Backtracking:-

General Method:-

In the backtracking method, the desired solution is expressable as an n-tuple $(x_1, x_2, x_3, \ldots x_n)$ where x_i is choosen from finite set S_i. The solution maximises, minimises or satisfies a criterian function $C(x_1, x_2, \ldots x_n)$. The problem in the backtracking can be categorised into 3 categories:

1) Decision problem - find whether there is an feasible solution
2) Optimisation problem - find the best solution
3) Enumeration problem - find the feasible solutions among all the feasible solutions.

In this method for problem 'P', C be the constraint and we need to get the solution by satisfying 'C'.

The basic idea of Backtracking is to build up a vector, one component at a time and to test whether the vector been formed any chances of success. The major advantage of this method is that we can realise the fact that the partial vector does not lead to an optimal solution. In such a situation, that vector can be ignored. Backtracking is a Depth First search with some bounding functions. All solutions using Backtracking should satisfy a complex set of constraints, the constraints may be explicit (or) implicit.

→ Explicit constraints are rules which determine the tuples in the solution space actually satisfy the criterion function

→ Explicit Constraints are the rules which restrict each Vector element to be choosen from the given set.

Applications Of Backtracking:

1) 8-Queens Problem
2) Sum of subsets problem
3) Graph coloring
4) Hamiltonian cycle.
5) knapsack problem

Some terminologies used in BackTracking:

BT Algorithms determine the problem solutions by systematically searching the solution by tree structure.

1) Each node in a tree is called a problem state
2) All paths from the root to other nodes define the State Space of the problem.
3) The solution states are those problem states for which the path from root to problem state 's' defines a tuple in the Solution states
4) In some trees the leaves define the solution states

Answer States:

These are the leaf nodes which corresponds to an element in the set of solutions. These are the solutions which satisfy the implicit constraint.

Live Node:

A node which is been generated and all whose children have not yet been generated is called live node.

E-node: The live node whose children are currently being expanded is called E-node.

Dead node: It is generated where the tree is not expanded further (or) all the children have been generated.

Control abstraction: (algorithm for backtracking using recursive technique)

```
Algorithm Backtracking()
{
for(each a_k that belongs to T[a_1, a_2, ---a_{K-1}]) do
{
if (B_K(a_1, a_2, --- a_K) = true) then
{
if ((a_1, a_2, ---a_K) is a path to answer node) then
print (a[1], a[2], --- a[K]);
    if (k<n) then
     backtrack(k+1);
    }
  }
}
```

Algorithm for Backtracking using Non-recursive technique

```
Algorithm Non-RecBackTracking(n)
   {
   k := 1;
while (k ≠ 0) then
 {
if((any a[k] that belongs to T(a[1], a[2], -- a[k])
   remains untired) AND (B_k(a[1], a[2] -- a[k] = true))
  then
{
if ((a[1], a[2] -- a[k]) is a path to answer node) then
  write(a[1], a[2], --- a[k]);
  if (k<n) then
    {
    return k+1;
    }
  }
}
```

8-Queens Problem:-

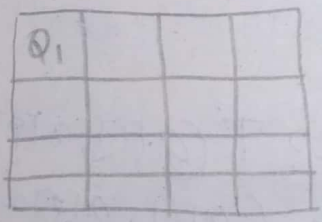
This is a problem based on chess games. With this problem it is stated that arrange the 8 queens on the chess board in such a way that no two queens can attack each other. Here attack means no two queens can be placed in the same row, same column or on the same diagonal. When we place the queens in the chess board we need to check the diagonal conflicts by using the following formula

$$i+l = k+l \text{ (or) } i-l = k-l$$

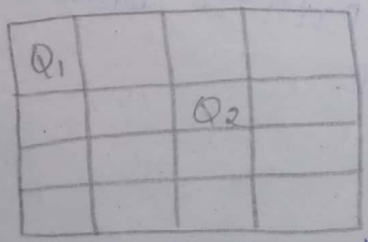
The above condition can also be checked as $|i-j| = |k-l|$

Example: First consider 4X4 chess board to place 4 queens.

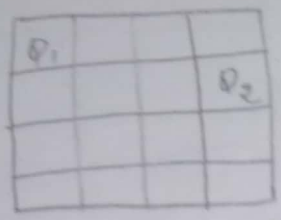
Solution: Now we start with empty chess board, place Queen 1 on the first possible location i.e on the first row and first column.



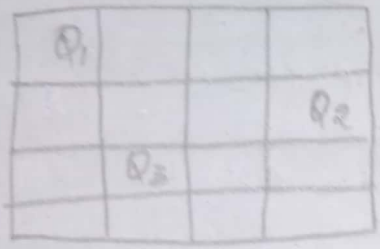
step 2: Then place Q_2 after trying some possible locations at (2,3)



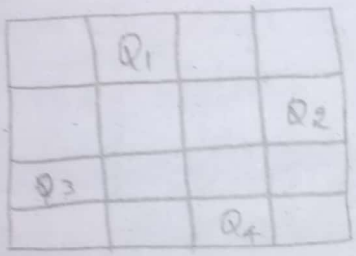
This is the dead end, because third queen cannot be placed in the next column as there is no acceptable position for Q_3. Hence apply the BT algorithm and place the second queen at (2,4) location



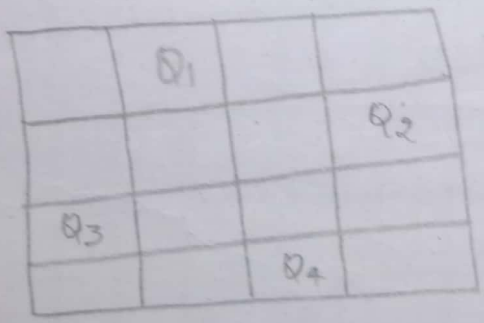
Now place the 3rd queen at (3,2) position but it is again another dead end as next queen '4' cannot be placed at permissible locations.



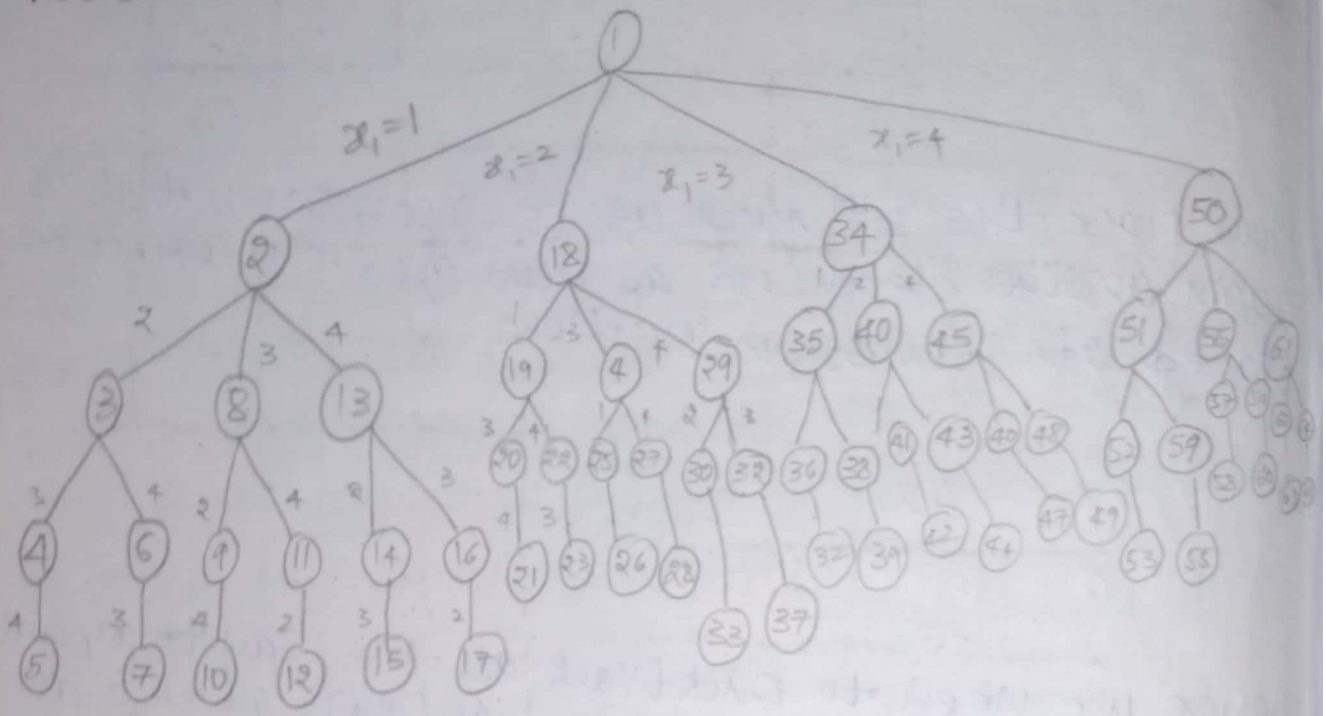
Hence we need to backtrack all the way to Q_1 and move it to (1,2) and place 'Q_2' at (2,4), queen '3' at (3,1) and queen '4' at (4,3)



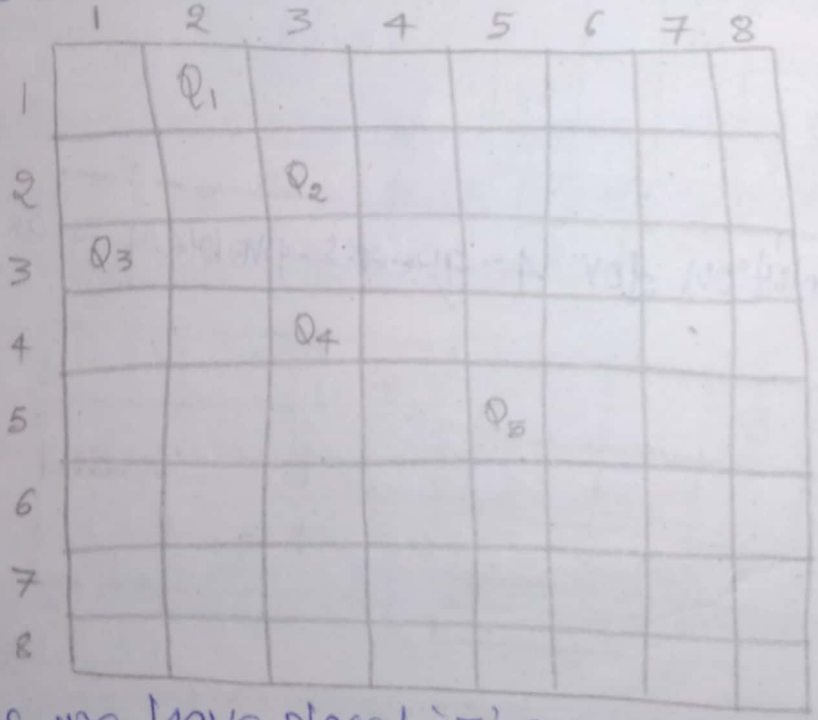
Therefore the solution for 4-queens problem is as follows:



Tree structure:



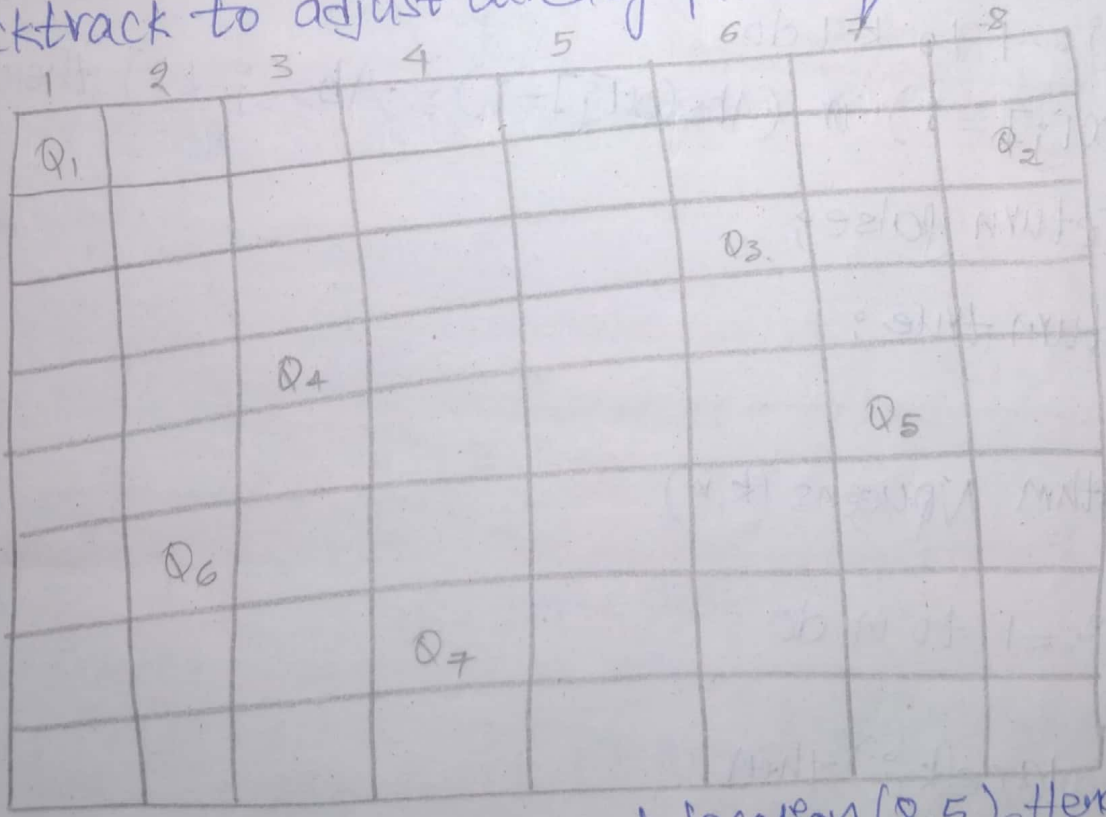
Now we will consider how to place 8-queens on 8x8 chessboard.



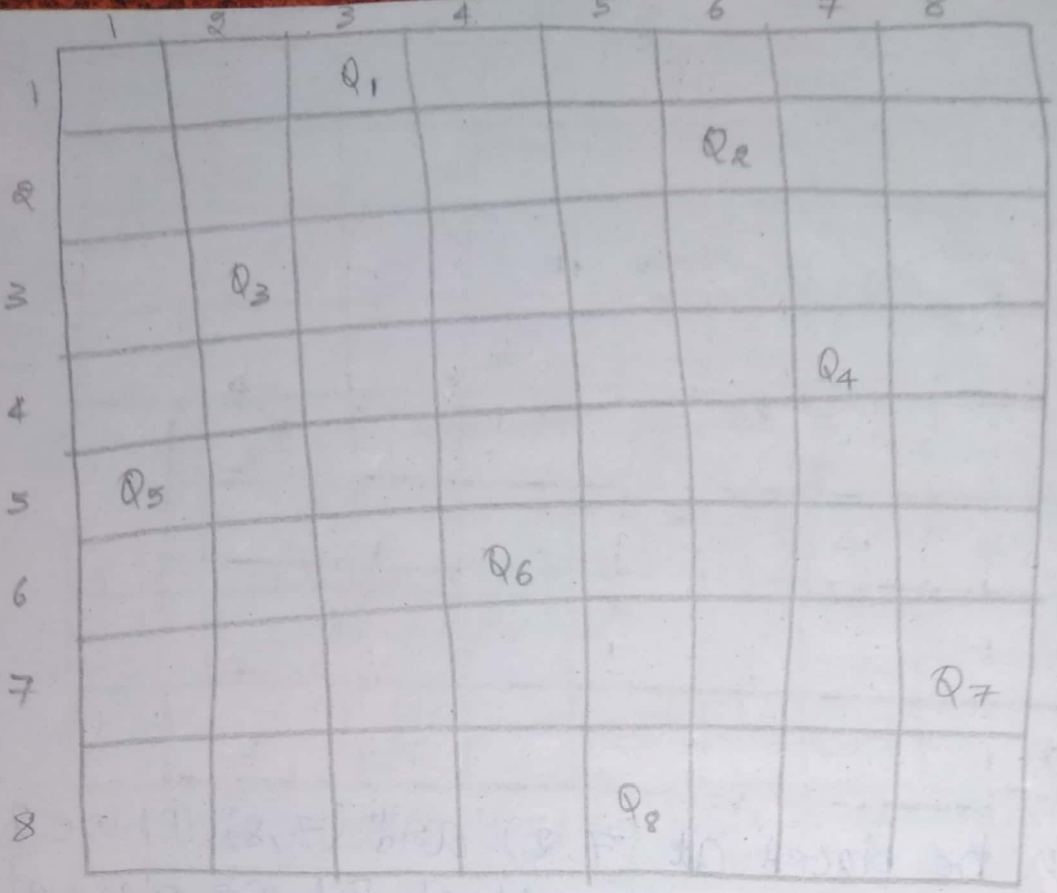
Thus we have placed '5' queens in such a way that no two queens can attack each other. Now we need to place Q_6 at location (6,6) then Q_5 can attack. If Q_6 is placed at (6,7) then Q_1 can attack; if Q_6 is placed at (6,8) then Q_2 can attack.

Therefore the Q_6 we need to backtrack and change the previously placed queen's position.

Q_7 can be placed at (7,2) and (7,8). If we place Q_7 at (7,2) then Q_6 can attack. If we place at (7,8) then Q_4 may attack Q_7. Hence we have to backtrack to adjust already placed queens



But Q_8 cannot be placed at location (8,5). Hence we need to backtrack finally the successful placement of 8-queens is given below




```
Algorithm place(k,i)
{
for j:=1 to k-1 do
if (x[j]=i) or (Abs(x[j]-i)=Abs(j-k)) then
   return false;
  return true;
}
```

(i → column conflict; Abs(x[j]-i)=Abs(j-k) → diagonal conflict)

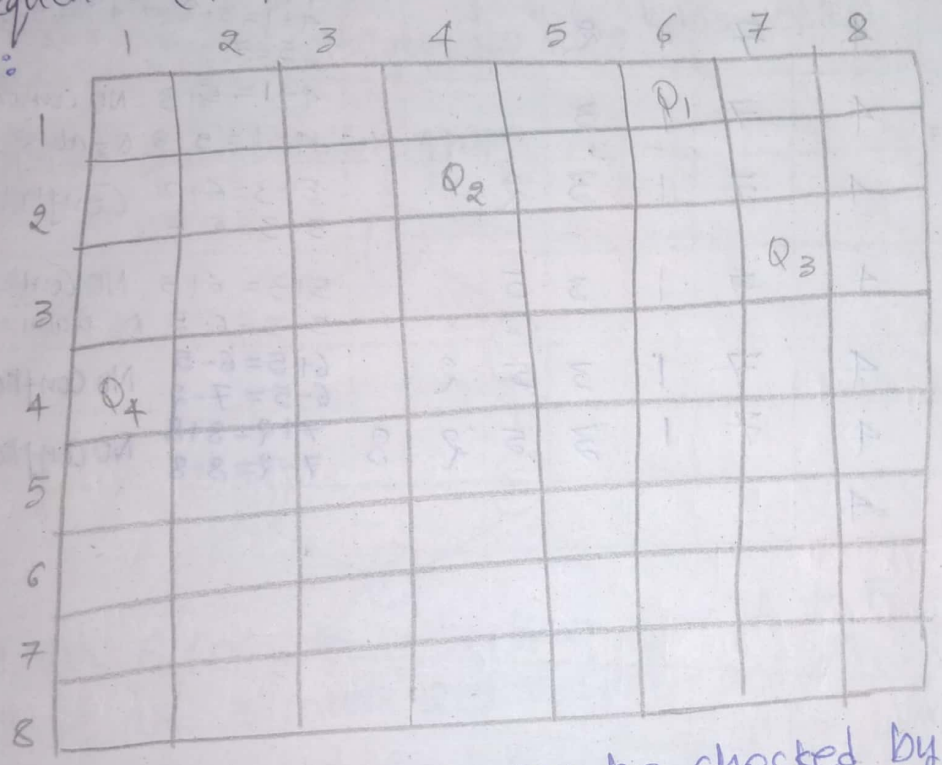
```
Algorithm Nqueens (k,n)
  {
for i:=1 to n do
  {{
 if place(k,i) then
   {
   x[k]=i;
if (k==n) then
 write (x[1:n]);
  else
```

(k → row no; n → total no of queens; place(k,i): row, column)

```
Nqueens(k+1,n);
  }
 }
}
```

1) Solve 8-queens problem with a feasible solution sequence (6,4,7,1). → column numbers by default

sol:



The diagonal conflicts can be checked by the formula

$$i - j = k - l \longrightarrow (1)$$

(or)

$$i + j = k + l \longrightarrow (2)$$

The eq(1) implies $j - l = i - k$

eq(2) $\Rightarrow j - l = k - i$

∴ 2 queens lie on the same diagonal if & only if

$|j - l| = |i - k|$

Let $P_1 = (i, j)$ and $P_2 = (k, l)$ are two positions the P_1 and P_2 are the positions that are on the same diagonal, if $i + j = k + l$ (or) $i - j = k - l$

Sno \ Row	Queens positions 1	2	3	4	5	6	7	8	Actions
1	6	4	7	1					
2	6	4	7	1	2				$i+j=k+l$ or $i-j=k-l$ Conflict $4+1=5+2$ or $4-1=5-2$ $5=7$ $3=3$
3	6	4	7	1	3				$4+1=5+3$ NO Conflict $4-1=5-3$ Q_5 at 3rd column
4	6	4	7	1	3	2			$5+3=6+2$ Conflict $5-3=6-2$
5	6	4	7	1	3	5			$5+3=6+5$ NO Conflict occur $5-3=6-5$ Q_6 placed at 6th col
6	6	4	7	1	3	5	2		$6+5=6-5$ No Conflict $6-5=7-2$
7	6	4	7	1	3	5	2	8	$7+2=8+8$ NO Conflict $7-2=8-8$
8	6	4							

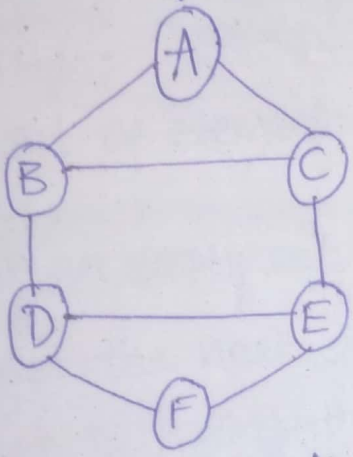
Graph Colouring:

Graph colouring is a problem of colouring each vertex in a graph in such a way that no two adjacent vertices have same colour and m colours are used to colour the vertices. Therefore this problem is also called as m colouring problem.

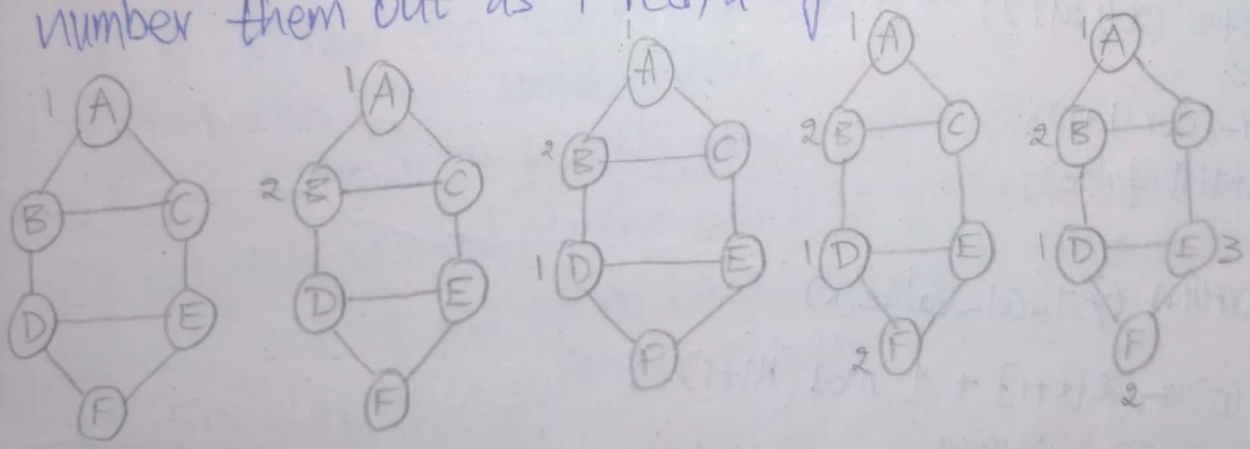
If the degree of given graph is 'd' then we can colour it with 'd+1' colours. The least number of colours required to colour the graph is called chromatic number.

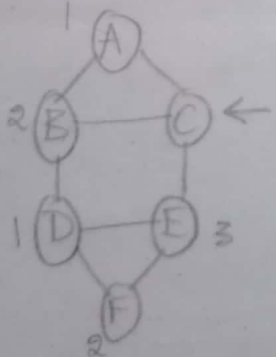
Eg: Hence we require 3 colours to colour the graph then the chromatic number of given graph is 3. We use backtracking technique to solve this problem.

Consider the following graph



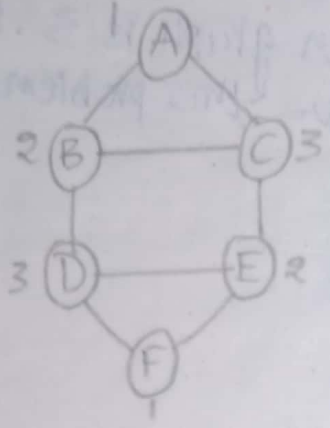
1) Graph G consists of vertices from A to F
2) There are 3 colours used red, green and blue. We will number them out as 1-red, 2- green, 3-blue.





We cannot assign the colour either 1 or 2 or 3 for 'c' because the graph colouring problem constraint is that no two adjacent vertices have the same colour.

If we assign colour 1 to C then adjacent Vertex of C i.e A is having colour 1. Similarly B is having 2 and E is having 3.
Therefore we need to backtracking.



Algorithm:

```
Algorithm Gr.Color(k)
{
 repeat
{
  Gen-Col-Value (k);
 if (x[k]=0) then
 return;
if (k=n) then
write (x[1:n]);
else
 Gr-Color(k+1);
} until (false);
}
Algorithm Gen_col_value(k)
{
 x(E) ← (x[k+1] + 1 mod (m+1));
  if (x[k] =0) then
    return;
for j ← 1 to n do
{
if (G[k,j] != 0) AND (x[k] = (x[j])) then
    break;
}
if (j=n+1) then
  {
    return
} until (false);
```

}

The graph Color takes computing time of $\sum_{i=0}^{n-1} m^i$

Hence total computing time for this algorithm is $O(nm^n)$

Sum of subsets problem:

Let $S=\{s_1, s_2, \ldots s_n\}$ be a set of n +ve integers they we have to find a subset whose sum is equal to given positive integer 'd'. It is always convenient to sort the sets elements in ascending order i.e., $s_1 \leq s_2 \leq \ldots \leq s_n$

General Algorithm:

Let s be a set of elements and d is expected sum of subsets then

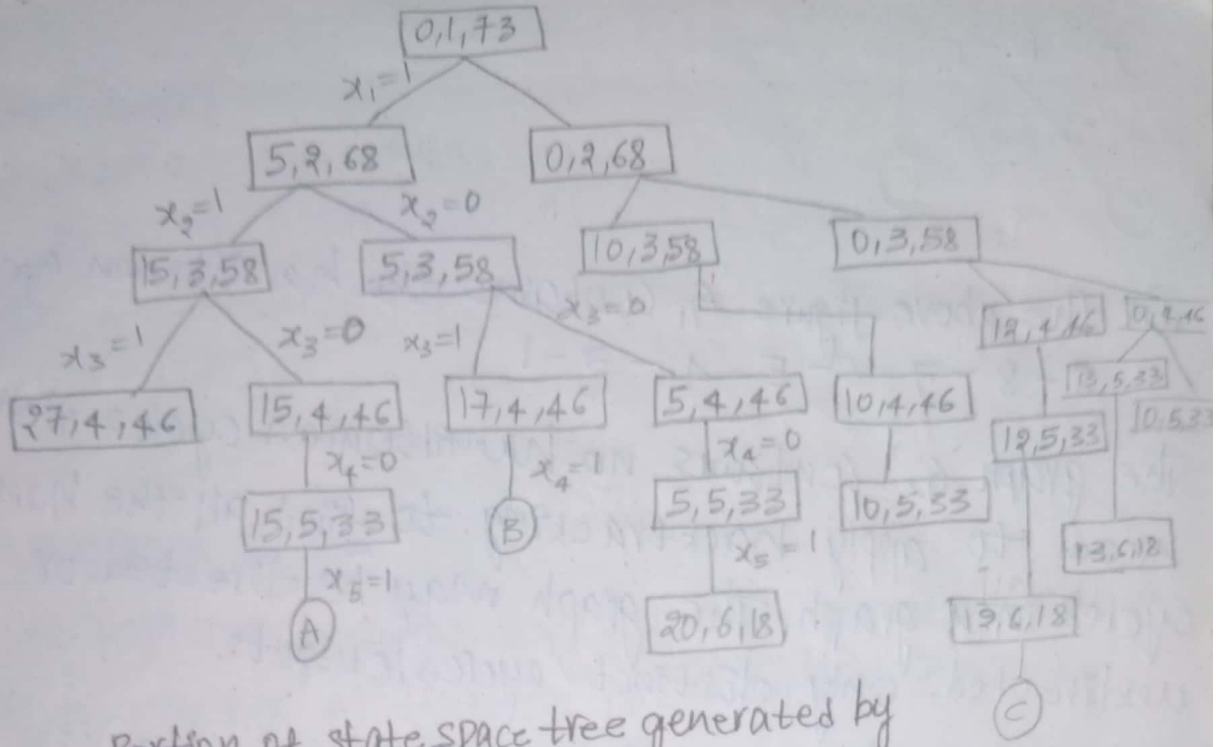
1) starts with an empty set
2) Add to subset the next element from the list
3) If the subset is having the sum = d then stop with that subset as solution
4) If the subset is not feasible (or) if we have reached the end of the set then backtrack through the subset until we find the most suitable value.
5) If the subset is feasible then repeat step '2'
6) If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop the algorithm without solution.

Eg: Consider set $S=\{5, 10, 12, 13, 15, 18\}$ and $d=30$
solve it for obtaining sum of subsets.

Sol: Initially Subset = { }	Sum=0 · 0 ≤ 30 ✓	Then add next eleme
5	Sum=5 5 ≤ 30 ✓	Then add next elem
5,10	Sum=15 15 ≤ 30 ✓	Then add next eleme
5,10,12	Sum=27 27 ≤ 30 ✓	Then add next ele
5,10,12,13	Sum=40 40 ≤ 30 ✗	Perform backtrack
5,10,12,15	Sum=42 42 ≤ 30 ✗	perform backtrack
5,10,12,18	Sum=45 45 ≤ 30 ✗	perform backtrack as sum exceeds 'd' value
5,10,13	Sum=28 28 ≤ 30 ✓	Then add next eleme
5,10,13,15	Sum=43 43 ≤ 30 ✗	Perform backtrack
5,10,13,18	Sum=46 46 ≤ 30 ✗	perform backtrack
5,10,15	Sum=30 30 ≤ 30	

∴ Subset = {5,10,15}

5,10,~~12~~,~~13~~,15,18



Portion of state space tree generated by SumOfSub

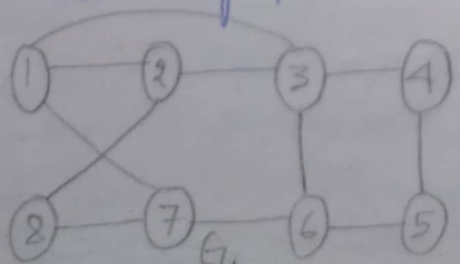
Algorithm:

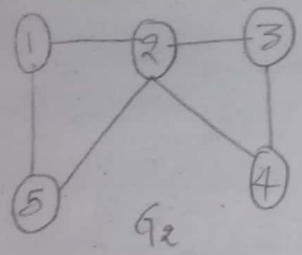
```
Algorithm SumOfSub(s,k,r)
{
  x[k]:=1;
if(s+w[k]=m) then write(x[1:k]);
  else if(s+w[k]+w[k+1]≤m)
  if((s+r-w[k]≥m) and (s+w[k+1]≤m)) then
  {
    x[k]:=0;
    SumOfSub(s,k+1,r-w[k]);
  }
}
```

Hamiltonian Cycle:-

Let $G=(V,E)$ be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along 'n' edges of graph G that visits every vertex once and returns to its starting position

Eg:





G_2

In the above figure 'G_1' contains the hamiltonian cycle
1-2-8-7-6-5-4-3-1

The graph 'G_2' contains no hamiltonian cycles. Here we need to apply backtracking to find all the hamiltonian cycles in a graph. The graph may be directed or undirected. only distinct cycles/outputs.

The backtracking solution vector $(x_1, \ldots x_n)$ is defined so that x_i represents the i^{th} visited vertex of proposed cycle.

Algorithm: To calculate next vertex

```
Algorithm NextValue (k)
{
 repeat
{
x[k]:=(x[k]+1) mod (n+1); //next vertex
If (x[k]=0) then return;
If (G[x[k-1], x[k]]≠0) then
{
for j:=1 to k-1 do if (x[j]=x[k]) then break;
 if (j=k) then
 if ((k<n) or((k=n) and G[x[n], x[1]]≠0))
 then return;
}
until(false);
 }
}
Algorithm Hamiltonian (k)
{
```

```
repeat
{
  nextvalue(k)
  if (x[k] = 0) then return;
    if (k=n) then write(x[1:n]);
     else Hamiltonian(k+1)
  } until (false);
}
```

0/1 knapsack problem:-

Given n positive weights w_i, n positive profits p_i and a positive number m that is the knapsack capacity, the problem calls for choosing a subset of weights such that:

$\sum_{1 \le i < n} w_i x_i \le m$ and $\sum_{1 \le i \le n} p_i x_i$ is maximized.

The x_i's constitute a zero-one-valued vector. The solution space for this problem consists of 2^n distinct ways to assign zero or one values to the x_i's

Bounding Functions are needed to kill some live nodes without expanding them. A good bounding function for this problem is obtained by using an upperbound on the value of the best feasible solution obtainable by expanding the given live node and any of its descendants. If this upperbound is not higher than the value of the best solution determined so far, than that live node can be killed.

We continue the discussion using the fixed tuple size formulation. If at node z the values of x_i, $1 \le i \le k$, have already been determined, then an upper bound for z can be obtained by relaxing the requirement $x_i = 0$ or 1.

```
Algorithm Bound (cp, cw, k)
{
b := cp; c := cw;
for i := k+1 to n do
  {
    c := c + w[i];
    if (c < m) then
      b := b + p[i];
    else
      return b + (1 - (c - m) / w[i]) * P[i];
  }
return b;
}
Algorithm Bknap(k, cp, cw)
{
// Generate left child
if (cw + w[k] ≤ m) then
{
y[k] := 1;
if (k < n) then Bknap (k+1, cp + p[k], cw + w[k]);
if ((cp + p[k] > fp) and (k = n)) then
{
  fp := cp + p[k]; fw := cw + w[k];
  for j := 1 to k do
  x[j] := y[j];
}
// Generate right child
if (Bound (cp, cw, k) ≥ fp) then
{ y[k] := 0;
if (k < n) then Bknap (k+1, cp, cw);
if ((cp > fp) and (k = n)) then
{
```

```
fp := cp; fw := cw;
for j := 1 to k do
x[j] = y[j];
}
}
```

Unit-4

Branch and Bound

General Method:

In Branch and Bound, a state space tree is generated and all the children of E-nodes are generated before any other node can be come a live node.

For exploring new nodes, we use either BFS or DFS technique. If we use BFS technique, we do the state space search and it is called FIFO's search. This is because the list of live nodes use FIFO list (queue). On the other hand, these D-search like state space search will be called LIFO search because the list of live nodes is LIFO list (stack)

In this method a state space tree of possible solutions is generated then partitioning (called as branching) is done at each node of the tree. We compute lower bound and upper bound at each node. This computation leads to the selection of answer node.

Bounding functions are used to avoid the generation of subtrees that do not contain answer node.

Control Abstraction :-

```
Algorithm BranchBound()
{
  E ← new(node);
  while (true)
  {
  If (E is a final leaf) then
  {
  write (path from E to the root);
```

```
    return;
    }
Expand(E);
if (H is empty) then
{
write("there is no solution");
return;
}
E ← delete_top(H);
  }
}
```

Backtracking
1) We generate a state space tree
2) Based on constraints
3) We produce a complete tree

Branch & bound
1) We generate a state space tree
2) Based on bounding function
3) We produce a subtree to find answer node

```
Algorithm Expand(E)
{
generate all the children of E;
Compute the approximate cost value of each node;
Insert each child into the heap H;
}
```

live node

upper bound & lower bound

Applications:

1) Travelling sales person problem
2) 0/1 knapsack problem

Travelling sales Person Problem:-

In the travelling sales person problem we start at a particular ~~point~~ node and visit all the ~~points~~ nodes exactly once and come back to the initial node (or) point with minimum cost.

The time complexity of the problem using LSDT is $O(n^2 \times 2^n)$ which is same as the dynamic programming.

Procedure:

step 1: Find out the reduced cost matrix from a given cost matrix. This can be obtained as follows

Find out

i) row reduction - take the minimum element from the first row and subract that element from the first row elements including that element, next take the minimum element from 2nd row. subract that element from 2nd row. simply apply the same procedure for all the rows, after applying row reduction we will get one resultant matrix, for that matrix apply ii) Column reduction

ii) Column reduction:

Take the minimum element from first column subract that element from first column (including that element). Similarly apply the same process for remaining columns.

Now we need to find row wise reduction sum and column wise reduction sum

row wise reduction sum - the sum of the elements which are subracted from each row

Column wise reduction sum - the sum of the elements which are subracted from column

Now we need calculate Cumulative reduction 'K'

K = row wise reduction sum + Column wise reduction sum

step 2: For starting node, we will take Cumulative reduction as lower bound and '∞' as upper bound.

2. a) If path (i,j) is considered then change all entries in row 'i' and column 'j' of 'A' to '∞'

2. b) set A[j,i] to '∞'

2. c) Apply row reduction and Column reduction except for

rows and columns containing infinity. R is calculated in step 'c'.

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

$\hat{c}(s)$ = ranking function (or) cost function

$\hat{c}(R)$ = indicates the lower bound of i^{th} node in (i,j) path

Repeat step 2 until all nodes are visited.

Example:

1) Given cost matrix $A =$ (minimum values of rows)

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

Find out the reduced cost matrix

This can be applie derived by applying the row reduction and column reduction.

Consider the row reduction method, then the ~~given matr~~ resultant matrix is

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

(20-10, 30-10, 16-2, 4-2, 2-2, 15-2, 10-10, 11-10)

Row reduction sum = 10 + 2 + 2 + 3 + 4 (sum of non-value of row)

= 21

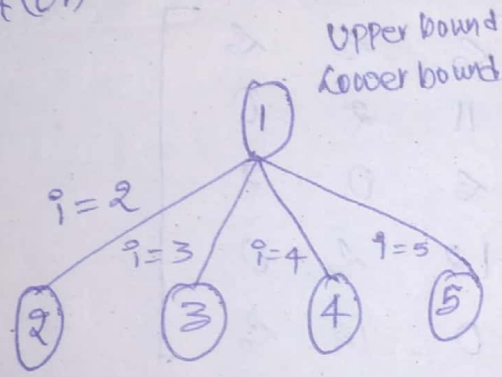
Apply the column reduction method, then the resultant matrix is shown below

$$A=\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$ → base matrix

Column Reduction sum = 1 + 0 + 3 + 0 + 0 = 4

Cumulative Reduction (r) = 21 + 4 = 25

starting vertex is '1'. So from vertex '1' he can travel visit 2, 3, 4 (or) 5th node. This can be shown below.



The variable 'i' indicates the next node which we will visit. Apply step 2 for reduced cost matrix 'A'.

Consider the path (1, 2)

2. a) Change all entries of 1st row and 2nd column of A to infinity.

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

2. b) set A[2,1] to infinity then the resultant matrix is

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

2. c) Apply row and column reduction excluding the rows & columns containing all entries as infinity.

row reduction method:

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

Column reduction method:

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

r = 0+0 (row reduction + Column reduction)

= 0

$\hat{C}(S) = \hat{C}(R) + A(1,2) + r$

$= 25 + 10 + 0$

$= 35$

Similarly apply the same procedure for remaining paths (1,3), (1,4), (1,5)

Consider the path (1,3)

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

Row reduction method:

~~∞ 10~~

~~A=~~

2. a) change all entries of row '1' and column '3' to '∞'

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

2. b) set A[3,1] to ∞

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

11 0 0 0,

Column Reduction method:

$$A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

see in box matrix

$$\hat{C}(S) = \hat{C}(R) + A(1,3) + r$$

$$= 25 + 17 + 11$$

$$= 53$$

Consider the path (1,4)

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

2.a) Change all the entries of row '1' and column '4' to ∞

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

2.b) Set [4,1] to '∞'

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

2.c) Apply row reduction and column reduction excluding the rows & columns containing all entries as infinity.

After applying row reduction and column reduction the matrix remains the same

r = row reduction sum + column reduction sum

r = 0 + 0 = 0

$\hat{C}(S) = \hat{C}(R) + A[1,4] + r$

$= 25 + 0 + 0$

$= 25$

Consider the path (1,5)

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

2.a) Change all the entries of row '1' and column '5' to '∞'

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

2.b) set [5,1] to '∞'

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

2.c) Apply row reduction and column reduction excluding the rows & colums containing all entries as infinity

row reduction sum = 2+0+3+0

= 5

Column reduction sum = 0+0+0+0

= 0

Cumulative sum (r) = 5+0

= 5

$\hat{C}(S) = \hat{C}(R) + A[1,5] + r$

= 25+1+5

= 31

Conclusion:

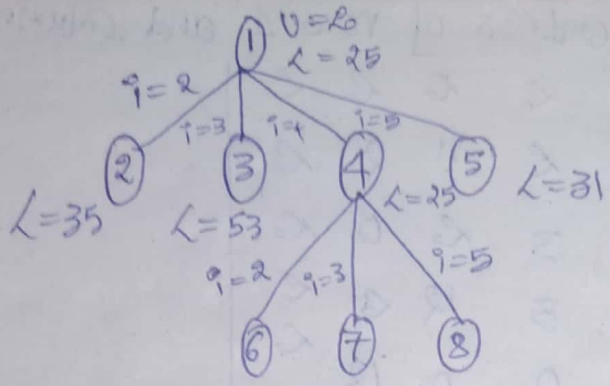
(1,2) = 35

(1,3) = 53

(1,4) = 25

(1,5) = 31

The minimum path is (1,4). Hence we traverse from node '1' to '4'



We Consider the path (1,4) which is having the minimum cost function. Now we need to Consider the matrix obtained for the path (1,4) as reduced cost matrix.

Consider the (4,2):

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

2.a) Change all the entries of row '4' and column '2' to '∞'

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

[2,1]

2.b) set [4,2] to ∞

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

2. c) By applying the row reduction and column reduction the matrix remains the same.

Row reduction sum = 0+0+0
= 0

Column reduction sum = 0+0+0
= 0

r = row reduction sum + column reduction sum
= 0+0
r = 0

$\hat{C}(S) = \hat{C}(R) + A[4,2] + r$ (→ lower bound of root node ('4'))
= 25+3+0
= 28

Consider the path (4,3):

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

2.a) change all the entries of row '4' and column '3' to '∞'

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

2.b) set [3,1] to '∞'

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

11 0 0

2.c) By applying the row reduction and column reduction the matrix remains the same

row reduction sum = 0+2+0
= 2

Column reduction sum = 0+0+0
= 0

Cumulative sum (r) = 2+0
= 2

$\hat{C}(S) = \hat{C}(R) + A[4,3] + r$
= 25 + 12 + 0 = 37
= 37 + 11 = 48

Consider the path [4,5]:

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

2.a) Change all the entries of row '4' and column '5' to ∞

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

[5,1]

2.b) set [4,5] to ∞

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

By applying the row reduction the matrix changes to

$$A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

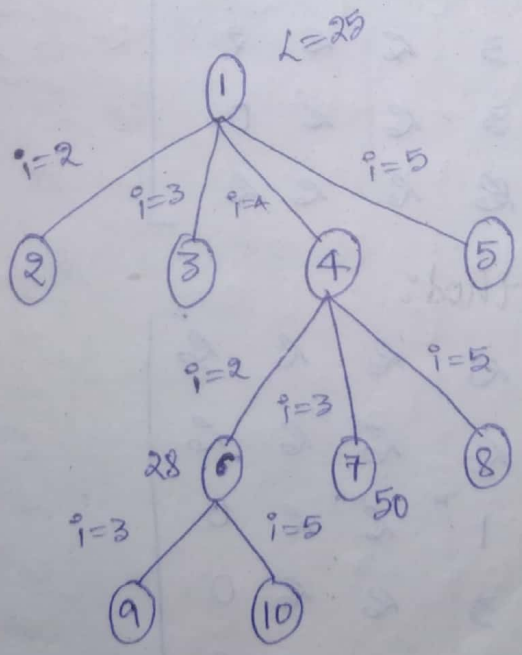
By applying column reduction the matrix remains the same

row reduction sum = 11 + 0 + 0
= 11

Column reduction sum = 0 + 0 + 0
= 0

Cumulative sum (r) = 11 + 0
= 11

$\hat{C}(S) = \hat{C}(R) + A[4,5] + r$
= 25 + 0 + 11
= 36



Consider the path (2,3):

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

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

2. a) Change all the entries of row 2 and column 3 to ∞

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

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

2. b) set [3,1] to ∞

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

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

Row Reduction method:

$$A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

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

11 0 0 0

Column reduction method:

$$A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

Row reduction sum = 2+0+0 = 2

= 2+11 = 13

Column reduction sum = 11+0+0 = 11

0+0

Cumulative sum(r) = 2+11 = 13

13+0 = 13

$\hat{C}(s) = \hat{C}(R) + A[2,3] + r$

$= 28 + 11 + 13$

$= 52$

Consider the path (2,5):

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

2.a) Change all the entries of row '2' and column '5' to ∞

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

2.b) set [5,1] to ∞

$$A = \begin{bmatrix} c & c & c & c & c \\ c & c & c & c & c \\ 0 & B & c & c & c \\ c & B & B & c & c \\ c & 0 & 0 & c & c \end{bmatrix}$$ 0 0 0

Row Reduction method:

$$A = \begin{bmatrix} c & c & c & c & c \\ c & c & c & c & c \\ 0 & B & c & c & c \\ c & B & B & c & c \\ c & 0 & 0 & c & c \end{bmatrix}$$ 0 0 0

Column Reduction method:

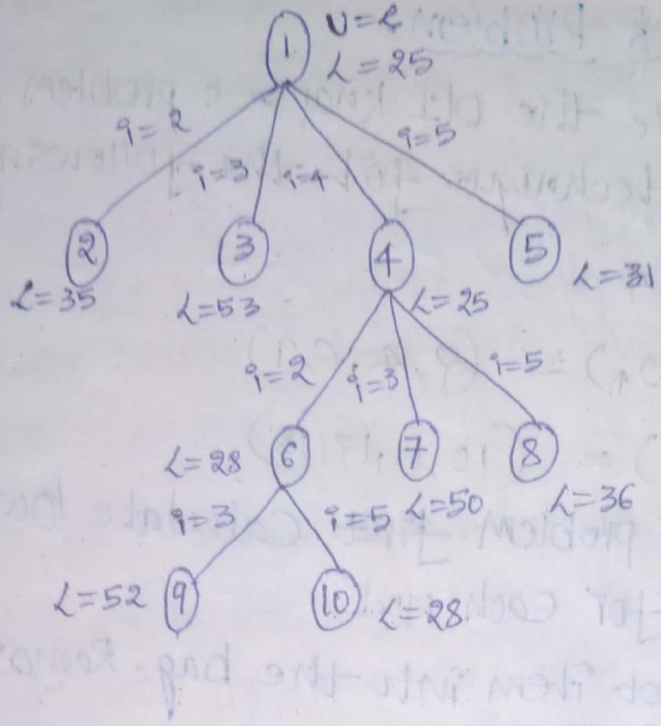
$$A = \begin{bmatrix} c & c & c & c & c \\ c & c & c & c & c \\ 0 & 3 & c & c & c \\ c & 0 & 9 & c & c \\ c & 0 & 0 & c & c \end{bmatrix}$$

Column reduction sum = 0+0+0
= 0

Row reduction sum = 0+B
= B

Cumulative sum(r) = 0+B
= B

$\hat{C}(S) = \hat{C}(R) + A[2,5] + r$
= 28 + B + B
= 28



We Consider the path (2,3) which is having the minimum cost function

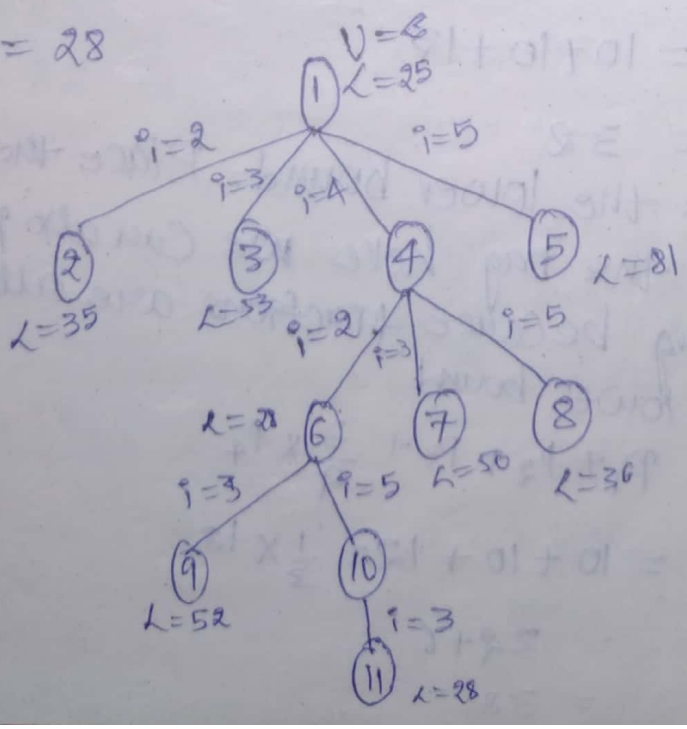
Consider the path (3,5) (5,3)

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

$$\hat{c}(S) = \hat{c}(R) + A[5,3] + r$$

$$= 28$$

$$= 28$$



0/1 knapsack Problem:

Example: Solve the 0/1 knapsack problem using branch and bound technique for the following values

$M = 15, n = 4$

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

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

Sol: In this problem first calculate lower bound and upper bound for each node.

Place the first item into the bag. Remaining weight of the bag $15 - 12 = 13$

Place next item w_2 in the bag then the remaining bag weight is $13 - 4 = 9$

Place the 3rd item in the bag and remaining weight of bag is $9 - 6 = 3$

No Fractions are allowed in calculating the upper bound so 4th item can't be placed in the bag because 4th item weight is 9 and remaining bag capacity is 3

Upper bound $= P_1 + P_2 + P_3$

$= 10 + 10 + 12$

$= 32$

Now Calculate the lower bound. Place the 2nd item & 3rd item in the bag here we can also place 4th item in the bag because fractions are allowed in Calculation of lower bound

Lower bound $= P_1 + P_2 + P_3 + \frac{3}{9} \times P_4$

$= 10 + 10 + 12 + \frac{1}{3} \times 18$

$= 32 + 6$

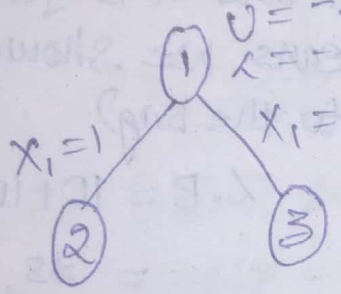
$= 38$

knapsack problem is a maximisation problem but Branch & Bound technique is applicable only for minimisation problems.

Therefore to convert the maximisation problem into minimisation problem we need to convert +ve upper bound and lower bound values into -ve values.

∴ U.B = -32 L.B = -38

We choose the path which has minimum difference b/n U.B and L.B. If the difference is equal then we choose the path by comparing the U.B's and discard the node with maximum U.B



Now calculate U.B and L.B for nodes 2, 3

For Node 2: $x_1 = 1$ (means we should place the first item in the bag)

U.B = 10 + 10 + 12

U.B = 32

U.B = -32

L.B = $10 + 10 + 12 + \frac{1}{3} \times 18$

L.B = 32 + 6

L.B = 38

L.B = -38

For node 3: $x_1 = 0$ (means we should not place the 1st item in the bag)

U.B = 10 + 12

U.B = 22

U.B = -22

L.B = $10 + 12 + \frac{5}{9} \times 18$

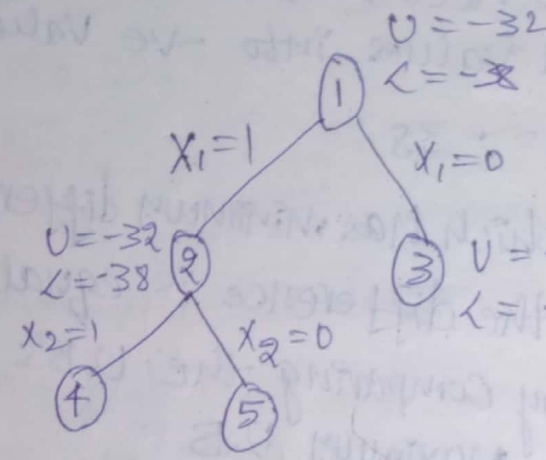
= 22 + 10

= -32

Now we need to calculate the difference b/n U.B & ∠.B for node 2 and node 3

for node 2: U.B − ∠.B = −32 + 38 = +6

For node 3: U.B − ∠.B = −22 + 32 = 10



Now Calculate the U.B and ∠.B for nodes 4,5

For node 4: $x_2 = 1$ (means we should place 2nd item into the bag)

U.B = 10 + 10 + 12 *

U.B = 32

U.B = −32

∠.B = 10 + 10 + 12 + $\frac{1}{3}$ × 18

= 38

∠.B = −38

For node 5: $x_2 = 0$ (means we should not place 2nd item)

U.B = 10 + 12 *

U.B = 22

U.B = −22

∠.B = 10 + 12 + $\frac{7}{9}$ × 18

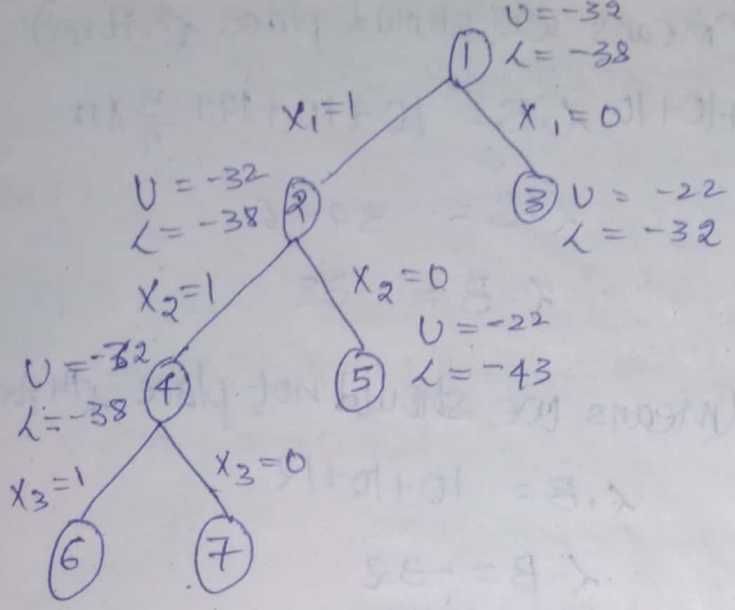
∠.B = 22 + 21

∠.B = 43

∠.B = −43

For node 4: U.B − ∠.B = −32 + 38 = 6

For node 5: U.B − ∠.B = −22 + 43 = 21



For Node 6: $X_3 = 1$ (means we should place 3rd item)

$U.B = 10 + 10 + 12$

$U.B = -32$

$L.B = 10 + 10 + 12 + \frac{3}{9} \times 18$

$= 32 + 6$

$L.B = -38$

For Node 7: $X_3 = 0$ (means we should not place 3rd item)

$U.B = 10 + 10 + 18$

$U.B = -38$

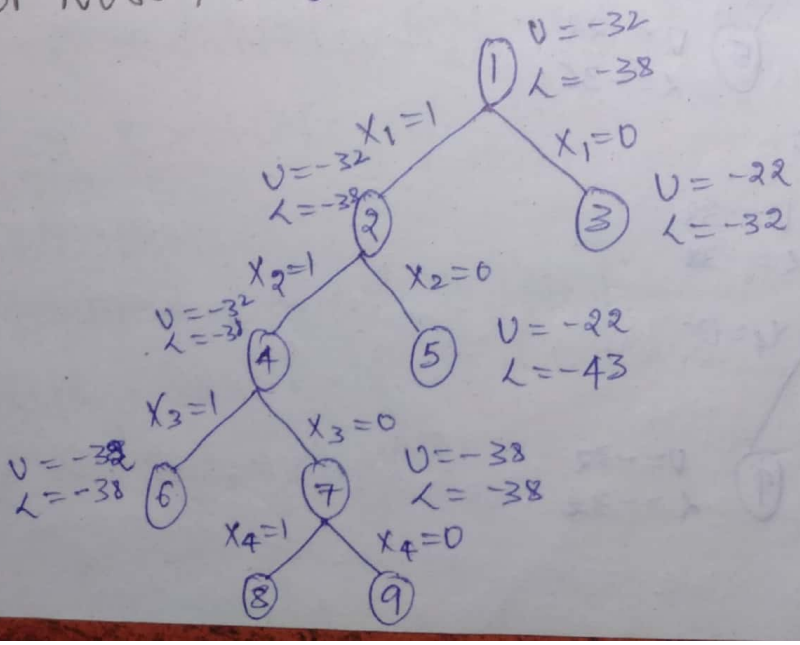
$L.B = 10 + 10 + 18$

$L.B = -38$

Now we calculate the difference b/n U.B and L.B of Node 6 & Node 7

For Node 6: $U.B - L.B = -32 + 38 = 6$

For Node 7: $U.B - L.B = -38 + 38 = 0$



For Node 8: $x_4 = 1$ (means we should place 4th item)

U.B = 18+10+10 $\quad$ ℓ.B = $10+10+12+\frac{3}{9}\times 18$

U.B = -38 $\quad$ ℓ.B = 32+6

ℓ.B = -38

For Node 9: $x_4 = 0$ (means we should not place 4th item)

U.B = 10+10+12 $\quad$ ℓ.B = 10+10+12

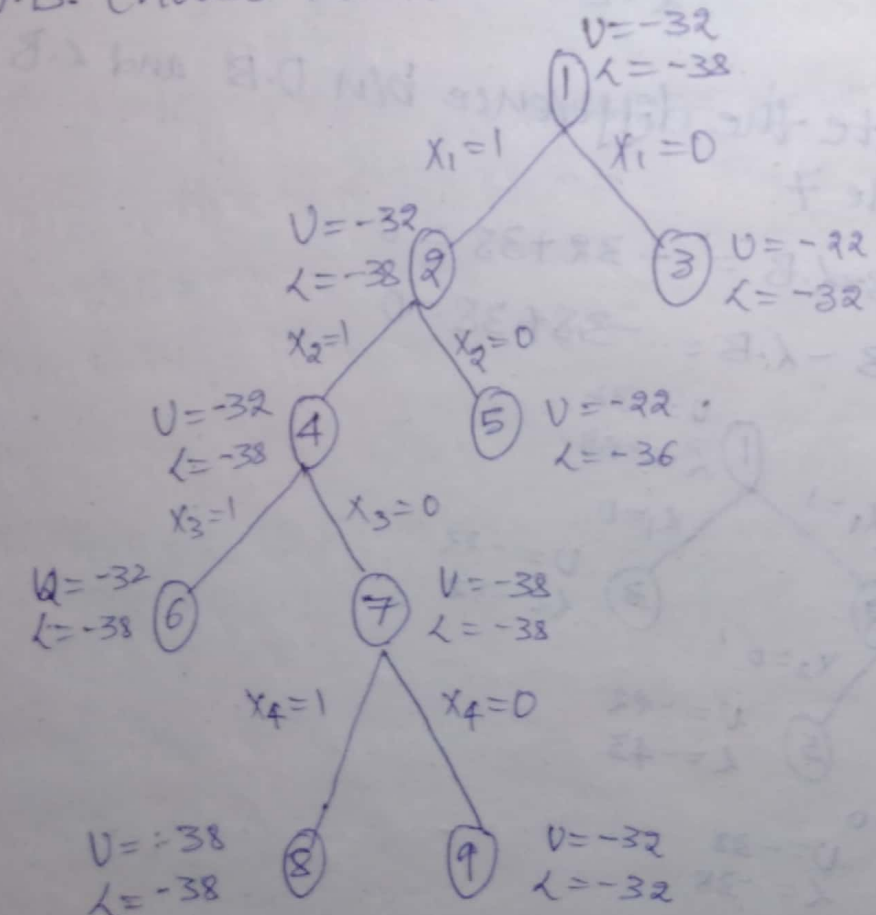
U.B = -32 $\quad$ ℓ.B = -32

Difference b/n ℓ.B's and U.B's of Node 8' and Node '9'

For Node 8: U.B − ℓ.B = −38+38 = 0

For Node 9: U.B − ℓ.B = −32+32 = 0

Here the difference b/n Nodes 8, 9 are same. So, we need to compare the upper bounds of Node '8' and Node '9' and choose the Node '8' which has minimum U.B. Choose the Node 8 and discard the Node '9'



Consider the path 1 to 2, 2 to 4, 4 to 7, 7 to 8

$2 \rightarrow 4 \rightarrow 7 \rightarrow 8$

$X_1 = 1 \quad X_2 = 1 \quad X_3 = 0 \quad X_4 = 1$

$$\sum P_i X_i = P_1X_1 + P_2X_2 + P_3X_3 + P_4X_4$$
$$= 10*1 + 10*1 + 12*0 + 18*1$$
$$= 10 + 10 + 0 + 18$$
$$= 38$$

II-chapter : Lower Bound Theory

Comparison Trees :-

Comparison Tree is derived from binary tree. To determine lower bound ~~theories~~ problems, comparison trees are used and these trees solve problems like sorting, searching and selection. Here we use model based algorithms to design the Comparison trees to provide the solution.

Searching :

In this problem, the elements are arranged in an ascending order, we use either linear search or binary search. We check 3 conditions

1) $x = A[i]$ - stop procedure
2) $x < A[i]$ - left tree is searched
3) $x > A[i]$ - right tree is searched

Selection :

Required element is first searched and then select that element. Here we have 2^{n-1} external nodes, 'n-1' internal nodes

Sorting:

In this technique comparison-tree restrictions should satisfy

i) $A[i] < A[j]$ - left-tree

ii) $A[i] > A[j]$ - right tree

Elements are arranged in sorted order and best technique is Radix sort.

Lower Bounds through Reductions:

i) Multiplying triangular matrices

A is a $n \times n$ matrix of triangular matrix. We have 2 types of triangular matrices

i) Upper triangular matrix $\rightarrow A = \{a_{ij}\}$

$a_{ij} = 0, \; i > j$

ii) Lower triangular matrix $\rightarrow A = \{a_{ij}\}$

$a_{ij} = 0, \; i < j$

Here we concentrate on lower-triangular matrices.

Multiplication of 2 lower-triangular matrices is time complexity is $M_t(n)$.

General matrix multiplication AXB, time complexity is $M(n)$ and $M(n) \geq M_t(n)$

To prove this, we need to prove the lemma

$M_t(n) = \Omega(M(n))$

* Lemma $M_t(n) = \Omega(M(n))$

Here we need to show that problem P_1 reduces to problem P_2 in $O(n^2)$ time

$P_1 = M(n)$ is the time needed to multiply 2 $n \times n$ matrices

$P_2 =$ ~~the~~ $M_t(n)$ is the time needed to multiply 2 lower-triangular matrices

Note that $M(n) = \Omega(n^2)$, since there are two n^2 elements in the i/p and one n^2 in the Output.

Let the two matrices to be multiplied be A & B of size nxn each. The instance of P_2 to be created in the following way:

$$A' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & A & 0 \end{bmatrix} \qquad B' = \begin{bmatrix} 0 & 0 & 0 \\ B & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Here 0 stands for the zero matrix. Both A' & B' are of size 3nx3n each. Multiply the 2 matrices then we get

$$A'B' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ AB & 0 & 0 \end{bmatrix}$$

Thus the product AB is easily obtainable from the product A'B'. Problem P_1 reduces to P_2 in $O(n^2)$ time. This reduction implies that

$$M(n) \leq M_t(3n) + O(n^2)$$
$$M_t(n) \geq M(n/3) - O(n^2)$$
$$M(n) = \Omega(n^2)$$
$$M(n/3) = \Omega(M(n))$$

Hence $M_t(n) = \Omega(M(n))$

Inverting a lower triangular matrix:

Let A be an nxn matrix also let I be the nxn identity matrix and whose other element is 0. Every element of I is 0 except for the diagonal elements which are all. Once, if there exists an nxn matrix 'V' such that AB = I then we say that 'V' is the inverse of A and A is said to be invertible. The inverse of A is also denoted as A^{-1}

In this section we obtain the lower bound on the inversion time of a lower triangular matrix. It can be shown that a triangular matrix is invertible if and only if all its diagonal elements are non-zero.

Let P_1 be the problem of multiplying two full matrices and P_2 be the problem of inverting a lower triangular matrix. Let $M(n)$ and $I_t(n)$ be the corresponding time complexities. Here we need to show that $M(n) = O(I_t(n))$

Lemma: $M(n) = O(I_t(n))$

Proof: Note that P_1 reduces to P_2 in $O(n^2)$ time.

Let A & B be the two $n \times n$ full matrices to be multiplied. Construct the following lower triangular matrix in $O(n^2)$ time

$$C = \begin{bmatrix} I & 0 & 0 \\ B & I & 0 \\ 0 & A & I \end{bmatrix}$$

where the zeroes and I's are the $n \times n$ zero matrices and Identity matrices respectively.

C is a $3n \times 3n$ matrix. The inverse of C is

$$C^{-1} = \begin{bmatrix} I & 0 & 0 \\ -B & I & 0 \\ AB & -A & I \end{bmatrix}$$

where $-A$ refers to A with all the elements negated. Here the product of AB is obtained easily from the inverse of C. Thus we get $M(n) \leq I_t(3n) + O(n^2)$

Hence $M(n) = O(I_t(n))$

Lemma: $I_t(n) = O(M(n))$

Proof: Let 'A' be a $n \times n$ lower-triangular matrix to be inverted. Here A is partitioned into four submatrices of size $\frac{n}{2} \times \frac{n}{2}$ each as follows:

$$A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}$$

Both A_{11} & A_{22} are the lower triangular matrices and A_{21} be a full matrix. The inverse of A is

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{22}^{-1}(A_{21}A_{11}^{-1}) & A_{22}^{-1} \end{bmatrix}$$

(D)

The above equation suggest divide & conquer algorithm for inverting A. To invert A of size $n \times n$, it suffices to invert the lower triangular matrices of size $\frac{n}{2} \times \frac{n}{2}$ and perform two matrix multiplications and negate matrix '0'. D can be negated in $\frac{n^2}{4}$ time

The runtime of a divide & conquer algorithm satisfies the following recurrance relation

$$I_t(n) \leq 2I_t(n/2) + 2M(n/2) + \frac{n^2}{4} \rightarrow \text{'0' time comp}$$

Using repeated substitution, we get

$$I_t(n) \leq 2M\left(\frac{n}{2}\right) + 2^2 M\left(\frac{n}{2^2}\right) + \cdots + O(n^2)$$

since $M(n) = \Omega(n^2)$

$$I_t(n) = O((M(n) + n^2) = O(M(n))$$

Hence $I_t(n) = O(M(n))$ the above two lemma's together imply that $I_t(n) = O(M(n))$

~~$= O(I_t(n))$~~

Computing the transitive closure:

Let 'G' be a directed graph whose adjacency matrix is 'A' and the transitive closure of G, denoted as A^*, is a matrix such that $A^*(i,j)=1$ if & only if there is a directed path of length '0' or more from node i to node j in G

Lemma: $M(n) \leq T(3n) + O(n^2)$ and hence

$M(n) = O(T(n))$

Proof: If A and B are the given nxn matrices to be multiplied, form the following 3nx3n matrix seen in $O(n^2)$ time

$$C = \begin{bmatrix} 0 & A & 0 \\ 0 & 0 & B \\ 0 & 0 & 0 \end{bmatrix}$$

C^2 is given by

$$C^2 = \begin{bmatrix} 0 & 0 & AB \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Also $C^k = 0$ for $k \geq 3$

$\therefore A^* = I + A + A^2 + \cdots + A^{n-1} \rightarrow (1)$

Using eq (1)

$C^* = I + C + C^2 + \cdots + C^{n-1} = I + C + C^2 = \begin{bmatrix} I & A & AB \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$

Given C^*, it is to obtain the product AB

Lemma : $T(n) = O(M(n))$

Proof : Let $G(V,E)$ be the graph and A is its adjacency matrix. The matrix A is partitioned into 4 submatrices of size $\frac{n}{2} \times \frac{n}{2}$ each, then

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

row i of A corresponds to edges going out of node i. Let V_1 be the set of nodes corresponding to rows $1, 2, \ldots \frac{n}{2}$ of A and V_2 be the set of nodes corresponding to the rest of rows.

the entry $A^*(i,j) = 1$ if & only if there is a path from node $i \in V_1$ to node $j \in V_1$ all of the intermediate nodes are also from V_1.

A similar property holds for A^*_{22}.

Let $D = A_{12} A_{21}$ and let $u \& v \in V_1$ then $D(u,v) = 1$ if and only if there exists a $w \in V_2$ such that $<u,w>$, & $<w,v>$ are in edge set.

A similar statement holds for $A_{21} A_{12}$

Let the transitive closure of G be given by $A^* = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

Our goal is to derive a divide & conquer algorithm for computing A^*. Therefore we should find a way of computing $C_{11}, C_{12}, C_{21}, C_{22}$ from A^*_{11} & A^*_{22}

First we consider the computation of C_{11}, note that C_{11} corresponds to paths from i to j, where i & j are

in V_1. Of course the intermediatory nodes in such paths could as well be from V_2. Therefore

$$\therefore C_{11} = (A_{11} + A_{12} A_{22}^* A_{21})^*$$

Similarly the rest of A^* can be determined

$$C_{12} = C_{11} A_{12} A_{22}^*$$

$$C_{21} = A_{22}^* A_{21} C_{11}$$

$$C_{22} = A_{22}^* + A_{22}^* A_{21} C_{11} A_{12} A_{22}^*$$

Thus the above divide & conquer algorithm for computing A^* performs two transitive closures on matrices of size $\frac{n}{2} \times \frac{n}{2}$ each. 6 matrix multiplications, 2 matrix additions on matrices of size $\frac{n}{2} \times \frac{n}{2}$ each

∴ We get the recurrance relation as.

$$T(n) \leq 2T(n/2) + 6M(n/2) + O(n^2)$$

(2 additions, 6 matrix multiplications; P(n) reduced to P(n/2) in DC)

Repeated substitution yields

$$T(n) \leq [6M(n/2) + 12M(n/4) + 24M(n/8) + \cdots] + O(n^2)$$

But $M(n) \geq n^2$ & hence $M(n/2) \leq 4M(n)$

Using the above equations, we get

$$T(n) = O(M(n) + n^2) = O(M(n))$$

Hence $T(n) = O(M(n))$

The above two lemma's together implies $T(n) = \Theta(M(n))$