UNIT 5 : BACKTRACKING

5.1 LEARNING OBJECTIVE

After going through this unit, you will be able to:

- describe about the concept of backtracking
- know the 8-Queen problem and its solution using backtracking
- elaborate the graph coloring problem
- know the Hamiltonian cycle and its solution using backtracking

5.2 INTRODUCTION

Backtracking is a method for searching a set of solutions or find an optimal solution for satisfies some given constraint to a problem. The name backtrack was first introduce by D. H. Lehmer in 1950's. In this unit we will introduce you the backtracking method. Moreover, we will discuss here about a set of new problems like 8-Queens problem, Graph coloring problem and Hamiltonian Cycle.

5.3 GENERAL STRATEGY FOR BACKTRACKING

In backtracking method the solution set can be represented by an **n** tuple $(x_1,x_2,\ldots\ldots,x_n)$, where x_i are chosen from some finite set S_i. This method can be used for optimization problem to find one or more solution vector that maximize or minimize or satisfy a given criterion function $C(x_1, x_2,\ldots,x_n)$. For example sorting n data of an array **A [1:n]** is a problem to find the solution set, that has an **n**- tuple $(x_1, x_2,\ldots x_i, x_{i+1},\ldots\ldots,x_n)$. For this problem x_i is the index of i^{th} smallest element in the array A .The criterion function is $C(a [x_i] \leq a [x_{i+1}])$ and S_i is the finite set of integers in the range ***[1,n].***

Let the set S_i has size m_i. There are $m = m_1 m_2 \ldots m_n$, **n**- tuples for the solution set that satisfy the criterion function **C**. If brute force approach is applied to find the solution then it will form all n-tuple and solve each on to determine optimal solutions. But if backtracking methods applied here instead of brute force it will take less than m trial to determine the solution. Back tracking method at each step forms a partial solution set $(x_1, x_2,\ldots\ldots,x_i)$ and check it if this has any chances to find a solution depending upon the criterion function C. If this partial solution set no way lead to an optimal solution, then ignore the test vectors from m_{i+1} to m_n entirely.

All the solutions in backtracking require a set of constraints to satisfy. It is divided into two categories:

1. ***Explicit constraint :***

   Explicit constraints are rules that restrict each x_i to take on values only from a given set. This constraint depends on the particular instance I of problem being solved – All tuples that satisfy the explicit constraints define a possible solution space for I.

   Examples of explicit constraints

   i) $x_i \geq 0$, or all nonnegative real numbers

   ii) $x_i = \{0, 1\}$

2. ***Implicit constraints:***

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

There are two types of solution space tuple formulation:

1. ***Variable size tuple:***

   In this method for the solution vector ($x_1, x_2, \ldots, x_k$), x_i will represent indices of i^{th} choices for $1 \leq i \leq k$. Here size of the solution vector can varies for a problem.

2. ***Fixed sized tuple:***

   In this method for solution vector($x_1, x_2, x_3, \ldots x_n$), x_i € { 0,1 } and $1 \leq i \leq n$, such that x_i is 0 if i^{th} element not chosen and 1 otherwise. Here solution vector sizes are same for a problem.

5.4 TREE ORGANIZATION FOR SOLUTION SPACE IN BACKTRACKING

- Backtracking method determine solution of a problem by searching for the solution set in the solution space. This searching can be organized in a tree called ***state space tree.***
- Each node in the tree can be defined as ***problem state.***
- A path from root to any other node defines a ***partial solution vector***, can be called as ***state space.***
- A ***solution state*** is a node ***s*** for which each node from ***root node*** to node ***s*** together can represent a tuple in solution set.
- ***Answer states*** are solution state which satisfies an implicit constraint.
- The ***tree organization*** of ***solution space*** is referred to as ***state space tree.***
- In state space tree problem state are generated from root node and then generated other nodes.
- A ***live node*** is a generated node, for which all of its children node have not yet generated.
- A ***E-node (Expanded node)*** is a live node, whose children are currently being generated.
- A ***dead node*** is that , which is not expanded further and all of its children is generated.
- ***Bounding functions*** are used to bound the searching in the tree. It ***kills a live node without generating children if it does not lead to a feasible solution.***
- Depth first node generated with bounding function is called ***backtracking.***
- In ***state space tree***
    i. ***root*** of the tree represent ***0 choices***.
    ii. ***1^{st} level node*** represents ***1^{st} choices***
    iii. ***2^{nd} level node*** represent ***2^{nd} choices***.
    iv. ***n^{th} level node*** represent ***n^{th} choices***.

