Upper bound(9) = u(9) = -10-10 = - 20

The cost of node 9 is -20. -20 > - 38. so, node 9 is killed.

The node 8 is alive node.



The solution is, The sub set form : S = {x_1, x_2, x_4}

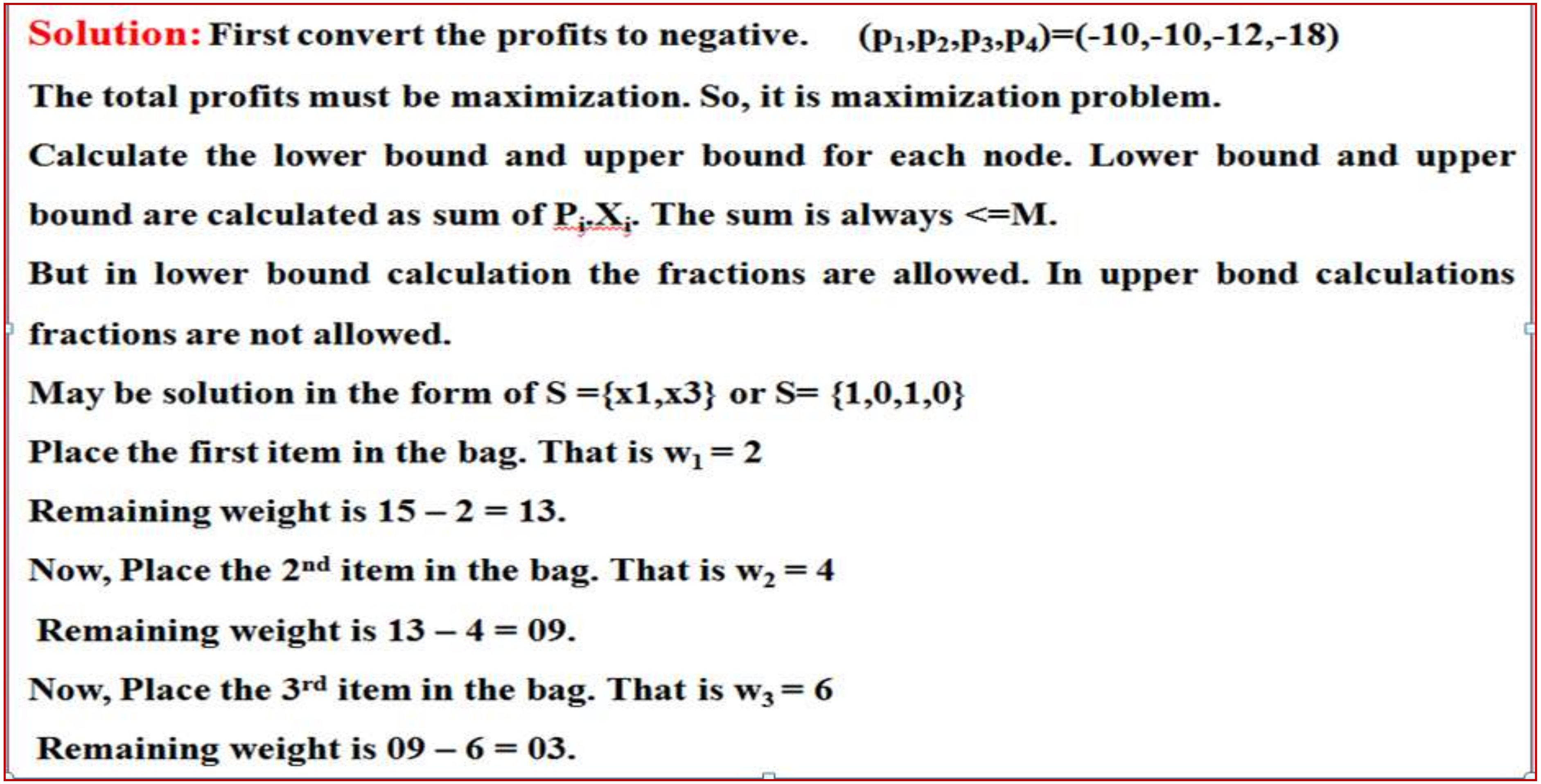
The included or not included form: S = x{1,1,0,4}

The profit is -10 + -10 + 0 + -18 = -38

The maximum profit is 38

And their weights are 2 + 4 + 0 + 9 = 15.

Example : Draw the State Space tree generated by FIFOBB by the following 0/1 Knapsack Problem with 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 Knapsack size m = 15.



Solution: First convert the profits to negative. $(p_1,p_2,p_3,p_4)=(-10,-10,-12,-18)$

The total profits must be maximization. So, it is maximization problem.

Calculate the lower bound and upper bound for each node. Lower bound and upper bound are calculated as sum of $P_i.X_i$. The sum is always <=M.

But in lower bound calculation the fractions are allowed. In upper bond calculations fractions are not allowed.

May be solution in the form of S ={x1,x3} or S= {1,0,1,0}

Place the first item in the bag. That is $w_1 = 2$

Remaining weight is 15 – 2 = 13.

Now, Place the 2nd item in the bag. That is $w_2 = 4$

Remaining weight is 13 – 4 = 09.

Now, Place the 3rd item in the bag. That is $w_3 = 6$

Remaining weight is 09 – 6 = 03.

Initially, the upper bound is ∞.

Now we are calculating the upper bound of node 1, fractions are not allowed.

$\hat{u}(1) = -10 + -10 + -12 = -32$

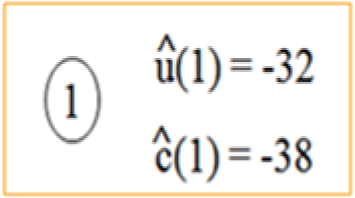
Upper bound is $\cancel{\infty}$

Now - 32

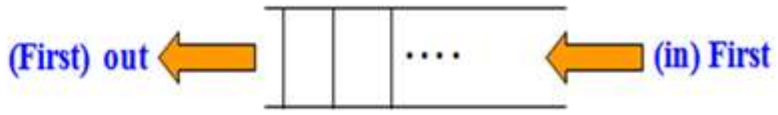
Therefore, $\infty > -32$, the upper bound is -32

To calculate the lower bound, place the 4th item in the bag. Since fractions are allowed.

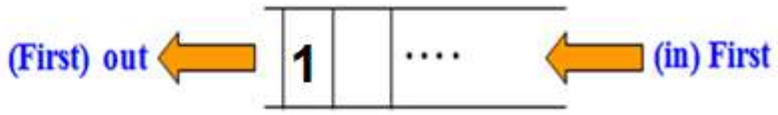
$\hat{c}(1) = -10 + -10 + -12 – 18(3/9) = - 38$



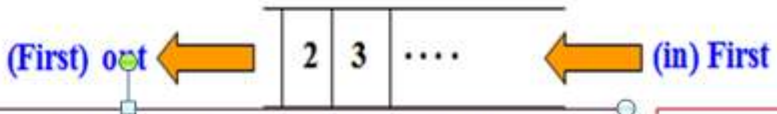
The FIFOBB method uses data structure Queue. Initially Queue is empty



The FIFOBB proceeds with node 1 as the root node and make it as E-node.



Hence node 2 and 3 are generated . Therefore node 2 and 3 are sent to Queue



For Node 2: (x_1 item included. That is x_1=1)

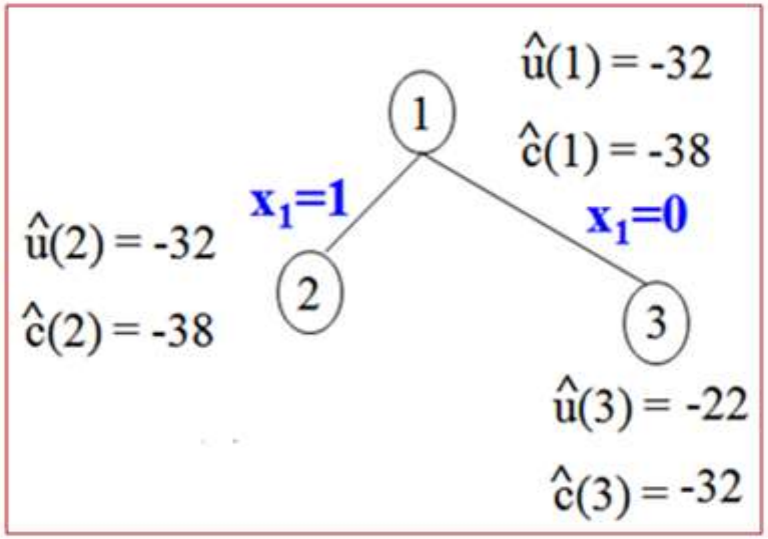
Lower bound (2) = $\hat{c}(2)$ = -10 -10-12-(3/9) X 18 = -38

Upper bound(2) = $\hat{u}(2)$ = -10 -10-12 = -32

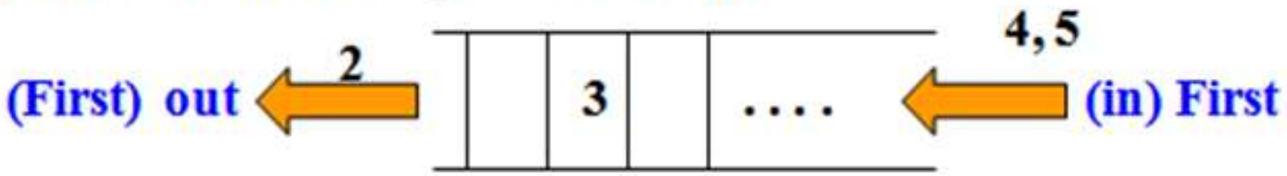
For Node 3: (x_1 item not included. That is x_1=0)

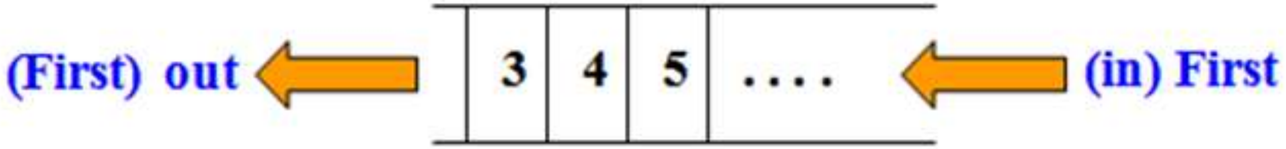
Lower bound (3) = $\hat{c}(3)$ = -10-12-(5/9) X 18 = -32

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



As 2 staying first in the Queue, it becomes E-node and it produces node 4 and 5 as children . Hence the Queue becomes





For Node 4: (x_2 item included. That is x_2=1)

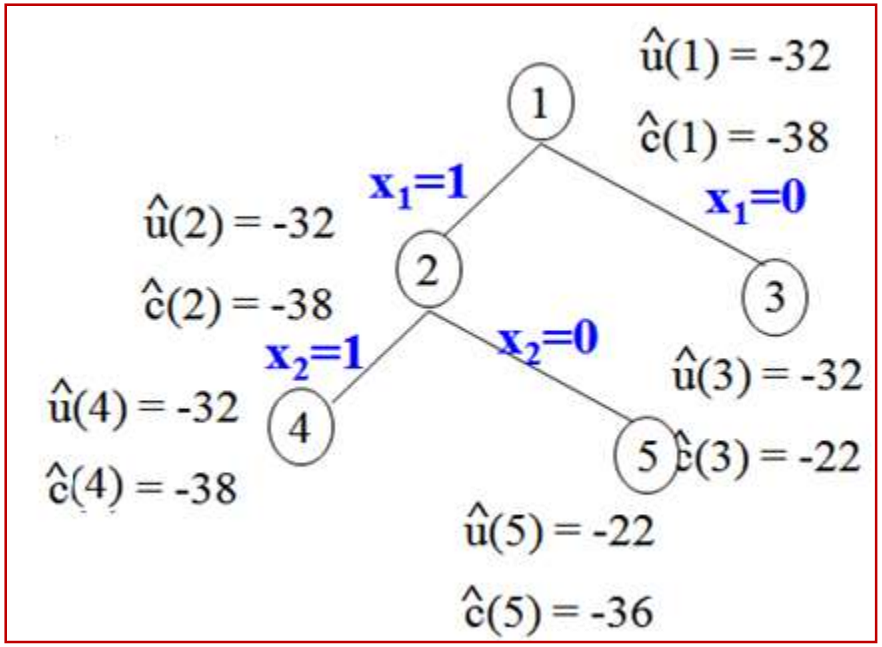
$\hat{c}(4) = -10 - 10 - 12 - (3/9) \times 18 = -38$

$\hat{u}(4) = -10 - 10 - 12 = -32$

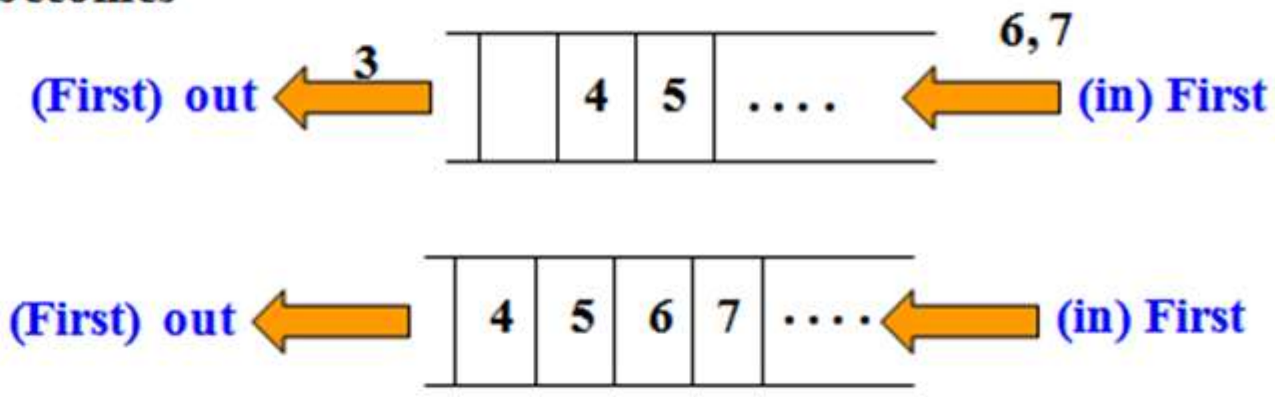
For Node 5: (x_2 item not included. That is x_2=0)

$\hat{c}(5) = -10 - 12 - (7/9) \times 18 = -36$

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



Node 3is the next E-node and it produces node 6 and 7 as children . Hence the Queue becomes



The node 7 is immediately killed because c(7)> upper bound.

