VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

Analysis and Design of Algorithms

Submitted by

ANOSHOR B. PAUL
(1BM21CS024)

In partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
In
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2023 to July-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Anoshor B. Paul (1BM21CS024),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Prof. Sunayana S
Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

1. Experiments

1.1 Experiment - 1

1.1.1 Question:

Write program to do the following:

(a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

(b) Check whether a given graph is connected or not using DFS method.

1.1.2 Code:

(a) BFS method:

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

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
	for(i=1;i<=n;i++)
	if(a[v][i] && !visited[i])
	q[++r]=i;
	if(f<=r)
	{
		visited[q[f]]=1;
		bfs(q[f++]);
	}
}

void main()
{
	int v;

	printf("\n Enter the number of vertices:");
	scanf("%d",&n);

	for(i=1;i<=n;i++)
	{
		q[i]=0;
		visited[i]=0;
	}

	printf("\n Enter graph data in matrix form:\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&a[i][j]);
```

```
	printf("\n Enter the starting vertex:");
	scanf("%d",&v);
	bfs(v);

	printf("\n The node which are reachable are:\n");
	for(i=1;i<=n;i++)
		if(visited[i])
		printf("%d\t",i);

	getch();
}
```

(b) <u>DFS method</u>:

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

int a[20][20],reach[20],n;

void dfs(int v)
{
	int i;
	reach[v]=1;
	for(i=1;i<=n;i++)
	if(a[v][i]&&!reach[i])
	{
		printf("\n%d->%d",v,i);
		dfs(i);
	}
}

int main()
{
	int i,j,count=0;
	printf("\nEnter no of vertices : ");
	scanf("%d",&n);

	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			reach[i]=0;
			a[i][j]=0;
		}
	printf("\nEnter adjacency matrix : \n");

	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
```

```
                scanf("%d",&a[i][j]);
                dfs(1);

        for(i=1;i<=n;i++)
        if(reach[i])
        count++;

        if(count==n)
                printf("\nGraph is connected.");
        else
                printf("\nGraph is disconnected.");

        getch();
        return(0);
}
```

1.1.3 Output:

(a) BFS method:

```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

 Enter the starting vertex:1

 The node which are reachable are:
2       3       4
```

DFS method

```
Enter no of vertices : 4

Enter adjacency matrix :
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

1->2
2->4
4->3
Graph is connected.
```

```
Enter no of vertices : 4

Enter adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0

1->2
2->3
Graph is disconnected.
```

1.2 Experiment - 2

1.2.1 Question:

Write program to obtain the Topological ordering of vertices in a given digraph.

1.2.2 Code:

```
#include <stdio.h>

int main()
{
   int i,j,k,n,a[10][10],indeg[10],visited[10],count=0;

   printf("Enter the no of vertices:\n");
   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++)
   {
      indeg[i]=0;
      visited[i]=0;
   }

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         indeg[i]=indeg[i]+a[j][i];

   printf("\nThe topological order is:");

   while(count<n)
      {
      for(k=0;k<n;k++)
      {
         if((indeg[k]==0) && (visited[k]==0))
         {
            printf("%d ",(k+1));
            visited [k]=1;
```

```
        }

        for(i=0;i<n;i++)
        {
           if(a[i][k]==1)
              indeg[k]--;
        }
      }
      count++;
   }
   return 0;
}
```

1.2.3 Output:

```
Enter the no of vertices:
7
Enter the adjacency matrix:
Enter row 1
0 0 0 0 1 0 0
Enter row 2
1 0 1 0 0 0 0
Enter row 3
0 0 0 1 0 0 0
Enter row 4
0 0 0 0 1 1 0
Enter row 5
0 0 0 0 0 1 0
Enter row 6
0 0 0 0 0 0 1
Enter row 7
0 0 0 0 0 0 0

The topological order is:2 1 3 4 5 6 7
```

1.3 Experiment - 3

1.3.1 Question:

Implement Johnson Trotter algorithm to generate permutations.

