MODULE - 05

Backtracking: Branch and-bound algorithms are used to solve combinatorial problems. It is an improvement over exhaustive search.

The solutions are constructed one component at a time. Partially constructed solutions are evaluated.

If no potential values of remaining components can lead to a solution, they are not generated.

The solutions are constructed with the help of state-space tree.

State-Space Tree:

- Root represents initial state before the search for the solution begins.
- The nodes in first level in the tree represents the choice made for first component, second for second component and so-on.
- A node in state-space tree is said to be promising if it leads to a solution. Other wise it is a non-promising node.

- leaves represent either non-promising dead ends or the solution found by the algorithm.

- State-Space tree is constructed in the manner of depth-first search (DFS).

- If current node is promising, its child is generated by adding remaining legitimate options.

- If current node is non-promising, the algorithm backtracks to node's parent to consider next possible option.

- If algorithm reaches a complete solution to problem, it either stops or continues searching another solution.

N-Queens Problem

The N-Queens problem is to place n queens on an n-by-n chessboard so that no two queens attack each other by being in the same row, column or diagonal.

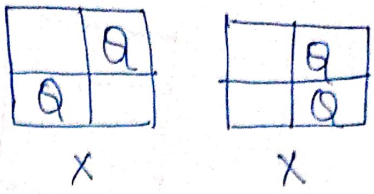
for n=1 → 1 Queen

→ 1 X 1 chess board

↳ solution

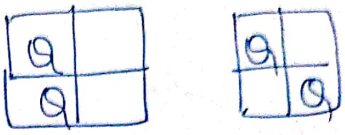
for n=2 → 2 Queens

2x2 chess board



X X

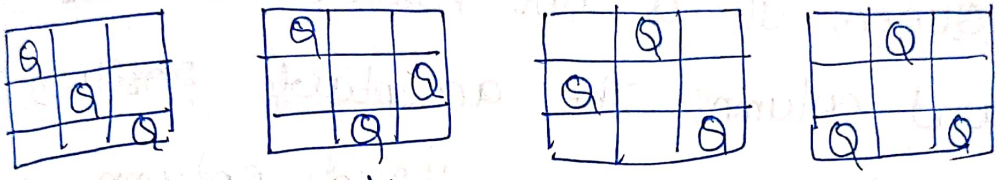
NO - solution.



X X

for n=3 → 3 Queens

→ 3x3 Chessboard

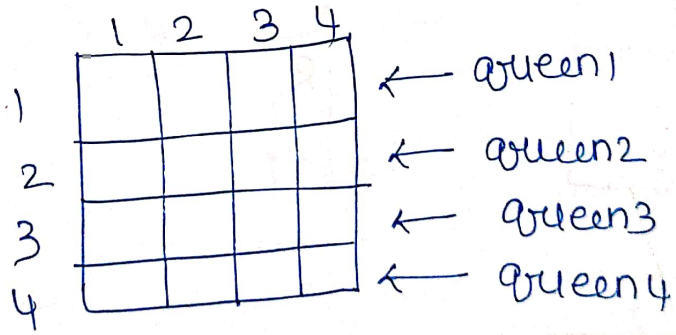


X X X X

NO-solution.