At node 6: (x_2 item included $x_2 = 1$)

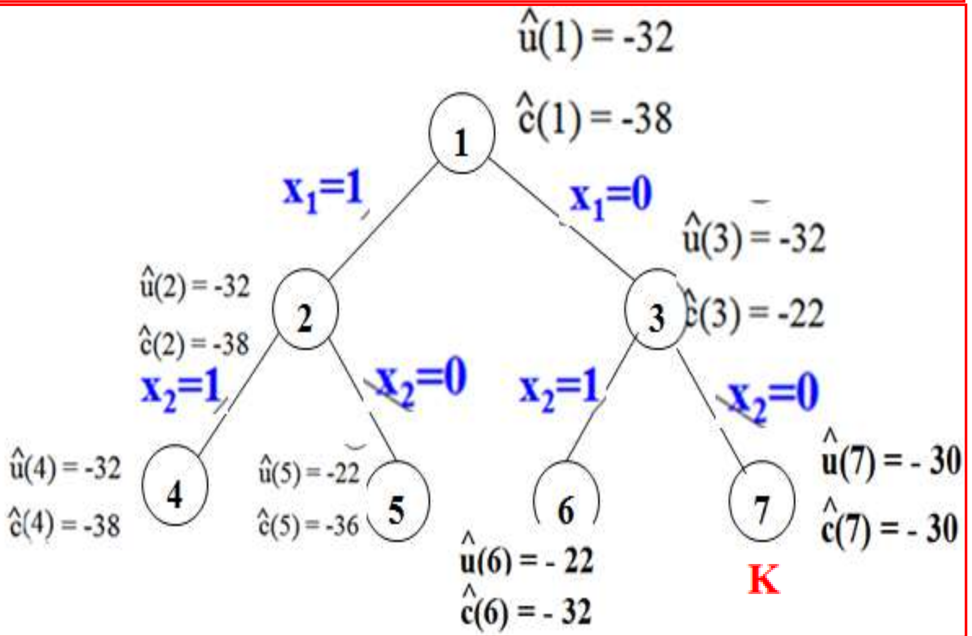
$\hat{u}(6) = -10-12 = -22$

$\hat{c}(6) = -10 -12 - (18/9)X5 = -32$

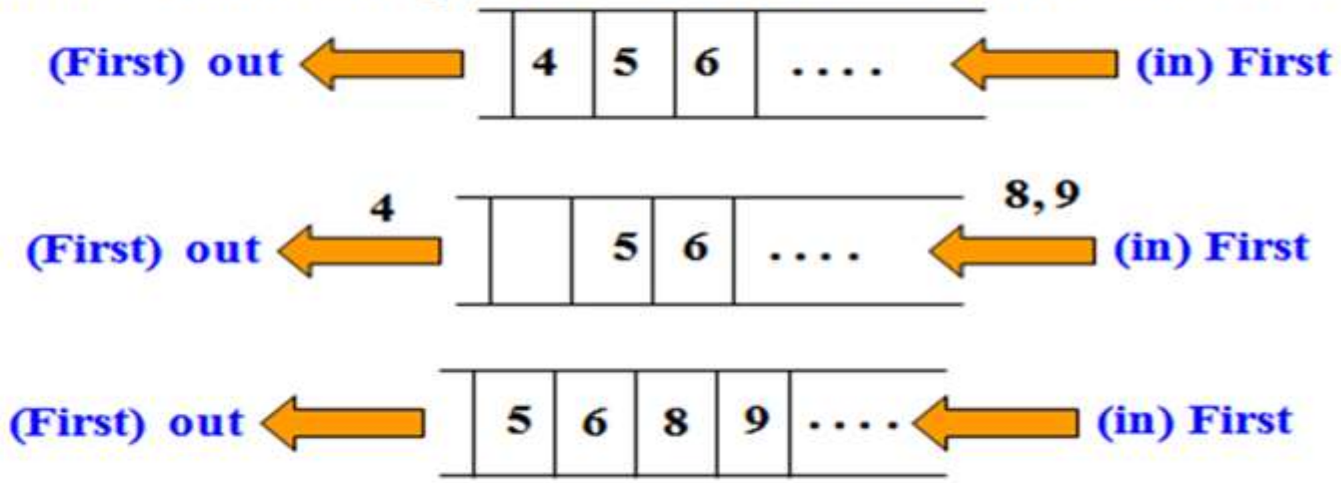
At node 7: (x_2 item not included $x_2 = 0$)

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

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



Then nodes 8 and 9 are produced from the current E-node 4. now the Queue is



At node 8: (x_3 item included $x_3 = 1$)

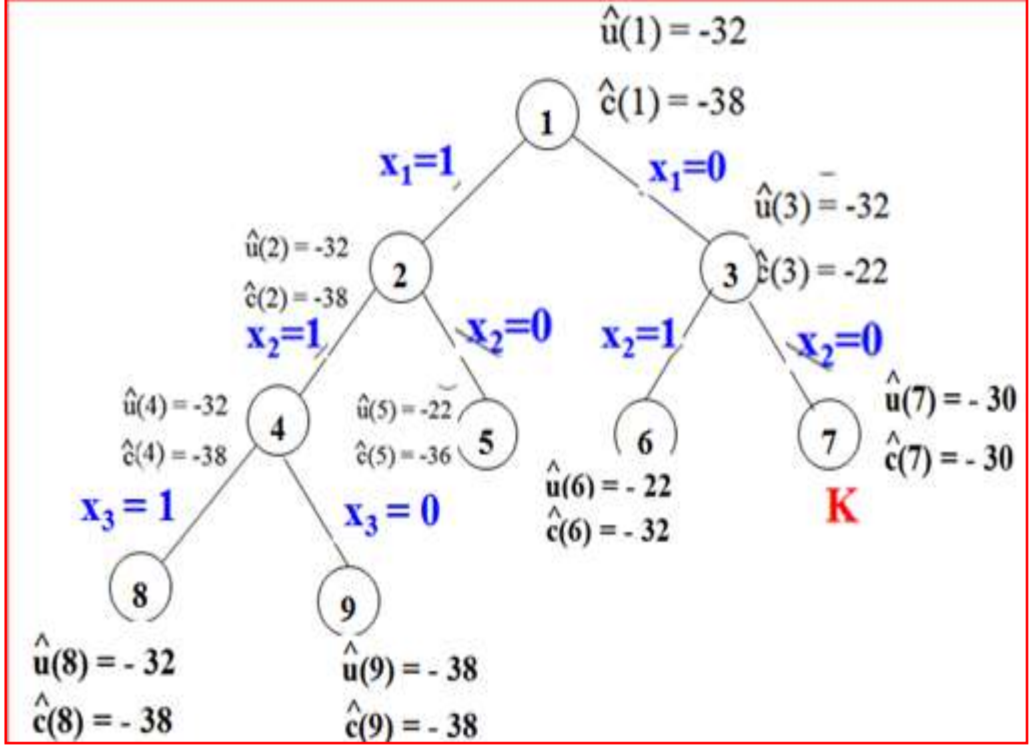
$$\hat{u}(8) = -10 -10 -12 = -32$$

$$\hat{c}(8) = -10 -10 -12 - (18/9)X3 = -38$$

At node 9: (x_3 item not included $x_3 = 0$)

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

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



Observe the upper bound of node 9 is -38 and it is less than the previous upper bound(-32).

Therefore update the upper bound as -38.

Upper bound is -~~32~~

Now - 38

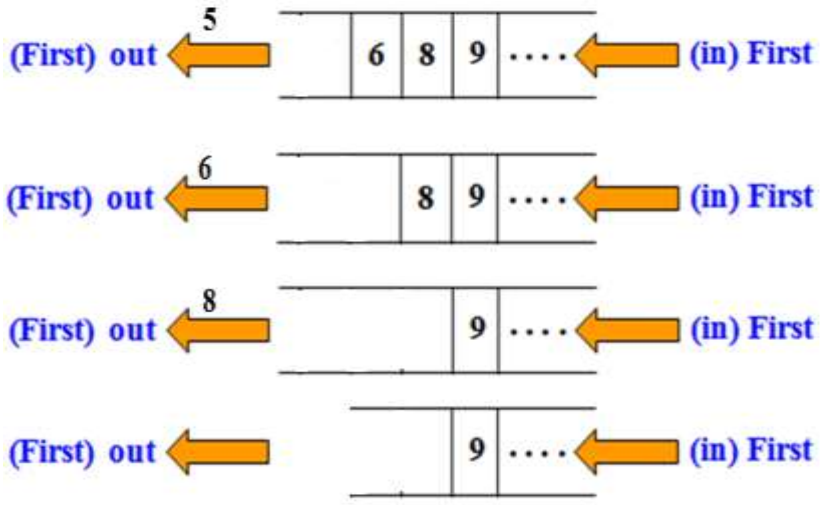
Now observe the costs of node 5 and node 6.

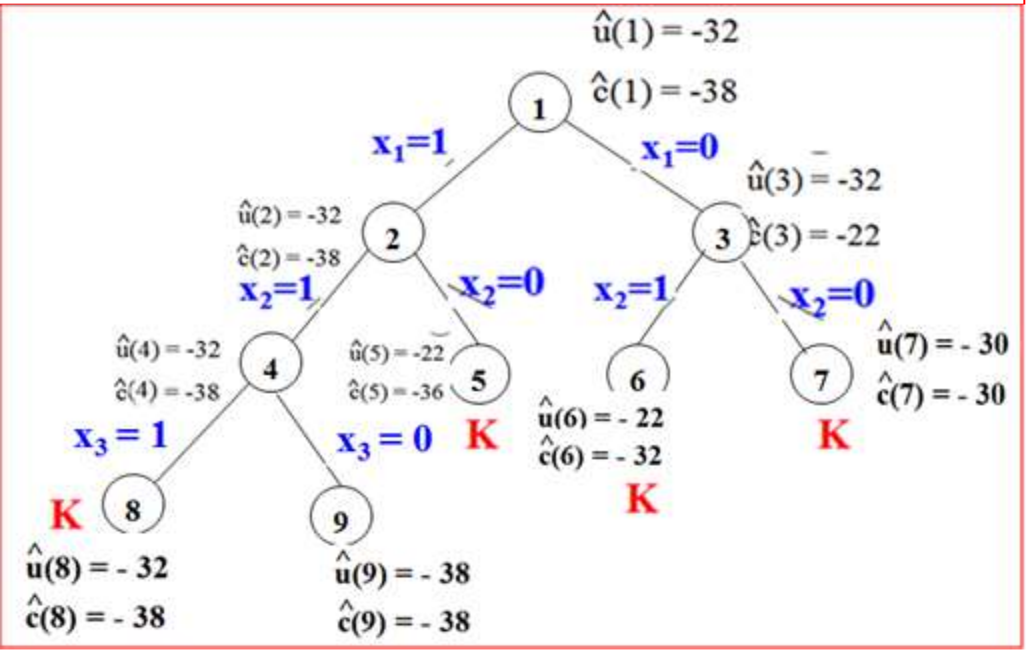
Node 5 cost is -36> upper bound(-38). So kill this node.

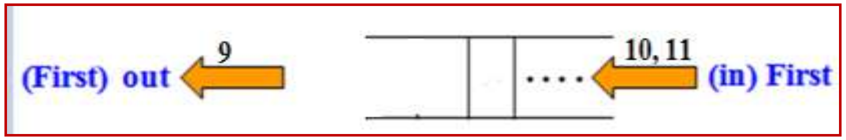
Node 6 cost is -32> upper bound(-38). So kill this node.

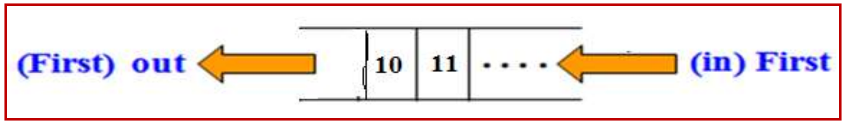
If we expand node 8 and if we add 4th item, the total weight is 2 + 4+ 6 + 9 = 21 and it is grater than the knapsack size(M=15). So it is infeasible solution. So kill this node.

The remaining live node in the queue is 9 and expand it.









At node 10: (x_4 item included $x_4 = 1$)

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

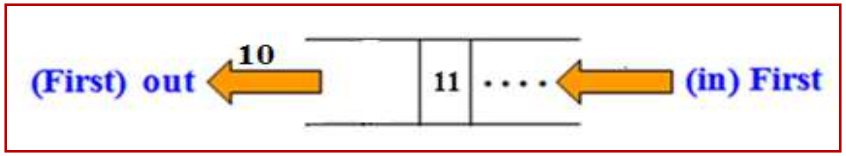
$\hat{c}(10) = -10 -10 -12 - 18 = -38$

At node 11: (x_4 item not included $x_4 = 0$)

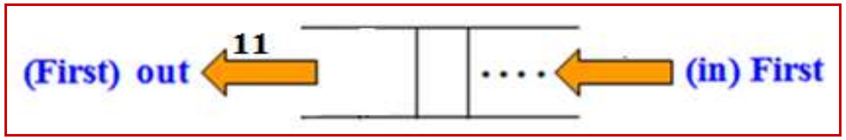
$\hat{u}(11) = -10 -10 = -20$

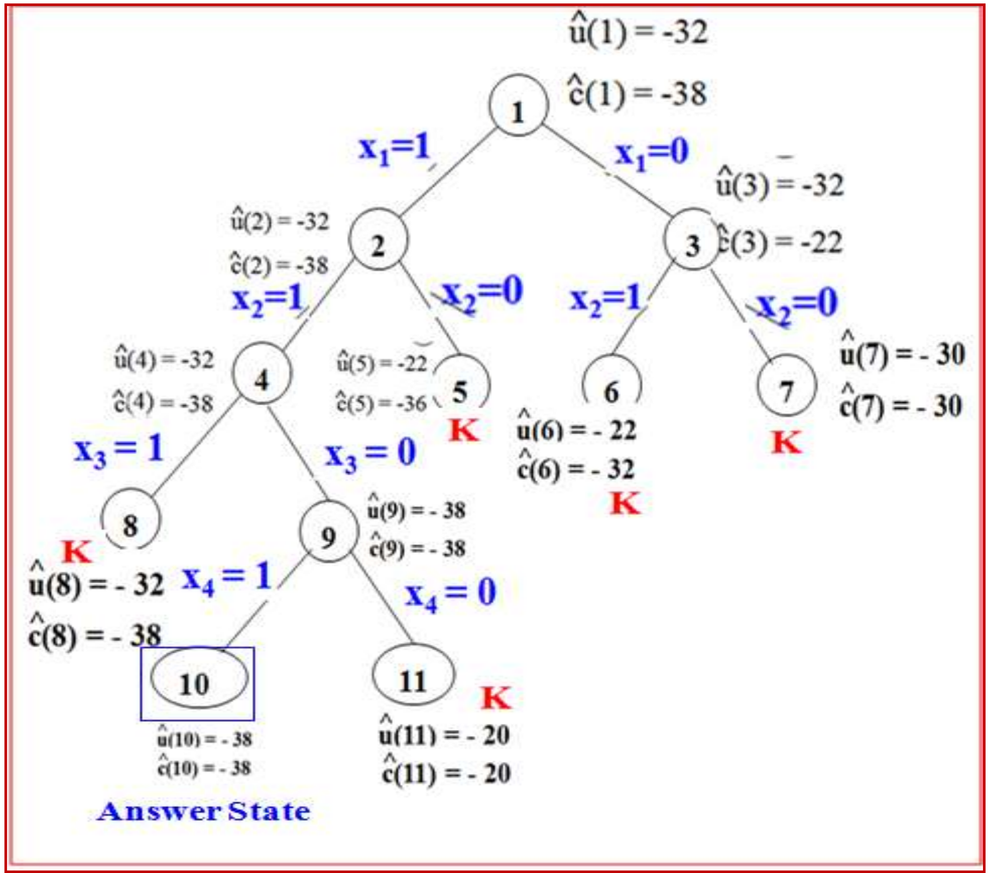
$\hat{c}(11) = -10 - 10 = -20$

Now observe the costs of node 10 and 11. Node 10 cost is -38= upper bound(-38). So it is the Answer node.



