LAB MANUAL

Lab Name	**:**	Analysis of Algorithms Lab
Lab Code	**:**	5IT4-23
Branch	**:**	Information Technology
Year/Semester	**:**	3rd Year/V

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Department of Information Technology
Jaipur Engineering College and Research Centre, Jaipur
(Affiliated to RTU, Kota)

INDEX

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTER

Department of Information Technology

Branch: Information Technology **Semester: 5th**
Course Name: Analysis of Algorithm Lab **Code: 5IT4-23**
External Marks: 20 **Practical hrs: 2 hr/week**
Internal Marks: 30 **Total Marks: 50**

VISION & MISSION OF INSTITUTE

VISION

To become a renowned centre of outcome based learning, and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities.

MISSION

- Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
- Identify, based on informed perception of Indian, regional and global needs, areas of focus and provide platform to gain knowledge and solutions.
- Offer opportunities for interaction between academia and industry.
- Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

Department of Information Technology Engineering

VISION

To be recognized as Centre for providing outcome based education and prepare students to take challenges as per present technological scenario.

MISSION

M1: Practice OBE for professional accomplishment of graduate attributes.
M2: Provide platform to gain knowledge and solutions as per social needs and requirement.
M3: Provide platform to enhance knowledge for inter-disciplinary challenges and motivation towards achieving excellence.

PEO

1. To enrich students with fundamental knowledge, effective computing, problem solving and communication skills enable them to have successful career in Information Technology.
2. To enable students in acquiring Information Technology's latest tools, technologies and management principles to give them an ability to solve multidisciplinary engineering problems.
3. To impart students with ethical values and commitment towards sustainable development in collaborative mode.
4. To imbibe students with research oriented and innovative approaches which help them to identify, analyze, formulate and solve real life problems and motivates them for lifelong learning.
5. To empower students with leadership quality and team building skills that prepare them for employment, entrepreneurship and to become competent professionals to serve societies and global needs.

PROGRAM OUTCOMES

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems in IT.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences in IT.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations using IT.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions using IT.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations in IT.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice using IT.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development in IT.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice using IT.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in IT.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage IT projects and in multidisciplinary environments.

12. **Life –long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes needed in IT.

PSO OF THE DEPARTMENT:

PSO1: Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.

PSO2: Ability to apply the knowledge of cloud computing, artificial intelligence, machine learning and deep learning under realistic constraints.

LIST OF EXPERIMENTS as per RTU syllabus

5IT4-23: Analysis of Algorithm Lab

Class: 5th Sem. B. Tech. 3rd year	**Evaluation**
Branch: IT **Credits: 1** **Schedule per week: 2 Hrs (Practical)**	**Examination Time = Three (2) Hours** **Maximum Marks = 50** **[Internal Assessment/Sectional (30) & End-term Exam(20)]**

S. No.	**Contents**
1.	Write a Program to Sort a given set of elements using the Quick sort method and determine the time required to sort the elements.
2.	Write a program to implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements.
3.	a. Write a program to obtain the Topological ordering of vertices in a given digraph. b. Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.
4.	Write a program to implement 0/1 Knapsack problem using Dynamic Programming.
5.	Write a program to find shortest paths to other vertices from a given vertex in a weighted connected graph using Dijkstra's algorithm.
6.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.
7.	a. Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method. b. Write a program to check whether a given graph is connected or not using DFS method.
8.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.
9.	Write a program to implement All-Pairs Shortest Paths Problem using Floyd's algorithm.
10.	Write a program to implement N Queen's problem using Back Tracking algorithm.
11.	Write a Program to implement Travelling Salesperson problem using Dynamic programming. **(Content Beyond)**

Reference Books:

1. Introduction of Algorithms: Thomas H. Corman
2. Data Structures with C: Seymour Lipschutz
3. Introduction to Analysis of Algorithms: Anany Levitin

Course Outcome:

Upon successful completion of this Lab the student will be able to:

CO1: To Design and development of Divide and Conquer strategy algorithms.
CO2: To Design and development of Greedy Programming algorithm.

CO3: To Design and development of Dynamic Programming algorithm.

CO4: To Design and development of Backtracking and Branch and Bound algorithm.

CO-PO Mapping:

SEM	SUBJECT WITH CODE	L/P/T	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
V	ANALYSIS OF ALGORITHM(5IT4-23)	0/0/2	To Design and development of Divide and Conquer strategy algorithms.	3	3	3	2	1		1	1	3	2		1
			To Design and development of Greedy Programming algorithm.	3	3	3	1			1		2	2		1
			To Design and development of Dynamic Programming algorithm.	3	2	3	1		1	1	2	2	1	1	2
			To Design and development of Backtracking and Branch and Bound algorithm.	3	3	3	1		1	1	2	2	1	1	2

CO-PSO Mapping:

5IT4-23: ANALYSIS OF ALGORITHM(5IT4-23)		PSO-1	PSO-2	**Average mapping with course**	**Average mapping with course**
	CO-1	1	1	1	1
	CO-2	1	1		
	CO-3	1	1		
	CO-4	1	1		

Introduction about Laboratory & Applications

In theoretical analysis of algorithms, it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input. The term "analysis of algorithms" was coined by Donald Knuth.