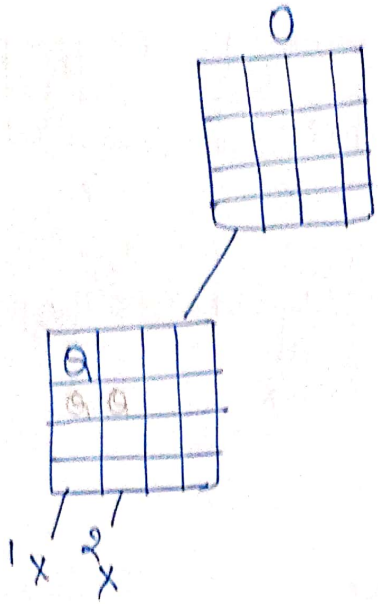
4-Queen's Problem

The objective is to place 4 queens in 4x4 chess board

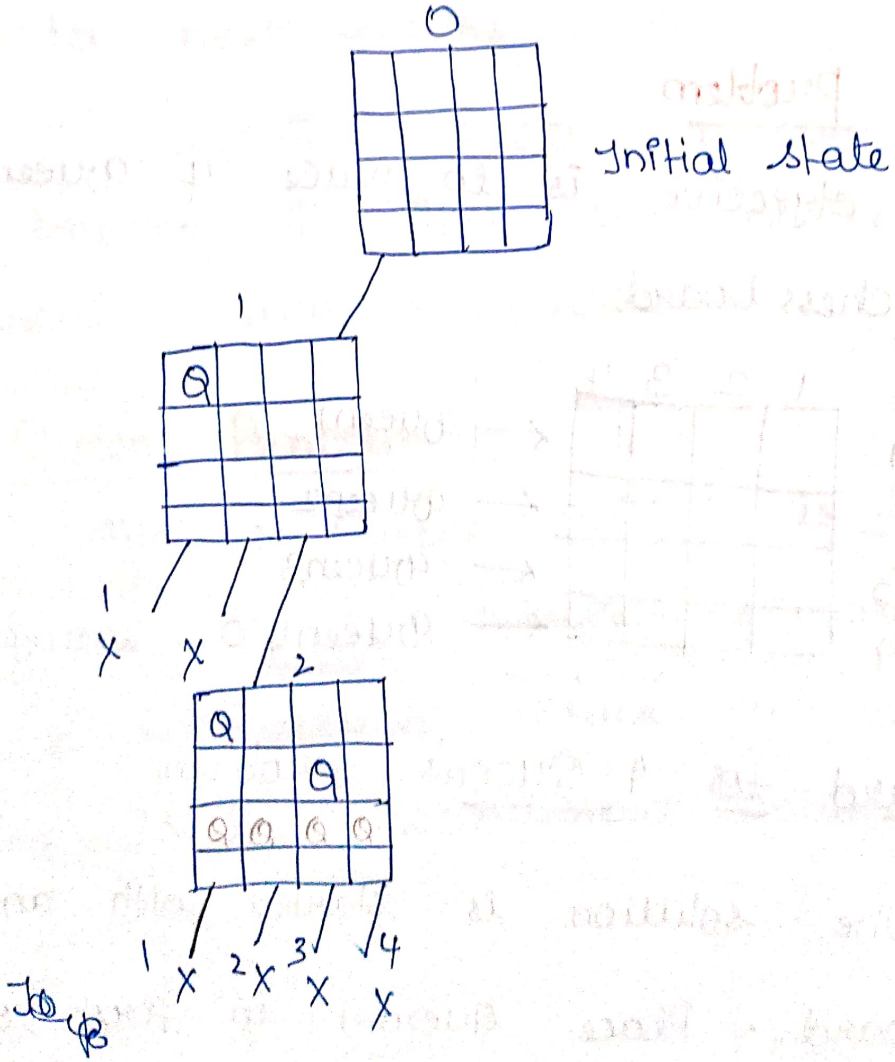


Board for 4-Queen's Problem.

The solution is started with an empty board. Place Queen 1 in first possible Place i.e; 1 (1 row, 1 column)



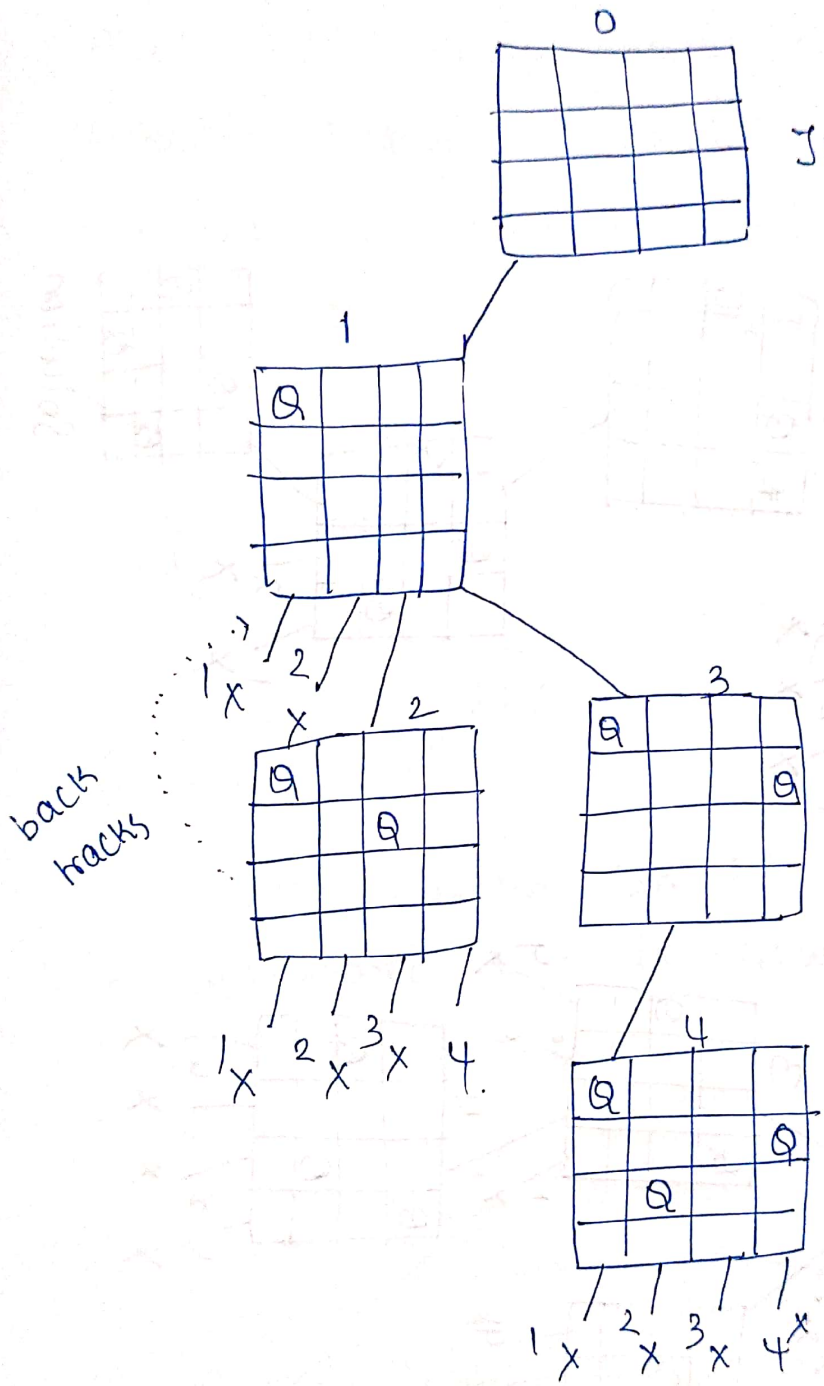
To place 2 Queen, It is not possible in ~~seco~~ 1 and second columns. The acceptable position is square (2,3) - second row third column.



