3|5

UNIT - 4

BACK TRACKING :

In case of greedy & dynamic programming technique, we use Brute Force approach i.e., evaluting all possible solution for a given n value & then seleting 1 solution as optimal. In this we will get the same optimal solution with less than no. of n trials. It is more efficient compared to greedy & dynamic programming. In this, we use bounding functions (or, criterian Function)

1. Explicit constraints : The rules that restrict each x_i to take a values only from a given set

   Ex: $x_i \geq 0$ or $S_i = \{$all non-negative real numbers$\}$

   $x_i = 0$ or 1 or $S_i = \{0,1\}$

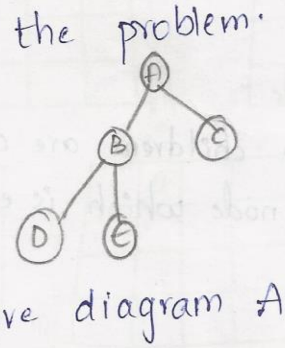
It completely depends upon the particular instance of the problem. All tuples must satisfy explicit constraints in order to define a possible solution.

2. Implicit constraints : Rules that determine which of tuples in solution space of i satisfies the criterian function. It describes the way the which x_i must relate to each other

   Ex: 0/1 knapsack problem.

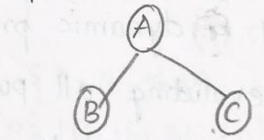
3. Criterian Function (Bounding Function) : It is a function $P(x_1, x_2 -- x_n)$ which needs to be maximized or minimized for a given problem.

4. Solution Space : All touples that satisfies the explicit constraints defines a possible solution for a particular instance for i in the problem.



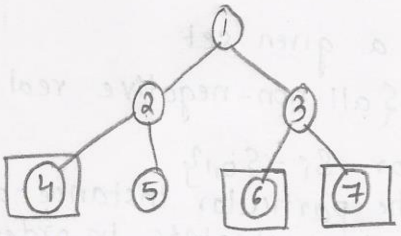
In above diagram ABD, ABE, AC are the tuples of solution space.

Problem State : Each node in the tree organization defines a problem state.



A, B, C are nodes of problem state.

a) Solution State : These are those problem state S for which the path from root to S defines a tuple in the Solution space. □ indicates the solution space.

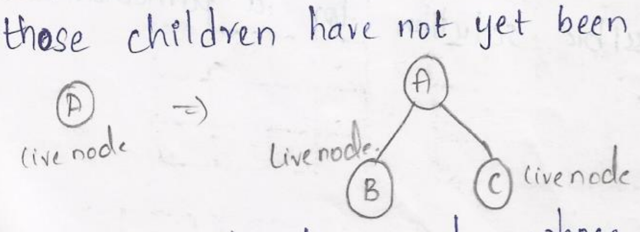


In above diagram there are 3 solution states which are represented in form of tuples i,e., (1,2,4), (1,3,6) (1,3,7)

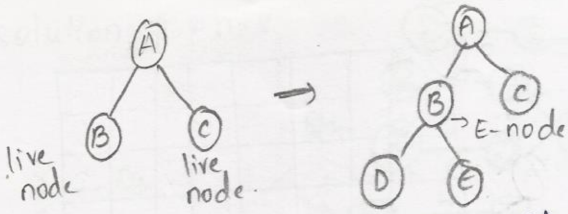
b) Answers State : The solution space 'S' for which the path from root to S defines tuple which is a member of set of solutions (which satisfies implicit constraints of the problem)

In above diagram, answer states are 4,6,7.

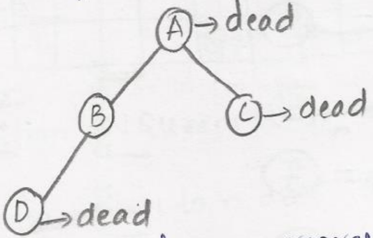
c) Live Node : A node which has been generated and all of those children have not yet been generated is called Live node.



d) E-node : The live nodes whose childrens are currently being generated is called E-node (the node which is explanded)



Dead Node : It is generated that is either not to be expanded further or one for which all of his childrens have been generated.



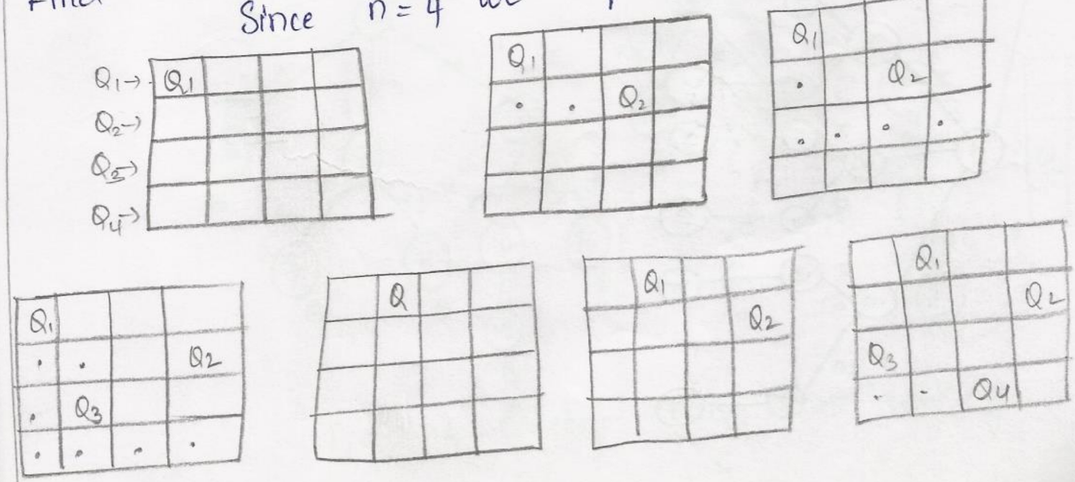
e) **State Space Tree** : If we represent a solution space in form of a tree, then the tree is called as state space tree.

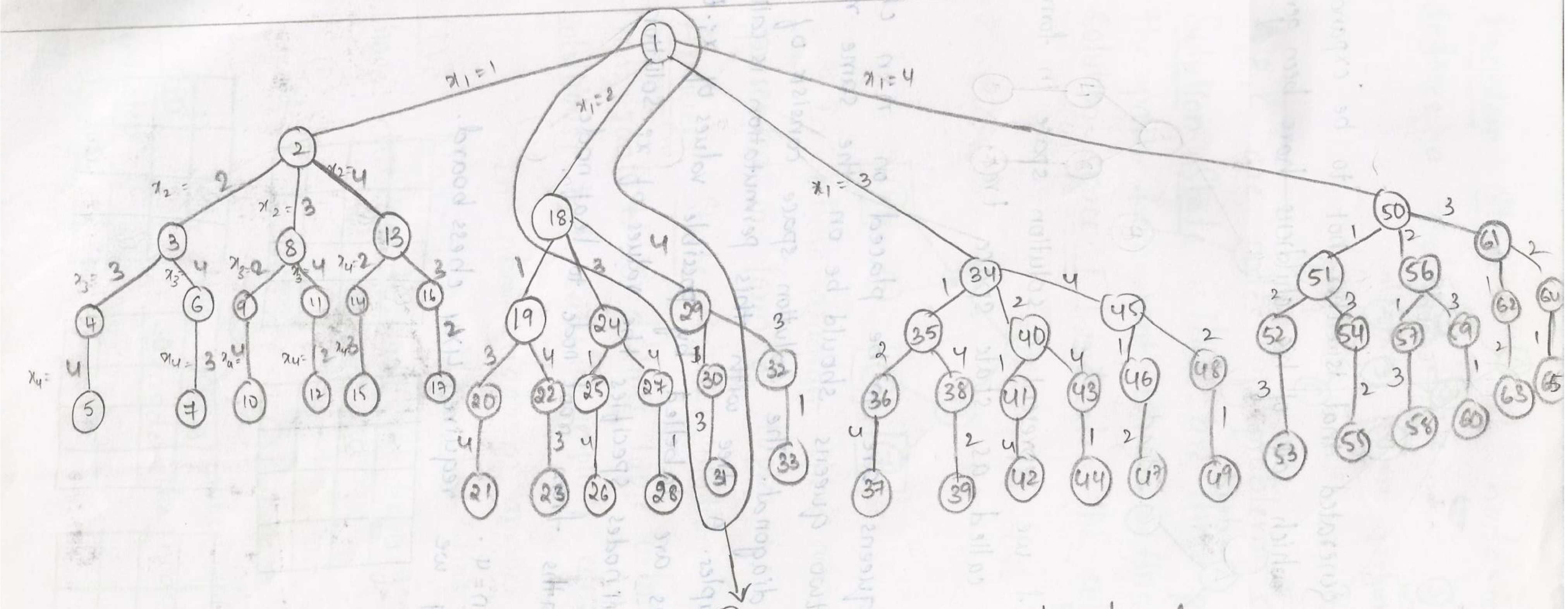
* **n - Queens Problem** :

The n queens are to be placed on nxn chess board so that no two queens should be on the same row, same column, same diagonal. The solution space consists of n! permutations of n tuples. A tree with this permutation is called permutation tree. Edges are labelled by possible values of x_i. Edges from level i to level i+1 nodes specifies the values of x_i. Solution space is defined by all paths from root node to leaf node.

→ Find solution for n=4.

Since n=4 we require 4x4 chess board.





Solution of 4x4 chess boards

Fig: Solution Space tree for $n=4$ Using depth first Search.

Find the solution for n=8

		Q1					
					Q2		
	Q3						
						Q4	
Q5							
			Q6				
							Q7
				Q8			

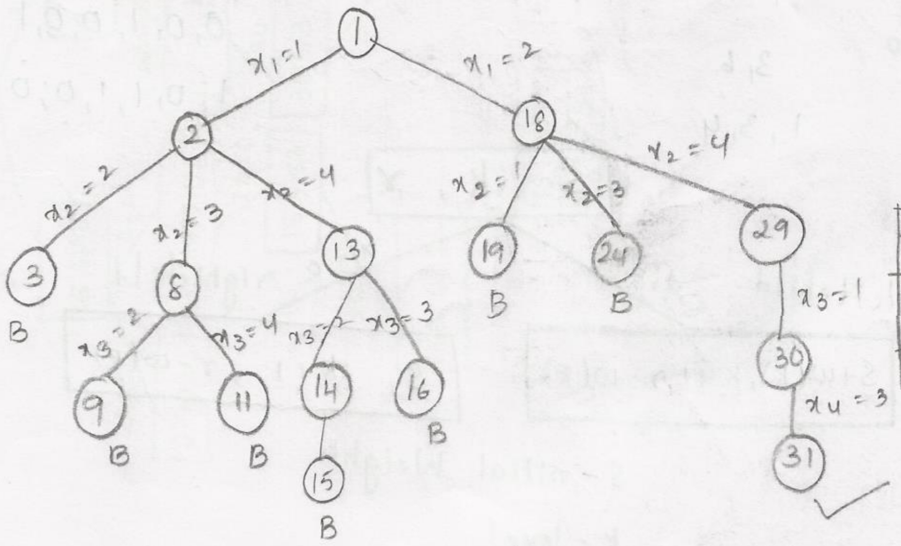
3, 6, 2, 7, 1, 4, 8, 5

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);
        }
    }
}
```

Justify your answer for which value of n, the n queens problem will fail. (n=3)

Q1	Q1		
Q2			Q2
Q3			



	Q1		
			Q2
Q3			
		Q4	

Fig: 4-Queens Using BackTracking.

The above 4 queens generation of nodes using depth first search with bounding function is called BackTracking.
The bounding function is nothing but kill all nodes which is not generating solution, indicated by B.

Sum of Subsets :-

For a given n distinct positive numbers usually called weights, to find all combination of these weights whose sums are m, this is called the sum of subsets proplem. It is solved using the fixed sized tuples and variable sized tuples. We consider the back tracking solution using fixed sized tuples. The element x_i of solution vector is either 0 or 1 (∵ 0 → weight is not consider ; 1 → weight is considered). For a node at level i the left child corresponds to $x_i = 1$ & right child corresponds to $x_i = 0$. A simple choice for bounding is $B_k(x_1 \cdots x_k)$ = true iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m \quad \& \quad \sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m.$$

1) Find out solution for a given $n = 6$ $m = 30$, $w[1:6] = \{5, 10, 12, 13, 15, 18\}$

Variable tuples

(5, 10, 15) 1, 2, 5
⇓
30

3, 6

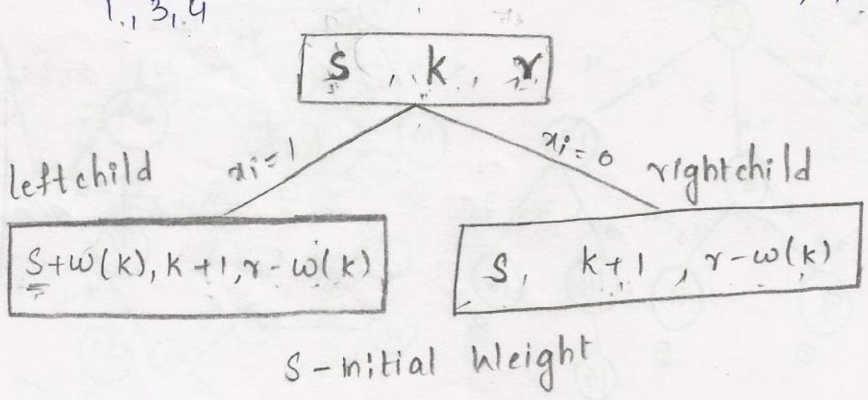
1, 3, 4

Fixed Tuples

1, 1, 0, 0, 1, 0 — (A)

0, 0, 1, 0, 0, 1 — (B)

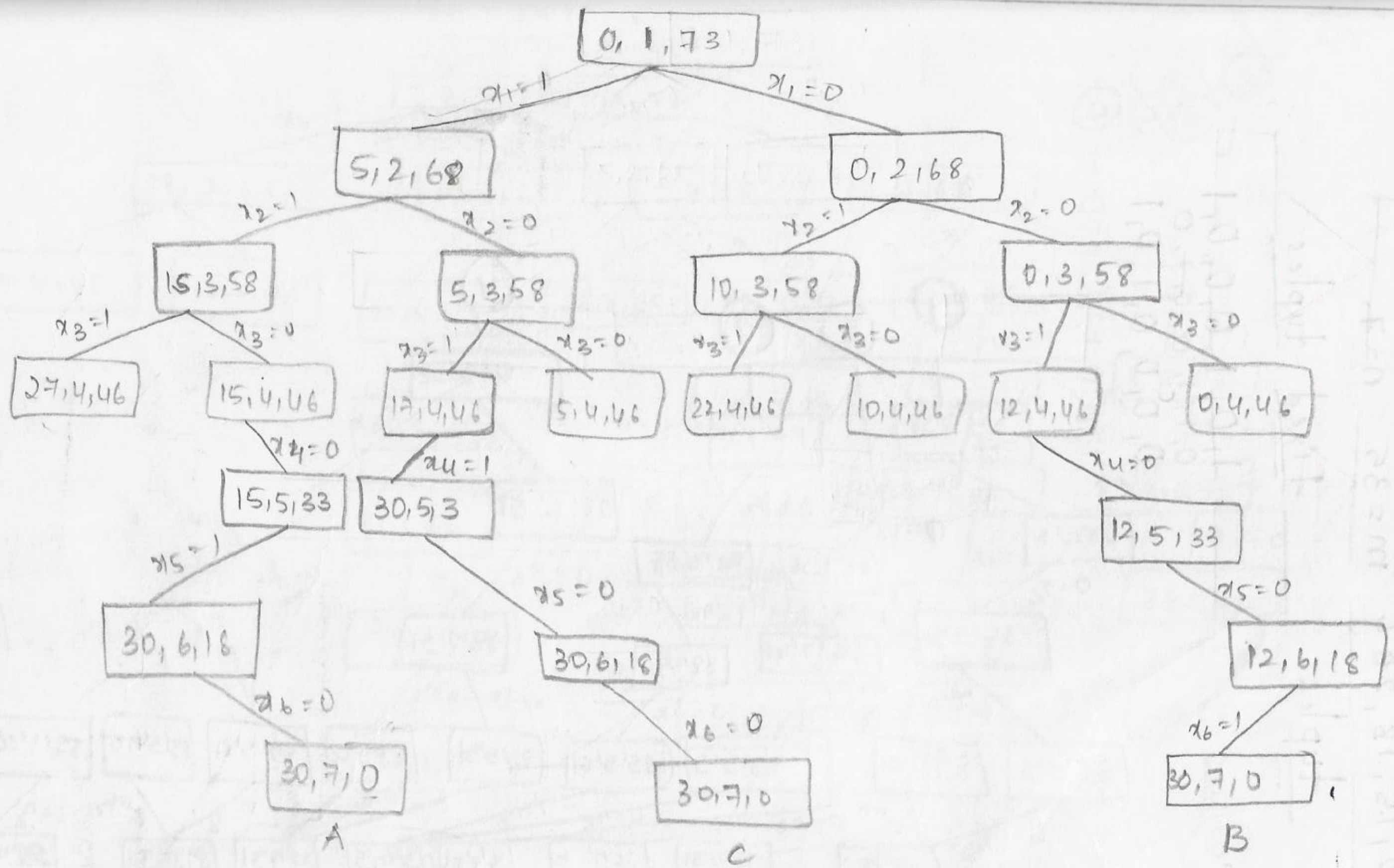
1, 0, 1, 1, 0, 0 — (C)



S - initial weight

k - level

r - sum of all weights


0, 1, 73
x1=1
x1=0
5,2,68
0,2,68
x2=1
x2=0
15,3,58
5,3,58
10,3,58
0,3,58
x3=1
x3=0
27,4,46
15,4,46
17,4,46
5,4,46
22,4,46
10,4,46
12,4,46
0,4,46
x4=0
x4=1
15,5,33
30,5,3
12,5,33
x5=1
x5=0
30,6,18
12,6,18
x6=0
x6=1
30,7,0
A
C
B

x_1 x_2

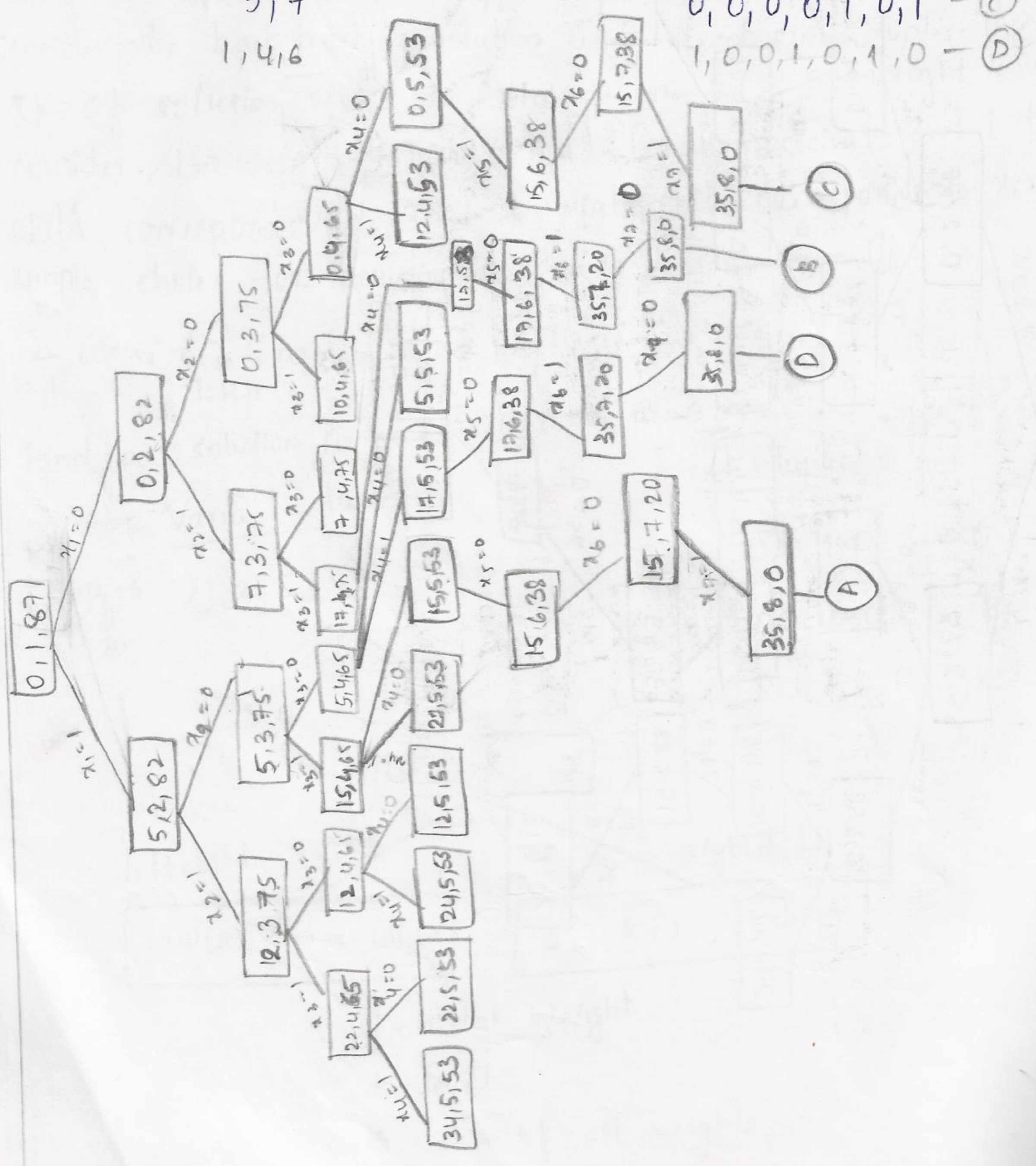
2. $w = \{5, 7, 10, 12, 15, 18, 20\}$ $m = 35$ $n = 7$.

3. $w = \{15, 7, 20, 5, 18, 10, 12\}$ $m = 35$ $n = 7$

4. $w = \{20, 18, 15, 12, 10, 7, 5\}$ $m = 35$ $n = 7$.

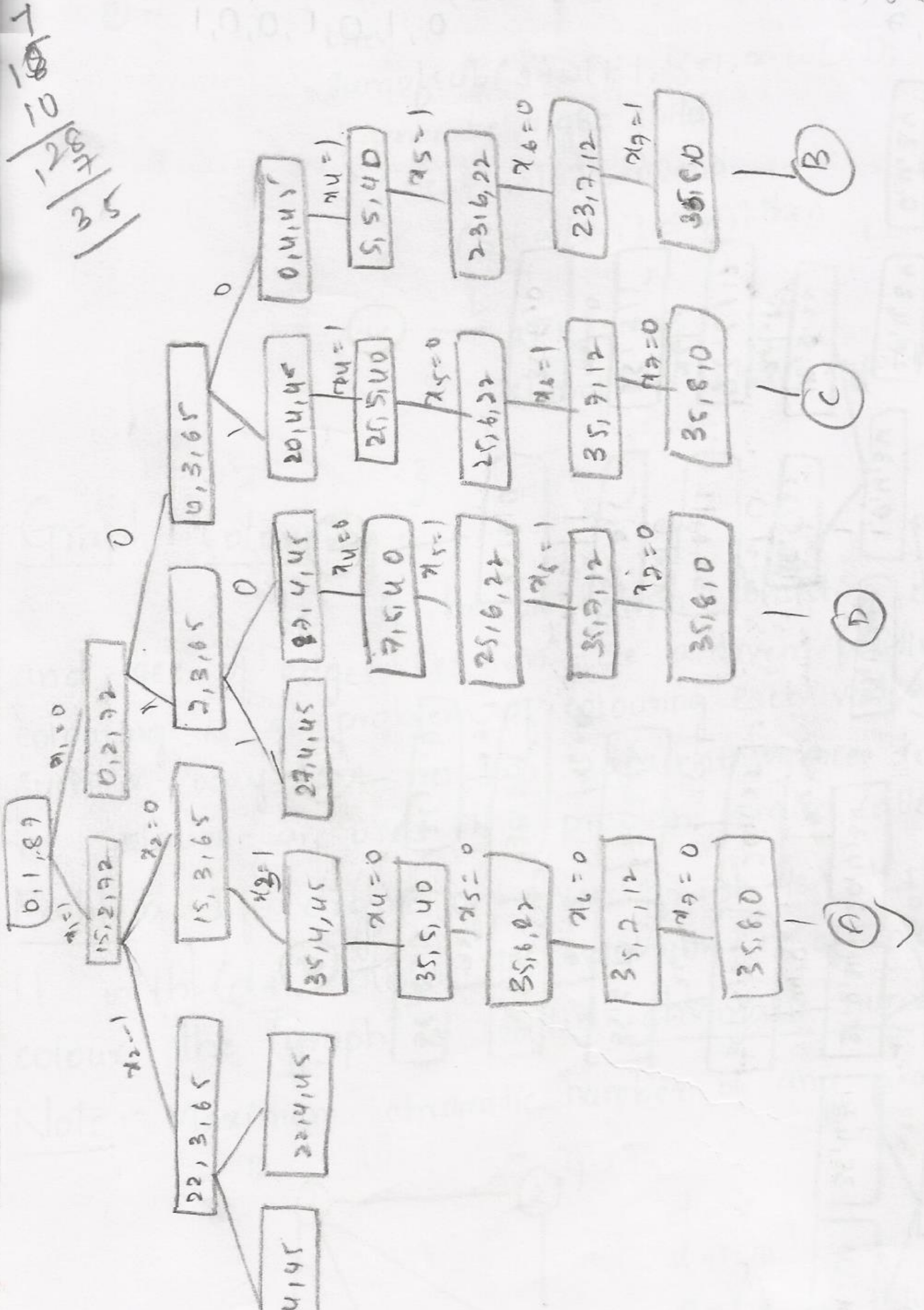
2. $w = \{\overset{1}{5}, \overset{2}{7}, \overset{3}{10}, \overset{4}{12}, \overset{5}{15}, \overset{6}{18}, \overset{7}{20}\}$ $m = 35$ $n = 7$. 87.

Variable tuples	Fixed tuples	
1, 3, 7.	1, 0, 1, 0, 0, 0, 1	A
2, 3, 6	0, 1, 1, 0, 0, 1, 0	B
5, 7.	0, 0, 0, 0, 1, 0, 1	C
1, 4, 6	1, 0, 0, 1, 0, 1, 0	D



$$w = \{\overset{1}{15}, \overset{2}{7}, \overset{3}{20}, \overset{4}{5}, \overset{5}{18}, \overset{6}{10}, \overset{7}{12}\} \quad m = 35 \quad n = 7.$$

Variable tuples	Fixed tuples
1,3	1,0,1,0,0,0,0 — A
4,5,7.	0,0,0,1,1,0,1 — B
3,4,6	0,0,1,1,0,1,0 — C
2,5,6	0 1 0 0 1 1 0 — D



4 $w = \{20, 18, 15, 12, 10, 7, 5\}$ $n = 7$ $m = 35$

Variable tuples

1, 3

1, 5, 7

2, 5, 6

2, 4, 7

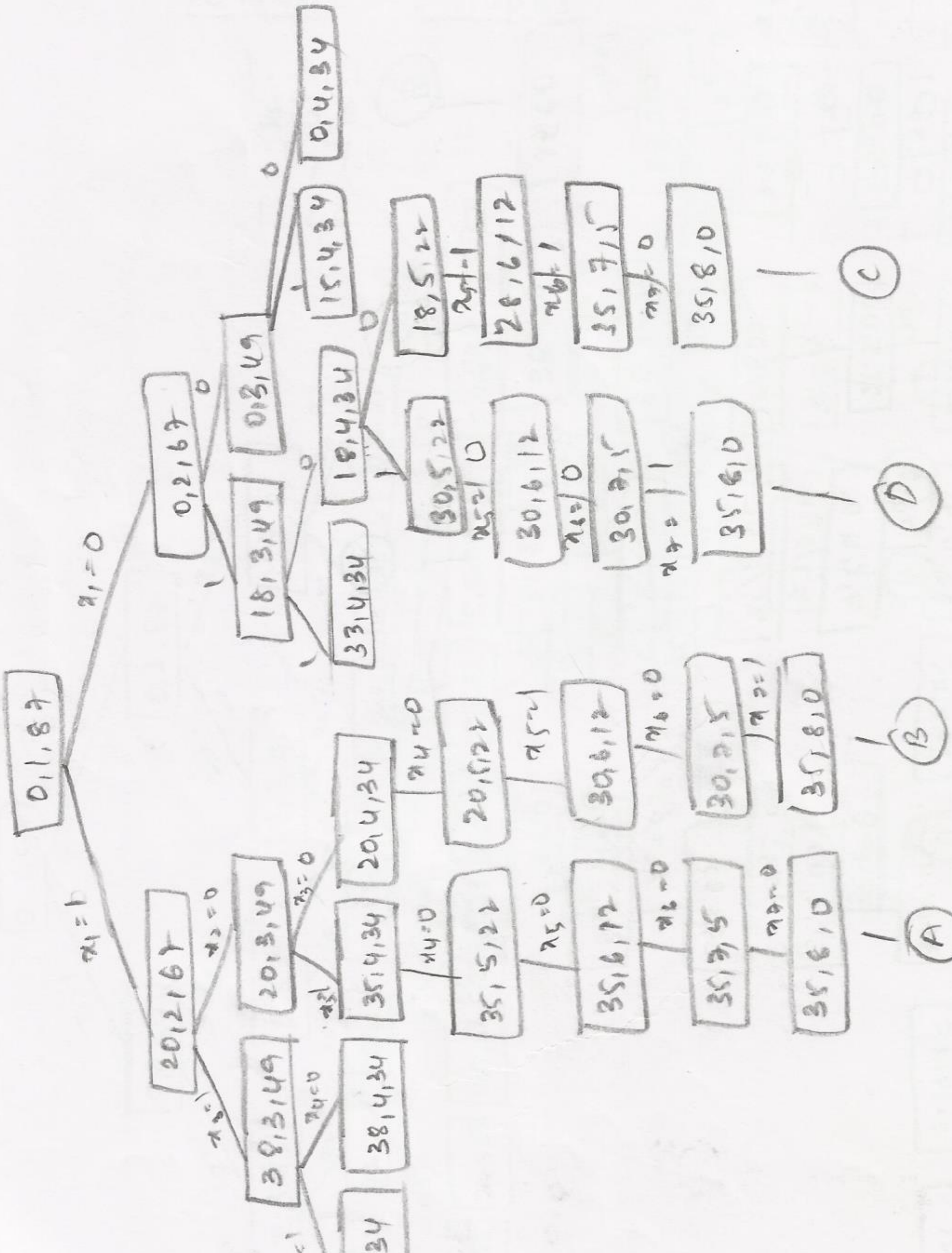
Fixed tuples

1, 0, 1, 0, 0, 0, 0 — A

1, 0, 0, 0, 1, 0, 1 — B

0, 1, 0, 0, 1, 1, 0 — C

0, 1, 0, 1, 0, 0, 1 — D



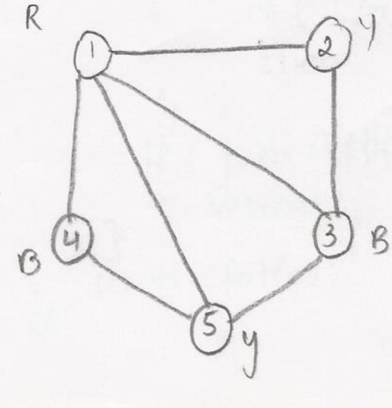
Algorithm :

```
Algorithm sumofsub(s,k,r)
{
    x[k] := 1; //generate leftchild.
    if(s+w[k] = m) then
      write(x[1:k]); //subsetfound
     else if(s+w[k] + w[k+1] ≤ m)
      then
      sumofsub(s+w[k], k+1, r-w[k]);
      //generate right child.
      if((s+r-w[k]≥m) and
           (s+w[k+1] ≤ m)) then
           {
              x[k] := 0;
              sumofsub(s, k+1, r-w[k]);
           }
}
```

Graph Colouring :-

Let G be a graph consisting of set of vertices and set of edges. Let 'm' be a given positive integer. 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. This problem is also called as M-Colouring Problem. If degree of given graph is d, then we can colour it with (d+1) colours. The minimum no. of colours required to colour the graph is called chromatic number.

Note :- Maximum chromatic number of any planar graph is 4.



$d = 3$
$d+1 = 4$
Red, Green, Blue, Yellow

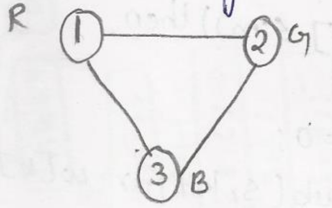
We use backtracking problem ~~using~~ for graph colouring as follows.

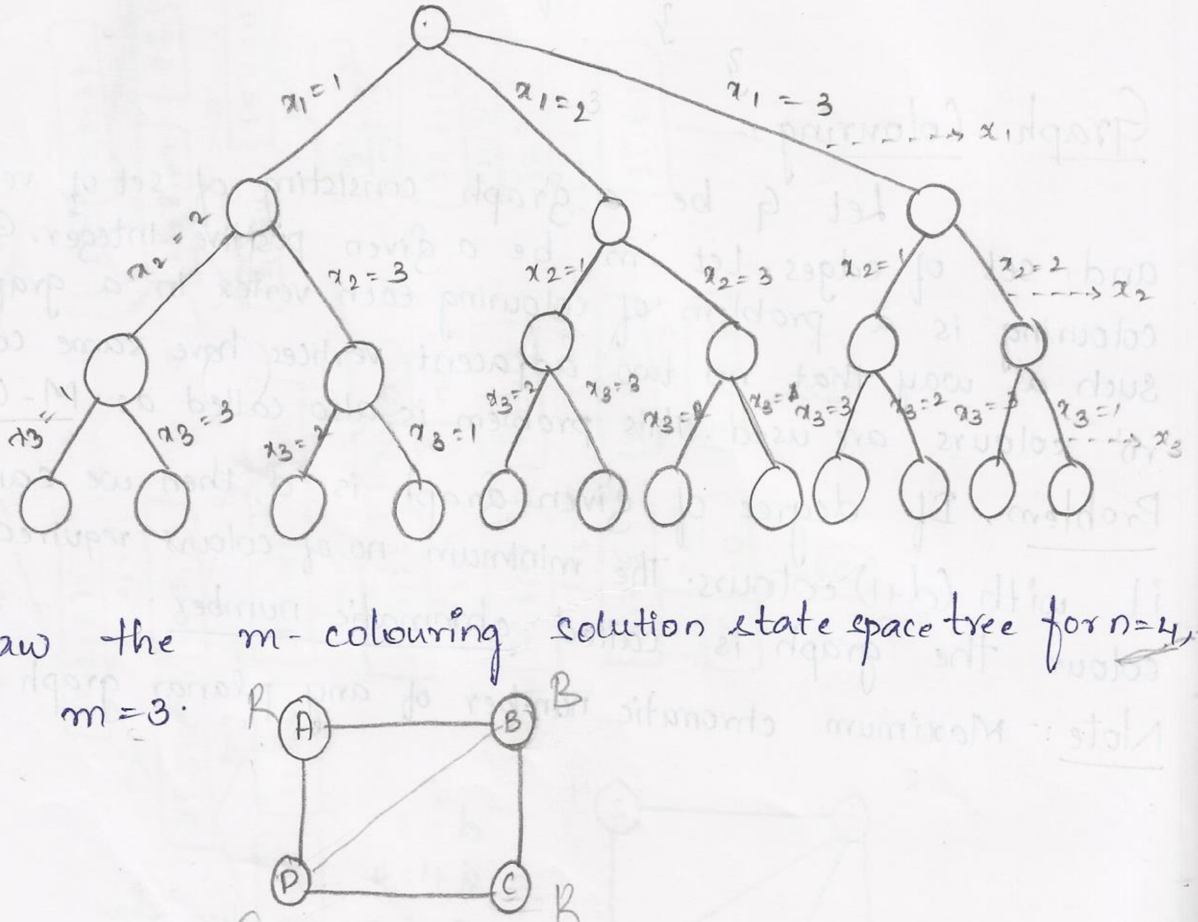
→ Let g be a graph consisting of 'n' vertices with adjacent matrices $A = [a_{ij}]_{n \times n}$ where $a_{ij} = 1$ if $(i,j) \in E(G)$

