Backtracking

Course Objective:

To get acquainted with algorithm design technique-backtracking.

Syllabus:

Backtracking: General method, applications-n queen's problem, sum of subsets problem, graph- coloring, and Hamiltonian cycle.

Course Outcomes:
Student will be able to:

- Demonstrate general method of backtracking.
- Solve, design and analyze algorithms for n-queens, sum of subsets, graph- coloring, and Hamiltonian cycle problems by applying backtracking technique.
- Design algorithms for n-queens, sum of subsets, graph- coloring, and Hamiltonian cycle problems by applying backtracking technique.
- Analyze algorithms for sum of subsets, graph- coloring, and Hamiltonian cycle problems.

Backtracking: General Method:

- Backtracking represents one of the most general techniques.
- Problems which deal with searching for a set of solutions or which ask for an optimal solution satisfying some constraints can be solved using the backtracking formulation.
- The name backtrack was first coined by D.H.Lehmerin the 1950.

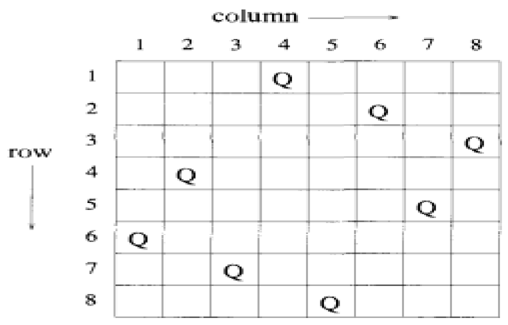
- In the backtrack method, the desired solution is expressible as an n-tuple (x_i,…,x_n), where the x_i are chosen from some finite set Sj.
- The problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a criterion function P(x_i, …., x_n).
- Suppose m_i is the size of set Si. Then there are m = $m_1 m_2 \ldots m_n$ n-tuples that are possible candidates for satisfying the function P.
- The basic idea of backtracking is to build up the solution vector one component at a time and to use modified criterion functions Pi (x_1, …. xi) (sometimes called bounding functions) to test whether the vector being formed has any chance of success.
- The major advantage of this method is: if it is realized that the partial vector (x_1, x_2, ….. x_i) in no way lead to an optimal solution, then m_{i+1} to m_n possible test vectors can be ignored entirely.
- The problems we solve using backtracking require that all the solutions satisfy a complex set of constraints. For any problem these constraints can be divided into two categories: **explicit and implicit.**
    1. **Explicit constraints** are rules that restrict each X_i to take on values only from a given set. The explicit constraints depend on the particular instance I of the problem being solved. All tuples that satisfy the explicit constraints define a possible **Solution Space** for I.

       Examples of explicit constraints are:

       $$\begin{array}{rcrcl} x_i \geq 0 & \text{or} & S_i & = & \{\text{all nonnegative real numbers}\} \\ x_i = 0 \text{ or } 1 & \text{or} & S_i & = & \{0,1\} \\ l_i \leq x_i \leq u_i & \text{or} & S_i & = & \{a : l_i \leq a \leq u_i\} \end{array}$$

    2. **Implicit constraints** are rules that determine which of the tuples in the solution space of I satisfy the criterion function. Thus implicit constraints describe the way in which the X_i must relate to each other.

- **Example 1: 8 –Queens**
  - A classic combinatorial problem is to place eight queens on an 8 x 8 chessboard so that no two "attack" that is, so that no two of them are on the same row, column, or diagonal. Let us number the rows and columns of the chessboard1 through 8.
  - The queens can also be numbered1 through 8.Since each queen must be on a different row, we can without loss of generality assume queen i is to be placed on row i. All solutions to the 8-queensproblem can tbe represented as 8-tuple : **(X_1, ..., X_8)** where X_i is the column on which queen **i** is placed.
  - The **explicit constraints** using this formulation are: **S_i={1,2,3,4,5,6,7,8}.** 1<=i<=8. Therefore the solution space consists of 8^8 8-tuples.
  - The **implicit constraints** for this problem are:
    - No two x_i's can be the same (i.e., all queens must be on different columns). This implies that all the solution s are the permutation of {1,2,3,4,5,6,7,8}. This reduces the size of solution space from 8^8 to 8!.
    - No two queens can be on the same diagonal.
  - The solution for 8-queens is as follows:



