Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

60009210033

JHANVI PAREKH

CSE(Data Science)

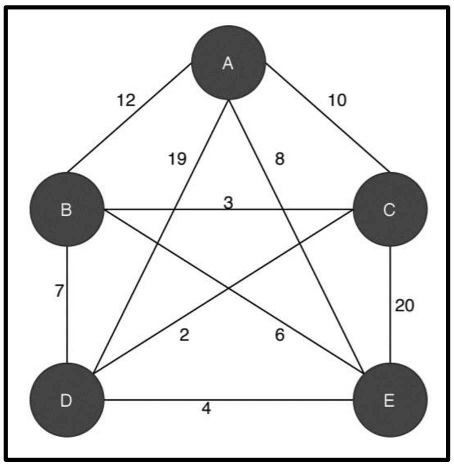
Experiment 7 (Dynamic Programming)

Aim: Implementation of Travelling salesperson problem using dynamic programming.

Theory:

In the TSP, given a set of cities and the distance between each pair of cities, a salesman needs to choose the shortest path to visit every city exactly once and return to the city from where he started.

Let's take an example to understand the TSP in more detail:



Here, the nodes represent cities, and the values in each edge denote the distance between one city to another. Let's assume a salesman starts the journey from the city A. According to TSP, the salesman needs to travel all the cities exactly once and get back to the city A by choosing the shortest path. Here the shortest path means the sum of the distance between each city travelled by the salesman, and it should be less than any other path.

With only 5 cities, the problem already looks quite complex. As the graph in the example is a complete graph, from each city, the salesman can reach any other cities in the graph. From each city, the salesman needs to choose a route so that he doesn't have to revisit any city, and the total distance travelled should be minimum.

Algorithm:

Step 1: Let d[i, j] indicates the distance between cities i and j. Function C[x, V – { x }]is the cost of the path starting from city x. V is the set of cities/vertices in given graph. The aim of TSP is to minimize the cost function.

Step 2: Assume that graph contains n vertices V1, V2, ..., Vn. TSP finds a path covering all vertices exactly once, and the same time it tries to minimize the overall traveling distance.

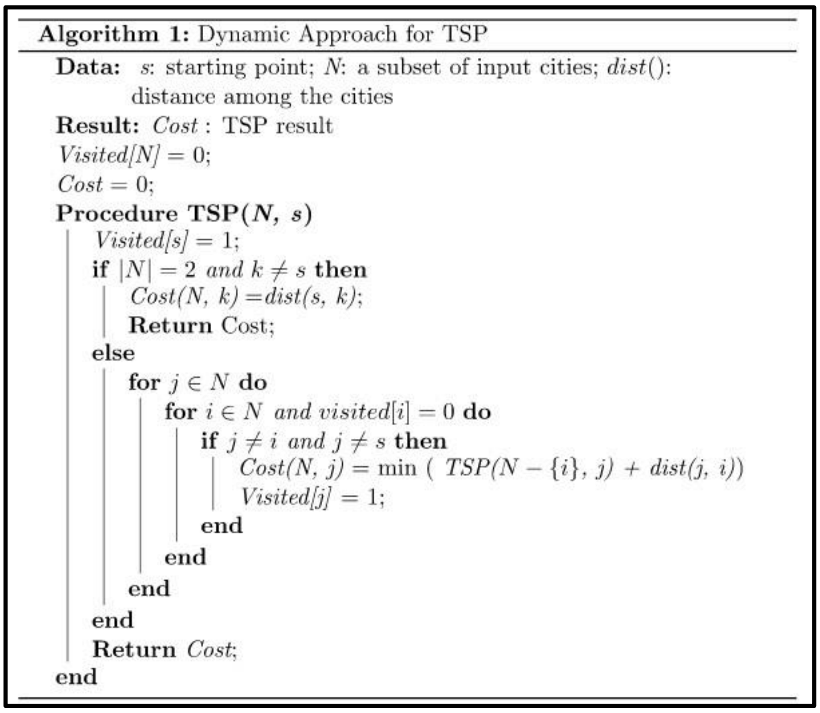
Department of Computer Science and Engineering (Data Science)

Step 3: Mathematical formula to find minimum distance is stated below:

C(i, V) = min { d[i, j] + C(j, V – { j }) }, j ∈ V and i ∉ V.

TSP problem possesses the principle of optimality, i.e. for d[V1, Vn] to be minimum, any intermediate path (Vi, Vj) must be minimum.