$= 0$ if $(i,j) \notin E(G)$

→ Let colours are represented by integers $(1, 2, \ldots m)$ and $(x_1, x_2, \ldots x_n)$ be solution where x_i = colour of vertex i.

*. Draw the m-colouring solution state space tree for $n=3$ $m=3$.

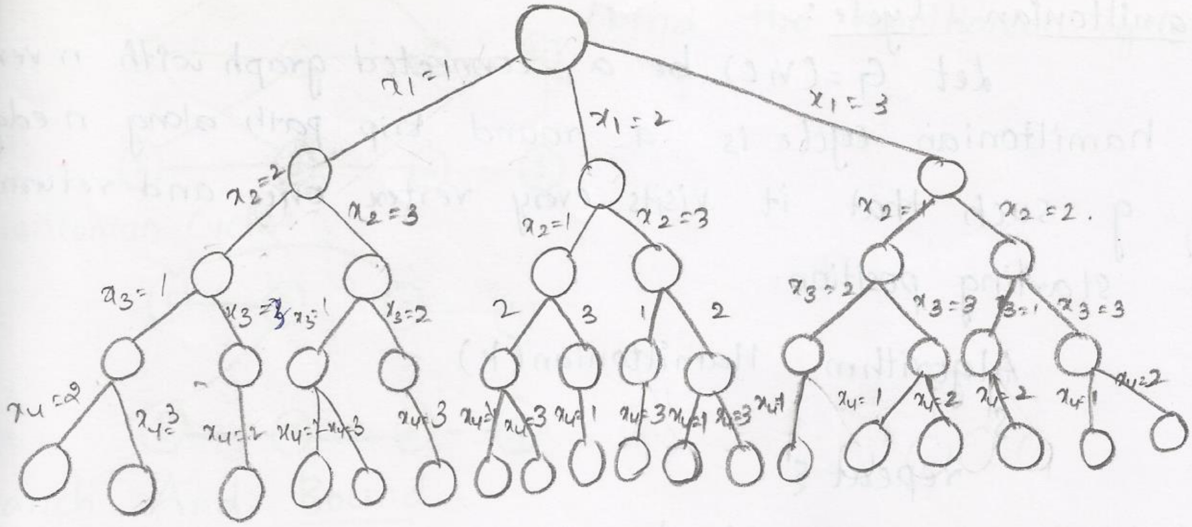




* Draw the m-colouring solution state space tree for $n=4$, and $m=3$.

$d = 2$

$d+1 = 3$



Algorithm :

```
Algorithm m_colouring(K)
{
  repeat
  {
    nextvalue(K);
    if (x[K]= 0) then return;
    if (K=n) then
      write(x[1:n]);
    else mcolouring(K+1);
  }
  until(false);
}
Algorithm nextvalue(K)
{
    repeat
    {
     x [K]: = (x[K]+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
     }
     if ( j=n+1) then
          return;
    }
    until(false);
}
```

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 g such that it visits every vertex once and returns to it starting position.

```
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;
  }
 Algorithm nextvalue(k)
 {
    repeat {
       x[k]:=(x[k+1])mod(n+1);
       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);
   }
```