Tutorial No :- 6

Q.1. Explain general backtracking method
Write note on "Backtracking"

→ - Backtracking represents one of the most general techniques
- Problems which deal with searching for a set of solution or
- Which ask for an optimal soln satisfying some constraints
- can be solved using backtracking formulation.
- In many applications of backtrack method, desire soln is expressible as an n-tuple $(x_1, x_2, \ldots x_n)$ where x_i are shown chosen from some finite set S_i
- Often the problem to be solved calls for finding one vector that, maximizes (or minimizes or satisfies) a criterion fun^n $P(x_1, x_2, \ldots x_n)$
- Sometimes it seeks all vectors that satisfy P.
- e.g. Sorting - Criteria a fun^n P is inequality
- Suppose m_i is the size of set S_i
- Then there are $m = m_1, m_2, m_3, \ldots m_n$ n-tuples
- that are possible candidates for satisfying the fun^n 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 backtrack algorithm has as it's virtue
- The ability to yield the same answer with for fewer than m trials.
- It's basic idea is to build up soln vector one component at time.
- and to use modified criterion fun^n
- $P_i(x_1, \ldots, x_i)$ (sometimes called bounding fun^n)
- To test whether the vector being formed has any chance of success.
- The major advantage of this method is this:

- If it is realized that the partial vector $(x_1, x_2, \ldots x_n)$ can in no way lead to an optimal soln,
- then $m_{i+1} \ldots m_n$ possible test vectors can be ignored entierely.
- Many of problems we solve using backtracking require that all the solns satisfy a complex set of consteaints
- For any problem these consteaints can be divided into two categories:
- explicit and implicit.

Q.2 Define.

a) Implicit Consteaints:

→ - The implicit consteaints are rules that determine which of the tuples in the soln space of I satisfy the criterian function.
- Thus, implicit consteaints describe the way in which the x_i must relate to each other.

b) Explicit Consteaints:

→ - Explicit consteaints are rules that restrict each x_i to take an values only from a given set.
- Common examples of explicit consteaints are

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

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

$l_i \leq x_i \leq u_i$ or $S_i = \{a : l_i \leq a \leq u_i\}$.

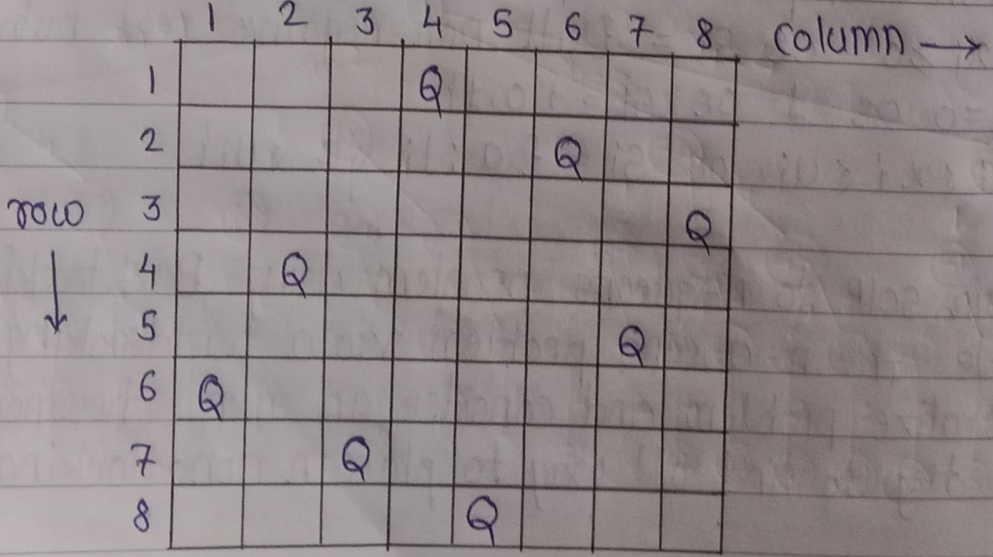
Q.3. Explain soln to n-queens problem using Backtracking method

→ - Tackle the n-queens problem via a backtracking solution.
- Generalize problem and consider an nxn chessboard
- And try to find all way to place n nonattacking queens
-

- Let $(x_1, \ldots, x_n)$ represents a solution in which x_i is the column of the i^{th} row where i^{th} queen is placed.

e.g.

