UNIT – IV

DESIGN AND ANALYSIS OF ALGORITHMS

BACK TRACKING

BASIC METHOD

Back Tracking

The name backtracking was first coined by D.H. Lehmer. in 1950's.

In many applications of the back track method, the desired solution is expressible as an n-tuple $(x_1, x_2, \cdots x_n)$, where the x_i is chosen from some finite set. Often the problem to be solved calls for finding one vector that maximizes (minimizes or satisfies) a criterian function $P(x_1, x_2, \cdots x_n)$. Sometimes it seeks all vectors that satisfies P.

Suppose m_i is the size of set S_i, then there are $m = m_1 m_2 \cdots m_n$ n-tuples that are possible candidates for satisfying the function P. The brute force approach would be to form all these n-tuples, evaluate each one with P, and save those which yield the optimum. The back tracking algorithm has as its virtue the ability to yield the same answer with fewer than m trials.

BASIC METHOD


```
Algorithm Backtrack (k)
{
for each x[k] ∈ T(x[1], x[2], ... x[k-1]) do
    {
    if ( B (x[1], x[2] .... x[k]) ≠ 0) then
        {
        if (x[1], ... x[k]) is a path to
                an answer node)
        then write (x[1:k]);
    if (k<n) then backtrack (k+1);
        }
    }
}
```

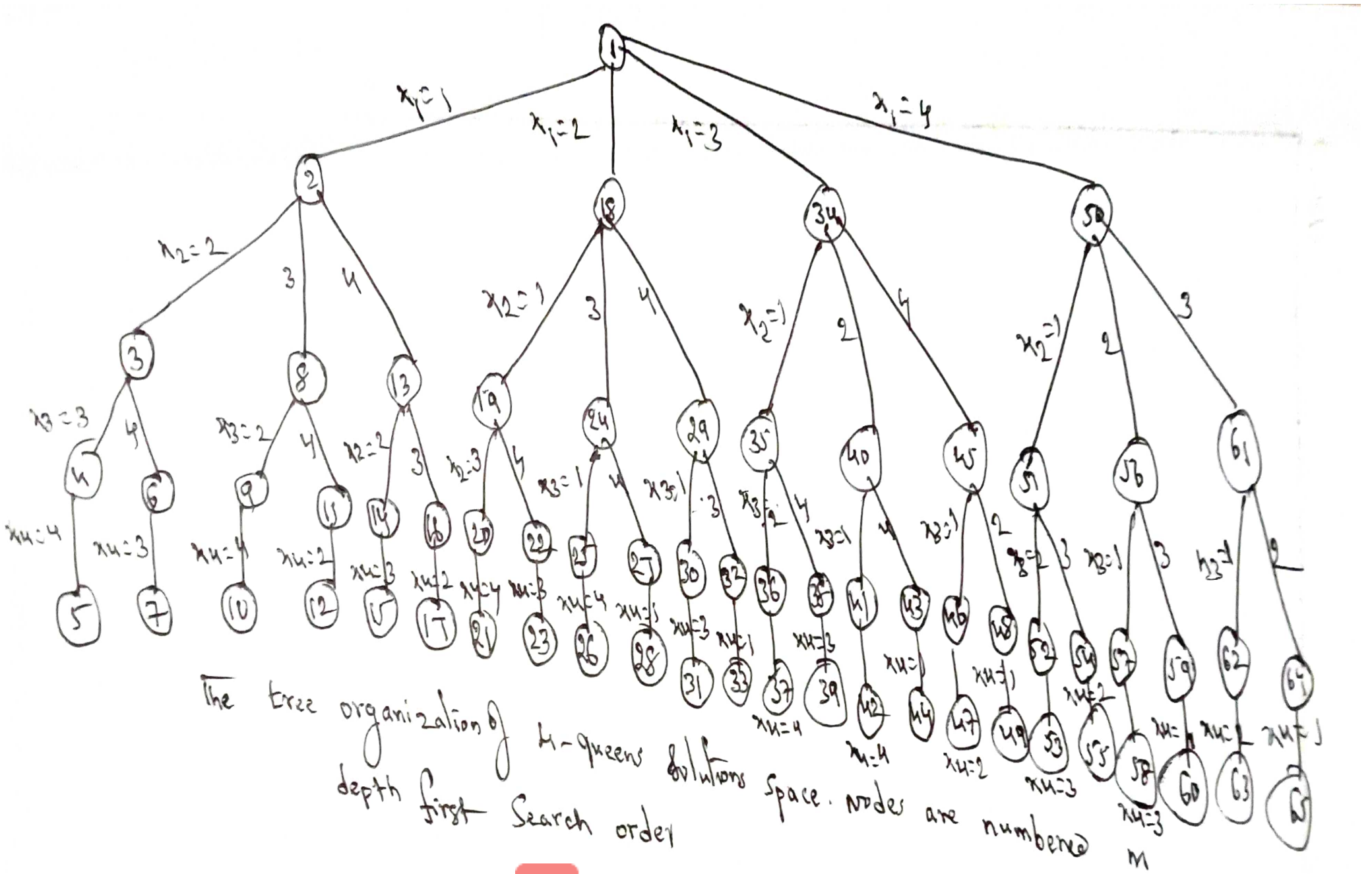
BASIC METHOD

Its basic idea is to build up the solution vector one component at a time and to use modified criterion function $P_i(x_1, x_2, \cdots x_i)$ (sometimes called bounding function) to test whether the vector being found has any chance of success. The major advantage of this method is this: "if it is realized that the partial vector $(x_1, x_2, \cdots, x_i)$ can in no way lead to an optimal solution, then $m_{i+1}, \cdots, m_n$ possible test vectors can be ignored entirely.

E-node: E-node is a node currently being expanded.