Node 11 cost is -20> upper bound(-38). So kill this node.





The solution is, The sub set form : S = {x_1, x_2, x_4}

The included or not included form: S = x{1,1,0,1}

The profit is -10 + -10 + 0 + -18 = - 38

The maximum profit is 38

And their weights are 2 + 4 + 0 + 9 = 15.

FIFO Branch and Bound-Steps to be followed in FIFOBB

Step 1:Compute c(.) and u(.) for a generated node, then if c>upper(initially upper = u(root node)) immediately kill the node otherwise add the node to the list of live nodes.

Step 2:Update upper = u(.), if upper is greater than u(.).

Step 3:The next node in the queue (FIFO) will be the next E-node.

Step 4:Whenever a node about to become an E-node, check if c(.) is greater than upper, if yes kill the node otherwise generate its children. Also kill the nodes which are not feasible.

Step 5: Continue step 1 to 4 till all the live nodes are expanded.

Step 6 : Finally the highest u value (absolute value) node will be the answer node. Trace the path in backward direction from answer node to root node for solution subset.

Example:- Draw the portion of state space tree generated by LCBB by the following Knapsack Problem. n=5, $(p_1,p_2,p_3,p_4,p_5)=(10,15,6,8,4)$ $(w_1,w_2,w_3,w_4,w_5)=(4,6,3,4,2)$ and m=12

Solution: First convert the profits to negative

$(p_1,p_2,p_3,p_4,p_5)=(-10,-15,-6,-8,-4)$

Calculate the lower bound and upper bound for each node

Place the first item in the bag. That is $w_1 = 4$

Remaining weight is 12 – 4 = 08.

Now, Place the 2nd item in the bag. That is $w_2 = 6$

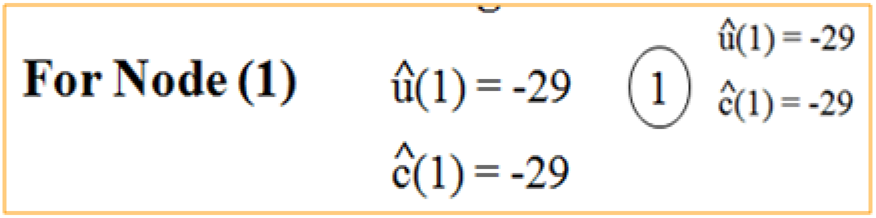
Remaining weight is 08 – 6 = 02.

Since fractions are not allowed in calculation of upper bound, so we can not place the third and 4th items.

Place 5th item. There fore profits earned = -10 -15 - 4 = -29 = Upper bound

To calculate the lower bound, place the third item in the bag. Since fractions are allowed.

Lower bound $\hat{u}$ = -10 -15-(2/3)*6 = -29



<u>For node 2:</u> (x_1 = 1 means we should place first item in the bag)

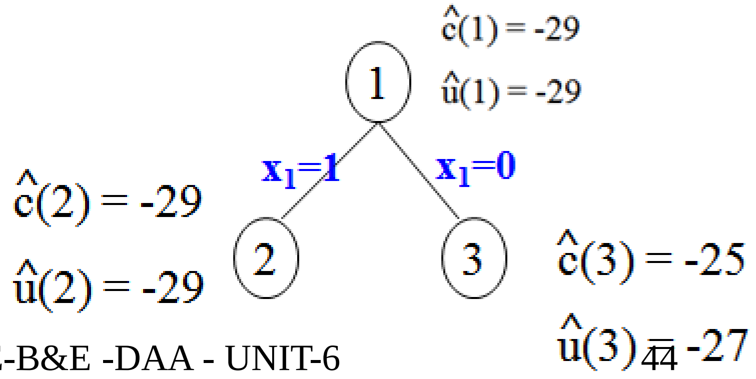
$\hat{u}(2)$ = Lower bound (2) = -10 – 15 – (2/3)*6 = -29

$\hat{c}(2)$ = Upper bound (2) = -10 – 15 – 4 = -29

<u>For node 3:</u> (x_1 = 0 means not included first item in the bag)

$\hat{u}(3)$ = Lower bound(3) = -15 – 6 – (3/4)*8 = -27

$\hat{c}(3)$ = Upper Bound(3) = – 15 – 6 – 4 = -25



Select the minimum lower bound

That is, min{ $\hat{u}(2)$, $\hat{u}(3)$} = min {-29,-27} = -29 = $\hat{u}(2)$

There fore, choose the node 2

There fore, first item is selected, i.e $x_1 = 1$

Consider the 2nd variable to take the decision at 2nd level.

<u>For node 4:</u> (x_2 = 1 means we should place 2nd item in the bag)

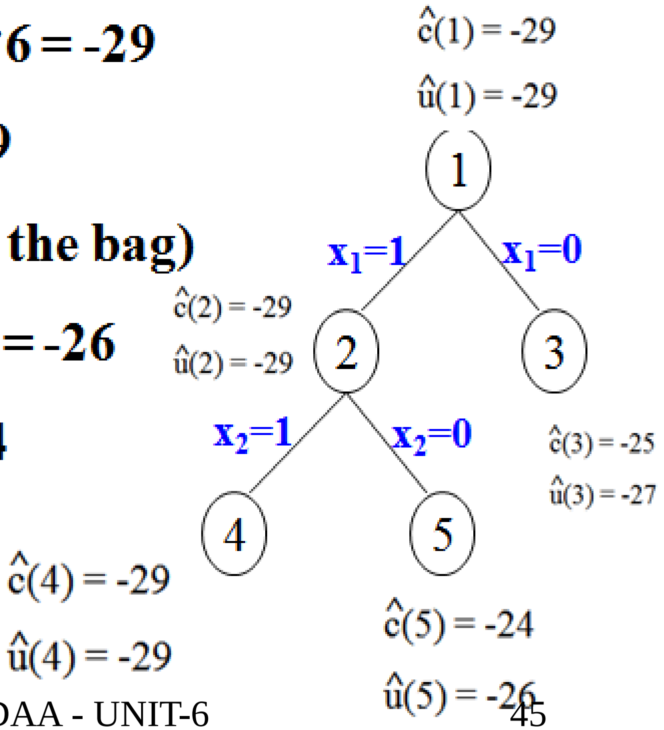
$\hat{u}(4)$ = Lower Bound(4) = -10 – 15 – (2/3)*6 = -29

$\hat{c}(4)$ = Upper bound(4) = -10 – 15 – 4 = -29

<u>For node 5:</u> (x_2 = 0 means not included 2nd item in the bag)

$\hat{u}(5)$ = Lower Bound(4) =-10–6–8–(1/2)*4 = -26

$\hat{c}(5)$ = Upper bound(4) = – 10 – 6 – 8 = -24



Select the minimum lower bound

That is, min{ $\hat{u}(4)$, $\hat{u}(5)$} = min {-29,-26} = -29 = $\hat{u}(4)$

There fore, choose the node 4

There fore, 2nd item is selected, i.e $x_2 = 1$

Consider the 3rd variable to take the decision at 3rd level.

For node 6: (x_3 = 1 means we should place 3rd item in the bag)

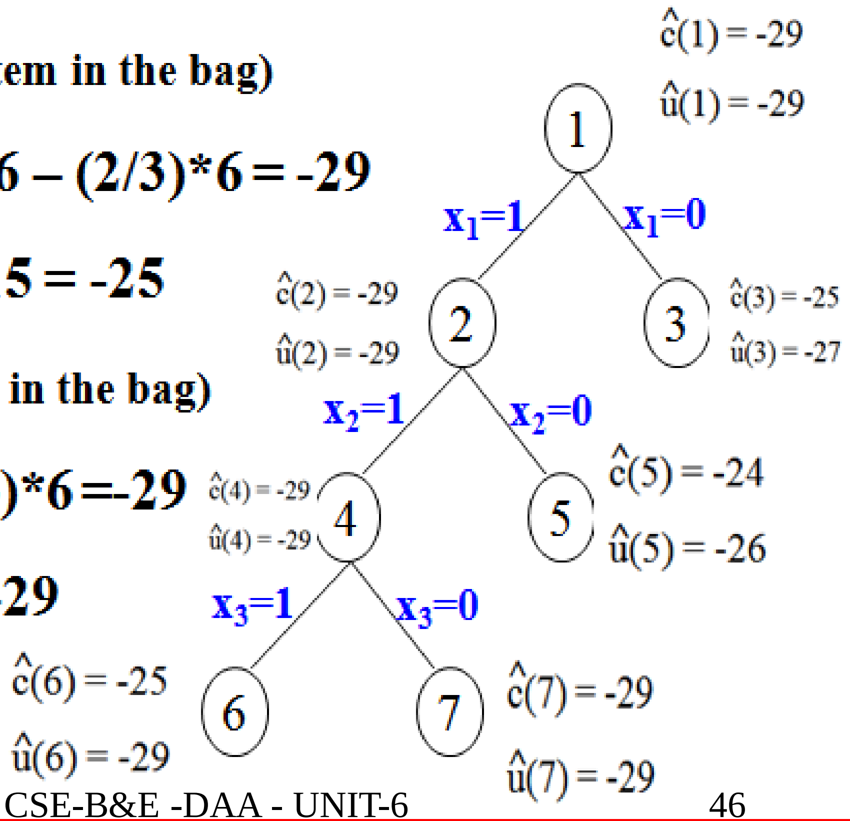
$\hat{u}(6)$ = Lower bound(6) =-10 – 16 – (2/3)*6 = -29

$\hat{c}(6)$ = Upper Bound(6) = -10 – 15 = -25

For node 7: (x_3 = 0 means not included 3rd item in the bag)

$\hat{u}(7)$ = Lower bound(7) = -10–15–(2/3)*6 =-29

$\hat{c}(7)$ = Upper Bound(7) = –10–15–4= -29



Select the minimum lower bound

That is, $\min\{\hat{u}(6), \hat{u}(7)\} = \min\{-29,-29\}$ Both are Equal.

So, select the minimum upper bound

That is, $\min\{\hat{c}(6), \hat{c}(7)\} = \min\{-25,-29\} = \hat{c}(7)$

There fore, choose the node 7

There fore, 3rd item is not selected, i.e $x_3 = 0$

Consider the 4th variable to take the decision at 4th level.

<u>For node 8:</u> **($x_4 = 1$ means we should place 4th item)**

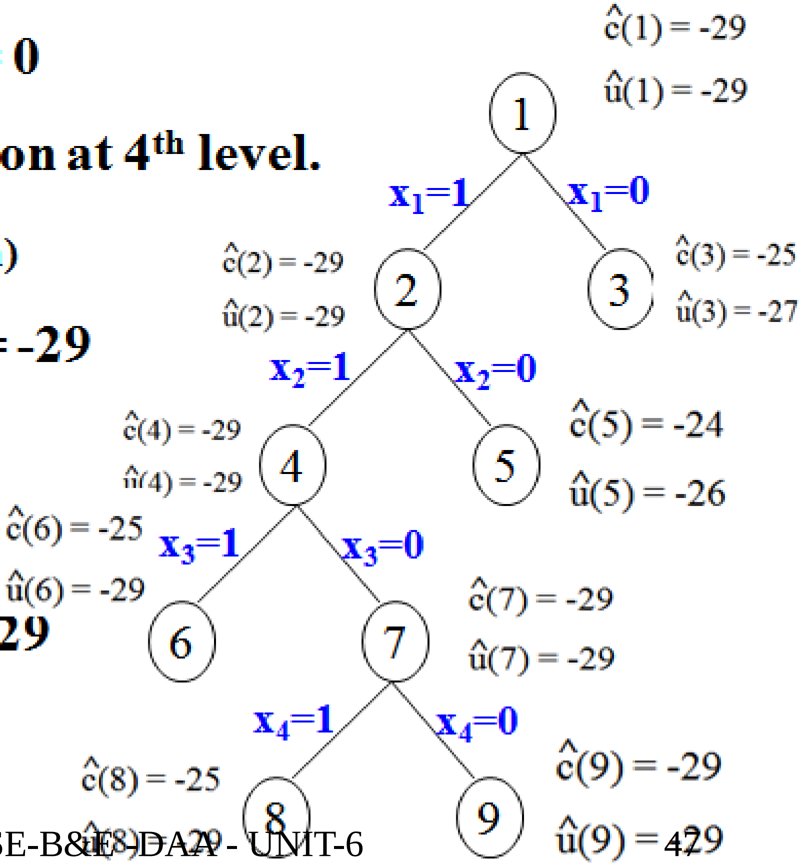
$\hat{u}(8)$ = Lower bound(8) = -10–15–(2/4)*8 = -29

$\hat{c}(8)$ = Upper bound(8)= -10–15 = -25

<u>For node 9:</u> **($x_4 = 0$ means not included 4th item)**

$\hat{u}(9)$=Lower bound(9)= -10–15–(2/3)*6 = -29

$\hat{c}(9)$ = Upper bound(9)– 10 – 15 – 4 = -29



Since the lower bound are equal.

So, select the minimum upper bound

That is, $\min\{\hat{c}(8), \hat{c}(9)\} = \min\{-25,-29\} = \hat{c}(9)$. There fore, choose the node 9.

There fore, 4th item is not selected, i.e $x_4 = 0$

Consider the 5th variable to take the decision at 5th level.