- The 8-queens problem can be represented as 8-tuples $(x_1, \ldots, x_8)$, $S_i = \{1,2,3,4,5,6,7,8\}$, $1 <= i <= 8$.
- The solution space consists of 8^8 8 tuples.
- The implicit constraints for this problem are that no two x_i's can be the same
- (i.e. all queens must be an different columns) & no two queens can be on the same diagonal.
- This realization reduces the size of the soln space from 8^8 tuples to 8! tuples.
- Imagine the chessboard squares being numbered as the indices of the two-dimensional array a[1:n, 1:n],
- Then observe that,
- every element on the same diagonal that runs from the upper left to the lower right
- has the same row-column value.
- For example, in figure consider the queen at a[4,2]
- The squares that are diagonal to this queen a[3,1], a[5,3], a[6,4], a[7,5] and a[8,6]. All these squares have row-column value of 2



- Also, every element on the same diagonal that goes from the upper right to lower left has the same row + column value.
- Suppose two queen are placed at positions (i,j) & (k,l).
- Then by the above they are on the same diagonal only if
- $i-j = k-l$ or $i+j = k+l$.
- The first equation implies & 2nd eqn emplies.
- $j-l = i-k$ & $j-l = k-i$.

∴ two queens lie on the same diagonal if & only if
$|j-l| = |i-k|$.

Q.4. State n-queens problem & write an algo. to test that no. two queens are placed in same diagonal.
Write algorithm to place n non-attacking queens on a chessboard of size nxn using backtracking approach.

→ Ans of Q.3. complete ----

- Algorithm Place (k,l)

```
{
  for i=1 to k-1 do
    if (x[i]=l)          // Two queens in same columns
      OR
    Abs (x[i]-l) = Abs (i-k) // 2 queens on same dia
      Then return false;
    Return True;
}
```

- Algorithm Nqueens (k,n)

```
{
  for l:= 1 to n do
  { if place (k,l) then
    { x[k] := l;
```

```
    if (k=n) then write (x[1:n]);
    else Nqueens (k+1, n);
  }
}
```

- Permutation Tree is used to show the soln space consisting of n! permutations of n-tuples (1,2,---,n)
- The edges are labeled by —
- There are 4! = 24 leaf nodes.

Q.5. Draw and explain state space tree / Permutation tree) Tree organization for 4 queen problem.

→ - Permutation tree is used to show soln space consisting of n! permutations of n-tuples (1,2,---,n).
- The edges are labeled by possible values of x_i.
- Edges from level 1 to level 2 nodes specify the values for x_1
- Leftmost subtree contains all soln with $x_1=1$, $x_2=2$ & so on.
- Edges from level i to i+1 are labeled with values of x_i.
- The soln space is defined by all paths from root node to a leaf node.
- There are 4! = 24 leaf nodes.