4-queen Problem: Let us see how back tracking works on the 4-queens problem. As a bounding function, we use the obvious criteria that if $(x_1, x_2, \cdots x_i)$ is the path to a current E-node, then all children nodes with parent-child labeling x_{i+1} are such that $x_1, x_2, \cdots x_{i+1}$ represents a chess board configuration in which no two queens are attacking each other. We start with the root node as the only live node. This become the E-node and the path is ( ). We generate one child. (Children are generated in ascending order)

N-Queens Problem



The tree organization of 4-queens solutions space. Nodes are numbered depth first search order

N-Queens Problem

Thus node number 2 of above fig is generated and the path is now (1) this corresponds to placing queen 1 in column 1. node 2 becomes the E-node. node 3 is generated and immediately killed. The next node is node 8 and the path becomes (1,3). node 8 becomes E-node it gets killed as all its children represent bound configuration that cannot lead to an answer node. we back track to node 2 and generate another child, node 13. The path is now (1,4). Below fig (2) shows the bound Configuration as back tracking proceeds. This fig (3) shows graphically the stages of back tracking algorithm goes through as it tries to find a solution.

* Live node: Live node is a node that has been generated but whose children have not yet been generated
* Dead node: It is a generated node that is not to be expanded or explored any further.

No two queens should be there in same row, same column or same diagonal.

N-Queens Problem

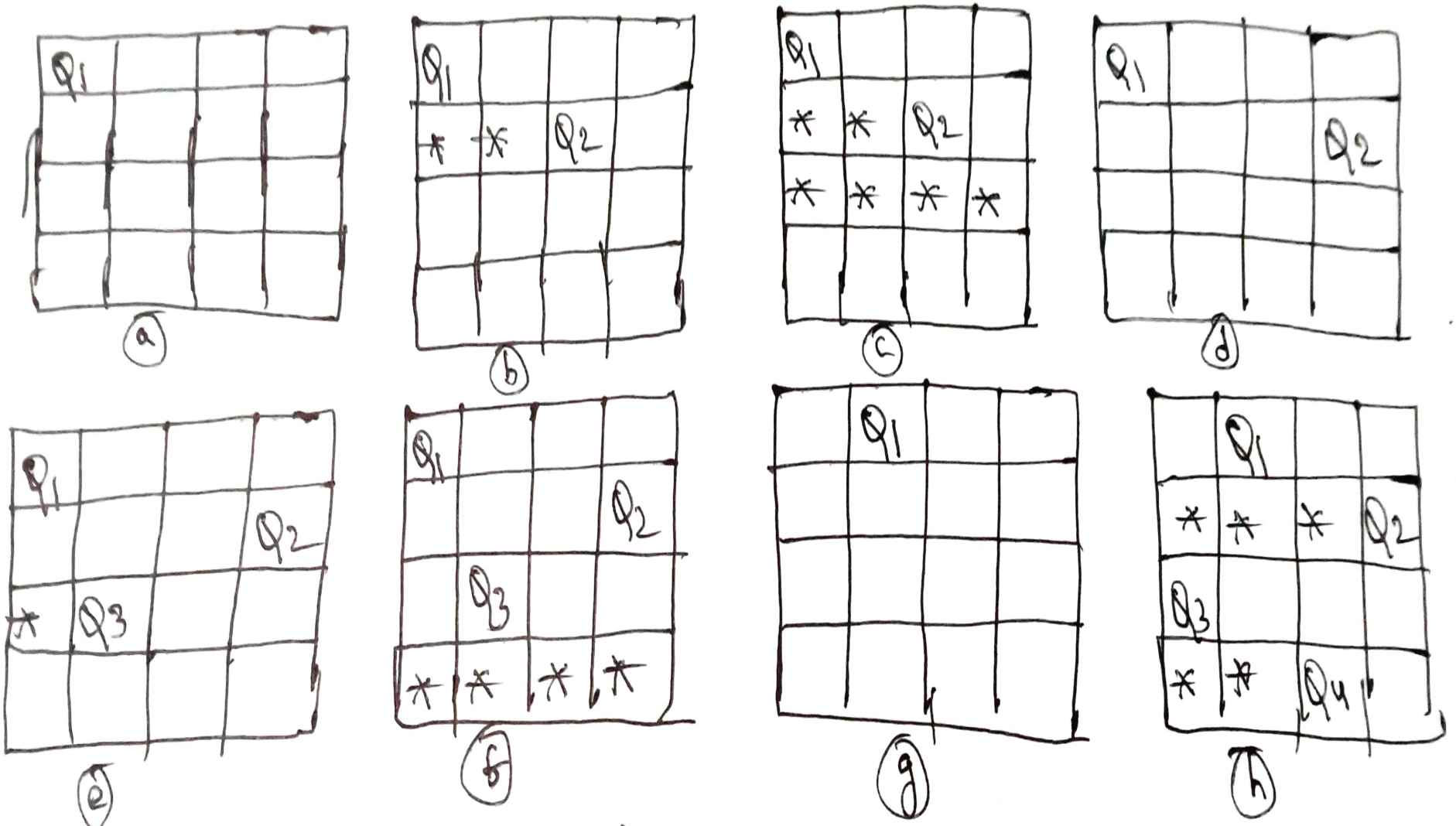


fig (2) Back Tracking solution to 4-Queens problem

N-Queens Problem

Below fig shows the part of the fig (1). The node that gets killed as a result of the bounding function has a B-under that node Fig (1) is containing 65 nodes (State space tree) and below fig (solution tree) contains only 16 node. and this is one solution of 4-Queens problem.