To ~~p~~

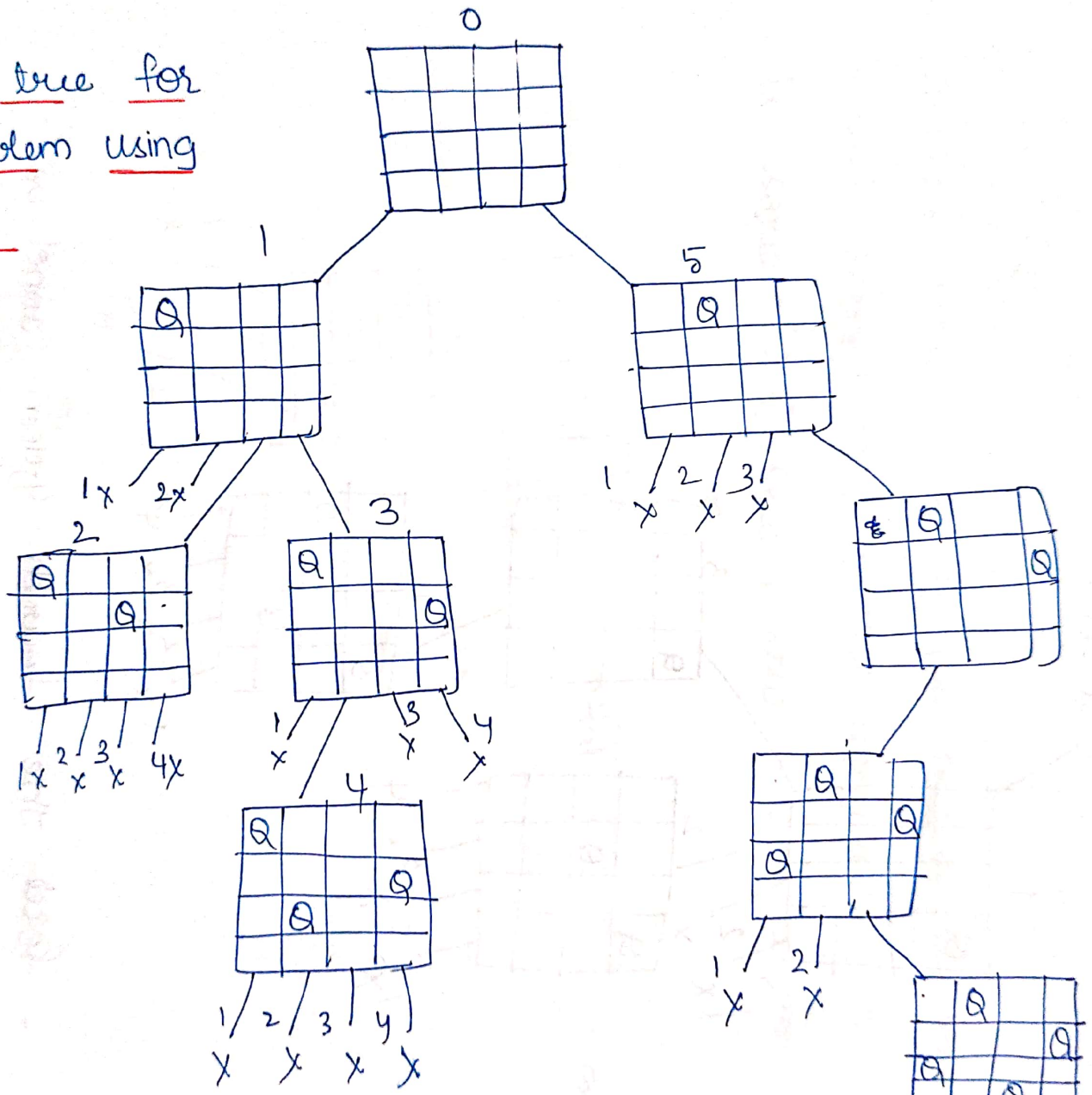
Third Queen cannot be placed because it leads to a dead end.

So, the algorithm back-tracks and puts Queen 2 in next possible slot i.e; (2,4)



- ~~Next~~ The fourth queen cannot be Placed and it backtracks to 1 and moves Queen's-1 position to try next possibility and the state-space tree is given as follows:

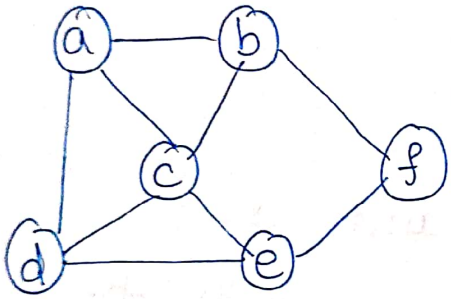
State-space tree for 4-Queen problem using Backtracking