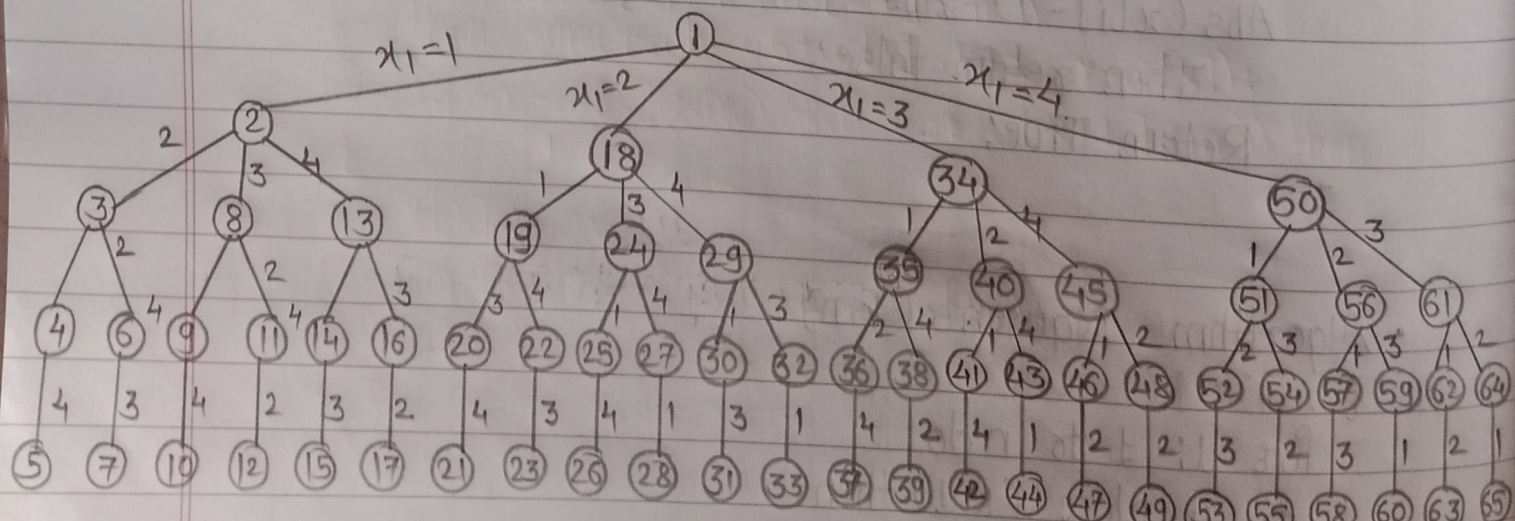
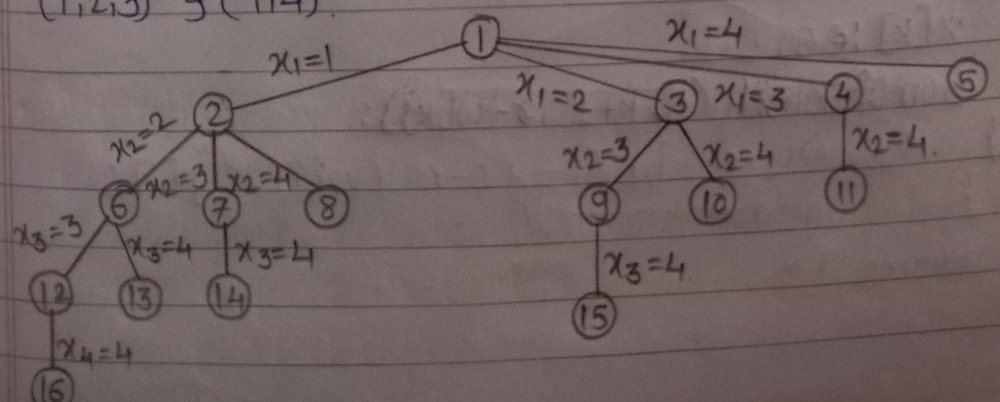


Fig. The organization of the 4-queens solution space nodes are numbered as in depth first search.

Q.6. With suitable example explain sum of subsets problem using backtracking method.

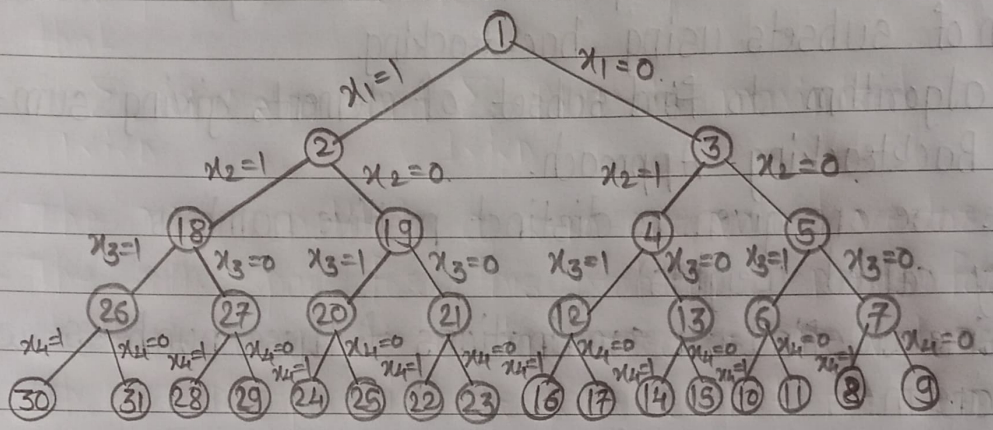
- Explain Backtracking soln to sum of subset problem.
- Explain sum of subset problem & bounding funn used in soln to sum of subsets using backtracking.
- Write algorithm to find subset of elements giving sum = m using Backtracking Approach.

