Topics in Algorithms

Assignment

Dynamic Programming for Travelling Salesperson Problem

Submitted to
Dr . R Subashini

Submitted by
Shahin John JS
B160943CS

Travelling Salesperson Problem

Input:

A complete Graph (G) and distance between vertices (D).

Problem:

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Output:

A sequence of vertices that represents the optimal path for TSP.

Comments:

- Travelling salesperson is an NP-complete problem.
  This can be easily done by reducing Hamiltonian cycle problem to travelling salesperson problem since Hamiltonian cycle in np-complete TSP is also np-complete.
- Travelling salesperson is a permutation problem
- Permutation problems are harder to solve
  - n! different permutations of n objects
  - 2^n different subsets of n objects
  - $n! > 2^n$

Mathematical Formulation:

Find a permutation p of $\{1,2,..,n-1\}$,such that $d(0,p(1)) + d(p(1),p(2)) + \ldots + d(p(n-1),0)$ is minimized.

Problem Specifications:

- Given a directed graph G = (V,E) with edge cost c_{ij}.
- c_{ij} is defined such that $c_{ij} > 0$ for all i and j and $c_{ij} = \infty$ if $(i, j) \notin E$.
- $|V| = n$ and $n > 1$
- A tour of G is a directed cycle that includes every vertex in V , and no vertex occurs more than once except for the starting vertex
- Cost of a tour is the sum of the cost of edges on the tour
- Travelling salesperson problem is to find a tour of minimum cost

Greedy algorithm:

- Start with vertex v_1; call it v_i
- Visit the vertex v_j that is nearest to v_i, or can be reached from vi with least cost
- Repeat the above starting at vertex v_j (call it as new v_i) taking care never to visit a vertex already visited

Optimal substructure property:

A given problems has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its sub problems. Travelling salesperson problem exhibits optimal substructure property.

Recursive algorithm:

Let S be a subset of {1,2,…n} and i,j ∈ {1,2,…n}

  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.

    C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.

Running time of recursive algorithm is $O(n^n)$

  Removing an element from a set takes O(n) time complexity. So the recurrence relation is

  For non-leaf nodes

    T(n) = (n-1)T(n-1) + n-1.

  For leaf nodes

    T(1) = 1

Solving the recurrence relation will give

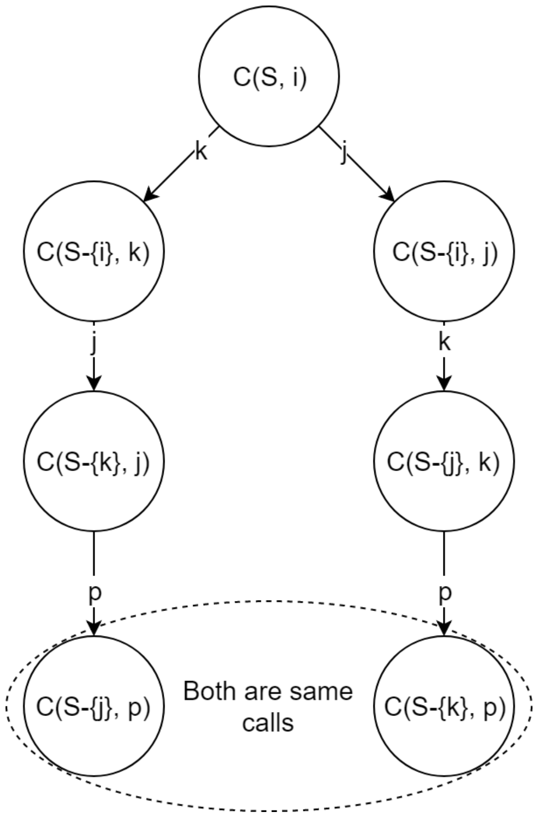
  $T(n) = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \ldots + \binom{n}{n} = O(n^n)$.

This is because travelling salesperson problem is a permutation problem.

There for the recursive algorithm takes $O(n^n)$ time complexity which is pretty high.

Using dynamic programming, we can achieve a far better time complexity.

Repeated sub problems:



Here the last calls have the same arguments, set S-{I,j,k}, and a city p. Instead of calling it two times, we can store the return value of it in memory and reuse that value when we encounter the same call. This is what we do in dynamic programing.