1.3.2 Code:

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

#define MAX_N 10

void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void printPermutation(int permutation[], int direction[], int n)
{
   for (int i = 0; i < n; i++)
   {
      printf("%d", permutation[i]);
   }
   printf("\n");
}

void generatePermutations(int n)
{
   int permutation[MAX_N];
   int direction[MAX_N];
   bool mobile[MAX_N];

   for (int i = 0; i < n; i++)
   {
      permutation[i] = i + 1;
      direction[i] = -1;
      mobile[i] = true;
   }

   printPermutation(permutation, direction, n);
```

```
int mobileElement, mobileIndex, temp;

while (true)
{
   mobileElement = -1;
   mobileIndex = -1;

   for (int i = 0; i < n; i++)
   {
      if (direction[i] == -1 && i > 0 && permutation[i] > permutation[i - 1] && mobile[i])
      {
         if (mobileElement == -1 || permutation[i] > mobileElement)
         {
            mobileElement = permutation[i];
            mobileIndex = i;
         }
      }

      if (direction[i] == 1 && i < n - 1 && permutation[i] > permutation[i + 1] && mobile[i])
      {
         if (mobileElement == -1 || permutation[i] > mobileElement)
         {
            mobileElement = permutation[i];
            mobileIndex = i;
         }
      }
   }

   if (mobileIndex == -1)
   {
      break;
   }

   if (direction[mobileIndex] == -1)
   {
      swap(&permutation[mobileIndex], &permutation[mobileIndex - 1]);
      swap(&direction[mobileIndex], &direction[mobileIndex - 1]);
   }
   else
   {
      swap(&permutation[mobileIndex], &permutation[mobileIndex + 1]);
```

```
            swap(&direction[mobileIndex], &direction[mobileIndex + 1]);
        }

        for (int i = 0; i < n; i++)
        {
            if (permutation[i] > mobileElement)
            {
                direction[i] *= -1;
            }
        }
        printPermutation(permutation, direction, n);
    }
}

int main()
{
    int n;

    printf("Enter the value of n: ");
    scanf("%d", &n);

    if (n < 1 || n > MAX_N)
    {
        printf("Invalid input!\n");
        return 0;
    }

    generatePermutations(n);
    return 0;
}
```

1.3.3 Output:

```
Enter the value of n: 2
12
21
```

```
Enter the value of n: 3
123
132
312
321
231
213
```

```
Enter the value of n: 4
1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134
```

1.4 Experiment - 4

1.4.1 Question:

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

1.4.2 Code:

```
#include <iostream>
#include <vector>

using namespace std;

void merge(int low, int mid, int high, vector<int>& arr) {
    int i=low,j=mid+1;
    vector<int> a;
    while(i<=mid && j<=high) {
        if(arr[i]<arr[j]) {
            a.push_back(arr[i]);
            i++;
        }
        else {
            a.push_back(arr[j]);
            j++;
        }
    }

    while(i<=mid) {
        a.push_back(arr[i]);
        i++;
    }

    while(j<=high) {
        a.push_back(arr[j]);
        j++;
    }
    int k=0;
    for(int i=0;i<(high-low+1);i++) {
        arr[low+i]=a[i];
    }
}
void mergesort(int low, int high, vector<int>& arr) {
    int n = arr.size();

    // if(n==1) return;

    if(low<high) {
```

```
        int mid=low+(high-low)/2;

        mergesort(low,mid,arr);
        mergesort(mid+1,high,arr);
        merge(low, mid, high, arr);
    }
}

int main() {
    vector<int> arr = {2,3,1,5};
    mergesort(0,arr.size()-1,arr);

    for(int i=0;i<arr.size();i++) {
        cout<<arr[i]<<" ";
    }

    return 0;
}
```

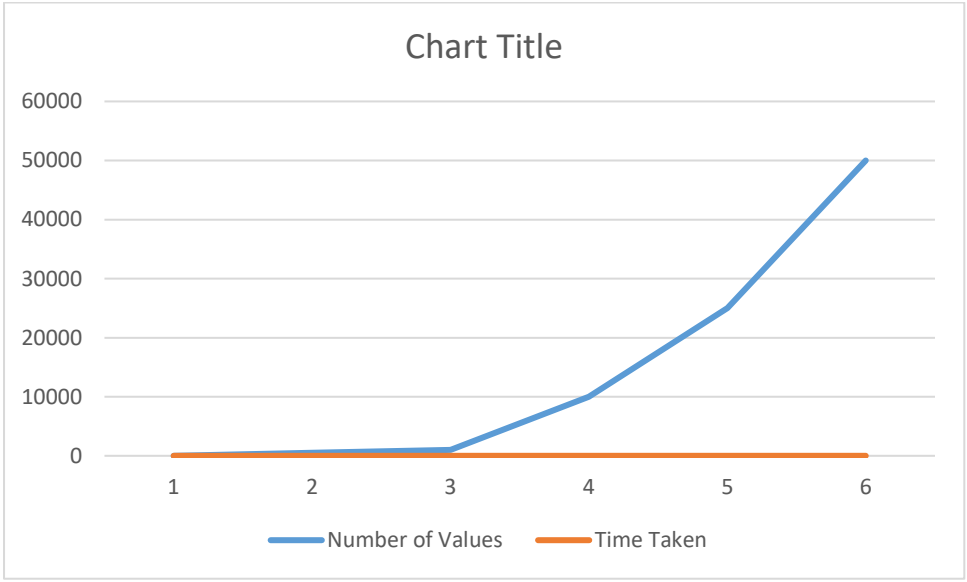
1.4.3 Output:

```
Enter the number of elements: 50

Enter array elements: (By Random)
Starting of the program, start_t = 1166
End of the program, end_t = 1387
Total time taken by CPU: 0.000221

Sorted array is:
273 304 768 1028 1368 1650 1652 2035 2230 4247 4313 4402 4470 460
3 4990 5026 5106 5133 5542 5862 5919 5953 6070 6185 6470 6688 669
6 6715 7300 7355 7515 7681 7836 7857 7887 7935 7971 8462 8750 907
9 9166 9214 9353 9384 9415 9777 9799 9820 9920 9949
```

1.4.4 Example Graph (Not Done in class)



1.5 Experiment - 5

1.5.1 Question:

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

1.5.2 Code:

```
#include <iostream>
#include <vector>

using namespace std;

int partition(int low, int high, vector<int>& arr) {
   int i=low,j=high;
   int pivot=arr[low];
   while(i<j) {
      while((arr[i]<=pivot) && (i<=(high+1))) {
         i++;
      }
      while((arr[j]>pivot)  &&  (j>=(low-1))) {
         j--;
      }
      if(i<j) {
         swap(arr[i],arr[j]);
      }
   }

   swap(arr[low], arr[j]);
   return j;
}

void quicksort(int low, int high, vector<int>& arr) {
   if(low<high) {
```

```
        int p = partition(low, high, arr);
        quicksort(low,p-1,arr);
        quicksort(p+1,high,arr);
    }
}


int main() {

    vector<int> arr = {2,1,5,3};

    quicksort(0,arr.size()-1,arr);

    for(auto x: arr) {
        cout<<x<<" ";
    }
    // cout<<"e";
    return 0;
}
```

1.5.3 Output:

```
enter number of elements 5
enter the elements: 3 5 4 1 7
1 3 4 5 7
```

1.6 Experiment - 6

1.6.1 Question:

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

1.6.2 Code:

```
#include <bits/stdc++.h>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;


void heapify(vector<int>& arr, int N, int i)
{

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int l = 2 * i + 1;

    // right = 2*i + 2
    int r = 2 * i + 2;

    // If left child is larger than root
    if (l < N && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest
    // so far
    if (r < N && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected
        // sub-tree
        heapify(arr, N, largest);
    }
}
vector<int> heapSort(vector<int>& arr, int N) {
```

```
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    for (int i = N - 1; i > 0; i--) {


        swap(arr[0], arr[i]);


        heapify(arr, i, 0);
    }

    return arr;

}

int main() {

  vector<int> arr = {3,1,4,2,8};

  heapSort(arr, arr.size());

  for(auto x : arr) { cout<<x<<" "; }

  return 0;
}
```

1.6.3 Output:

```
Enter the number of elements: 5
Enter 5 elements:
3 6 1 2 8
Sorted array: 1 2 3 6 8
```

1.7 Experiment - 7

1.7.1 Question:

Implement 0/1 Knapsack problem using dynamic programming.

1.7.2 Code:

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;


int knapsack(int w, vector<int> p, vector<int> weights) {
    int n = p.size();
    vector<vector<int>> v(n+1,vector<int>(w+1));

    // for(int i=1;i<v.size();i++) {
    //     for(int j=1;j<v[0].size();j++) {
    //         if(weights[i-1]>j) {
    //             v[i][j] = v[i-1][j];
    //         }
    //         else {
    //             v[i][j] = max(v[i-1][j], v[i-1][j-weights[i-1]]+p[i-1]);
    //         }
    //     }
    // }

    for(int i=0;i<v.size();i++) {
        for(int j=0;j<v[0].size();j++) {
            if(i == 0 || j == 0) {
                v[i][j] = 0;
            }
            else if(weights[i-1]>j) {
                v[i][j] = v[i-1][j];
            }
            else {
                v[i][j] = max(v[i-1][j], v[i-1][j-weights[i-1]]+p[i-1]);
            }
        }
    }

    return v[n][w];
}
int main() {
    // vector<int> weights = {2,1,3,2};
```

```
    // vector<int> profit = {12,15,25,10};

    vector<int> weights = {2,3,1,2,4};
    vector<int> profit = {10,12,25,13,11};

    cout<<knapsack(5,profit,weights);

    return 0;
}
```

1.7.3 Output:

```
Enter the number of items: 4
Enter the values of the items: 3 4 5 6
Enter the weights of the items: 2 3 4 5
Enter the capacity of the knapsack: 5
The maximum profit is 7
The objects selected for the optimal solution are: 1 2
```

1.8 Experiment - 8

1.8.1 Question:

Implement All Pair Shortest paths problem using Floyd's algorithm.

1.8.2 Code:

```
#include <stdio.h>

#define INFINITY 999

int nV;
void printMatrix(int matrix[][nV]);

void floyd(int graph[][nV])
{
   int matrix[nV][nV], i, j, k;

   for (i = 0; i < nV; i++)
      for (j = 0; j < nV; j++)
         matrix[i][j] = graph[i][j];

   for (k = 0; k < nV; k++)
   {
      for (i = 0; i < nV; i++)
      {
         for (j = 0; j < nV; j++)
         {
            if (matrix[i][k] + matrix[k][j] < matrix[i][j])
            matrix[i][j] = matrix[i][k] + matrix[k][j];
         }
      }
   }
   printMatrix(matrix);
}

void printMatrix(int matrix[][nV])
{
   printf("\nAll Pairs Shortest Path is :\n");
   for (int i = 0; i < nV; i++)
   {
      for (int j = 0; j < nV; j++)
      {
         if (matrix[i][j] == INFINITY)
            printf("%4s", "INF");
         else
            printf("%4d", matrix[i][j]);
```

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

int main()
{
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &nV);

    int graph[nV][nV];
    printf("Enter the weight of edges in the graph:\n");
    for (int i = 0; i < nV; i++)
    {
        for (int j = 0; j < nV; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    floyd(graph);
}
```

1.8.3 Output:

```
Enter the number of vertices in the graph: 4
Enter the weight of edges in the graph:
0 4 9 999
999 0 7 999
999 999 0 6
2 3 999 0

All Pairs Shortest Path is :
   0   4   9  15
  15   0   7  13
   8   9   0   6
   2   3  10   0
```

1.9 Experiment - 9

1.9.1 Question:

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

1.9.2 Code:

(a) Prim's Algorithm:

```
#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()
{
   printf("Prim's algorithm:\n");

        printf("Enter the number of nodes:");
        scanf("%d",&n);

        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
      {
         scanf("%d",&cost[i][j]);
         if(cost[i][j]==0)
            cost[i][j]=999;
      }
        visited[1]=1;
        printf("\n");

   printf("\nThe edges of Minimum Cost Spanning Tree are:");
        while(ne < n)
        {
                for(i=1,min=999;i<=n;i++)
          for(j=1;j<=n;j++)
             if(cost[i][j]< min)
             if(visited[i]!=0)
             {
                min=cost[i][j];
                a=u=i;
                b=v=j;
             }

                 if(visited[u]==0 || visited[v]==0)
```

```
            {
                printf("\nEdge %d:(%d,%d) Weight:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\nMinimun Cost=%d",mincost);
}
```

(b) Kruskal's Algorithm:

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[10][10], parent[9];

int find(int);
int uni(int, int);

void main()
{
 printf("Kruskal's algorithm:\n");

 printf("Enter the no. of vertices:\n");
 scanf("%d", &n);

 printf("\nEnter the cost adjacency matrix:\n");
 for (i = 1; i <= n; i++)
 {
  for (j = 1; j <= n; j++)
  {
   scanf("%d", &cost[i][j]);
   if (cost[i][j] == 0)
    cost[i][j] = 999;
  }
 }

 printf("\nThe edges of Minimum Cost Spanning Tree are\n");
 while (ne < n)
 {
  for (i = 1, min = 999; i <= n; i++)
  {
   for (j = 1; j <= n; j++)
   {
    if (cost[i][j] < min)
```

```
        {
          min = cost[i][j];
          a = u = i;
          b = v = j;
        }
      }
    }

    u = find(u);
    v = find(v);

    if (uni(u, v))
    {
      printf("Edge %d:(%d,%d) Weight:%d\n", ne++, a, b, min);
      mincost += min;
    }

    cost[a][b] = cost[b][a] = 999;
  }

  printf("\nMinimum cost = %d\n", mincost);
}

int find(int i)
{
  while (parent[i])
    i = parent[i];
  return i;
}

int uni(int i, int j)
{
  if (i != j)
  {
    parent[j] = i;
    return 1;
  }

  return 0;
}
```

1.9.3 Output:

(a) Prim's Algorithm:

```
Prim's algorithm:
Enter the number of nodes:5

Enter the adjacency matrix:
0 1 7 10 5
1 0 3 999 999
7 3 0 4 999
10 999 4 0 2
5 999 999 2 0


The edges of Minimum Cost Spanning Tree are:
Edge 1:(1,2) Weight:1
Edge 2:(2,3) Weight:3
Edge 3:(3,4) Weight:4
Edge 4:(4,5) Weight:2

Minimun Cost=10
```

(b) Kruskal's Algorithm:

```
Kruskal's algorithm:
Enter the no. of vertices:
5

Enter the cost adjacency matrix:
0 1 7 10 5
1 0 3 999 999
7 3 0 4 999
10 999 4 0 2
5 999 999 2 0

The edges of Minimum Cost Spanning Tree are
Edge 1:(1,2) Weight:1
Edge 2:(4,5) Weight:2
Edge 3:(2,3) Weight:3
Edge 4:(3,4) Weight:4

Minimum cost = 10
```

1.10 Experiment - 10

1.10.1 Question:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

1.10.2 Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits.h>

const int MAX_NODES = 100;
using namespace std;


vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)
   {
      // Code here

      priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;

      vector<int> dist(V, INT_MAX);

      dist[S] = 0;

      pq.push({0,S});

      while(!pq.empty()) {
         int dis = pq.top().first;
         int node = pq.top().second;

         pq.pop();

         for(auto it: adj[node]) {
            int edgeWeight = it[1];

            int adjNode = it[0];

            if(dis+edgeWeight < dist[adjNode]) {
               dist[adjNode] = dis + edgeWeight;

               pq.push({dist[adjNode], adjNode});
            }
         }
      }
```

```
    return dist;
}


int main() {

    vector<vector<int>> adj[MAX_NODES];
    int V;
    cout << "Enter the number of vertices: ";
    cin >> V;

    while(true) {
        int u,v,w;
        cout <<endl<< "Enter edge (u v w): ";
        cin >> u >> v >> w;

        if(u==-1) {
            break;
        }
        adj[u].push_back({v,w});
        adj[v].push_back({u,w});
    }

    vector<int> res;

    res = dijkstra(V, adj, 0);

    for(auto x: res) {
        cout<<x<<" ";
    }

    return 0;
}
```

1.10.3 Output:

```
Enter the number of vertices: 4

Enter the cost adjacency matrix:
0 3 999 7
3 0 2 999
999 2 0 1
7 999 1 0

Enter the source vertex: 1
Shortest Path:
1->1=0
1->2=3
1->3=5
1->4=6
```

1.11 Experiment - 11

1.11.1 Question:

Implement “N-Queens Problem” using Backtracking.

1.11.2 Code:

```
#include <iostream>
#include <cmath>

using namespace std;

bool canPlace(int x[], int k, int i) {
    for (int j = 1; j <= k - 1; j++) {
        if (x[j] == i || abs(x[j] - i) == abs(j - k)) {
            return false;
        }
    }
    return true;
}

void nQueens(int n) {
    int x[n + 1];
    int k = 1;
    x[k] = 0;

    while (k > 0) {
        x[k]++;

        while (x[k] <= n && !canPlace(x, k, x[k])) {
            x[k]++;
        }
```

```
        if (x[k] <= n) {
            if (k == n) {
                cout << "Solution: ";
                for (int i = 1; i <= n; i++) {
                    cout << "(" << i << ", " << x[i] << ") ";
                }
                cout << endl;
            } else {
                k++;
                x[k] = 0;
            }
        } else {
            k--;
        }
    }
}

int main() {
    int n;
    cout << "Enter the number of queens: ";
    cin >> n;

    nQueens(n);

    return 0;
}
```

1.11.3 Output:

```
N Queens Problem Using Backtracking:

Enter number of Queens:4


Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```