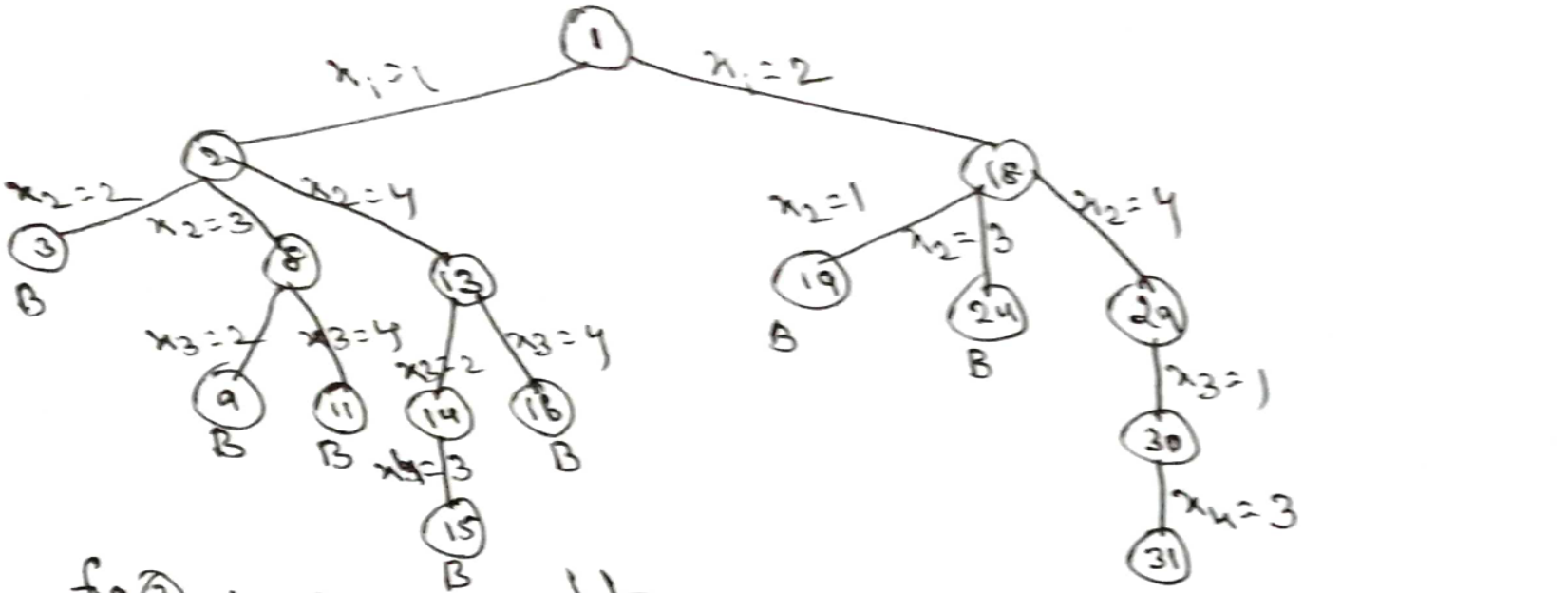


fig (3) 4-Queens solution space tree. Nodes are numbered in DFS order

N-Queens Problem

N-Queens Problem

Consider an $n \times n$ chess board (square board) and try to find all ways to place n non attacking queens.

If we imagine that chess board squares being numbered as the indices of the two dimentional array $a[1:n, 1:n]$, then we observe that every element on the same diagonal that runs from the upper left to the lower right has the same "row - column" value. Also every element on the same diagonal that goes from the upper right to the lower left has the same "row + column" value.

Suppose two queens are placed at positions (i, j) and (k, l) then by the above two are on the same diagonal only if

