Experiment No. : 5

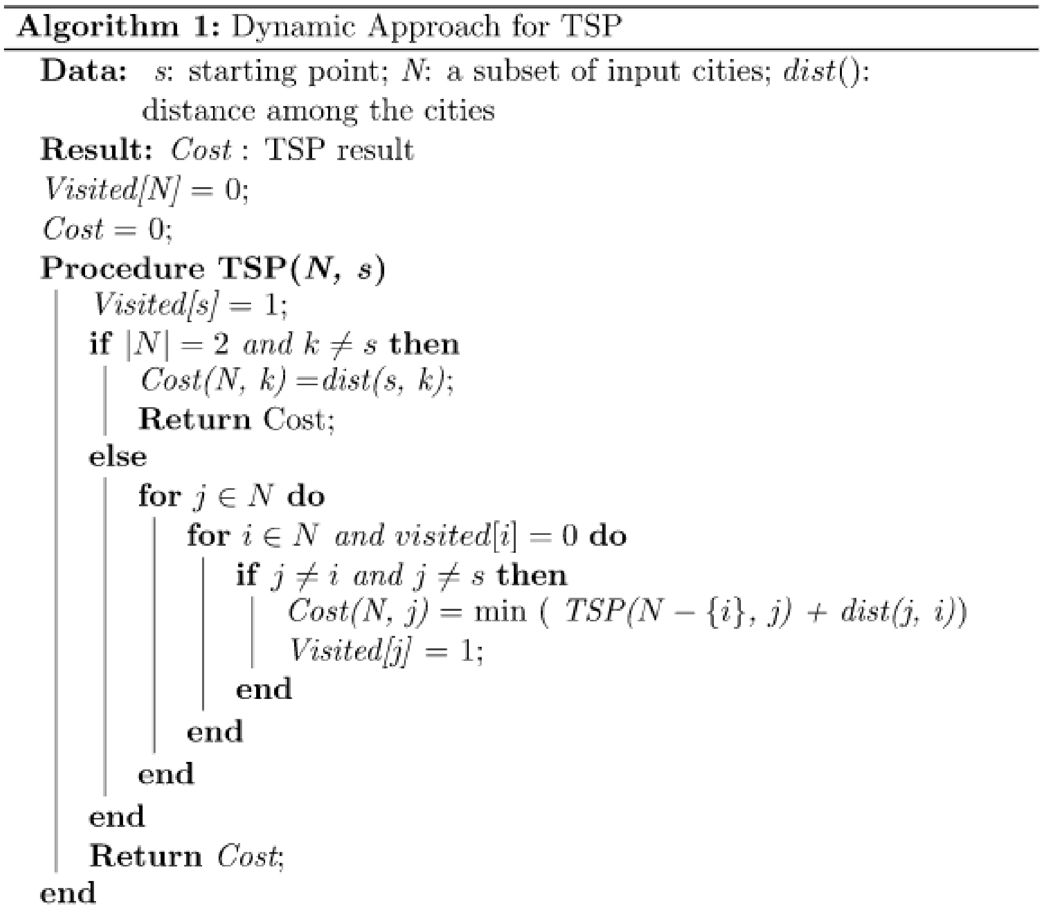
Title: Implement Travelling Salesman Problem using Dynamic approach

Batch:A2 **Roll No.: 16010421073** **Experiment No.: 5**

Aim: To Implement Travelling Salesman Problem for minimum 6 vertices using Dynamic approach and analyse its time Complexity.

Algorithm of Travelling Salesman Problem:

```
Algorithm: Traveling-Salesman-Problem
C ({1}, 1) = 0
for s = 2 to n do
   for all subsets S Є {1, 2, 3, … , n} of size s and containing 1
      C (S, 1) = ∞
   for all j Є S and j ≠ 1
      C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}
Return minj C ({1, 2, 3, …, n}, j) + d(j, i)
```



```
Algorithm 1: Dynamic Approach for TSP
  Data: s: starting point; N: a subset of input cities; dist():
        distance among the cities
  Result: Cost : TSP result
  Visited[N] = 0;
  Cost = 0;
  Procedure TSP(N, s)
     Visited[s] = 1;
     if |N| = 2 and k ≠ s then
        Cost(N, k) = dist(s, k);
        Return Cost;
     else
        for j ∈ N do
           for i ∈ N and visited[i] = 0 do
              if j ≠ i and j ≠ s then
                 Cost(N, j) = min ( TSP(N − {i}, j) + dist(j, i))
                 Visited[j] = 1;
              end
           end
        end
     end
     Return Cost;
  end
```

Explanation and Working of Travelling Salesman Problem:

Problem Statement
A traveller 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?

Solution

Travelling salesman problem is the most notorious computational problem. We can use brute-force approach to evaluate every possible tour and select the best one. For n number of vertices in a graph, there are (n - 1)! number of possibilities.

Instead of brute-force using dynamic programming approach, the solution can be obtained in lesser time, though there is no polynomial time algorithm.

Let us consider a graph G = (V, E), where V is a set of cities and E is a set of weighted edges. An edge e(u, v) represents that vertices u and v are connected. Distance between vertex u and v is d(u, v), which should be non-negative.

Suppose we have started at city 1 and after visiting some cities now we are in city j. Hence, this is a partial tour. We certainly need to know j, since this will determine which cities are most convenient to visit next. We also need to know all the cities visited so far, so that we don't repeat any of them. Hence, this is an appropriate sub-problem.

For a subset of cities S Є {1, 2, 3, ... , n} that includes 1, and j Є S, let C(S, j) be the length of the shortest path visiting each node in S exactly once, starting at 1 and ending at j.

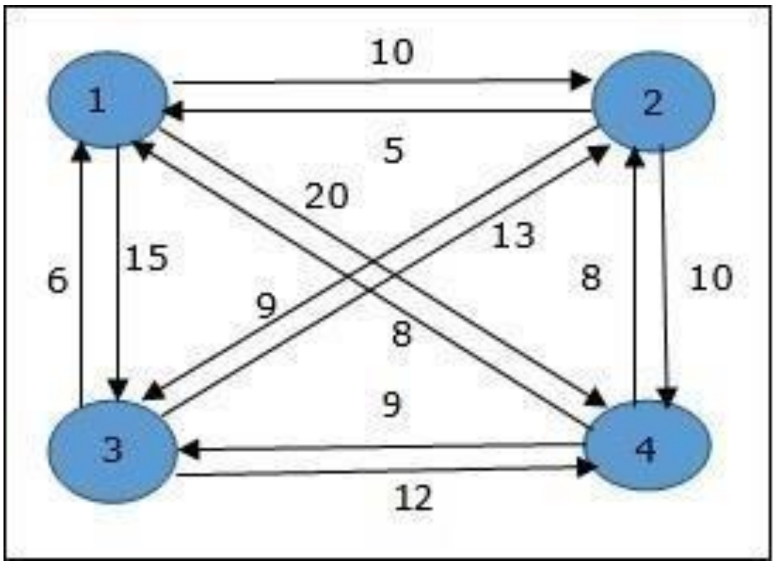
When |S| > 1, we define C(S, 1) = a since the path cannot start and end at 1.

Now, let express C(S, j) in terms of smaller sub-problems. We need to start at 1 and end at j. We should select the next city in such a way that

C(S,j)=minC(S−{j},i)+d(i,j)wherei∈Sandi≠jc(S,j)=minC(s−{j},i)+d(i,j)wherei∈Sandi≠j

Example

In the following example, we will illustrate the steps to solve the travelling salesman problem.



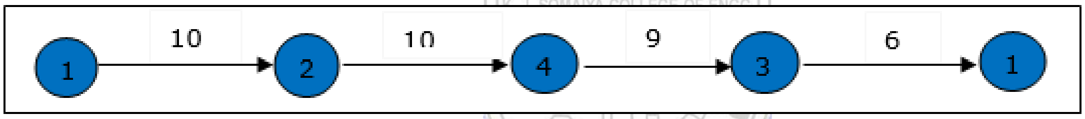
From the above graph, the following table is prepared.

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

The minimum cost path is 35.

Start from cost **{1, {2, 3, 4}, 1}**, we get the minimum value for **d [1, 2]**. When **s = 3**, select the path from 1 to 2 (cost is 10) then go backwards. When **s = 2**, we get the minimum value for **d [4, 2]**. Select the path from 2 to 4 (cost is 10) then go backwards.

When **s = 1**, we get the minimum value for **d [4, 3]**. Selecting path 4 to 3 (cost is 9), then we shallgo to then go to **s = Φ** step. We get the minimum value for **d [3, 1]** (cost is 6).


$$\begin{tikzcd} 1 \arrow[r, "10"] & 2 \arrow[r, "10"] & 4 \arrow[r, "9"] & 3 \arrow[r, "6"] & 1 \end{tikzcd}$$

Derivation of Travelling Salesman Problem:

Time complexity Analysis
In the dynamic algorithm for TSP, the number of possible subsets can be at most N * 2N. Each subset can be solved in O(N) times. Therefore, the time complexity of this algorithm would be O(N2 * 2N)

Program:

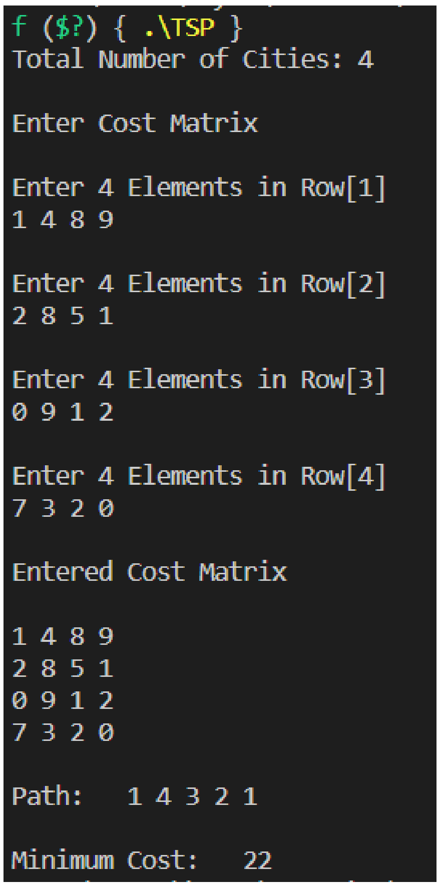
```
#include <stdio.h>
#include <stdlib.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;
int tsp(int c)
{
    int count, nearest_city = 999;
    int minimum = 999, temp;
    for(count = 0; count < limit; count++)
    {
        if((matrix[c][count] != 0) && (visited_cities[count] == 0))
        {
            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("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++)
    {
```

```c
        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(o) of Travelling Salesman Problem:



```
f ($?) { .\TSP }
Total Number of Cities: 4

Enter Cost Matrix

Enter 4 Elements in Row[1]
1 4 8 9

Enter 4 Elements in Row[2]
2 8 5 1

Enter 4 Elements in Row[3]
0 9 1 2

Enter 4 Elements in Row[4]
7 3 2 0

Entered Cost Matrix

1 4 8 9
2 8 5 1
0 9 1 2
7 3 2 0

Path:   1 4 3 2 1

Minimum Cost:   22
```

Post Lab Questions:- Explain how Travelling Salesman Problem using greedy method is different from Dynamic Method in detail.

Answer:

1. Let the given set of vertices be {1, 2, 3, 4,….n}.
2. Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once.
3. Let thecost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far. Now the question is how to get cost(i)?
4. To calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.
5. Let us define a term C(S, i) be the cost of the minimum cost path visitingeach vertex in set S exactly once, starting at 1 and ending at i.
6. We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, thenwe calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.
7. If size of S is 2, then S must be {1, i}, C(S, i) = dist(1, i)Else if size of S is

greater than 2.

8. C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.
9. For a set of size n, we consider n-2 subsets each of size n-1 such that all subsets don't have nth in them.
10. Using the above recurrence relation, we can write dynamic programming based solution.
11. There are at most O(n*2n) subproblems, and each one takes linear time to solve. The total running time is therefore O(n2*2n).
12. The time complexity is much less than O(n!), but still exponential.
13. Space required is also exponential. So this approach is also infeasible even for slightly higher number of vertices.

Conclusion: (Based on the observations):
We Successfully Implemented Travelling Salesman Problem using Dynamic approach.

Outcome:
CO2: Implement Greedy and Dynamic Programming algorithms.

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.

K. J. SOMAIYA COLLEGE OF ENGG.