→ - Suppose we are given n distinct positive numbers
- (usually called weights)
- We desire to find all combinations of these numbers whose sums are m.
- This is called the sum of subsets problem
- Consider backtracking soln using the fixed tuple size strategy.
- e.g. Given positive number Wi, $1 \leq i <= n$ & m,
- This problem calls for finding all subsets of Wi whose sums are m.
- For example, if n=4 $(w_1, w_2, w_3, w_4) = (7, 11, 13, 24)$ & m=31.
- Then the desired subsets are (11, 13, 7) & (24, 7). Rather than represent the soln vector by the w_i which sum to m,
- We could represent the soln vector by giving the indices of these w_i. Now the two solns are described by the vectors (1, 2, 3) & (1, 4).



- In this case the element x_i of the solution vector is either one or zero.
- depending on whether the weight w_i is included or not.

S = sum upto k-1 element.
k = current element.
R = sum from k to n.



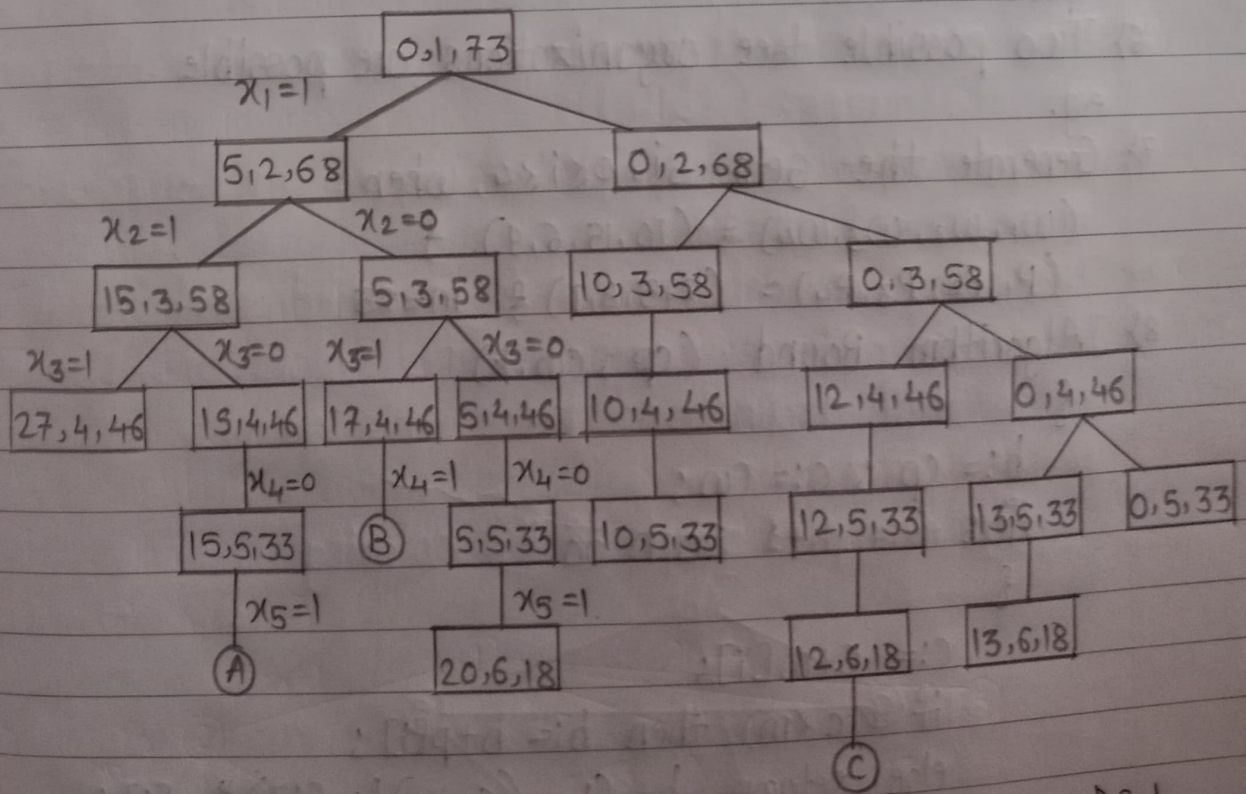
* Algorithm SumOfSub (s, k, r)

```
{
  // Generate left child. Note : s + w[k] ≤ m since
   Bk-1 is true
   x[k] := 1;
   if (s+w(k) = m) then write (x[1:k]);
   else if ((s+w[k]) + (w[k+1] ≤ m))
      then SumOfSub (s+w[k], k+1, r-w[k]);
   if ((s + r - w[k] ≥ m) and (s + w[k+1] ≤ m)) then
   {
      x[k] := 0;
      SumOfSubset (s, k+1, r-w[k]);
   }
}
```

Q.7. Draw possible soln space organization diagram for sum of subset problem.