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

i.e. two queens are on the same diagonal if and only if $|j - l| = |i - k|$

N-Queens Problem

```
Algorithm N-Queens(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 N-Queens(k+1,n);
         }
    }
}

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

N-Queens Problem

For an 8×8 chess board, there are $\binom{64}{8}$ possible ways to place 8 queens, or approximately 4.4 billion 8-tuples to examine. However by allowing only placements of queens on distinct rows and columns we require the examination of at most 8! or only 40,320 8-tuples

The total number of nodes in the 8-queens state space tree are

$$1 + \sum_{i=0}^{7} \prod_{j=0}^{i} (8-j) = 69,281$$

So the estimated number of unbounded node is only about 2.34% of the total no. of nodes in the 8-queen state space tree

Sum of Subsets problem

Sum of Subsets Problem

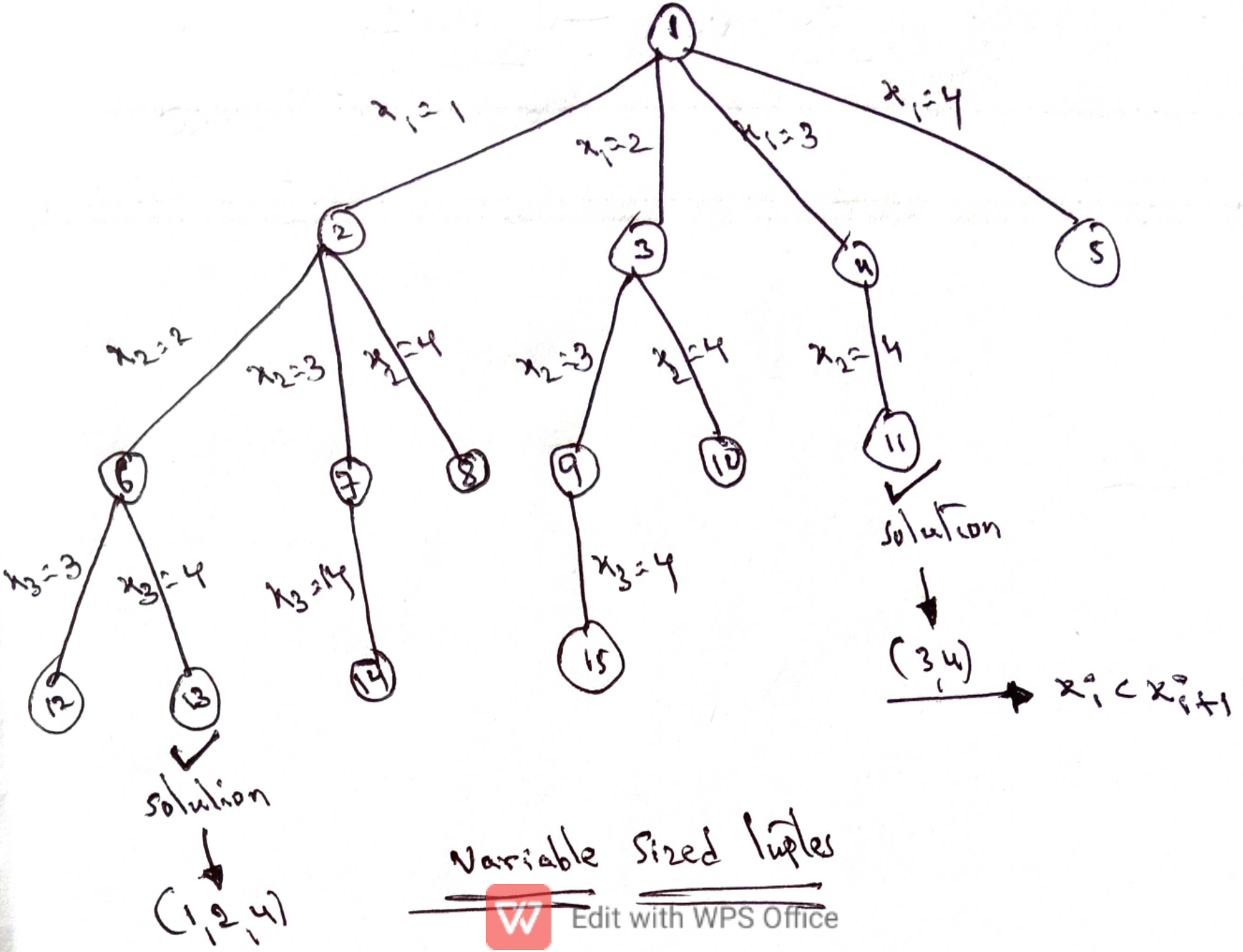
Suppose we are given n-distinct positive numbers (weights) we desire to find all combinations of these numbers whose sums are m. This is called the Sum of the Subsets problem.

We formulate using either "fixed" or "variable sized" tuple size strategy. In this case the element x_i of the solution vector is either 0 or 1 depending on whether the weight w_i is included or not.

Ex: Given positive numbers w_i, $1 \le i \le n$ and m, the problem calls for finding all subsets of the w_i whose sums are m. For ex: $n=4$ and $(w_1, w_2, w_3, w_4) = (4, 13, 24, 7)$ and $m=31$, then the desired solutions are $(1,2,4)$ and $(3,4)$. In general all solutions are n-tuple $(x_1, x_2, \ldots x_n)$

Different solutions have different sized tuples. variable sized tuples

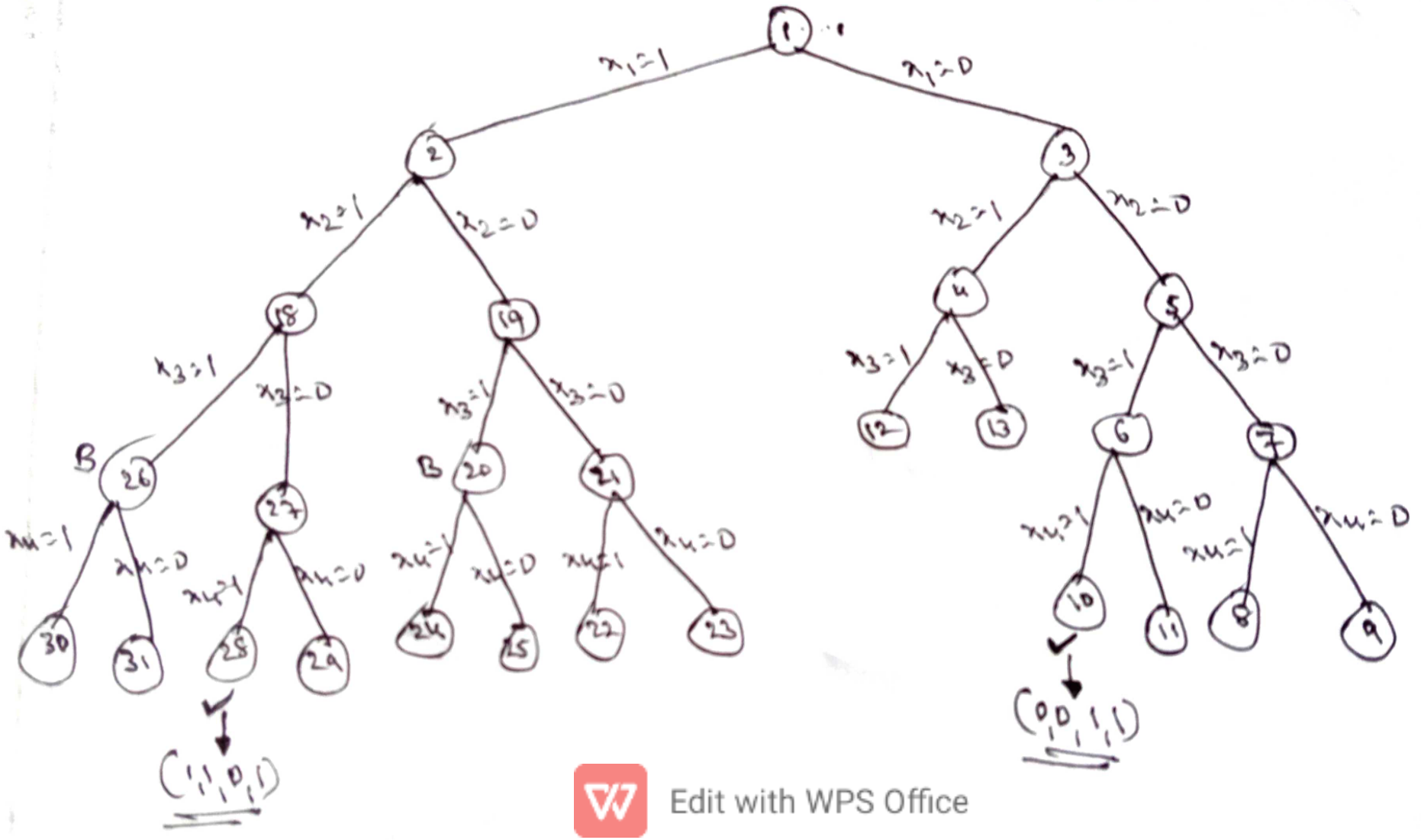
Sum of Subsets problem



Variable Sized Tuples

Sum of Subsets problem

Another problem formulation is, each solution subset is represented by an n-tuple $(x_1, x_2, \ldots x_n)$ such that $x_i \in (0,1)$. $1 \le i \le n$. $x_i = 0$ if w_i is not chosen and $x_i = 1$, w_i is chosen. The solution to the problem of $n=4$, $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$ are $(1,1,0,1)$ and $(0,0,1,1)$ - "Fixed Sized Tuples"



Sum of Subsets problem

For node at level i, the left child corresponds to $x_i = 1$ and the right child corresponds to $x_i = 0$

A simple choice for the bounding function is

$B_k(x_1, \cdots x_k)$ is true if

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

If this condition not satisfied then the above equation cannot lead to solution.

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} > m$$ → cannot lead to answer node

The boundary functions we use are :-

$B_k(x_1, x_2, \cdots x_k) =$ true if —

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

Sum of Subsets problem

Below fig shows the portion of state space tree generated by Sum of subsets on the instance n=6, m=30, and w[1:6]={5, 10, 12, 13, 15, 18}.

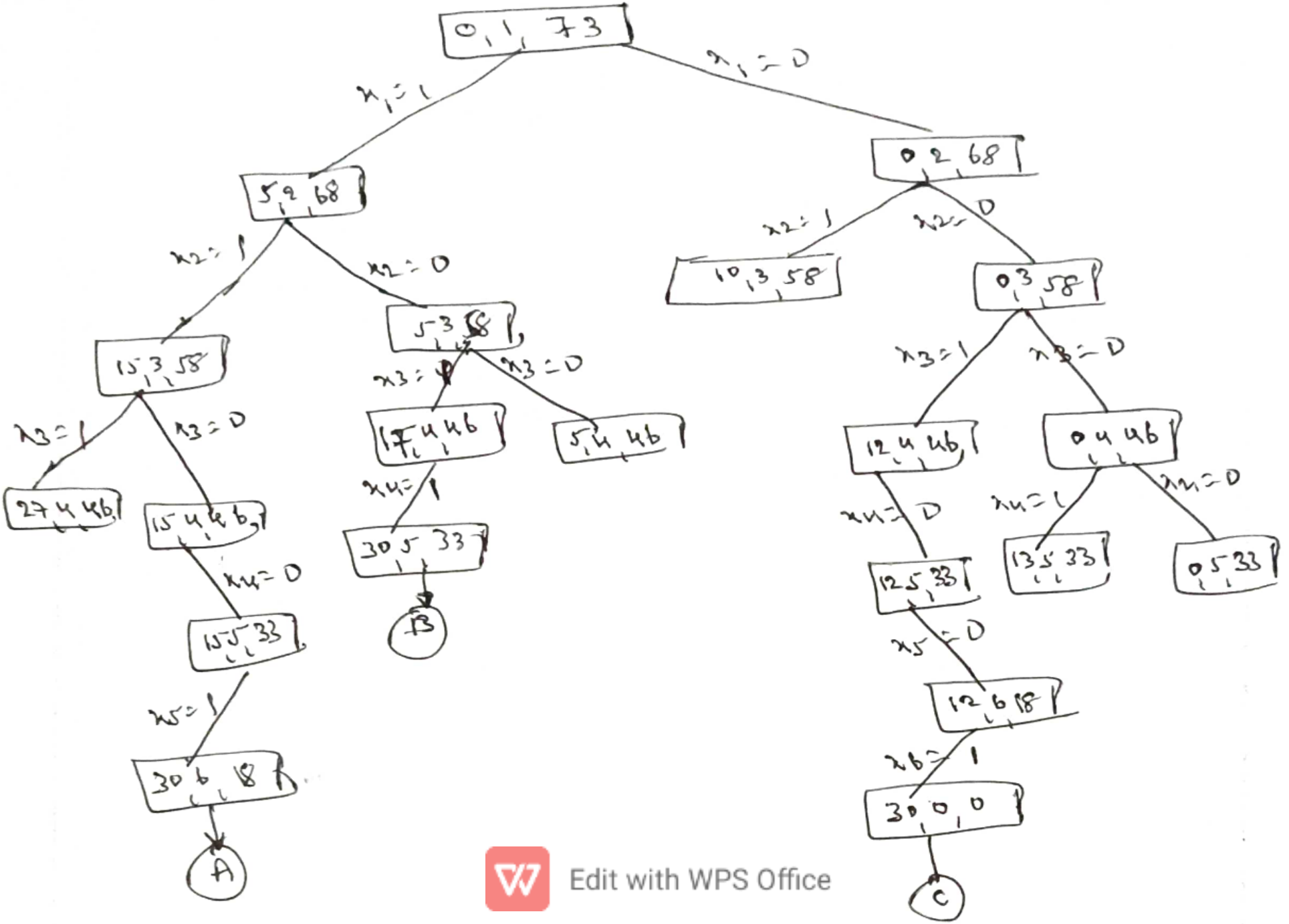
The rectangle node list the values of s, k and r.

s → sum selected k → element number to select

r → remaining elements weight.

Consider nodes represent point at which subset with sum m are printed out. At nodes A, B, C the output represents (1, 1, 0, 0, 1), (1, 0, 1, 1) and (0, 0, 1, 0, 0, 1).

Sum of Subsets problem



Sum of Subsets problem

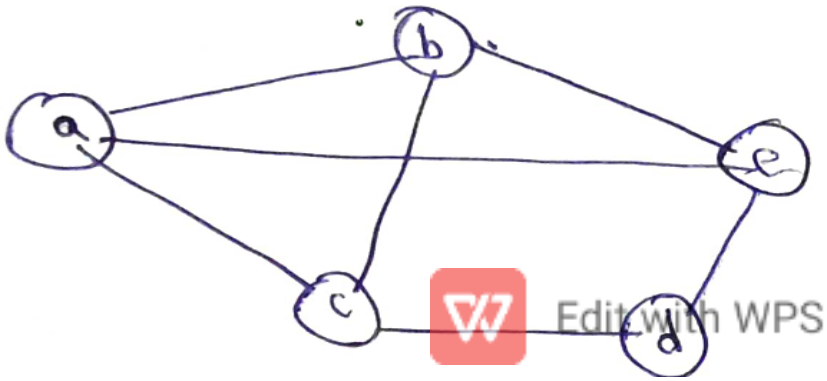
```
Algorithm Sum-of-Sub(s, k, r)
// s = Σ_{j=1}^{k-1} w[j] * x[j]
