# PART A

(PART A : TO BE REFFERED BY STUDENTS)

## Experiment No.06

### A.1 Aim:

Implementation of Dynamic programming approach algorithm - Traveling sales person problem

### A.2 Prerequisite:

C programming

### A.3 Outcome:

**After successful completion of this experiment students will be,**

Apply concepts of Trees and Graphs to a given problem.

Apply the concept of Greedy and Dynamic Programming approach to solve problems.

### A.4 Theory:

**Dynamic Programming Approach:**

Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems. It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

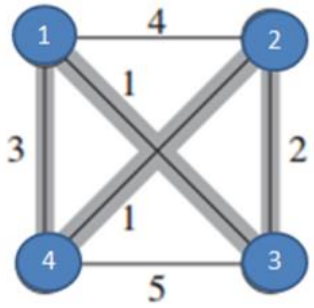
Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.

**Travelling Sales Person Problem:**

The Travelling Salesman Problem (TSP) is the challenge of finding the shortest yet most efficient route for a person to take given a list of specific destinations.

The traveling salesman problems abide by a salesman and a set of cities. The salesman has to visit every one of the cities starting from a certain one (e.g., the hometown) and to return to the same city. The challenge of the problem is that the traveling salesman needs to minimize the total length of the trip.

**Example:**




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 4 | 1 | 3 |
| 2 | 4 | 0 | 2 | 1 |
| 3 | 1 | 2 | 0 | 5 |
| 4 | 3 | 1 | 5 | 0 |

T (i, S) = min [ w(i,j) + T(j, {S-j} ]

T (1, {2,3,4} = min (w(1,2) + T(2, {3,4}) ] = 4 +7 = 11

T ( 1, {2,3,4} ) = minimum of
= { (1,2) + T (2, {3,4} ) 4+6=10
= { (1,3) + T (3, {2,4} ) 1+3=4
= { (1,4) + T (4, {2,3} ) 3+3=6

T (2, {3,4} ) = minimum of
= { (2,3) + T (3, {4} ) 2+5=7
= { (2,4) + T {4, {3} ) 1+5=6

T (3, {2,4} ) = minimum of
= { (3,2) + T (2, {4} ) 2+1=3
= { (3,4) + T {4, {2} ) 5+1=6

T (4, {2,3} ) = minimum of
= { (4,2) + T (2, {3} ) 1+2=3
= { (4,3) + T {3, {2} ) 5+2=7

T ( 3, {4} ) = (3,4) + T (4, {} ) 5+0=5

T ( 4, {3} ) = (4,3) + T (3, {} ) 5+0=5

T ( 2, {4} ) = (2,4) + T (4, {} ) 1+0=1

T ( 4, {2} ) = (4,2) + T (2, {} ) 1+0 = 1

T ( 2, {3} ) = (2,3) + T (3, {} ) 2+0 = 2

T ( 3, {2} ) = (3,2) + T (2, {} ) 2+0=2

T ( 1, {2,3,4} ) = minimum of

= { (1,2) + T (2, {3,4} ) 4+6=10 in this path we have to add +1 because this path ends with 3. From there we have to reach 1 so 3->1 distance 1 will be added total distance is 10+1=11

= { (1,3) + T (3, {2,4} ) 1+3=4 in this path we have to add +3 because this path ends with 3. From there we have to reach 1 so 4->1 distance 3 will be added total distance is 4+3=7

= { (1,4) + T (4, {2,3} ) 3+3=6 in this path we have to add +1 because this path ends with 3. From there we have to reach 1 so 3->1 distance 1 will be added total distance is 6+1=7

- Minimum distance is 7 which includes path 1->3->2->4->1.

# PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

| | |
|---|---|
| Roll. No.71 | Name: Omkar Gujja |
| Class: 2 | Batch: MTRX |
| Date of Experiment: 20/03/2021 | Date of Submission: 27/03/2021 |
| Grade: | |

## B.1 Software Code written by student:

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 10
```

```
struct edge
{
      int u;
       int v;
};

int n;
int adj[MAX][MAX];
int cost;
int completed[MAX];

void create_graph();
void mincost(int city)
{
          int i,ncity;
          completed[city]=1;
          printf("%d--->",city+1);
          ncity=least(city);
          if(ncity==999)
          {
                    ncity=0;
```

```
                printf("%d",ncity+1);
                cost+=adj[city][ncity];
                return;
        }
        mincost(ncity);
}
int least(int c)
{
        int i,nc=999;
        int min=999,kmin;
        for(i=0;i < n;i++)
        {
                if((adj[c][i]!=0)&&(completed[i]==0))
                        if(adj[c][i]+adj[i][c] < min)
                        {
                                min=adj[i][0]+adj[c][i];
                                kmin=adj[c][i];
                                nc=i;
                        }
        }
        if(min!=999)
                cost+=kmin;
```

```
        return nc;
}
int main()
{
    struct edge tree[MAX];
    create_graph();
    printf("\n\nThe Path is:\n");
        mincost(0); //passing 0 because starting vertex
            printf("\n\nMinimum cost is %d\n ",cost);
    return 0;
}/*End of main()*/
void create_graph()
{
    int i,origin,destin,wt;
    printf("\nEnter number of vertices : ");
    scanf("%d",&n);
    for(i=1; i<=n*n; i++)
    {
        printf("\nEnter edge %d(-1 -1 to quit) : ",i);
        scanf("%d %d",&origin,&destin);
        if((origin == -1) && (destin == -1))
            break;
```

```
            printf("\nEnter weight for this edge : ");
            scanf("%d",&wt);
            if( origin >= n || destin >= n || origin < 0 || destin < 0)
            {
                  printf("\nInvalid edge!\n");
                  i--;
            }
            else
            {
                  adj[origin][destin] = wt;
            }
      }
}
```

## B.2 Input and Output:

```
Enter number of vertices : 4

Enter edge 1(-1 -1 to quit) : 0 0

Enter weight for this edge : 0

Enter edge 2(-1 -1 to quit) : 0 1

Enter weight for this edge : 4

Enter edge 3(-1 -1 to quit) : 0 2

Enter weight for this edge : 1

Enter edge 4(-1 -1 to quit) : 0 3

Enter weight for this edge : 3

Enter edge 5(-1 -1 to quit) : 1 0

Enter weight for this edge : 4

Enter edge 6(-1 -1 to quit) : 1 1

Enter weight for this edge : 0

Enter edge 7(-1 -1 to quit) : 1 2

Enter weight for this edge : 2

Enter edge 8(-1 -1 to quit) : 1 3
```

```
Enter weight for this edge : 2

Enter edge 11(-1 -1 to quit) : 2 2

Enter weight for this edge : 0

Enter edge 12(-1 -1 to quit) : 2 3

Enter weight for this edge : 5

Enter edge 13(-1 -1 to quit) : 3 0

Enter weight for this edge : 3

Enter edge 14(-1 -1 to quit) : 3 1

Enter weight for this edge : 1

Enter edge 15(-1 -1 to quit) : 3 2

Enter weight for this edge : 5

Enter edge 16(-1 -1 to quit) : 3 3

Enter weight for this edge : 0


The Path is:
1--->3--->2--->4--->1

Minimum cost is 7


...Program finished with exit code 0
Press ENTER to exit console.
```

## B.3 Observations and learning:

We have learned how Dynamic Programming works.

## B.4 Conclusion:

We have understood the program of Travelling Sales Problem.

## B.5 Question of Curiosity

Q.1 Travelling salesman problem is an example of

a. Dynamic Algorithm
b. Greedy Algorithm
c. Recursive Approach
d. Divide & Conquer

Soln.

Dynamic Algorithm