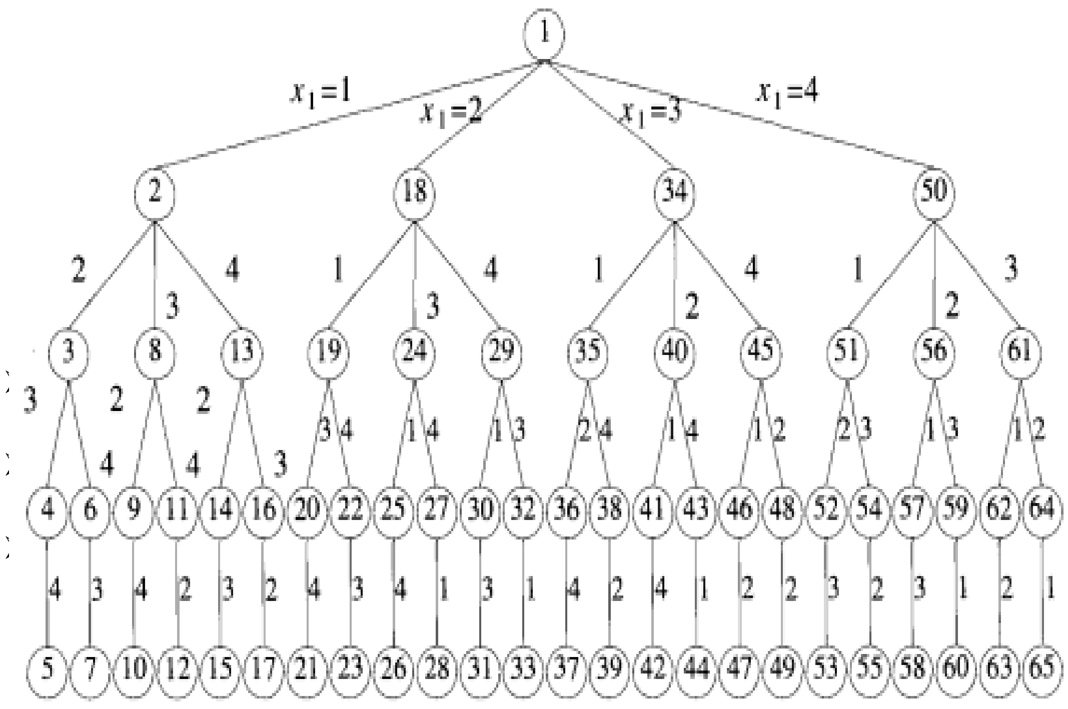
- **Example 2: Sum of subsets**

  - Given positive numbers Wi, 1<= i<= n and m, this problem calls for finding all subsets of the Wi whose sums are m.
  - The **explicit constraints** require X_i € {j \ j is an integer and 1< =j <= n}.
  - The **implicit constraints** require that:
    - No two be the same.
    - The sum of the corresponding w_i's be m.
    - We wish to avoid generating multiple instances of the same subset another implicit constraint that is imposed is that $x_i < x_{i+1}$.
  - If n = 4, (w1, w2, w3, w4) = (11,13,24, 7), and m = 31,then the desired subsets are (11,13,7) and (24,7). The solution vector represented in the following ways:
  - We could represent the solution vector by giving the **indices of these W_i**. Now the two solutions are described by the vectors: **(1, 2, 4) and (3, 4).** Different solutions may have different sized tuples.
  - Each solution subset is represented by an n-tuple (x_i, ……., x_n) suchthat x_i€ {0, 1} 1<=i<=n. Then x_i = 0 if W_i is not chosen and x_i = 1 if W_i is chosen. The solutions to the above instance are : (1,1,0, 1) and (0, 0,1,1). All solutions are of fixed-size.

Tree Organizations

- Backtracking algorithms determine problem solutions by systematically searching the solution space for the given problem instance. This search is facilitated by using a **tree organization for the solution space.** For a given solution space many tree organizations may be possible.

- **Example 1: n-queens**
  - The n-queens problem is a generalization of the 8- queen's problem. Now n-queens are to be placed on an n x n chess board so that no two attack; i.e., no two queens are on the same row, column, or diagonal.
  - The solution space consists of all n! permutations of the n-tuple (1,2,... ,n). The below figure shows a possible tree organization for the case n = 4. Such a tree is called a **"permutation tree".**
  - The edges are labeled by possible values of X_i. Edges from level 1 to level 2 nodes specify the values for x_i. Thus, the leftmost sub tree contains all solutions with x = 1; its left most sub tree contains all solutions with x_1 = 1and x_2=2 and so on.
  - Edges from level i to level i +1 are labeled with the 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 tree.



➢ **Example 2: Sum of Subsets**

- There are two possible formulations of the solution space for the sum of subsets problem: (a) Different sized tuples (b) Fixed size tuples. The tree organizations for these two formulations are given below:

- **Variable tuple size formulation**: The edges are labeled such that an edge from a level i node to a level i +1 node represent a value for x_i. At each node, the solution space is partitioned into sub solution spaces. The solution space is defined by all paths from the root node to any node in the tree, since any such path corresponds to a subset

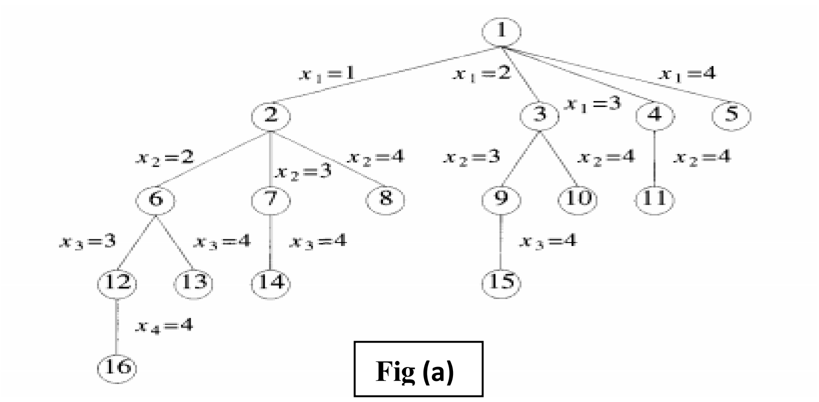


Fig (a)

satisfying the explicit constraints. The left most sub tree defines all subsets containing w_1, the next sub tree defines all subsets containing w_2 but not w_1, and so on.