Hamiltonian Circuit problem

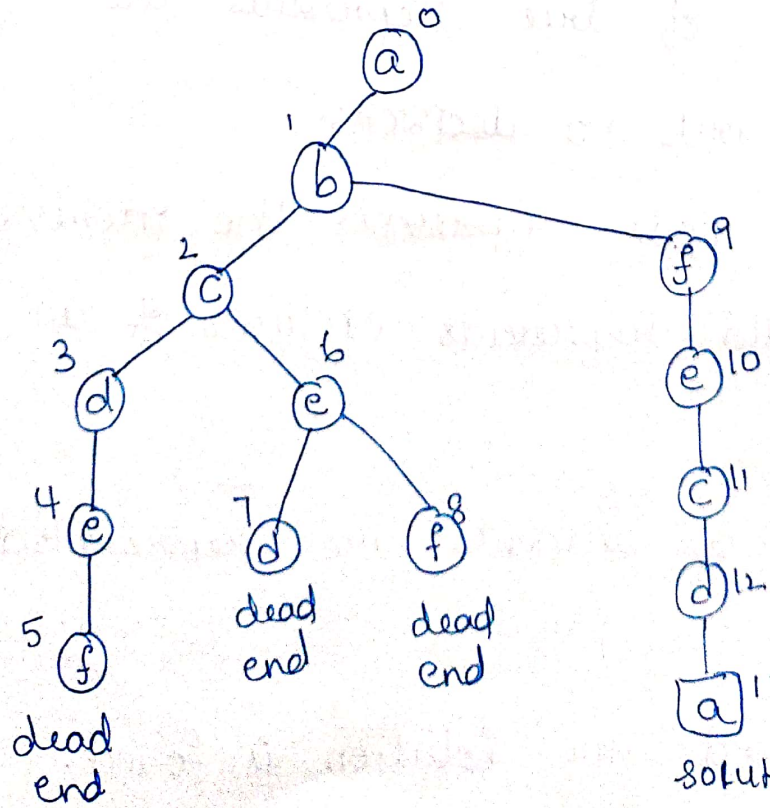
A Hamiltonian circuit is a circuit that visits every vertex exactly once with no repeats. Being a circuit it must start and end at the same vertex.

Hamiltonian Circuit problem is to find Hamiltonian circuit in a given graph.



Starting vertex = 'a'

This problem can be solved using Backtracking. The state-space tree is given as follows:



State-space tree

Subset-sum problem

Subset-sum problem is to find a subset of a given set $S = \{s_1, s_2, s_3 \ldots s_n\}$ of 'n' positive integers whose sum is equal to a given positive integer 'd'.

Ex:1 $S = \{1, 2, 5, 6, 8\}$ and $d = 9$. there are two solutions.

$\{1, 2, 6\}$ and $\{1, 8\}$.

Consider $s_1 \leq s_2 \leq s_3 \ldots s_n$.

Ex2: Given $S = \{3, 5, 6, 7\}$ and $d = 15$.

The solution can be constructed with the help of backtracking and a state-space-tree.

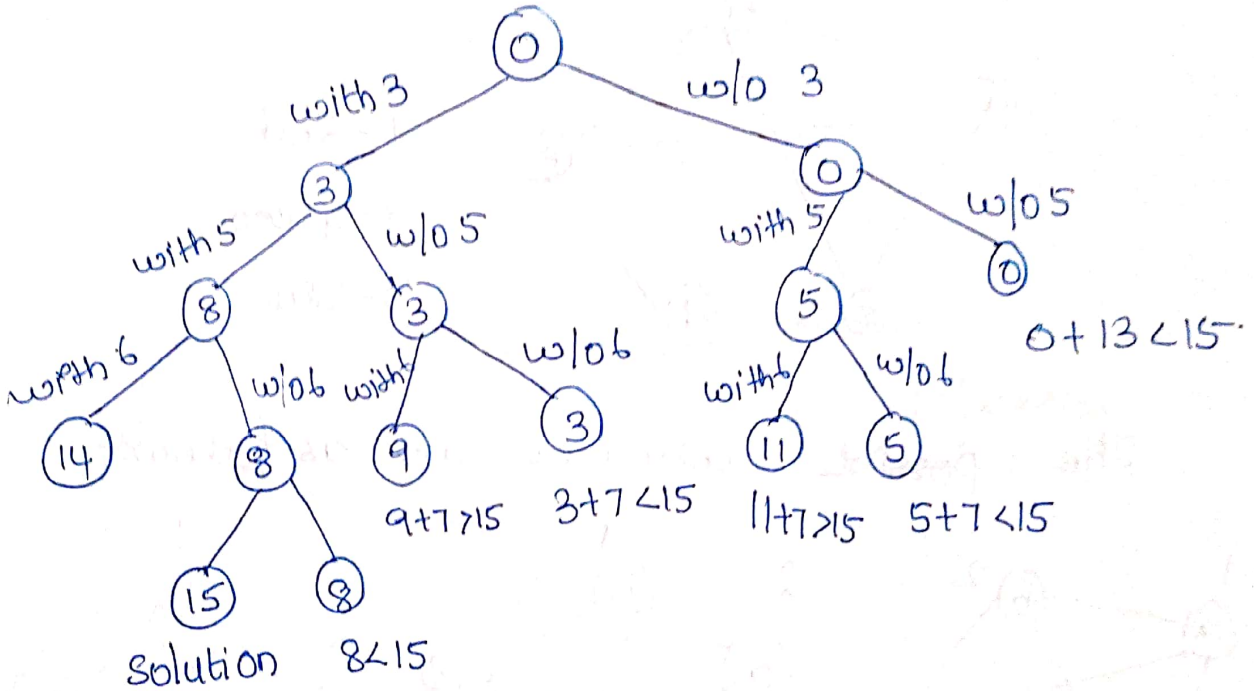
- The root of tree represents the starting point with no decisions.
- The left node represents the inclusion and right child represents exclusion of an element.
- The sum of elements are represented in the node.
- If sum = d the solution is found.