Dynamic programming algorithm:

Pseudocode:

function to find the minimum cost and optimal path

```
    DP_tsp(cost[][],n)
    {
        for every S ⊆ {2,3,…n}
            for j= 1 to n
                if(S == ∅)
                    g[S][j] = cost(1,j);
                minval = INT_MAX;
                for every x ∈ S
                    if(minval < cost(j,x)+D[S-{x}][x])
                        minval = cost(j,x)+D[S-{x}][x];
                        Sol[S][j] = x;
                g[S][j] = minval;
    }
```

function to print the Travelling Salesman Optimum path

```
    Solution(Sol[][],S,j)
    {
        if(S == ∅)
            print("1\n");
        else
            print(Sol[S][j]);
            Solution(Sol[][] , S-{Sol[S][j]} , Sol[S][j]);
    }
```

Regard the tour to be a simple path that starts and ends at vertex 1.Every tour consists of an edge (1, k) for some $k \in V - \{1\}$ and a path from vertex k to vertex 1.The path from vertex k to vertex 1 goes through each vertex in $V - \{1, k\}$ exactly once. If the tour is optimal, then the path from k to 1 must be a shortest k to 1 path going through all vertices in $V - \{1, k\}$. Let g(S, i) be the length of a shortest path starting at vertex i, going through all vertices in S, and terminating at vertex 1g(V − {1},1) is the length of an optimal salesperson tour.

- From the principal of optimality

  $g(V - \{1\},1) = \min_{(2 \le k \le n)} \{c_{1k} + g(V - \{1, k\},k)\}$ (1)

- Generalizing (for $i \in S$)

  $g(S,i) = \min_{(j \in S)} \{c_{ij} + g(S - \{j\},j)\}$ (2)

Equation 1 may be solved for $g(V - \{1\},1)$ if we know $g(V - \{1, k\},k)$ for all values of k. The g values may be obtained by using Equation 2

- $g(\emptyset,i) = c_i,1,\ 1 \le i \le n$
- We can use Equation 2 to obtain g(S,i) for all S of size 1
- Then we can obtain g(S,i) for S with $|S| = 2$
- When $|S| < n - 1$, the values of i and S for which g(S,i) is needed are such that $i \neq 1$, $1 \notin S$, and $i \notin S$.

Correctness:

Outer loop-invariant:

g[S][j] = minimum cost of travelling from starting city j to all cities in the set S terminating at 1.

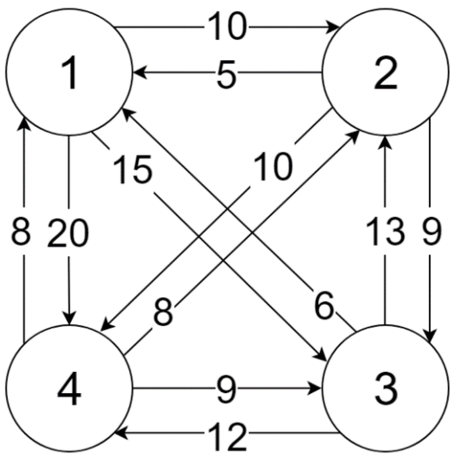
Proof for the correctness of the loop invariant:
Initially:
    S = ∅
    So, g[S][j] = cost of going from j to 1. (trivial)
Maintenance:
    Assume the loop invariant is true for some value of S, j then there are two cases
    Case 1 : g[S][j+1]
        We know the minimum value for any g[A][k] where A ⊂ S and 1≤k≤n.
        So for computing the minimum value of g[S][j+1] can easily be done by the Equation 1
    Case 2 : g[R][1] where R is some set such that S ⊂ R
        Same as above. We know the minimum value for any g[S][k] where S ⊂ R and 1≤k≤n. So for computing the minimum value of g[R][1] can easily be done by the Equation 2
    So in both the cases loop invariant maintained.

Termination:
    After exiting the loop, every entries in the g table will be the minimum cost for travelling.
This shows the correctness of the algorithm.

Illustration with an Example:

Solving travelling salesperson problem with dynamic programming – example:

- Consider the directed graph presented below



C_{ij}	1	2	3	4
1	**0**	**10**	**15**	**20**
2	**5**	**0**	**9**	**10**
3	**6**	**13**	**0**	**12**
4	**8**	**8**	**9**	**0**

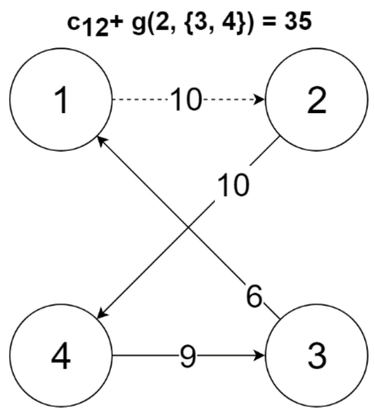
C_{ij} denotes the cost of going from i to j node

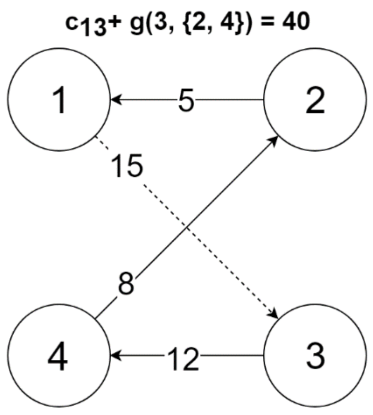
- Solving for 2, 3, 4
  - g(2, ∅) = c_{21} = 5
  - g(3, ∅) = c_{31} = 6
  - g(4, ∅) = c_{41} = 8
- Using Equation 2, we get

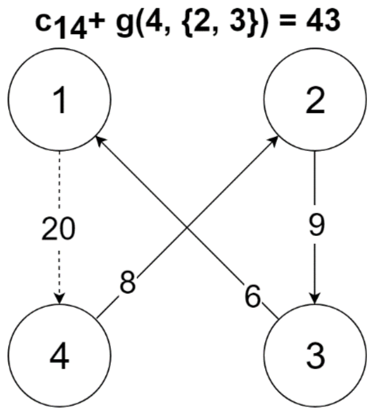
- $g(2, \{3\}) = c_{23} + g(3, \emptyset) = 15$
- $g(2, \{4\}) = c_{24} + g(4, \emptyset) = 18$
- $g(3, \{2\}) = c_{32} + g(2, \emptyset) = 18$
- $g(3, \{4\}) = c_{34} + g(4, \emptyset) = 20$
- $g(4, \{2\}) = c_{42} + g(2, \emptyset) = 13$
- $g(4, \{3\}) = c_{43} + g(3, \emptyset) = 15$

• Next, we compute $g(i, S)$ with $|S| = 2$, $i \neq 1$, $1 \notin S$, and $i \notin S$
  - $g(2, \{3, 4\}) = \min\{c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\})\} = 25$
  - $g(3, \{2, 4\}) = \min\{c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\})\} = 25$
  - $g(4, \{2, 3\}) = \min\{c_{42} + g(2, \{3\}), c_{43} + g(3, \{2\})\} = 23$

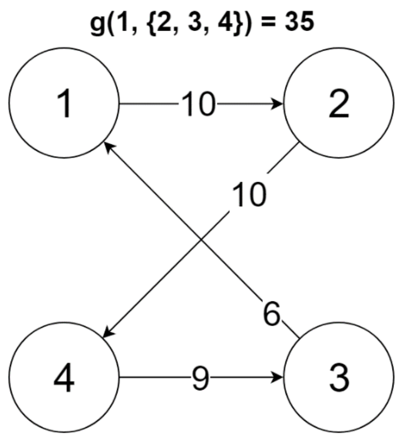
• Finally, from Equation 1, we obtain
  - $g(1, \{2, 3, 4\}) \quad = \min\{c_{12} + g(2, \{3, 4\}), c_{13} + g(3, \{2, 4\}), c_{14} + g(4, \{2, 3\})\}$







$= \min\{35, 40, 43\}$



$= 35$

• G table:

G(i,S)	1	2	3	4
{}	0	5	6	8
{2}	15	5	18	13
{3}	21	15	6	15
{2,3}	25	15	18	23
{4}	28	18	20	8
{2,4}	28	18	25	13
{3,4}	35	25	20	15
{2,3,4}	35	25	25	23

- Optimal tour:
    - Has cost 35
    - A tour of this length may be constructed if we retain with each g(S,i) the value of j that minimizes the right hand side of Equation 2
    - Let this value be called J(i, S)
    - Then, J(1, {2, 3, 4}) = 2
    - Thus the tour starts from 1 and goes to 2
    - The remaining tour may be obtained from g(2, {3, 4})
    - Now, J(2, {3, 4}) = 4
    - Thus the next edge is (2, 4)
    - The remaining tour is for g(4, {3})
    - J(4, {3}) = 3
    - The optimal tour is 1, 2, 4, 3, 1

Analysis of travelling salesperson:

- Let N be the number of g(S,i)s that have to be computed before Equation 1 may be used to compute g(1, V −{1})
- For each value of |S|, there are n − 1 choices of i
- The number of distinct sets S of size k not including 1 and i is $\binom{n-2}{k}$.
- Hence,

$$N = \sum_{k=0}^{n-2} \binom{n-2}{k} (n-k-1) = (n-1)2^{n-2}$$

- An algorithm that finds an optimal tour using Equations 1 and 2 will require $\theta(n^2 2^n)$ time as the computation of g(S,i) with |S| = k requires k − 1 comparisons when solving Equation 2
- Better than enumerating all $n!$ different tours to find the best one
- The most serious drawback of the dynamic programming solution is the space needed $\theta(n^2 2^n)$
    - This can be too large even for modest values of n.

Link to code in C/C++ and test cases:

https://drive.google.com/open?id=1CyiDDOV7X5qCvhtGZ52faDEji5nXNVhb

Implementation:

The implementation was carried out in c, c++.

Data structures used:

- linked list – sets are represented using linked list

```
struct SubSet
{
        int data;                       //city number
        struct SubSet *next;            //pointer to next city in the set
};
```

- C table : (2-d array)
  Each entry contains the cost of travelling from one city to another.
- g table : (2-d array)
  row (S)   - all subsets of {1,2,...n}

column (j) - {1,2,…n}

Each entry contains the total cost of the optimal route that cover S cities starting from jth city and terminating at 1.

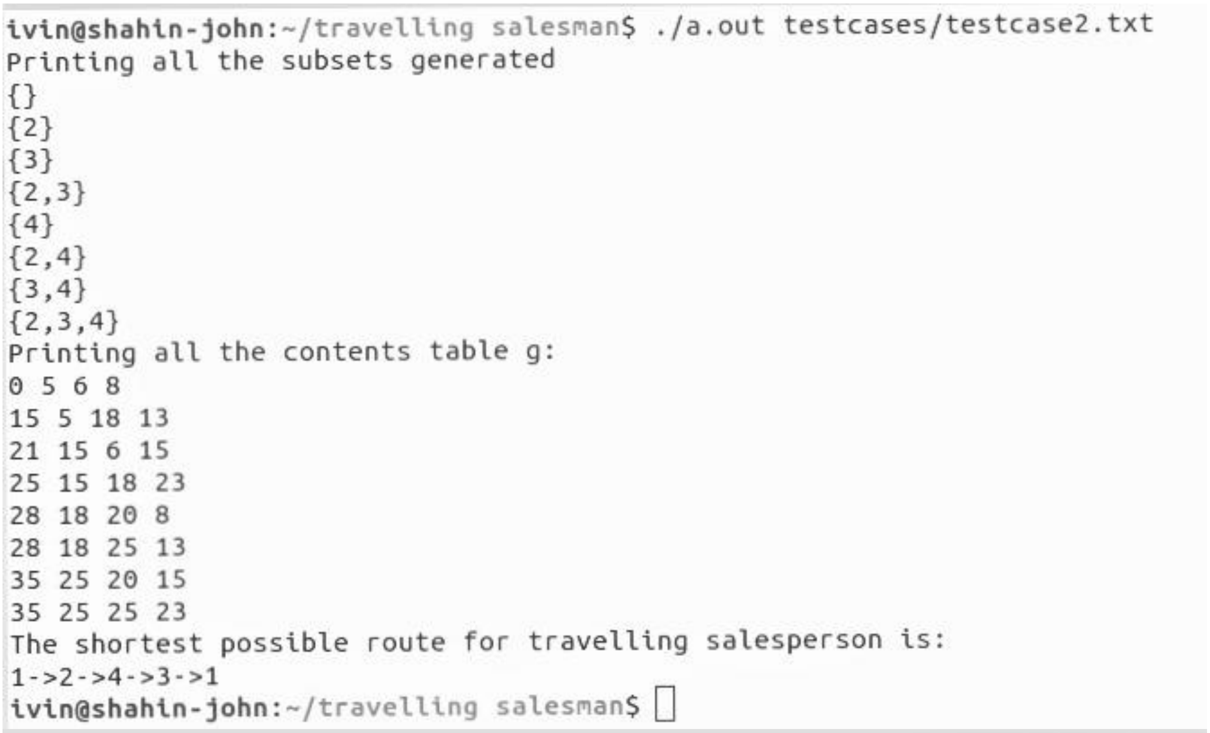
- Sol table: (2-d array)
  row (S) - all subsets of {1,2,…n}
  column (j) - {1,2,…n}

  Each entry contains the next city to traverse from the jth city to get an optimal path.

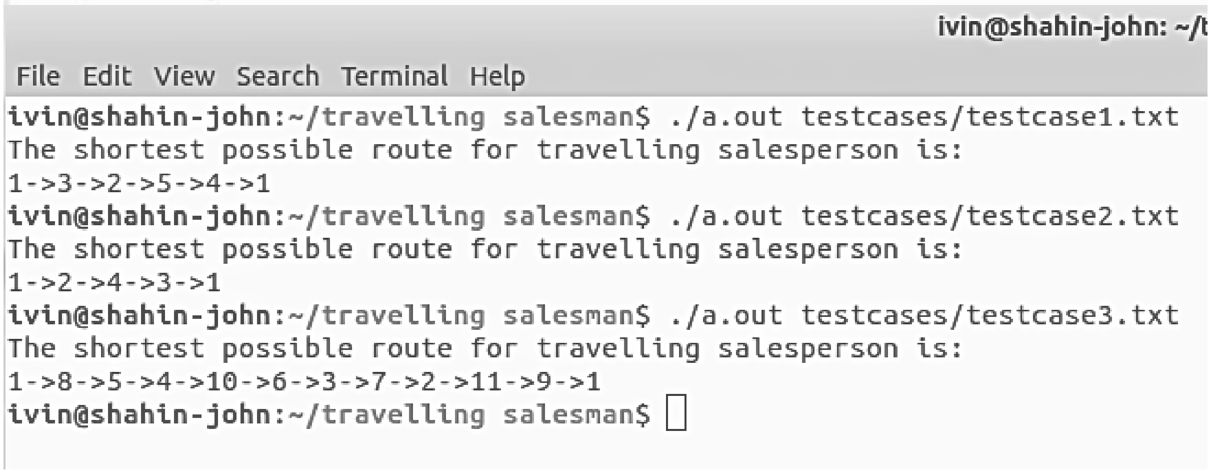
The major functions used in the implementation are:

- void DP_tsp(int **C,int n)
  Fills up the g table and Sol table.
- void Solution(int **Sol,int i,int j)
  Outputs the optimal route for travelling salesperson.
- struct SubSet **Create_SubSet(int n)
  Returns the pointer to the first subset of super set of n cities.

Screenshot:



```
ivin@shahin-john:~/travelling salesman$ ./a.out testcases/testcase2.txt
Printing all the subsets generated
{}
{2}
{3}
{2,3}
{4}
{2,4}
{3,4}
{2,3,4}
Printing all the contents table g:
0 5 6 8
15 5 18 13
21 15 6 15
25 15 18 23
28 18 20 8
28 18 25 13
35 25 20 15
35 25 25 23
The shortest possible route for travelling salesperson is:
1->2->4->3->1
ivin@shahin-john:~/travelling salesman$ 
```



```
ivin@shahin-john:~/travelling salesman$ ./a.out testcases/testcase1.txt
The shortest possible route for travelling salesperson is:
1->3->2->5->4->1
ivin@shahin-john:~/travelling salesman$ ./a.out testcases/testcase2.txt
The shortest possible route for travelling salesperson is:
1->2->4->3->1
ivin@shahin-john:~/travelling salesman$ ./a.out testcases/testcase3.txt
The shortest possible route for travelling salesperson is:
1->8->5->4->10->6->3->7->2->11->9->1
ivin@shahin-john:~/travelling salesman$ 
```

Conclusion:

Dynamic Programming Approach of solving np-complete problems was studied and implemented for travelling salesperson problem with a time complexity of $\theta(n^2 2^n)$.

Reference:

- ‘Dynamic programming treatment of the travelling salesman problem’, Richard Bellman, *Journal of Assoc. Computing Mach.* 9. 1962.
- ‘A dynamic programming approach to sequencing problems’, Michael Held and Richard M. Karp, *Journal for the Society for Industrial and Applied Mathematics* 1:10. 1962