For node 10: (x_5 = 1 means we should place 5th item)

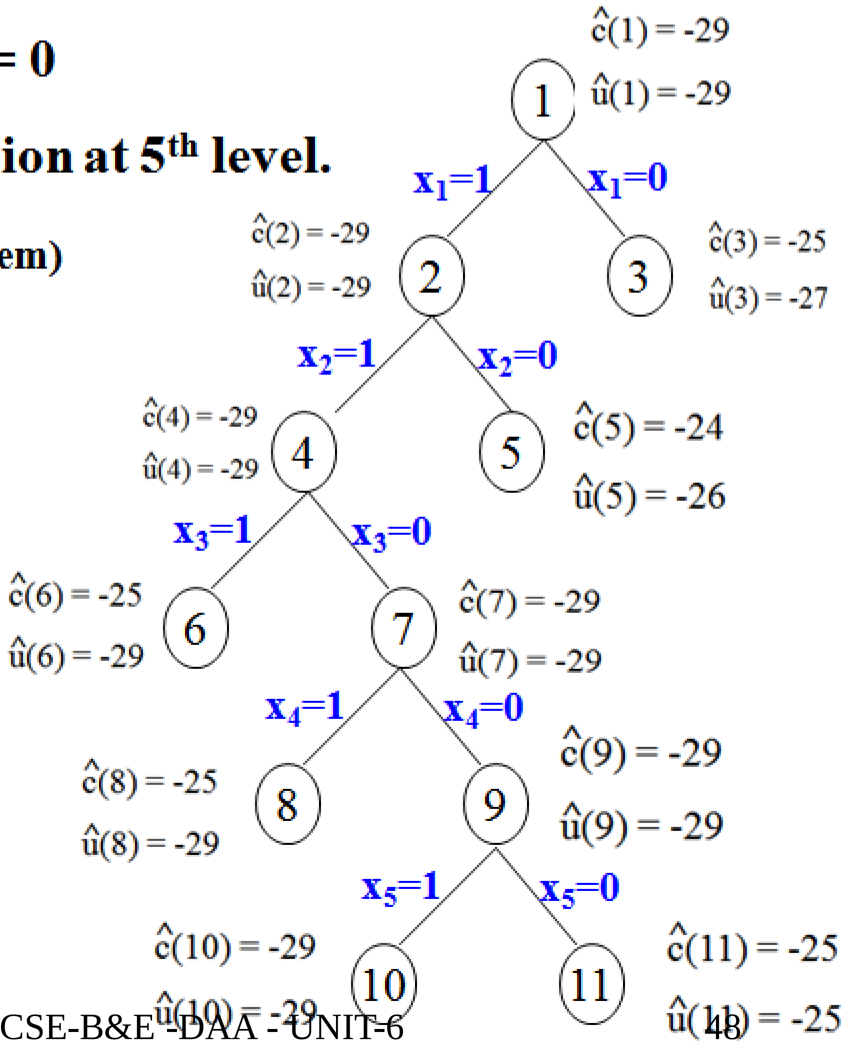
$$\hat{u}(10) = -10 - 15 - 4 = -29$$

$$\hat{c}(10) = -10 - 15 - 4 = -29$$

For node 11: (x_5 = 0 means not included 5th item)

$$\hat{u}(11) = -10 - 15 = -25$$

$$\hat{c}(11) = -10 - 15 = -25$$



So, select the minimum lower bound

That is, min{ $\hat{u}(10)$, $\hat{u}(11)$} = min {-29,-25} = -29 = $\hat{u}(10)$.

There fore, choose the node 10. There fore, 5th item is selected, i.e $x_5 = 1$

There fore the path is 1 – 2 – 4 - 7 – 9 – 10

The solution for Knapsack is $(x_1, x_2, x_3, x_4, x_5)=(1, 1, 0, 0, 1)$

Maximum Profit = 10 + 15 + 4 = 29

****

Home Work :-

1.Draw the portion of the state space tree generated by LCBB and FIFIBB for the following knapsack problem.

a)n=5, (p1,p2,p3,p4,p5)=(10,15,6,8,4), (w1,w2,w3,w4,w5)=(4,6,3,4,2) and m=12

b) n=5, (p1,p2,p3,p4,p5)=(w1,w2,w3,w4,w5)=(4,4,5,8,9) and m=15.

Introduction:

In Computer Science, many problems are solved where the objective is to maximize or minimize some values, whereas in other problems we try to find whether there is a solution or not.

Hence, the problems can be categorized as follows

1. Optimization Problems are those for which the objective is to maximize or minimize some values.

Examples:

- **Finding the minimum number of colors needed to color a given graph.**
- **Finding the shortest path between two vertices in a graph.**

❖**This group consists of the problems that can be solved in polynomial time.**

❖**Example: Searching of an elements from the list O(log n), sorting of elements O(log n).**

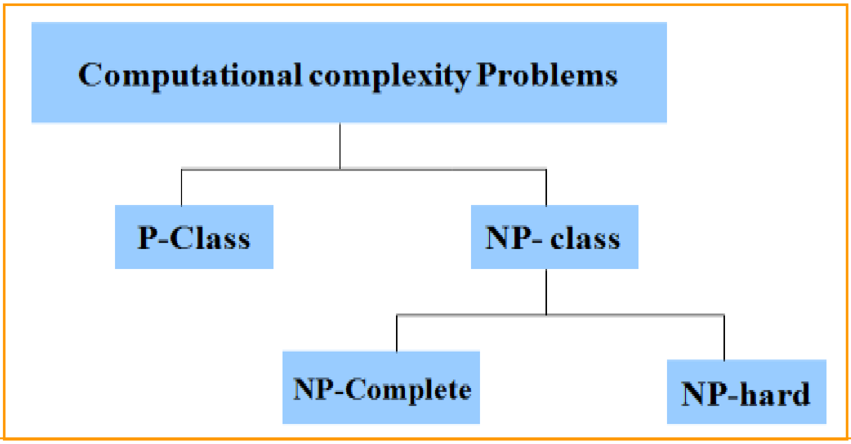
2. Decision Problems **There are many problems for which the answer is a Yes or a No. These types of problems are known as Decision Problems.**

Example:

- **Whether a given graph can be colored by only 4-colors.**
- **Finding Hamiltonian cycle in a graph is not a decision problem, whereas checking a graph is Hamiltonian or not is a decision problem.**

This group consists of problems that can be solved in Non-deterministic Polynomial time.

Example: **Knapsack problem $O(2^{n/2})$ and Travelling Salesperson problem $O(n^2 2^n)$**



P Class Problems

- The class P consists of those problems that are solvable in Polynomial time, i.e. these problems can be solved in time O(n^k) in worst-case, where k is constant. These problems are called Tractable.
- **Example:**
  - 1. Linear Search - O(n)
  - 2. Binary Search – O(logn)
  - 3. Merge Sort
  - 4. Bubble sort
  - 5. Insertion Sort etc…

NP Class Problems

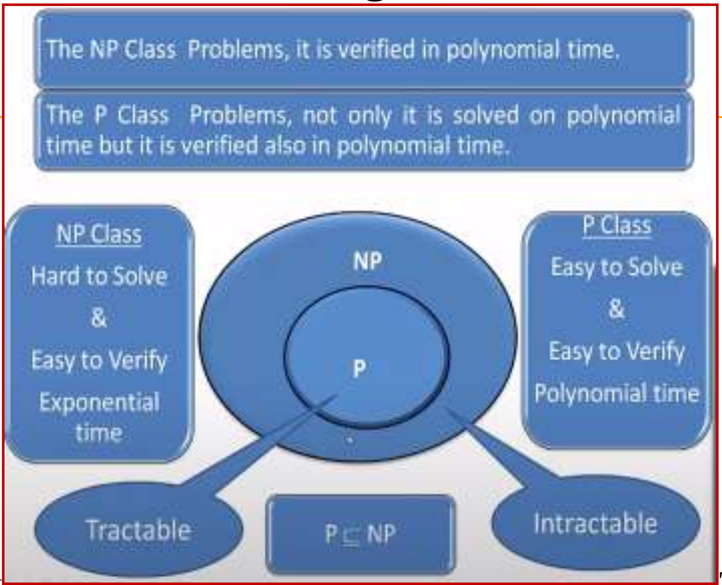
The problem which can not be solved on polynomial time but is verified in polynomial time is known as Non deterministic Polynomial or NP Class problem

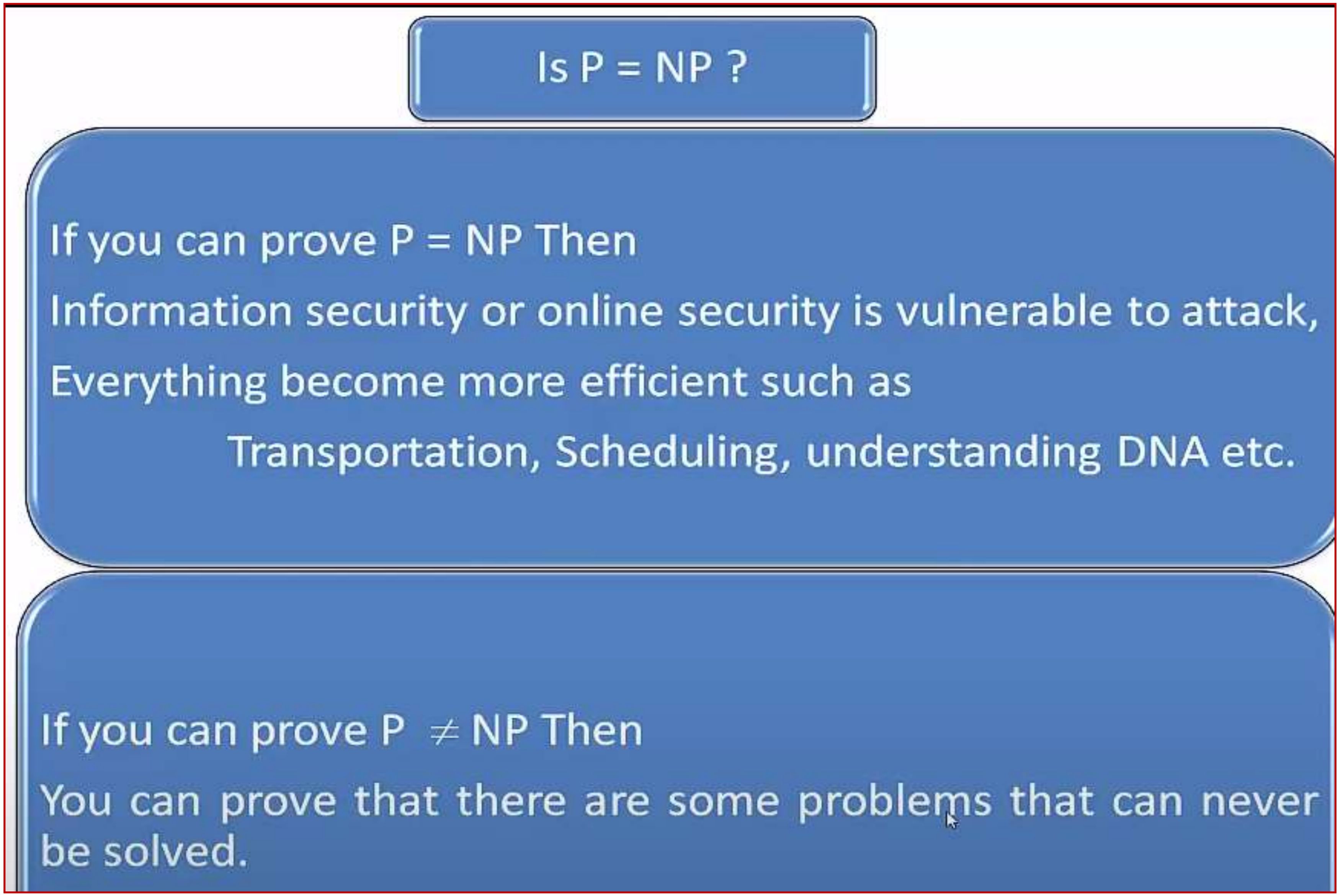
Examples are

Su-Do-Ku, Prime Factor, Scheduling Travelling Sales Person Problem etc…

- **Now there are a lot of programs that don't run in polynomial time on a regular computer, but do run in polynomial time on a nondeterministic Turing machine. These programs solve problems in NP, which stands for Non Deterministic Polynomial time.**

The NP class problems can be further categorized into NP-complete and NP hard problems.


The NP Class Problems, it is verified in polynomial time.
The P Class Problems, not only it is solved on polynomial time but it is verified also in polynomial time.
NP Class
Hard to Solve
&
Easy to Verify
Exponential time
NP
P
P Class
Easy to Solve
&
Easy to Verify
Polynomial time
Tractable
P ⊆ NP
Intractable



Is P = NP ?

If you can prove P = NP Then

Information security or online security is vulnerable to attack,

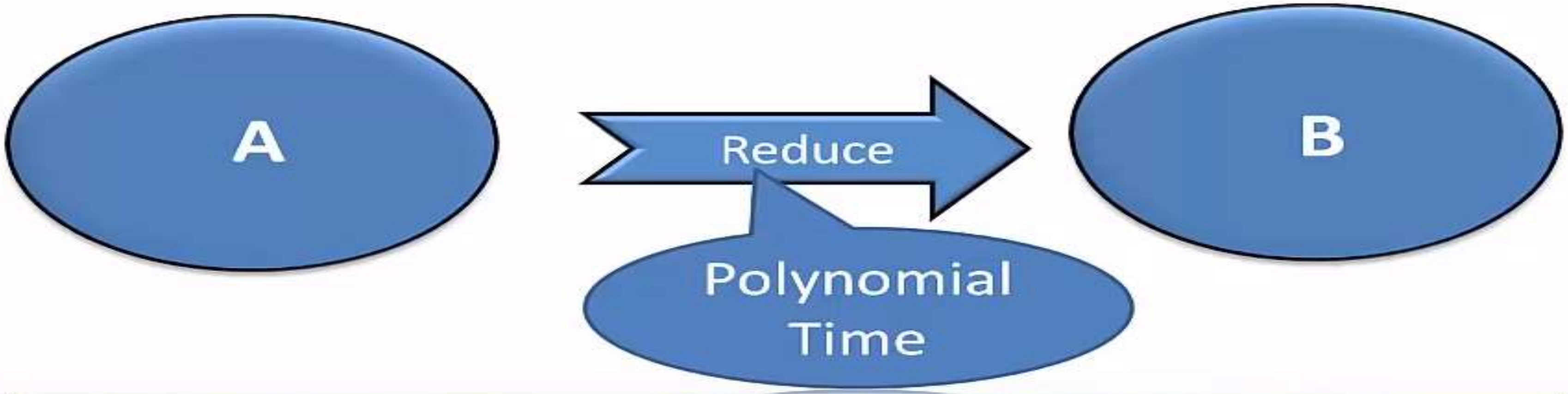
Everything become more efficient such as

Transportation, Scheduling, understanding DNA etc.

If you can prove P $\neq$ NP Then

You can prove that there are some problems that can never be solved.

Reduction:



Let A and B are two problems then problem A reduces to problem B iff there is a way to solve A by deterministic algorithm that solve B in polynomial time.