→ - Diagram shows the portion / space organization tree generated by function sumofsub while working on the instance n=6, m=30, & w[1:6] = {5,10,12,13,15,18}. The rectangular nodes list the values of s, k & r on each of the calls to SumofSub Circular nodes represent points at which subsets with sums m are printed out.

- At nodes A, B & C the output is respectively (1,1,0,0,1), (1,0,1,1) and (0,0,1,0,0,1)
- Note that the tree of diagram contains only 23 rectangular nodes.
- The full state space tree for n=6 contains $2^6 - 1 = 63$ nodes from which calls that could be made



Dia. Space Organization tree generated by SumofSub

Q.8 Explain solution to 0/1 knapsack problem using Backtrack method.
Write algorithm to solve 0/1 knapsack problem using backtracking Approach.

→ 1) Given n positive weights w_i, n positive profits p_i and positive number m that is a knapsack capacity.

2) This problem calls for choosing a subset of weights such that

$$\sum_{1 \le i \le n} w_i x_i \le m \quad \text{and} \quad \sum_{1 \le i \le n} p_i x_i \text{ is maximized.}$$

3) The x_i's constitute a zero-one valued vector.

4) The soln space for this problem consists for the 2^n distinct ways to assign zero or one values to x_i's.

5) Thus soln space is the same as that for sum of subsets problem.

6) Two possible tree organizations are possible.

e.g.

7) Generate the sets s_i, $0 \le i \le 4$, when
$(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$ &
$(P_1, P_2, P_3, P_4) = (2, 5, 8, 1)$ & $m = 25$.

8) Algorithm Bound (cp, cw, k)

```
{
    b := cp ; c := cw;
    for i := k+1 to n do
    {
        c := c + w[i];
        if (c < m) then b := b + p[i];
        else return b + (1 - (c-m)/w[i]) * p[i];
    }
    return b;
}
```

g) Algorithm Bknap (k, cp, cw)
{
  if (cw + w[k] ≤ m) then
  {
    y[k] := 1;
    if (k<n) then Bknap (k+1, cp + p[k], cw + w[k]);
    if ((cp + p[k] > fp) and (k=n)) then
    {
      fp := cp + p[k]; fw := cw + w[k];
      for j := 1 to k do x[j] := y[j];
    }
  }
  if (Bound (cp, cw, k) ≥ fp) then
  {
    y[k] := 0; if (k<n) then Bknap (k+1, cp, cw);
    if ((cp > fp) and (k=n)) then
    {
      fp := cp; fw := cw;
      for j := 1 to k do x[j] := y[j];
    }
  }
}