Pseudocode:



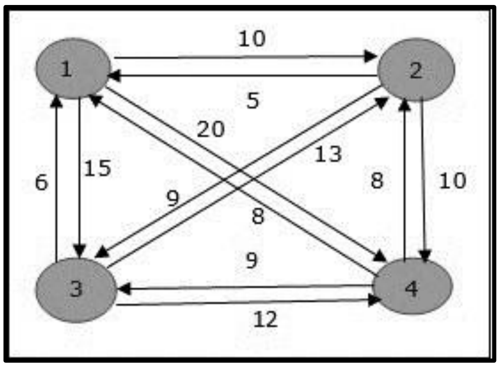
```
Algorithm 1: Dynamic Approach for TSP
  Data: $s$: starting point; $N$: a subset of input cities; $dist()$:
        distance among the cities
  Result: $Cost$ : TSP result
  $Visited[N] = 0$;
  $Cost = 0$;
  Procedure TSP($N$, $s$)
      $Visited[s] = 1$;
      if $|N| = 2$ and $k \neq s$ then
          $Cost(N, k) = dist(s, k)$;
          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
```

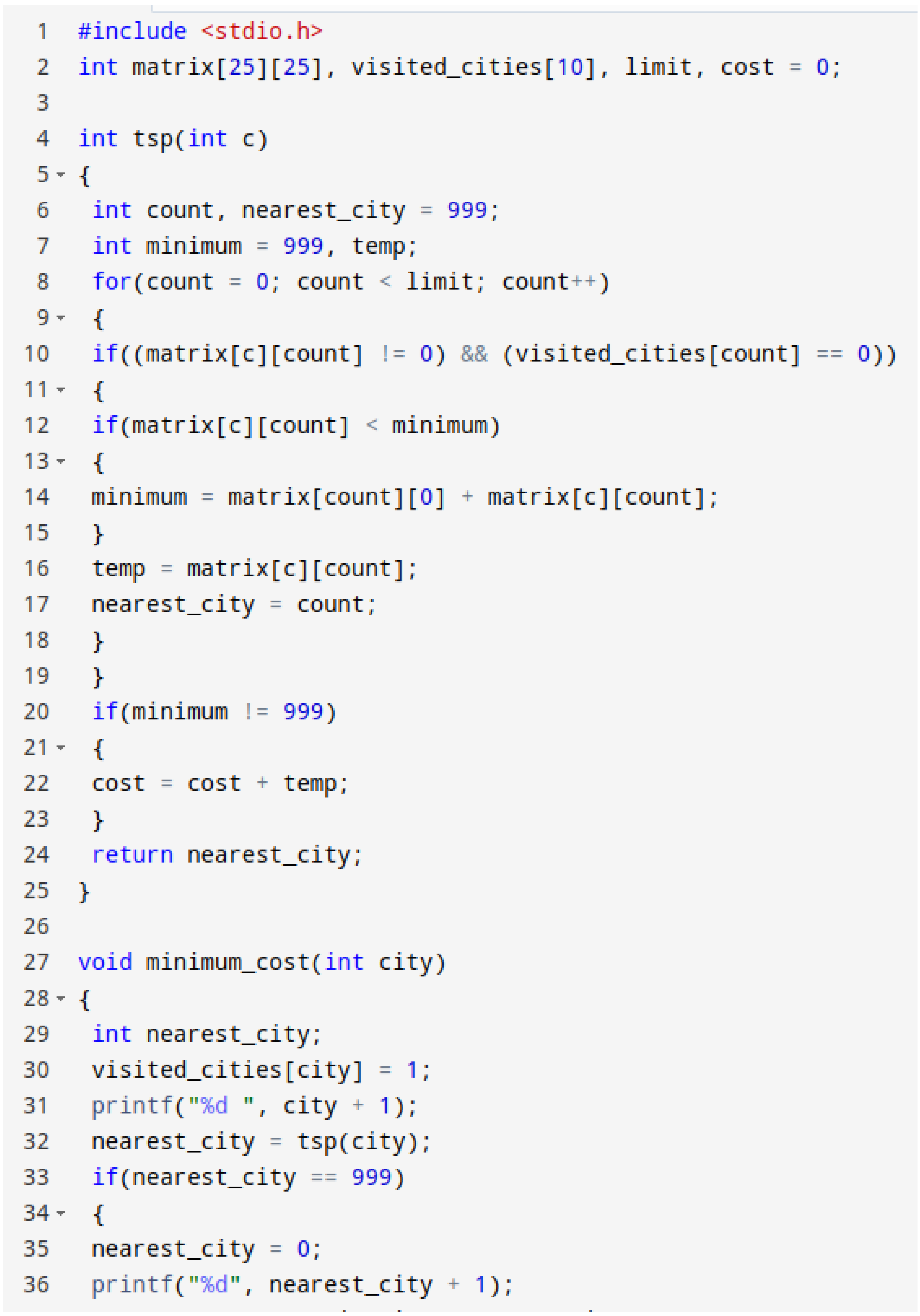
Complexity:

Time Complexity: O(n^2 * 2^n)

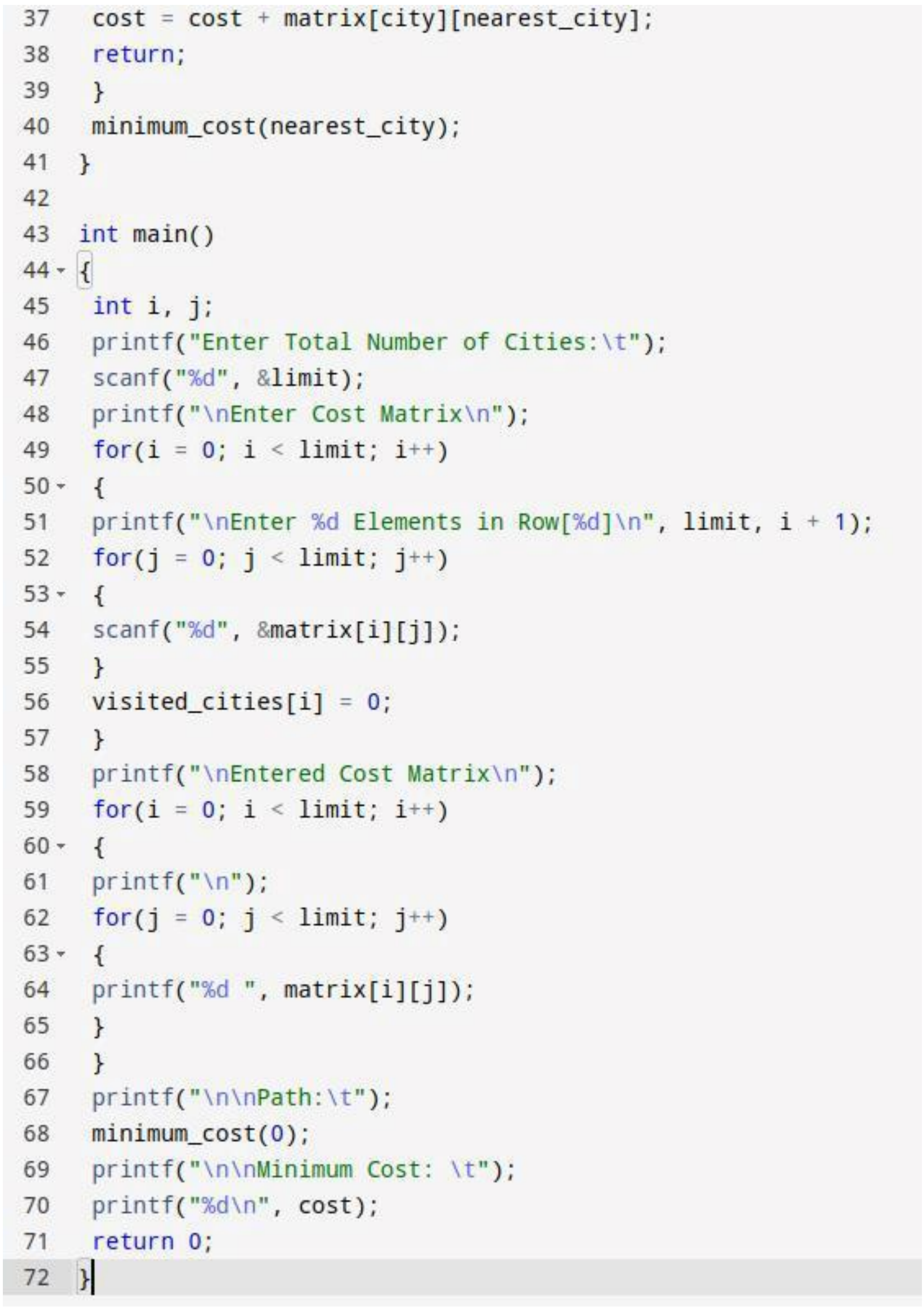
Lab Assignment to Complete:

A traveler 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?

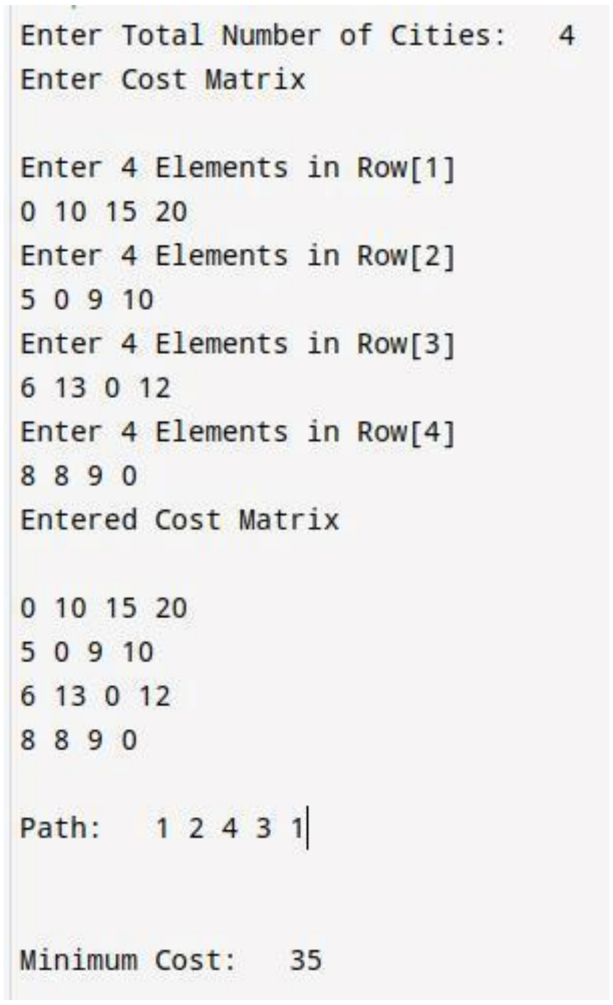

1
2
3
4
10
5
20
13
6
15
9
8
8
10
9
12



```c
#include <stdio.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;

int tsp(int c)
{
 int count, nearest_city = 999;
 int minimum = 999, temp;
 for(count = 0; count < limit; count++)
 {
 if((matrix[c][count] != 0) && (visited_cities[count] == 0))
 {
 if(matrix[c][count] < minimum)
 {
 minimum = matrix[count][0] + matrix[c][count];
 }
 temp = matrix[c][count];
 nearest_city = count;
 }
 }
 if(minimum != 999)
 {
 cost = cost + temp;
 }
 return nearest_city;
}

void minimum_cost(int city)
{
 int nearest_city;
 visited_cities[city] = 1;
 printf("%d ", city + 1);
 nearest_city = tsp(city);
 if(nearest_city == 999)
 {
 nearest_city = 0;
 printf("%d", nearest_city + 1);
```



```c
 cost = cost + matrix[city][nearest_city];
 return;
 }
 minimum_cost(nearest_city);
}

int main()
{
 int i, j;
 printf("Enter Total Number of Cities:\t");
 scanf("%d", &limit);
 printf("\nEnter Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
 printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
 for(j = 0; j < limit; j++)
 {
 scanf("%d", &matrix[i][j]);
 }
 visited_cities[i] = 0;
 }
 printf("\nEntered Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
 printf("\n");
 for(j = 0; j < limit; j++)
 {
 printf("%d ", matrix[i][j]);
 }
 }
 printf("\n\nPath:\t");
 minimum_cost(0);
 printf("\n\nMinimum Cost: \t");
 printf("%d\n", cost);
 return 0;
}
```

