Unit - 4

Backtracking

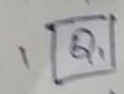
1. N-Queen's problem

Consider an N×N chess board on which we have to place N queens, so that no two queens attack each other by being in the same row or in the same column or in the same diagonal

The following figures illustrates the N-queen's problem for N = 1, 2, 3

For N=1, we have to place one queen on 1×1 chess board. This problem has a trivial soln because it contains one cell, there is no alternate position for queen 1.

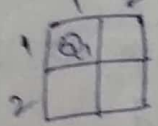
N=1



Trivial soln

For N=2, we have to place two queens on 2×2 chess board. Every row corresponding queens and column contains one queen. There is no solution for N=2 because Q_1 is placed on (1, 1) position, Q_2 is cannot be placed.

N=2

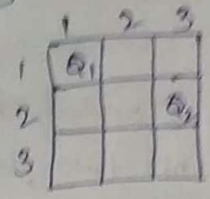


No soln

For N=3, we have to place three queens on 3×3 chess board. Q_1 can be placed on (1,1) position, Q_2 can be place on (2,3), but there is no position for Q_3. Hence we can say that

there is no sol^n of $N=3$

$N=3$



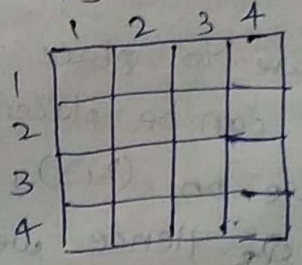
No solution

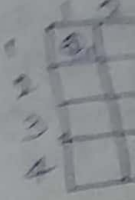
Let us, $N=4$ and solve it by using the back-tracking technique. We have to place 4 queens on 4×4 chess board. Each queen has to be placed in its own row i.e., Q_1 is placed in 1st row, Q_2 is placed on 2nd row, Q_3 is placed on 3rd row, Q_4 is placed on 4th row. Our objective is to assign a column number for each queen on 4×4 chess board. This problem has 2 constraints.

1. Implicit constraint
2. Explicit constraint

The implicit constraints for this problem is, no two queens are on the same column and no two queens are on the same diagonal.

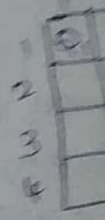
The explicit constraints for this problem is: The finite set $S=\{1,2,3,4\}$, the sol^n vector can take the values from the finite set S. Now, we start with empty chess board





Place Q_2 at (2,

after trying



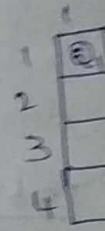
Place Q_3 on 3rd

(3,1), (3,2), (3,3)

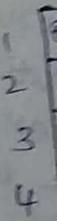
for Q_3. Hence

check the possible

one position i.e



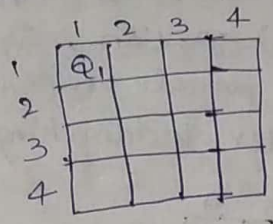
Now place Q_3



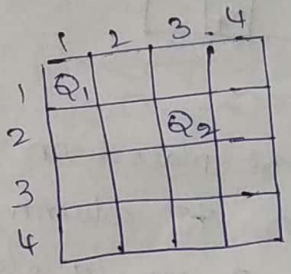
Now place

position for

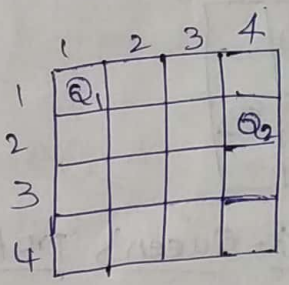
place Q_1 in the first possible position of its row i.e., on first row & first column



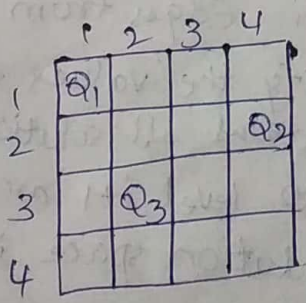
Place Q_2 at (2,3) position i.e 2nd row & 3rd column after trying unsuccesful positions (2,1) & (2,2)



Place Q_3 on 3rd row by trying the possible positions (3,1), (3,2), (3,3), (3,4) all are unsuccessful positions for Q_3. Hence the algorithm backtracks and check the possible position for Q_2. Q_2 has only one position i.e., (2,4)

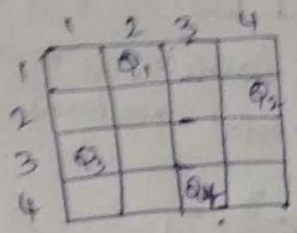


Now place Q_3 at (3,2) position



Now place Q_4 on 4th row. There is no possible position for placing Q_4 on 4th row. After trying

all unsuccessful positions (4,1), (4,2), (4,3), (4,4). Hence backtrack Q_3, Q_2, Q_1. Finally, we have to place Q_1 at (1,2) position, Q_2 can be placed at (2,4), Q_3 can be placed on (3,1) and Q_4 can be placed on (4,3). After backtracking Q_3, Q_2, Q_1

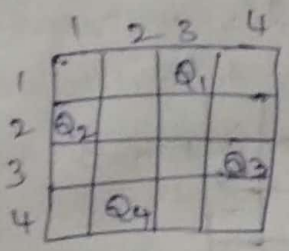


$X = [2, 4, 1, 3]$

That means Q_1 can be placed on 2nd column, Q_2 can be placed on 4th column, Q_3 is placed on 1st column and Q_4 can be placed on 3rd column.

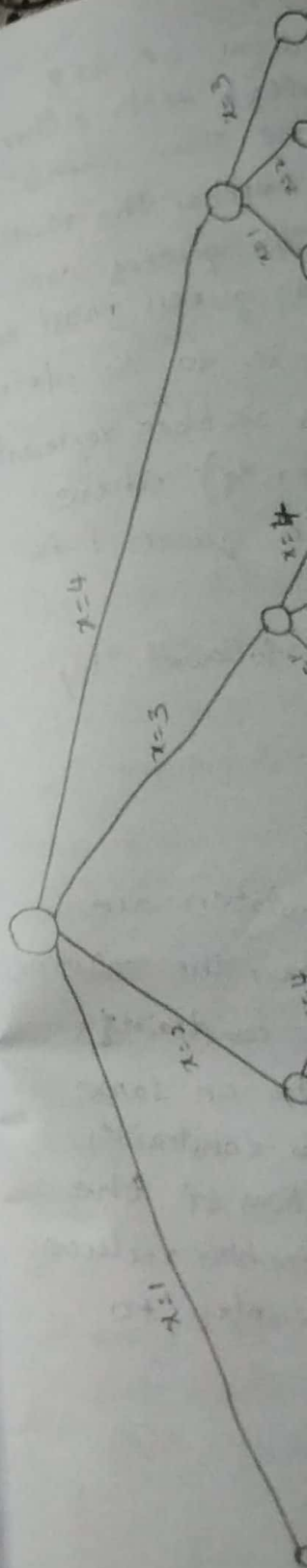
The another soln of 4 queens problem, mirror image of the above soln.

$X = [3, 1, 4, 2]$



State Space Tree for 4-Queen's problem

In above state space tree, the edges are labelled by possible values of x_i. Edges from Level 1 to Level 2 nodes specify the values for x_i. The left most subtree contains all solutions with $x_1 = 1$. Edges from Level i to level i+1 are labelled with values of x_i. The solution space is defined by all paths from the root node to a leaf node. There are $4! = 24$ leaf nodes in the above state space tree



Scanned by CamScanner

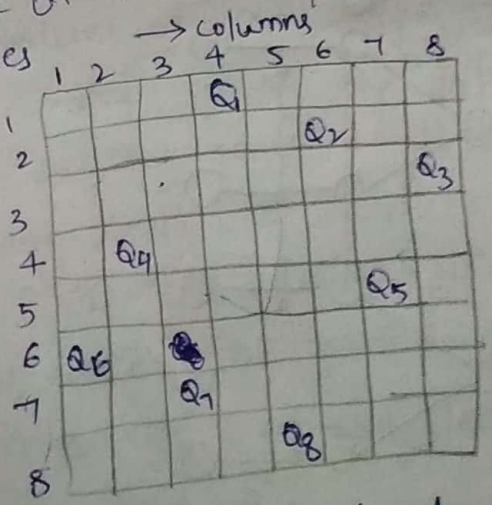
8-Queens problem

8-Queens problem is to place 8 queens on 8×8 chess board so that no two queens attack each other i.e., no two queens are on the same row, same column or same diagonal. Let us number the rows and columns of chess board 1 to 8. The queens can also numbered from 1 to 8. Since each queen must be on a different row. Assume queen i is to be placed on row i. All sol"s to the 8 queens problem represented as 8-tuples $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ where x_i is the column number on which queen i is to be placed.

There are two types constraints must followed by the 8-queens problem -

1. Explicit constraints
2. Implicit constraints

The explicit constraints using this formulation are $S_i = \{1,2,3,4,5,6,7,8\}$ $1 \le i \le 8$. Therefore, the solution space contains 8^8 tuples. The implicit constraints for this problem are no two x_i can be on same column or on same diagonal. These two constraints implies that all solutions are permutations of the 8 tuples (1,2,3,4,5,6,7,8). This realization reduces the size of the sol" space from 8^8 tuples to 8! tuples



8×8 chessboard

Solution vecto

Suppose two
(k,l). These
if and only
|j-l| =
Two queens
Both queens
(2-3) = 1
For an 8
ways to pl
placement
we require
The total
tree is 6
The t
space tr
Algorithm fo
Algorithm
// k is the
// n is the
// using ba
placemen
that
{
fo
{

Solution vector $X = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$
$= [4, 6, 8, 2, 7, 1, 3, 5]$

Suppose two queens are placed at positions (i, j) & (k, l). These two queens like on the same diagonal if and only if

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

Two queens Q_1 & Q_2 are lie on $(1,2)$ and $(3,3)$. Both queens are not on the diagonal. Because

$|2 - 3| = |1 - 3|$

For an 8×8 chessboard there are 64_{C_8} possible ways to place 8 queens. However by allowing only placements of queens on distinct rows & columns we require 8! ways.

The total no of nodes in the 8 queens state space tree is 69,821 nodes.

The total no of nodes in the 4 queens state space tree is 65 nodes.

Algorithm for N-Queen's problem

```
Algorithm N-Queens(k,n)
// k is the row number or Queen number
// n is the number of columns
// using backtracking, this procedure prints all posible
placements of N-Queens on nxn chess board so
that queens are non attacking
{
    for i:=1 to n do
    {
        If place(k,i) then
        {
            x[k]:=i;
```

```
        If (k==n) then write(x[1:n]);
        else
            return N-Queens(k+1,n);
    }
}
Algorithm place(k,i)
{
// This procedure
// k is the row number (or) queen number
// i is column number
// This procedure returns true if a queen can be
   placed in kth row & ith 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
    // Two Queens on the same diagonal
    If ((x[J]=i) or (Abs[x[J]-i)= Abs(J-k))) then
    // Two Queens on the same column
        return false;
    else
        return true;
}
```

Backtracking general

Backtracking is o
algorithm design
Many problems wh
of solutions or
some constraints
formulation
The name backt
in 1950. If th
$(x_1, x_2, \ldots x_n)$ w
finite set S_i,

All the soln's
constraints
(i) Explicit co
(ii) Implicit

Explicit constra
each x_i to

Ex: (i) $x_i \geq 0$
(ii) $x_i = 0$

Explicit const
of the proble
the explicit
space for i
The implici
which of t
objective f
Generally
They are
(i) Enume
In

Backtracking General method:

Backtracking is one of the most important algorithm design technique.

Many problems which deal with searching for a set of solutions or ask for an optimal sol^n satisfying some constraints can be solved using the backtracking formulation

The name backtrack was defined by D.H. Lemmer in 1950. If the sol^n is expressable as an n tuples $(x_1, x_2, \ldots x_n)$ where x_i are chosen from some finite set S_i, then we can apply backtracking

All the sol^{n}s using backtracking must satisfy two constraints

(i) Explicit constraints

(ii) Implicit constraints

Explicit constraints are the rules which restrict each x_i to take on values from a given set S_i

Ex: (i) $x_i \geq 0$ where $S_i = \{$All non-negative real no's$\}$

(ii) $x_i = 0$ or $x_i = 1$ where $S_i = \{0, 1\}$

Explicit constraints depends on the particular instance of the problem being solved - All tuples that satisfies the explicit constraints define a possible solution space for i.

The implicit constraints are rules that determine which of the tuples in solution space satisfies the objective function

Generally backtracking solve three types of problems.

They are.

(i) Enumeration problems

In enumeration problems, we find all the possible

feasible solutions.

(ii) Decision problems

In decision problems we find whether there is any feasible sol^n or not for a given problem. The problem is also called as yes or no type problem.

(iii) Optimization problems

In optimization problems, we find whether there exists any best solution

All solutions are represented by using a tree called state space tree. Each state space tree contains 3 types of nodes. They are live node, Enode, dead node

A node which has been generated and whose children have not yet been generated is called a live node

A live node whose childrens are currently being generated is called a Enode.

A node which is already expanded and there is no use for future is called a dead node.

Each node in the state space tree defines a problem. All paths from root node to leaf node or other node defines a state space

In backtracking we have to use bounding functions, they will be used to kill live nodes without generating all their children

The tree organization of the solution space is referred to as the state space tree.

(OR)

In backtracking, while solving a given problem, a tree is constructed based on the choices made.

such a tree wi
state space tree.

In state space tr
(i) Promising node
(ii) Non-Promising
Promising node:
A node in a st
if it corresponds
may lead to a co
Non-promising node
A node in a st
if it cannot

Algorithm for

Algorithm B
// This algorith
using recu
x[1] if [
x[1:n] ha

// x[], n a
{
for (each
{
If (
{

}
}
}

such a tree with all possible sol^{n}s is called a state space tree.

In state space tree there are 2 types of nodes
(i) Promising node
(ii) Non-Promising node.

Promising node:
A node in a state space tree is said to be promising if it corresponds to a partially constructed sol^n that may lead to a complete solution

Non-promising node:
A node in a state space tree is said to be non-promising if it cannot lead to a complete sol^n.

Algorithm for General Method of Backtracking

```
Algorithm Backtrack(k)
//This algorithm describes the backtracking process
  using recursion. On entering, the first (k-1) values
  x[1] x[2], ... x[k-1] of the solution vector
  x[1:n] have been assigned.
// x[], n are global
{
   for (each x[k] ∈ T(x[1], x[2], ..., x[k-1]) do
   {
      If (Bk(x[1], x[2], ..., x[k]) ≠ 0) then
      {
         If (x[1], x[2], ..., x[k] is a path to an
                                   answer node) then
            write (x[1:n]);
         If (k<n) then Backtrack(k+1);
      }
   }
}
```

Graph coloring pr

Let G be a gra
graph coloring p
of graph G can
that no two a
only m colors
decision problem
1. m-colorability
2. m-colorabilit

The m colourat
smallest integer
colored. This
number of th

Re

Ex

The above
Red, Blue,
the vertice
number of
Chromatic

The
all v
chroma

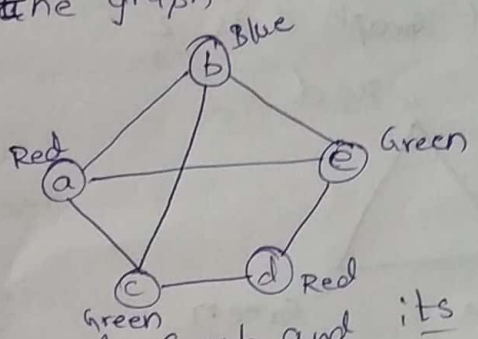
If d
colored

Graph coloring problem

Let G be a graph, n be a positive integer. The graph coloring problem is to find whether the nodes of graph G can be can be colored in such a way that no two adjacent nodes have the same color, yet only m colors are used. It is called as m-colorability decision problem. This problem has two versions

1. m-colorability decision problem.
2. m-colorability optimization problem.

The m colourability optimization problem ask for the smallest integer m for which the graph G can be colored. This integer is referred to as chromatic number of the graph



Example Graph and its coloring

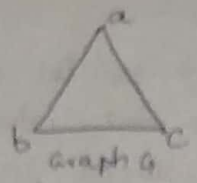
The above graph can be colored with 3 colors Red, Blue, Green. 3 colors are needed to color all the vertices of the given graph G. Hence the chromatic number of the given graph G is 3.

Chromatic number:

The minimum no. of colors required to color all vertices of given graph G is called as choomatic number of graph G.

If d is the degree of given graph, then it can be colored with d+1 colors

Ex:



Graph G

For above graph, how many no of colors required to coloured all vertices of graph G?

The degree of given graph G is 2, it can be coloured with $d+1$ colors.

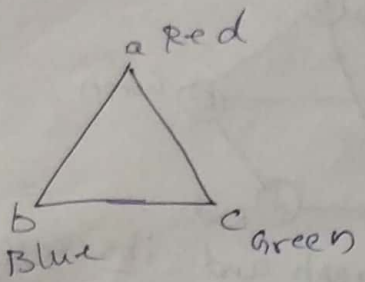
$2+1=3$ colors

Degree of vertex a is 2

" b is 2

" c is 2

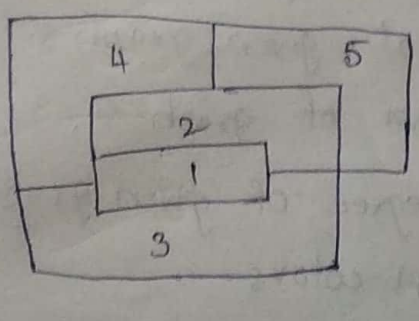
∴ The degree of Graph G is 2

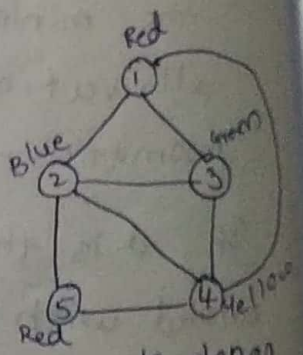


Chromatic number of ~~the~~ graph G is 3

Colouring of planar graph:

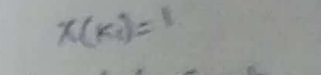
A graph G is said to be planar graph if and only if it can be drawn in a plane in such a way that no two edges are crossed each other.





A map and its planar graph representation

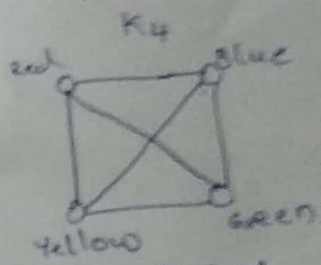
The above planar graph
the chromatic number
The time complexity
$O(nm^2)$
The number of int
is $\sum_{i=0}^{n-1} m_i$ at
Speed by next valu
to legal colorings
Note: The chromatic
defined as follow
Complete Graph
K_1



$\chi(K_1)=1$

Complete Graph

K_4



$\chi(K_4)=4$

A complete g
n colors to
The chromatic
chromatic num
In a bip
given ver
The first
group 1
in class

The above planar graph can be coloured with 4 colors the chromatic number of the planar graph G is 4

The time complexity of graph coloring problem is $O(nm^n)$

The number of internal nodes in stace space tree is $\sum_{i=0}^{n-1} m^i$; at each internal node $O(mn)$ time is spend by next value to determine children corresponding to legal colorings

Note; The chromatic number of complete graph can be defined as follows:

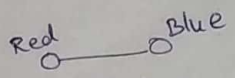
Complete Graph

K_1

$\chi(K_1)=1$

Complete Graph

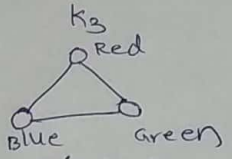
K_2



$\chi(K_2)=2$

Complete Graph

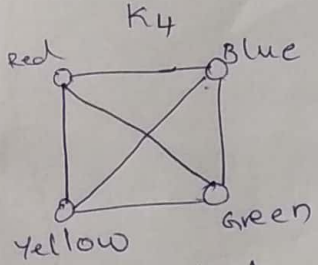
K_3



$\chi(K_3)=3$

Complete Graph

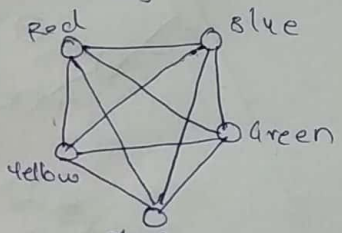
K_4



$\chi(K_4)=4$

Complete Graph

K_5



$\chi(K_5)=5$

A complete graph with n vertices (K_n) requires n colors to color all the vertices of the given graph

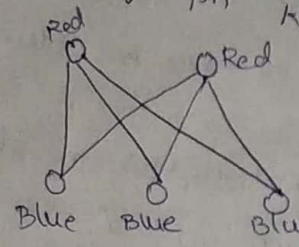
The chromatic num of a given graph is denotes $\chi(G)$

chromatic number of bipartiate graph:

In a bipartiate graph we have to divide the given vertices in graph G into two classes.

The first coordinate contains no of vertices in group 1 and 2nd coordinate denotes no of vertices in class 2 / graph 2:

How many no. of colors required to color all vertices of given bipartite graph $K_{2,3}$



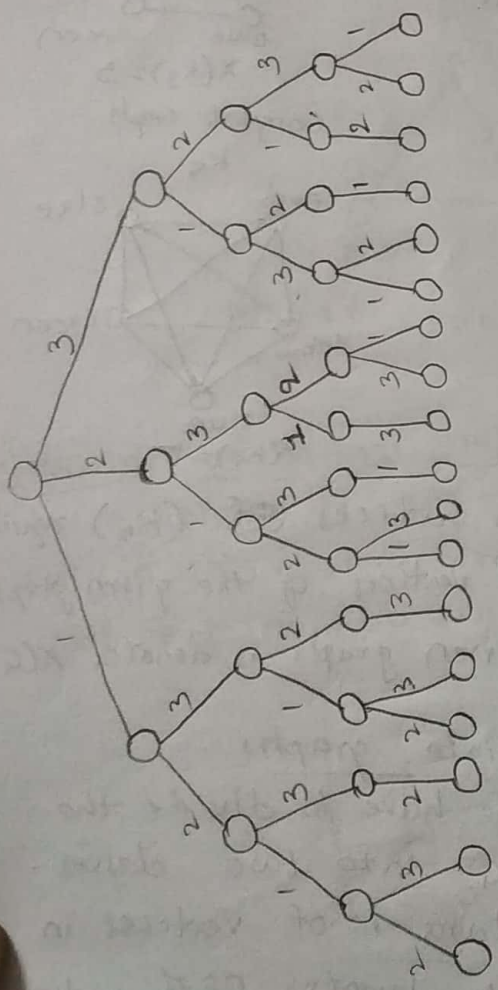
class I

class - II

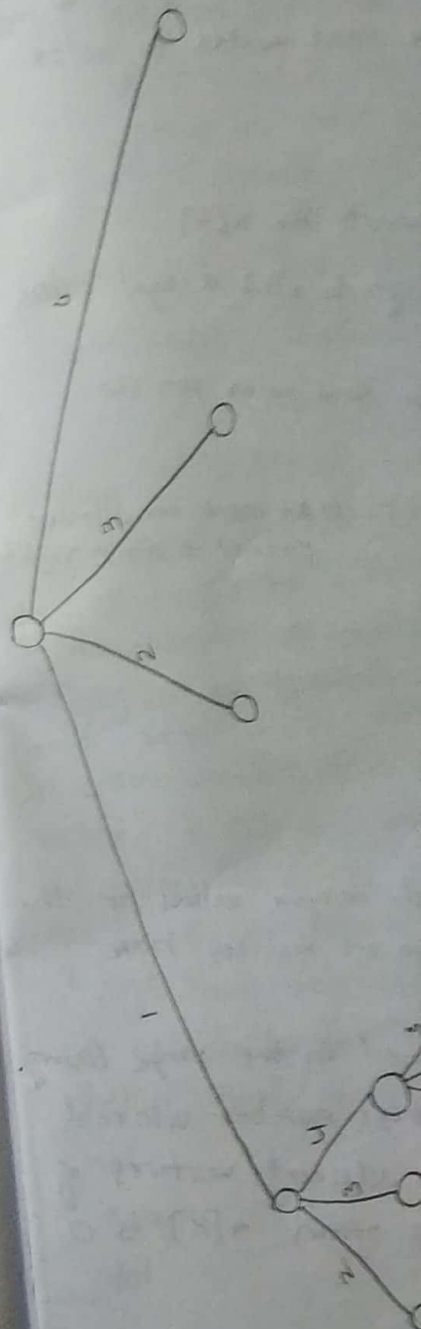
$\chi(K_{2,3}) = 2$

Note: The chromatic number of any bipartite graph is 2

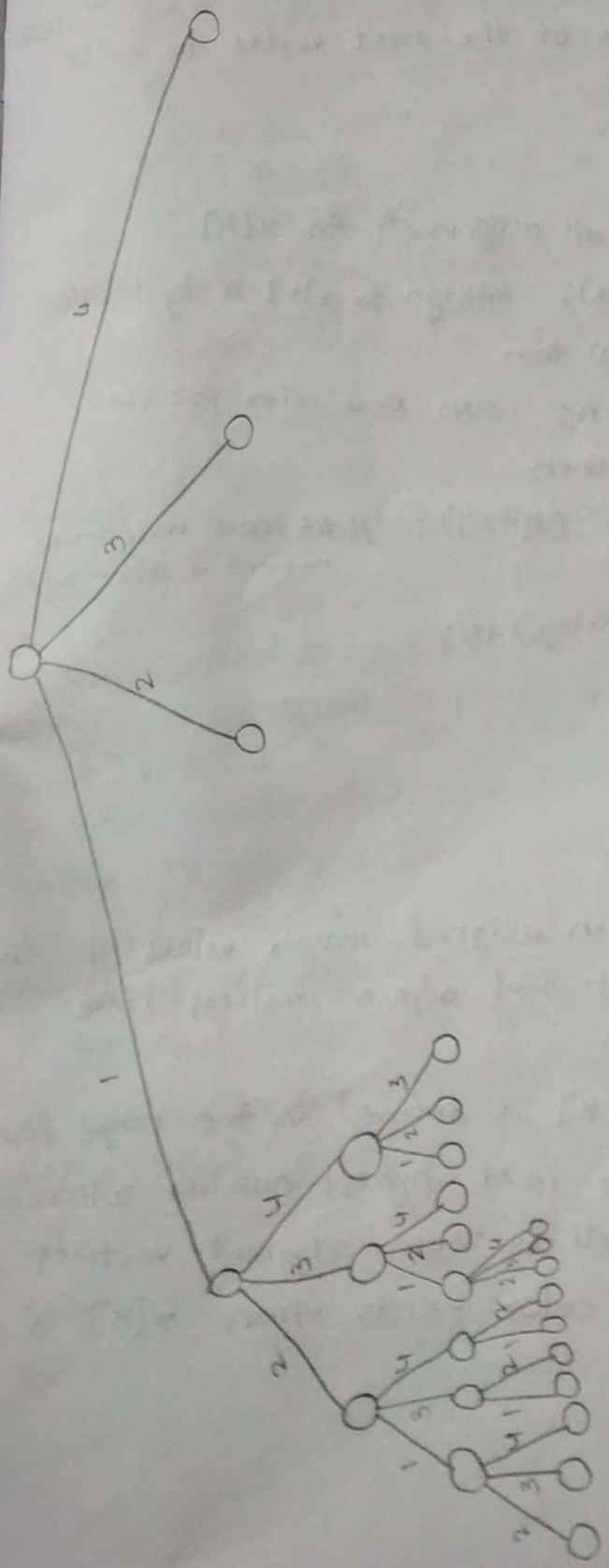
State space Tree for Graph coloring problem of Graph G with 4 vertices and 3 colours



Space Space Tree for
n=4, m=4



State Space Tree for Graph coloring Problem for n=4, m=4



Algorithm for graph colouring problem

```
Algorithm mcoloring(k)
// This graph is represented by its toolean adjacency
   matrix G[1:n,1:n]
// All assignments of 1,2,...m to the vertices of graph
   such that adj vertices are assigned distinct integers are printed
// k is the index of the next vertex to color
{
    repeat
    {
        // Generate all assignments for x[k]
        Nextvalue(k);  //Assign to x[k] a legal color
        If(x[k] =0) then
            return;   //No New color possible
        If(k==n) then
            write (x[1:n]);  // At most m colours
                                needed to color n vertices
        else
            mcoloring(k+1);
    }until(false);
}
```

```
Algorithm Nextvalue(k)
// x[1]...x[k-1] have been assigned integer values in the
   range [1,m] such that adjacent vertices have
   distinct integers
// x[k] The value of x[k] is defined in the range [0,m]
   x[k] is assigned the next highest number colored
   while maintaining distinctness from adjacent vertices of
   vertex k. If no such color exists, then x[k] is 0
{
    repeat
    {
```

```
x[k]:=x[k]+1 mod
If(x[k]=0) the
for j:=1 to n
{   // check of th
    colors
    If ((G(k,j) ≠0
    // If (k,j)
    have the
        brea
}
If(j=n+1) th
} until(false);  //
}
```

Finding Hamiltonian cycl

Let G=(V,E) be
A hamiltonian
n edges of G
and returns
In other words,
vertex $V_1 \in G$
the order V_1,
are in E, $1 \le$
V_1 and V_{n+1} u

Let us consider
whether these

hol

Graph
G1:

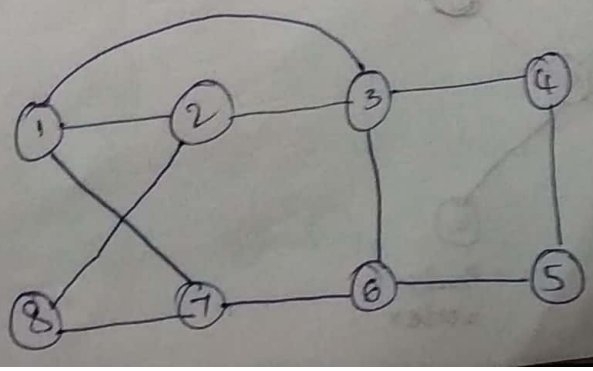
```
x[k] := x[k]+1 mod (m+1);   // Next highest color
If(x[k] = 0) then return;   // All colors have been used.
for j := 1 to n do
{   // check of this color is distinct from adjacent
    colors
    If ((G(k,j) ≠ 0) and (x[k] = x[j])) then
    // If (k,j) is an edge and If adjacent vertices
    have the same color
        break;
}
If (j = n+1) then return;
} until(false);   // otherwise try to find another color
}
```

Finding Hamiltonian cycle / circuit Problem:

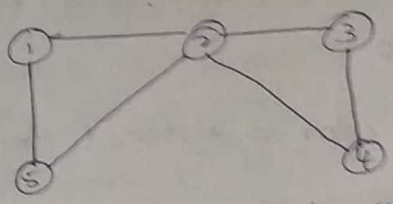
Let $G=(V,E)$ be a connected graph with n vertices. A hamiltonian cycle is a round trip path along n edges of G that visits every vertex exactly once and returns to the starting vertex. In other words, if a hamiltonian cycle begins at some vertex $V_1 \in G$ and the vertices of G are visited in the order $V_1, V_2, V_3, \ldots, V_{n+1}$, then the edges (V_i, V_{i+1}) are in E, $1 \le i \le n$ and the V_i are distinct except for V_1 and V_{n+1} which are equal.

Let us consider the following graphs G_1 and G_2 check whether these graphs containing the hamiltonian cycle or not.

Graph G1:



Graph G2:



The first Graph G1 contains the Hamiltonian cycle

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

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

By observing the above two Hamiltonian cycles, starting vertex and ending vertex are same, the remaining vertices in that Hamiltonian cycle are visits exactly once

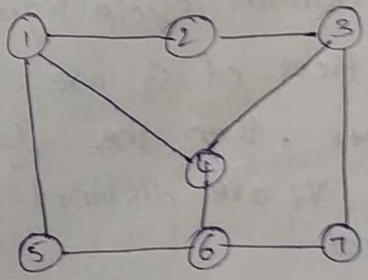
Consider Graph G2, it doesnot contain any Hamiltonian cycle.

Hamiltonian cycle problem is a NP-Hard problem.

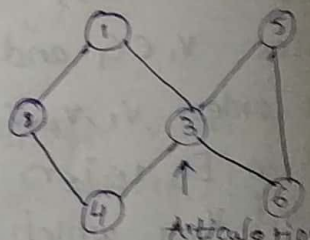
If any graph contains the articulation point, then we can say that there is no Hamiltonian cycle for that graph.

If any graph contains any pendant vertex, then we can say that there is no Hamiltonian cycle.

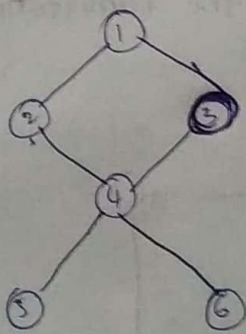
G3:



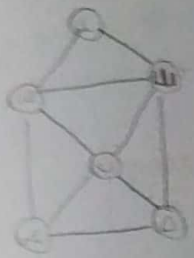
G4:

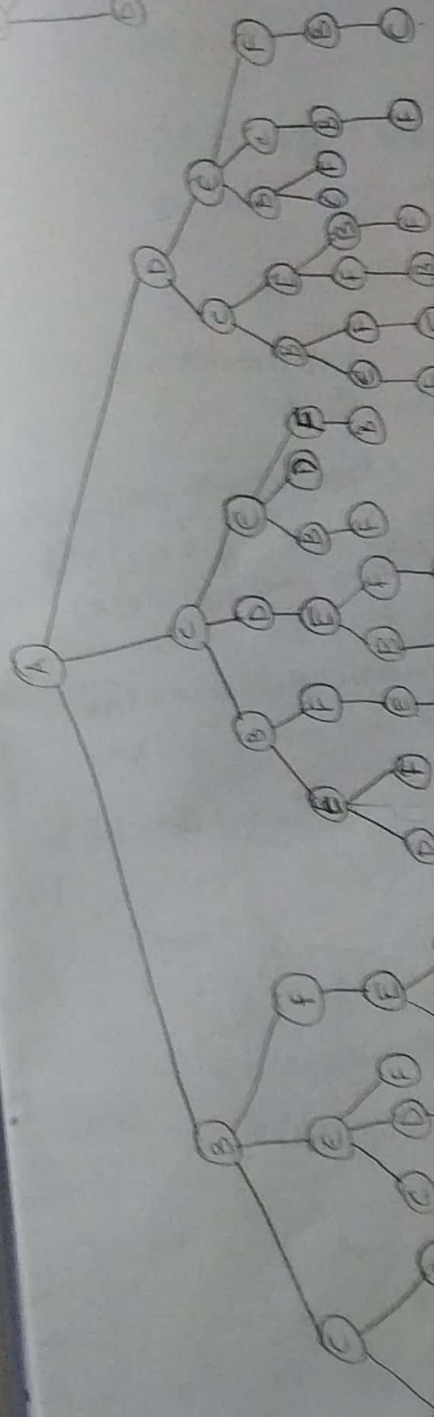


G5:



Draw the state space tree for cycles for the given graph

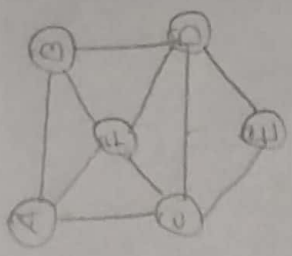




Draw the State Space tree for finding out all Hamiltonian cycles for the given graph

The Hamiltonian cycles are

① A - B - F - E - C - D - A
② A - B - F - E - D - C - A
③ A - C - B - F - E - D - A
④ A - C - D - E - F - B - A
⑤ A - D - C - E - F - B - A
⑥ A - D - E - F - B - C - A



Algorithm
// This algorit
to find
// This graph
// All the
{
repe
{
}
}
Algorit
// x[
(x[
exe
ver
an
// f
{

```
Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation of backtracking
 to find all the Hamiltonian cycles of a graph.
// This graph is stored as an adjacency matrix G(1:n,1:n)
//All the Hamiltonian cycles begin at node 1.
{
    repeat
    {
        Nextvalue(k);  //Generate not values to x[k] and
                          assign a next value to x[k]
        If (x[k]==0) then return;
        If (k==n) then write (x[1:n]);
        else
            Hamiltonian(k+1);
    } until(false);
}
Algorithm Nextvalue(k)
// x[1:k-1] is a path of (k-1) distinct vertices. If
 (x[k]=0), then no vertex has been assigned to x[k]. After
 execution, x[k] is assigned to the next highest numbered
 vertex which doesn't already visited in x[1:k-1].
 and is connected by an edge to x[k-1] otherwise x[k]=0
// If k=n, then x[k] is connected to x[1]
{
  repeat
  {
    x[k]:=(x[k]+1) mod (n+1);  // Next vertex.
    If (x[k]==0) then return;  //Backtrack
    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; //checks for vertex
                                          distinctness
```

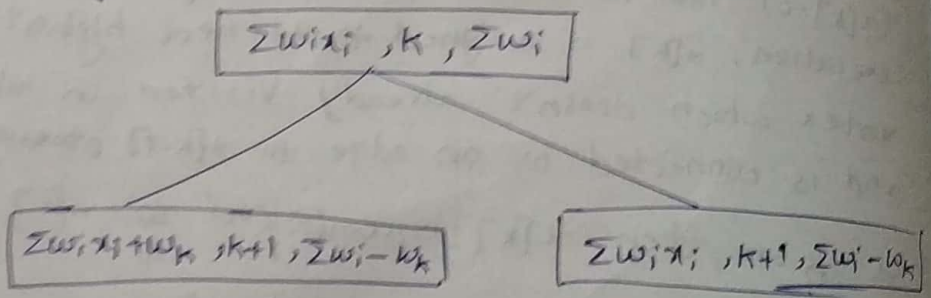
```
If (G=k) then
{
    If ((k=n) or ((k=n) && (G(x[n],x[1])≠0)) then
        return;                          //Edge b/w first and last Vertex
    }
}
}until false;
}
```

The time complexity of Hamiltonian cycle problem is $O(n!)$ or $O(n^n)$

Sum of Subsets

- Draw the state space tree for sum of subsets problem instance, when $w[1:4] = \{7, 11, 13, 24\}$, $M = 31$ and $n = 4$.

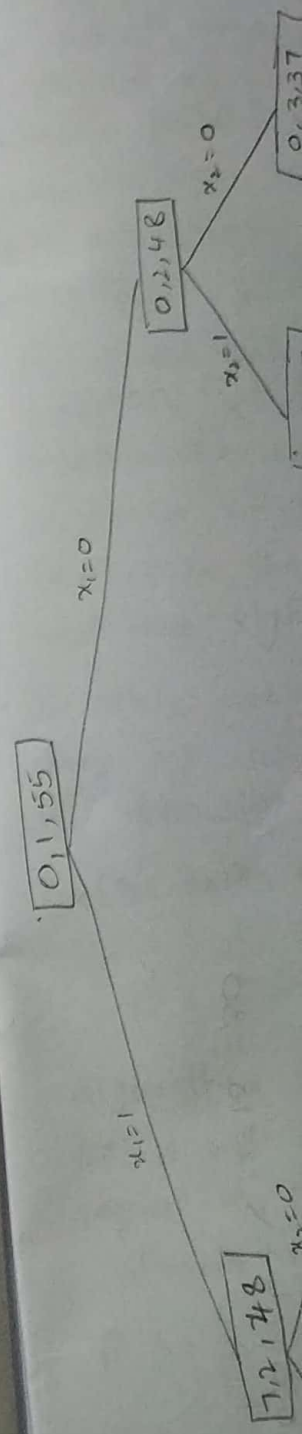
A. The state space can be drawn by using the following rule.



Initially $\Sigma w_i x_i = 0$ because we are not consider any object weight

k is the index of the object in the weight vector $w[1:n]$

Σw_i = Sum of the weights of the n objects



2. Draw state space tree for n=6, M=30
w[1:6] = {5, 10, 12, 13, 15, 18} for sum of subsets problem.

Suppose we are given
usually called the we
the subsets whose sum
is called as sum of
subsets problem can
variable size tuple
a backtracking sol
strategy.

In fixed tuple size
solution is either
whether the weigh
children of any
level i be the le
and the right c

In this problem
they are used
the following co

$B_K(x_1, x_2, x_3, \ldots$

Algorithm Sum
//find out all
equal to M.
already been

// s = $\sum_{i=1}^{k-1} w[i]$

// It is ass

{

// Genera

x[

If

Suppose we are given n distinct positive numbers usually called the weights and we have to find out all the subsets whose sum is equal to M. This problem is called as sum of subsets problems. Sum of subsets problem can be solved using either fixed or variable size tuple formulation. Here we consider a backtracking soln using the fixed tuple size strategy.

In fixed tuple size strategy the element x_i of the solution is either one or zero. Depending on whether the weight w_i is included or not. The children of any node easily. For a node at level i be the left child corresponding to $x_i = 1$ and the right child corresponding $x_i = 0$.

In this problem we have to use bounding functions they are used to kill the live nodes by satisfying the following conditions

$B_k(x_1, x_2, x_3, \ldots, x_k) = \text{True}$ Iff $\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq M$

and

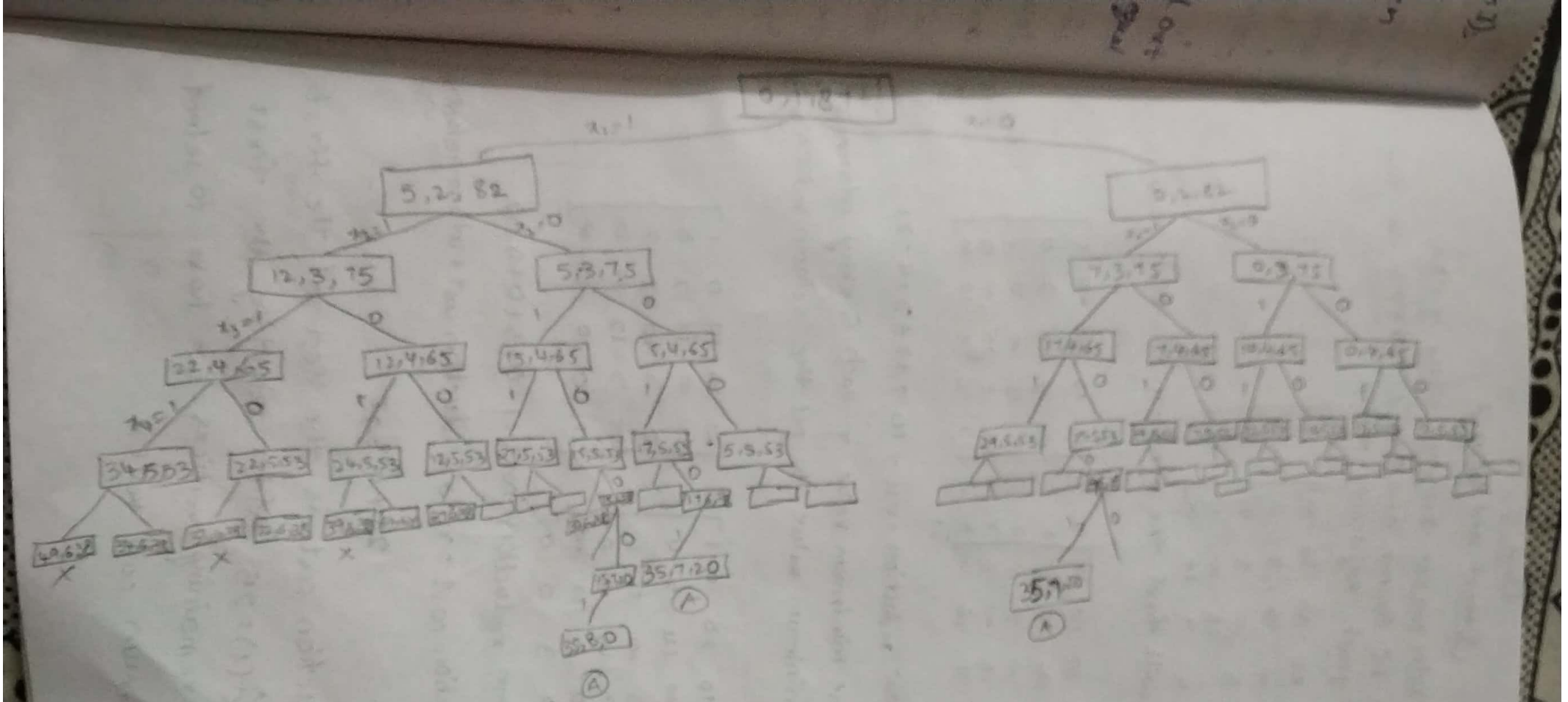
$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq M$

Algorithm Sum of Subsets(s, k, r)
// find out all subsets of w[1:n] whose sum is equal to M. The values of x[j], $1 \leq n \leq j$ have already been determined.
// $s = \sum_{j=1}^{k-1} w[j] * x[j]$ and $r = \sum_{j=k}^{n} w[j]$
// It is assumed that $w[1] \leq M$ and $\sum_{i=j}^{n} w[i] \geq M$
{
// Generate left child
x[k] := 1;
If (s + w[k] = M) then write (x[1:k]);
//Subset found.

```
else if (s+w[k]+w[k+1] ≤ M) then
        SumofSubsets(s+w[k], k+1, r-w[k]);

// Generate right child and evaluate B_k
If((s+r-w[k] ≥ M) and (s+w[k+1] ≤ M)) then
{
    x[k]=0;
    SumofSubsets(s, k+1, r-w[k]);
}
}
```

1. Let $W=\{5,7,10,12,15,18,20\}$ $M=35$, $N=7$. Find out all possible subsets subsets of W whose sum is equal to M. Draw the state space tree.



A = {1, 0, 1, 0, 0, 0, 1} => 5 + 10 + 20 = 35

B = {1, 0, 0, 1, 0, 1, 0} => 5 + 12 + 18 = 35

C = {0, 1, 1, 0, 0, 1, 0} => 7 + 10 + 18 = 35