Experiment No. : 5

Title: Implement Travelling Salesman Problem using Dynamic approach

Batch: A2 **Roll No.: 16010421059** **Experiment No.: 05**

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* $k \neq s$ **then**
    $Cost(N, k) = dist(s, k)$;
    Return Cost;
  else
    for $j \in N$ **do**
      for $i \in N$ *and* $visited[i] = 0$ **do**
        if $j \neq i$ *and* $j \neq s$ **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.

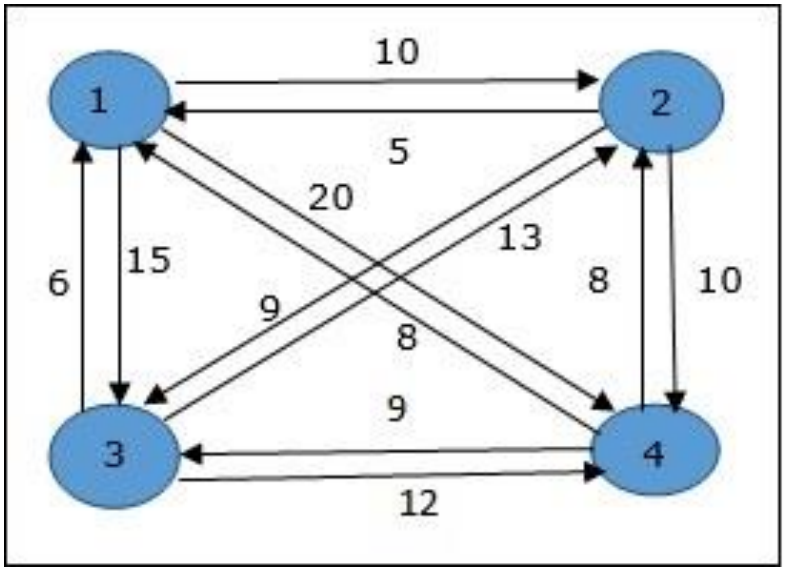
When |S| > 1, we define C(S, 1) = ∝ since the path cannot start and end at 1.

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

$$C(S,j)=\min C(S-\{j\},i)+d(i,j) \text{ where } i \in S \text{ and } i \neq j$$

Example

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



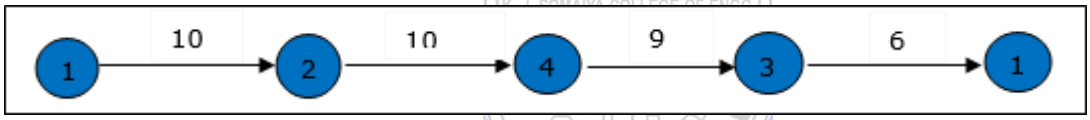
From the above graph, the following table is prepared.

	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 shall go to then go to **s = Φ** step. We get the minimum value for **d [3, 1]** (cost is 6).



Derivation of Travelling Salesman Problem:

Time complexity Analysis

The Travelling Salesman Problem (TSP) is an NP-hard problem, which means that there is no known algorithm that can solve it in polynomial time. Therefore, the time complexity of TSP algorithms is typically analyzed in terms of their worst-case performance.

For a brute-force algorithm that considers all possible tours, the time complexity is O(n!), where n is the number of cities. This means that the number of computations required to find the optimal solution grows very rapidly as the number of cities increases. For example, if there are 20 cities, there are 2.43 x 10^18 possible tours to consider, which is clearly infeasible.

However, there are several algorithms that have been developed to solve the TSP more efficiently. The time complexity of these algorithms varies, but most of them have a worst-case time complexity that is proportional to n^2 * 2^n.

Program(s) of Travelling Salesman Problem:

```
#include<stdio.h>

int ary[1000][1000],completed[1000],n,cost=0;

void takeInput()
{
   int i,j;

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

   for(i=0;i < n;i++)
   {
      printf("\nEnter Elements of Row %d:\n",i+1);

      for( j=0;j < n;j++)
      scanf("%d",&ary[i][j]);

      completed[i]=0;
   }

   printf("\n\nThe cost list is:");

   for( i=0;i < n;i++)
   {
      printf("\n");

      for(j=0;j < n;j++)
      printf("\t%d",ary[i][j]);
   }
}

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+=ary[city][ncity];

        return;
    }

    mincost(ncity);
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;

    for(i=0;i < n;i++)
    {
        if((ary[c][i]!=0)&&(completed[i]==0))
            if(ary[c][i]+ary[i][c] < min)
            {
                min=ary[i][0]+ary[c][i];
                kmin=ary[c][i];
                nc=i;
        }
    }

    if(min!=999)
    cost+=kmin;

    return nc;
}

int main()
{
    takeInput();

    printf("\n\nThe Path is:\n");
    mincost(0);

    printf("\n\nMinimum cost: %d\n ",cost);

    return 0;
}
```


Output(o) of Travelling Salesman Problem:

```
Enter the number of rows/columns: 4
Enter Elements of Row 1:
1 2 3 4
Enter Elements of Row 2:
4 5 6 7
Enter Elements of Row 3:
7 8 9 0
Enter Elements of Row 4:
0 1 2 3
The cost list is:
    1   2   3   4
    4   5   6   7
    7   8   9   0
    0   1   2   3

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

Minimum cost: 18
```

Post Lab Questions:-

Q. Explain how Travelling Salesman Problem using greedy method is different from Dynamic Method in detail:

Ans. The Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem that involves finding the shortest possible route that visits each city in a given list exactly once and returns to the starting city. There are several algorithms that have been developed to solve the TSP, and two of the most commonly used methods are the greedy method and the dynamic programming method. In this answer, I will explain the differences between these two methods.

The Greedy Method:

The greedy method is a simple and intuitive approach to solving the TSP. It starts by selecting an arbitrary city as the starting point and then repeatedly choosing the nearest unvisited city until all cities have been visited. This results in a tour, but it is not necessarily the optimal tour. The greedy method is fast and easy to implement, but it often produces suboptimal solutions.

The Dynamic Programming Method:

The dynamic programming method, on the other hand, is a more complex algorithm that guarantees an optimal solution. It works by breaking the TSP into smaller subproblems and solving each sub problem recursively. Specifically, the dynamic programming approach involves computing a table of optimal solutions for all possible subsets of cities that include the starting

city. The optimal tour is then found by selecting the shortest path that visits all cities exactly once and returns to the starting city.

The main difference between the greedy method and the dynamic programming method is the approach they take to solving the TSP. The greedy method is a heuristic that relies on making locally optimal choices at each step, while the dynamic programming method is an exact algorithm that computes the optimal solution by solving smaller subproblems.

While the greedy method can be fast and easy to implement, it can sometimes produce suboptimal solutions, especially for larger instances of the TSP. The dynamic programming method, on the other hand, is guaranteed to produce the optimal solution, but it can be more computationally intensive and may not be practical for very large instances of the TSP.

Conclusion: (Based on the observations):

Implemented Travelling Salesman Problem for minimum 6 vertices using Dynamic approach and analysed its time Complexity successfully.

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.