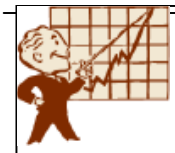
- A node n is called ***non-promising*** if it can not lead to a feasible solution and for this node n bounding function ***B(n) = 0.*** Otherwise, it is called ***promising node*** and bounding function ***B(n)=1.***
- If node is non-promising then it is bounded or kill using bounding function. Then for this node its sub-trees are not generated.
- A state space tree is called ***pruned state space tree*** if it consist of only expanded node .

5.5 MAIN IDEA FOR BACKTRACKING

Backtracking method do depth first search of a state space tree. If a node is promising i.e B(n)=1 then search is continue to its child node , otherwise if a node is non promising i.e B(n)=0 ,backtrack to its parent node.

Recursive algorithm for general backtracking is-

```
Backtrack (node n)
{
    if C(n) = 1
            Report feasible solution n
    else
            Stop
    if (B(n) = 0) return;
    for every child n' of n Backtrack(n')
}
The procedure is invoked by Backtrack(root)
```



CHECK YOUR PROGRESS

1. State True/False:

i. Backtracking method can reduced search space in the state space tree.

ii. Nodes whose children are being generated is called live nodes.

iii. In variable sized tuple method size of solution vector can varies.

iv. Pruned state space tree consist only expanded node.

5.6 8-QUEEN'S PROBLEM

Given a chess board of field 8 x 8. The 8-Queen problem is to place 8-Queen on the chess board, so that no two Queen can "attack" each other. A Queen can attack vertically, horizontally and diagonally.

N-Queen Problem:

It is a generalized problem of 8-Queen problem. N Queens are placed on a chess board of size n x n, without having attack each other.

In chess, queens can move all the way horizontally, vertically or diagonally (if there is no other queen in the way).But, no two Queen can attack each other. So, due to this restriction, each queen must be on a different row and column.

Backtracking strategy for 8-Queen problem is as follows-

1. Let us, in the chess board rows and columns are numbered from 1 to 8 and also queens are numbered from 1 to 8.

2. Without loss of generality, assume that i^{th} queen can be placed in i^{th} row, because no two queen can place in the same row.
3. All solution can represented as 8-tuple (x_1, x_2,…. x_8), where x_i is the column number of the i^{th} row of i^{th} queen placed.
4. Here explicit constraints are S_i = {1, 2, 3, 4, 5, 6, 7, 8}, $1 \le i \le 8$ and the solution space will consist of 8^8 8-tuple.
5. According to the implicit constraints no two queen can on the same row.
6. So, all solution are permutation of 8-tuple(1, 2, 3, 4, 5, 6, 7, 8)
7. Thus the searches is reduce to 8^8 8-tuple to 8! tuple.

We know from the above that, in 8 -Queen problem all the solution can represented as 8-tuple (x_1, x_2,….,x_8), where x_i is the column number of i^{th} row where.i^{th} queen placed. These all x_i's are distinct because of the implicit constraint that no two queen can placed in same column. We assume already in no. 2 that i^{th} queen can be placed in i^{th} row only. So, no two queen can placed in same row. Now, we have only to decide whether two queens are on the same diagonal or not.

If chess board square fields are numbered as two dimensional array a [1:8] [1:8] then we find that for all diagonal element "row-column" value is same. For example if 1^{st} queen is placed in a [1,3] square then the 2^{nd} queen will placed diagonally if it placed in a [2,4] square. Here "row-column" values are 2 and it is same for both queen.

Let two queen are placed in position (m, n) and (x, y)
Then two queens can placed diagonally if

m - n = x - y----------(1)

m + n = x + y----------(2)

(1) => n – y = m - x

(2) => n – y = x – m

Therefore two queens can be on the same diagonal if and only if

| n – y | = | m – x |

This is an another implicit constraint .