- **Fixed size tuple formulation:** Edges from level i nodes to level i + 1 nodes are labeled with the value of x_i, which is either zero or one. All

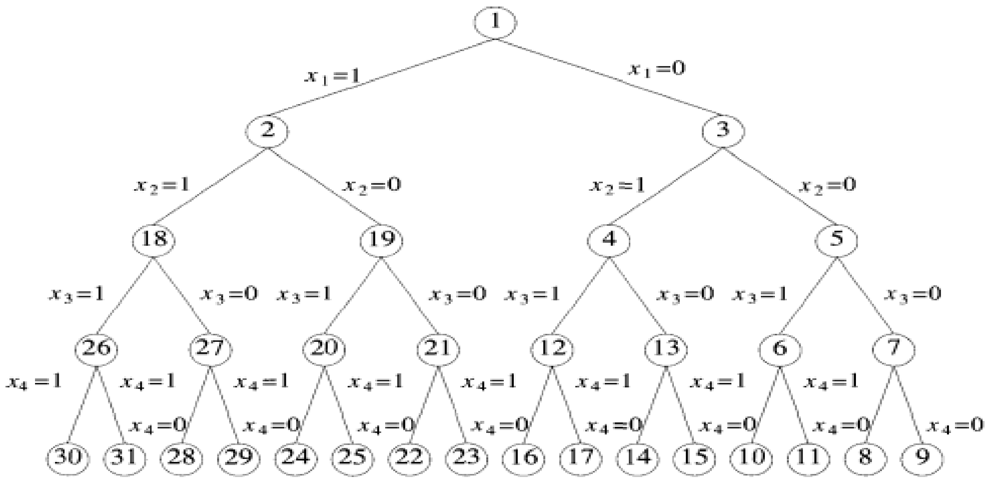


Fig (b)

paths from the root to a leaf node define the solution space. The left sub tree of the root defines all subsets containing w_1, the right sub tree defines all subsets not containing w_1, and so on. Now there are 2^4 leaf nodes which represent16 possible tuples.

- At each internal node in the space tree the solution space is partitioned into disjoint sub-solution space.

➢ Each node in the tree defines a **problem state.** All paths from the root to other nodes define the **State space** of the problem. **Solution states** are those problem states s for which the path from the root to s defines a tuple in the solution space. In the tree of Fig (a) all nodes are solution states where as in the tree of Fig (b) only leaf nodes are solution states. **Answer states** are those solution states s for which the path from the root to s defines a tuple that is a member of the set of solutions (i.e., it satisfies the implicit constraints) of the problem. The tree organization of the solution space is referred to as the state space tree.

➢ **Static Tree:** the tree organizations are independent of the problem instance being solved.

- **Dynamic Tree:** It is advantageous to use different tree organizations for different problem instances. In this case the tree organization is determined dynamically as the solution space is being searched. Tree organizations that are problem instance dependent are called **dynamic trees.**

- Once a state space tree has been conceived of for any problem, this problem can be solved by systematically generating the problem states, determining which of these are solution states, and finally determining which solution states are answer states. There are **two fundamentally different ways** to **generate the problem states**. Both of these begin with the root node and generate other node:

  - A node which has been generated and all of whose children have not yet been generated is called a "**live node"**. The live node whose children are currently being generated is called the **E-node (node being expanded**).A "**dead node"** is a generated node which is not to be expanded further or all of whose children have been generated.

  - In both methods of generating problem states, we have a list of live nodes.

    - In the first of these two methods as soon as a new child C of the current E-node R is generated, this child will become the new E-node. Then R will become the E-node again when the sub tree C has been fully explored. This corresponds to a depth first generation of the problem states.

    - In the second state generation method, the -E-node remains the .E-node until it is dead.

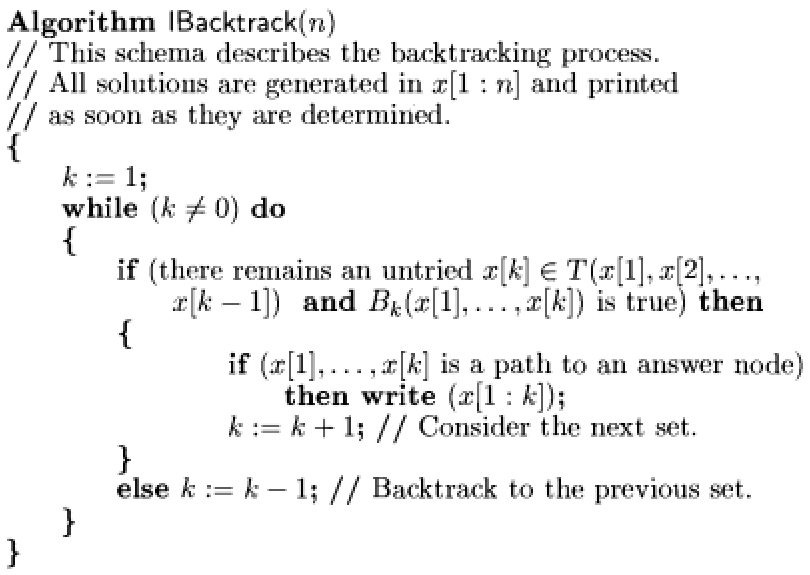
- In both methods, bounding functions are used to kill live nodes without generating all their children. This is done carefully enough that at the conclusion of the process at least one answer node is always generated or all answer nodes are generated if the problem requires us to find all solutions. Depth first node generation with bounding functions is called **backtracking.** State generation methods in which the E-node remains the E-node until it is dead lead to **branch-and-bound methods.**

