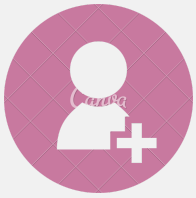
RAMADAN GROCERIES

ID: 3951671 Name : Alanoud Alakder

ID : 3950489 Name: Teef Alzulfi

Section: C5B

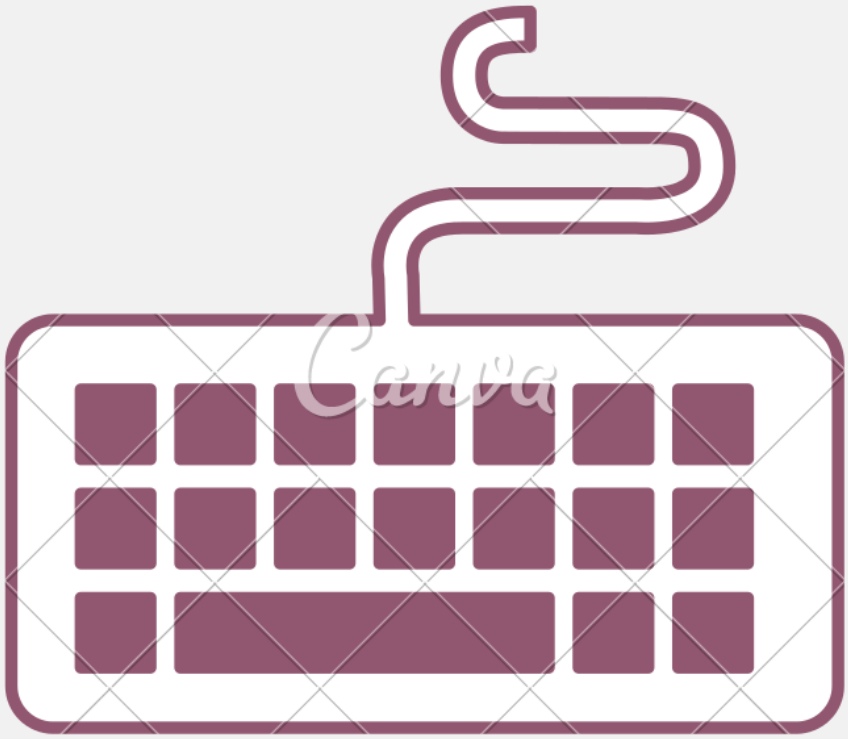
Method Code



```
public static int findMinCost(int[][] cost) {

int M = cost.length;
int N = cost.length;

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

for (int i = 0; i < M; i++) {
for (int j = 0; j < N; j++) {
T[i][j] = cost[i][j];

if (i == 0 && j > 0) {
T[0][j] += T[0][j - 1];
}

else if (j == 0 && i > 0) {
T[i][0] += T[i - 1][0];
}

if (i > 0 && j > 0) {
T[i][j] += Integer.min(T[i - 1][j], T[i][j -
1]);
}
}
}

return T[M - 1][N - 1];}
```

Main Code

```
public static void main(String args[]) {
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter the dimensions of the board:");
  int r = sc.nextInt();
  int c = sc.nextInt();
  int[][] cost = new int[r][c];
  System.out.println("Enter the values of the  Secures:\n");
  for (int i = 0; i < r; i++) {
   for (int j = 0; j < c; j++) {
    cost[i][j] = sc.nextInt();
   }
  }
System.out.print("The minimum total cost is " +
findMinCost(cost));
 }
}
```



ALGORITHM

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)

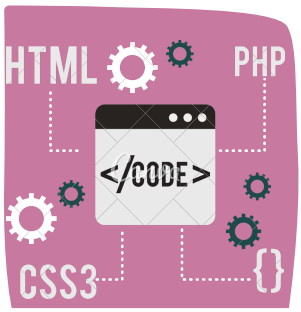
" the length of the input taken by memory algorithm to run ".

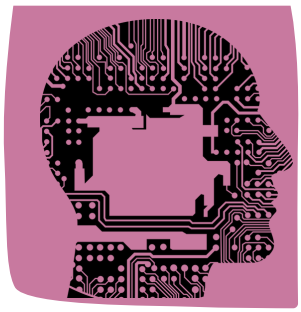
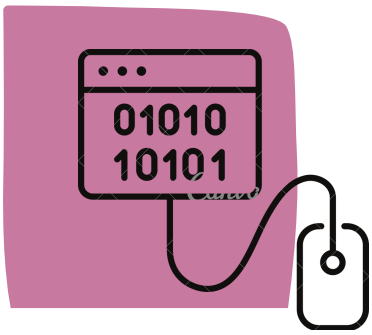
Time Complexity :

"Nested loop"

O(N^2)

WORK Distribution



TEEF :

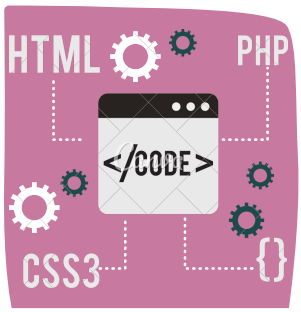
1- M × N matrix
int M = cost.length;
int N = cost.length;

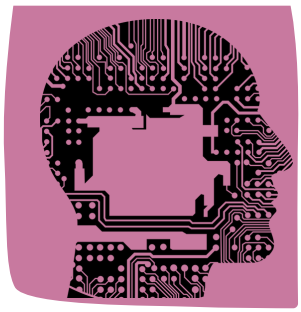
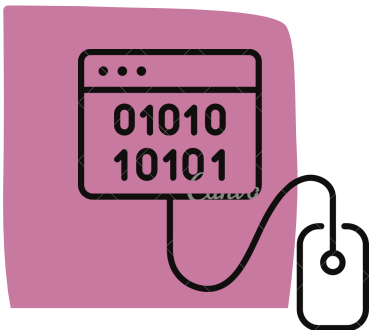
2- T[i][j] maintains the minimum cost to reach cell (i, j) from cell (0, 0)
int[][] T = new int[M][N];

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];
}

WORK Distribution



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];}
```

RUN:-

```
                }
            }
        }

        return T[M - 1][N - 1];
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the dimensions of the board:");
        int r = sc.nextInt();
        int c = sc.nextInt();
        int[][] cost = new int[r][c];
        System.out.println("Enter the values of the  Secures:\n");
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                cost[i][j] = sc.nextInt();
            }
        }

        System.out.print("The minimum total cost is " + findMinCost(cost));
    }
```

Console

<terminated> Graph [Java Application] C:\Users\Quevr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15

```
Enter the dimensions of the board:
3 3
Enter the values of the  Secures:

84 71 90
68 35 98
41 89 19
The minimum total cost is 295
```