// r = Σ_{j=k}^{n} w[j] , w[j] are in increasing order.
{
  x[k] := 1
if (s + w[k] = m) then write (x[1:k])
else
  if (s + w[k] + w[k+1] ≤ m)
    then Sum-of-Sub (s + w[k], k+1, r - w[k]);
if ( (s + r - w[k] ≥ m) and (s + w[k+1] ≤ m) ) then
    {
    x[k] := 0
Sum-of-Sub( s, k+1, r - w[k]);
  }
```

Graph Coloring

Graph Coloring

Let 'G' be a graph and m be a positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used. This is termed the "m-colorability decision problem". Note that if d is the degree of the given graph then it can be colored with d+1 colors.

The m-colorability optimization problem ask for the smallest integer m for which the graph can be colored.

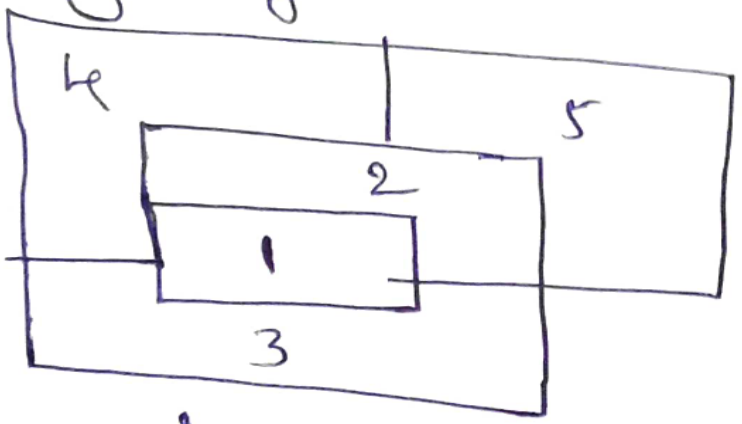


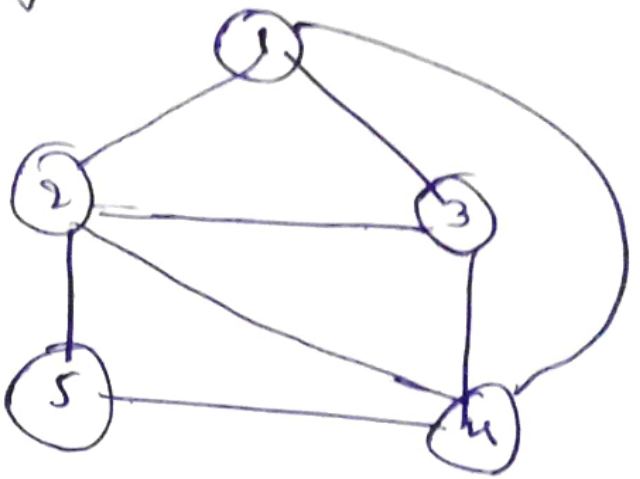
This graph can be colored with 3-colors. So the chromatic number is 3.

Graph Coloring

A graph is said to be planer iff it can be drawn in a plane in such a way that no two edges cross each other. A famous special case of m-coloring decision problem is 4-color problem for planar graph.

Given any map, can the regions be colored in such a way that no two adjacent regions have the same color yet only four colors are needed. Now convert map into graph as if two regions are adjacent Then connect them using edge

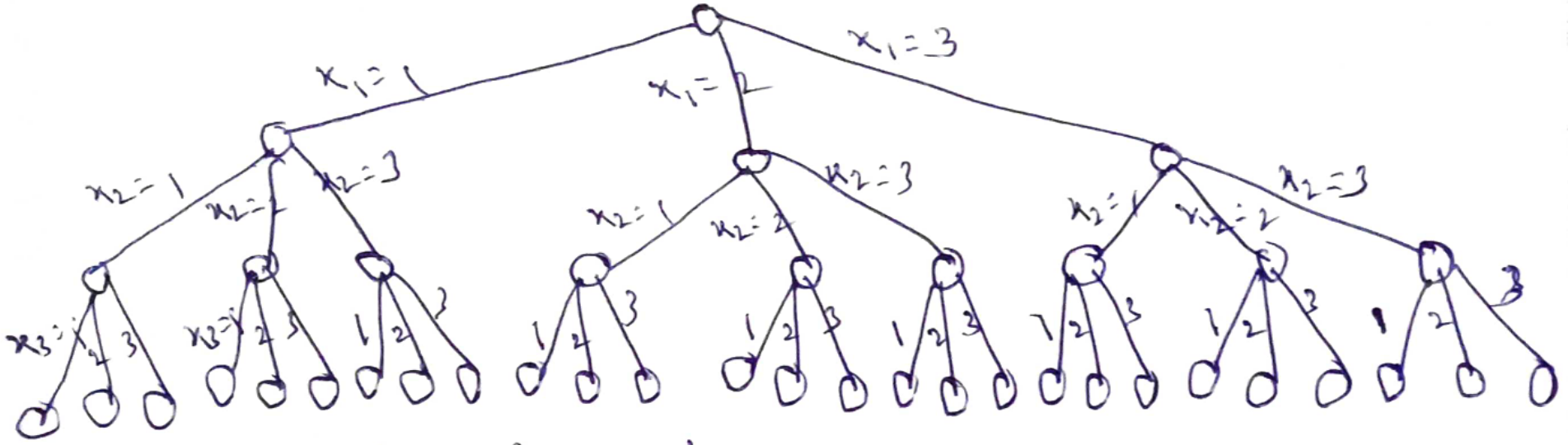




A map and its planar graph representation

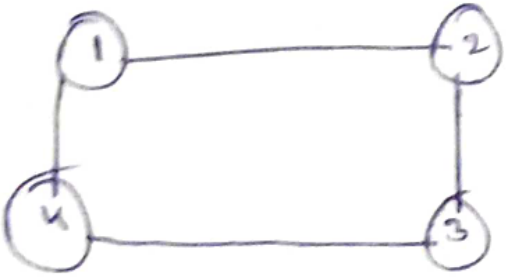
Graph Coloring

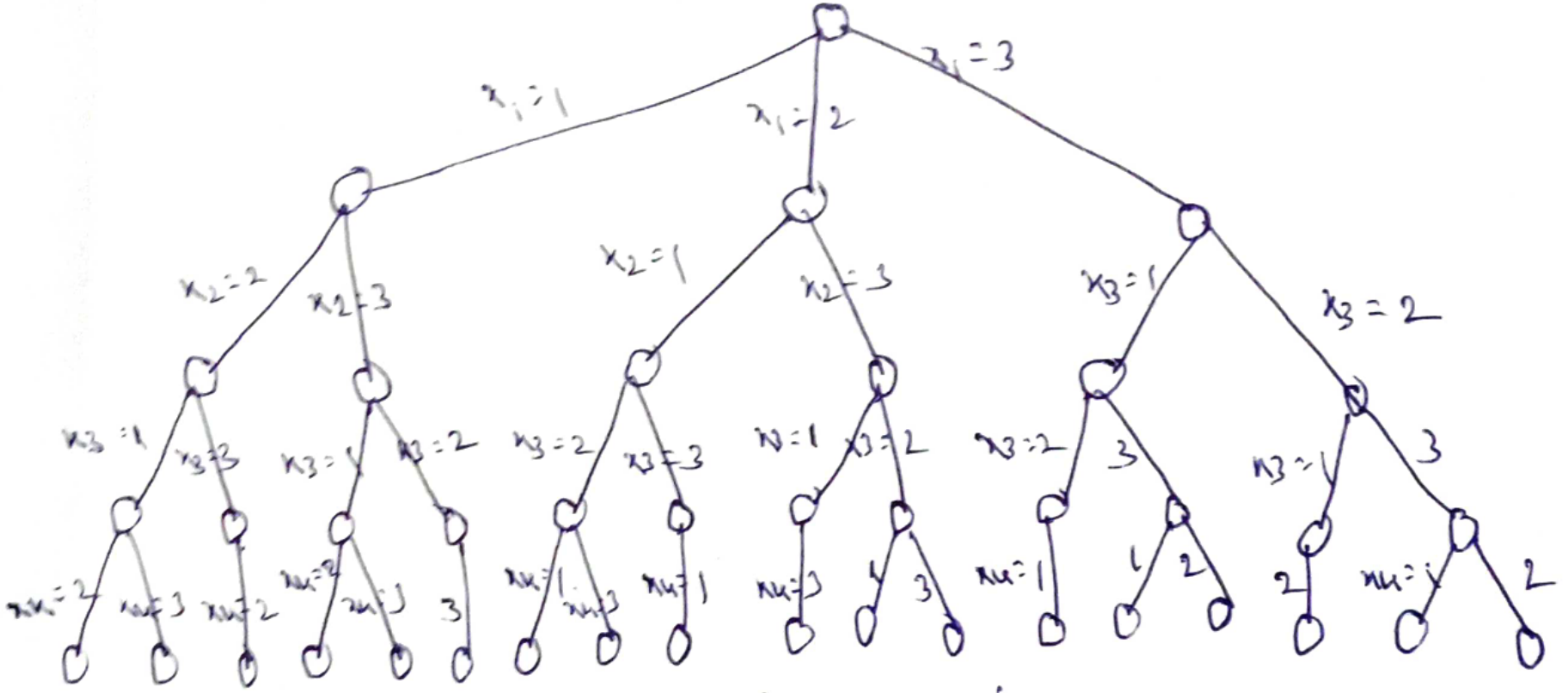
Suppose we represent graph by its adjacency matrix G[1:n, 1:n], where G[i,j] = 1 if (i,j) is an edge of G and G[i,j] = 0 otherwise. The colors are represented by the integers 1, 2, ..., m and the solutions are given by the n-tuple $(x_1, x_2, \ldots x_k)$ where x_i is the color of node i.