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Most algorithms are designed to work with inputs of arbitrary length. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity, or volume of memory, known as space complexity.

An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. It is an efficient method that can be expressed within finite amount of time and space. An algorithm is the best way to represent the solution of a particular problem in a very simple and efficient way. If we have an algorithm for a specific problem, then we can implement it in any programming language, meaning that the algorithm is independent from any programming languages.

Algorithm Design: The important aspects of algorithm design include creating an efficient algorithm to solve a problem in an efficient way using minimum time and space. To solve a problem, different approaches can be followed. Some of them can be efficient with respect to time consumption, whereas other approaches may be memory efficient. However, one has to keep in mind that both time consumption and memory usage cannot be optimized simultaneously.

INSTRUCTIONS OF LAB

DO's

1. Please switch off the Mobile phone before enter into the Lab.
2. Check whether all peripheral are available at your desktop before proceeding for program.
3. Intimate the lab technician whenever you face any problem related to hardware and software.
4. Arrange all the peripheral and seats before leaving the lab.
5. Properly shutdown the system before leaving the lab.
6. Keep the bag outside.
7. Maintain the decorum of the lab.

DON'TS

1. No one is allowed to use pen drives without permission of lab technician in the lab.
2. Don't mishandle the system.
3. Don't bring any external material in the lab.
4. Don't make noise in the lab.
5. Don't litter in the lab.
6. Don't delete or make any modification in system files.
7. Don't carry any lab equipments outside the lab.

INSTRUCTIONS FOR STUDENT

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next program.
- Students are supposed to bring the practical file and the lab copy.
- Assignment given in previous labs should be written in the practical file.
- Print out of diagram should be pasted in the lab file.
- Any student not following these instructions will be denied entry in the lab.

WHILE WORKING IN THE LAB

- Adhere to experimental schedule as instructed by the lab in-charge.
- Get the previously executed program signed by the instructor.
- Get the output of the current program checked by the instructor in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and do not play games.
- If anyone caught red handed carrying any equipment of the lab, then he will have to face serious consequences.

Lab Objective

This laboratory course is intended to make the students experiment on the basic techniques of analysis of algorithms construction and tools that can used to perform Greedy method and Branch and Bound method of a high-level programming language into an executable code. Students will design and implement language processors in C by using tools to automate parts of the implementation process. This will provide deeper insights into the more advanced semantics aspects of programming languages, code generation, machine independent optimizations, dynamic memory allocation, and object orientation.

Content of Lab Experiments

S. No.	Contents
1.	Write a Program to Sort a given set of elements using the Quick sort method and determine the time required to sort the elements.
2.	Write a program to implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements.
3.	a. Write a program to obtain the Topological ordering of vertices in a given digraph. b. Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.
4.	Write a program to implement 0/1 Knapsack problem using Dynamic Programming.
5.	Write a program to find shortest paths to other vertices from a given vertex in a weighted connected graph using Dijkstra's algorithm.
6.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.
7.	a. Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method. b. Write a program to check whether a given graph is connected or not using DFS method.
8.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.
9.	Write a program to implement All-Pairs Shortest Paths Problem using Floyd's algorithm.
10.	Write a program to implement N Queen's problem using Back Tracking algorithm.
11.	Write a Program to implement Travelling Salesperson problem using Dynamic programming. **(Content Beyond)**

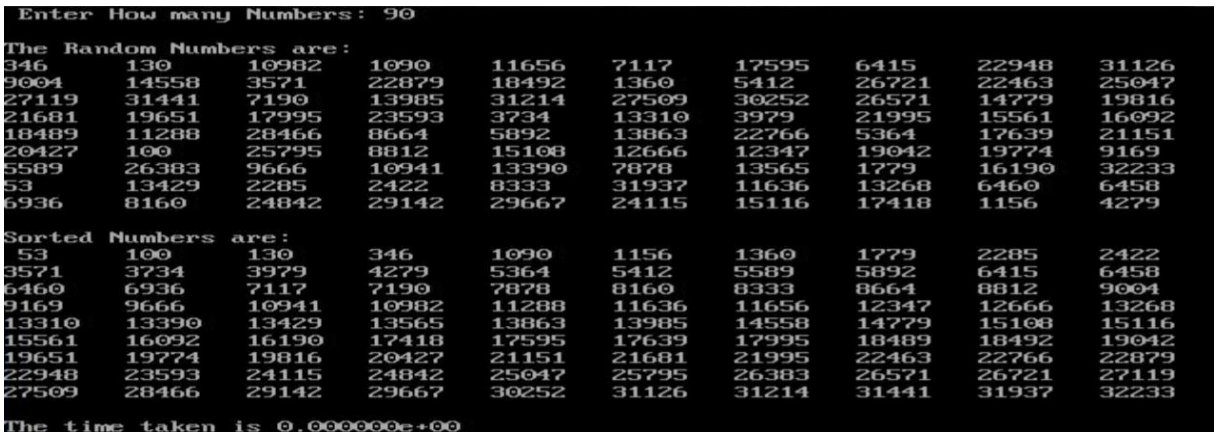
Experiment No.1

Aim: - 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, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
1. include <stdio.h>
2. include <conio.h>
3. include <time.h>