Example:

Here is an example of 4-queen problem-

The state space tree generated by 4-queen problem is as follows-

Here node at level i represent i^{th} queen placed at i^{th} row. i.e at level 1 it represents as 1^{st} queen places in 1^{st} row. X_i in i^{th} level represents column number of i^{th} queen placed in i^{th} row. i.e $x_2 = 3$ at level 2 means 2^{nd} queen is in 3^{rd} column of the 2^{nd} row.

Nodes are generated in depth first search manner.

A the path from root to leaf will represent a tuple in solution space. All tuples are distinct and some tuples may not lead to a feasible solution.

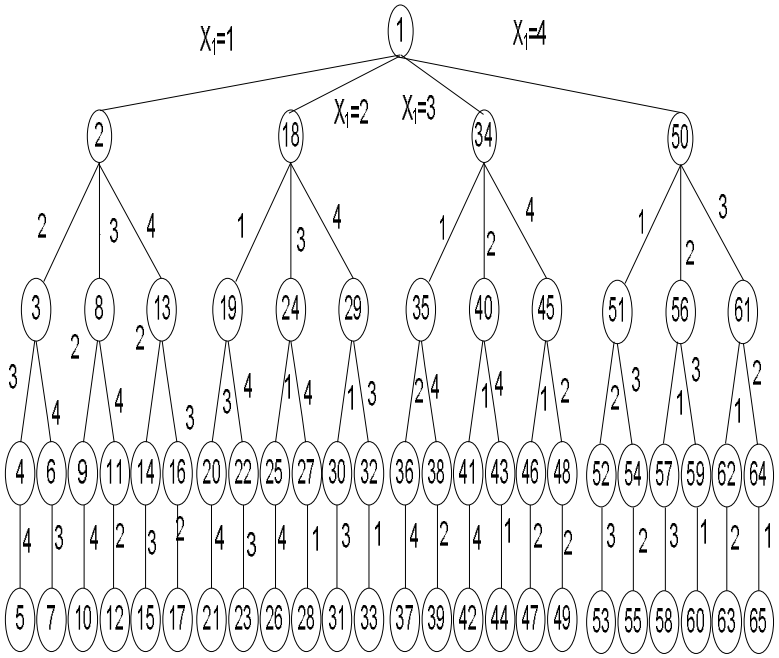


Fig. 5.1 State space tree

Now by using backtracking method we can bound the search of the state space tree using some constraint so that searching require less time.

For this problems to bound a node n constraints or bounding conditions B(n) are-

1. No two queens can place in same row i.e x_i always represents i^{th} queen is in i^{th} row.
2. No two queen in same column i.e values of x_i's are always distinct.
3. For two queen placed in (m, n) and (x, y) position in the chess , value of | n – y | cannot same as | m – x |

When a node is bounded using bounding condition it will not generate any nodes in its sub-tree because nodes in its sub-tree will not give a feasible solution any more.

The portion of pruned state space tree after applying bounding condition is as follows:

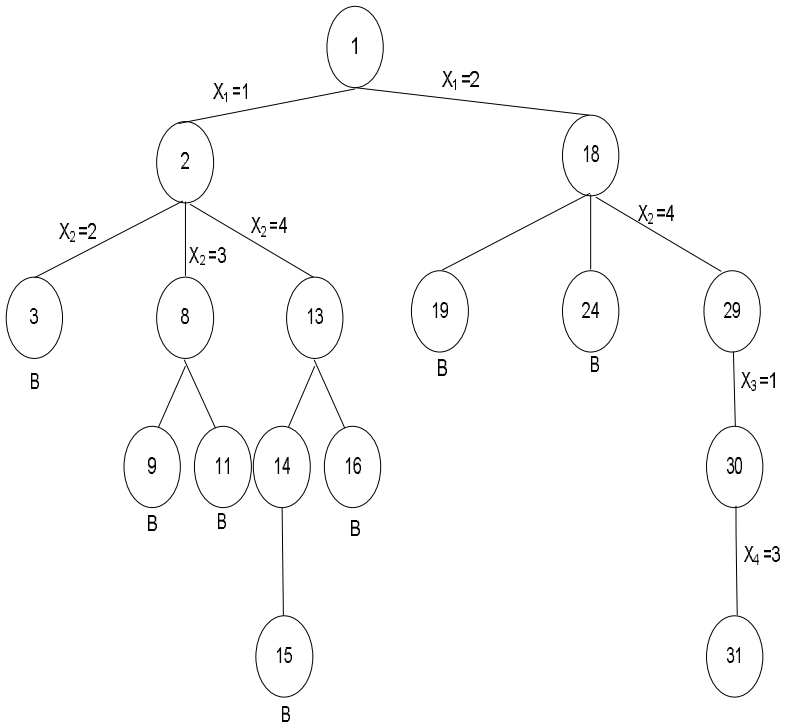


Fig. 5.2 Pruned state space tree

Here node 3 is bounded because-

At level 1, x_1=1 means first time 1st queen is placed in 1st row, 1st column i.e position is (1,1)

At level 2 , x_2=2 means second time 2nd queen is placed in 2nd row, 2nd column i.e position(2,2)

Thus they will place in diagonally. It will violet the implicit constraint or bounding condition. So this combination can not give a feasible solution any more. So, the children of node 3 will not generated further. Hence node 3 will bound.

Here is a path from root 1 to leaf 31 and this will generate one feasible solution set (2, 4, 1, 3) where x_1=2, x_2=4, x_3=1, x_4=3.

Position of the 4 queens are (1,2), (2,4), (3,1) and(4,3) respectively.

A recursive backtracking function for n-Queen problem:

/* placed search for a new queen*/

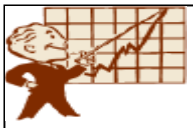
```
bool QPlace ( int k, int i )
  {
    for ( int m = 1; m < k; m++ )
      {
      if (( x [m] == i ) || (abs ( x[m] – i ) == abs ( m – k )))
      return (false);
      return (true);
      }
  }
```

/* Solution to n queen*/

```
void nQueen ( int k,  int i )
{
  for ( int i =1; i ≤ n; i++)
    if ( QPlace( k, i ))
     {
        x [k] = i;
       if ( k == n )
         {
           for ( int m = 1; m ≤ n; m++ )
           {cout << x [m] <<' '<<;
              cout<<endl;
           }
     }
        else
           nQueen ( k + 1, n );
      }
    }
```

Here QPlace (k, i) will return a boolean value true or false. The function return true if k^{th} queen can placed in i^{th} column and assigned it to x [k]. This value x [k] = i is distinct from x [1]…… x [k-1]. It also ensures that no two queen is placed in same diagonal.

Next nQueen (k, n) will solve the n-Queen problem recursively using backtracking method.



. **CHECK YOUR PROGRESS**

2. State True/False:
   i. In n- Queen Problem two Queens can attack each other.
   ii. If two Queens placed diagonally it will violet the implicit constraint.

5.7 GRAPH COLORING PROBLEM

Let G be a graph and m be a given integer. Is there any way to color the vertices of graph G using m color in such a way that no two adjacent vertices have same color? This is called as m-color ability decision problem. According to the graph coloring theory if d is the degree of a given graph, then it can be colored with d +1 color. The minimum number of color required in graph coloring problem to color vertices is called chromatic number. The m-color ability optimization problem is to determine the chromatic number of the graph G. This is called graph coloring problem.

For example graph in the following fig can be colored with minimum 3 colors 1, 2, 3. So, chromatic number is 3.