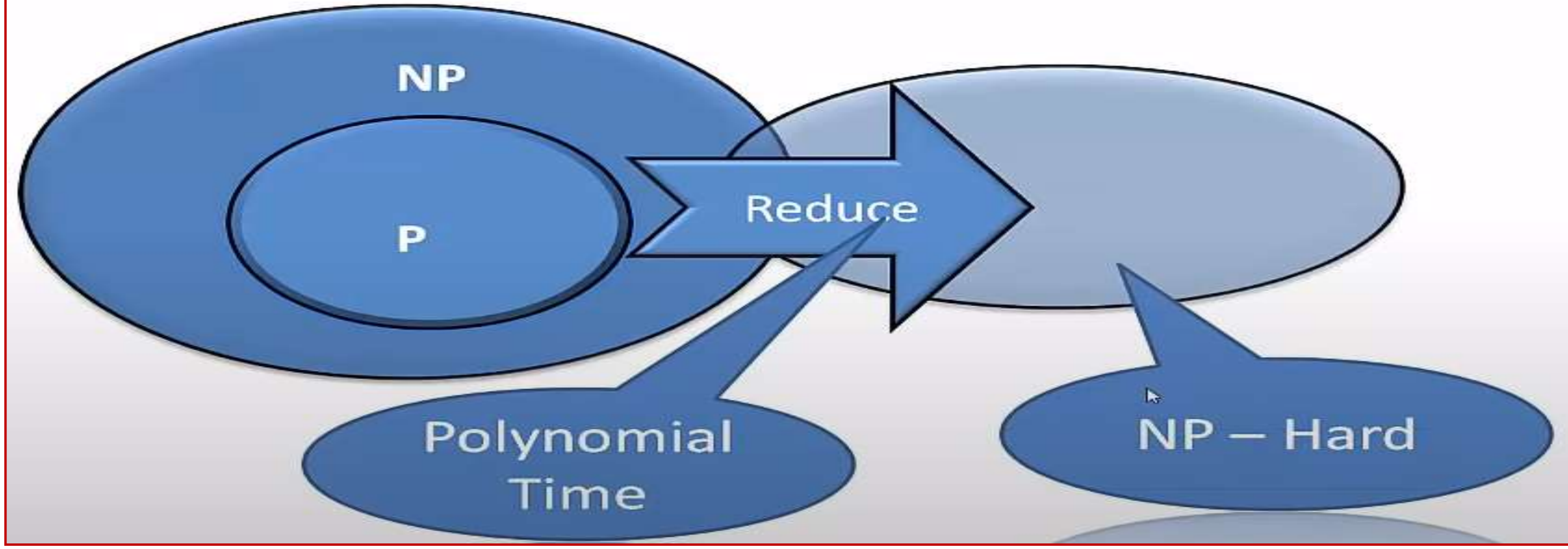
If A is reducible to B we denote it by $A \propto B$

Properties:

1. if A is reducible to B and B in P then A in P.
2. A is not in P implies B is not in P

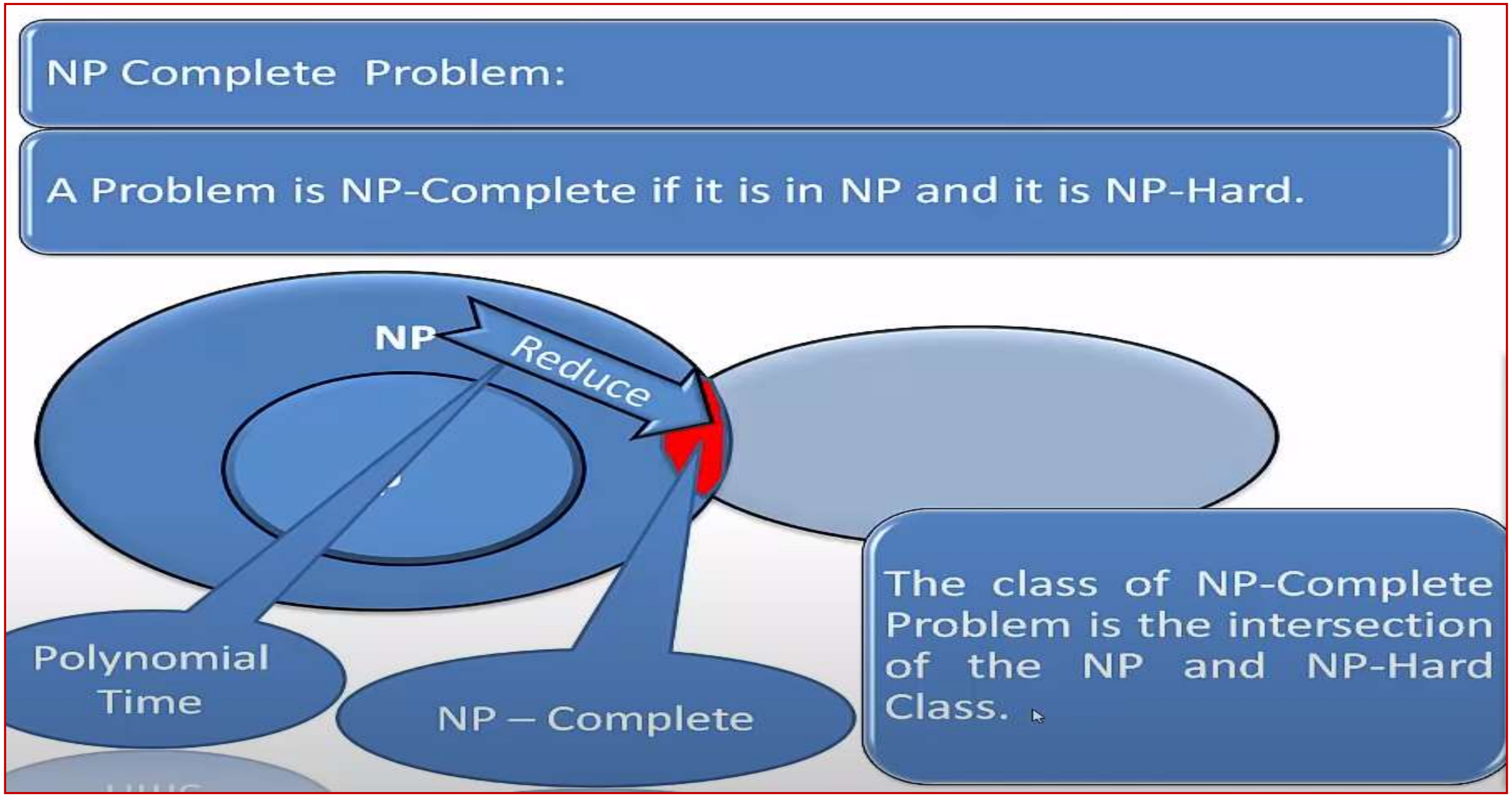
NP Hard Problem:

A Problem is NP-Hard if every problem in NP can be polynomial reduced to it.



Examples are

- **The circuit-satisfiability problem**
- **Set Cover**
- **Vertex Cover**
- **Travelling Salesman Problem**



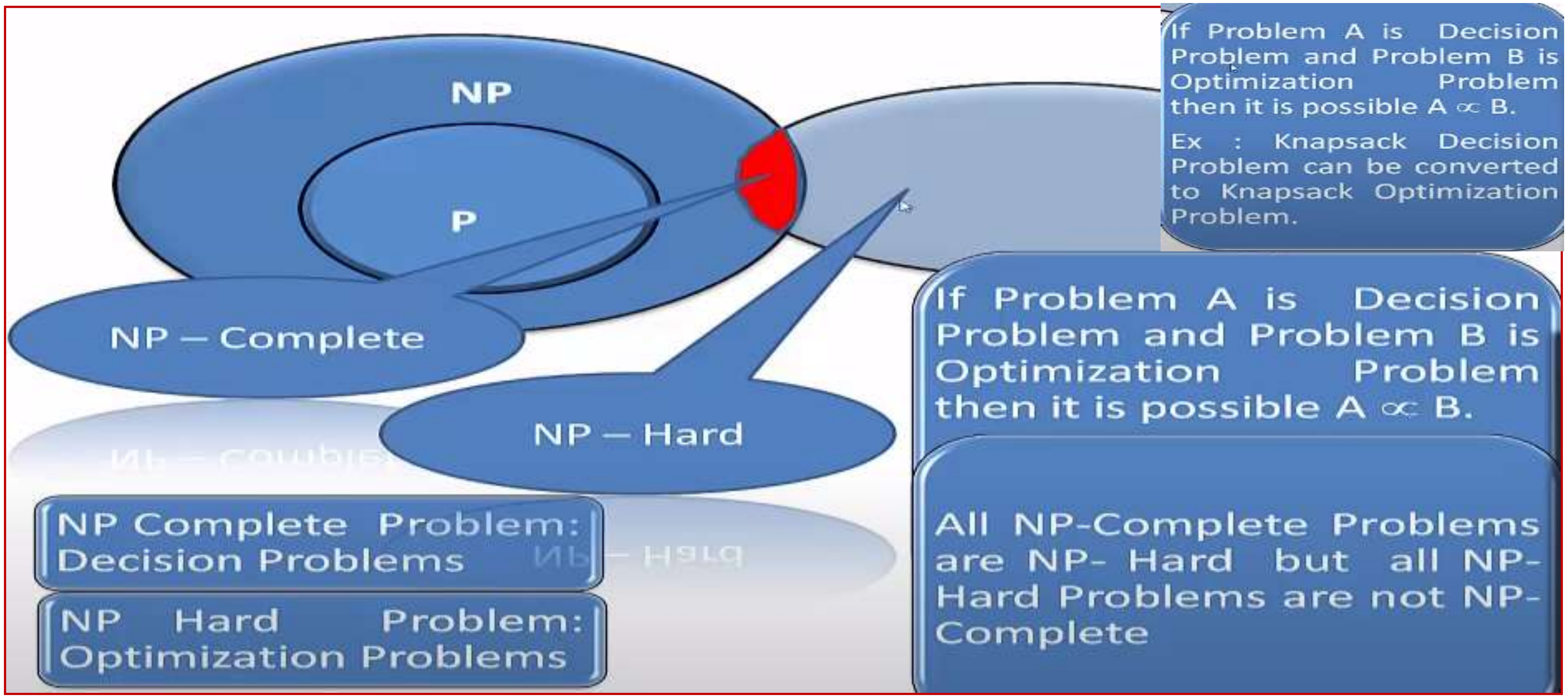
NP Complete Problem:

A Problem is NP-Complete if it is in NP and it is NP-Hard.

The class of NP-Complete Problem is the intersection of the NP and NP-Hard Class.

Examples are

- **Determining whether a graph has a Hamiltonian cycle**
- **Determining whether a Boolean formula is satisfiable etc.**



If Problem A is Decision Problem and Problem B is Optimization Problem then it is possible A $\propto$ B.

Ex : Knapsack Decision Problem can be converted to Knapsack Optimization Problem.

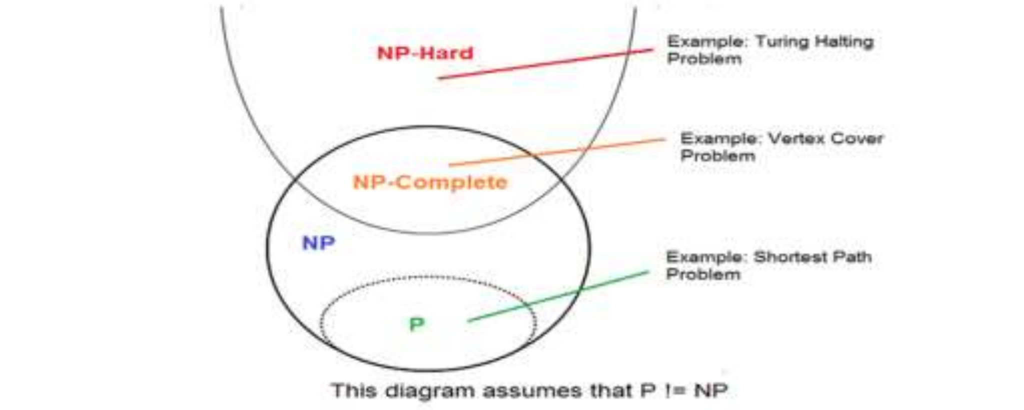
If Problem A is Decision Problem and Problem B is Optimization Problem then it is possible A $\propto$ B.

All NP-Complete Problems are NP- Hard but all NP-Hard Problems are not NP-Complete

NP Complete Problem: Decision Problems

NP Hard Problem: Optimization Problems

Relationship between P, NP, NP - Complete and NP- Hard Problems


NP-Hard
Example: Turing Halting Problem
NP-Complete
Example: Vertex Cover Problem
NP
P
Example: Shortest Path Problem
This diagram assumes that P != NP

NP Complete & NP Hard Class Problems

A language B is NP-complete if it satisfies two conditions

- **B is in NP**
- **Every A in NP is polynomial time reducible to B.**

- **If a language satisfies the second property, but not necessarily the first one, the language B is known as NP-Hard.**
- **Another way:**

Take two NP problems A and B.

Reducibility: If we can convert one instance of a problem A into problem B (NP problem) then it means that A is reducible to B.

NP-hard: Now suppose we found that A is reducible to B, then it means that B is at least as hard as A.

NP-Complete: The group of problems which are both in NP and NP-hard are known as NP-Complete problem.

Non Deterministic Algorithm

•The algorithm in which every operation is uniquely defined is called **Deterministic Algorithm**.

•The algorithm in which every operation may not have unique result, rather there can be specified set of possibilities for every operation. Such an algorithm is called **Non Deterministic Algorithm**.

Non deterministic means that no particular rule is followed to make a guess.

•The non deterministic algorithm is a **two stage** algorithm

Non deterministic ("Guessing") stage: generate an arbitrary string that can be thought of as a candidate solution.

