Experiment No. : 6

Title: Implement Travelling Salesman Problem using Dynamic approach

Batch: B-4 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 06

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

Algorithm of Travelling 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 a 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 a dynamic programming approach, the solution can be obtained in less 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.

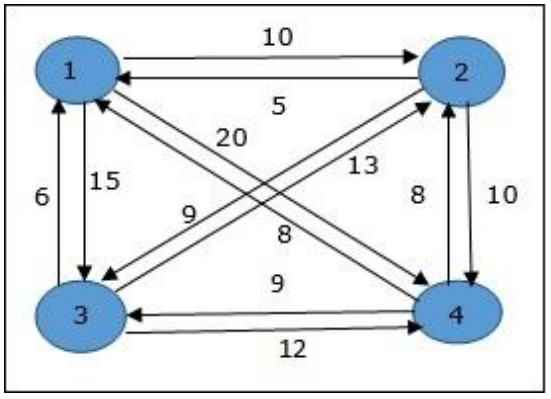
When |S| > 1, we define C(S, 1) = ∝ since the path cannot start and end at 1.

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.



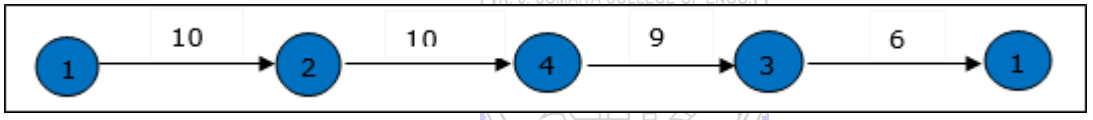
From the above graph, the following table is prepared.

	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 time complexity analysis of the Travelling Salesman Problem (TSP) when solved using the Dynamic Programming (DP) approach, specifically the Held-Karp algorithm, provides a clear example of how DP can offer exact solutions at the cost of high computational resources. Here's how the time complexity of the DP solution for TSP can be derived and understood. The DP approach to solving the TSP, often through the Held-Karp algorithm, involves considering each subset of the set of cities and determining the shortest path that visits each city in a subset exactly once and returns to the start. This method uses recursion and memoization to store and reuse the results of subproblems, thus avoiding the recomputation of the same subproblems.

The key to understanding the time complexity lies in two factors:

1. Subsets of Cities: For `$n$` cities, there are (2^n) possible subsets.

2. Transition Choices: For each subset, the algorithm considers transitioning to the next city. If the subset size is (k), there are up to ($k - 1$) choices for the next city (since one city is already visited).

Subproblem Identification: Each subproblem is defined by a subset of cities and the last city visited. Therefore, the total number of subproblems corresponds to the number of subsets of cities times the number of choices for the last city visited. There are (2^n) subsets and, for each subset, (n) choices for the last city, leading to ($n \times 2^n$) subproblems.

Computational Steps per Subproblem: For each subproblem, the algorithm calculates the minimum cost path by considering all possible previous cities. This involves (O(n)) operations to find the minimum among all potential previous stops.

Combining the number of subproblems with the computational steps required for each, we arrive at the overall time complexity for the DP approach to the TSP:

[O($n^2 \times 2^n$)]

This means that for each of the ($n \times 2^n$) subproblems, the algorithm performs (n) operations to find the minimum cost path, leading to a total of ($n^2 \times 2^n$) operations.

The derivation of the time complexity for the DP solution to the TSP highlights the computational intensity of this approach, explaining why it is impractical for large numbers of cities. Although it guarantees an optimal solution, the exponential growth in time and space requirements restricts its use to problems with a relatively small number of cities. This analysis underscores the trade-off between accuracy and computational feasibility in algorithm design, particularly for NP-hard problems like the TSP.

Program(s) of Travelling Salesman Problem:

```
#include <stdio.h>
#include <limits.h>
int dist[10][10];
int dp[1<<10][10];
int N;
int VISITED_ALL;
int tsp(int mask, int pos) {
   if (mask == VISITED_ALL) {
      return dist[pos][0];
   }
   if (dp[mask][pos] != -1) {
     return dp[mask][pos];
   }
   int ans = INT_MAX;
```

```
    for (int city = 0; city < N; city++) {
        if ((mask & (1 << city)) == 0) {
            int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
            ans = ans < newAns ? ans : newAns;
        }
    }
    return dp[mask][pos] = ans;
}
int main() {
    printf("Enter the number of cities: ");
    scanf("%d", &N);
    VISITED_ALL = (1<<N) - 1;
    printf("Enter the distances between the cities:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &dist[i][j]);
        }
    }
    for (int i = 0; i < (1<<N); i++) {
        for (int j = 0; j < N; j++) {
            dp[i][j] = -1;
        }
    }
    printf("The minimum cost of visiting all the cities: %d\n", tsp(1, 0));
    return 0;
}
```


Output(o) of Travelling Salesman Problem:

```
"C:\Users\chand\Downloads\I
Enter the number of cities: 4
Enter the distances between the cities:
0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0
The minimum cost of visiting all the cities: 35

Process returned 0 (0x0)   execution time : 35.247 s
Press any key to continue.
```

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

Ans: The Travelling Salesman Problem (TSP) is a classic algorithmic problem in the field of computer science and operations research. It involves finding the shortest possible route that visits a set of cities exactly once and returns to the original city. The challenge of the TSP lies in its computational complexity; as the number of cities increases, the number of possible routes increases exponentially, making it increasingly difficult to solve. There are various methods to approach the TSP, among which the Greedy Method and Dynamic Programming (DP) Method are two popular strategies. Each has its own set of characteristics, advantages, and disadvantages.

Greedy Method

The Greedy Method for solving the TSP is straightforward and intuitive. It builds up a solution piece by piece, always choosing the next piece that offers the most immediate benefit. In the context of the TSP, a greedy strategy might involve starting at an arbitrary city and, at each step, traveling to the nearest unvisited city until all cities are visited.

Characteristics:

- Simplicity:It is relatively easy to understand and implement.
- Speed:It can produce solutions quickly, which is beneficial for problems involving a large number of cities.
- Short-sighted:The method does not consider the future consequences of its choices, which can lead to suboptimal solutions.

Advantages:

- Fast and efficient for small to moderate-sized problems.
- Requires less computational resources compared to more complex algorithms.

Disadvantages:

- It does not guarantee an optimal solution. The route chosen by the greedy method may not be the shortest possible route.
- It is not efficient for solving large instances of the TSP due to its tendency to get stuck in local optima.

Dynamic Programming (DP) Method

Dynamic Programming is a more sophisticated approach that solves problems by breaking them down into simpler subproblems. It is used to avoid computing multiple times the same subproblem, thus significantly reducing the computational burden for certain types of problems. In the context of the TSP, the DP approach typically uses the principle of optimality to construct the shortest path.

Characteristics:

- Comprehensive:It considers all possible routes to ensure the selection of the shortest route.
- Memory Intensive:It stores the results of subproblems to avoid recalculating them, which can require a significant amount of memory.
- Optimal:It guarantees to find the shortest possible route if one exists.

Advantages:
- It always finds the optimal solution.
- It is more effective for problems where the greedy method fails to find the optimal solution.
Disadvantages:
- It is computationally intensive, especially for large numbers of cities, due to the exponential growth of possible solutions (factorial in nature).
- It requires significant memory storage for the subproblems.

Comparison
The key difference between the Greedy Method and Dynamic Programming for solving the TSP lies in their approach and efficiency. The Greedy Method is fast and straightforward but does not guarantee an optimal solution. It is suitable for quick approximations or when the problem size is small. On the other hand, Dynamic Programming, while computationally more demanding and memory-intensive, guarantees to find the optimal solution by exploring all possible combinations and remembering past results to avoid recalculations.

In summary, the choice between the Greedy Method and Dynamic Programming for the TSP depends on the specific requirements of the problem, including the number of cities to visit, the computational resources available, and whether an optimal solution is necessary.

Conclusion: (Based on the observations)
The experiment reaffirms the DP approach's effectiveness in solving the TSP for a limited number of vertices, providing an optimal solution within manageable computational limits. However, the significant increase in time and resources required for larger numbers of cities highlights the critical challenge of scalability. This experiment underscores the importance of choosing the right algorithm based on the size and requirements of the problem, balancing between the need for optimal solutions and the practical limitations of computational resources.

Outcome: 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.