DSA Virtual Lab

QUESTION 1 : N Queen problem

CODE :

```
#define N 4
#include<stdio.h>
#include<stdbool.h>
void printSolution(int board[N][N])
{
   int i, j;
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)
      printf(" %d ", board[i][j]);
   printf("\n");
   }
}
bool isSafe(int board[N][N], int row, int col) {
   int i, j;
   for (i = 0; i < col; i++)
      if (board[row][i])
      return false;
   for (i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j])
      return false;
   for (i=row, j=col; j>=0 && i<N; i++, j--)
      if (board[i][j])
      return false;
   return true;
}
bool solveNQUtil(int board[N][N], int col) {
   int i;
   if (col >= N)
      return true;
```

```
        for (i = 0; i < N; i++) {
        if ( isSafe(board, i, col) ) {
            board[i][col] = 1;
            if ( solveNQUtil(board, col + 1) )
        return true;
    board[i][col] = 0; // BACKTRACK
    }
    }
    return false;
}

bool solveNQ() {
    int board[N][N] = { {0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}};
    if( solveNQUtil(board, 0) == false ) {
        printf("Solution does not exist");
    return false;
    }
    printSolution(board);
    return true;
}
int main() {
    solveNQ();
    return 0;
}
```

OUTPUT :

4 queen problem

```
0  0  1  0
1  0  0  0
0  0  0  1
0  1  0  0

Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

8 queen problem

```
1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0

Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

QUESTION 2 : Sum of subsets

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))
static int total_nodes;
void printSubset(int A[], int size)
{
   int i;
   for(i = 0; i < size; i++)
   {
      printf("%*d", 5, A[i]);
   }
   printf("\n");
}
void subset_sum(int s[], int t[],int s_size, int t_size,int sum, int ite,int const target_sum)
{
   total_nodes++;
```

```
    if( target_sum == sum ) {
        printSubset(t, t_size);
        subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
        return;
    }
    else
    {
        int i;
        for(i = ite; i < s_size; i++ ) {
            t[t_size] = s[i];
            subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
        }
    }
}
void generateSubsets(int s[], int size, int target_sum) {
    int *tuplet_vector = (int *)malloc(size * sizeof(int));
    subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
    free(tuplet_vector);
}
int main() {
    int weights[] = {5, 10, 12, 13, 15, 18};
    int size = ARRAYSIZE(weights);
    generateSubsets(weights, size, 30);
    printf("Nodes generated %d\n", total_nodes);
    return 0;
}
```

OUTPUT: weights = {5, 10, 12, 13, 15, 18}, required sum = 30

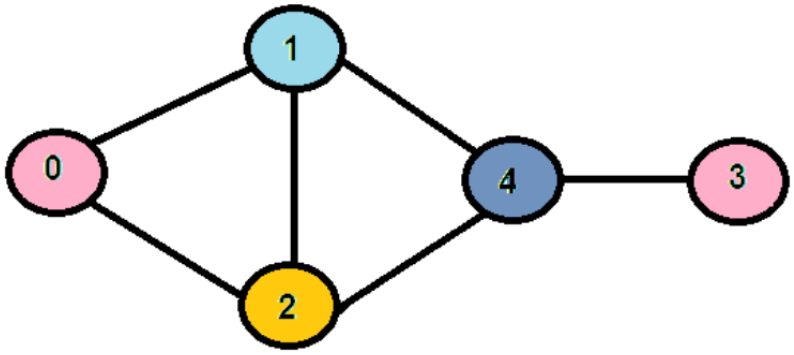
```
    5   10   15
    5   12   13
   12   18
Nodes generated 64

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

QUESTION 3 : Graph Coloring

CODE:

```
#include<stdio.h>
int G[50][50],x[50];  //G:adjacency matrix,x:colors
void next_color(int k){
  int i,j;
  x[k]=1;  //coloring vertex with color1
  for(i=0;i<k;i++){ //checking all k-1 vertices-backtracking
    if(G[i][k]!=0 && x[k]==x[i])  //if connected and has same color
      x[k]=x[i]+1;  //assign higher color than x[i]   }}
int main(){
 int n,e,i,j,k,l;
 printf("Enter no. of vertices : ");
 scanf("%d",&n);
 printf("Enter no. of edges : ");
 scanf("%d",&e);
 for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    G[i][j]=0;
 printf("Enter indexes where value is 1-->\n");
 for(i=0;i<e;i++){
  scanf("%d %d",&k,&l);
  G[k][l]=1;
  G[l][k]=1;
 }
 for(i=0;i<n;i++)
  next_color(i);  //coloring each vertex
 printf("Colors of vertices -->\n");
 for(i=0;i<n;i++)  //displaying color of each vertex
  printf("Vertex[%d] : %d\n",i+1,x[i]);
 return 0;
}
```



OUTPUT

```
Enter no. of vertices : 5
Enter no. of edges : 6
Enter indexes where value is 1-->
0 1
0 2
1 2
1 4
2 4
4 3
Colors of vertices -->
Vertex[1] : 1
Vertex[2] : 2
Vertex[3] : 3
Vertex[4] : 1
Vertex[5] : 2
```

QUESTION 4 : Hamiltonian Cycle

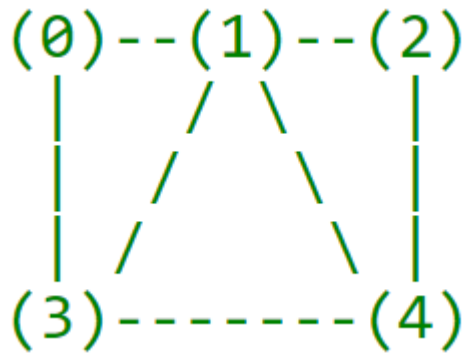
CODE :

```
#include<stdio.h>
#define V 5
#include<stdbool.h>
#include <malloc.h>
void printSolution(int path[]);
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
   int i;
   if (graph [ path[pos-1] ][ v ] == 0)
      return false;
   for (i = 0; i < pos; i++)
```

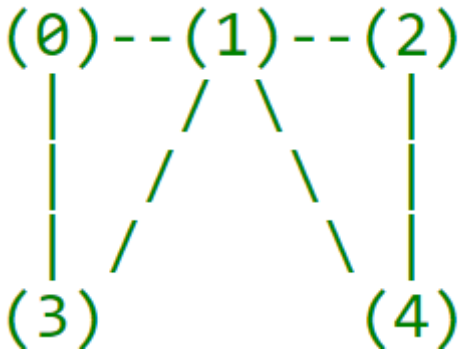
```
        if (path[i] == v)
            return false;
    return true;
}
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
    int v;
    if (pos == V)
    {
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
          return true;
        else
         return false;
    }
    for (v = 1; v < V; v++)
    {
        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;
            if (hamCycleUtil (graph, path, pos+1) == true)
               return true;
            path[pos] = -1;
        }}
    return false;
}bool hamCycle(bool graph[V][V])
{
    int i;
    int *path =  malloc(sizeof(int[V]));
    for (i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == false )
```

```
    {
        printf("\nSolution does not exist");
        return false;
    }
    printSolution(path);
    return true;
}
void printSolution(int path[])
{
    int i;
    printf ("Solution Exists:"
            " Following is one Hamiltonian Cycle \n");
    for (i = 0; i < V; i++)
        printf(" %d ", path[i]);    printf(" %d ", path[0]);
    printf("\n");
}
int main(){
   bool graph1[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 1},
                        {0, 1, 1, 1, 0},
                       };
    hamCycle(graph1);
    bool graph2[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 0},
                        {0, 1, 1, 0, 0},
                       };
    hamCycle(graph2);
    return 0;}
```

Example 1 Hamiltonian Graph



Example 2 Non-Hamiltonian Graph



Output

```
Solution Exists: Following is one Hamiltonian Cycle
 0  1  2  4  3  0

Solution does not exist
```