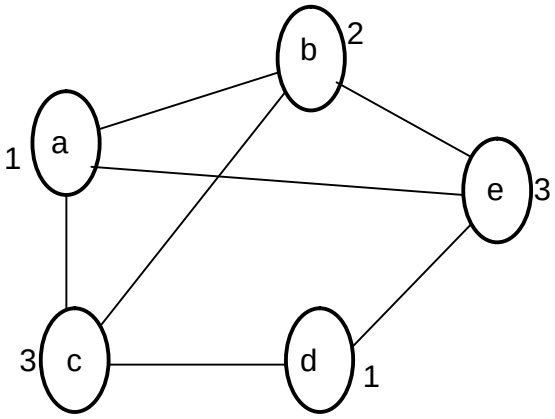


Fig. 5.3 A graph G

Solution using backtracking:

Suppose we represent graph G of n vertices by adjacency matrix G [1:n] [1:n], where G [i] [j] =1,if there is an edge between vertex i and j in G and G [i] [j] =0, otherwise.

The colors of the graph can be numbered from 1 to m. The solution set are represented by (x_1, x_2,…..,x_n), where x_i is the color of vertex i. The state space tree for this problem has degree m and height n + 1. Each node at level i has m children correspond to m color and represents i^{th} vertex of graph G. The left most node has assigned color 1 and rightmost vertex has assigned color m. Node at level n + 1 is leaf.

Example:

G is a graph which has 3 vertices (n = 3) and we have to solve the graph coloring problem for this graph G using 3 color (m = 3)

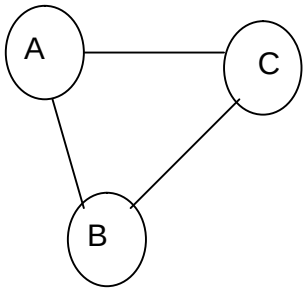


Fig. 5.4 A graph with 3-vertices

The general state space tree for this problem is as follows-

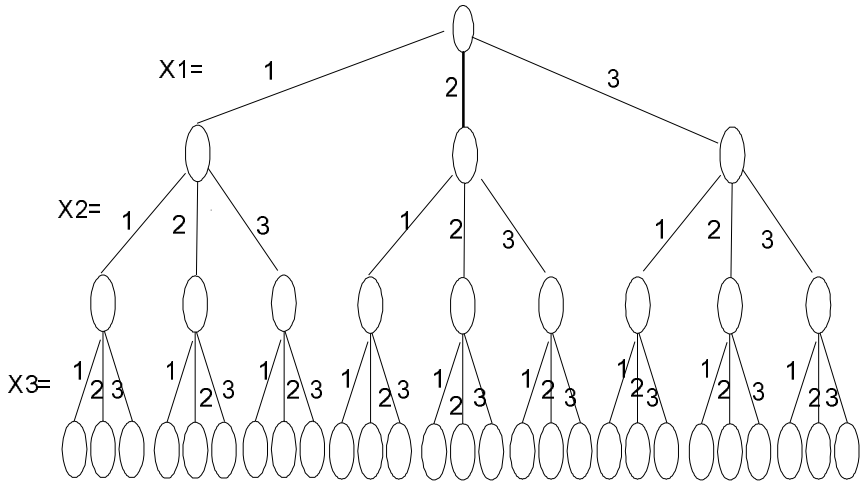


Fig. 5.5 A state space tree

There are possible three color to color vertex A. Hence x_1 has three values, x_1=1, x_1=2, x_1=3.

If we color the vertex A using color 1 then for 2nd vertex B there are three possible colors1, 2,3. Hence, x_2=1,x_2=2,x_2=3 and similarly for vertex C .

Now if we apply backtracking method to solve the problem then it will use a bound in function to kill a node in the state space tree. The bounding function for this problem is that no two adjacent vertexes have same color.

