INDEX

9]	Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.	20-02-2024	24-25	
10]	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.	27-02-2024	26-28	
11]	Find Single source shortest path of a given undirected graph using Dijikstra’s algorithm.	27-02-2024	29-30	
12]	Find a subset of a given set S = {sl, s2,....., sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution	05-03-2024	31-31	
13]	Implement N Queen's problem using Back Tracking.	12-03-2024	32-33	

1. **Design and implement linear search and binary search algorithms. Analyze the efficiencies of algorithms. Repeat the experiment for different values of n.**

```
 #include<stdio.h>
int main()
{
   int choice, n, key, temp = 0;
   printf("1-Linear Search 2-Binary Search\n");
   printf("Enter your choice: ");

   scanf("%d", &choice);

   switch (choice)
   {
      case 1:
      {
         int a[20],i;
         printf("Enter total elements:\n");
         scanf("%d", &n);

         printf("Enter array elements:\n");
         for (i = 0; i < n; i++)
            scanf("%d", &a[i]);

         printf("Enter the element to be searched:\n");
         scanf("%d", &key);

         for (i = 0; i < n; i++)
         {
            temp++;
            if (a[i] == key)
            {
               printf("Element found at index %d\n", i);
               break;
            }
         }

         if (temp > n)
            printf("Element not found\n");

         printf("Number of loops = %d\n", temp);
         break;
      }
```

```
        case 2:
        {
            int array[100],i;
            printf("Enter the number of elements:\n");
            scanf("%d", &n);

            printf("Enter %d integers in sorted order:\n", n);
            for (i = 0; i < n; i++)
                scanf("%d", &array[i]);

            printf("Enter the value to be found:\n");
            scanf("%d", &key);

            int low = 0, high = n - 1, mid;

            while (low <= high)
            {
                temp++;
                mid = low + (high - low) / 2;
                if (array[mid] == key)
                {
                    printf("%d found at index %d\n", key, mid);
                    break;
                }
                else if (array[mid] < key)
                    low = mid + 1;
                else
                    high = mid - 1;
            }

            if (low > high)
                printf("Not found! %d isn't present in the list\n", key);

            printf("Number of loops = %d\n", temp);
            break;
        }
        default:
            printf("Invalid choice\n");
    }
    return 0;
}
```

OUTPUT:

1-Linear Search 2-Binary Search
Enter your choice: 1
Enter total elements:
5
Enter array elements:
32
14
25
56
96
Enter the element to be searched:
56
Element found at index 3
Number of loops = 4

1-Linear Search 2-Binary Search
Enter your choice: 2
Enter the number of elements:
5
Enter 5 integers in sorted order:
12
23
45
56
78
Enter the value to be found:
78
78 found at index 4
Number of loops = 3

2. **Sort a given set of elements using the bubble sort and selection sort method and determine the time required to sort the elements. Repeat the experiment for different values of n.**

a)

```
#include<stdio.h>
#include<stdlib.h>
void sort(int a[ ],int n)
{ int i,j,temp;
 for(i=0;i<n-1;i++)
 {
	for(j=0;j<n-i-1;j++)
	{
		if(a[j]>a[j+1])
		{
			temp=a[j];
			a[j]=a[j+1];
			a[j+1]=temp;
		}
	}
 }
}
int main()
{
 int n,i;
 printf("enter the numbers to be sort\n");
 scanf("%d",&n);
 int a[n];
 printf("enter the array elements\n");
 for(i=0;i<n;i++)
 {
	scanf("%d",&a[i]);
 }

 sort(a,n);

 printf("After sorting\n");
 for(i=0;i<n;i++)
 {
	printf("%d\n",a[i]);
 }
```

```
 return 0;
}
```

OUTPUT:

enter the numbers to be sort
5
enter the array elements
45
62
215
453
120
After sorting
45
62
120
215
453

b)

```
 #include<stdio.h>
#include<stdlib.h>
void sort(int a[],int n)
{int small,j,i;
for(i=0;i<n;i++)
 {
	small=i;
	for(j=i+1;j<n;j++)
	{
		if(a[small]>a[j])
	{
		small=j;
	}
	if(small!=i)
	{
		int temp=a[i];
		a[i]=a[small];
		a[small]=temp;
	}
 }
}
}
int main()
```

```
{
 int n,i;
 printf("enter your number\n");
 scanf("%d",&n);
 int a[n];
 printf("enter the array elements\n");
 for(i=0;i<n;i++)
 {
 	scanf("%d",&a[i]);
 }

 sort(a,n);

printf("sorted elements are\n");
for(i=0;i<n;i++)
{
 printf("%d\n",a[i]);
}

return 0;
}
```