void Exch(int *p, int *q)
{
                                int temp = *p;
                                *p = *q;
                                *q = temp;
 }

void QuickSort(int a[], int low, int high)
{
                                int i, j, key, k;
                                if(low>=high)
                                        return;
                                key=low; i=low+1; j=high;
                                while(i<=j)
                                {
                                        while ( a[i] <= a[key] ) i=i+1;
                                        while ( a[j] > a[key] )  j=j-1;
                                        if(i<j) Exch(&a[i], &a[j]);
                                }
                                Exch(&a[j], &a[key]);
                                QuickSort(a, low, j-1);
                                QuickSort(a, j+1, high);
}

void main()
{
                                int n, a[1000],k

                                clock_t st,et;
                                double ts;
                                clrscr();
                                printf("\n Enter How many Numbers: ");
                                 scanf("%d", &n);
                                printf("\nThe Random Numbers are:\n"); for(k=1; k<=n;
                                k++)
                                    {
                                        a[k]=rand();
                                        printf("%d\t",a[k]);
                                    }
                                st=clock();
                                QuickSort(a, 1, n);
                                et=clock();
                                ts=(double)(et-st)/CLOCKS_PER_SEC;
```

```
                printf("\nSorted Numbers are: \n ");
                for(k=1; k<=n; k++)
                        printf("%d\t", a[k]);
                printf("\nThe time taken is %e",ts);
                getch();
}
```

OUTPUT:



Viva Questions:

Q1- Define the term "Quick Sort".
Q2- Evaluate Upper Bound Complexity of Quick sort.
Q3- Evaluate Best case of Quick sort.
Q4- Evaluate Worst case of Quick sort.
Q5- Explain method used by Quick sort.

Experiment No. 2

Aim: Using OpenMP, implement a parallelized 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, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

.

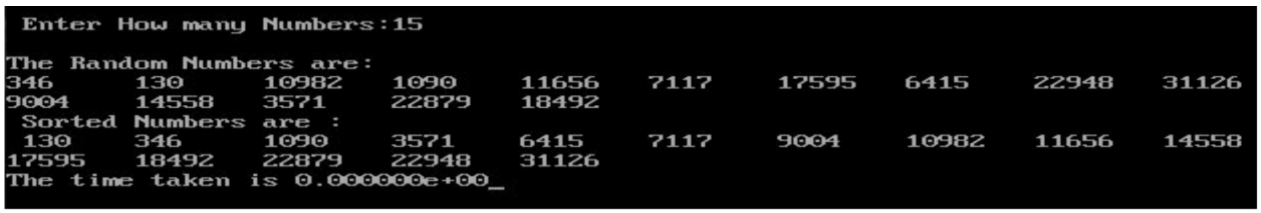
Tools / Software: Turbo C++ /Code Blocks

Description: In computer science, **merge sort** (also commonly spelled as **mergesort**) is an efficient, general-purpose, and comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. Merge sort is a divide and conquer algorithm

```
1. include <stdio.h>
2. include <conio.h>
#include<time.h>
void Merge(int a[], int low, int mid, int high)
{
        int i, j, k, b[20]; i=low; j=mid+1;
        k=low; while ( i<=mid && j<=high
        )
        {
                if( a[i] <= a[j] ) b[k++] =
                        a[i++] ;
                else
                        b[k++] = a[j++] ;
        }
        while (i<=mid)          b[k++] = a[i++] ;
        while (j<=high)  b[k++] = a[j++] ;
                for(k=low; k<=high; k++)
                        a[k] = b[k];
}
void MergeSort(int a[], int low, int high)
{
        int mid;
        if(low >= high)
                return;
        mid = (low+high)/2 ;
        MergeSort(a, low, mid);
        MergeSort(a, mid+1, high);
        Merge(a, low, mid, high);
}
void main()
{

        int n, a[2000],k;
        clock_t st,et;
        double ts;
        clrscr();
        printf("\n Enter How many Numbers:"); scanf("%d",
        &n);
        printf("\nThe Random Numbers are:\n");
        for(k=1; k<=n; k++)
```