The pruned state space tree after applying bounding function is as follows-

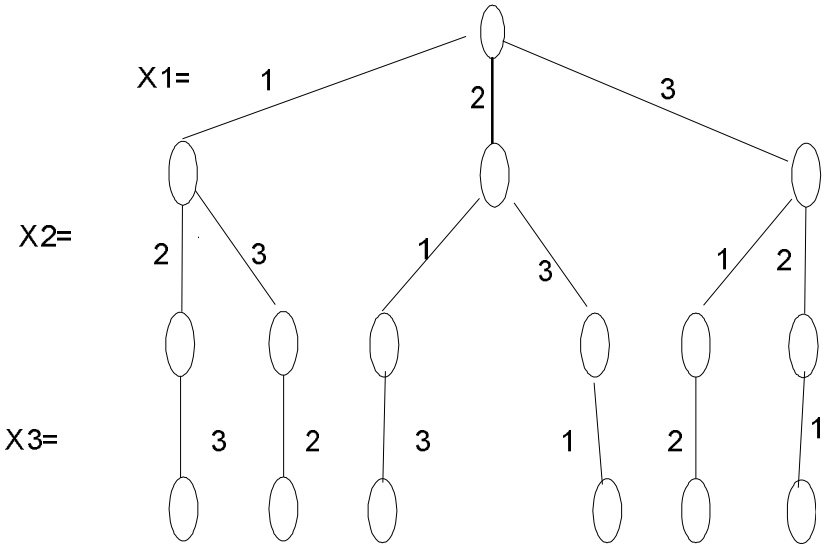


Fig. 5.6 A pruned state space tree

For the graph in fig vertex 1 is the adjacency of vertex 2.So, if we color vertex 1 using color 1 i.e x_1=1 then we cannot color vertex 2 with color 2 i.e x_2=1 because it will violet the bounding condition. Hence node 3 can not lead to a feasible solution and it will bounded. Other nodes are bounded similarly.

Some feasible solution sets are

i) (1,2,3), where x_1=1,x_2=2,x_3=3,

ii) (2,1,3), where x_1=2,x_2=1,x_3=3

iii) (3,2,1), where x_1=3,x_2=2,x_3=1

Here x_i represents i^{th} vertex color value.

***Recursive algorithm for graph coloring problem**:*

/* finding color value of all vertex*/ \

```
void mcoloring ( int k )
{     do
      {
         NextValue ( k );
            if ( ! x [ k ] )  break;
                if ( k == n )
                     {  for ( int i =1; i ≤ n; i++ )
                          cout  <<  x[ i ] ;
                          cout << endl;
                      }
                  else
                     mcoloring ( k + 1 );
 }
  while (1);
  }
```

/* Generating next color */

```
void NextValue ( int k )
{     do
      {
        x [ k ] = x [ k + 1] % ( m + 1 );
          if ( ! x [ k ] ) return;
            for ( int j = 1; j ≤ n; j++ )
                  {
                     if (G [ k ] [ j ] && (x [ k ] == x [ j ] ))
                       break;
                   }
              if ( j == n + 1 ) return;
          }
           while(1);
   }
```

Here k is the next vertex to color

n is the number of vertices

x [i] is the color of i^{th} vertex.

The function mcoloring first create the adjacency matrix G [i] [j] of the graph G and then initialize x [k] = 0, for all $1 \leq k \leq n$. First invoke the procedure by mcoloring (1). Function NextValue produces possible color for vertex k and assign it to x_k. Function mcoloring repeatedly picks a color value and assign it to x_k and then calls mcoloring recursively.

CHECK YOUR PROGRESS

3. What is the decision problem of graph coloring?
4. What is the chromatic number of a graph?

5.8 HAMILTONIAN CYCLE

A Hamiltonian cycle of a connected undirected graph with n vertices is a cyclic path along n edges, such that each vertex visits once in graph G and return to the starting vertex. It is named after William Hamilton.

Example :

Following is graph G with 8 vertices.

