Data Structures and Algorithms

SY B. Tech. E&TC | Batch 2017-21

Mini-Project Report

Travelling Salesman Problem

Name of the Student 1: Shivangi Srivastava		**PRN 1: 17070123099**
Name of the Student 2: Ramya Bardae		**PRN 2: 17070123080**
Academic Year : 2018-19	**Semester :** III	**Division: B**

Objective:

Write a C program for the following problem**:**

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

Theoretical Background:

An explanation of the Data Structure concepts is given below.

1. Arrays:

Here we have used basic concept of array to store the input given by the user.

We have created separate arrays to store the city number, the cities visited, and the corresponding value associated with each city that is either 1 or 0.

2. Recursion:

We have used recursion to create the function for minimum distance. After every iteration of the loop, the function repeats itself to check whether the distance between two cities is minimum out of the possible combinations.

3. Branch and bound algorithm:

The concept of branch and bound is used here to make the problem simpler and to give the most optimum result even when the no. of cities is more.

Branch and branch algorithm breaks down the problem into smaller parts so that all parts can be solved simultaneously, and the compile time is reduced.

The time complexity using naïve programming is (n-1!) where n is the no. of nodes or cities but branch and bound reduces this time.

It is similar to the 0 -1 knapsack issue and the same concept is used here.

A brief explanation to the approach towards solving this problem:

The travelling salesman problem is an NP-hard problem, which means that the complexity increases as we increase the no. of cities and so does the time taken to solve it.

There exists a no. of methods used to solve this problem including backtracking and divide and conquer, using branch and bound algorithm and using dynamic programming.

The base equation of solving this method using dynamic programming is :
T (i , s) = min ((i , j) + T (j , S – { j })) ; S!= Ø   ; j € S ;

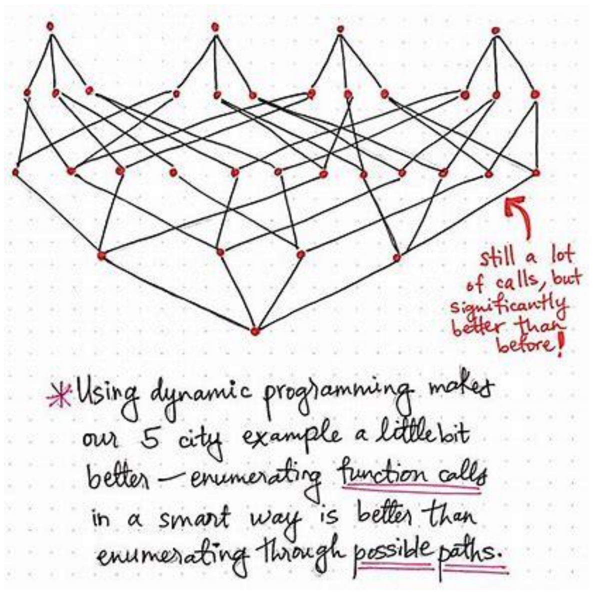
S is set that contains non-visited vertices

T (i,s) means that we start from ‘i’ and check every element of the set S. After checking minimum condition for each element we return back to ‘i’.

(i,j) here is the distance of path from one node to another.

T(j , S-{j}) denotes recursion , where we check the same condition.

Let us assume we have 5 cities A, B, C, D, E. A is the starting point. In hierarchal format it would look like this:



*Using dynamic programming makes our 5 city example a little bit better — enumerating function calls in a smart way is better than enumerating through possible paths.

Algorithm:

STEP 1) Start the program and declare the global variables.

STEP 2) Create a function for tsp

STEP 3) Initialize the values of minimum and nearby to the maximum value say 9999

STEP 4) Create a for loop counting the number of cities.

STEP 5) Check for the minimum value of the routes and add the sum of the routes.

STEP 6) Keep adding the new shortest distance to the previous distances till the loop continues.

STEP 7) Create a function for minimum distance called min_dist and initialize a variable x

STEP 8) If the city is already visited assign the value 1 else assign the value 0, the cities with value 1 go to the visited [] array.

STEP 9) Using recursion keep continuing the same process for the given number of cities

STEP 10) Now, create a main function

STEP 11) take the number of cities to be entered from the user

STEP 12) Create a for loop for entering the element in each row

STEP 13) Create a for loop to display the input in the matrix form to calculate the shortest distance.

STEP 14) Using call by reference use the min_dist function to calculate the shortest path and display the minimum distance and the route.

STEP 15) stop.

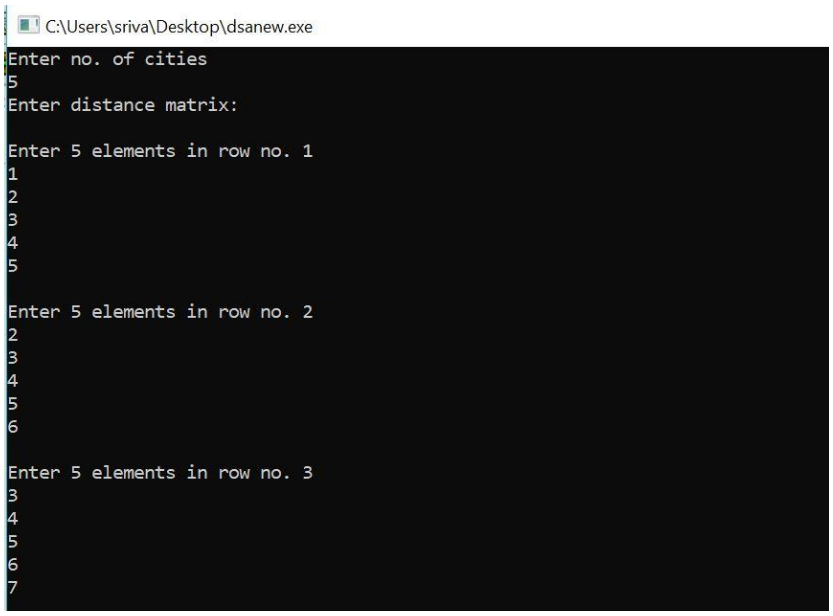
Program:

```
/*dsa miniproject*/

/*travelling salesman problem*/

#include <stdio.h>

#include <stdlib.h>

int n;

int a [50][50];

int visited [20],dist=0;

/* creating function for tsp*/

int tsp(int b)

{

int count,nearby=9999,temp,min=9999;

/*min and nearby initialized to a max value*/

for (count=0;count<n;count++)

{

     if((a[b][count]!=0)&&(visited[count]==0))

/*checking for available cities*/

     {

          if(a[b][count]<min)/*checking if route is shortest*/

          {

               min=a[count][0]+a[b][count];

/*reassigning value of shortest route to compare with next
possible route*/
```

```
          }

          temp=a[b][count];

          nearby=count;

     }

}

if (min !=9999)

{dist=dist+temp;}

/*adding the new shortest distance to the previous distance*/

return nearby;

}

void min_dist(int x)

{    int nearby;

     visited[x]=1;

/*assigning check value 1 to the visited city*/

     printf("%d--->",x+1);

     nearby=tsp(x);/*calling function tsp for shortest path*/

     if(nearby==9999)

     {    nearby=0;

/*assigning value 0 to the visited city if it is not the
shortest path*/

          printf("%d",nearby+1);

          dist=dist+a[x][nearby];

          return;
     }

     min_dist(nearby);

/*using recursion*/

}



int main() {

     int i,j;

     printf("Enter no. of cities\n");

     scanf("%d",&n);

     printf("Enter distance matrix:\n");

/*taking input from user*/

     for(i=0;i<n;i++)

     {

     printf("\nEnter %d elements in row no %d\n",n,i+1);

          for(j=0;j<n;j++)

          {scanf("%d",&a[i][j]);

          }

          visited[i]=0;

     }

printf("Entered distance matrix is;\n");

/*display input in matrix form*/

     for(i=0;i<n;i++)
```

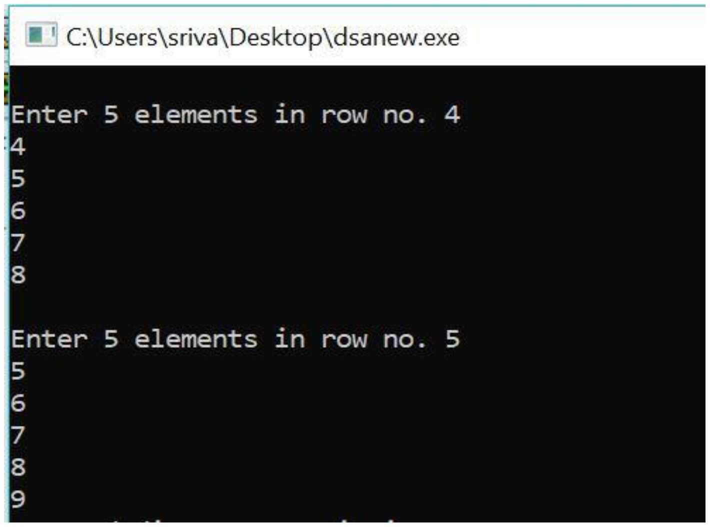
```
    {printf("\n");

        for(j=0;j<n;j++)

        {printf("%d\t",a[i][j]);

        }

    }

    printf("\nShortest path is:\n");

/*display output*/

    min_dist(0);

    printf("\nMinimum distance is :\n");

    printf("%d",dist);

    return 0;

}
```

Output Screenshots:



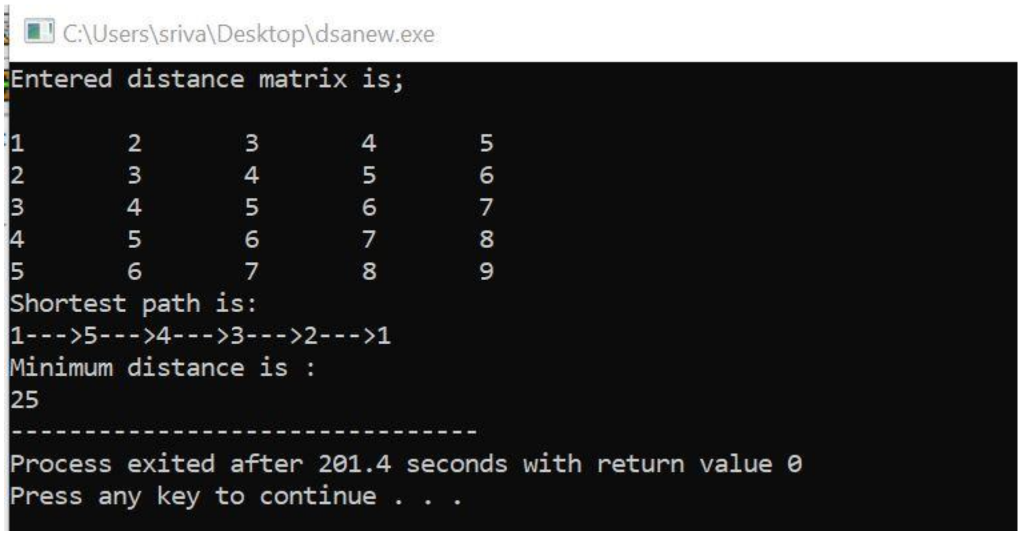
```
C:\Users\sriva\Desktop\dsanew.exe
Enter no. of cities
5
Enter distance matrix:

Enter 5 elements in row no. 1
1
2
3
4
5

Enter 5 elements in row no. 2
2
3
4
5
6

Enter 5 elements in row no. 3
3
4
5
6
7
```

1. Taking input from user



```
C:\Users\sriva\Desktop\dsanew.exe

Enter 5 elements in row no. 4
4
5
6
7
8

Enter 5 elements in row no. 5
5
6
7
8
9
```

2. Taking input from user



```
C:\Users\sriva\Desktop\dsanew.exe
Entered distance matrix is;

1       2       3       4       5
2       3       4       5       6
3       4       5       6       7
4       5       6       7       8
5       6       7       8       9
Shortest path is:
1--->5--->4--->3--->2--->1
Minimum distance is :
25
--------------------------------
Process exited after 201.4 seconds with return value 0
Press any key to continue . . .
```

3. Display input in matrix form and display output of shortest path and minimum distance.

Conclusion:

The travelling salesman problem is a complex problem which can be solved by a lot of methods. The method which gives the most optimum result is that of branch and bound algorithm along with dynamic. The travelling salesman problem is an issue of analyzing a dataset and optimizing it to achieve the best and least time consuming result. This problem has a lot of modern day implications , like:

- Genetics: Determining the path of a mutant gene.
- Medicine: time taken by cancerous cells to infect other cells of the body.
- Route optimization: like cab booking services and apps show us the shortest route
- Maps: like google maps which show the fastest route to the destination etc.,

To solve this problem , set of arrays can be created for each variable present in the problem.

Each variable can be a assigned a dynamic and changeable value which acts as a key or checkpoint. The problem is broken down into small parts to make it easier to solve and reduce the complexity, thus allowing us to reach the the solution in the smallest time.

In this way we can solve the travelling salesman problem.