Algorithms for Backtracking

➢ **Recursive Backtracking algorithm:**

```
Algorithm Backtrack($k$)
// This schema describes the backtracking process using
// recursion. On entering, the first $k-1$ values
// $x[1], x[2], \ldots, x[k-1]$ of the solution vector
// $x[1:n]$ have been assigned. $x[\,]$ and $n$ are global.
{
    for (each $x[k] \in T(x[1], \ldots, x[k-1])$ do
    {
        if ($B_k(x[1], x[2], \ldots, x[k]) \neq 0$) then
        {
            if ($x[1], x[2], \ldots, x[k]$ is a path to an answer node)
                then write ($x[1:k]$);
            if ($k < n$) then Backtrack($k+1$);
        }
    }
}
```

- **Iterative Backtracking Algorithm:**



```
Algorithm IBacktrack(n)
// This schema describes the backtracking process.
// All solutions are generated in $x[1:n]$ and printed
// as soon as they are determined.
{
    $k := 1$;
    while ($k \neq 0$) do
    {
        if (there remains an untried $x[k] \in T(x[1], x[2], \ldots,$
            $x[k-1])$ and $B_k(x[1], \ldots, x[k])$ is true) then
        {
                if ($x[1], \ldots, x[k]$ is a path to an answer node)
                    then write ($x[1:k]$);
                $k := k + 1$; // Consider the next set.
        }
        else $k := k - 1$; // Backtrack to the previous set.
    }
}
```

- The efficiency of both the backtracking algorithm depends on four factors:

  (1) the time to generate the next x_k

  (2) the number of x_k satisfying the explicit constraints,

  (3) the time for the bounding functions B_k and

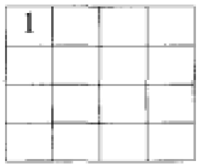
  (4) the number of x_k satisfying the B_k.

<u>n queen's problem</u>

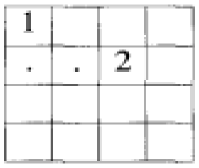
- Let us see how backtracking works on the 4-queens problem. As a bounding function, we use the obvious criteria that if (x_1, x_2 , ….. x_i) is the path to the current E-node, then all children nodes with parent-child labeling x_{i+1} are such that (x1, ….. x_{i+1}) represents a chess board configuration in which no two queens are attacking.
- We start with the root node as the only live node. This becomes the E-node and the path is (). We generate one child. Let us assume that the children are generated in ascending order. Thus, node number 2 is

generated and the path is now (1).This corresponds to placing queen1on column 1. Node 2 becomes the E-node. Node 3 is generated and immediately killed. The next node generated is node 8 and the path becomes (1, 3). Node 8 becomes the E-node. However, it gets killed as all its children represent board configurations that cannot lead to an answer node. We backtrack to node 2 and generate another child, node 13.Thepath is now (1, 4).

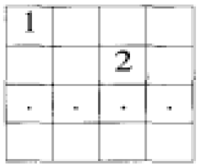
➢ The board configurations as backtracking proceeds are shown below:



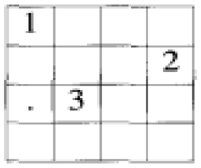
(a)



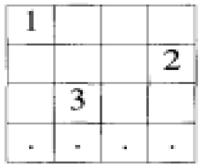
(b)



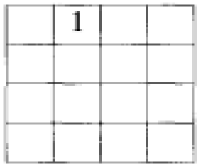
(c)



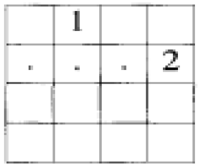
(d)



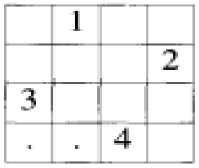
(e)



(f)



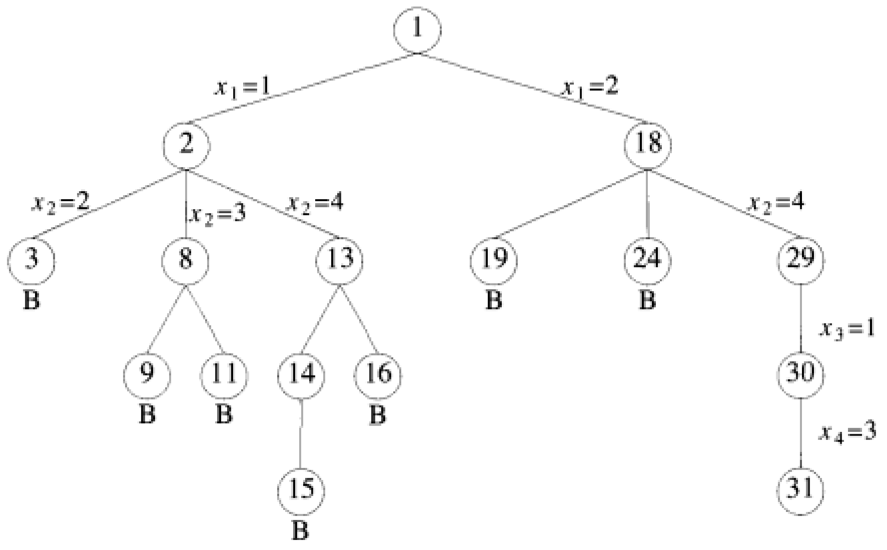
(g)