The node can be terminated as non-promising node based on two conditions.

$s' + s_{i+1} > d$ (sum is too large)

$s' + \sum_{j=1}^{n} s_j < d$ (sum is too small)

State-space tree:



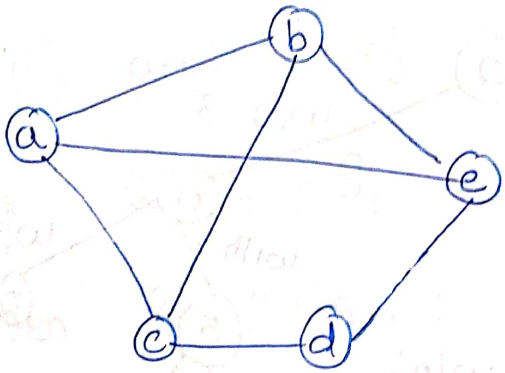
Graph Colouring - problem

Let G be the graph and 'm' be a given positive integer. The nodes of 'G' are coloured in such a way that, no two adjacent nodes have the same color by using m colours. This is termed as m-colorability Decision Problem.

- The m-colorability optimization problem asks for the smallest integer 'm' for which graph can be coloured.

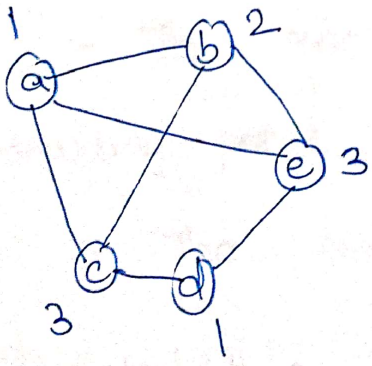
- This integer is referred to chromatic number of graph.

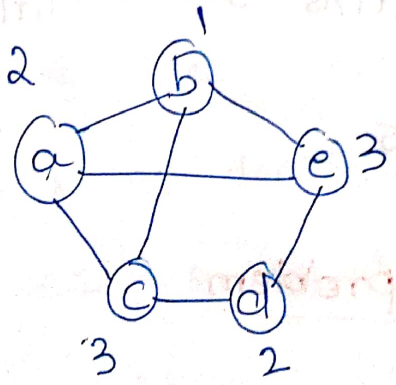
Ex: consider the following graph.

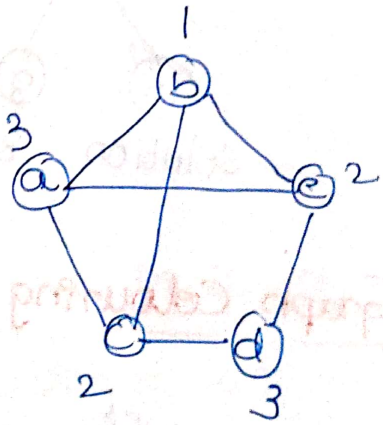


1 - Red
2 - Green
3 - Blue.

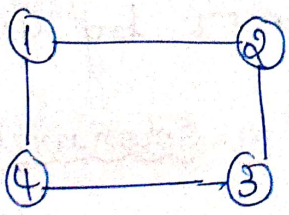
The possible solutions are as follows;



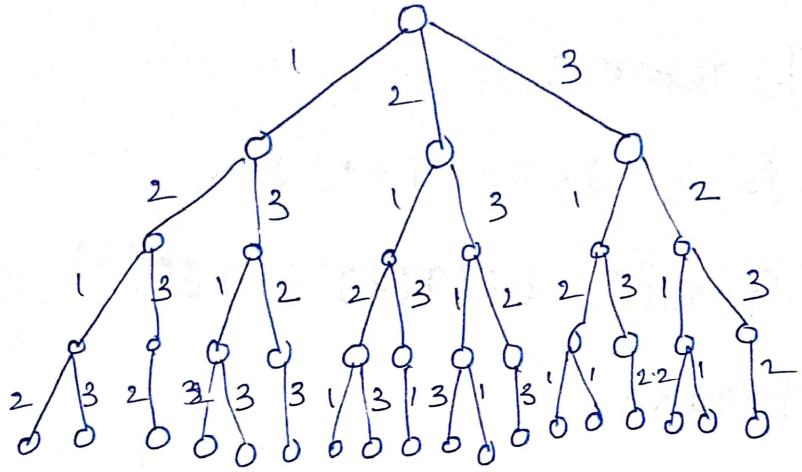




Ex2: Consider a graph with 4 nodes and m = 3.



The state-space tree for coloring the graph. i.e; all possible solutions are given as follows:



Pseudocode :

```
void m-coloring (int k) {
// Program implements backtracking
// Graph is represented with adjacency
   matrix G[1...n][1...n]

do {
      NextValue(k);
      if (!x[k]) break;
      if (k==n) {
            for (int i=1; i<=n; i++);
                  print x[i].
      }
      else mcoloring(k+1);
   } while (1);
}
```

```
void Nextvalue (int k) {
  do {
    x[k] = (x[k]+1) % (m+1);
    if (!x[k]) return;
    for (int j=1; j<=n; j++) {
        if (G[k][j] && (x[k] == x[j]))
            break;
    }
    if (j == n+1) return;
  } while (1);
}
```

Branch and Bound:

Compared to backtracking, branch-and-bound requires two additional items:

1. Bound on the best value of the objective function.
2. The value of best solution seen so far.

In general, the search path is terminated in a state-space tree of a branch-and-bound for the following reasons:

- The value of node's bound is not better than the value of the best solution.
- Node represents no solutions because constraints are already violated.