```
        {
                a[k]=rand();
                printf("%d\t", a[k]);
        }
        st=clock();
        MergeSort(a, 1, n);
        et=clock();
        ts=(double)(et-st)/CLOCKS_PER_SEC;
        printf("\n Sorted Numbers are : \n ");
        for(k=1; k<=n; k++)
                printf("%d\t", a[k]);
        printf("\nThe time taken is %e",ts);
        getch();
}
```

OUTPUT:



Viva Questions:

Q1- Define “Merge Sort”.
Q2- Evaluate Upper case of Complexity of Merge Sort.
Q3- Evaluate Best case Complexity of Merge Sort.
Q4- Evaluate Worst case Complexity of Merge Sort.
Q5- Explain the method used by Merge Sort.

Experiment No. 3

Aim- a. Obtain the Topological ordering of vertices in a given digraph.

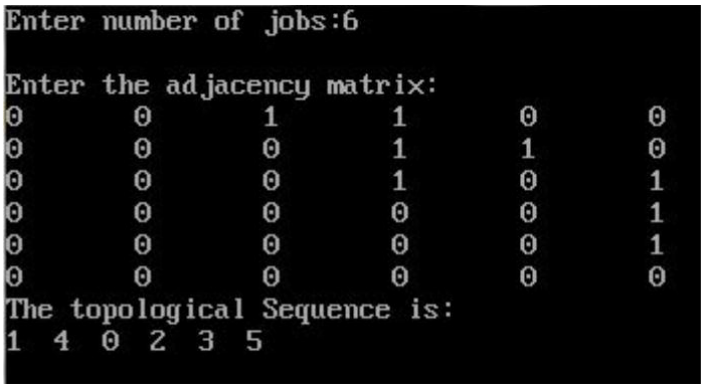
Tools / Software: Turbo C++ /Code Blocks

Source Code:

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n,indegre[10];
void find_indegre()
{int j,i,sum;
        for(j=0;j<n;j++)
        {
                sum=0;
                for(i=0;i<n;i++)
                        sum+=a[i][j];
                indegre[j]=sum;
        }
}
void topology()
{
        int i,u,v,t[10],s[10],top=-1,k=0;
        find_indegre();
        for(i=0;i<n;i++)
        {
                if(indegre[i]==0) s[++top]=i;
        }
        while(top!=-1)
        {
                u=s[top--];
                t[k++]=u;
                for(v=0;v<n;v++)
                {
                        if(a[u][v]==1)
                        {
                                indegre[v]--;
                                if(indegre[v]==0) s[++top]=v;
                        }
                }
        }
        printf("The topological Sequence is:\n");
        for(i=0;i<n;i++)
                printf("%d  ",t[i]);

}
void main()
{
        int i,j;
        clrscr();
        printf("Enter number of jobs:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=0;i<n;i++)
```

```
        {
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
        }
        topology();
        getch();
}
```

OUTPUT:



Viva Questions:

Q1- What is the first step of Topological ordering.
Q2- What is the purpose of Topological sorting .
Q3- Describe Application of Topological sorting.
Q4- What is efficient time complexity of Topological sorting.
Q5- What are applications of Topological sorting.

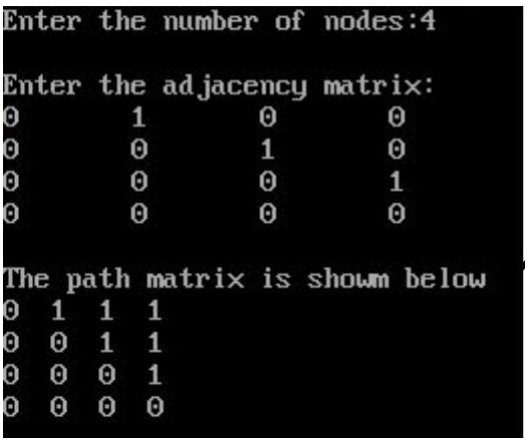
Experiment No.3(B)

Aim: Compute the transitive closure of a given directed graph using Warshall's algorithm.

Tools / Software: Turbo C++ /Code Blocks

```
5. include <stdio.h>
6. include <conio.h>
int n,a[10][10],p[10][10];
void path()
{
        int i,j,k;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        p[i][j]=a[i][j];
        for(k=0;k<n;k++)
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                                if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;
}
void main()
{
        int i,j;
        clrscr();
        printf("Enter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
        path();
        printf("\nThe path matrix is showm below\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%d  ",p[i][j]);
                printf("\n");
        }
        getch();
}
```

Output:



Viva Questions:

Q1- Explain Warshall algorithm?
Q2- Evaluate formula for Warshall algorithm.
Q3- Differentiate between Warshall and Floyd algorithm.
Q4- Explain the approach followed by Warshall algorithm.
Q5- Who proposed Floyd-Warshall algorithm?

Experiment No. 4

Aim: Implement 0/1 Knapsack problem using Dynamic Programming.

Tools / Software: Turbo C++ /Code Blocks

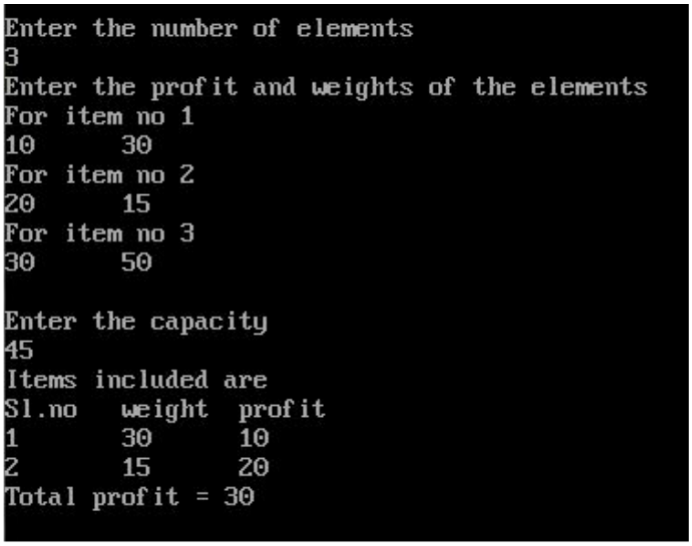
Description: C program to find the frequency of characters in a string: This program counts the frequency of characters in a string, i.e., which character is present how many times in the string. For example, in the string "code" each of the characters 'c,' 'd,' 'e,' and 'o' has occurred one time. Only ***lower case alphabets*** are considered, other characters (uppercase and special characters) are ignored. You can easily modify this program to handle uppercase and special symbols.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j)
{
	return ((i>j)?i:j);
}
int knap(int i,int j)
{
	int value;
	if(v[i][j]<0)
	{
		if(j<w[i])
			value=knap(i-1,j);
		else
			value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i])); v[i][j]=value;
	}
	return(v[i][j]);
}
void main()
{
	int profit,count=0;
	clrscr();
	printf("\nEnter the number of elements\n");
	scanf("%d",&n);
	printf("Enter the profit and weights of the elements\n");
	for(i=1;i<=n;i++)
	{
		printf("For item no %d\n",i);
		scanf("%d%d",&p[i],&w[i]);
	}
	printf("\nEnter the capacity \n");
	scanf("%d",&cap);
	for(i=0;i<=n;i++)
		for(j=0;j<=cap;j++)
			if((i==0)||(j==0))

				v[i][j]=0;
			else
				v[i][j]=-1;
```

```
        profit=knap(n,cap);
        i=n;
        j=cap;
        while(j!=0&&i!=0)
        {
                if(v[i][j]!=v[i-1][j])
                {
                        x[i]=1;
                        j=j-w[i];
                        i--;
                }
                else
                        i--;
        }
        printf("Items included are\n");
        printf("Sl.no\tweight\tprofit\n");
        for(i=1;i<=n;i++)
                if(x[i])
                        printf("%d\t%d\t%d\n",++count,w[i],p[i]); printf("Total
        profit = %d\n",profit); getch();

}
```

OUTPUT:



Viva Questions:

Q1- What is knapsack problem?
Q2- What is dynamic programming?
Q3- Describe a real life example of 0/1 knapsack problem.
Q4- Which methods can be used to solve the knapsack problem?
Q5- Show the implementation of 0/1 knapsack problem.

Experiment No. 5

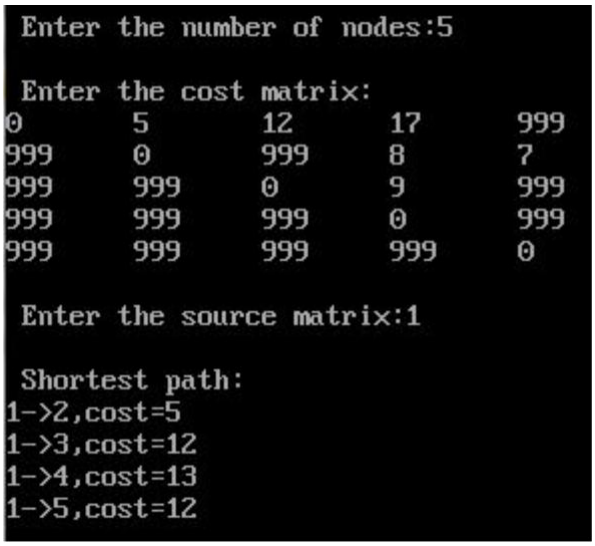
Aim: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Tools / Software: Turbo C++ /Code Blocks

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[100])
{
	int i,u,count,w,flag[10],min;
	for(i=1;i<=n;i++)
		flag[i]=0,dist[i]=cost[v][i];
	count=2;
	while(count<=n)
	{
		min=99;
		for(w=1;w<=n;w++)
			if(dist[w]<min && !flag[w])
				min=dist[w],u=w;
		flag[u]=1;
		count++;
		for(w=1;w<=n;w++)
			if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
				dist[w]=dist[u]+cost[u][w];
	}
}

void main()
{
	int n,v,i,j,cost[10][10],dist[10];
	clrscr();
	printf("\n Enter the number of nodes:");
	scanf("%d",&n);
	printf("\n Enter the cost 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]=infinity;
```

```
        }
    printf("\n Enter the source matrix:");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n Shortest path:\n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
    getch();
}
```

Output:



Viva Questions:

Q1- What is Dijkstra's algorithm?
Q2- What is time complexity of Dijkstra algorithm?.
Q3-How many priority queue operations are involved in Dijkstra's algorithm?.
Q4- Dijkstra's algorithm is prime example for _____.
Q5- Dijkstra's algorithm is used to solve _____ problems.

Experiment No. 6

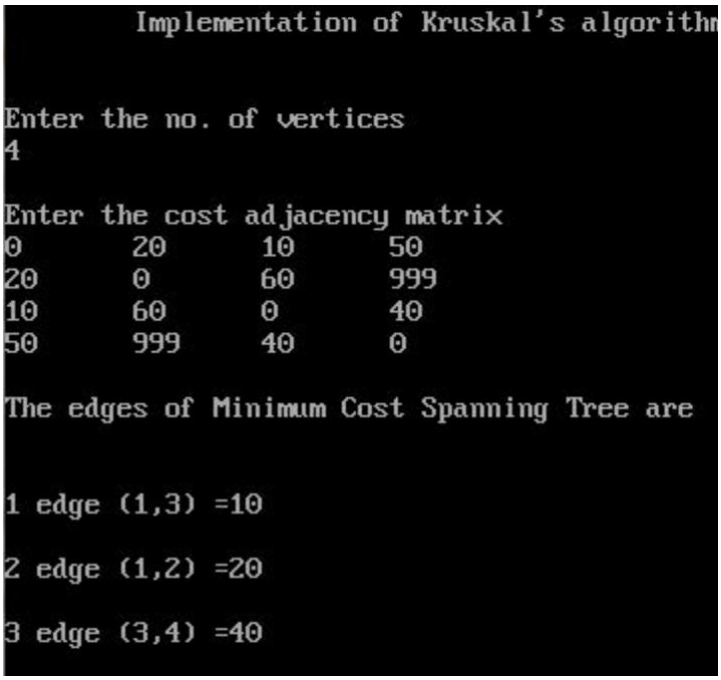
Aim: Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Tools / Software: Turbo C++

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
        clrscr();
        printf("\n\n\tImplementation of Kruskal's algorithm\n\n"); printf("\nEnter 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\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("\n%d edge (%d,%d) =%d\n",ne++,a,b,min); mincost +=min;
                }
                cost[a][b]=cost[b][a]=999;
        }
```

```
        printf("\n\tMinimum cost = %d\n",mincost); getch();
}
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;

}
```

Output:



Viva Questions:

Q1- Describe Kruskals Algorithm.
Q2- What is Minimum cost spanning tree?
Q3- What approach is used by Kruskal's Algorithm?
Q4- What is time complexity of Kruskal's Algorithm?
Q5- Kruskal's Algorithm is best suited for the dense graphs than the prim's algorithm true or false?

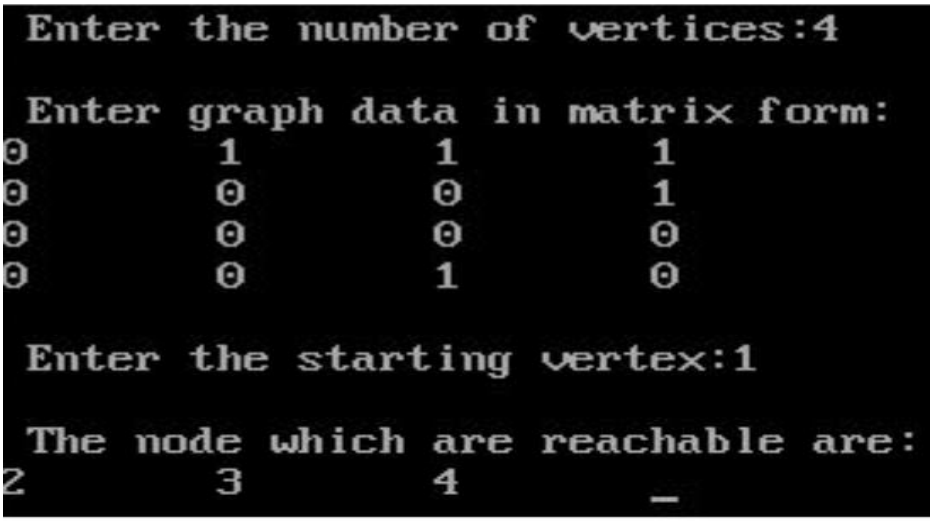
Experiment No.7(a)

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

Tools / Software: Turbo C++

```
#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;
 clrscr();
 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();
}
```

OUTPUT



Viva Questions:

Q1- Describe BFS method in brief.
Q2- Differentiate between BFS and DFS.
Q3- When the BFS of a graph is unique?
Q4- The BFS traversal of a graph will result into?
Q5- The data structure used in standard implementation of BFS is?

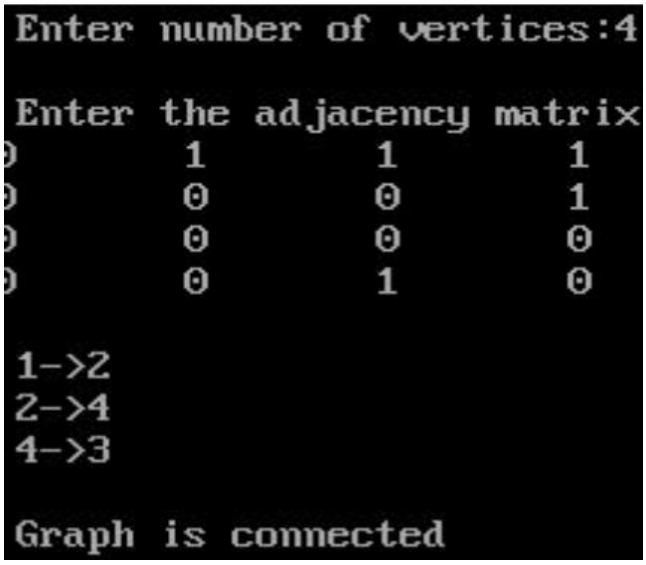
Experiment No. 7(B)

Aim: Check whether a given graph is connected or not using DFS method.

Tools / Software: Turbo C++.

```
#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);
        }
}
void main()
{
        int i,j,count=0;
        clrscr();
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                reach[i]=0;
                for(j=1;j<=n;j++)
                        a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for(i=1;i<=n;i++)
        {
                if(reach[i])
                count++;

        }
        if(count==n)
                printf("\n Graph is connected");
        else
                                printf("\n Graph is not connected");
getch();
}
```

OUTPUT



Viva Questions:

Q1- Describe DFS method in brief.
Q2- Define backtracking in brief.
Q3- Describe State Space Tree in brief

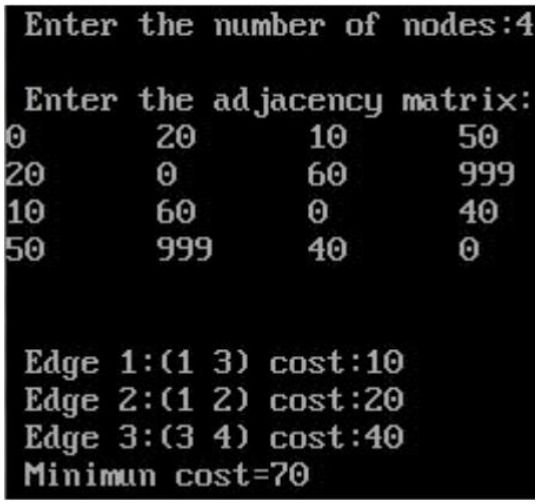
Experiment No. 8

Aim: Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm

Tools / Software: Turbo C++

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
        clrscr();
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter 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");
        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("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;
                        visited[b]=1;
                }

                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost); getch();
}
```

OUTPUT:



Viva Questions:

Q1- Describe Prims algorithm in brief.
Q2- Describe the method used by Prims algorithm.
Q3- Differentiate between Prims and Kruskals algorithm..

Experiment No. 9

Aim: Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

Tools Used: Turbo C++

```
#include<stdio.h>
#include<conio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
        int i,j,k;
        for(k=1;k<=n;k++)
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                if(i==j)
                                        p[i][j]=0;
                                else
                                        p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
        if(a<b)
                return(a);
        else
                return(b);
}
void main()
{
        int p[10][10],w,n,e,u,v,i,j;;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        printf("\n Enter the number of edges:\n");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        p[i][j]=999;
        }
        for(i=1;i<=e;i++)

        {
                printf("\n Enter the end vertices of edge%d with its weight \n",i);
                scanf("%d%d%d",&u,&v,&w);
                p[u][v]=w;
        }
        printf("\n Matrix of input data:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf("%d \t",p[i][j]);
                printf("\n");
        }
        floyds(p,n);
```

```
        printf("\n Transitive closure:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf("%d \t",p[i][j]);
                printf("\n");
        }
        printf("\n The shortest paths are:\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                {
                        if(i!=j)
                                printf("\n <%d,%d>=%d",i,j,p[i][j]);
                }
        getch();
}
```

OUTPUT:

```
 Enter the number of vertices:4

 Enter the number of edges:
5

 Enter the end vertices of edge1 with its weight
1       3       3

 Enter the end vertices of edge2 with its weight
2       1       2

 Enter the end vertices of edge3 with its weight
3       2       7

 Enter the end vertices of edge4 with its weight
3       4       1

 Enter the end vertices of edge5 with its weight
4       1       6_
```

```
999     999     3       999
2       999     999     999
999     7       999     1
6       999     999     999

 Transitive closure:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0

 The shortest paths are:

 <1,2>=10
 <1,3>=3
 <1,4>=4
 <2,1>=2
 <2,3>=5
 <2,4>=6
 <3,1>=7
 <3,2>=7
 <3,4>=1
 <4,1>=6
 <4,2>=16
 <4,3>=9
```

Viva Questions:

Q1- Describe Floyd algorithm in brief.
Q2- Explain the implementation of Floyd algorithm.
Q3- Why Floyd Warshall Algorithm is dynamic programming?

Experiment No.10

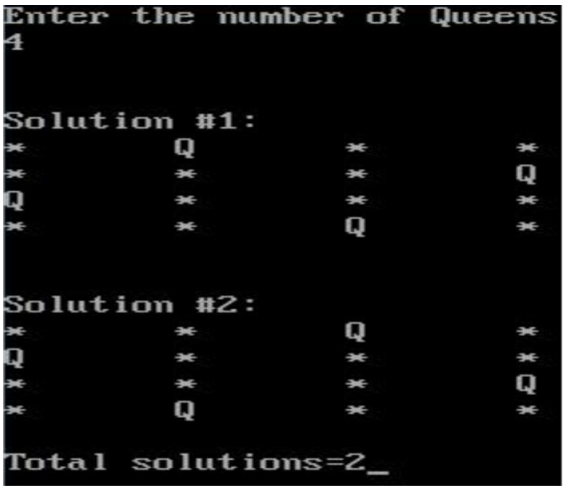
Aim: Implement N Queen's problem using Back Tracking.

Tools Used: Turbo C++

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
{
	int i;
	for(i=1;i<pos;i++)
	{
		if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
			return 0;
	}
	return 1;
}
void print_sol(int n)
{
	int i,j;
	count++;
	printf("\n\nSolution #%d:\n",count);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i]==j)
				printf("Q\t");
			else
				printf("*\t");
		}
		printf("\n");
	}
}
void queen(int n)
{
	int k=1;
	a[k]=0;
	while(k!=0)
	{

		a[k]=a[k]+1;
		while((a[k]<=n)&&!place(k))
			a[k]++;
```

```
                if(a[k]<=n)
                {
                        if(k==n)
                                print_sol(n);
                        else
                        {
                                k++;
                                a[k]=0;
                        }
                }
                else
                        k--;
        }
}
void main()
{
        int i,n;
        clrscr();
        printf("Enter the number of Queens\n");
        scanf("%d",&n);
        queen(n);
        printf("\nTotal solutions=%d",count);
        getch();
}
```

OUTPUT:



Viva Questions:

Q1: What is N Queen Problem and explain with example.
Q2: Which type of Algorithm is used to solve the N Queen Problem.
Q3: How many solutions exist for 4 Queen and 8 Queen problem.

Experiment No. 11

Aim: Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

Tools / Software: Turbo C++

```
#include <stdio.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;

int tsp(int c)
{
 int count, nearest_city = 999;
 int minimum = 999, temp;
 for(count = 0; count < limit; count++)
 {
 if((matrix[c][count] != 0) && (visited_cities[count] == 0))
 {
 if(matrix[c][count] < minimum)
 {
 minimum = matrix[count][0] + matrix[c][count];
 }
 temp = matrix[c][count];
 nearest_city = count;
 }
 }
 if(minimum != 999)
 {
 cost = cost + temp;
 }
 return nearest_city;
}

void minimum_cost(int city)
{
 int nearest_city;
 visited_cities[city] = 1;
 printf("%d ", city + 1);
 nearest_city = tsp(city);
 if(nearest_city == 999)
 {
 nearest_city = 0;
 printf("%d", nearest_city + 1);
 cost = cost + matrix[city][nearest_city];
 return;
 }
```

```
 minimum_cost(nearest_city);
}

int main()
{
 int i, j;
 printf("Enter Total Number of Cities:\t");
 scanf("%d", &limit);
 printf("\nEnter Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
 printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
 for(j = 0; j < limit; j++)
 {
 scanf("%d", &matrix[i][j]);
 }
 visited_cities[i] = 0;
 }
 printf("\nEntered Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
 printf("\n");
 for(j = 0; j < limit; j++)
 {
 printf("%d ", matrix[i][j]);
 }
 }
 printf("\n\nPath:\t");
 minimum_cost(0);
 printf("\n\nMinimum Cost: \t");+
 printf("%d\n", cost);
 return 0;
```

OUTPUT:

```
[student@localhost ~]$ cd S
[student@localhost S]$ gcc tsp.c
[student@localhost S]$ ./a.ou
bash: ./a.ou: No such file or directory
[student@localhost S]$ ./a.out
Enter Total Number of Cities:   4

Enter Cost Matrix

Enter 4 Elements in Row[1]
1 2 3 4

Enter 4 Elements in Row[2]
5 6 7 8

Enter 4 Elements in Row[3]
3 4 5 6

Enter 4 Elements in Row[4]
9 8 4 3

Entered Cost Matrix

1 2 3 4
5 6 7 8
3 4 5 6
9 8 4 3

Path:   1 4 3 2 1

Minimum Cost:   17
[student@localhost S]$
```

Viva Questions:

Q1- Describe Travelling Salesperson problem in brief.
Q2- Discuss method used by Travelling Salesperson problem.
Q3- Give simple approach for Travelling Salesperson problem.