Travelling sales person problem

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. Thus, maintaining a higher complexity.

However, instead of using brute-force, using the dynamic programming approach will obtain the solution in lesser time, though there is no polynomial time algorithm.

Travelling Salesman Dynamic Programming 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 ϵ� {1,2,3,...,n}** that includes **1**, and **j ϵ� 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 **𝐶C(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)=minC(S−{j},i)+d(i,j)whereiϵSandi≠j�(�,�)=����(�−{�},�)+�(�,�)�ℎ������� ����≠�

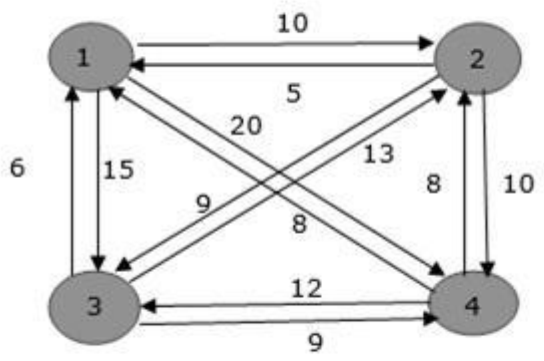
```
Algorithm: Traveling-Salesman-Problem
C ({1}, 1) = 0
for s = 2 to n do
  for all subsets S ϵ {1, 2, 3, … , n} of size s and containing 1
    C (S, 1) = ∞
  for all j ϵ S and j ≠ 1
    C (S, j) = min {C (S – {j}, i) + d(i, j) for i ϵ S and i ≠ j}
Return minj C ({1, 2, 3, …, n}, j) + d(j, i)
```

Analysis

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

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

S = ΦΦ

S = ΦΦ

Cost(2,Φ,1)=d(2,1)=5����(2,Φ,1)=�(2,1)=5

Cost(3,Φ,1)=d(3,1)=6����(3,Φ,1)=�(3,1)=6

Cost(4,Φ,1)=d(4,1)=8����(4,Φ,1)=�(4,1)=8

S = 1

Cost(i,s)=min{Cos(j,s−(j))+d[i,j]}����(�,�)=���{���(�,�−(�))+�[�,�]}

Cost(2,{3},1)=d[2,3]+Cost(3,Φ,1)=9+6=15����(2,{3},1)=�[2,3]+����(3,Φ,1)=9+6=15

Cost(2,{4},1)=d[2,4]+Cost(4,Φ,1)=10+8=18����(2,{4},1)=�[2,4]+����(4,Φ,1)=10+8=18

Cost(3,{2},1)=d[3,2]+Cost(2,Φ,1)=13+5=18����(3,{2},1)=�[3,2]+����(2,Φ,1)=13+5=18

Cost(3,{4},1)=d[3,4]+Cost(4,Φ,1)=12+8=20����(3,{4},1)=�[3,4]+����(4,Φ,1)=12+8=20

Cost(4,{3},1)=d[4,3]+Cost(3,Φ,1)=9+6=15����(4,{3},1)=�[4,3]+����(3,Φ,1)=9+6=15

Cost(4,{2},1)=d[4,2]+Cost(2,Φ,1)=8+5=13����(4,{2},1)=�[4,2]+����(2,Φ,1)=8+5=13

S = 2

Cost(2,{3,4},1)=min{d[2,3]+Cost(3,{4},1)=9+20=29d[2,4]+Cost(4,{3},1)=10+15=25=25����(2,{3,4},1)=���{�[2,3]+����(3,{4},1)=9+20=29�[2,4]+����(4,{3},1)=10+15=25=25

Cost(3,{2,4},1)=min{d[3,2]+Cost(2,{4},1)=13+18=31d[3,4]+Cost(4,{2},1)=12+13=25=25����(3,{2,4},1)=���{�[3,2]+����(2,{4},1)=13+18=31�[3,4]+����(4,{2},1)=12+13=25=25

Cost(4,{2,3},1)=min{d[4,2]+Cost(2,{3},1)=8+15=23d[4,3]+Cost(3,{2},1)=9+18=27=23����(4,{2,3},1)=���{�[4,2]+����(2,{3},1)=8+15=23�[4,3]+����(3,{2},1)=9+18=27=23

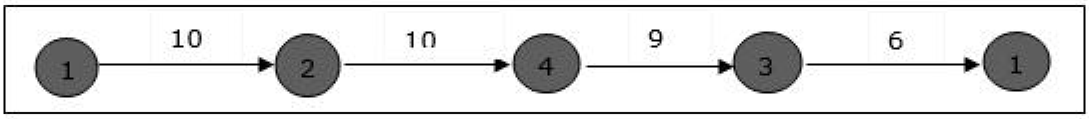
S = 3

Cost(1,{2,3,4},1)=min ⎛⎝⎫⎪⎪ d[1,2]+Cost(2,{3,4},1)=10+25=35d[1,3]+Cost(3,{2,4},1)=15+25=40d[1,4]+Cost(4,{2,3},1)=20+23=43=35

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, 2] but 2 and 4 is already selected. Therefore, we select d [4, 3] (two possible values are 15 for d [2, 3] and d [4, 3], but our last node of the path is 4). Select path 4 to 3 (cost is 9), then go to s = ϕ step. We get the minimum value for d [3, 1] (cost is 6).



```
#include <stdio.h>

#include <limits.h>

#define MAX 9999

int n = 4;

int distan[20][20] = {{0, 22, 26, 30},

                {30, 0, 45, 35},

                {25, 45, 0, 60},

                {30, 35, 40, 0}};

int DP[32][8];

int TSP(int mark, int position) {
    int completed_visit = (1 << n) - 1;
    if (mark == completed_visit) {
        return distan[position][0];
    }
    if (DP[mark][position] != -1) {
        return DP[mark][position];
    }
    int answer = MAX;
    for (int city = 0; city < n; city++) {
        if ((mark & (1 << city)) == 0) {
            int newAnswer = distan[position][city] + TSP(mark | (1 << city), city);
            answer = (answer < newAnswer) ? answer : newAnswer;
        }
    }
    return DP[mark][position] = answer;
}
int main() {
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            DP[i][j] = -1;
        }
    }
    printf("Minimum Distance Travelled -> %d\n", TSP(1, 0));
    return 0;
}
```

Output

```
Minimum Distance Travelled -> 122
```