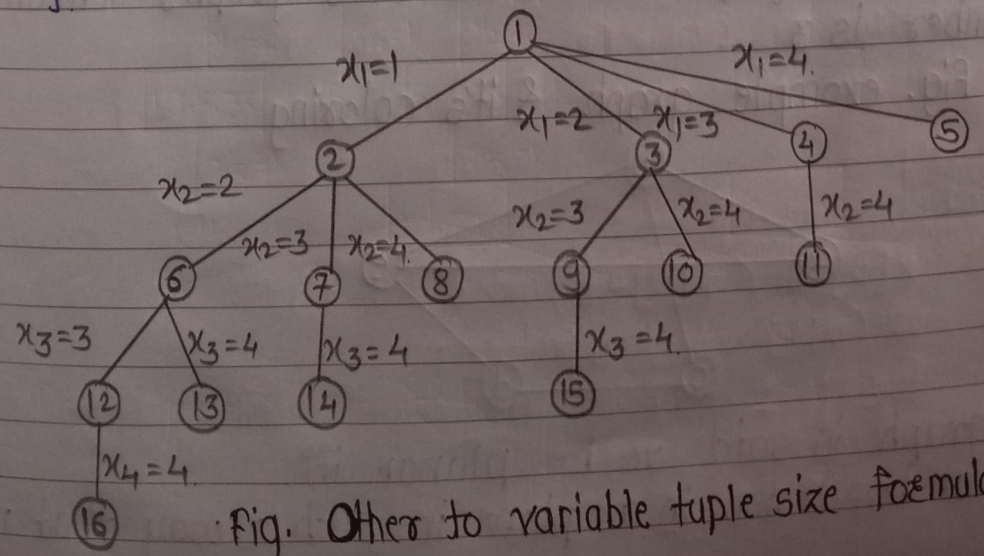
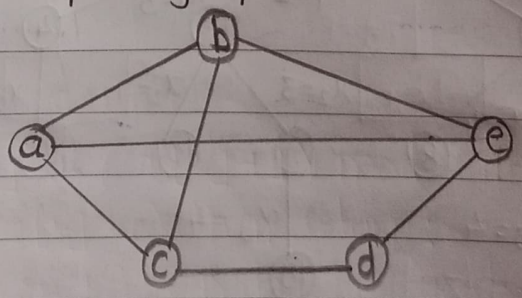


Fig. Other to variable tuple size formulation.

Q.9 - Write note on Graph coloring problem.
- Give backtracking soln to GCP
- Write algorithm to color vertices of graph with unique colors from adjacent vertices using Backtracking Approach.

→ 1) Let G be a graph and m be a given positive integer.
2) We want to discover whether the nodes of G can be colored
3) in such a way that no 2 adjacent nodes have same color
4) Yet only m colors are used.
This is termed the m-colorability decision problem.
5) if d is degree of given graph,
then it can be colored with d+1 colors.
6) The m-colorability optimization problem asks
7) For the smallest integer m for which the graph G can be colored.
8) This integer is referred to as the chromatic number of the graph.
9) For example, the graph of figure can be colored with three colors 1, 2 & 3.
10) The color of each nodes is indicated next to it.
11) It can also be seen that three colors are needed to color this graph. and hence this graph's chromatic number is 3.

Fig. example graph & it's coloring

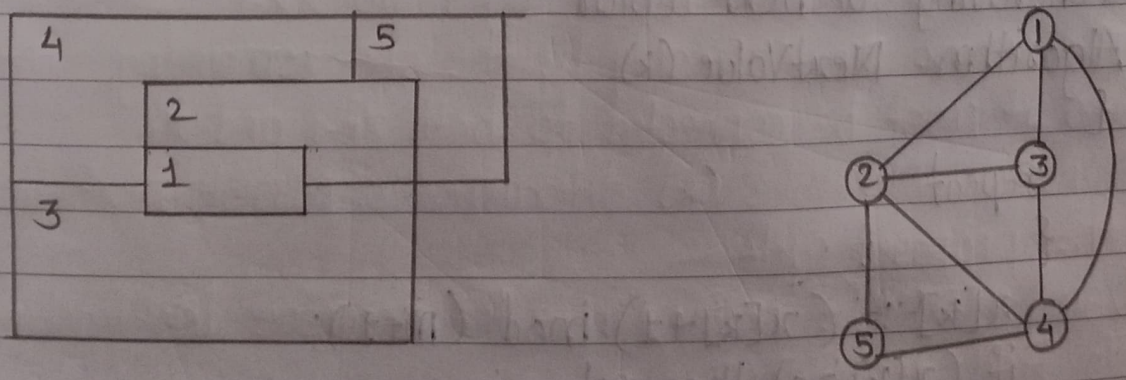


- A graph is said to be planar iff
- it can be drawn in a plane in such a way that no two

edges cross each other.

- A famous special case of m-colorability decision problem is 4-color problem for planar graphs.
- This problems asks the following question.
- Given any map, can the regions be colored in such a way that
- no two adjacent regions have same color yet only four colors are needed.
- This turns out to be problem for which graphs are very useful,
- because a map can easily be transformed into a graph.
- Each region of map becomes a node, & if two regions are adjacent, then the corresponding nodes are joined by an edge.
- Figure shows a map with five regions & its corresponding graph.
- This map requires four colors.
- For many years it was known that five colors were sufficient to color any map, but no map that required more than four colors had ever been found.

