**Team**

|  |  |  |
| --- | --- | --- |
| Name | ID | Section |
| Teef Khadeer AL-zulfi.  Alanoud Saleh AL-akhdar. | 3950489  3951671 | C5B |
|  |  |  |

Ramadan groceries

# Algorithm findMinCost(**int**[][] cost)

# **1**- Input Adjacency Matrix : int M ← cost.length , int N ← cost.length

# **2- int**[][] T ← **new** **int**[M][N]

# **3** -fill the matrix in a bottom-up manner using Nested loop: for i ←0 to i smallest than M,i+1

# 4 - for j ←0 to j smallest than N,j+1

# 5 - T[i][j] ← cost[i][j]

# 6 - fill the first row: if i equal 0 and j greater than 0 then

# 7 - T[0][j] ← T[0][j] + T[0][j - 1].

# 8 -fill the first column : else if j equal 0 and i greater than 0 then

# 9 - T[i][0] ← T[i][0] + T[i - 1][0].

# 10 - fill the rest with the matrix : else if j greater than 0 and i greater than 0 then T[i][j] ← T[i][j] + Integer.*min*(T[i - 1][j], T[i][j - 1]) .

# 11 - last cell of T[][] stores the minimum cost to reach destination cell: **return** T[M - 1][N - 1].

# Algorithm Analysis

## Space Complexity O(M\*N)

## " the length of the input taken by memory algorithm to run ".

## Time Complexity

## O() "Nested loop"

# Work Distribution

# TEEF :

1- M × N matrix

**int** M = cost.length;

**int** N = cost.length;

2- T[i][j] maintains the minimum cost to reach cell (i, j)

from cell (0, 0)

**int**[][] T = **new** **int**[M][N];

3- fill the matrix in a bottom-up manner

**for** (**int** i = 0; i < M; i++)

{

**for** (**int** j = 0; j < N; j++)

{

T[i][j] = cost[i][j];

4- fill the first row (there is only one way to reach any cell

in the first row from its adjacent left cell)

**if** (i == 0 && j > 0) {

T[0][j] += T[0][j - 1];

}

# Alanoud:

5-fill the first column (there is only one way to reach any cell

in the first column from its adjacent top cell)

**else** **if** (j == 0 && i > 0) {

T[i][0] += T[i - 1][0];

}

6- fill the rest with the matrix (there are two ways to reach any cell in the rest of the matrix, from its adjacent left cell or adjacent top cell)

**if** (i > 0 && j > 0) {

T[i][j] += Integer.*min*(T[i - 1][j], T[i][j - 1]);

}

}

}

7- last cell of T[][] stores the minimum cost to reach destination cell (M-1, N-1) from source cell (0, 0) **return** T[M - 1][N - 1];}