(h)

➢ The above figure shows graphically the steps that the backtracking algorithm goes through as it tries to find a solution. The dots indicate placements of a queen which were tried and rejected because another queen was attacking. In (b) the second queen is placed on columns1and 2 and finally settles on column3. In (c) the algorithm tries all four columns and is unable to place the next queen on a square. Backtracking now takes place. In (d) the second queen is moved to the next possible column,column4 and the third queen is placed on column2.The boards in (e),(f), (g),and (h)show the remaining steps that the algorithm goes through until a solution is found.

➢ The portion of the tree that is generated during back tracking is as follows:



➢ **A new queen can be placed as follows:**

- Let the chessboard squares are numbered as the indices of the two-dimensional array a[1:n, 1:n]. Every element on the same diagonal that runs from the **upper left to lower right** has the same (**row-column)** value.
- Consider a queen at a[4,2]. The squares that are diagonal to this queen are a[3,1], a[5,3], a[6,4], a[7,5], and a[8,6].All these squares have a **(row-column)** value of 2.
- Every element on the same diagonal that goes from the **upper right to the lower left** has the same **(row+column)** value.
- Let two queens are placed at positions (I, j) and (k, l). Then they are on the same diagonal only if:

i-j=k-l or i+j=k+l

This implies: j-l=i-k

j-l=k-i

- Two queens lie on the same diagonal if and only if **|j-l| = |i-k| .**

➢ **The solution to n-queen problem:**

```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k − 1) values have been set.
// Abs(r) returns the absolute value of r.
{
    for j := 1 to k − 1 do
        if ((x[j] = i) // Two in the same column
             or (Abs(x[j] − i) = Abs(j − k)))
                 // or in the same diagonal
            then return false;
    return true;
}
```