Deterministic ("Verification") stage: In this stage it takes as input the candidate solution and the instance to the problem and returns 'yes' if the candidate solution represents actual solution.

```
1. Algorithm Non_Determin( )
// A[1:n] is a set of elements. We have to determine the index i of A at which element x is located.
2. {   // The following for-loop is the guessing stage
3.          for i=1 to n do
4.              j := choose(i);   // Next is the verification (deterministic) stage
5.          if (A[j] = x) then
6.          {
7.              write(j);
8.              success();
9.          }
10.         write(0);
11.         fail();
12. }
```

In the above given non deterministic algorithm there are three functions used-

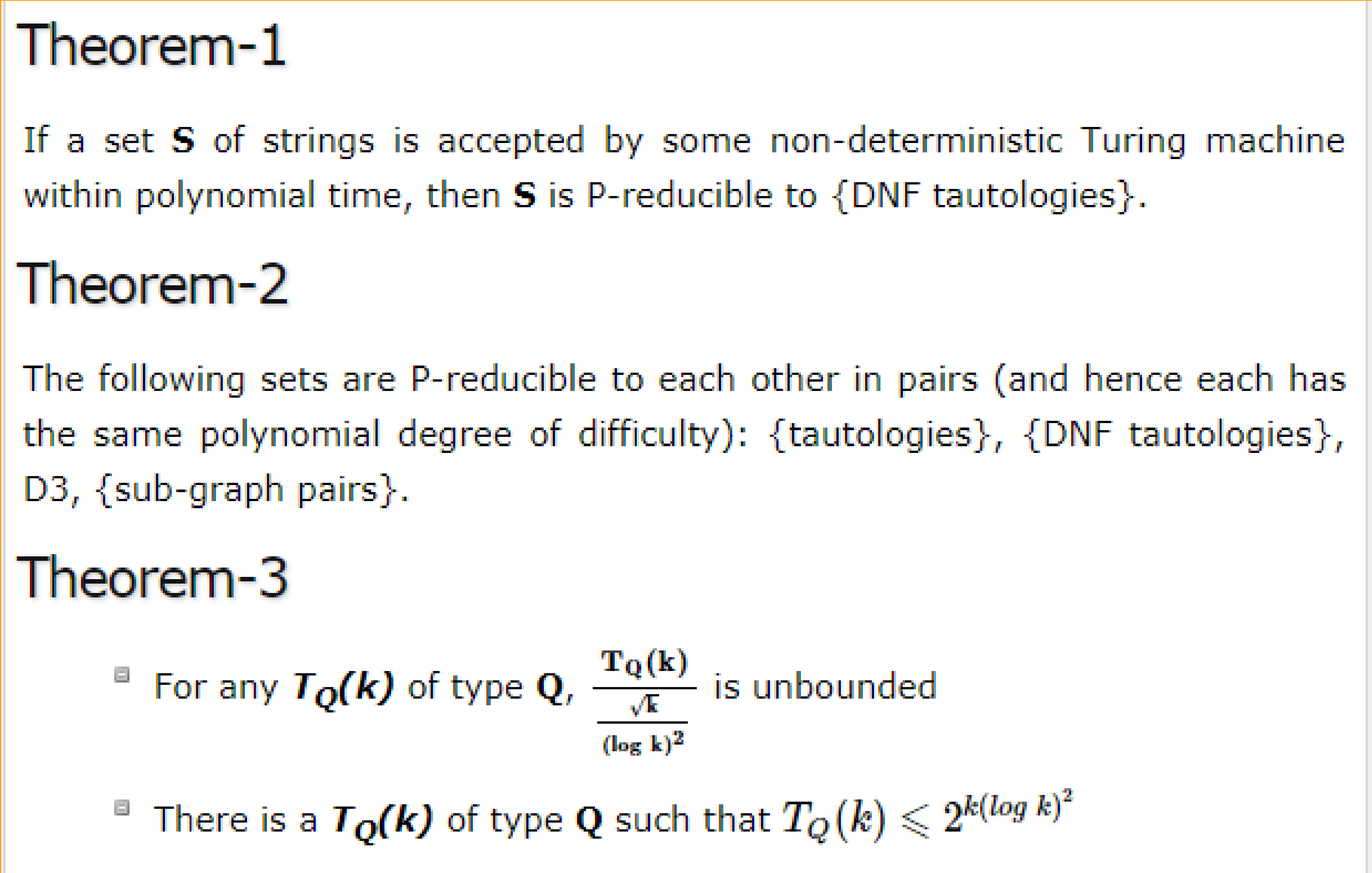
1)<u>Choose</u> **– arbitrarily chooses one of the element from given input set.**

2)<u>Fail</u> **– indicates the unsuccessful completion.**

3)<u>Success</u> **– indicates successful completion**

The algorithm is of non deterministic complexity O(1), when A is not ordered then the deterministic search algorithm has a complexity Ω(n)

Cook’s Theorem : Satisfiabilty problem is NP-Complete



Theorem-1

If a set **S** of strings is accepted by some non-deterministic Turing machine within polynomial time, then **S** is P-reducible to {DNF tautologies}.

Theorem-2

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D3, {sub-graph pairs}.

Theorem-3

- For any $\boldsymbol{T_Q(k)}$ of type $\mathbf{Q}$, $\dfrac{\mathbf{T_Q(k)}}{\frac{\sqrt{\mathbf{k}}}{(\mathbf{\log\ k})^2}}$ is unbounded
- There is a $\boldsymbol{T_Q(k)}$ of type $\mathbf{Q}$ such that $T_Q(k) \leqslant 2^{k(\log\ k)^2}$

Theorem-4

If the set S of strings is accepted by a non-deterministic machine within time $T(n) = 2^n$, and if $T_Q(k)$ is an honest (i.e. real-time countable) function of type Q, then there is a constant K, so S can be recognized by a deterministic machine within time $T_Q(K8^n)$.

First, he emphasized the significance of polynomial time reducibility. It means that if we have a polynomial time reduction from one problem to another, this ensures that any polynomial time algorithm from the second problem can be converted into a corresponding polynomial time algorithm for the first problem.

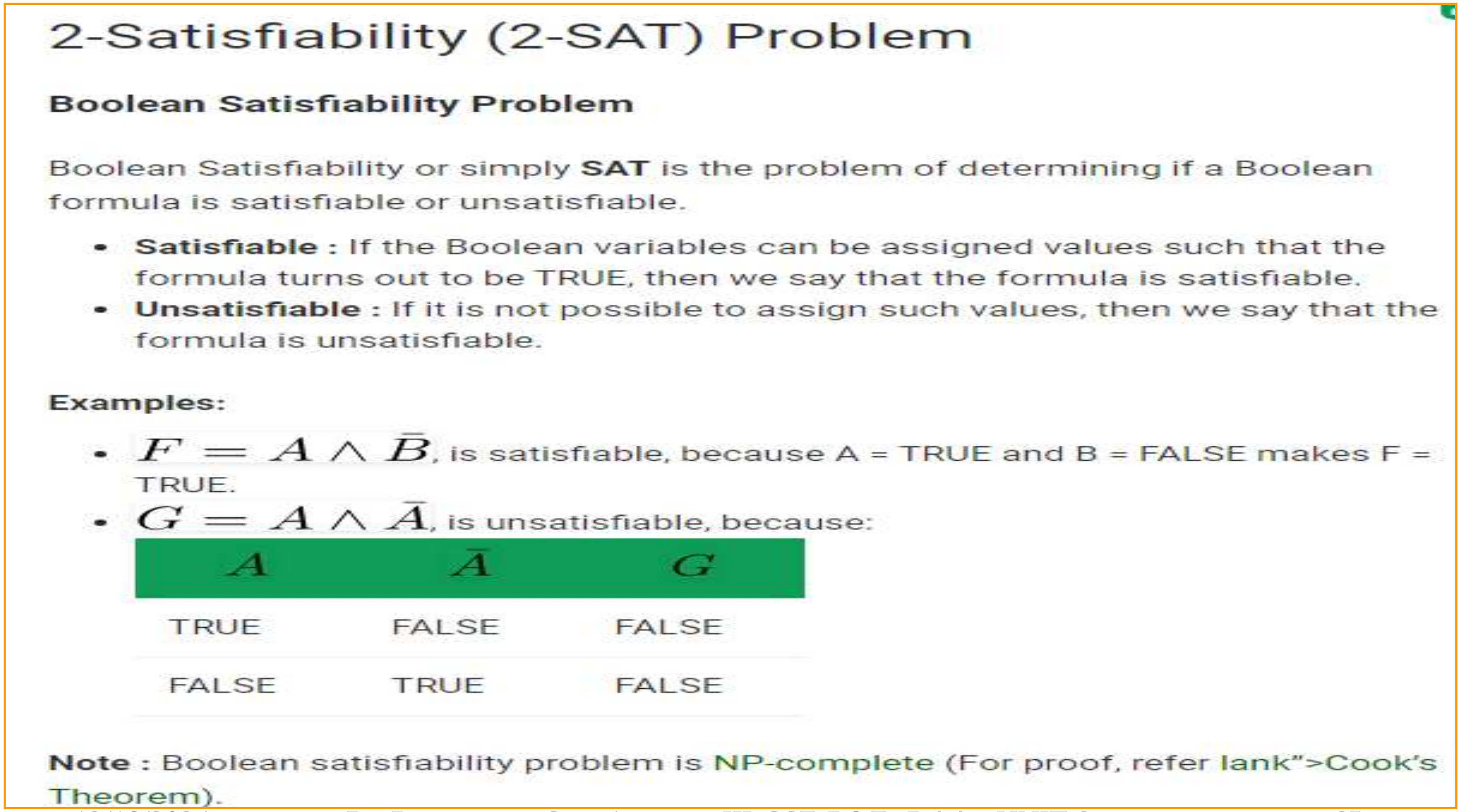
Second, he focused attention on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer. Most of the intractable problems belong to this class, NP.

Third, he proved that one particular problem in NP has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then every problem in NP can also be solved in polynomial time. If any problem in NP is intractable, then satisfiability problem must be intractable. Thus, satisfiability problem is the hardest problem in NP.

Fourth, Cook suggested that other problems in NP might share with the satisfiability problem this property of being the hardest member of NP.

N-Satisfiability is NP Hard Problem and it runs in exponential time.

If it solve in polynomial time then this problem is called as NP-Complete



2-Satisfiability (2-SAT) Problem

Boolean Satisfiability Problem

Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

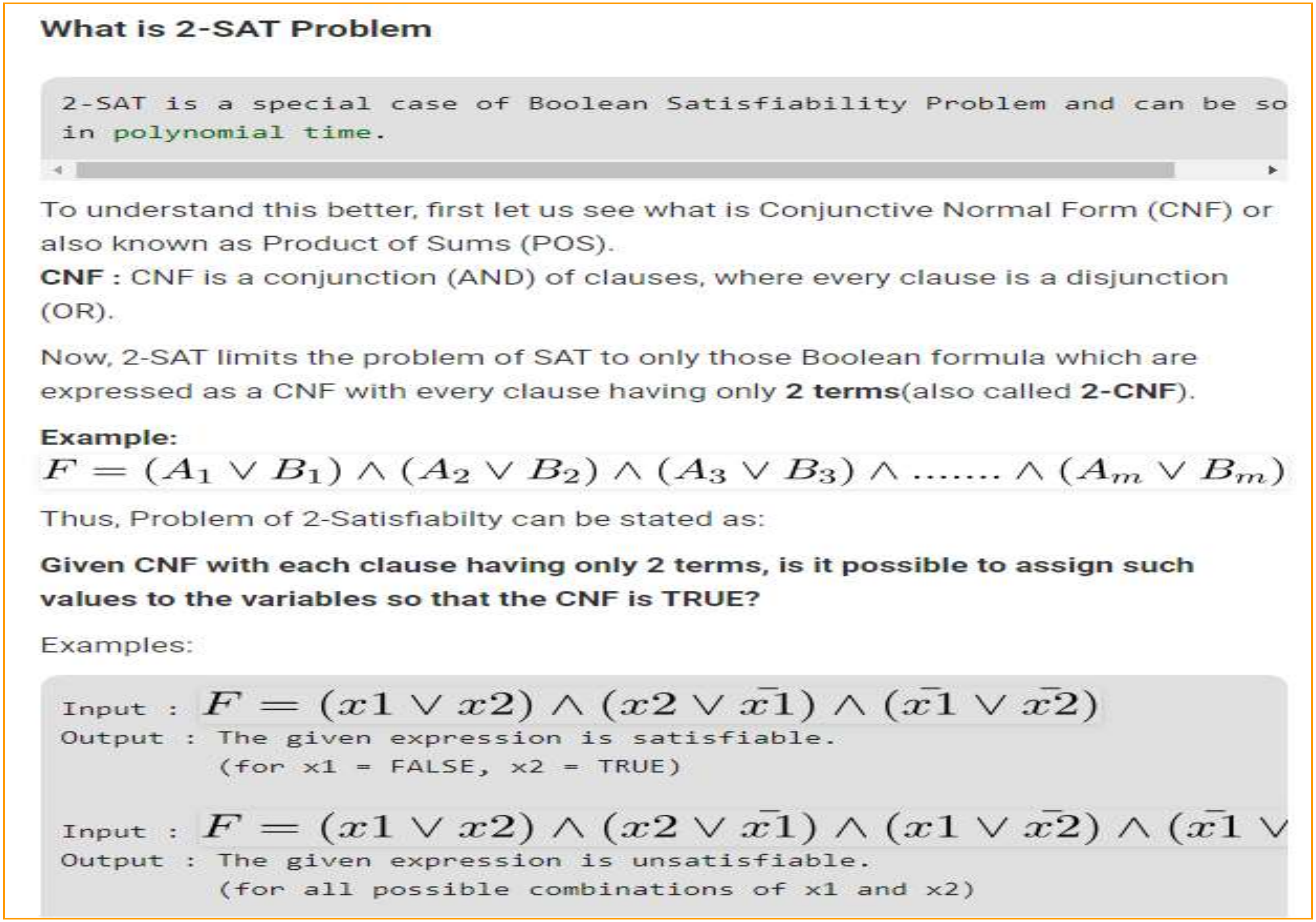
- **Satisfiable :** If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
- **Unsatisfiable :** If it is not possible to assign such values, then we say that the formula is unsatisfiable.

Examples:

- $F = A \wedge \bar{B}$, is satisfiable, because A = TRUE and B = FALSE makes F = TRUE.
- $G = A \wedge \bar{A}$, is unsatisfiable, because:

A	$\bar{A}$	G
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

Note : Boolean satisfiability problem is NP-complete (For proof, refer lank">Cook's Theorem).



What is 2-SAT Problem

```
2-SAT is a special case of Boolean Satisfiability Problem and can be so
in polynomial time.
```

To understand this better, first let us see what is Conjunctive Normal Form (CNF) or also known as Product of Sums (POS).

CNF : CNF is a conjunction (AND) of clauses, where every clause is a disjunction (OR).

Now, 2-SAT limits the problem of SAT to only those Boolean formula which are expressed as a CNF with every clause having only **2 terms**(also called **2-CNF**).

Example:

$$F = (A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge (A_3 \vee B_3) \wedge \ldots\ldots \wedge (A_m \vee B_m)$$

Thus, Problem of 2-Satisfiabilty can be stated as:

Given CNF with each clause having only 2 terms, is it possible to assign such values to the variables so that the CNF is TRUE?