OUTPUT:

```
enter your number
5
enter the array elements
41
23
69
84
96
sorted elements are
41
23
69
84
96
```

3. Sort a given set of elements using the quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n.

```
#include<stdio.h>
 int partition (int a[], int start, int end)
{
    int key = a[end];
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {

        if (a[j] < key)
        {
            i++;
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

void quick(int a[], int start, int end)
    {
          if(start<end)
          {

        int p = partition(a, start, end);
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }

}


int main()
{  int num,i;
```

```
 printf("enter the number of elements\n");
 scanf("%d",&num);
 printf("enter the element\n");
   int a[num];
        for (i = 0; i < num; i++)
        {
                     scanf("%d",&a[i]);
            }

   printf("Before sorting array elements are - \n");
 for (i = 0; i < num; i++)
 {
      printf("%d ", a[i]);
 }



   quick(a, 0, num - 1);

   printf("\nAfter sorting array elements are - \n");
  for (i = 0; i < num; i++)
  {
      printf("%d ", a[i]);
  }

   return 0;
}
```

OUTPUT:

```
enter the number of elements
4
enter the element
56
62
57
90
Before sorting array elements are -
56 62 57 90
After sorting array elements are -
56 57 62 90
```

4. Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n.

```
#include<stdio.h>
 void merge(int a[],int l,int m,int h)
{
 int i,j,k=l;
 int p=m-l+1;
 int q=h-m;
 int b[p],c[q];
 for(i=0;i<p;i++)
 b[i]=a[l+i];
 for(j=0;j<q;j++)
 c[j]=a[m+j+1];
 i=0,j=0;
 while(i<p && j<q)
 {
		if(b[i]<=c[j])
		{
			a[k]=b[i];
			i++;
		}
		else
		{
			a[k]=c[j];
			j++;
		}
		k++;
 }
 while(i<p)
		{
			a[k]=b[i];
		   i++;
		   k++;
		}
 while(j<q)
		{
			a[k]=c[j];
		   j++;
```

```
            k++;
        }

}
void mergesort(int a[],int low,int high)
{
 if(low<high)
 {
        int m=low+(high-low)/2;
        mergesort(a,low,m);
        mergesort(a,m+1,high);
        merge(a,low,m,high);
 }
}
int main()
{
 int i,n;
 printf("enter size of array:\n");
 scanf("%d",&n);
 int a[n];
 printf("enter array elements\n");
 for(i=0;i<n;i++)
 {
        scanf("%d",&a[i]);
 }
 printf("original array:\n");
 for(i=0;i<n;i++)
 {
        printf("%d\t",a[i]);
 }

 mergesort(a,0,n-1);

 printf("\nsorted array:\n");
 for(i=0;i<n;i++)
 {
        printf("%d\t ",a[i]);

 }
 return 0;
}
```

OUTPUT:

enter size of array:
5
enter array elements
4
9
50
10
30
original array:
4 9 50 10 30
sorted array:
4 9 10 30 50

5. **Implement the following algorithms to**
    a. **Print all the nodes reachable from a given starting node in digraph using BFS method.**
    b. **Check whether a given graph is connected or not using DFS method.**

a)
```
#include <stdio.h>
void bfs(int);
int visited[10], que[10], f = -1, r = -1, a[10][10], n, i, j;

int main() {
    int v;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }    }
    for (i = 0; i < n; i++) {
        que[i] = 0;
        visited[i] = 0;
    }
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);
    printf("Nodes reachable from vertex %d are:\n", v);
    for (i = 0; i < n; i++) {
        if (visited[i]) {
            printf("%d\n", i);
        }
    }
    return 0;
}
void bfs(int v) {
    int i;
    for (i = 0; i < n; i++) {
        if (a[v][i] == 1 && !visited[i]) {
            que[++r] = i;
            visited[i] = 1;
        }
    }
```

```
        if (f <= r) {
            bfs(que[++f]);
        }
    }
```