State Space tree for m-coloring with m = 3 and n = 3

Graph Coloring

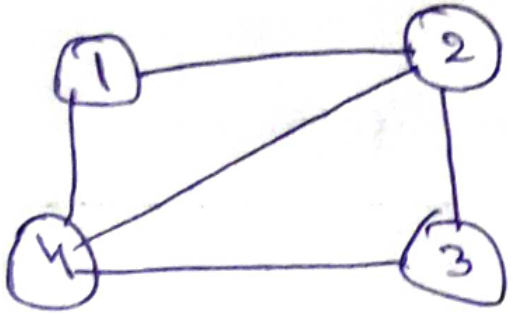




A 4-node graph and all possible 3-colorings.

Graph Coloring

Ex:



G: $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

$n = 4$ and $m = 3$

Initially $x[i] = 0$, $1 \le i \le 4$

mcoloring(1):

$x[1] = (x[1] + 1) \bmod 4$

$= (0 + 1) \bmod 4 = 1 \bmod 4$

$= 1.$

adjacent nodes of 1 are 2 and 4.

$x[2] = 0$ & $x[4] = 0$

So $x[1] \neq x[2]$ and $x[1] \neq x[4]$.

So $\boxed{x[1] = 1}$

Graph Coloring

mcoloring(2): $x[2] = (x[2]+1) \bmod 4$

$= (0+1) \bmod 4 = 1 \bmod 4$

$= 1$

adjacent nodes of node 2 are 1, 3 and 4.

$x[1] = 1$, $x[3] = 0$, $x[4] = 0$

$x[2] = x[1]$.

- $x[2] = (x[2]+1) \bmod 4$

$= (1+1) \bmod 4 = 2 \bmod 4 = 2$

adjacent node colors are $x[1] = 1$, $x[3] = 0$ and $x[4] = 0$

$x[2] \neq x[1]$, $x[2] \neq x[3]$ and $x[2] \neq x[4]$

So $\boxed{x[2] = 2}$

Graph Coloring

mcoloring(3):

$x[3] = (x[3] + 1) \bmod 4 = (0+1) \bmod 4$

$= 1 \bmod 4 = 1$

adjacent nodes to 3 are 2 and 4.

$x[2] = 2$ and $x[4] = 2$

$x[3] \neq x[2]$ and $x[3] \neq x[4]$.

So $\boxed{x[3] = 1}$

mcoloring(4): $x[4] = (x[4] + 1) \bmod 4 = (0+1) \bmod 4 =$

$= 1 \bmod 4 = 1$.

adjacent nodes to node 4 are nodes 1, 2 and 3.

$x[1] = 1$, $x[2] = 2$, $x[3] = 1$.

$x[4] = x[1] = x[3]$.

Graph Coloring

- $x[4] = (x[4]+1) \bmod 4 = (1+1) \bmod 4 = 2 \bmod 4$

$= 2$

$x[1] = 1, \quad x[2] = 2, \quad x[3] = 1$

$x[4] = x[2]$.

- $x[4] = (x[4]+1) \bmod 4 = (2+1) \bmod 4$

$= 3 \bmod 4 = 3$

adjacent node colors are $x[1] = 1$, $x[2] = 2$ and $x[3] = 1$

$x[4] \neq x[1]$, $x[4] \neq x[2]$ & $x[4] \neq x[3]$.

So $\boxed{x[4] = 3}$

Chromatic number of the graph is 3

Graph Coloring