Fig. A map and it's planar graph representation.



- We are interested in determining
- all the different ways in which a given graph can be colored using at most m colors.

- Suppose we represent a graph by it's adjacency matrix G [1:n, 1:n],
- Where G [i,j] = 1 if (i,j) is an edge of G, & G [i,j] = 0 otherwise.
- The colors are represented by the integers 1, 2, ---, m
- and the soln are given by the n-tuple $(x_1, \ldots, x_n)$
- Where x_i is the color of node i.
- Using the recursive backtracking formulation the resulting algorithm is mColoring.

- Algorithm mColoring (k)

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

- Generating a next color

```
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]))
```

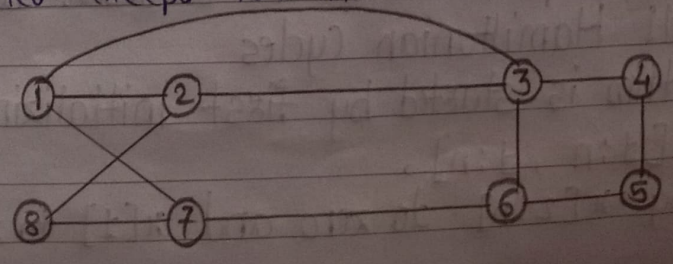
```
        then break;
      }
      if (j = n+1) then return;
    } until (false);
  }
```

- The underlying state space tree used is a tree of degree m & height n+1.
- Each node at level i has m children corresponding to the m possible assignment to x_i, $1 \le i \le n$.
- Nodes at level n+1 are leaf nodes.
- Figure shows the state space tree when n=3 & m=3.
- Computing times is = $O(nm^n)$.

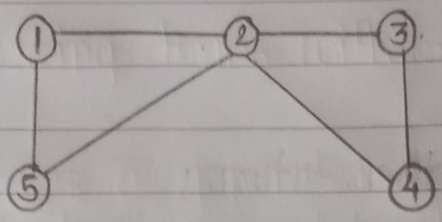
Q.10 - Write note on Hamiltonian Cycle.
- Give backtracking soln to Hamiltonian Cycle.
- Write algorithm to find Hamiltonian Cycle using Backtracking Approach.

→ - 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 once and returns to it's starting position.
- 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 $v_1, v_2, \ldots, v_{n+1}$
- 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 & v_{n+1} which are equal.

G1:



G2:



- The graph G1 contains the Hamiltonian Cycle 1,2,8,7,6,5,4,3,1.
- The graph G2 contains no Hamiltonian Cycle.
- There is no known easy way to determine whether a given graph contains a Hamiltonian Cycle.
- Backtracking algorithm can find all the Hamiltonian cycles in a graph.
- The graph may be directed or undirected.
- Only distinct cycles are output.
- The backtracking soln vector $(x_1, \ldots, x_n)$ is defined as so, that x_i represent the ith visited vertex of the proposed cycle.
- Now all the we need to do is determine how to compute the set of possible vertices.
- For x_k if $x_1, \ldots, x_{k-1}$ have already been choosen.
- To avoid printing the same cycle n times
- We requiree that $x_1 = 1$, if $1 < k < n$, then 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 edges to x_{k-1}.
- The vertex x_n can only be the one remaining vertex.
- and it must be connected to both x_{n-1} and x_1.
- Function NextValue (k) determines a possible next vertex for the proposed cycle.
- Using NextValue particularize the recursive backtracking schema.
- To find all Hamiltonian cycles
- This algorithm is started by first initializing the adjancency matrix G[1:n, 1:n],
- Then setting x[2:n] to zero and x[1] to 1.

and then executing Hamiltonian (2).

- Generating a next vertex.

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

• Finding all Hamiltonian Cycle :-

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);
}
```

Q.11. Explain the following terms with suitable example.

→ a) Live node:

Live node is a generated node for which all of the children have not been generated yet.

b) E-node:

E-node is live node whose children are currently using being generated or explored.

c) Dead node:

Dead node is a generated node that is not to be expanded any further.

- All the children of a dead node are already generated.
- Live nodes are killed using a bounding function to make them dead nodes.

d) Bounding Function:

modified Criteria function.

- Bounding funⁿ is needed to help kill some live nodes without expanding them.
- E.g. Sorting - Comparison.