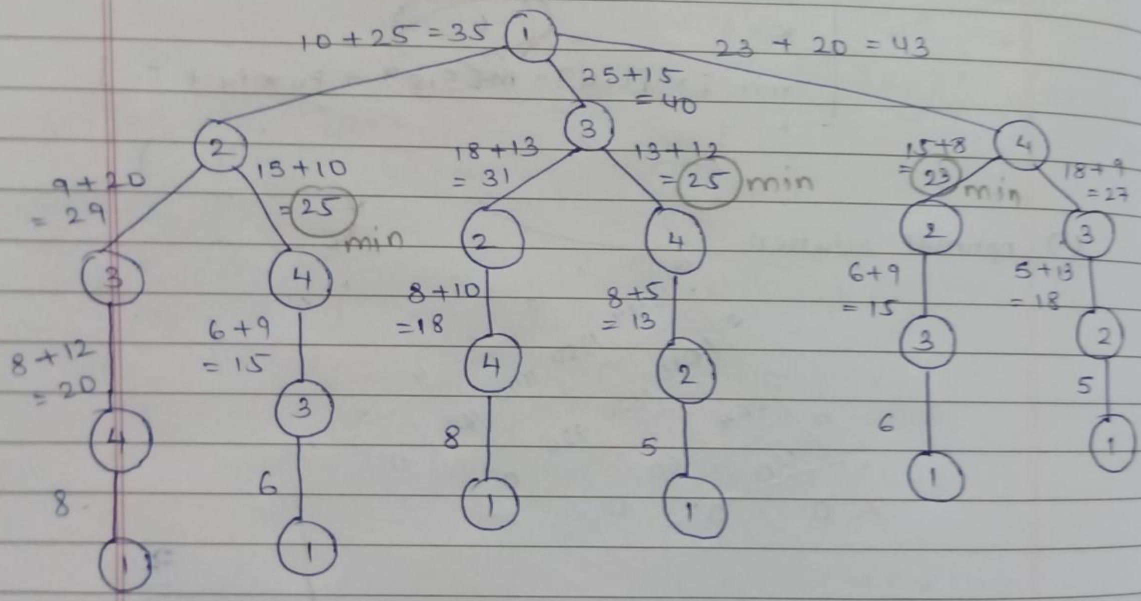


```
Enter Total Number of Cities:   4
Enter Cost Matrix

Enter 4 Elements in Row[1]
0 10 15 20
Enter 4 Elements in Row[2]
5 0 9 10
Enter 4 Elements in Row[3]
6 13 0 12
Enter 4 Elements in Row[4]
8 8 9 0
Entered Cost Matrix

0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0

Path:   1 2 4 3 1


Minimum Cost:   35
```

WRITTEN:

$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 9 & 10 \\ 3 & 6 & 13 & 0 & 12 \\ 4 & 8 & 8 & 9 & 0 \end{array}$$

Recursive tree



Final path: $1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 3 \longrightarrow 1$

Recursive formulae :

$$c(i, s) = \min_{j \in s} \{ w(i,j) + c(j, (s-j)) \}$$

PAGE :
DATE : / /

step 1. : $c(2,1) = 5$

$c(3,1) = 6$

$c(4,1) = 8$

$c(1,1) = 0$

step 2. $c(\overset{i}{2},\overset{j}{3}) = w(2,3) + c(3,\{3-3\}) = 15$

$c(2,4) = \min\{ w(2,4) + c(4,\{4-4\}) = 18$

$c(3,2) = \min\{ w(3,2) + c(2,\{2-2\}) = 18$

$c(3,4) = \min\{ w(3,4) + c(4,\{4-4\}) = 20$

$c(4,2) = \min\{ w(4,2) + c(2,\{2-2\}) = 13$

$c(4,3) = \min\{ w(4,3) + c(3,\{3-3\}) = 15$

step 3. $c(2,\{3,4\}) = \min \begin{cases} w(2,3) + c(3,4) \\ w(2,4) + c(4,3) \end{cases}$

$= \min \begin{cases} 9+20 = 29 \\ 10+15 = \underline{\underline{25}} \end{cases}$

$c(3,\{2,4\}) = \min \begin{cases} w(3,2) + c(2,4) \\ w(3,4) + c(4,2) \end{cases}$

$= \min \begin{cases} 13+18 = 31 \\ 12+13 = 25 /\!/ \end{cases}$

$c(4,\{2,3\}) = \min \begin{cases} w(4,2) + c(2,3) \\ w(4,3) + c(3,2) \end{cases}$

$= \min \begin{cases} 8+15 = 23 /\!/ \\ 9+18 = 27 \end{cases}$

$c(1,\{2,3,4\}) = \min \begin{cases} w(1,2) + c(2,\{3,4\}) = 25+10 = 35 \\ w(1,3) + c(3,\{2,4\}) = 15+25 = 40 \\ w(1,4) + c(4,\{2,3\}) = 20+23 = 43 \end{cases}$

$c(1,\{2,3,4\}) = 35$

Final path : $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$