OUTPUT:

```
Enter number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting vertex: 0
Nodes reachable from vertex 0 are:
1
2
```

b)

```
#include<stdio.h>
#define max 100
int adj[max][max];
int visited[max];
int n;
void dfs(int i)
{
    printf("%d",i);
    visited[i]=1;
    for(int j=1;j<=n;j++)
    {
            if(!visited[j]&&adj[i][j])
                    dfs(j);
    }
}
int main()
{
    int e;
    printf("Enter number of vertices\n");
    scanf("%d",&n);
    printf("Enter number of edges\n");
    scanf("%d",&e);
    for(int i=1;i<=n;i++)
    {
            visited[i]=0;
            for(int j=1;j<=n;j++)
                    adj[i][j]=0;
```

```
    }
    printf("Enter edges (source,destination):\n");
    for( int i=1;i<=e;i++)
    {
        printf("Edge %d: ",i);
        int src,des;
        scanf("%d%d",&src,&des);
        adj[src][des]=1;
        adj[des][src]=1;
    }
    printf("Adjacency matrix:\n");
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            printf("%d  ",adj[i][j]);
        printf("\n");
    }
    dfs(1);
    return 0;
}
```

OUTPUT:

```
Enter number of vertices
4
Enter number of edges
5
Enter edges (source,destination):
Edge 1: 1
2
Edge 2: 2
4
Edge 3: 4
3
Edge 4: 3
1
Edge 5: 1
4
Adjacency matrix:
0  1  1  1
1  0  0  1
1  0  0  1
1  1  1  0
1243
```

6. **Implement Horspool string matching algorithm to search for a pattern in the text.**

```
#include<stdio.h>
#include<string.h>
#define MAX 256
void shiftTable(char pattern[],int table[]){
   int m=strlen(pattern);
   for(int i=0;i<MAX;i++)
      table[i]=m;
   for(int i=0;i<m-1;i++)
      table[(unsigned char)pattern[i]]=m-i-1;}
int horspoolSearch(char text[],char pattern[]){
   int n=strlen(text);
   int m=strlen(pattern);
    int table[MAX];
    shiftTable(pattern,table);
    int i=m-1;
   while(i<n){
      int j=0;
      while(j<m && text[i-j]==pattern[m-1-j])
         j++;
      if(j==m)
         return i-m+1;
      i+=table[(unsigned char)text[i]]; }
   return -1;
}
int main(){
   char text[MAX];
   char pattern[50];
   printf("Enter the string (text):");
   gets(text);
   printf("Enter the pattern:");
   gets(pattern);
   int result=horspoolSearch(text,pattern);
   if (result!=-1)
      printf("Pattern found at position %d\n",result);
   else
      printf("Pattern not found\n");
   return 0;
}
```

OUTPUT:

Enter the string (text):casgchhj
Enter the pattern:sg
Pattern found at position 2

7. **Implement the following algorithms to**
   a. Compute the transitive closure of a given directed graph using Warshall's algorithm.
   b. Compute the all pairs shortest path matrix using Floyd's algorithm.

a)

```
#include<stdio.h>
int main()
{
    int n,i,j,k;
    printf("Enter number of vertices: ");
    scanf("%d",&n);
    int a[n][n];
    printf("Enter adjacency matrix\n");
    for(i=0;i<n;i++)
            for(j=0;j<n;j++)
            {
                    printf("a[%d][%d]: ",i,j);
                    scanf("%d",&a[i][j]);
            }
    for(k=0;k<n;k++)
            for(j=0;j<n;j++)
                    for(i=0;i<n;i++)
                            a[i][j]=a[i][j] || (a[i][k] && a[k][j]);

    for(i=0;i<n;i++)
    {
            for(j=0;j<n;j++)
                    printf("%d\t",a[i][j]);
            printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 3
Enter adjacency matrix
a[0][0]: 0
a[0][1]: 1
a[0][2]: 0
a[1][0]: 0
a[1][1]: 0
a[1][2]: 1
a[2][0]: 0
a[2][1]: 0
a[2][2]: 0
0	1	1
0	0	1
0	0	0
```

b)

```
#include<stdio.h>
int main()
{
	int n,i,j,k;
	printf("enter number of vertices: ");
	scanf("%d",&n);
	int a[n][n];
	printf("Enter adjacency(weighted) matrix\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
		{
			printf("a[%d][%d]: ",i,j);
			scanf("%d",&a[i][j]);
		}
	for(k=0;k<n;k++)
		for(j=0;j<n;j++)
			for(i=0;i<n;i++)
				a[i][j]=(a[i][j]<(a[i][k] + a[k][j]))? a[i][j]:(a[i][k] + a[k][j]);
	printf("shortest path:\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			printf("%d\t",a[i][j]);
```

```
        printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
enter number of vertices: 4
Enter adjacency(weighted) matrix
a[0][0]: 0
a[0][1]: 999
a[0][2]: 3
a[0][3]: 999
a[1][0]: 2
a[1][1]: 0
a[1][2]: 999
a[1][3]: 999
a[2][0]: 999
a[2][1]: 7
a[2][2]: 0
a[2][3]: 1
a[3][0]: 6
a[3][1]: 999
a[3][2]: 999
a[3][3]: 0
shortest path:
0   10      3       4
2   0       5       6
7   7       0       1
6   16      9       0
```

8. Implement Knapsack problem using Dynamic Programming approach.

```
#include<stdio.h>
int max(int a, int b) {
  if(a>b)
  {
    return a;
  }
  else
  {
    return b;
  }
}
int knapsack(int W, int wt[], int val[], int n)
 {
  int i, w;
  int knap[n+1][W+1];
  for (i = 0; i <= n;i++)
   {
    for (w = 0; w <= W; w++)
     {
      if (i==0 || w==0)
        knap[i][w] = 0;
      else if (wt[i-1] <= w)
        knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
      else
        knap[i][w] = knap[i-1][w];
    }
  }
  return knap[n][W];
}
int main()
{
  int val[10];
  int wt[10];
  int W, n;
  printf("enter size of knapsack\n");
  scanf("%d",&n);
  printf("enter values\n");
  for(int i=0;i<n;i++)
  {
      printf("val[%d]:",i);
      scanf("%d",&val[i]);
```

```
  }
  printf("Enter weights\n");
  for(int i=0;i<n;i++)
  {
      printf("wt[%d]:",i);
      scanf("%d",&wt[i]);
  }
  printf("enter maximum weight:\n");
  scanf("%d",&W);
  printf("The solution is : %d", knapsack(W, wt, val, n));
  return 0;
}
```

OUTPUT:

```
enter size of knapsack
4
enter values
val[0]:42
val[1]:12
val[2]:40
val[3]:25
Enter weights
wt[0]:7
wt[1]:3
wt[2]:4
wt[3]:5
enter maximum weight:
10
The solution is : 65
```

9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#include<stdio.h>
#include<stdbool.h>
#include<string.h>
#define INF 9999999
int main() {
   int V,no_edge,x,y;
   printf("Enter the number of vertices: ");
   scanf("%d", &V);

   int G[V][V];

   printf("Enter the adjacency matrix for the graph (%d x %d):\n", V, V);
   for (int i = 0; i < V; i++)
    {
      for (int j = 0; j < V; j++)
      {
         scanf("%d", &G[i][j]);
      }
   }

   int selected[V];
   memset(selected, false, sizeof(selected));
   no_edge = 0;
   selected[0] = true;

   printf("Edge : Weight\n");

   while (no_edge < V - 1) {
      int min = INF;
      x = 0;
      y = 0;

      for (int i = 0; i < V; i++) {
         if (selected[i]) {
            for (int j = 0; j < V; j++) {
               if (!selected[j] && G[i][j]) {
                  if (min > G[i][j]) {
                     min = G[i][j];
                     x = i;
                     y = j;
                  }
               }
```

```
            }
          }
        }
        printf("%d - %d : %d\n", x, y, G[x][y]);
        selected[y] = true;
        no_edge++;
    }

    return 0;
}
```

OUTPUT:

Enter the number of vertices: 4
Enter the adjacency matrix for the graph (4 x 4):
0
7
2
3
7
0
1
999
2
1
0
5
3
999
5
0
Edge : Weight
0 - 2 : 2
2 - 1 : 1
0 - 3 : 3

10. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
#include <stdio.h>
#include <stdlib.h>

struct Edge {
    int src, dest, weight;
};

struct Subset {
    int parent;
    int rank;
};

int find(struct Subset subsets[], int i);
void Union(struct Subset subsets[], int x, int y);
int compare(const void* a, const void* b);
void KruskalMST(struct Edge* edges, int V, int E);

int main() {
    int V, E;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    struct Edge* edges = (struct Edge*)malloc(E * sizeof(struct Edge));

    printf("Enter source, destination, and weight for each edge:\n");
    for (int i = 0; i < E; ++i)
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);

    KruskalMST(edges, V, E);

    free(edges);
    return 0;
}

int find(struct Subset subsets[], int i) {
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
```

```
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int compare(const void* a, const void* b) {
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight - b1->weight;
}

void KruskalMST(struct Edge* edges, int V, int E) {
    struct Edge result[V];
    int e = 0;
    int i = 0;

    qsort(edges, E, sizeof(edges[0]), compare);
    struct Subset* subsets = (struct Subset*)malloc(V * sizeof(struct Subset));

    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    while (e < V - 1 && i < E) {

        struct Edge next_edge = edges[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }

    printf("Edges of Minimum Cost Spanning Tree:\n");
    for (i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);

    free(subsets);
}
```

OUTPUT:

Enter the number of vertices and edges: 4 5
Enter source, destination, and weight for each edge:
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges of Minimum Cost Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

11.Find Single source shortest path of a given undirected graph using Dijikstra's algorithm.

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{

int min = INT_MAX, min_index,v;
for (v = 0; v < V; v++)
if (sptSet[v] == false && dist[v] <= min)
min = dist[v], min_index = v;
return min_index;
}
void printSolution(int dist[])
{
 int i;
printf("Vertex \t\t Distance from Source\n");
for (i = 0; i < V; i++)
printf("%d \t\t\t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{
int dist[V],i,count,v;
bool sptSet[V];
for (i = 0; i < V; i++)
dist[i] = INT_MAX, sptSet[i] = false;
dist[src] = 0;
for (count = 0; count < V - 1; count++) {
int u = minDistance(dist, sptSet);
sptSet[u] = true;
for (v = 0; v < V; v++)
if (!sptSet[v] && graph[u][v]
&& dist[u] != INT_MAX
&& dist[u] + graph[u][v] < dist[v])
dist[v] = dist[u] + graph[u][v];

}
printSolution(dist);
}
int main()
{
int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
 { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
 { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
 { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
 { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
 { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
```

```
{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },
{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },
{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
dijkstra(graph, 0);
return 0;
```

OUTPUT:

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

12.**Find a subset of a given set S = {sl, s2,....., sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution.**

```
#include <stdio.h>
#include <stdbool.h>
bool subsetSum(int S[], int n, int d) {
   for (int i = 0; i <= n; i++)
      dp[i][0] = true;
   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= d; j++) {
         if (S[i - 1] <= j)
            dp[i][j] = dp[i - 1][j] || dp[i - 1][j - S[i - 1]];
         else
            dp[i][j] = dp[i - 1][j])  }
   }
   if (dp[n][d]) {
      printf("Subset with sum %d: ", d);
      int i = n, j = d;
      while (i > 0 && j > 0) {
         if (dp[i][j] != dp[i - 1][j]) {
            printf("%d ", S[i - 1]);
            j -= S[i - 1]; }
      i--; }
      printf("\n");
      return true;
   } else {
      printf("No subset found with sum %d\n", d);
      return false;
   }
}
int main() {
   int S[] = {1, 2, 5, 6, 8};
   int d = 9;
   int n = sizeof(S) / sizeof(S[0]);
   subsetSum(S, n, d);
   return 0;
}
```

OUTPUT :

Subset with sum 9: 1 2 6

13.**Implement N Queen's problem using Back Tracking.**

```
#include <stdio.h>
#include <stdbool.h>
#define N 8
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
        printf("\n");    }}
bool isSafe(int board[N][N], int row, int col) {
    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) {
    if (col >= N)
        return true;
    for (int 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;        }}}
bool solveNQ() {
    int board[N][N] = { {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {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 :

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