BACKTRACKING

UNIT-5

- → The General Method.
- → The 8-Queens problem. Sum
- → Sum of Subsets.
- → Graph Coloring.
- → Hamiltonian Cycles.

GENERAL METHOD:

→ In the Backtracking Method

1. The desired Solution is expressible as an n tuples $(x_1, x_2, \cdots x_n)$ where x_i is chosen from some finite set S_i.
2. The Solution Maximizes & Minimizes & Satisfies a Criterion function $C(x_1, x_2, \cdots x_n)$.

→ The problem can be catagorized into three catagories. For Instance - For a problem P let C be the set of Constraints for P. Let D be the Set Containing all Solutions satisfying C then

i) Finding whether there is any feasible Solution? is the decision problem.

ii) What is the best Solution? - is the optimization problem.

iii) Listing of all the feasible Solution? - is the enumeration problem.

→ The basic idea for backtracking is to build up a vector, one Component at a time and to test whether the vector being formed has any chance of success.

→ The Major Advantage of this algorithm is that we can realize the fact that the partial vector generated 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 function. All solutions using backtracking are required to satisfy a complex set of constraints. The constraints may be explicit & Implicit.

→ Explicit constraints are rules, which restrict each vector element to be chosen from the given set. Implicit constraints are rules, which determine which of the tuples in the solution space, actually satisfy the criterion function.

N-QUEENS PROBLEM:

We can place N No.of Queen's (Q) in NXN Chessboard N-Queen problem is Majorly divided into two types.

1. 4 Queen's problem.
2. 8 Queen's problem.

procedure:

1. No two of them attack each other.
2. No two Queen's placed in same row.
3. No two Queen's placed in same colaum.
4. No two Queen's placed in diagonal of a chessboard.

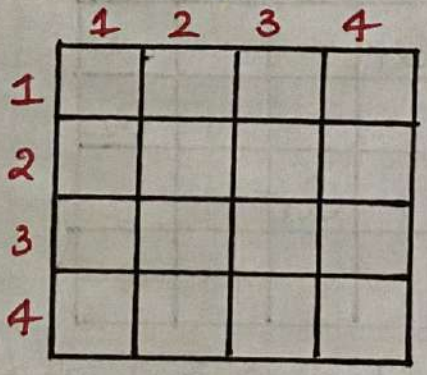
4x4 QUEEN'S PROBLEM:

→ 4 Number of Queen's can be placed in 4x4 chessboard.

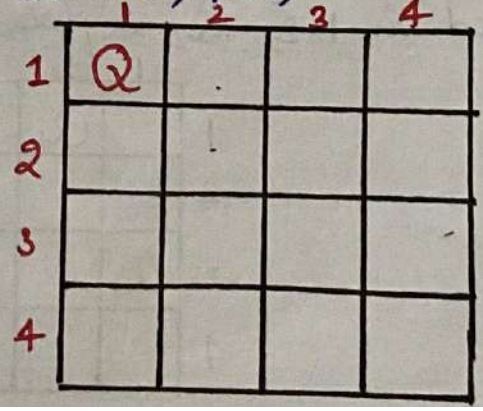
→ 4x4 chessboard Contains 16 Squares boxes

→ In 16 Square boxes we can place only 4 Queen's at the particular positions.

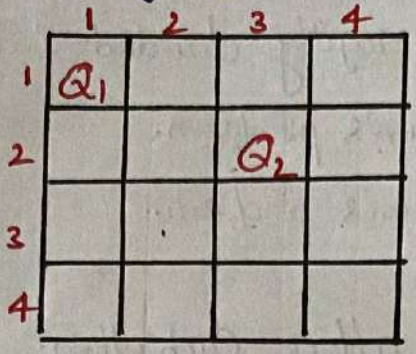
→ Intially 4x4 chessboard is



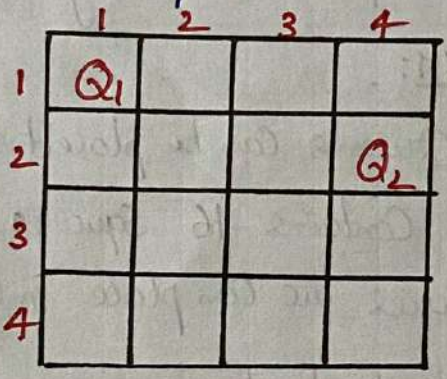
Step 1: Q_1 is placed in 1x1 position in the chessboard. The possible positions of Q_1 is 1x2, 1x3, 1x4



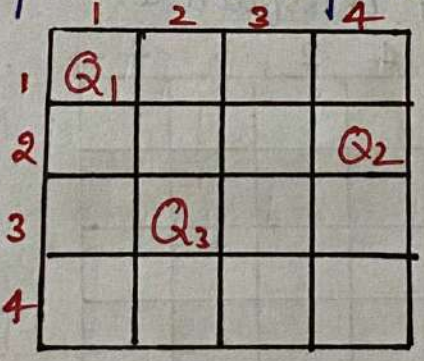
Step2: Now Q_2 is placed in 2×3 position in the Chess board. The possible positions of Q_2 is 2×4 only.



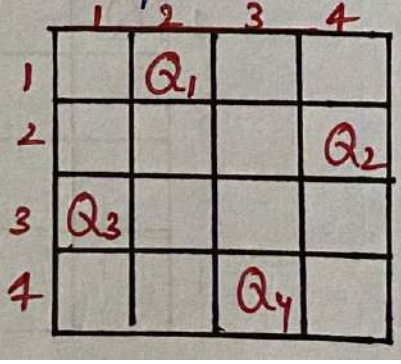
Step3: Now Q_3 is not placed in any position of 3×1, 3×2, 3×3 and 3×4 Hence we can apply back tracking on Q_2. Now Q_2 is placed in 2×4.



Step4: Now Q_3 is placed in 3×2 position There is no possible positions.



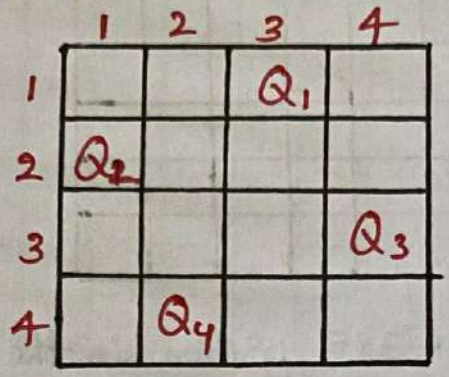
Step5: Now Q_4 is Not placed in any position of 4×1, 4×2, 4×3 4×4. Hence, apply Backtracking on Q_3, Q_2, & Q_1. Now. Q_1 is placed in 1×2, Q_2 is placed in 2×4, Q_3 is placed in 3×1



→ Hence possible positions are 1×2, 2×4, 3×1, 4×3

→ Hence the feasible solution is 2, 4, 1, 3

→ other possible solution is mirror Image.



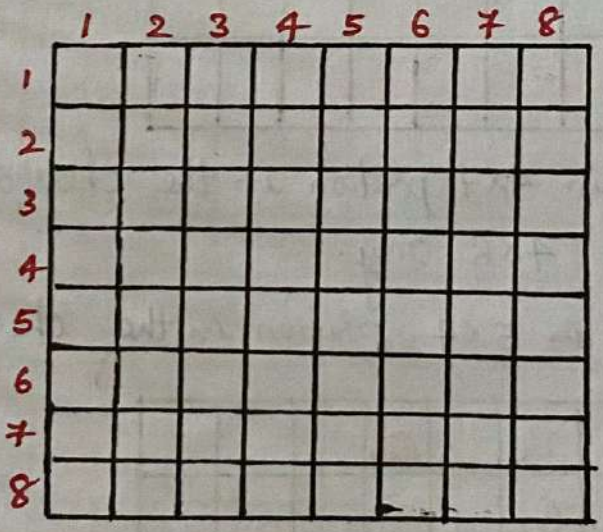
8×8 QUEENS PROBLEM:

→ 8 No. of Queen's Can be placed in 8×8 chessboard.

→ 8×8 chessboard Contain 64 Square boxes.

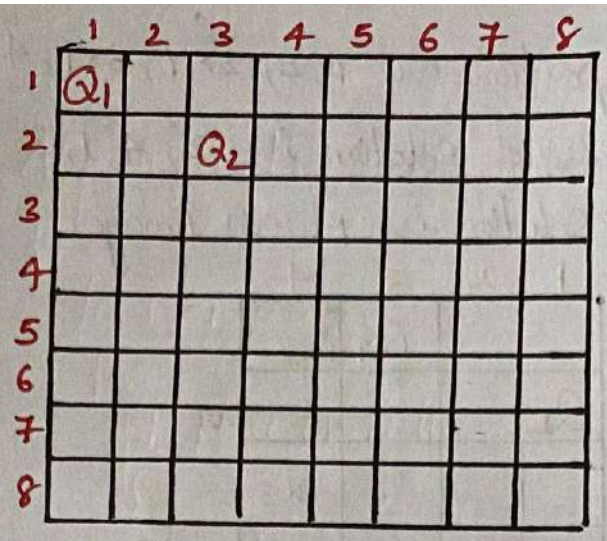
→ In 64 Square boxes we Can place only 8 Queens at the particular position.

→ Intially 8×8 Chessboard is

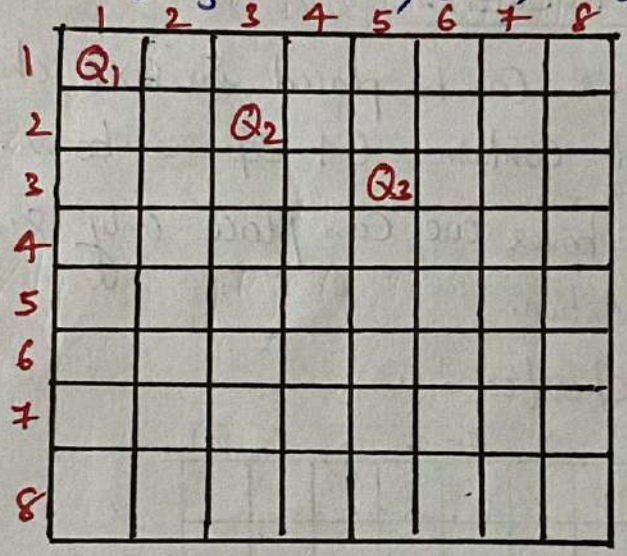


Step1: Q_1 is placed in 1×1 position in the Chessboard. The possible positions ~~are~~ of Q_1 is 1×2, 1×3, 1×4, 1×5, 1×6 1×7, 1×8.

step2: Q_2 is placed in 2×3 position in the Chessboard. The possible positions of Q_2 is 2×4, 2×5, 2×6, 2×7, 2×8.

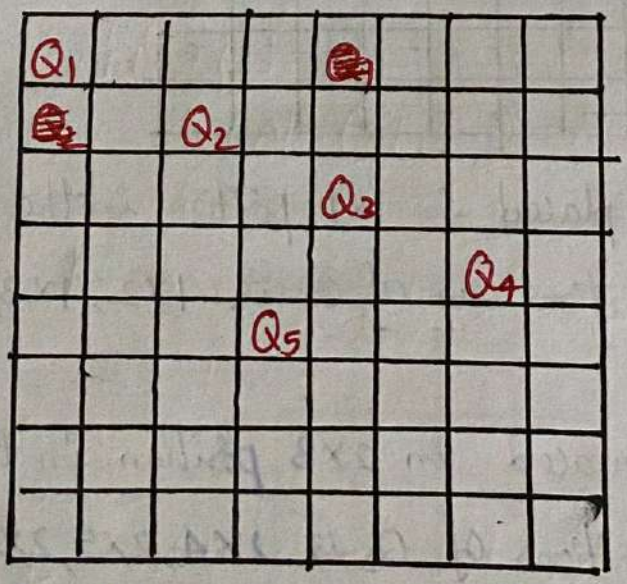


Step3: Q_3 is placed in 3X5 position in the chessboard. The possible positions of Q_3 is 3X6, 3X7, 3X8



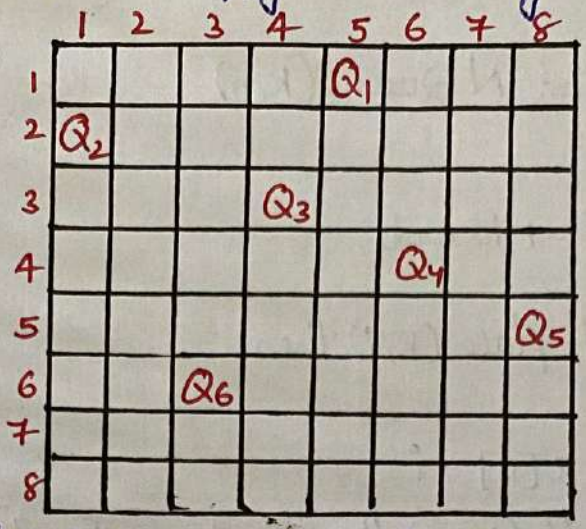
Step4: Q_4 is placed in 4X7 position in the chessboard the possible positions of Q_4 is 4X8 only.