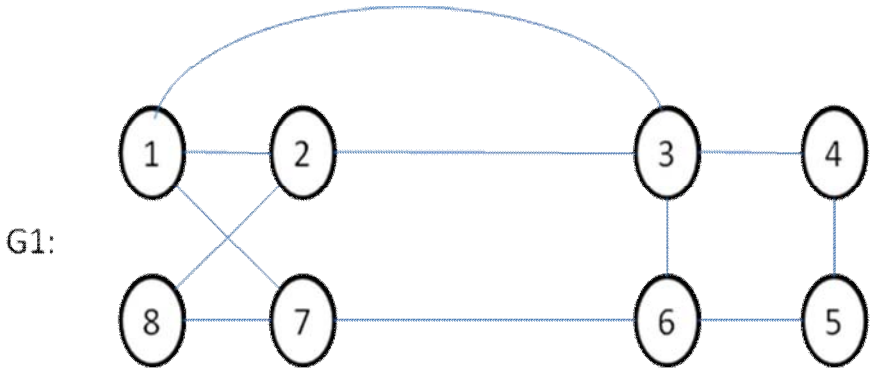


Fig. 5.7 A graph with 8 vertices

The Hamiltonian cycle of this graph is- 1, 2, 8, 7, 6, 5, 4, 3, 1

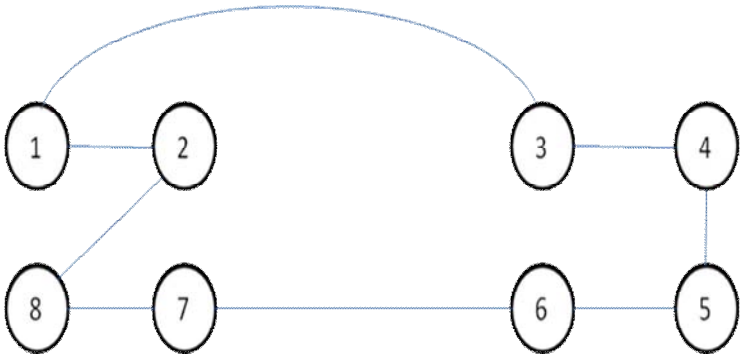


Fig. 5.8 Hamiltonian cycle

Backtracking method for Hamiltonian cycle:

Now, using backtracking method we can find out the Hamiltonian cycles in a graph which has n vertices. The solution set can represented as (x_1, x_2,……,x_n), where $1 \leq i \leq n$ and x_i represents the i^{th} visited vertex of the current considered cycle.

We have to determine value of x_i i.e possible vertex to select. For i =1, x_1 can be any vertex chooses from n vertex. To determine value of x_i we have already determine x_1, x_2,……..,x_{i-1}. Hence, the x_i can be choose as

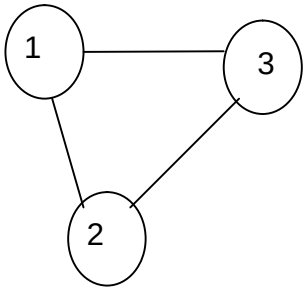
i) any vertex v which is not assigned to x_1, x_2…..and x_{i-1} from the n vertices.

ii) v is connected by an edge to x_{i-1}

The last vertex x_n must be connected to both x_{n-1} and x_1.

Example:

Consider the following graph and find out all the Hamiltonian cycle.



The state space tree for the graph is-

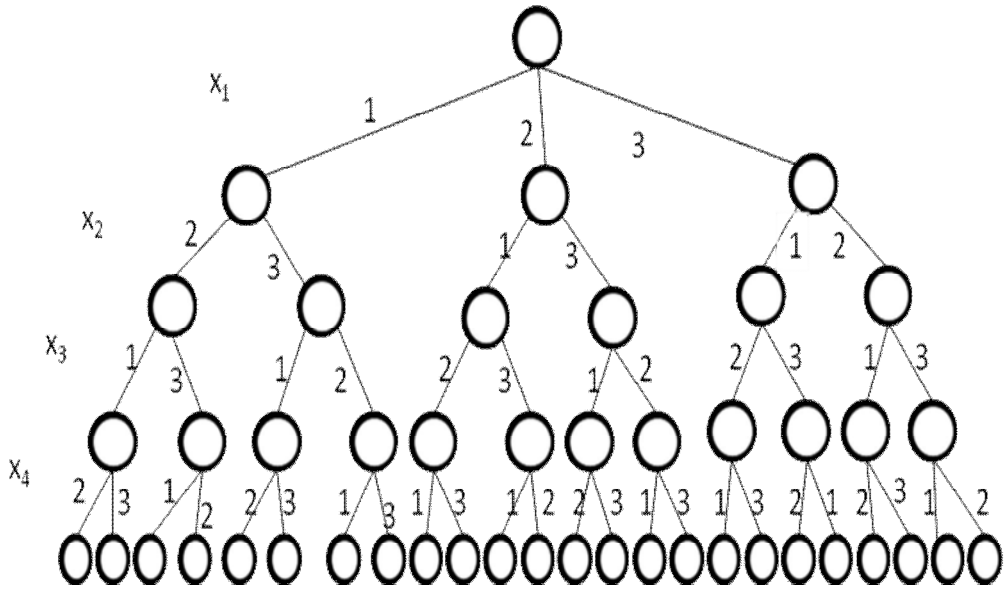


Fig. 5.9 State space tree

For using backtracking the bounding functions are-

i) The solution vector (x_1, x_2,……x_n) is defined such that value of x_i's are distinct, for all $1 \leq i \leq n$, because one vertex visit only once.