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
    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);
        }
    }
}
```

➢ Place(k, i) returns a Boolean value that is true if the kth queen can be placed in column i. It tests :

- Whether I is distinct from all previous values x[1], x[2], …..,x[k-1].
- Whether there is no other queen on the same diagonal

<u>**Sum of Subsets Problem**</u>

➢ **Problem Definition:** Given positive numbers w_i, 1< =i<=n, and m, this problem calls for finding all subsets of the w_i , whose sums are m.

➢ For a node at level i the left child corresponds to x_i=1 and right to x_i=0.

- The bounding function $B_k(x_1, \ldots\ldots, x_k)$= true iff

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

$x_1, \ldots\ldots, x_k$ cannot lead to answer if the condition is not true.

- If we assume that the w_i's are initially in n on-decreasing order, then $x_1, \ldots\ldots, x_k$ cannot lead to solution if:

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} > m$$

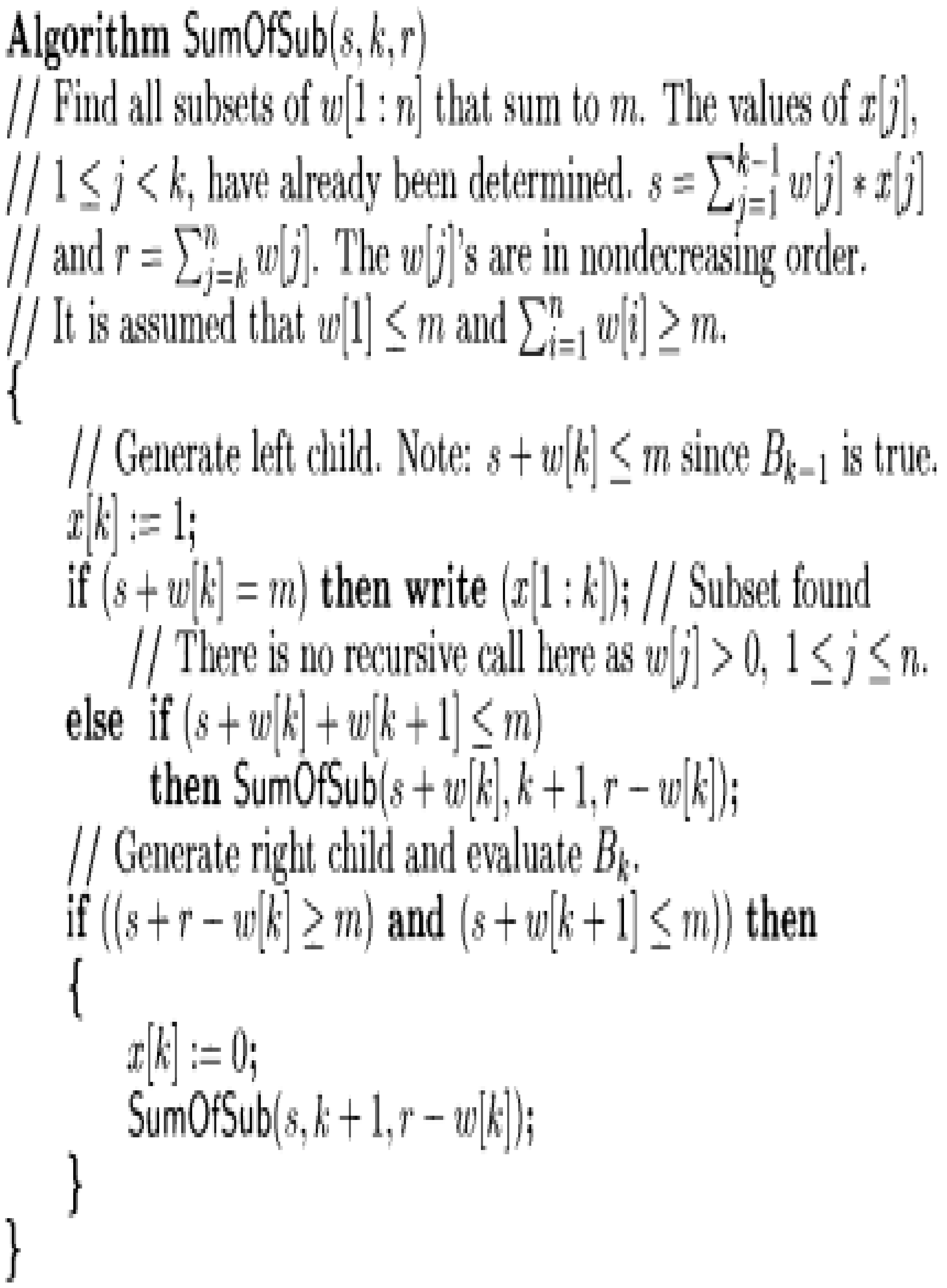
The bounding functions are :

$$B_k(x_1, \ldots, x_k) = true \text{ iff } \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

- If x_k=1, and $\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$ then

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

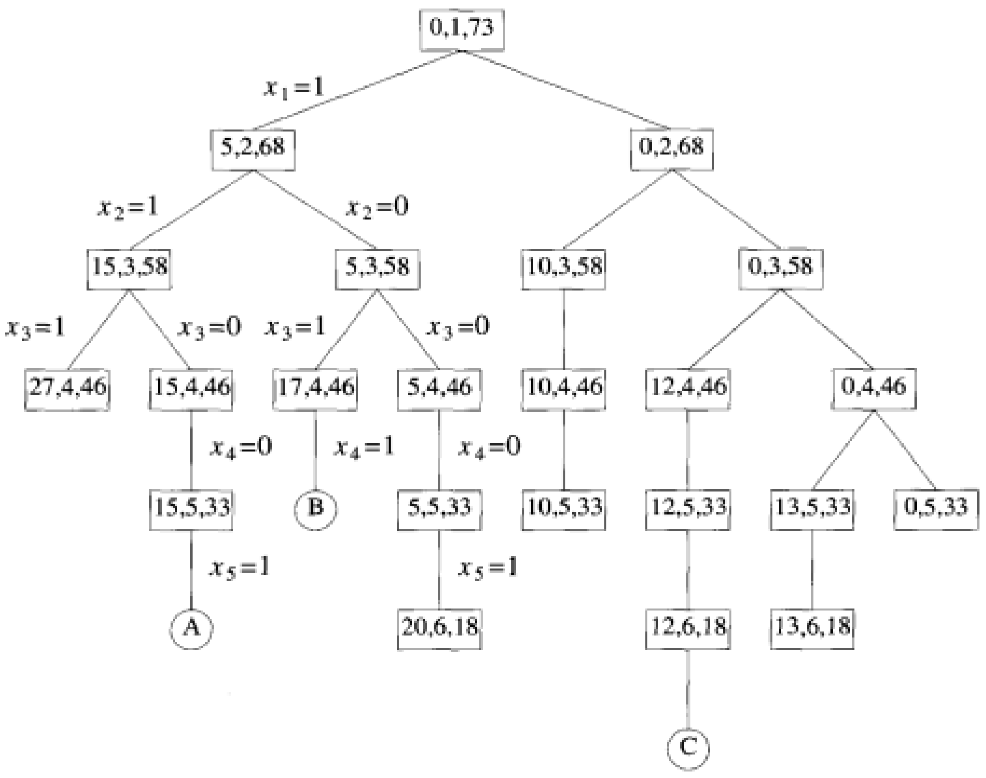
- **Algorithm:**