Examples:

Input : $F = (x1 \vee x2) \wedge (x2 \vee \bar{x}1) \wedge (\bar{x}1 \vee \bar{x}2)$

```
Output : The given expression is satisfiable.
         (for x1 = FALSE, x2 = TRUE)
```

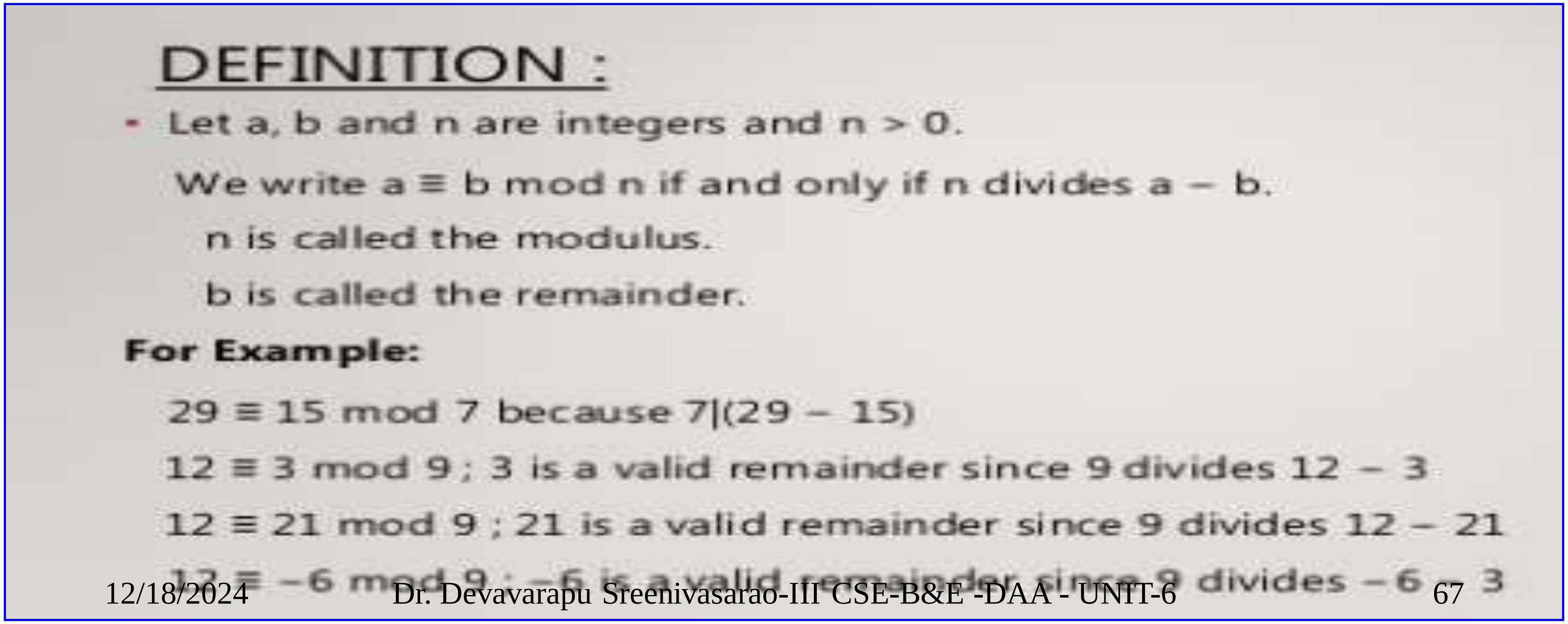
Input : $F = (x1 \vee x2) \wedge (x2 \vee \bar{x}1) \wedge (x1 \vee \bar{x}2) \wedge (\bar{x}1 \vee$

```
Output : The given expression is unsatisfiable.
         (for all possible combinations of x1 and x2)
```

Halting Problem: It is an example of NP-Hard decision problem that is not NP-Complete.

The halting problem A and an input I whether algorithm A with input I ever terminates. This problem un decidable. That is there exist no algorithm to solve this problem.

So it clearly can not be in NP.



DEFINITION :

- Let a, b and n are integers and n > 0.

  We write a ≡ b mod n if and only if n divides a − b.

  n is called the modulus.

  b is called the remainder.

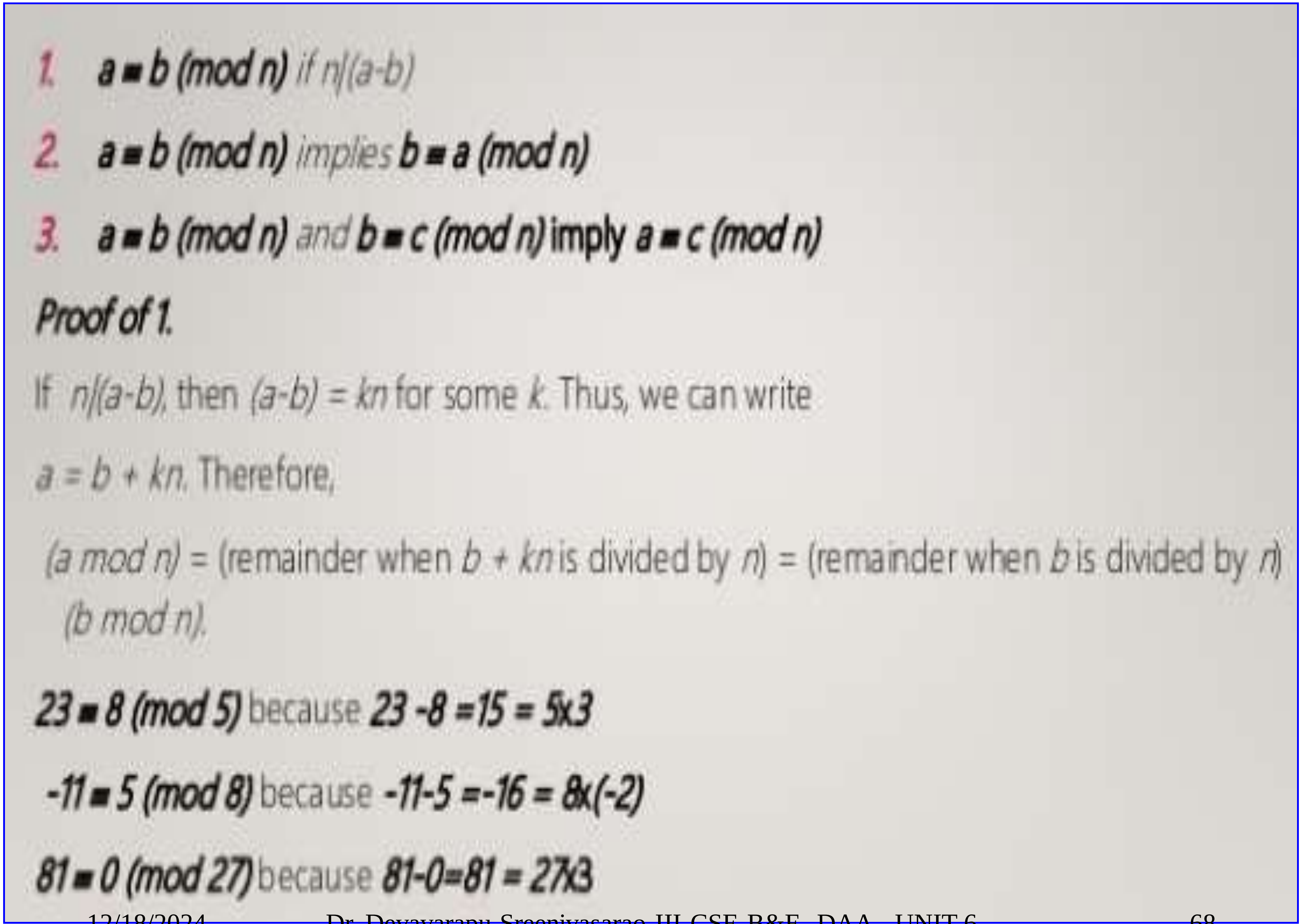
For Example:

29 ≡ 15 mod 7 because 7|(29 − 15)

12 ≡ 3 mod 9 ; 3 is a valid remainder since 9 divides 12 − 3

12 ≡ 21 mod 9 ; 21 is a valid remainder since 9 divides 12 − 21

12 ≡ −6 mod 9 ; −6 is a valid remainder since 9 divides −6 − 3



1. **a ■ b (mod n)** *if n|(a-b)*
2. **a ■ b (mod n)** *implies* **b ■ a (mod n)**
3. **a ■ b (mod n)** *and* **b ■ c (mod n) imply a ■ c (mod n)**

Proof of 1.

If *n|(a-b)*, then *(a-b) = kn* for some *k*. Thus, we can write

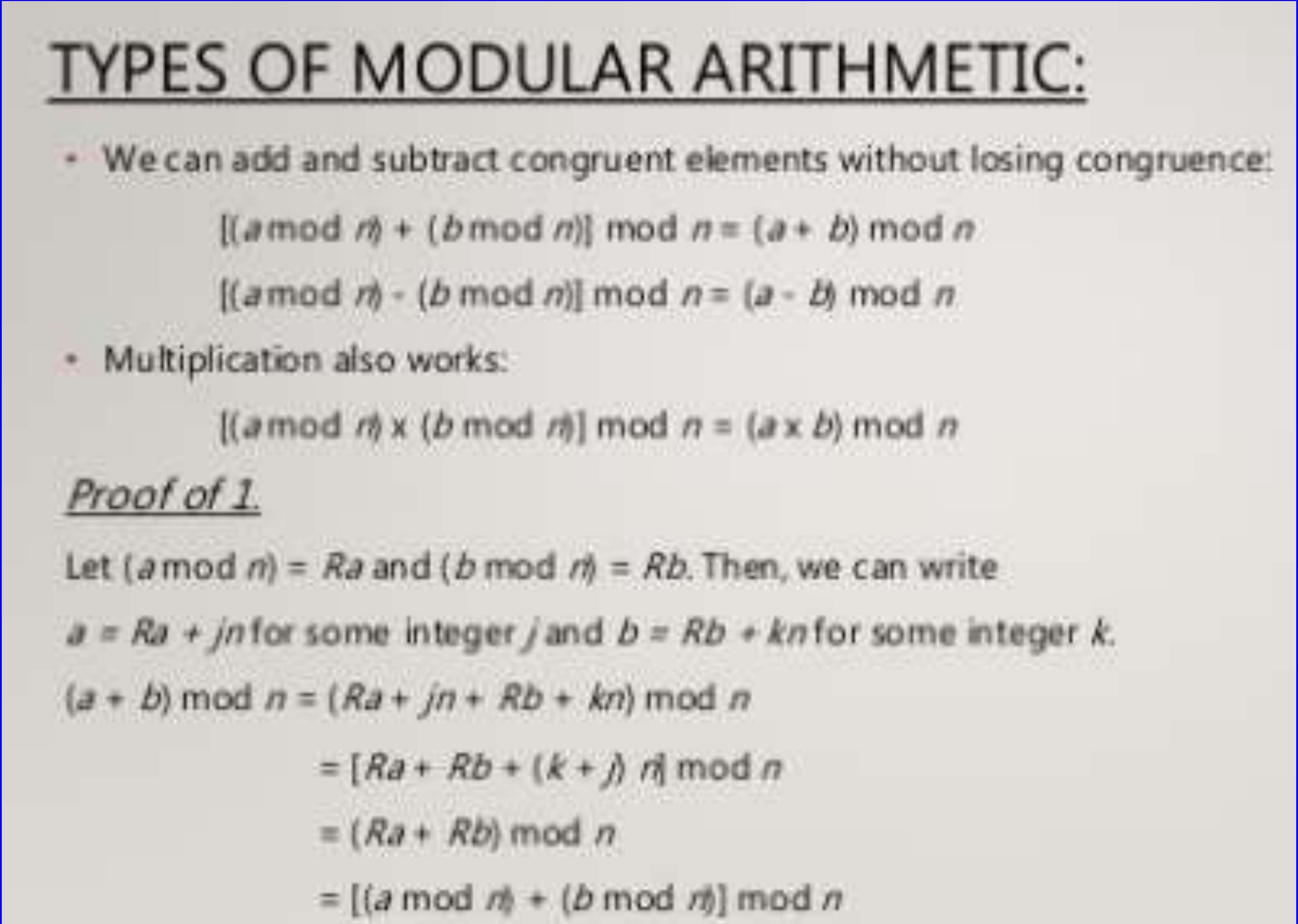
a = b + kn. Therefore,

(a mod n) = (remainder when *b + kn* is divided by *n*) = (remainder when *b* is divided by *n*) *(b mod n)*.

23 ■ 8 (mod 5) because ***23 -8 =15 = 5x3***

-11 ■ 5 (mod 8) because ***-11-5 =-16 = 8x(-2)***

81 ■ 0 (mod 27) because ***81-0=81 = 27x3***



TYPES OF MODULAR ARITHMETIC:

- We can add and subtract congruent elements without losing congruence:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$

$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$

- Multiplication also works:

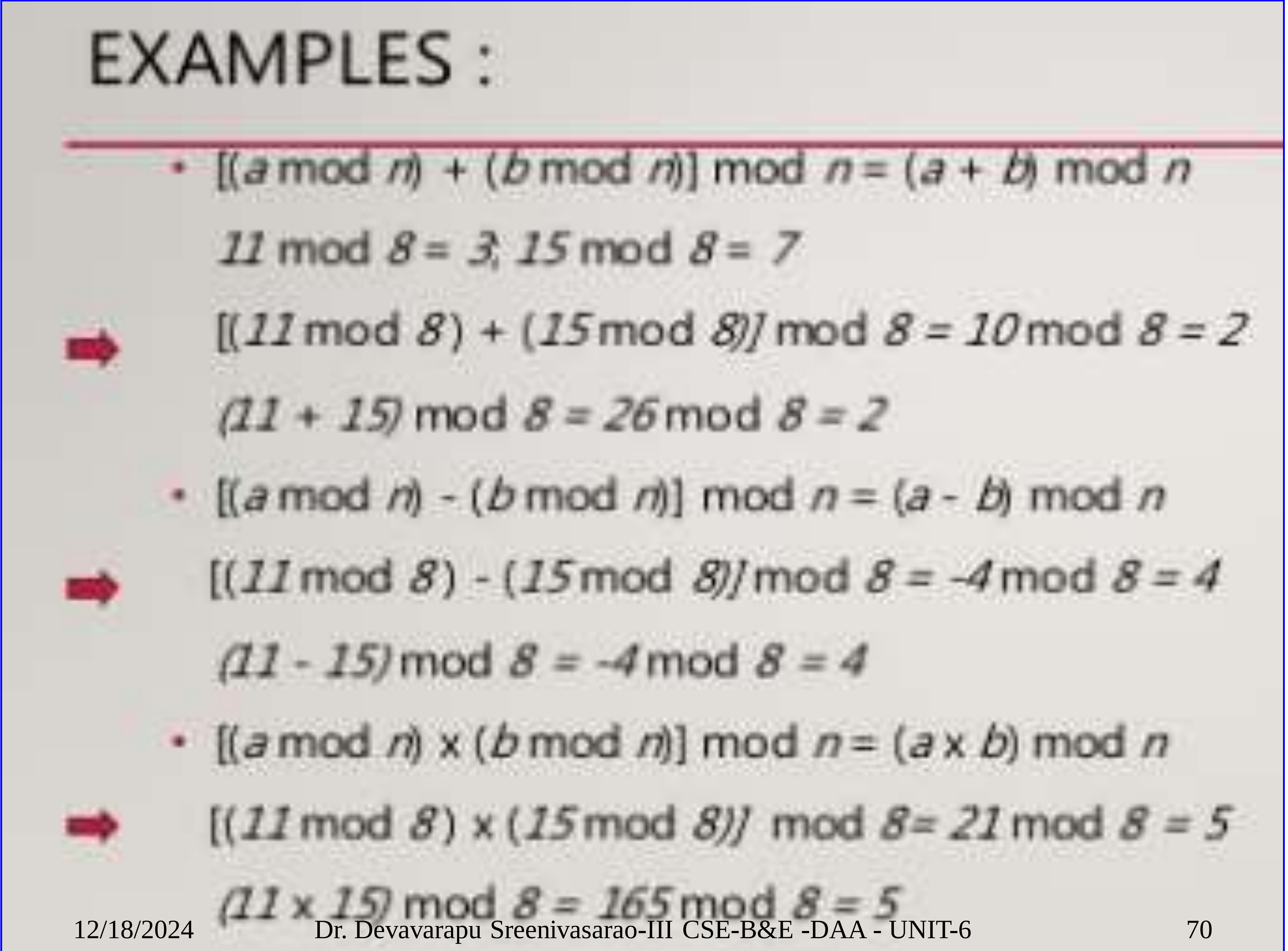
$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

Proof of 1.