Step5: Q_5 is placed in 5X4 position in the chessboard.



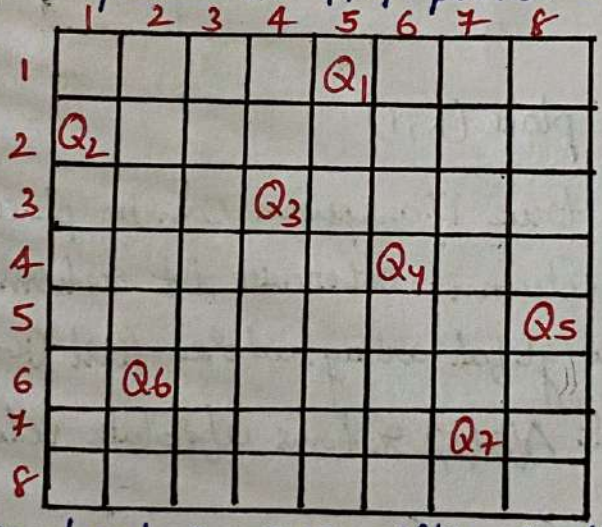
Step6: Q_6 is Not placed in any position of 6X1, 6X2, 6X3, 6X4, 6X5

6×6, 6×7, 6×8, Hence apply Backtracking on Q_5, Q_4, Q_3, Q_2, Q_1.

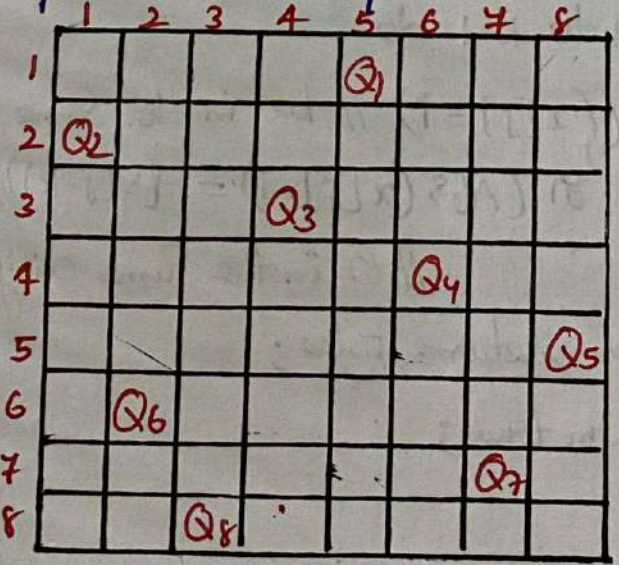


Now Q_1 is placed in 1×5, Q_2 is placed in 2×1, Q_3 is placed in 3×4 and Q_4 is placed in 4×6, Q_5 is placed in 5×8 and Q_6 is placed in 6×2.

Step 7: Q_7 is placed in 7×7 position in the chessboard.



Step 8: Q_8 is placed in 8×3 position in the chessboard.



Hence the possible positions are 1×5, 2×1, 3×4, 4×6 5×8, 6×2, 7×7, 8×3

→ Hence the feasible solution is (5,1,4,6,8,2,7,3)
another possible solution is (3,6,2,7,1,4,8,5)

ALGORITHM:

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

Algorithm place(k,i)
// Return true if a queen can be placed in k^th row
 and i^th column. otherwise it returns FALSE.
 x[] is a global array whose first (k-1) values have
 been set. Abs(r) returns absolute value of r.//
{
 for j=1 to k-1 do
    if((x[j]=i) // two in the same column
       or (Abs(x[j]-i)=Abs(j-k)))
                // or in the same diagonal
    then returns false;
   return true;
 }
```

→ Time Complexity is $O(n \times n) = \underline{O(n^2)}$

GRAPH COLORING:

→ Graph Coloring is a problem of coloring each vertex in graph in such a way that no two adjacent vertices have same color and yet m-colors are used. This problem is also called m-coloring problem.

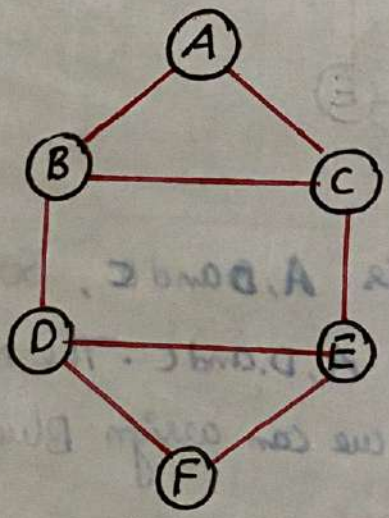
→ If the degree of given graph is d then we can color it with d+1 colors. The least number of colors needed to color the graph is called its chromatic number.

→ Graph Coloring problem having two parts

1. 'm' Coloring decision problem.
2. 'm' Colorability optimization.

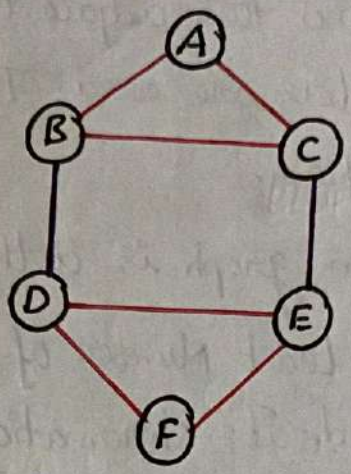
→ Here 'm' is called chromatic Number. where m represents chromatic number.

Ex:

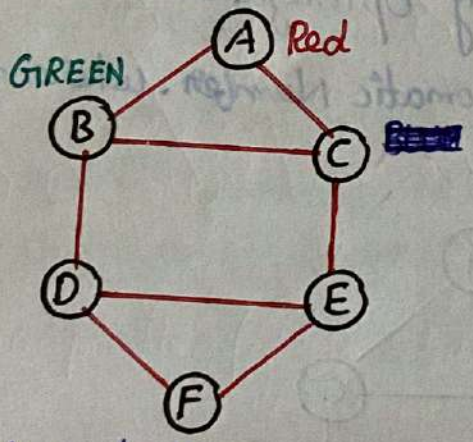


→ We require three colors to color the graph. Hence the chromatic number of given graph is 3. We can use Backtracking technique to solve the graph coloring problem.

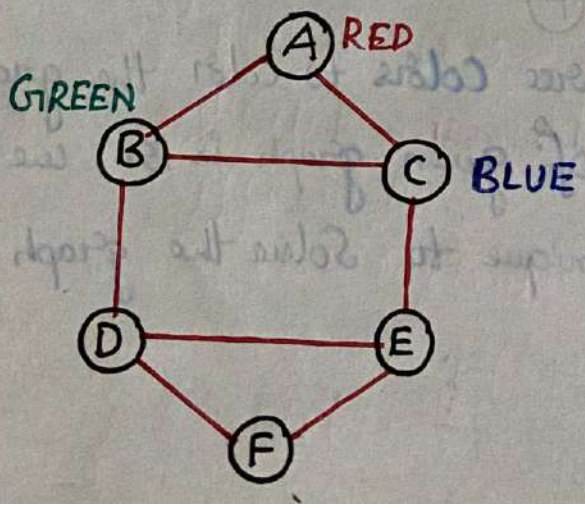
Step 1: A graph G1 Consists of vertices from A to F. There are three Colors used Red, Green and Blue.



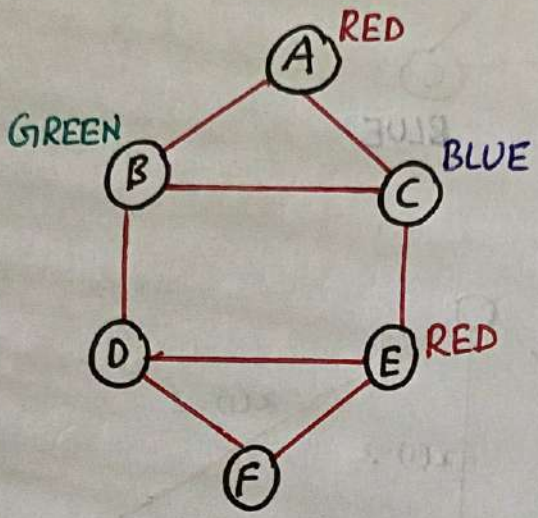
Step 2: Select Node A is filled by Red Color the adjacent of node A is B & C. So we Can not assign Red Color at B & C. Hence we Can fill node B with green.



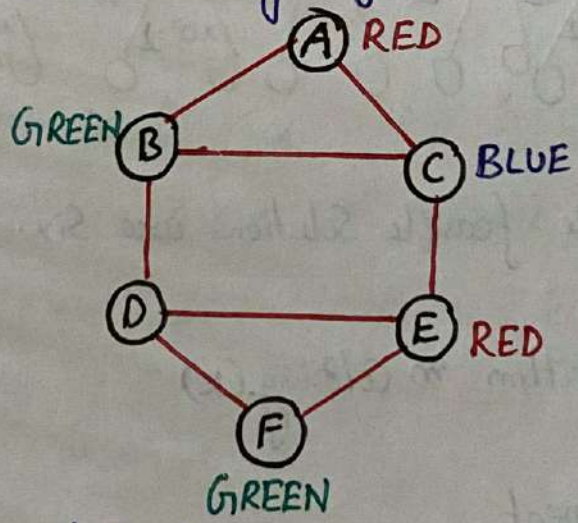
Step 3: The adjacent of B is A, D and E, So we Cannot assign Green Color to A, D. and C. The Node A is already fille with Red. Hence we Can assign Blue Color to 'B'



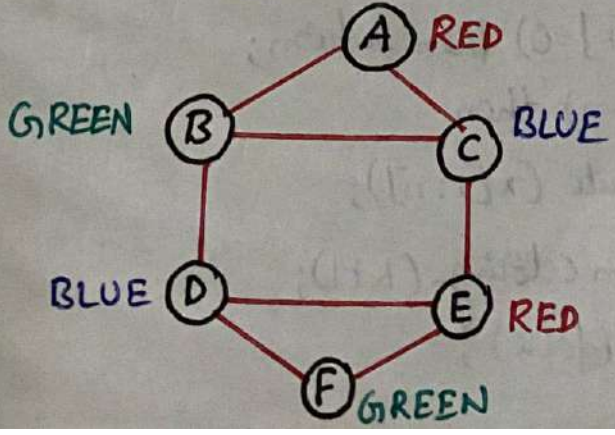
Step 4: The adjacent of C is A, B & E, So we cannot assign Blue to A, B & E. The Nodes A & B is already fill with Red & green. Hence we assign any of Red and green to Node 'E'



Step 5: The adjacent of E is C, D & F, so we cannot assign Red to C, D & F. The Node C is already fill with BLUE Color. Hence we assign green to 'F'.

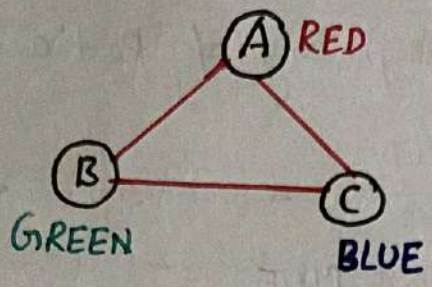


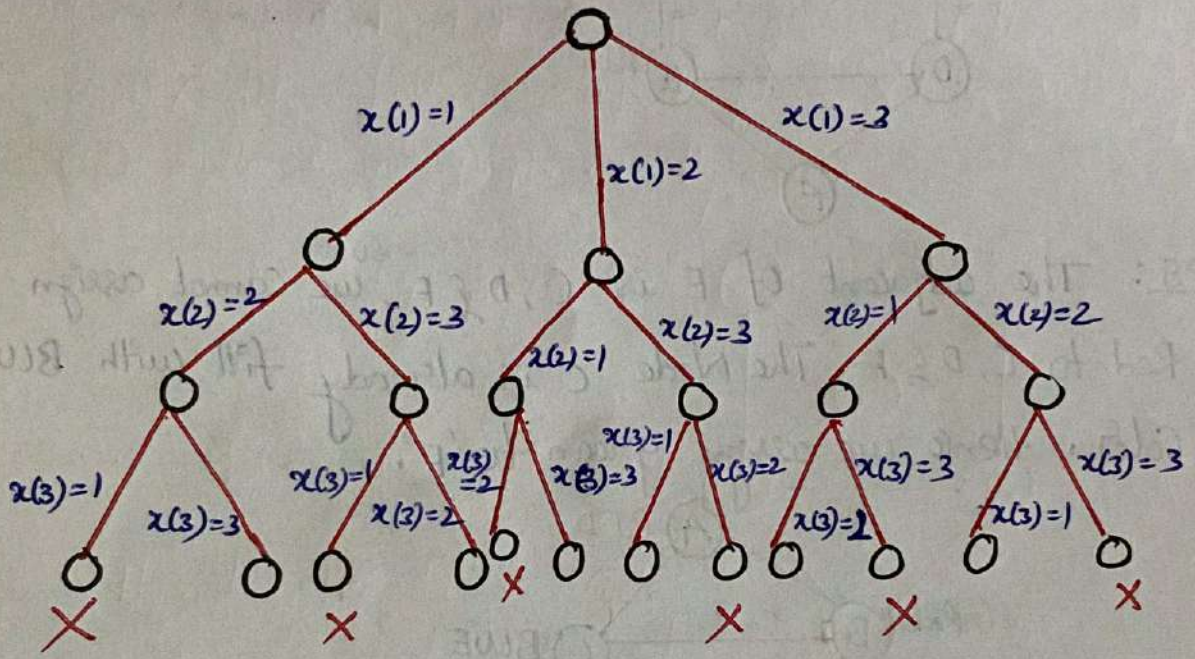
Step 6: The adjacent of D is B, F & E already Colored with Green & Red So we assign Blue to Node D.



Ex: The Three Colorful graph ($m = 3$) & $n = 3$ state space tree

→





The possible feasible solutions are Six.

ALGORITHM:

```
Algorithm m Coloring(k)
{
  Repeat
  {
    Next value (k);
    if (x[k]=0) then return;
    if (k=n) then
       Write (x[1:n]);
    else m coloring (k+1);
  } until (false);
}
```

Algorithm Next Value (k)

// x[1], --- x[k-1] Have been assigned integer values in the range [1,m] Such that adjacent vertices Have distinct integers. A value for x[k] is determined in the range [0,m]. x[k] is assigned the next highest numbered color While maintaing distinctness from the adjacent vertices of vertex k. IF no Such color exits, then x[k] is 0. //

```
{
 repeat
  {
   x[k] = (x[k]+1) mod (m+1); // next highest color.
   if (x[k]=0) then return; // All colors Have been used.
   for j=1 to n do
   {
     // check if this color is distinct from adjacent colors.
   if ((G[k,j] ≠0) and (x[k]=x[j]))
   // If (k,j) is and edge and if adjacent vertices
            Have the same color.
   then break;
   }
   if (j= n+1) then return; // New color found
  } until(False); // Otherwise try to find another color.
}
```

→ The time Complexity is $O(n \cdot m^n)$

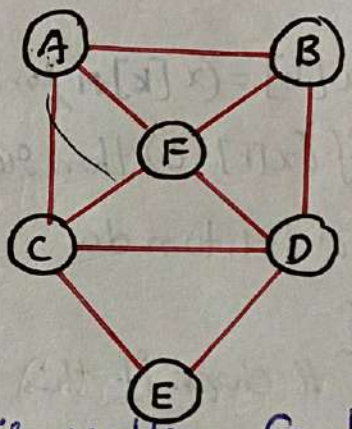
HAMILTONIAN CYCLES:

→ The great Mathematician william Hamiltonian, Introduced the dynamic Concept is called Hamiltonian Cycle (8) Hamiltonian Circuit

→ Formally a Hamiltonian Cycle of G1 Contain (V, E)
where E represents no. of edges.
V represents No. of vertices.

→ Given an undirected Connected graph and two nodes 'X' and 'Y' then find ~~from~~ a path from 'X' to 'Y' visiting Each node in the graph exactly once.

Ex: Consider the graph G1 is



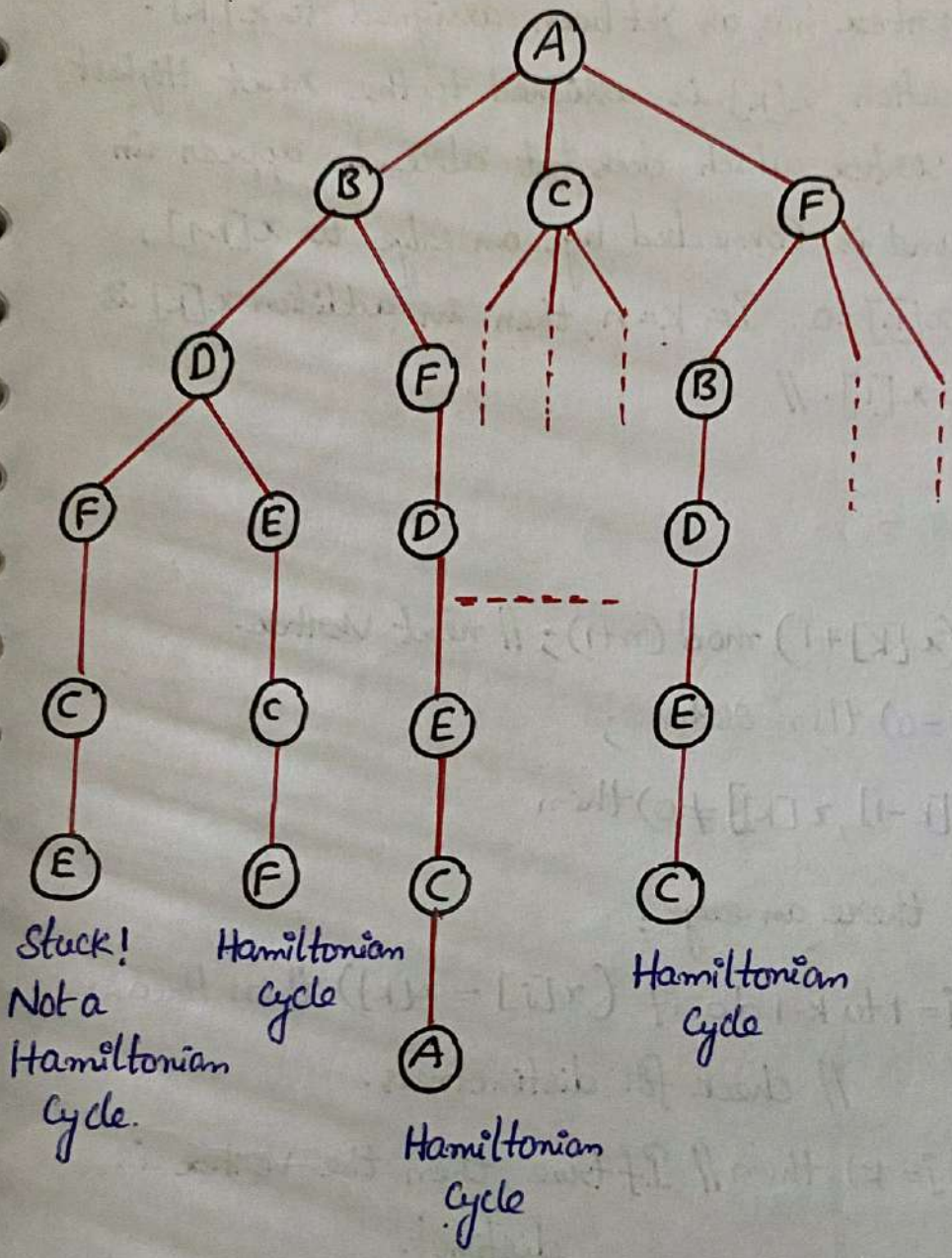
→ Then the Hamiltonian Cycle is A-B-D-E-C-F-A. This problem Can be Solved using Backtracking approach.

→ The state space tree is generated in order to find all the Hamiltonian Cycles in the graph. Only distinct Cycles are output of this algorithm. The Hamiltonian Cycles Can be identified as follows

→ In figure clearly the back-track approach is adopted. For Instances A-B-D-F-C-E; Here we get stuck.

→ For returning to 'A' we Have to revisit atleast one vertex. Hence we back-tracked and from D node

another path is choosen A-B-D-E-C-F-A which is Hamiltonian cycle.



ALGORITHM:

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

Algorithm Next Value (k)

// x[1:k-1] is a path of k-1 distinct vertices. IF x[k]=0, then no vertex has as yet been assigned to x[k]. After execution, x[k] is assigned to the next highest numbered vertex which does not already appear in x[1:k-1] and is connected by an edge to x[k-1]. otherwise x[k]=0. IF k=n, then in addition x[k] is connected to x[1]. //

```
{
  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
    { // Is there an edge?
      for j=1 to k-1 do if (x[j] = x[k]) then break;
                  // check for distinctness.
      if (j= k) then // If true, then the vertex is
                                distinct.
        if ((k<n) or ((k=n) and G[x[n], x[1]] ≠0))
        then return;
    }
  } until (False);
}
```

→ The time Complexity is $O((n-1)!) = \underline{O(n!)} = \underline{O(n^n)}$

SUM SUBSETS PROBLEM:

→ Suppose we are given 'n' distinct positive numbers and desire to find all Combinations of these Numbers whose Sum is M. This is called as the "Sum of Subsets problem".

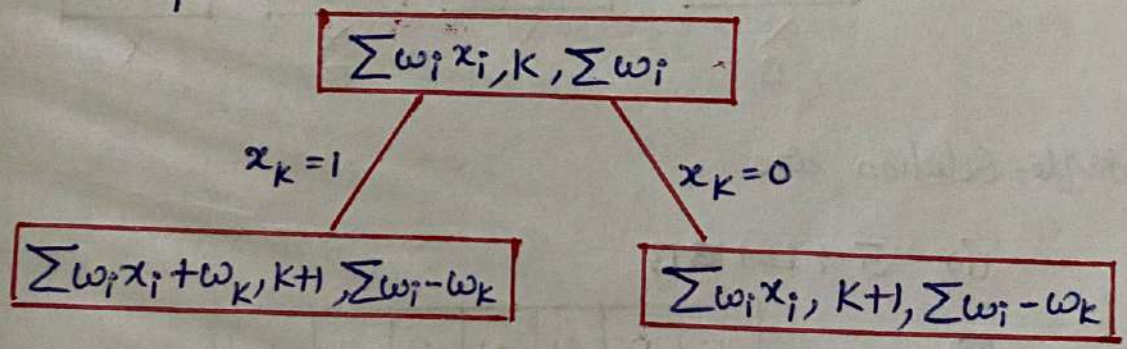
→ The element 'x_i' of the Solution Vector is either '1' or '0' depending on whether the weight 'w_i' is included or not.

→ For a node at level 'i' be the left child Corresponding to $x_i = 1$ and the Right child Corresponding to $x_i = 0$.

→ The Bounding function we use $B_K(x_1, x_2, \cdots x_K) = \text{true}$ if and only if

$$\sum_{i=1}^{K} w_i x_i + \sum_{i=K+1}^{n} w_i \geq M \quad (2) \qquad \text{and} \qquad \sum_{i=1}^{K} w_i x_i + w_{K+1} \leq M. \quad (1)$$

→ The State Space Tree Can be drawn using the following Technique.



EX: $n = 6 \quad M = 30$

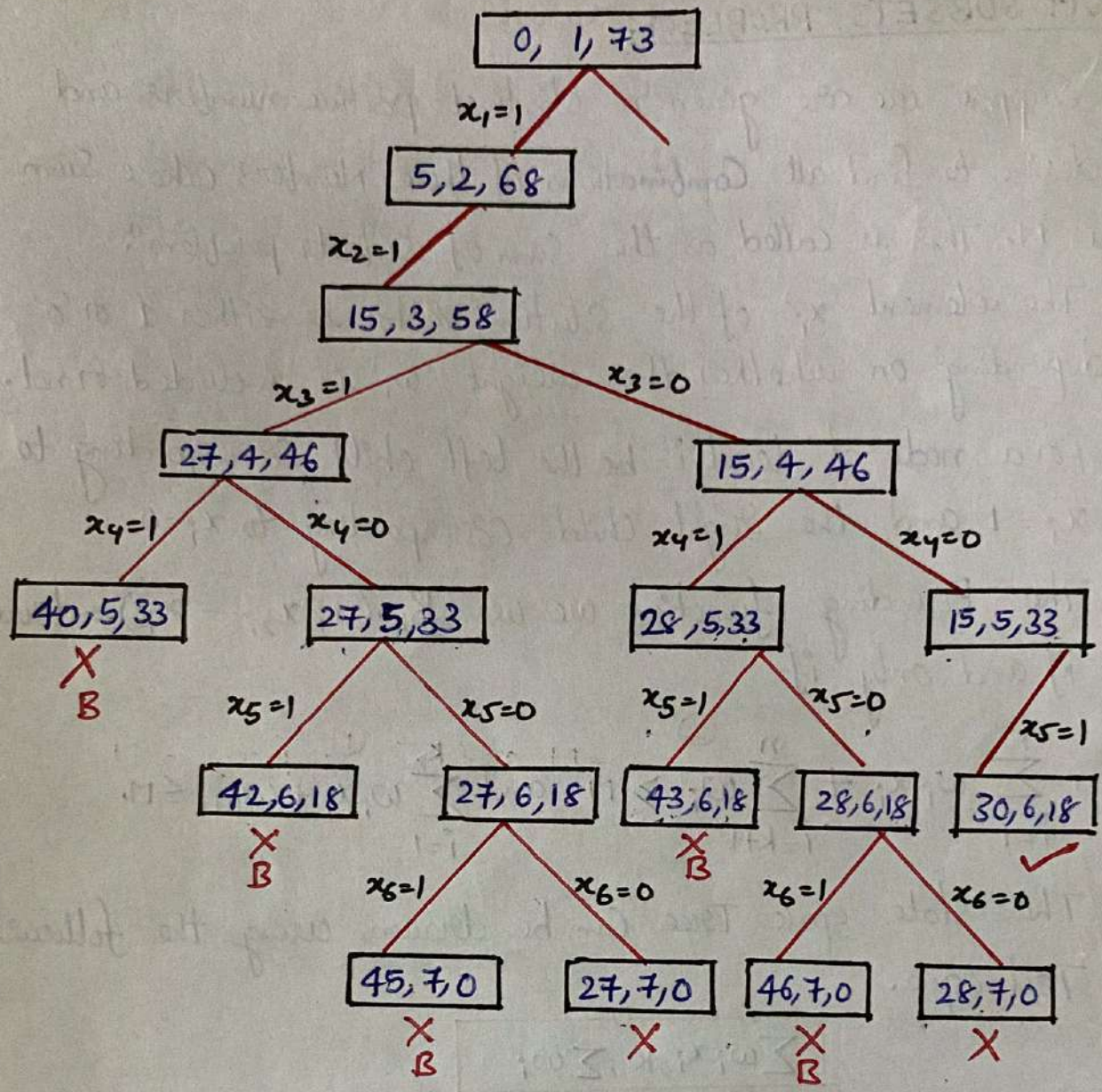
$W[1:6] = \{5, 10, 12, 13, 15, 18\}$

→ $x = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

$x_i = 0/1$

$\sum w_i = 73$

x	1	1	0	0	1	0
	1	2	3	4	5	6



→ possible Solution are

(i) 5 + 12 + 13

1	0	1	1	0	0

(ii) 5 + 10 + 15

1	1	0	0	1	0

(iii) 12 + 18

0	0	1	0	0	1

ALGORITHM:

```
Algorithm Sum of Sum (S, K, r)
// Find all subsets of w[1:n] that sum of m.
// The values of x[j], 1 ≤ j < K, have already been
   determined.
// s = Σ_{j=1}^{K-1} w[j] * x[j] and r = Σ_{j=K}^{n} w[j].
// The w[j]'s are in nondecreasing order.
// It is assumed that w[1] ≤ m and Σ_{i=1}^{n} w[i] ≥ m.
{
 // Generate left child. Note: s + w[K] ≤ m since
          B_{K-1} is true.
  x[K] = 1;
  if (s + w[K] = m) then Write (x[1:K]);
  // Subset found There is no recursive call here
          as w[j] > 0, 1 ≤ j ≤ n.
  else if (s + w[K] + w[K+1] ≤ m)
     then Sum of Sub (s + w[K], K+1, r - w[K]);
  // Generate right child and evalute B_K
  if ((s + r - w[K] ≥ m) and (s + w[K+1] ≤ m)) then
  {
     x[K] = 0;
     Sum of Sub (s, K+1, r - w[K]);
  }
}
```

→ The Time Complexity is $O(2^n)$

BRANCH AND BOUND

INTRODUCTION:

→ Branch and Bound is a General algorithm Method for finding optimal solutions of Various optimization problems.

→ Branch and Bounding Method is general optimization technique that applies where the greedy method and Dynamic programming fail. However, it is much slower.

→ Indeed, it often leads to exponential time Complexities in the worst case. On the other hand, if applied Carefully, it Can lead to algorithms that run reasonably fast on Average.

GENERAL METHOD:

→ In Branch and Bound Method a state space tree is built and all the children Of E-nodes (a live node whose children are currently being generated) are generated before any other node Can become a live node.

→ For Exploring new nodes either a BFS or D-(DFS) Search technique Can be used.

→ In Branch and Bound technique, BFS-like State Space Search will be called FIFO (FIRST IN FIRST OUT) Search. This is because the list of live node is FIRST IN FIRST OUT list (QUEUE). On the other hand the D-(DFS) Search like State Space Search will be called LIFO Search because the list of live node is Last in FIRST OUT list (stack).

→ In this Method a 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 at each node. This Computation leads to selection of answer node.

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

GENERAL ALGORITHM FOR BRANCH AND BOUND:

```
Algorithm Branch-Bound()
{
 // E is a node pointer;
    E ← new (node); // This root node which is the
                       dummy start node.
 // H is heap for all the live nodes.
 // H is Min-Heap for Minimization problems,
 // H is Max-Heap for Maximization problems.
 While (true)
  { if (E is a final leaf) then
    {   // E is an optimal Solution
     Write (path from E to the root);
     return;
    }
   Expand (E);
    if (H is Empty) then // if no element is present in
                            Heap
      {
          Write ("there is no solution);
          return;
```

```
    }
     E ← delete_top(H);
    }
  }

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

LEAST COUNT SEARCH;

→ In branch and bound Method the basic idea is Selection Of E-node. The Selection of E-node should so perfect that we will reach to answer node quickly.

→ Using FIFO and LIFO branch and bound Method the Selection of E-node is very Complicated and Somewhat blind.

→ For Speeding up the Search process we need to intelligent Ranking function for live nodes. Each time, the next E-node is Selected on the basis of this Ranking function. For this Ranking function additional Computation (Normally Called as Cost) is needed to reach to answer node from the live node.

→ The least cost (LC) Search is a Kind of Search in which least Cost is involved for reaching to answer node. At each E-node the probability of being an answer node is checked.

→ BFS and D-Search are Special Case of LC Search.

→ Each time the next E-node is Selected on the basis of the Ranking function (Smallest $c\hat{}(x)$). let $g\hat{}(x)$ be an estimate of the additional effort needed to reach an answer node from x. Let h(x) to be the Cost of reaching x from the root and f(x) to be any non-decreasing function such that

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

→ If we set $g\hat{}(x)=0$ and $f(h(x))$ to be level of node x then we have BFS.

→ If we set $f(h(x))=0$ and $g\hat{}(x) \leq g\hat{}(y)$ whenever y is a child of x then the Search is a D-Search.

→ An LC search with bounding functions is known as **LC** Branch and Bound Search.

→ In LC Search, the Cost function C(·) Can be defined as

(i) If x is an answer node then C(x) is the Cost Computed by the path from x to root in the state Space tree.

(ii) If x is not an Answer node Such that Subtree of x node is also not Containing the answer node $C(X) = \infty$.

(iii) Otherwise C(X) is equal to the Cost of Minimum Cost answer node in Subtree x.

→ $c\hat{}(\cdot)$ with $f(h(x)) = h(x)$ Can be approximation of C(·).

CONTROL ABSTRACTION FOR LC SEARCH :

```
Algorithm LC_Search()
// tr is a state space tree and X be the node in the tr
// E represents the E-node
// Intialize the list of live nodes to empty
{
  if (tr is answer node) then
  {
    Write (tr); // output the Answer node
    return;
  }
  E ← tr // Set the node
  repeat
  {
    for (each child X of E) do
    {
      if (X is answer node) then
      {
        output the path from X to root;
        return;
      }
// the new live node X will be added in the list of
   live nodes
      Add_Node (X);
      X → parent ← E; // pointing the path from X to root;
    }
    if (no more live nodes) then
    {
      write (" Can not have answer node!!");
      return;
    }
```

```
// E points to current E-node
        E ← least-count C); // finds the least cost node
                              to set next E-node
    } until (False);
}
```

In the above algorithm if x is in tr the c(x) be the Minimum Cost answer node in tr. The algorithm uses two functions least-cost and Add-node() function to delete & add the live node from the list of live nodes. using above algorithm we can obtain path from answer node to root. we can use parent to trace the parent of node X. Intially root is the E node. The for loop used in the algorithm examines all the children of E-node for obtaining the answer node. The function least-cost () looks for the next possible E-node.

15-PUZZLE PROBLEM:

→ This 15-puzzle problem can follows state space tree algorithm

→ The state space of an Intial state consists of all states that can be reached from intial state.

→ Given 4×4 board with 15 numbered tiles and 1 empty space (or) (empty spot).

→ By using this empty space we can arrange numbered value tiles in sequence order.

→ We can slide 4 adjacent (left, right, above & below) files into the empty space.

→ It is easy to see that there are 16! ($16! \cong 20.9 \times 10^{12}$) different arrangements of the files.

→ Each node x in the Search tree is associated with a cost. It is useful to determine the next node which contains least cost.

→ In 15-puzzle Algorithm can follows the below estimation cost.

Estimation Cost $c^{\wedge}(x) = f(x) + g^{\wedge}(x)$

Where $f(x)$ = length of the path from root node to x.

$g^{\wedge}(x)$ = No. of non blank tiles not in their goal position.

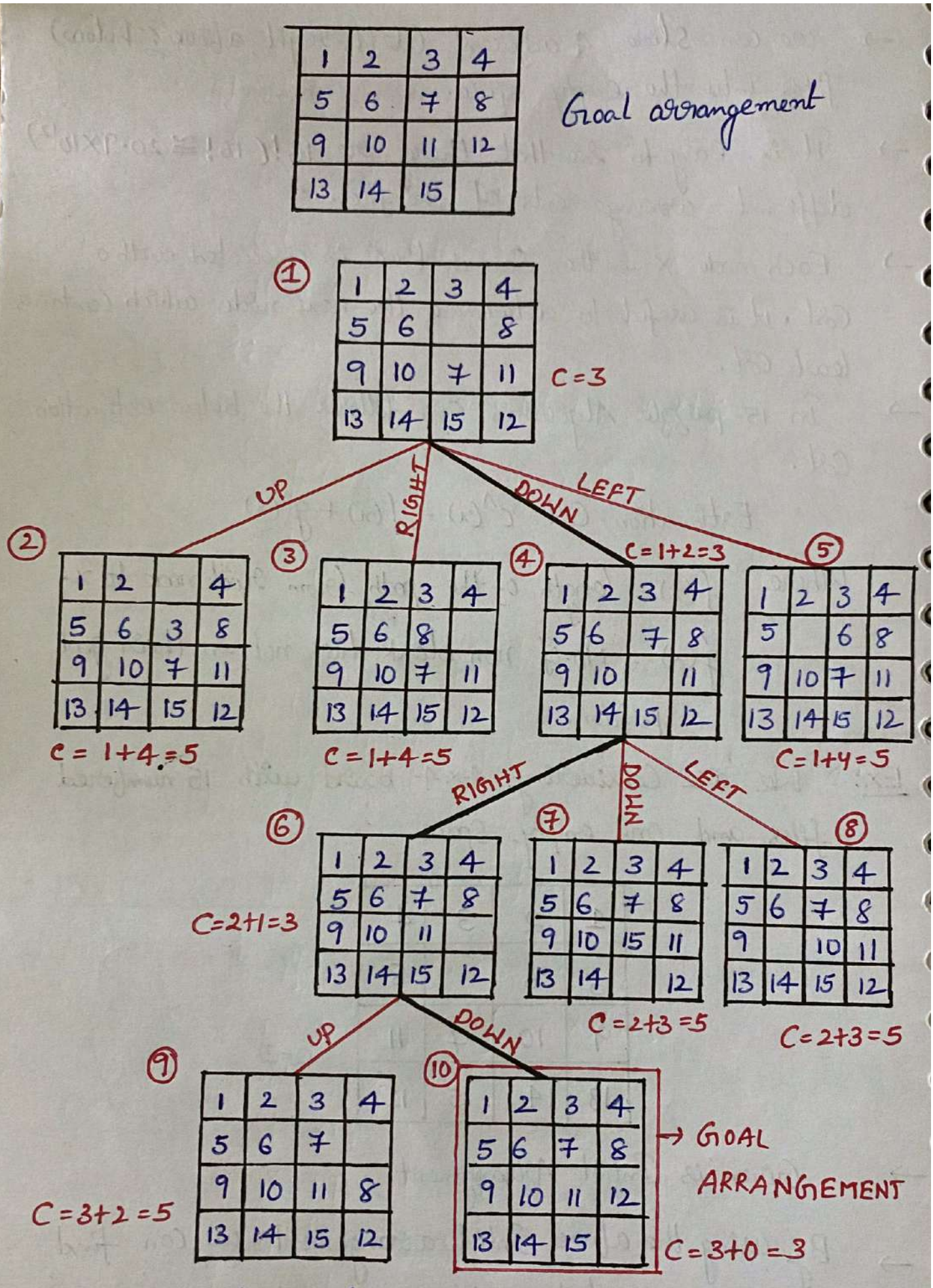
Ex: We are Considering 4×4 board with 15 numbered files and one empty space.

