Week 9:

Design and implement to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using Backtracking principle.

Problem:

- ✓ Given graph G with n vertices and E edges

  Find all possible Hamiltonian cycles in a graph G.
- ✓ The Hamiltonian cycle is the cycle that traverses all the vertices of the given graph G exactly once and then ends at the starting vertex.
- ✓ The input can be the directed or undirected graph.
- ✓ Graph G is represented as boolean valued adjacency matrix

  Where G[i,j]= 1 if there exist an edge between i,j

  = 0 otherwise

Backtracking Solution:

- ✓ Problem involves checking if the Hamiltonian cycle is present in a graph **G** or not.
- ✓ Following bounding functions are to be considered while generating possible Hamiltonian cycles
  1. The k^{th} visiting vertex in the path must be adjacent to the $(k-1)^{th}$ visiting vertex in any path (G[x[k-1],x[k]]==1).
  2. The starting vertex and the n^{th} vertex should be adjacent (G[x[n],x[1])==1.
  3. Each vertex should be visited only once.

Algorithm:

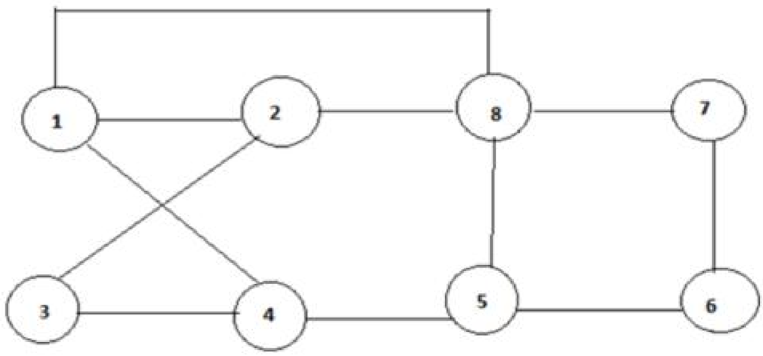
```
Algorithm NextValue(k)
// x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
// no vertex has as yet been assigned to x[k]. After execution,
// x[k] is assigned to the next highest numbered vertex which
// does not already appear in x[1 : k − 1] and is connected by
// an edge to 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);
}
```

```
Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation of
// backtracking to find all the Hamiltonian cycles
// of a graph. The graph is stored as an adjacency
// matrix G[1 : n, 1 : n]. All cycles begin at node 1.
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal 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);
}
```

- ✓ This algorithm uses the recursive formulated of backtracking to find all the Hamiltonian cycles of a graph.

- ✓ The graph is stored as an adjacency matrix g[1: n, 1: n].
- ✓ In the next value k, x [1: k-1] is a path with k-1 distinct vertices.
- ✓ if x[k] == 0 then no vertex has to be yet been assign to x[k].
- ✓ After execution x[k] is assigned to the next highest numbered vertex which does not already appear in the path x [1: k-1] and is connected by an edge to x [k – 1] otherwise x[k] == 0.
- ✓ If k == n, (here n is the number of vertices) then in addition x[n] is connected to x [1].

Example:



Hamiltonian cycle 1 8 7 6 5 4 3 2 1

Sample Implementation:

```
#include<stdio.h>
void displaycycl();
void nextvalue(int k);
int g[10][10],n,x[10],c=0;

//hamilton cycle algorithm
void hamiltonian(int k)
{
  while(1)
   {
     nextvalue(k);
     if(x[k]==0)
       {
         return;
       }
     if(k==n)
       {
            c=c+1;
       displaycycl();
       }
     else
       {
       hamiltonian(k+1);
       }
    }
}

//nextvalue algorithm to determine kth visiting vertex
void nextvalue(int k)
{
     int j;
   while(1)
   {
```

```
        x[k] = (x[k]+1)%(n+1);
          if(x[k]==0)
             {
                return;
              }
             if(g[x[k-1]][x[k]] != 0)
               {
                  for(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]] != 0 )))
                       {
                           return;
                       }
                  }
               }
       }
}

//function to display solutions
void displaycycl()
{
int i;

      for(i=1;i<=n;i++)
            printf("%d ",x[i]);
      printf("%d ", x[1]);
      printf("\n");
}

//main function
int main()
{
int i,j;
printf("\n enter the no of vertices...");
scanf("%d",&n);
printf("\n enter the graph info...");
for(i=1;i<=n;i++)
{
      for(j=1;j<=n;j++)
      {
       scanf("%d",&g[i][j]);
      }
}
for(i=1;i<=n;i++)
      x[i]=0;
x[1]=1;
printf("\n Hamiltonian cycles possible are....\n");
hamiltonian(2);
```

```
printf("total %d solutions",c);
if(c==0)
{
    printf("\n solutions not possible");
}

return 0;
}
```

Output 1:

```
enter the no of vertices...4

enter the graph info…

0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

Hamiltonian cycles possible are....

1 2 3 4 1
1 2 4 3 1
1 3 2 4 1
1 3 4 2 1
1 4 2 3 1
1 4 3 2 1

total 6 solutions
```

Output 2:

```
enter the no of vertices...5

enter the graph info…

0 1 1 0 0
1 0 1 0 0
1 1 0 1 1
0 0 1 0 1
0 0 1 1 0

Hamiltonian cycles possible are....
total 0 solutions
solutions not possible
```