```
Algorithm mcoloring(m)
{
repeat
    {
  x[k] = NextValue(k); // Assign to x[k] a legal color
   if(x[k] = 0) then return; //no new color possible
   if (x[k] =m) then write (x[1:n]);
else mcoloring(k+1);
    } until (false);
}
```

```
Algorithm NextValue(k)
  {
   repeat
     {
       x[k] := (x[k]+1) mode (k+1);
      if (x[k] =0) then return
   for j:= 1 to n do
          { if ((G[k,j] ≠0) and (x[k] = x[j])
      then break;
    }
  if (j = n+1) then return // new color found
  }until (false)
}
```

Hamiltonian Cycle

Hamiltonian Cycle

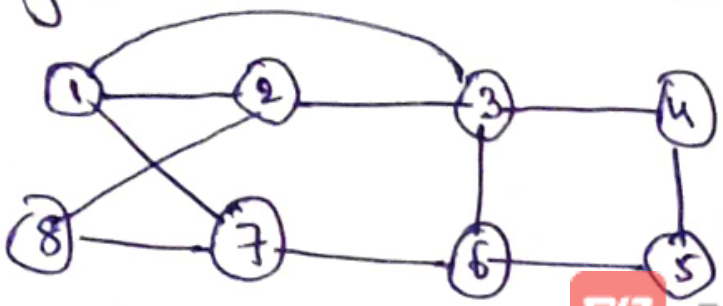
Let $G=(V,E)$ be a connected graph with n vertices.

A Hamiltonian cycle (Sir William Hamilton) is a round-trip path along n-edges of G that visit every vertex once and return to its starting position. In other words if a Hamiltonian cycle begins at some vertex $v_i \in G$ and the vertices of G are visited in the order $v_1, v_2, \ldots v_{n+1}$. Then the edge (v_i, v_{i+1}) are in E, $1 \le i \le n$. and the v_i's are distinct except for v_1 and v_{n+1}, which are same.

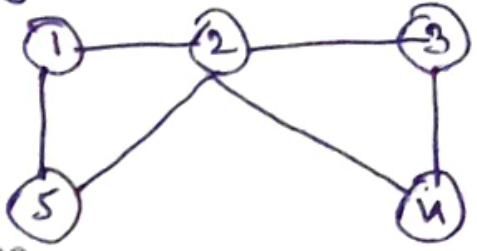
The graph G_1 of below fig, contain the Hamiltonian cycle 1,2,8,7,6,5,4,3,1.

The graph G_2 contain no Hamiltonian cycle.

G_1:



G_2:



Hamiltonian Cycle

We now look at backtracking algorithm that finds all the Hamiltonian cycles in the graph. The graph may be directed or undirected only distinct cycles are output.

The backtracking solution vector is $(x_1, x_2, \ldots x_n)$ is defined so that x_i represent the i^{th} visited vertex of a proposed cycle. Now all we need to do is determine how to compute the set of possible vertices of x_k if $x_1, \ldots, x_{k-1}$ have already been chosen. If $k=1$, then x_k can be any of the n vertices.

The x_k can be any vertex V that is distinct from $x_1, x_2, \ldots x_{k-1}$ and V is connected by an edge to x_{k-1}. The vertex x_n can only be the one remaining vertex which must be connected to both x_{n-1} and x_1.

We begin by presenting function "NextValue(k)", which determines a possible next value (vertex number) for the proposed cycle.

Hamiltonian Cycle

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

Hamiltonian Cycle

```
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 exit;
            if (j = k) then
                if ((k<n) or ((k=n) and G(x[k], x[1]) ≠ 0))
                then return;
        }
    } until (false);
}
```