```
Algorithm SumOfSub(s, k, r)
// Find all subsets of w[1 : n] that sum to m. The values of x[j],
// 1 ≤ j < k, have already been determined. $s = \sum_{j=1}^{k-1} w[j] * x[j]$
// and $r = \sum_{j=k}^{n} w[j]$. The w[j]'s are in nondecreasing order.
// It is assumed that $w[1] \le m$ and $\sum_{i=1}^{n} w[i] \ge m$.
{
    // Generate left child. Note: $s + w[k] \le m$ since $B_{k-1}$ is true.
    x[k] := 1;
    if (s + w[k] = m) then write (x[1 : k]); // Subset found
        // There is no recursive call here as $w[j] > 0, 1 \le j \le n$.
    else if (s + w[k] + w[k + 1] ≤ m)
            then SumOfSub(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;
        SumOfSub(s, k + 1, r − w[k]);
    }
}
```

- **Example: n=6, m=30, w[1:6]= {5,10,12,13,15,18}**

  The rectangular nodes list the values of s, k, and 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, and C the output is respectively

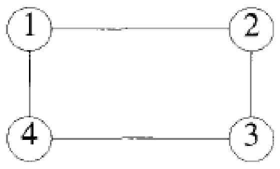


  (1,1,0,0,1), (1,0,1,1),and (0, 0,1,0,0, 1). The tree contains 23 rectangular nodes. The full state space tree contains 2^6-1=63 nodes.

GRAPH COLORING

- Let G be a graph and m be a given 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.
- If d is the degree of the given graph, then it can be colored with d + 1colors.

- The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the **chromatic number** of the graph.
- We are interested in determining all the different ways in which a given graph can be colored using atmost m colors.
- Suppose we represent a graph by its adjacency matrix G[1 : n, 1 : n],whereG[i, j] = 1if (i,j) is an edge of G, and G[i,j] = 0 otherwise.
- The colors are represented by the integers1,2,.mand the solutions are given by then-tuple (x1,........,x_n),where X_i is the color of node i.
- The underlying state space tree used is a tree of degree m and height n + 1.
- Each node at level i has m children corresponding to the m possible assignments to x_i, 1<= i <= n.
- Nodes at level n+1 are leaf nodes.



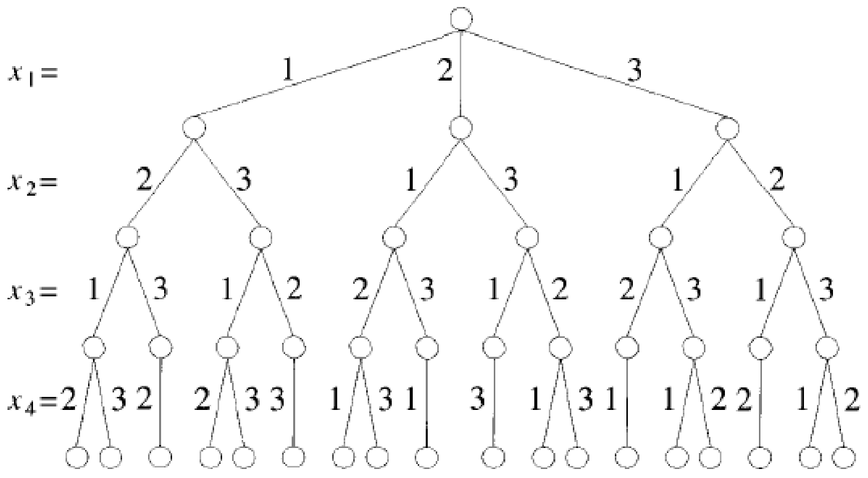


Fig. A 4-node graph and all possible 3-colorings

```
Algorithm mColoring(k)
// This algorithm was formed using the recursive backtracking
// schema. The graph is represented by its boolean adjacency
// matrix G[1 : n, 1 : n]. All assignments of 1, 2, ..., m to the
// vertices of the graph such that adjacent vertices are
// assigned distinct integers are printed. k is the index
// of the next vertex to color.
{
    repeat
    {// Generate all legal 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       // At most m colors have been
                              // used to color the n vertices.
            write (x[1 : n]);
        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. A value for x[k] is determined in the range
// [0, m]. x[k] is assigned the next highest numbered color
// while maintaining distinctness from the adjacent vertices
// of vertex k. If no such color exists, then x[k] is 0.
{
    repeat
    {
        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 if this color is
            // distinct from adjacent colors.
            if ((G[k, j] ≠ 0) and (x[k] = x[j]))
            // If (k, j) is and edge and if adj.
            // vertices have the same color.
                then break;
        }
        if (j = n + 1) then return; // New color found
    } until (false); // Otherwise try to find another color.
}
```

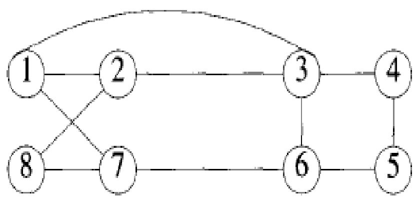
An upper bound on the computing time of mColoring can be arrived at by noticing that the number of internal nodes in the state space tree is $\sum_{i=0}^{n-1} m^i$. At each internal node, $O(mn)$ time is spent by NextValue to determine the children corresponding to legal colorings. Hence the total time is bounded by $\sum_{i=0}^{n-1} m^{i+1}n = \sum_{i=1}^{n} m^i n = n(m^{n+1} - 2)/(m - 1) = O(nm^n)$.

HAMILTONIANCYCLES

- Proposed by Sir William Hamilton.
- 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 its starting position.
- a Hamiltonian cycle begins at some vertex v1 G and the vertices of G are visited in the order v1, v2, v3, ... vn+1 then the edges (vi,vi+1)are in E.

- Consider the following graph:

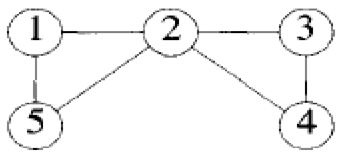
G1:



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

- Consider another graph.

G2:



- The graph G2 contains no Hamiltonian cycle.

The backtracking solution vector $(x_1, \ldots, x_n)$ is defined so that x_i represents the ith visited vertex of the proposed cycle. Now all we need do is determine how to compute the set of possible vertices for x_k if $x_1, \ldots, x_{k-1}$ have already been chosen. If $k = 1$, then x_1 can be any of the n vertices. To avoid printing the same cycle n times, we require 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 edge 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_n.

Using Next Value we can particularize the recursive backtracking schema to find all Hamiltonian cycles.