Assignment problem

Assignment problem is the task of assigning n people to n jobs so that total cost of assignment is as small as possible.

- select one element in each row of matrix so that no two selected items are in same column.

Ex:

	Job1	Job2	Job3	Job4	
	9	2	7	8	person a
	6	4	3	7	person b
	5	8	1	8	person c
	7	6	9	4	person d

The lower bound of the problem is calculated by adding the lower values in each row.

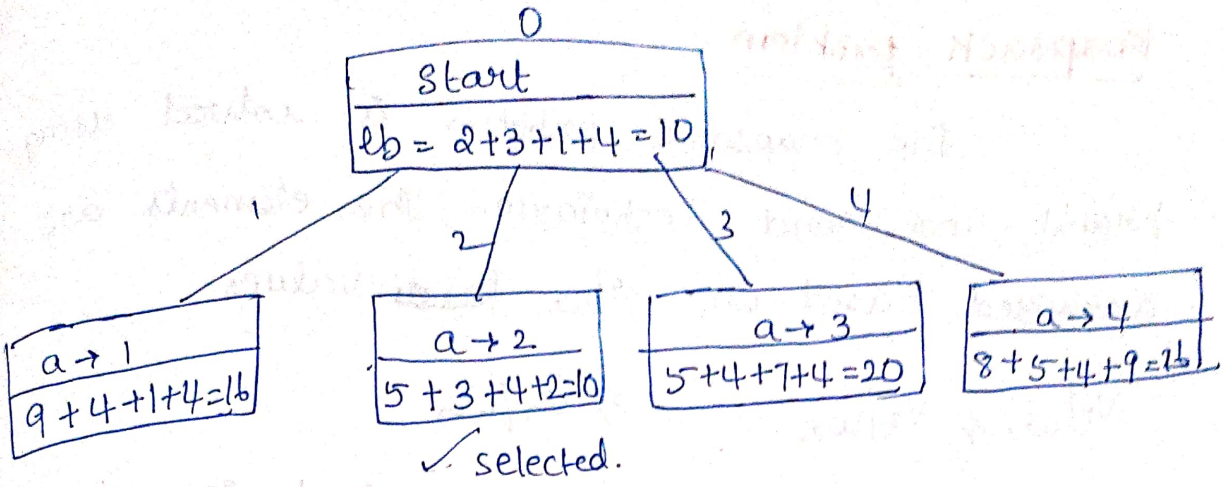
$= 2+3+1+4 = 10.$

The solution for the assignment problem can be constructed with the help of branch and-bound technique.

→ Starts with root where no elements are selected.

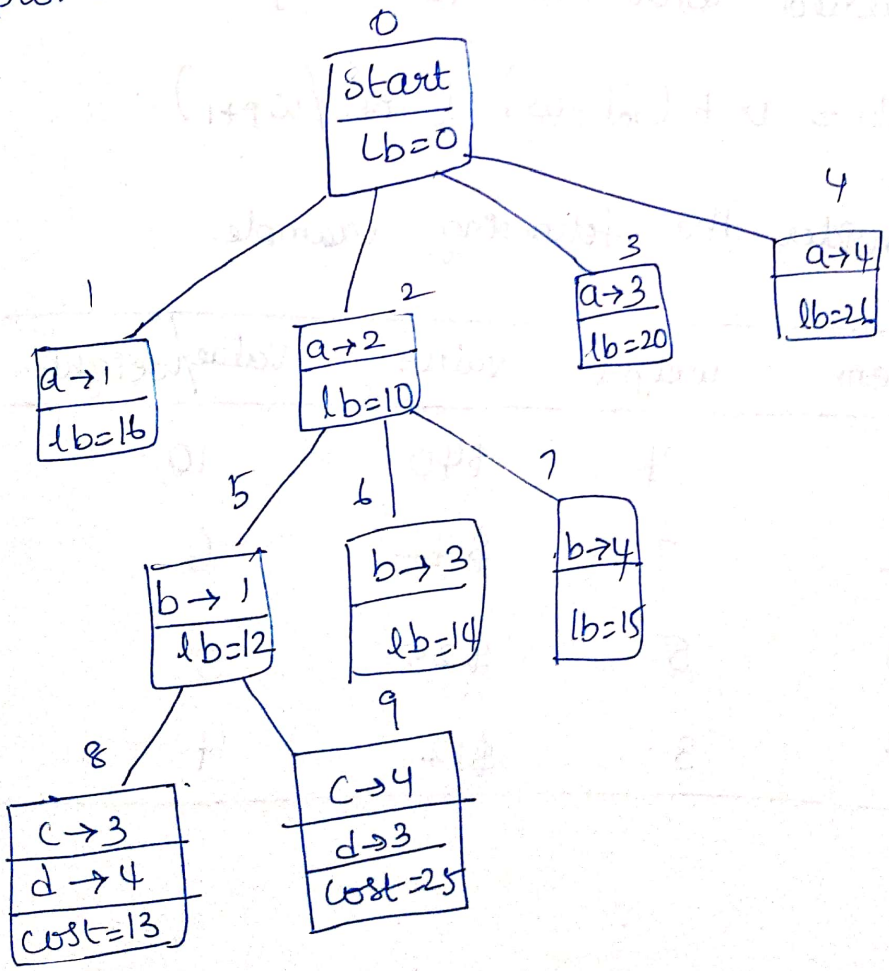
→ The nodes on the first level of tree corresponds to selection of an element in the first row of matrix.

→ The tree contains four leaves. A node from these is selected having lowest bound.



The Person a is given for Job 2 because that is giving the optimal solution.

from the previous selection, the nodes are explored



solution.

Knapsack problem :

The knapsack problem is solved using branch-and-bound technique. The elements are arranged based on v/w values.

$v_1/w_1 \geq v_2/w_2 \ldots\ldots \geq v_n/w_n$

A left branch indicates inclusion of an item and right branch indicates exclusion of an item.

The upper bound for the knapsack is calculated with the following formula:

$$ub = v + (W - w)(v_{i+1}/w_{i+1}).$$

Eg: Consider the following example.

Item	weight	value	value/weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

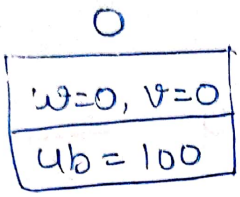
The knapsack capacity $W = 10$.

- At the root of state space tree, no items are selected.
- Hence $v = 0$ and $w = 0$.

The value of the upper bound is calculated as follows

$$ub = v + (w - w)\ (v_{i+1} / w_{i+1})$$

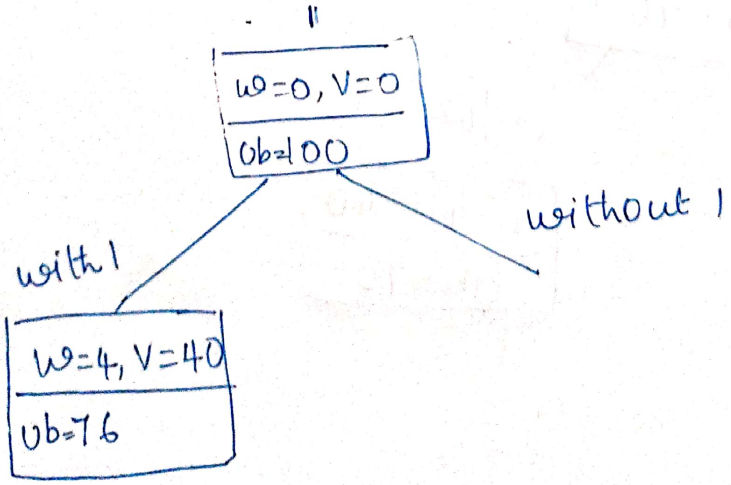
$$= 0 + (10 - 0) \cdot 10 \quad = 100$$



→ Node 1, ~~represents~~, the left child of the root represents the subsets that include item 1.

→ The total weight = 4 and value = 40. the upper bound is

$$U = 40 + (10 - 4) \cdot (6) \quad = 76.$$



→ It's children represents subsets with item 1 and, with and without item 2.

With item 2: w = 11 X exceeded the capacity

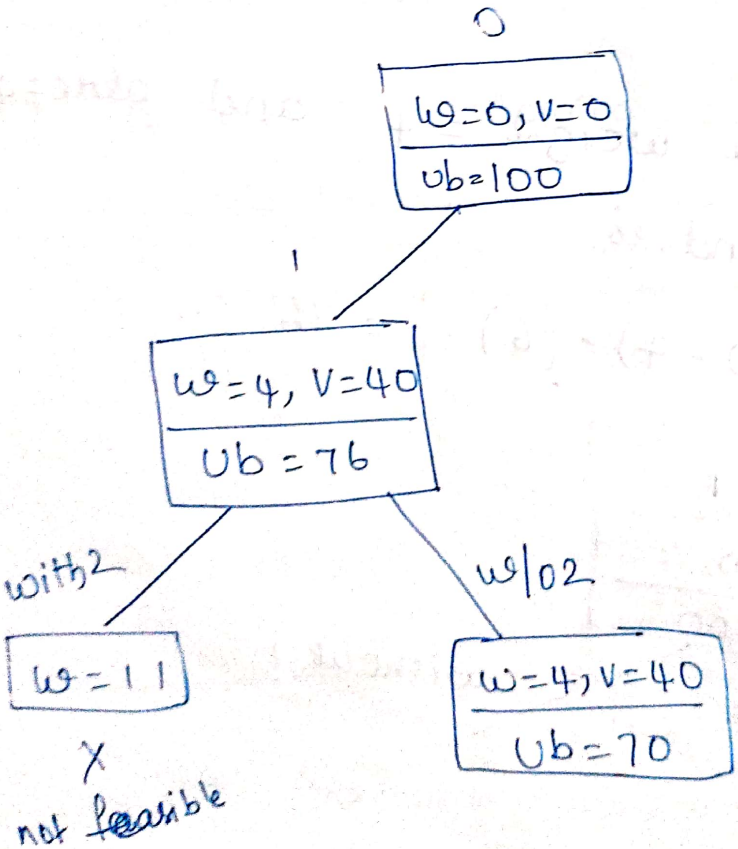
~~ub = 40 + (10-4)·5~~

without item2:

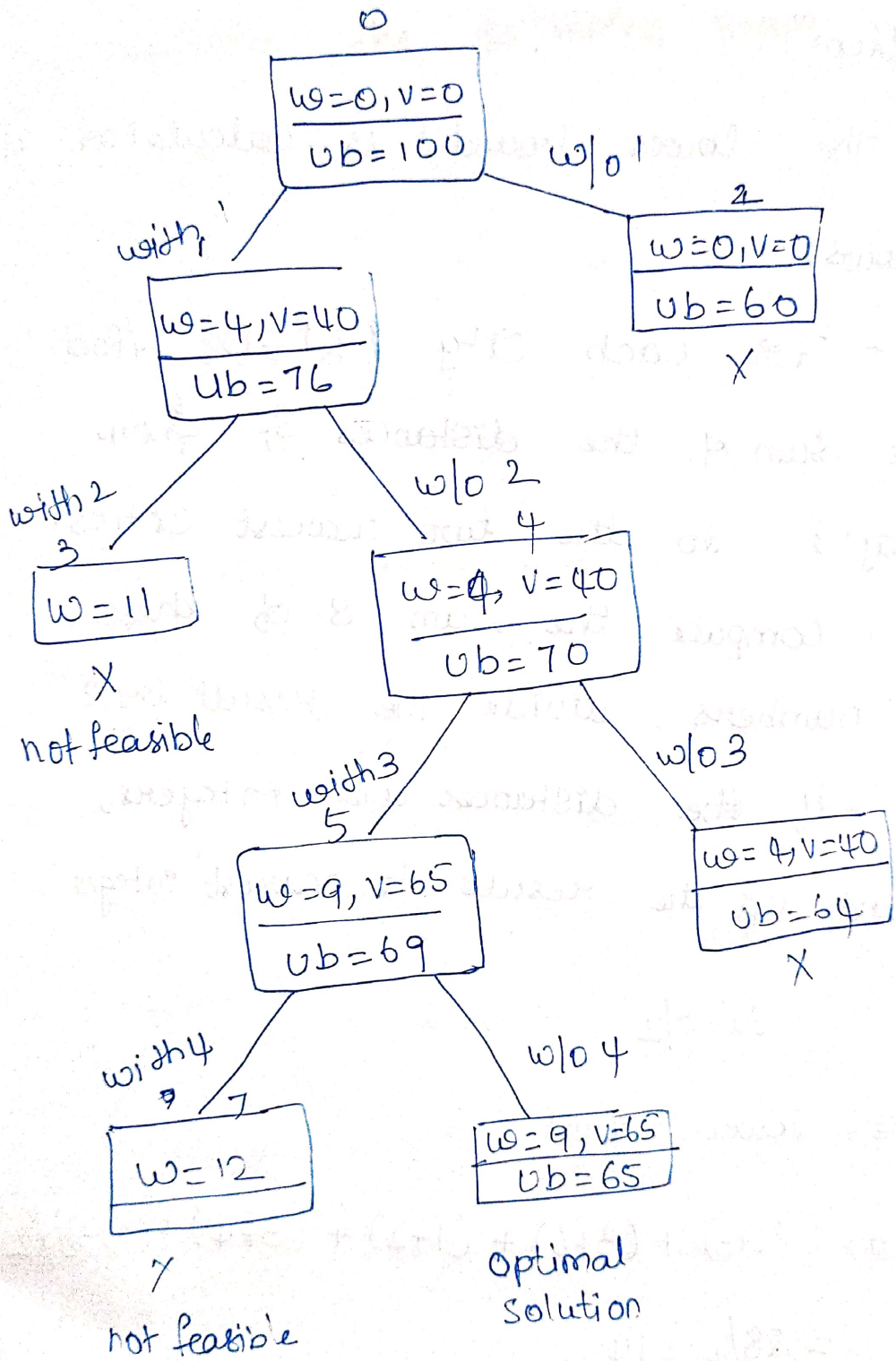
ub = 40 + (10-4)·5

= 70.

The nodes are updated as follows:



In the same way, all node possibilities are considered and the state space tree is constructed as follows:



The subset $\{1,3\}$ or $(1,0,1,0)$ is the optimal solution.

Travelling Salesman Problem:

Branch-and-Bound technique can be applied for "travelling salesman Problem".

The lower bound 'lb' is calculated as follows:

- for each city $1 \le i \le n$; find the sum of the distances s_i from city i to the two nearest cities.
- compute the sum s of these n numbers; divide the result by 2.
- if the distances are integers; round up the result to nearest integer

$$lb = s/2.$$

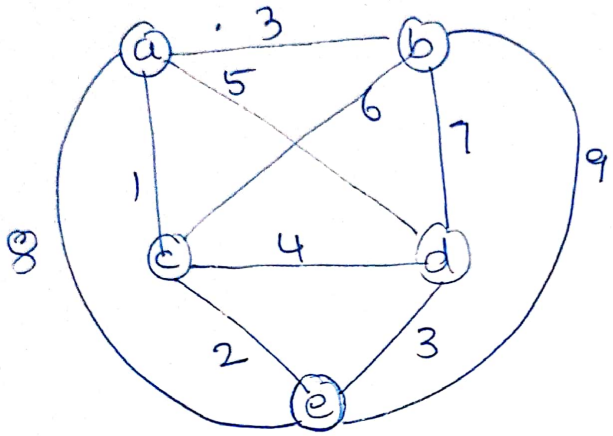
The lower bound is

$$lb = [(1+3) + (3+6) + (1+2) + (3+4) + (2+3)]/2$$

$$= 28/2 = 14.$$

- Moreover for any tour if it requires particular edge, the lower bound can include those edges.

Ex: consider the following graph.



The state space tree is constructed as follows: