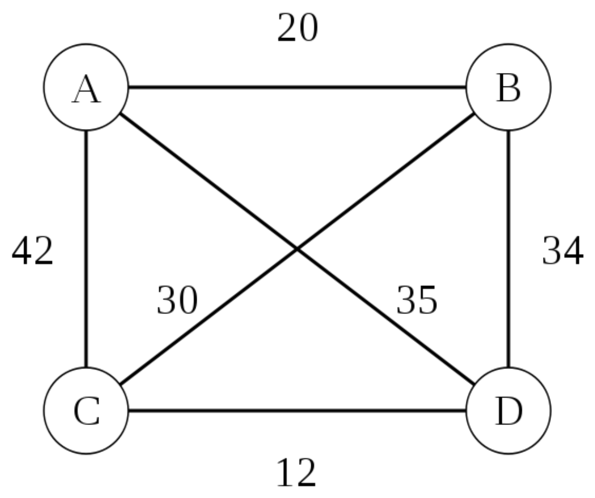
Traveling Salesman Problem Solution

Using Dynamic Programing Approach

Team (No. 22) Members	Sec
Ahmad Abdo Muhammad	1
Ahmad Ramadan Abbas	1
Anas Nabil Muhammad	1
Ahmad Samy Ahmad	1
Muhammad Abdel-Fttah Nasr	3

Problem Statement

A traveling salesman is getting ready for a big sales tour. Starting at his hometown, he will conduct a journey in which each of his target cities is visited exactly once before he returns home. Given the pairwise distances between cities, what is the best order in which to visit them, so as to minimize the overall distance traveled?



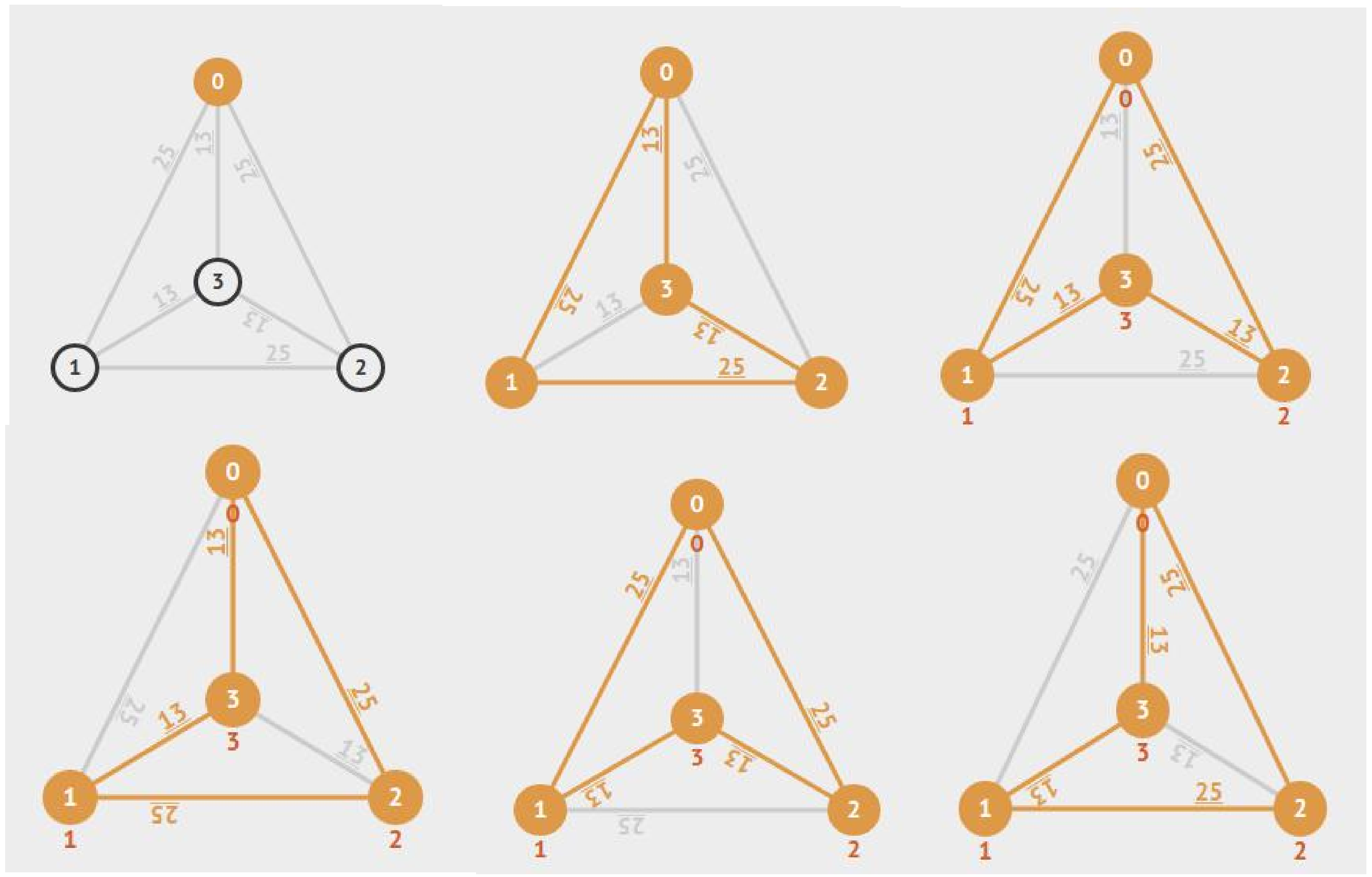
Naive Solution

Using Permatuation, which has O(n!) time complexity.

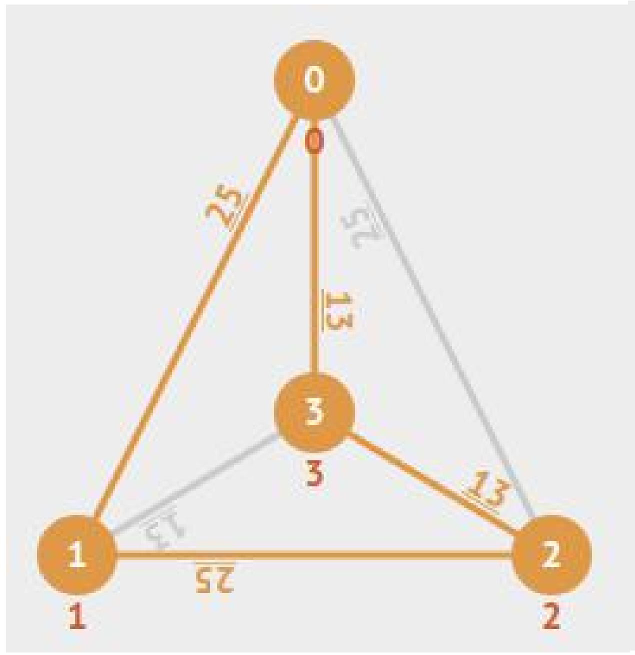
Steps:

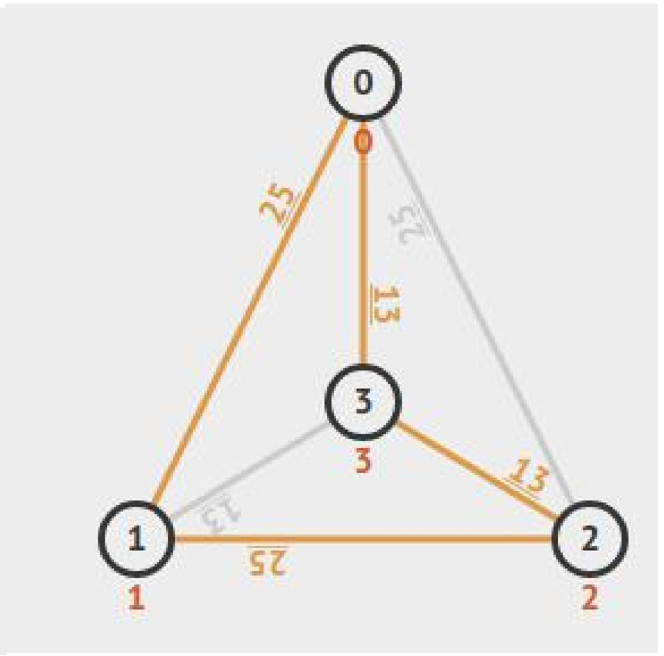
1- Consider city 1 as the starting and ending point.
2- Generate all (n-1)! Permutations of cities.
3- Calculate cost of every permutation and keep track of minimum cost permutation.
4-Return the permutation with minimum cost.

Naive Solution



Naive Solution





DP Solution

Let the given set of vertices be {1, 2, 3, 4,....n}.

Let us consider 1 as starting and ending point of output.

For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once.

Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values.

DP Solution

To calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.

We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

DP Solution

```
Recursion Algorithm :
//base case
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.
```

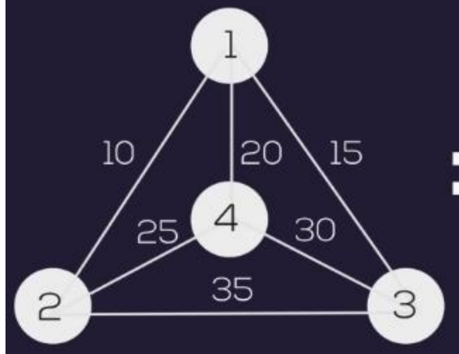
DP Solution

```
C(4, phi) = 20, C(3, phi) = 15, C(2, phi) = 10
C(2, {3}) = d(2, 3) + C(3, phi) = 35 + 15 = 50
C(2, {4}) = d(2, 4) + C(4, phi) = 45
C(3, {2}) = d(3, 2) + C(2, phi) = 45
C(3, {4}) = d(3, 4) + C(4, phi) = 50
C(4, {2}) = d(4, 2) + C(2, phi) = 35
C(4, {3}) = d(4, 3) + C(3, phi) = 45

C(2, {3, 4}) = min(d(2,3)+C(3, {4}),
        d(2,4) + C(4, {3}) = min(85, 80)
C(3, {2, 4}) = min (80 + 65) = 65
C(4, {2, 3}) = min(75, 75) = 75
```

S : subset of the graph which is not yet traversed

dist(i, 1) : distance from i to 1



$$= \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 10 & 0 & 35 & 25 \\ 3 & 15 & 35 & 0 & 30 \\ 4 & 20 & 25 & 30 & 0 \end{array}$$

```
C(1, {2, 3, 4}) = min(d(1,2) + C(2, {3, 4}), d(1, 3)+ C(3, {2, 4}), d(1, 4) + C(4, {2, 3})
= min (90, 80, 95) = 80
```

DP Solution Implementaion

In our program we use bitmask technique to donate cities had been visited by the salesman, also we use memoization to readuce space complexity of our algorithm.

As the cost of recursion could be very high in a graph of few nodes

Sapce complexity: (n * 2^n) , as (2^n) size of the tree generated for each recursion and (n) number of nodes that will generate these trees

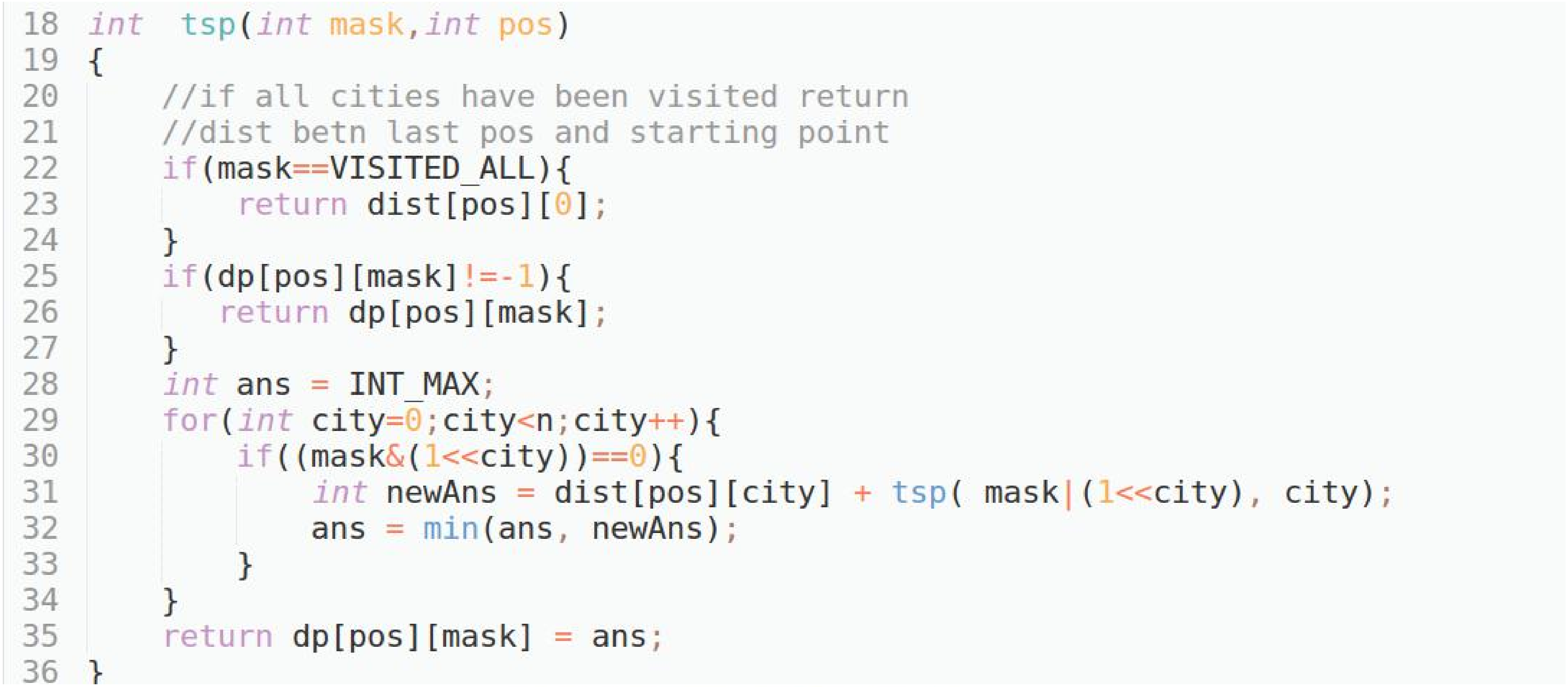
Time complexity: (n * n * 2^n) = (n^2 * 2^n)

DP Solution Implementaion

Recursion tree

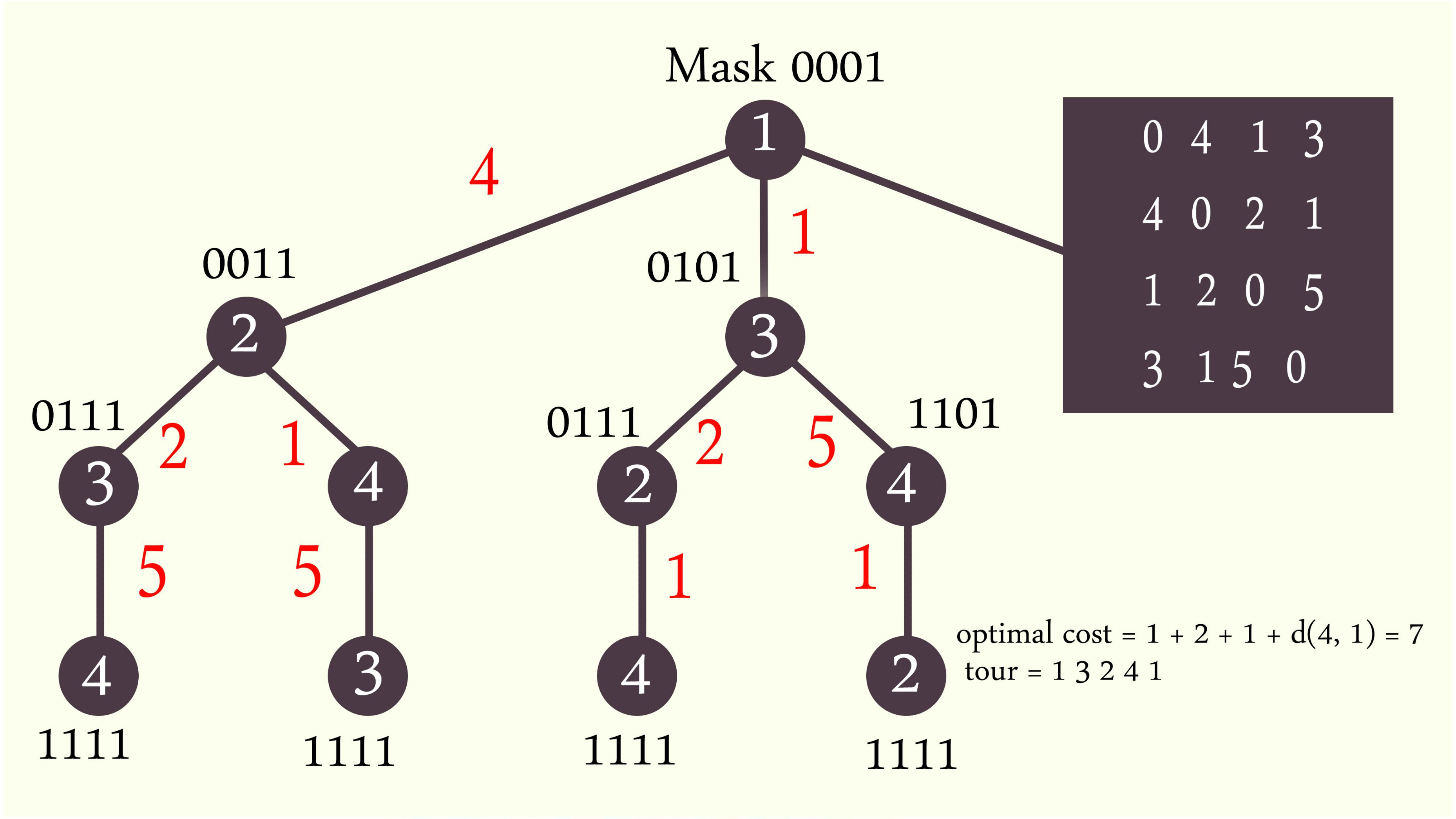
```
                             g(1,{2,3,4})
         /                        |                        \
   g(2,{3,4})                 g(3,{2,4})              g(4,{2,3})
   /        \                 /        \               /       \
g(3,{4})   g(4,{3})       g(2,{4})   g(4,{2})      g(2,{3})   g(3,{2})
  |            |              |          |            |           |
g(4,phi)   g(3,phi)       g(4,phi)    g(2,phi)      g(3,phi)   g(2,phi)
```

C Implementation



```c
int  tsp(int mask,int pos)
{
    //if all cities have been visited return
    //dist betn last pos and starting point
    if(mask==VISITED_ALL){
        return dist[pos][0];
    }
    if(dp[pos][mask]!=-1){
       return dp[pos][mask];
    }
    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 = min(ans, newAns);
        }
    }
    return dp[pos][mask] = ans;
}
```

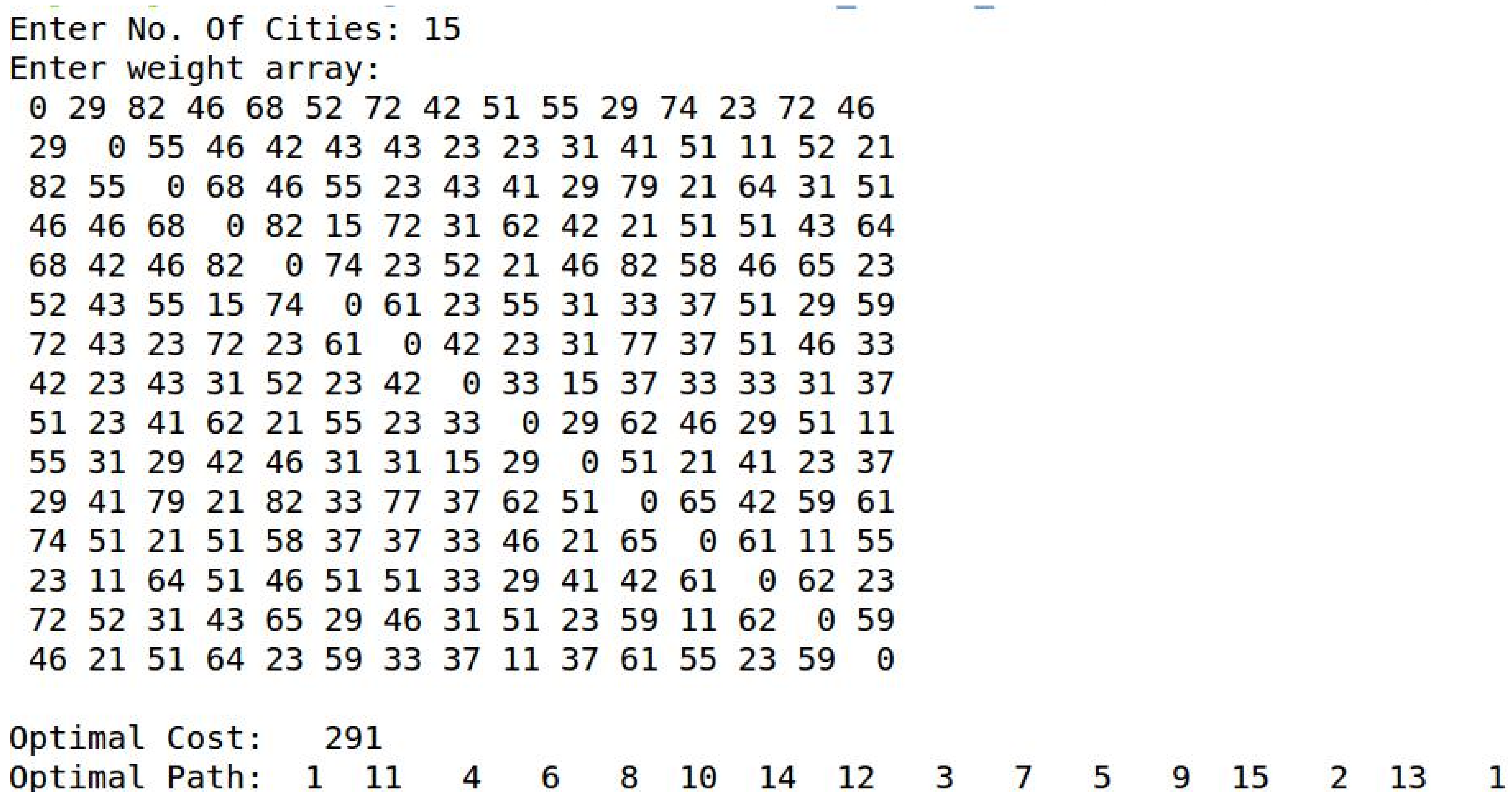
Example



0	4	1	3
4	0	2	1
1	2	0	5
3	1	5	0

optimal cost = 1 + 2 + 1 + d(4, 1) = 7
tour = 1 3 2 4 1

Screen Shot Of Running Program



```
Enter No. Of Cities: 15
Enter weight array:
  0 29 82 46 68 52 72 42 51 55 29 74 23 72 46
 29  0 55 46 42 43 43 23 23 31 41 51 11 52 21
 82 55  0 68 46 55 23 43 41 29 79 21 64 31 51
 46 46 68  0 82 15 72 31 62 42 21 51 51 43 64
 68 42 46 82  0 74 23 52 21 46 82 58 46 65 23
 52 43 55 15 74  0 61 23 55 31 33 37 51 29 59
 72 43 23 72 23 61  0 42 23 31 77 37 51 46 33
 42 23 43 31 52 23 42  0 33 15 37 33 33 31 37
 51 23 41 62 21 55 23 33  0 29 62 46 29 51 11
 55 31 29 42 46 31 31 15 29  0 51 21 41 23 37
 29 41 79 21 82 33 77 37 62 51  0 65 42 59 61
 74 51 21 51 58 37 37 33 46 21 65  0 61 11 55
 23 11 64 51 46 51 51 33 29 41 42 61  0 62 23
 72 52 31 43 65 29 46 31 51 23 59 11 62  0 59
 46 21 51 64 23 59 33 37 11 37 61 55 23 59  0

Optimal Cost:   291
Optimal Path:  1  11   4   6   8  10  14  12   3   7   5   9  15   2  13   1
```

Code Link: https://ideone.com/WBN4kl

Dataset Link: https://people.sc.fsu.edu/~jburkardt/datasets/tsp/p01.tsp