```
Algorithm NextValue(k)
// x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
// no vertex has as yet been assigned to x[k]. After execution,
// x[k] is assigned to the next highest numbered vertex which
// does not already appear in x[1 : k − 1] and is connected by
// an edge to x[k − 1]. Otherwise x[k] = 0. If k = n, then
// in addition x[k] is connected to x[1].
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        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;
                                  // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct.
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

UNIT-V

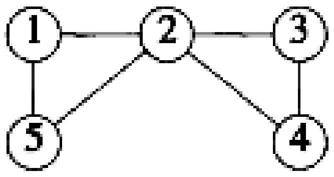
Assignment-Cum-Tutorial Questions

SECTION-A

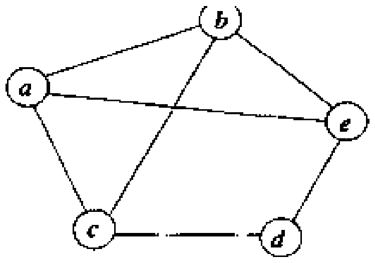
Objective Questions

1. Which of the following problem cannot be solved using a backtracking algorithm? [ ]
   A) Hamiltonian cycle problem B) N queen problem
   C) Tower of hanoi D) Graph coloring problem
2. Which algorithm design technique is used in solving the 8 Queens problem? [ ]
   A) Greedy method B) Dynamic programming
   C) Branch and Bound D) Backtracking.
3. The minimum number of colors needed to color a graph G is called [ ]
   A) Chromatic Number B) Vertex Number C) Edge count D) None
4. ____ are the rules that restrict x_i to take on values only from a given set. [ ]
   A) Explicit constraints B) Implicit constraints
   C) Dynamic constrains D) None
5. ______________ are the rules that determine which of the tuples in the solution space satisfy the criterion function. [ ]
   A) Explicit constraints B) Implicit constraints
   C) Dynamic constrains D) None
6. In which of the following cases n-queen problem does not exist [ ]
   A) n=2 & n=4 B) n=4 & n=6 C) n=2 & n=3 D) n=4 & n=8
7. A following is the solution for 8-queen's problem? [ ]
   A) (4,6,8,2,7,1,3,5) B) (4,4,5,6,7,8,1,2)
   C) (5,6,7,8,4,5,3,2) C) (4,3,3,2 6,1,7,8)

8. Does the following contains Hamiltonian cycle ? Yes/No

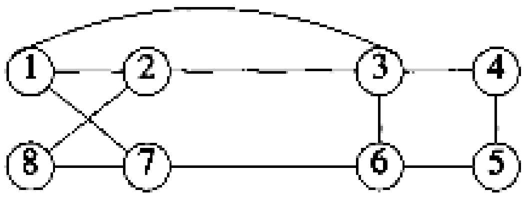


9. The chromatic number for the following graph is _______ [ ]



A) 2 B) 3 C) 4 D) 5

10. The Hamiltonian Cycle for the following graph is [ ]



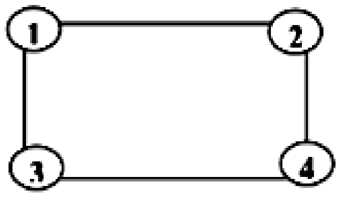
A) 1,2,8,7,6,5,4,3,1 B) 1,3,4,5,6,7 C) 1,2,7,8 D) 1,7,8,2

SECTION-B

SUBJECTIVE QUESTIONS

1. Write the differences between brute force approach and Backtracking.
2. Design a recursive algorithm for backtracking.
3. Write a recursive backtracking algorithm for N queen's problem.
4. Develop a recursive backtracking algorithm for sum of subsets problem.
5. Devise an algorithm for finding all m-colorings of a graph.
6. Formulate an algorithm for finding all Hamiltonian cycles of a graph.
7. Draw the tree organization of the 4-queen solution space and number the nodes using DFS.

8. Apply the backtracking algorithm to solve the following instance of sum of the subsets problem w= {5, 10, 12, 13, 15, 18} and m=30.
9. There are 5 distinct numbers.{1, 2, 3, 4, 5}Find the combinations of these numbers such that the sum is 9. Use the backtracking model to arrive at the solution.
10. Draw the state space tree for m coloring when n=3 and m=3.
11. For the graph given below, draw the portion of state space tree generated by MCOLORING.



12. Find the Hamiltonian circuit in the following graph using backtracking: