Experiment No. : 5

Title: Implement Travelling Salesman Problem using Dynamic approach

Batch: B1 **Roll No.: 16010420133** **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* $k \neq s$ **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.

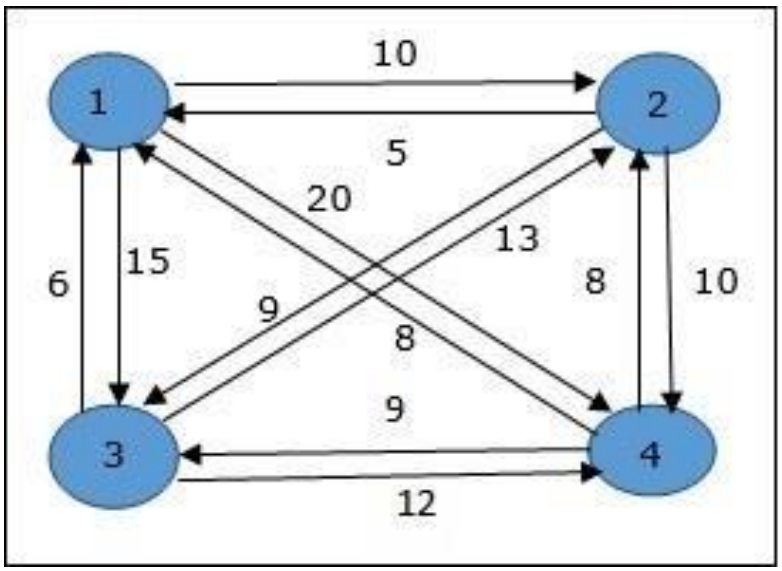
When |S| > 1, we define C(S, 1) = a since the path cannot start and end at 1.

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

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

Example

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



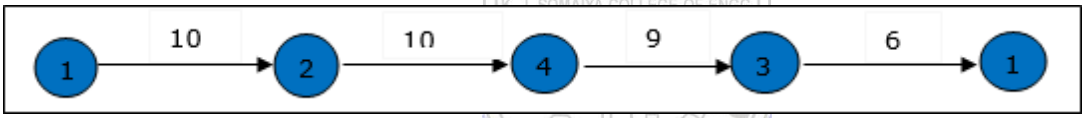
From the above graph, the following table is prepared.

	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

There are at the most 2n.n sub-problems and each one takes linear time to solve.
Therefore, the total running time is O(2n.n).

Program(s) of Travelling Salesman Problem:

```
#include <bits/stdc++.h>

using namespace std;

#define V 4

int travllingSalesmanProblem(int graph[][V], int s)

{
```

```
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);

    int min_path = INT_MAX;
    do {
        int current_pathweight = 0;
        int k = s;
        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];

                min_path = min(min_path, current_pathweight);

    } while (
        next_permutation(vertex.begin(), vertex.end()));

    return min_path;
}
int main()
{
    int graph[][V] = { { 0, 10, 15, 20 },
                       { 10, 0, 35, 25 },
                       { 15, 35, 0, 30 },
                       { 20, 25, 30, 0 } };
    int s = 0;
```

```
    cout << travllingSalesmanProblem(graph, s) << endl;

    return 0;

}
```

Output(o) of Travelling Salesman Problem:

Post Lab Questions:-

```
PS D:\C C++> cd "d:\C C++\" ; if ($?) { g++ cpl5.cpp -o cpl5 } ; if ($?) { .\cpl5 }
80
```

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

here is an important distinction between exact algorithms and heuristics. An exact algorithm is guaranteed to find the exact optimal solution. A heuristic is not, but it is designed to run quickly.

DP is an exact algorithm, at least as it is usually used. There are DP algorithms for TSP. Thus, these algorithms will solve the problem exactly.

The TSP cannot be solved exactly using greedy methods, hence any greedy method is a heuristic. By definition, therefore, DP will always find a better (or, no worse) feasible solution than a greedy heuristic will, for any instance of the TSP.

Note, however, that DP is not the dominant approach for solving TSP. Many other algorithms exist that are much more efficient. Some of the original papers on TSP used DP, and it is often formulated as an illustrative example, but it is not the way TSPs are usually solved in practice

---

Conclusion: (Based on the observations):

Thus, implemented the travelling salesman problem using dynamic approach.

Outcome:

CO3: 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.