ii) There is an edge between (x_{i-1}, x_i)and (x_i, x_{i+1})

iii) There is an edge between x_n and x_1, (for Hamiltonian cycle).

Now, the pruned state space tree is as follows-

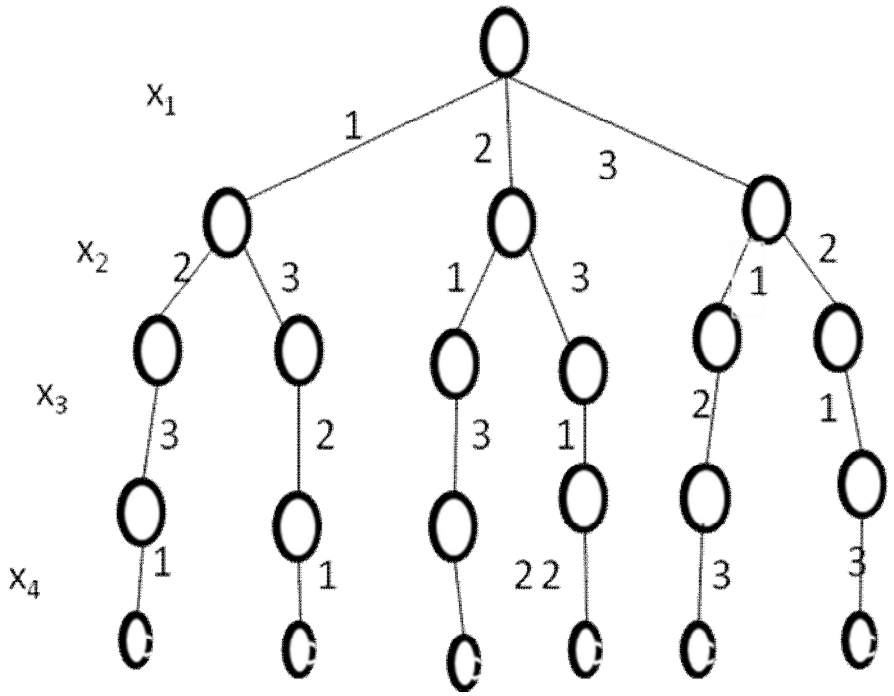


Fig. 5.10 Pruned state space tree

Recursive function for Hamiltonian cycle:

/* for finding Hamiltonian cycle*/

```
void Hamiltonian ( int k )
{
    do
     {
          NextValue ( k );
             if ( ! x [ k ] )  return;
```

```
            if ( k == n )
               {
                 for ( int i =1; i ≤ n; i++)
                     cout << x [ i ] <<" "<<"\n";
                }
             else
                 Hamiltonian ( k + 1 )
        }
    while(1);
}
```

/* generating next vertex*/

```
void NextValue ( int k )
    {
       do
          {
             x [ k ] = ( x [ k ] + 1 ) % ( n + 1 );          //Next vertex
              if ( ! x [ k ] ) return;
              if ( G [ x [ k – 1 ] [ x [ k ] ])
                {
                   for ( int j = 1; j ≤ k - 1; j++)
                      if ( x [ j ] == x [ k ] )  break ;
                if ( j == k )
                if ( ( k < n ) || (( k == n ) && G [ x [ n ] ] [ x [ 1] ] ))
                return;
             }
         }
   while(1);
 }
```

This program first initializes the adjacency matrix G [1:n] [1:n] and x [1] = 1 and x [2:n] = 0.

Hamiltonian function is invoked by Hamiltonian (2).