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

C=3

→ Given is Intial Arrangment

→ By using the above Intial arrangement we can find goal arrangement by using 15-puzzle position problem.


Goal arrangement
1
C=3
UP
RIGHT
DOWN
LEFT
2
C = 1+4 =5
3
C=1+4=5
4
C=1+2=3
5
C=1+4=5
RIGHT
DOWN
LEFT
6
C=2+1=3
7
C=2+3=5
8
C=2+3=5
UP
DOWN
9
C=3+2=5
10
GOAL ARRANGEMENT
C=3+0=3
∴ Space tree path = 1-4-6-10
Estimation Cost = 3+0 = 3

→ What we would like, is a more it "intelligent" Search Method, one that seeks out an answer node and adopts the path it seeks takes through the state space tree to the specific problem instance being solved.

→ We can associate a cost(x) with each node x in the state space tree. The cost(x) is the length of a path from root to a nearest goal node (if any) in the subtree with root x.

→ Thus, in figure C(1) = C(4) = C(6) = C(10) = 3. When such a cost function is available, a very efficient Search can be carried out.

→ We begin with the root as the E-node and generate a child node with C() value the same as the root.

→ Thus children nodes 2, 3, and 5 are eliminated and only node 4 becomes a live node. This becomes the next E-node.

→ It's first child, node 6 has C(6) = C(4) = 3. The remaining children are not generated. Node 4 dies and node 6 becomes the E-node.

→ In generating node 6's children, Node '9' is killed Immediately as C(9) > 3. Node '10' is generated next.

→ It is a goal node and the Search terminates.

→ In this Search Startegy, the only nodes to become E-nodes are nodes on the path from the root to a nearest goal node.

ALGORITHM:

```
Struct list_node
{
  list_node * next;
  list_node * parent;
  float cost;
};
  algorithm LC Search (list-node *t)
  {
    if (*t is an answer node)
    {
      print (*t);
      return;
  }
   E=t;
  Intialize the list of live nodes to be empty;
While (true)
  {
   for each child X of E
  {
    if X is an answer node {
    print the path from X to t;
   return; }
   Add(X);
    X→ parent = E; }
   if there are no more live nodes
{  print ("No answer node");
     return; }
   E = least();
} }
```

BOUNDING

→ As we know that the bounding functions are used to avoid the generation of subtrees that do not contain the answer nodes. In bounding lower bounds and upper bounds are generated at each node.

→ A cost function $c^{\wedge}(x)$ is such that $c^{\wedge}(x) \leq c(x)$ is used to provide the lower bounds on solution obtained from any node x.

→ let upper is an upper bound on cost of minimum-cost solution. In that case, all the live nodes with $c^{\wedge}(x) >$ upper can be killed.

→ At the start the upper is usually set to ∞. After generating the children of current E-node, upper can be updated by ~~Mani~~ Minimum cost answer node. Each time a new answer node can be obtained.

→ The solution represented as two ways

1) Variable size solution.
2) fixed size solution.

→ In Variable size solutions changes are possible.

→ But in fixed size solutions changes are not possible.

EX:

Considers Jobs = $\{X_1, X_2, X_3, X_4\}$

profits = $\{10, 5, 18, 13\}$

weights = $\{1, 2, 1, 2\}$

(1) Solution = $\{X_1, X_2, X_3\}$ → Variable size solution.

→ Here we can add (&) remove the variables

(2) Solution = {0,1,0,1} → fixed size solution

→ If we once fix the solution we cannot change the values.

→ In least count and FIFO the two factors are same

(i) Calculating the upper found

(ii) Cost

→ In upperfound, we should not consider the fraction values. It can be represented as '$\hat{v}$'

→ $\hat{v}$ = It is the sum of all profits according to the suitable weights.

→ In cost, consider the fraction values. It can be represented as '$\hat{c}$'

→ $\hat{c}$ = Sum of the total profits with fraction according to their weights

→ Bound values and cost always represented as Negation to the actual result.

→ The global founding value is 'infinity' (∞). The value must be changed if it gets smaller upper bound value.

JOB SEQUENCING WITH DEADLINES:

Let there be n Jobs with different processing times. with only one processor the Jobs are executed on or before given deadlines. Each Job i is given by a tuple (P_i, d_i, t_i) where t_i is the processing time required by Job i. IF processing of Job i is not completed by deadline d_i then Penalty P_i will occur. The objective of this problem is to Select a Subset J such that penalty will be Minimum among all possible Subsets. Such a J should be optimal Subset.

EX:

JOB INDEX	P_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

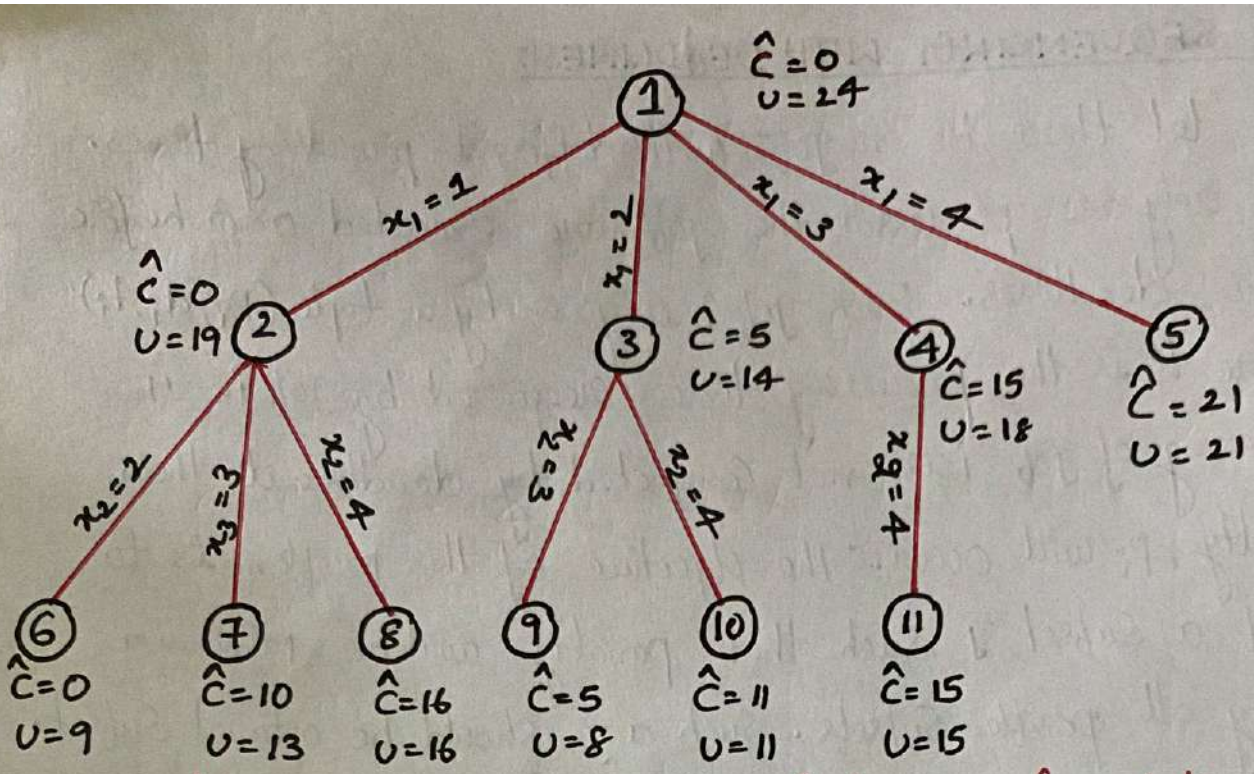
Then Select an optimal Subset J with optimal penalty. what will be the penalty Corresponding to optimal Solution?

→ The State Space tree Can be drawn for proper Selection of Job i for Creating J. There are two ways by which the State Space tree Can be drawn: Fixed tuple Size formulation and Variable tuple Size formulation.

→ The Variable Size formulation Can be shown below.

The function $\hat{c}(x)$ Can be Computed for each node x as

$$\hat{c}(x) = \sum_{i<m,\ i \notin S_x} P_i$$



State Space tree with Variable tuple Size formulation.

The upper bound can be Computed using function $u(x)$. Then $u(x) = \sum_{i \notin S_x} P_i$

Here $u(x)$ corresponds to the cost of the Solution S_x for node x.

→ For node 1 (root) there are 4 childrens. These children are for selection of either 1 or 2 or 3 for Job 4. Hence

$X_1 = 2$ or $X_1 = 3$ $X_1 = 4$

→ For node 2 we have with $x_1 = 1$ and therefore we can Select either 2, 3 or 4. Hence $x_2 = 2$ or $x_2 = 3$ or $x_2 = 4$ corresponds to node 6, 7 & 8. Continuing in this fashion, we have drawn the state space tree

At Node 1 :

$\hat{c} = 0$

$u = 24$

$\because \sum_{i=1}^{n} P_i = 5 + 10 + 6 + 3$

At Node 2:

$\hat{c} = 0$ $\because \sum P_i$ where $i<1 = 0$

$u = 19$ $\because$ the Sum of P_i excluding $x_1 = 1$ i.e. $P_1 = 5$.

At Node 3:

$\hat{c} = 5$ $\because \sum P_i$ where $i<2$

$u = 14$ $\because \sum P_i$ without $x_1 = 2$ i.e $24-10 = 14$

At Node 4:

$\hat{c} = 15$ $\because \sum P_i$ Where $i<3$

$u = 18$ $\because \sum P_i$ without P_3 i.e $24-6 = 18$

At Node 5:

$\hat{c} = 21$ $\because \sum P_i$ Where $i<4$

$u = 21$ $\because \sum P_i$ without P_4 i.e $24-3 = 21$

At Node 6:

$\hat{c} = 0$ $\because \sum_{i<2} P_i$ and not Containing $x_1 = 1$ i.e P_1

$u = 9$ $\because \sum_{i=1}^{n} P_i$ Such that $i \neq 2$ and $i \neq 1$.

Hence $6+3 = 9$

At Node 7:

$\hat{c} = 10$ $\because \sum_{i<3} P_i$ Where $i \neq 1$ and $i \neq 3$. Hence $P_2 = 10$

$u = 13$ $\because \sum_{i=1}^{n} P_i$ Such that $i \neq 1$ and $i \neq 3$. Hence

$P_2 + P_4 = 13$

At Node 8:

$\hat{c} = 16$ $\because \sum_{i<4} P_i$ and $i \neq 1$ and $i \neq 4$. Hence $P_2 + P_3 = 16$

$u = 16$ $\because \sum_{i=1}^{n} P_i$ and $i \neq 1$ and $i \neq 4$. Hence $P_2 + P_4 = 16$

At Node 9:

$c^{\wedge} = 5$ $\because \sum_{i<3} P_i$ but $i \neq 2$. Hence $P_1 = 5$

$u = 8$ $\because \sum_{i=1}^{n} P_i$ but $i \neq 2$ and $i \neq 3$. that is $P_1 + P_4 = 8$

At Node 10:

$c^{\wedge} = 11$ $\because \sum_{i<4} P_i$ but $i \neq 2$. Hence $P_1 + P_3 = 5+6 = 11$

$u = 11$

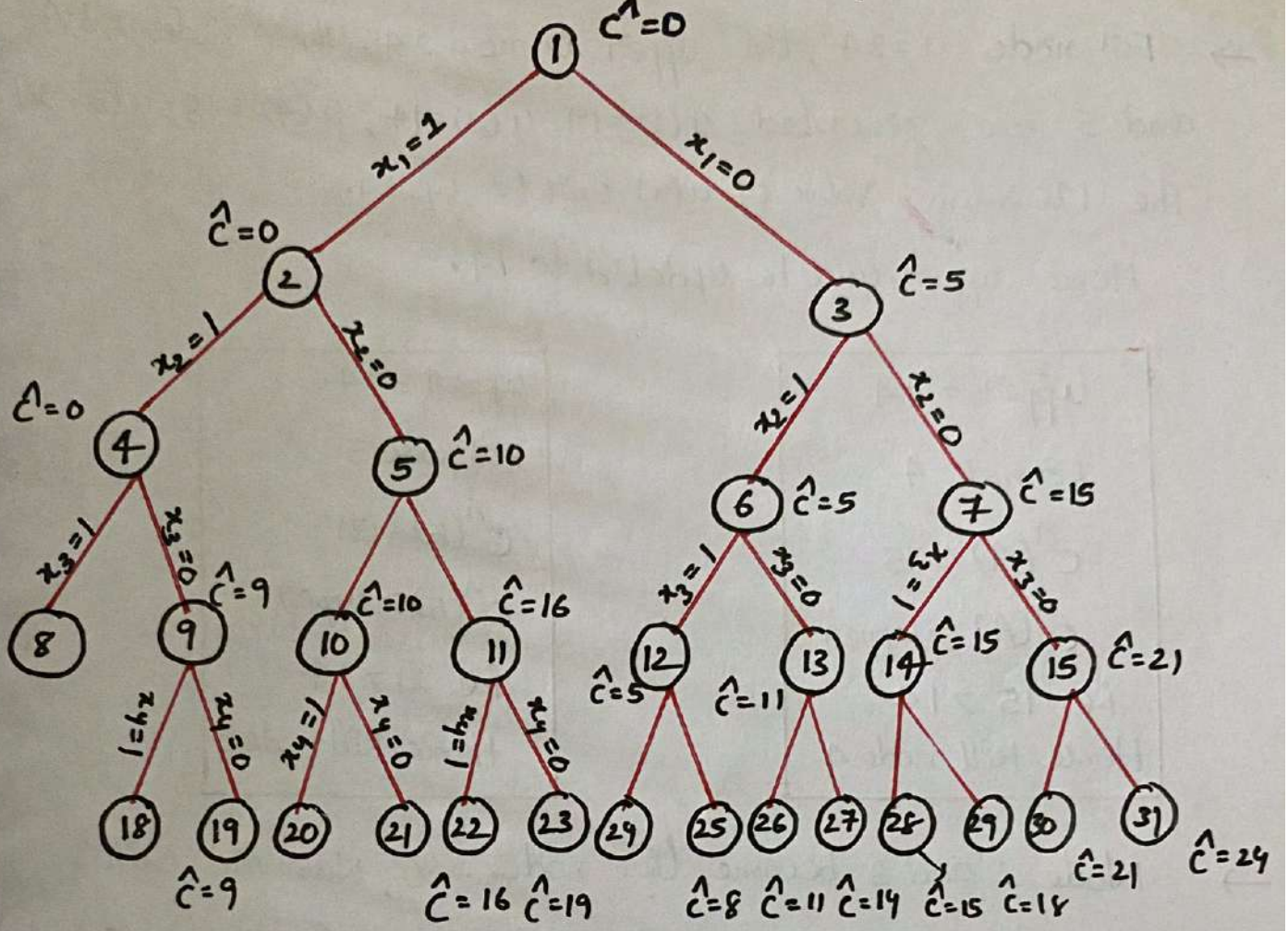
At Node 11:

$c^{\wedge} = 15$ $\because \sum_{i<4} P_i$ but $i \neq 3$. Hence $P_1 + P_2 = 5+10 = 15$

$u = 15$ $\because \sum_{i=1}^{n} P_i$ and $i \neq 3$ and $i \neq 4$. Hence $P_1 + P_2 = 5+10 = 15$.

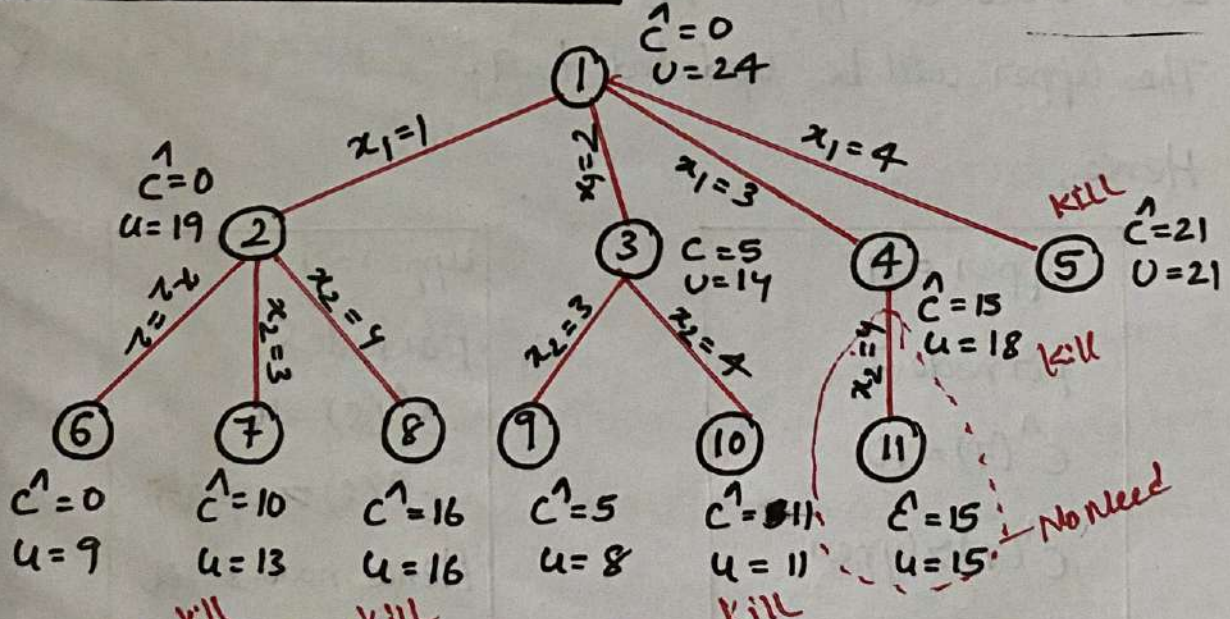
Now we will draw fixed tuple Size formulation.

IN fixed tuple Size formulation. $X_i = 1$ means Job i is Selected $X_i = 0$ means Job i is not Selected for formation of J (the optimal Subset). Intially, $c^{\wedge} = 0$ for node 1 as no Job is selected. For node 3, it indicates Ommision of Job 1. Hence, penalty is 5. For node 5 there is Omission of Job 2. Hence $c^{\wedge}$ = penalty = 10. For node 7 there is Omision of Job 1 and Job 2. Hence $c^{\wedge} = 15$. For node 13 there is Omission of Job 1 and Job 3. Hence penalty is 11. Therefore $c^{\wedge} = 11$. Continuing in this fashion $c^{\wedge}$ are Computed.



State space tree for fixed tuple Size formulation.

FIFO BRANCH AND BOUND:



To understand FIFO Branch and bound, we will Consider the Variable tuple Size formulation. The state space tree can be drawn as above.

→ For node u=24, the upper is now 24. Then nodes 2,3,4 and 5 are generated u(2)=19, u(3)=14, u(4)=18, u(5)=21. The Minimum Value of u(x) will be upper.

Hence upper will be updated to 14.

upper = 14
For node 4
$c^{\wedge}(4) = 15$
$c^{\wedge}(4) >$ upper
i.e 15 > 14
Hence kill node 4

upper = 14
For node 5
$c^{\wedge}(5) = 21$
$c^{\wedge}(5) >$ upper
i.e 21 > 14
Hence kill node 5

→ Node 2 and 3 become live nodes. So, Now we will Consider 2 and 3. we will expand node 2 (2 become E-node) u(6)=9, u(7)=13, u(8)=16. Hence Minimum of u(x) becomes upper. As u(6)=9 is Minimum. The upper will be updated to 9.

Hence,

upper = 9
For node 7
$c^{\wedge}(7) = 10$
$c^{\wedge}(7) >$ upper
Hence kill node 7.

upper = 9
For node 8
$c^{\wedge}(8) = 16$
$c^{\wedge}(8) >$ upper
Hence node 8 is not Considered.

→ The live node 3 being live node becomes E-node now.

Now children 9 and 10 are generated. $u(9) = 8$ and $u(10) = 11$. Hence upper is updated to 8.

Hence,

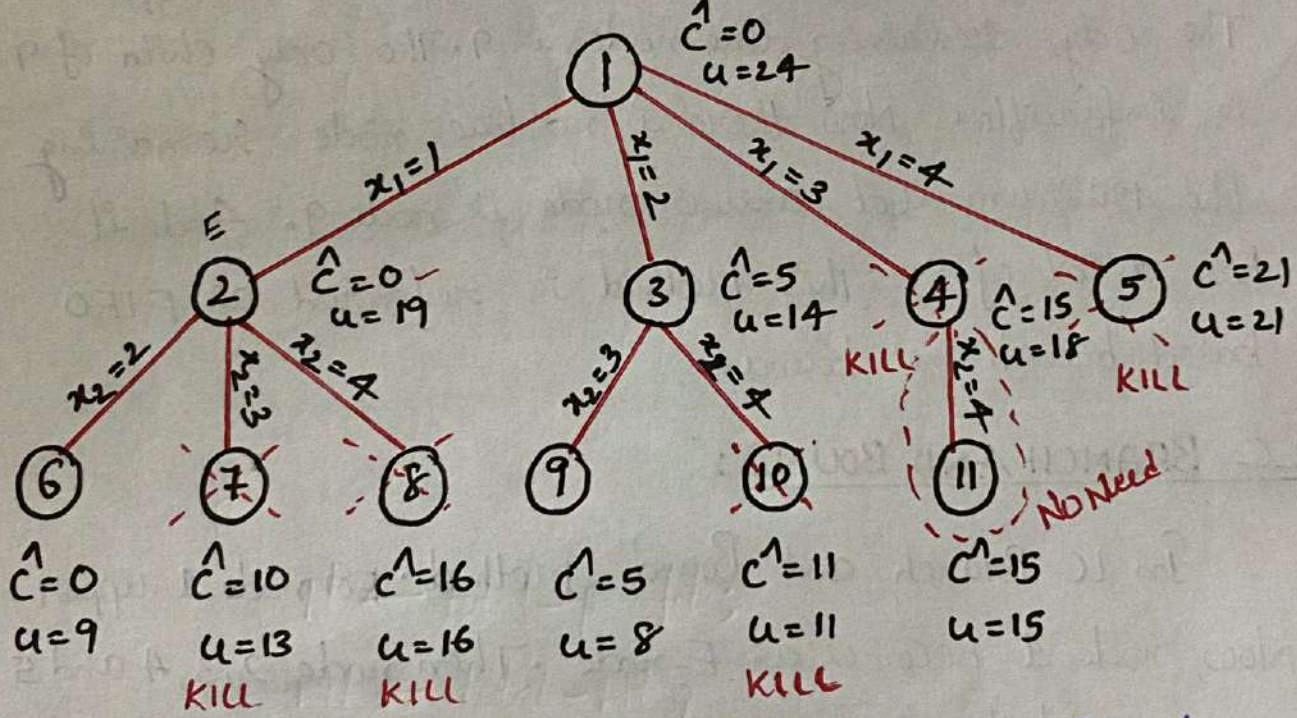
upper 8
For node 10
$c^{\wedge}(10) = 11$
$c^{\wedge}(10) >$ upper
Hence kill node 10.

→ Now node 6 becomes E-node. But as children of node 6 are infeasible we will not Consider node 6. The only remaining live node is 9. The only child of 9 is infeasible. Now there is no live node remaining the Minimum Cost answer node is node 9. And it has a cost of 8. This Method is referred as FIFO based branch and bound.

LC BRANCH AND BOUND:

In LC Branch and Bound Method. For node 1 upper = 24 Now, node 1 becomes an E-node. Then node 2, 3, 4 and 5 are generated. The upper is now 14. As $\hat{c}(4) >$ upper and $c^{\wedge}(5) >$ upper. Hence node 4 and 5 are Killed. For node 2 and 3 the cost $c^{\wedge}(2)$ is 0, node 2 becomes an E-node. Hence children node 6, 7 and 8 are generated The upper is Now 9 (because $u(6) = 9$). Node 7 and 8 are killed. Node 6 is Selected as it has Minimum Cost.

But Both the children of node 6 are infeasible. So kill node 6. Now, node 3 becomes E-node. Then node 9 and 10 are generated. The upper = 8 now, Since $\hat{c}(10) > 7$ upper we will kill node 10. Now only remaining node is 9. Next E-node becomes node 9. Its only child is infeasible. As there are no live nodes remaining, we will terminate Search with node 9 as an Answer node. The principle idea in LC Search is Choosing of Minimum cost node each time.



This Method of LC based branch and Bound with appropriate $\hat{c}(\cdot)$ and $u(\cdot)$ is called as LCBB (LIFOBB)

0/1 KNAPSACK PROBLEM:

PROBLEM STATEMENT:

The 0/1 knapsack problem states that - There are 'n' objects given and Capacity of knapsack is 'm'. Then select some objects to fill the knapsack in such a way that it should not exceed the capacity of knapsack and maximum profit can be earned. The knapsack problem is a Maximization problem. That means we will always seek for Maximum $P_i X_i$ (where p represents profit of object x_i). We also get $\sum P_i X_i$ Maximum iff $-\sum P_i X_i$ is minimum.

Minimize profit $-\sum_{i=1}^{n} P_i X_i$

Subject to $\sum_{i=1}^{n} W_i X_i$

Such that $\sum W_i X_i \leq m$ and

$x_i = 0$ and 1 where $1 \leq i \leq n$

We will discuss the Branch and Bound strategy for 0/1 knapsack problem using fixed tuple size formulation. We will design the state space tree and compute $c\hat{}(\cdot)$. and $u(\cdot)$ where $c\hat{}(x)$ represents the approximate cost used for computing the least cost $c(x)$.

Clearly $u(x)$ denotes the upper bound. As we know upper bound is used to kill those nodes in the state space tree which can not lead to the Answer node.

Let, X be the node at level j. Then we will draw the state space tree for fixed tuple formulation having levels $1 \le j \le n+1$.

Then we need to Compute $c^{\wedge}(x)$ and $u(x)$. Such that $c^{\wedge}(x) \le c(x) \le u(x)$ for every node.

ALGORITHM:

The algorithm for Computing $c^{\wedge}(x)$

```
Algorithm C_Bound (total profit, total Wt, k)
// total profit denotes the coorent total profit.
// total profit wt denotes the coorent total weight.
// K is the Index of last removed object.
// W[i] represents the weight of object i
// P[i] represents the profit of object i
// m is the Weight Capacity of Knapsack
 {
    Pt ← total_profit;
   Wt ← total - Wt;
    for (i ← k+1 to n) do
     {
        wt ← wt + W[i];
          if (wt<m) then pt ← Pt + P[i];
    else return (pt + (1-(wt-m)/w[i] * P[i]);
     }
     return pt;
 }
```

1. Draw State Space tree
2. Compute $c^{\wedge}(\cdot)$ and $u(\cdot)$ for each node.
3. If $c^{\wedge}(x) >$ upper kill node x.
4. Otherwise the Minimum cost $c^{\wedge}(x)$ becomes E-node. Generate children for E-node.
5. Repeat step 3 and 4 until all the nodes get Covered.
6. The Minimum Cost $c^{\wedge}(x)$ becomes the answer node. Trace the path in backward direction from x to root for solution subset.

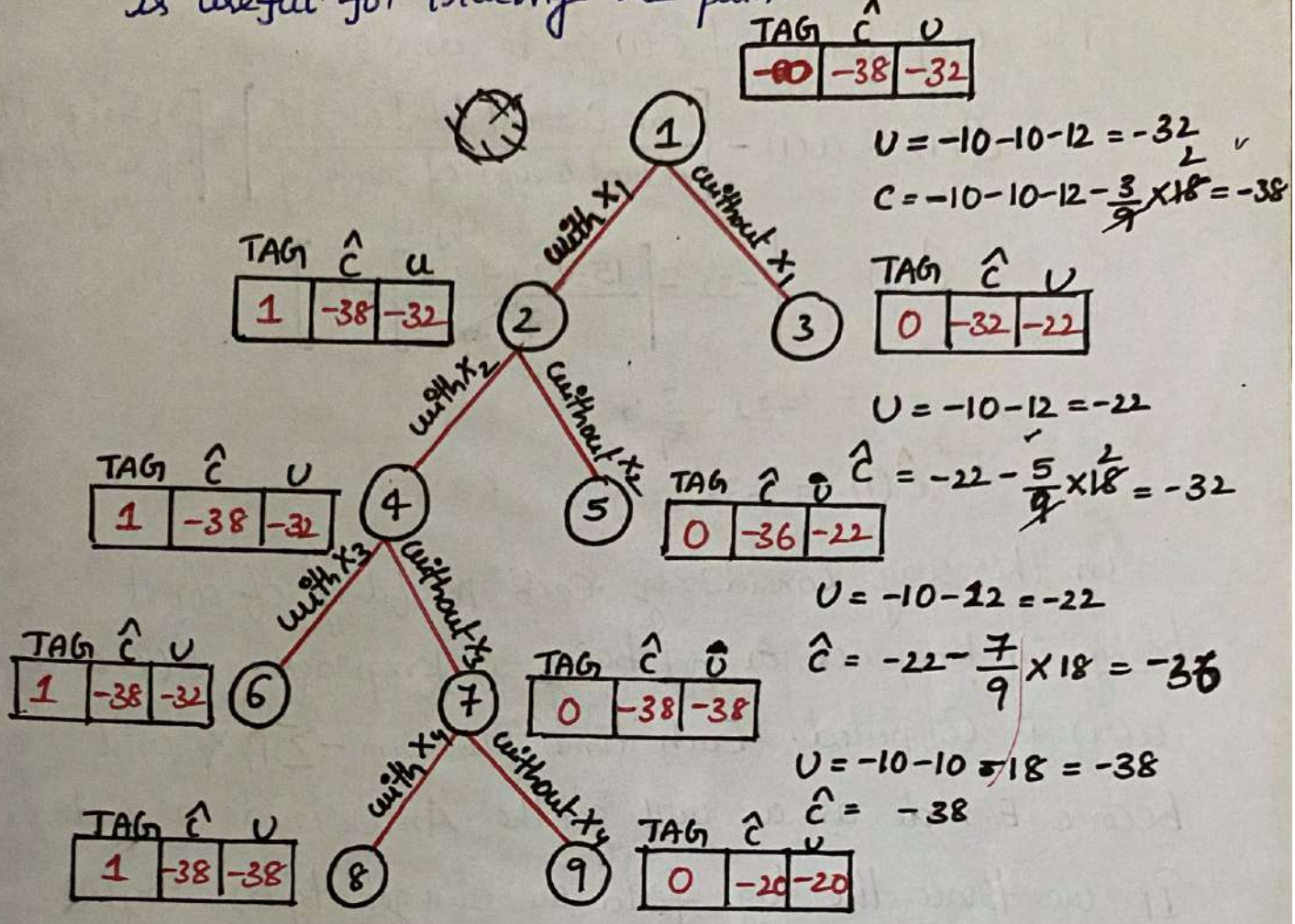
EX: Consider Knapsack instance $n = 4$ with capacity $m = 15$. Such that

Object i	P_i	W_i
1	10	2
2	10	4
3	12	6
4	18	9

→ Let us design state space tree using fixed tuple size formulation. The Computation of $c^{\wedge}(x)$ and $u(x)$ for each node x is done.

At each node a structure is drawn in which Computation of $c^{\wedge}(\cdot)$ and $u(\cdot)$ is given. The tag field

is useful for tracing the path.



1101

LCBB Solution Space tree

Using algorithm U-Bound $U(x)$ is Computed and using C-Bound $\hat{c}(x)$ is Computed.

$U(1)$ Can be Computed as

for $i = 1, 2,$ and 3

$$\therefore -\sum P_i = -(10+10+12)$$

$$U(1) = -32$$

If we Select $i = 4$, then it will exceed Capacity of Knapsack.

The Computation of $\hat{c}(1)$ Can be done as.

The Computation of $\hat{c}(1)$ can be done as

$$\hat{c}(1) = u(1) - \left[\frac{m - \text{Current total weight}}{\text{Actual weight of remaing object}}\right] * \left[\text{Actual profit of remaining object}\right]$$

$$\hat{c}(1) = \left[-32 - \left[\frac{15-(2+4+6)}{9}\right] * 18\right]$$

$$= -32 - \frac{3}{9} * 18$$

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

In this way Considering each possibility of object being in Knapsack & not being in Knapsack. $\hat{c}(x)$ and $u(x)$ is Computed. Each time Minimum $-\sum P_i X_i$ will become E-node an we will get the Answer node as node 8. IF we trace the tag field we will get tag(2) - tag(4) - tag(7) - tag(8). i.e 1101. Hence $X_4 = 1$, $X_3 = 0$, $X_2 = 1$ and $X_1 = 1$. we will select object X_4, X_2 and X_1 to fill up the Knapsack and gain Maximum profit.

FIFO BRANCH AND BOUND SOLUTION:

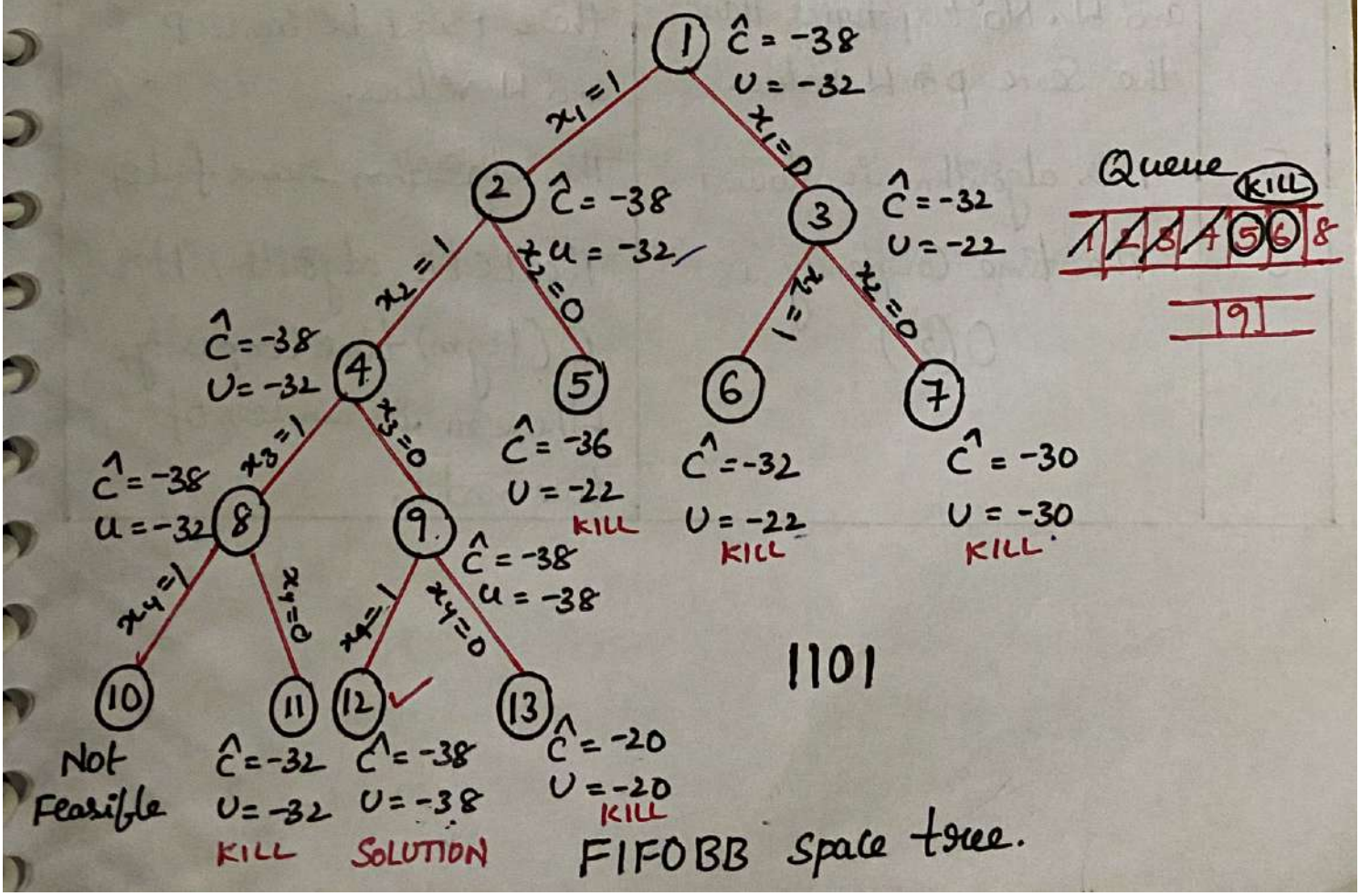
→ The Space tree with Variable tuple Size formulation Can be drawen and $\hat{c}(\cdot)$ and $u(\cdot)$ is Computed (we Have Considered the Same Knapsack problem which is discussed earlier)

→ Intially upper = $U(1) = -32$. Then children of node 1 are generated. Node 2 becomes E-node and Hence 4 and 5 are generated. Node 4 and Node 5 are added in the list of live nodes.

→ Next; node 3 becomes E-node and children 6 and 7 are generated. As $\hat{c}(7) >$ upper we will kill node 7. Hence node 6 will be added in the list of live nodes. Node 4 is E-node and children 8 and 9 are generated. The upper is updated and it is Now upper $= u(9) = -38$.

→ Nodes 8 and 9 are added in the list of live nodes. Node 5 and 6 becomes the next E-node but as $\hat{c}(5) >$ upper and $\hat{c}(6) >$ upper, Kill nodes 5 and 6. Node 8 becomes Next E-node and children 10 and 11 are generated.

→ As node 10 is infeasible do not Consider it. $\hat{c}(11) >$ upper. Hence kill node 11. Node 9 becomes Next E-node and upper $= -38$. Children 12 and 13 are generated. But $\hat{c}(13) >$ upper. So kill node 13. finally, node 12 becomes an Answer node. Therefore Solution is $X_1 = 1$, $X_2 = 1$, $X_3 = 0$ and $X_4 = 1$.



Difference between Dynamic and BB Knapsack:

S.No	DYNAMIC KNAPSACK	BRANCH AND BOUND KNAPSACK
1.	In Dynamic Knapsack method, the principle of optimality is used.	In Branch Bound Method a state space tree is built using bounding functions.
2.	A Sequence of decision $S^{(i)}$ is made, While Solving the problem using this option.	A root node is further expanded using E node.
3.	By Merging and purging rules the pairs (P,W) can be eliminated.	IF particular node posses $\hat{c}(\cdot)$ Value which is greater than upper than the node is not expanded further.
4.	In Dynamic programming the ordering is done using P and W. No two tuples have the same P or W value.	IN FIFO B.B, Many nodes may lie on the same level and there might be same P and W values.
5.	This algorithm is slower	The algorithm runs faster
6.	The time Complexity is $O(2^n)$	The LCBB algorithm Has $O(\log m)$ time Complexity Where m is number of live nodes.

TRAVELLING SALES PERSON (TSP):

PROBLEM STATEMENT:

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

→ Tour(x) is the path that begins at root and reaching to node x in a state space tree and returns to root. In branch and bound strategy cost of each node x is computed. The travelling salesperson problem is solved by choosing the node with optimum cost.

Hence $C(x)$ = cost of tour(x)

The $\hat{C}(x)$ is the approximation cost along the path from the root to x.

→ Consider graph G(V,E) of a weighted graph where

V = Set of vertices

E = Set of Edges

→ As per the Branch and Bound, the given graph must be converted into equivalent adjacency cost Matrix.

→ Check whether the Matrix what we developed is either reduced Matrix (or) Not.

REDUCED MATRIX:

If a Matrix Contains atleast one '0' either in all rows and in all Column's then it is said to be as "Reduced Matrix".

→ If it is not reduced, then by following below two steps we can make it is reduced.

(i) Row Minimization

(ii) Column Minimization.

ROW MINIMIZATION:

To understand Solving of travelling salesperson problem using branch and bound approach we will reduce the cost of the cost Matrix M, by using following formula.

$$\text{Red-Row}(M) = M_{ij} - \min\{M_{ij} / 1 \le j \le n\}$$

where $M_{ij} < \infty$

COLUMN MINIMIZATION:

Now we will reduce the Matrix by choosing Minimum from each column. The formula for column reduction of Matrix is

$$\text{Red-Col}(M) = M_{ji} - \min\{M_{ji} / 1 \le i \le n\}$$

where $M_{ji} < \infty$

Ex: Consider the Matrix M representing Cost between any two cities

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \end{array}$$

→ Check whether it is reduced (or) Not

→ If Not then 1, Row reduction

We will find Minimum of each row

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

21 total reduced cost

We will Subtract the row-Minimum value from Corresponding row. Hence,

$$\text{Red\_Row}(M) = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

Red_Col(M) be obtained as,

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

0 / Ignore

↑ ↑ ↑ ↑ ↑

1 0 3 0 0 total = 4

If row & column Contains at least one zero ignore Corresponding row & column.

$$\text{Red\_Col}(M) = \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Thus total reduced cost will be

$$= \text{Cost}(\text{Red-Row}(M)) + \text{Cost}(\text{Red-Col}(M))$$

$$= 21 + 4$$

$$= \underline{25}$$

Thus

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 1 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \xrightarrow{\text{Fully Reduced Matrix}} \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

DYNAMIC REDUCTION:

We obtained the total reduced cost as 25. That means all tours in the original graph have length at least 25.

using dynamic reduction we can make the choice of edge $i \to j$ with optimum cost.

Steps in Dynamic reduction technique:

1. Draw a state space tree with optimum cost at root node.
2. Obtain the cost of Matrix for path $i \to j$ by making i^{th} row and j^{th} column entries as ∞. Also set $M[j][i] = \infty$
3. Cost of corresponding node x with path i, j is optimum cost + reduced cost + $M[i][j]$
4. Set node with Minimum cost as E-node and generate its children. Repeat step 1 to 4 for Completing tour with Optimum cost.

Ex: The edge length of a directed graph are given by the below Matrix. using the traveling Salesperson algorithm, Calculate the optimal tour.

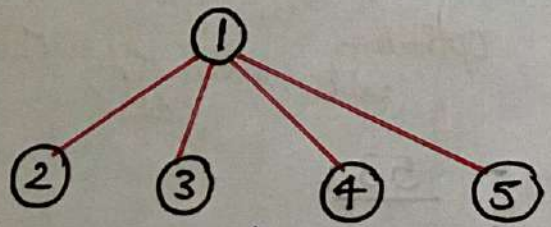
$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

→ Fully reduced Matrix is,

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

Optimum cost = 21 + 4 = 25

→ As we start from ① Node we can visit remaing Nodes like ②, ③, ④, ⑤



→ Consider path (1,2) Make 1st row and 2nd colum ∞. And Set M[2][1] = ∞

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

→ Ignore
→ Ignore
→ Ignore
→ Ignore
→ Ignore

↓ ↓ ↓ ↓ ↓

Ignore Ignore Ignore Ignore Ignore

Cost of node 2 is

$= 25 + 0 + 10$

↑ Optimum Cost ↑ Reduced cost ↑ old value of M[1][2]

$= \underline{35}$

Consider path (1,3). Make 1st row = ∞ • 3rd column = ∞ and M[3][1] = ∞

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

↓ ↓ ↓ ↓ ↓

11 1 1 1 1

Cost of node 3 is

$= 25 + 11 + 17$

↑ Optimum Cost ↑ Reduced cost ↑ M[1][3]

$= \underline{53}$

Consider path (1,4).

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

Cost of node 4 is = 25+0+0

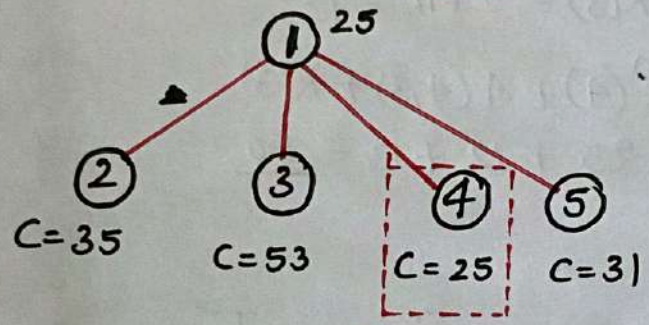
$= \underline{25}$

Consider path (1,5)

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

Cost of node 5 is = 25+5+1

= 31



Consider all the nodes, ④ is least cost, so find path from that node By following Branch and Bound principle.

→ As we start from ① and next ④, and we are moving from ④, so Consider (1,4) path Matrix as 'A_2'

$$\therefore A_2 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Consider path (4,2)

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

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

$$= 25 + 3 + 0 = 28$$

Consider path (4,3)

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

⑪ Row reduction Colum reduction

then $R(3) = 2+11 = 13$

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

Consider path (4,5)

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

Row reduction

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

Consider all the nodes, ② is least cost so find path from ② By following Branch and Bound principle.

So Consider (4,2) path Matrix as A_3

$$\therefore A_3 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

Consider path (2,3)

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

(11 ↑)

Row reduction — Colum reduction

$$R = 2 + 11 = 13$$

$$\hat{c}(3) = A(2,3) + R + \hat{c}(2)$$

$$= 11 + 13 + 28 = \underline{52}$$

Consider path (2,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \Rightarrow \hat{c}(5) = \hat{c}(2) + R + A(2,5)$$

$$= 28 + 0 + 0$$

$$= \underline{28}$$

Consider all the nodes, (5) is least cost so find path from (5) By following Branch and Bound principle.

So Consider (2,5) path Matrix as A_4

$$\therefore A_4 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

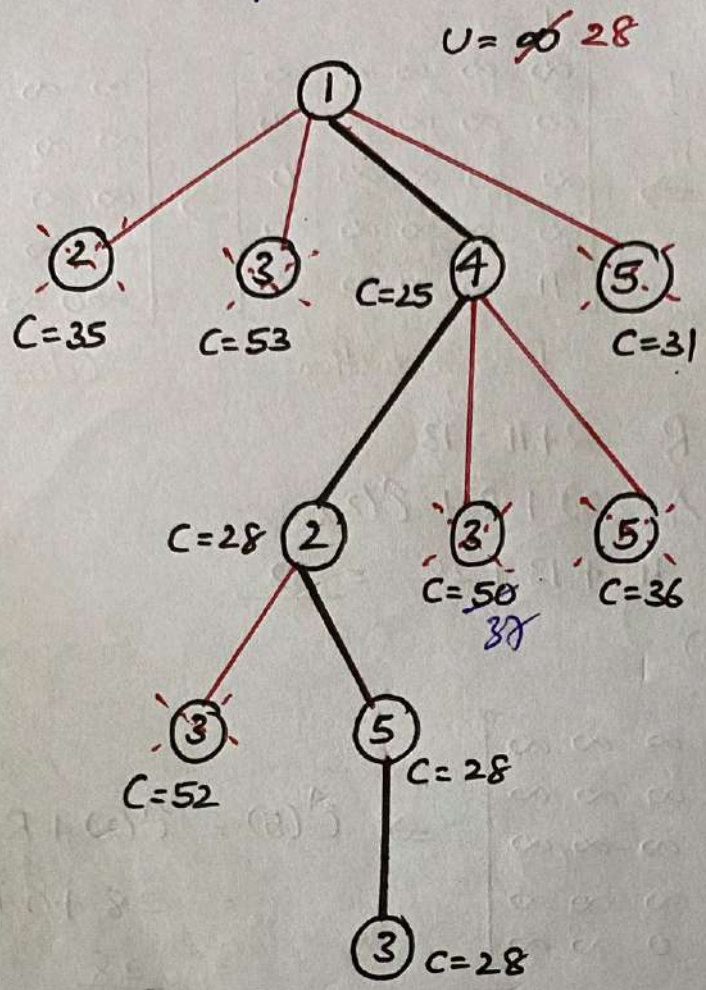
Consider path (5,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \Rightarrow \hat{c}(3) = A(5,3) + R + \hat{c}(5)$$

$$= 0 + 0 + 28$$

$$= \underline{28}$$

final state space tree is



∴ The path is 1-4-2-5-3-1

Cost = 28