Let $(a \bmod n) = Ra$ and $(b \bmod n) = Rb$. Then, we can write

$a = Ra + jn$ for some integer j and $b = Rb + kn$ for some integer k.

$$\begin{aligned}(a + b) \bmod n &= (Ra + jn + Rb + kn) \bmod n \\ &= [Ra + Rb + (k + j)\, n] \bmod n \\ &= (Ra + Rb) \bmod n \\ &= [(a \bmod n) + (b \bmod n)] \bmod n\end{aligned}$$



EXAMPLES :

- $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$

$11 \bmod 8 = 3;\ 15 \bmod 8 = 7$

➡ $[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$

$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$

- $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$

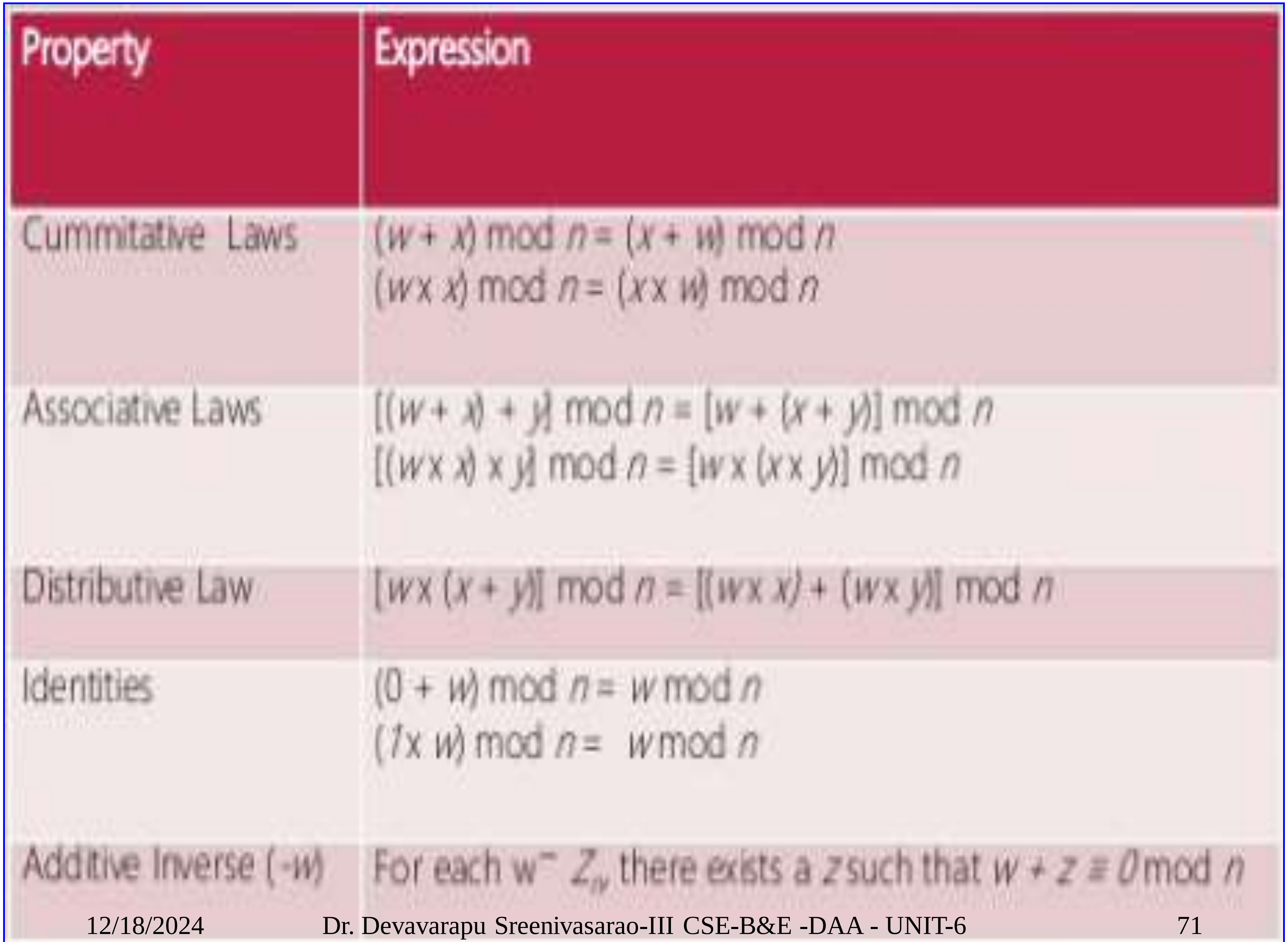
➡ $[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = -4 \bmod 8 = 4$

$(11 - 15) \bmod 8 = -4 \bmod 8 = 4$

- $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

➡ $[(11 \bmod 8) \times (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$

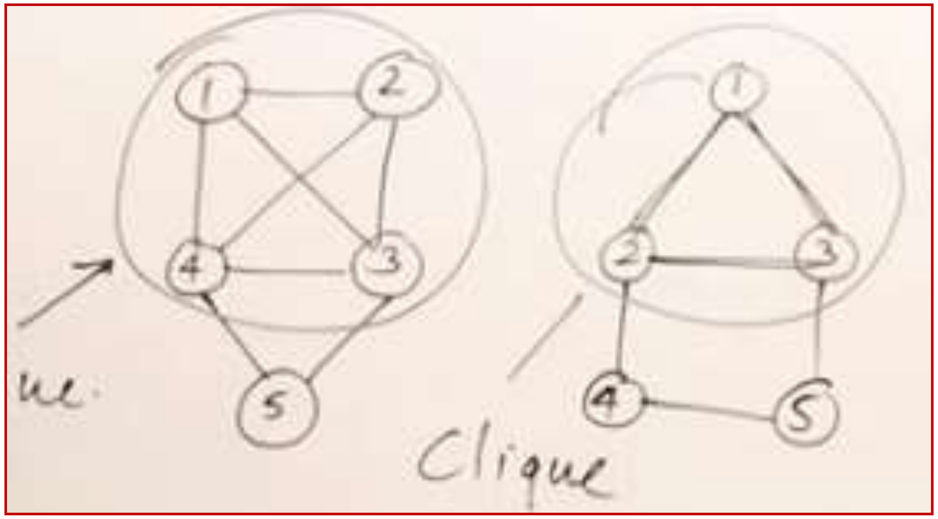
$(11 \times 15) \bmod 8 = 165 \bmod 8 = 5$

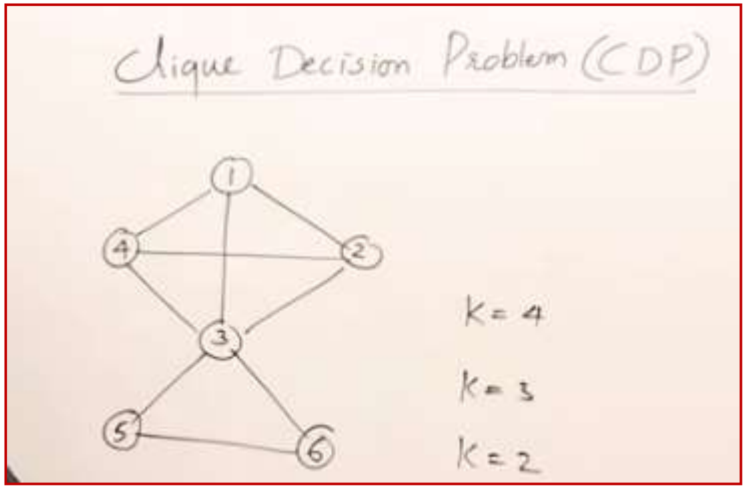


Property	Expression
Cummitative Laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative Laws	$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$ $[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$
Distributive Law	$[w \times (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 \times w) \bmod n = w \bmod n$
Additive Inverse (-w)	For each w⁻ Z_n there exists a z such that $w + z \equiv 0 \bmod n$

Clique Decision Problem:

In a graph, sub graph of a graph is complete graph then that sub graph is called a clique.

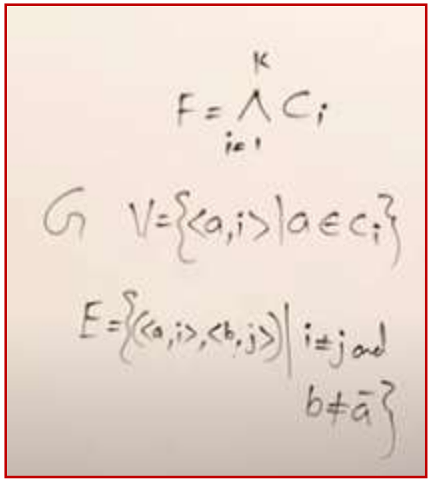




Decision Problem: Find if a graph is having a clique of size k

Optimization Problem: Find maximum clique in a graph

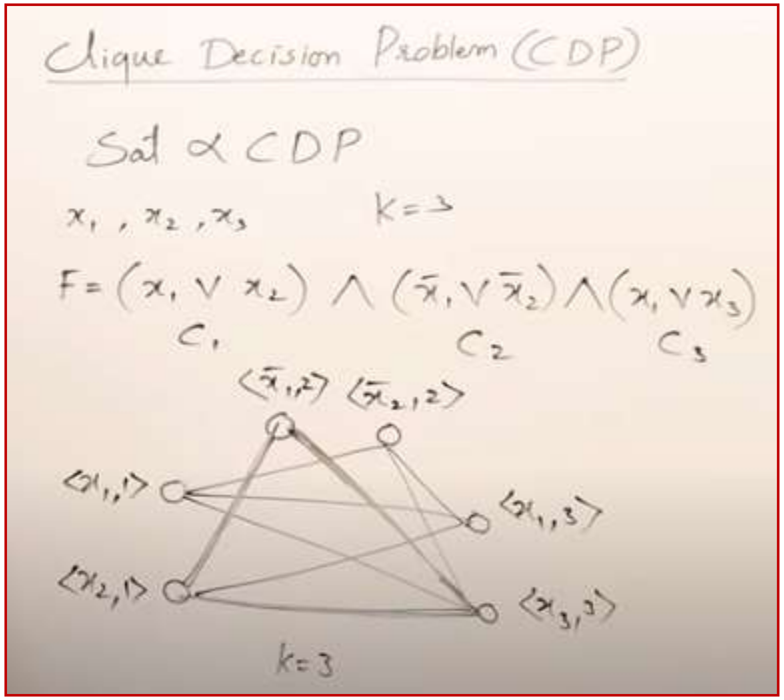
Now P.T CDP is NP-Hard



$$F = \bigwedge_{i=1}^{k} C_i$$

$G \quad V = \{\langle a,i\rangle \mid a \in C_i\}$

$E = \{(\langle a,i\rangle, \langle b,j\rangle) \mid i \neq j \text{ and } b \neq \bar{a}\}$

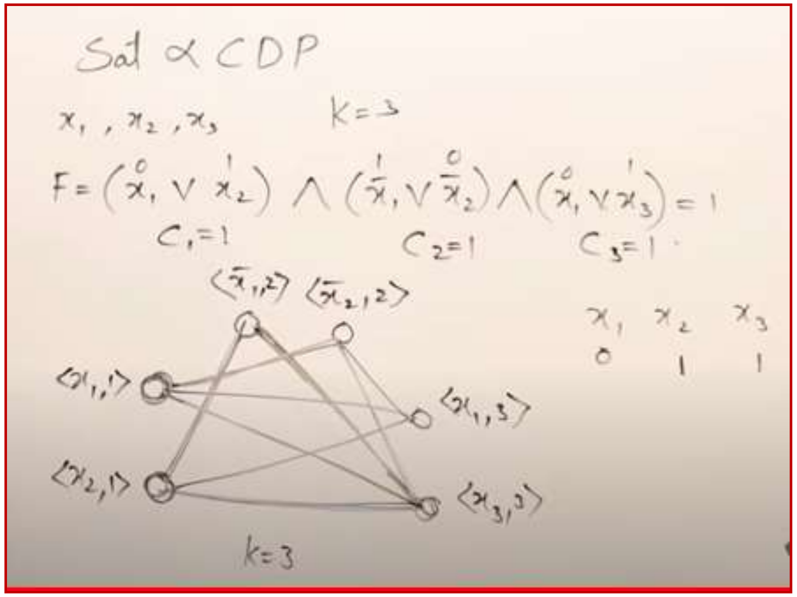


Clique Decision Problem (CDP)

Sat $\propto$ CDP

$x_1, x_2, x_3 \qquad k=3$

$$F = \underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2)}_{C_2} \wedge \underbrace{(x_1 \vee x_3)}_{C_3}$$



Sat $\propto$ CDP

$x_1, x_2, x_3 \qquad k=3$

$$F = (\overset{0}{x_1} \vee \overset{1}{x_2}) \wedge (\overset{1}{\bar{x}_1} \vee \overset{0}{\bar{x}_2}) \wedge (\overset{0}{x_1} \vee \overset{1}{x_3}) = 1$$

$C_1 = 1 \qquad C_2 = 1 \qquad C_3 = 1$

x_1	x_2	x_3
0	1	1