Signup and get free access to 100+ Tutorials and Practice Problems   Start Now

Algorithms

🛈 Solve any problem to achieve a rank

View Leaderboard

Topics: 2 Dimensional ▼

2 Dimensional

TUTORIAL   **PROBLEMS**

1. Introduction

There are many problems in online coding contests which involve finding a minimum-cost path in a grid, finding the number of ways to reach a particular position from a given starting point in a 2-D grid and so on. This post attempts to look at the dynamic programming approach to solve those problems. The problems which will be discussed here are :

1. Finding the Minimum Cost Path in a Grid when a Cost Matrix is given.
2. Finding the number of ways to reach from a starting position to an ending position travelling in specified directions only.
3. Finding the number of ways to reach a particular position in a grid from a starting position (given some cells which are blocked)
4. Edit Distance

1. Finding Minimum-Cost Path in a 2-D Matrix

Problem Statement : Given a cost matrix Cost[][] where Cost[i][j] denotes the Cost of visiting cell with coordinates (i,j), find a min-cost path to reach a cell (x,y) from cell (0,0) under the condition that you can only travel one step right or one step down. (We assume that all costs are positive integers)

Solution : It is very easy to note that if you reach a position (i,j) in the grid, you must have come from one cell higher, i.e. (i-1,j) or from one cell to your left , i.e. (i,j-1). This means that the cost of visiting cell (i,j) will come from the following recurrence relation:

```
MinCost(i,j) = min(MinCost(i-1,j),MinCost(i,j-1)) + Cost[i][j]
```

The above statement means that to reach cell (i,j) wit minimum cost, first reach either cell(i-1,j) or cell (i,j-1) in as minimum cost as possible. From there, jump to cell (i,j). This brings us to the two important conditions which need to be satisfied for a dynamic programming problem:

Optimal Sub-structure:- Optimal solution to a problem involves optimal solutions to sub-problems.

Overlapping Sub-problems:- Subproblems once computed can be stored in a table for further use. This saves the time needed to compute the same sub-problems again and again.

(You can google the above two terms for more details)

The problem of finding the min-Cost Path is now almost solved. We now compute the values of the base cases: the topmost row and the leftmost column. For the topmost row, a cell can be reached only from the cell on the left of it. Assuming zero-based index,

```
MinCost(0,j) = MinCost(0,j-1) + Cost[0][j]
```

i.e. cost of reaching cell (0,j) = Cost of reaching cell (0,j-1) + Cost of visiting cell (0,j) Similarly,

```
MinCost(i,0) = MinCost(i-1,0) + Cost[i][0]
```

i.e. cost of reaching cell (i,0) = Cost of reaching cell (i-1,0) + Cost of visiting cell (i,0)

Other values can be computed from them. See the code below for more understanding.

```cpp
#include <bits/stdc++.h>
    using namespace std;
    #define F(i,a,b) for(int i = (int)(a); i <= (int)(b); i++)
    #define RF(i,a,b) for(int i = (int)(a); i >= (int)(b); i--)
    int main()
    {
        int X,Y; //X:number of rows, Y: number of columns
        X = Y = 10; //assuming 10X10 matrix
        int Cost[X][Y];

        F(i,0,X-1)
        {
            F(j,0,Y-1)
            {
                //Take input the cost of visiting cell (i,j)
                cin>>Cost[i][j];
            }
        }
```

```
        int MinCost[X][Y]; //declare the minCost matrix

        MinCost[0][0] = Cost[0][0];

        // initialize first row of MinCost matrix
        F(j,1,Y-1)
            MinCost[0][j] = MinCost[0][j-1] + Cost[0][j];

        //Initialize first column of MinCost Matrix
        F(i,1,X-1)
            MinCost[i][0] = MinCost[i-1][0] + Cost[i][0];

        //This bottom-up approach ensures that all the sub-problems
    needed
        // have already been calculated.
        F(i,1,X-1)
        {
            F(j,1,Y-1)
            {
                //Calculate cost of visiting (i,j) using the
                //recurrence relation discussed above
                MinCost[i][j] = min(MinCost[i-1][j],MinCost[i][j-1]) +
Cost[i][j];
            }
        }

        cout<<"Minimum cost from (0,0) to (X,Y) is "<<MinCost[X-1][Y-1];
        return 0;
    }
```

Another variant of this problem includes another direction of motion, i.e. one is also allowed to move diagonally lower from cell (i,j) to cell (i+1,j+1). This question can also be solved easily using a slight modification in the recurrence relation. To reach (i,j), we must first reach either (i-1,j), (i,j-1) or (i-1,j-1).

```
MinCost(i,j) = min(MinCost(i-1,j),MinCost(i,j-1),MinCost(i-1,j-1)) +
Cost[i][j]
```

---

2. Finding the number of ways to reach from a starting position to an ending position travelling in specified directions only.

Problem Statement : Given a 2-D matrix with M rows and N columns, find the number of ways to reach cell with coordinates (i,j) from starting cell (0,0) under the

condition that you can only travel one step right or one step down.

Solution : This problem is very similar to the previous one. To reach a cell (i,j), one must first reach either the cell (i-1,j) or the cell (i,j-1) and then move one step down or to the right respectively to reach cell (i,j). After convincing yourself that this problem indeed satisfies the optimal sub-structure and overlapping subproblems properties, we try to formulate a bottom-up dynamic programming solution.

We first need to identify the states on which the solution will depend. To find the number of ways to reach to a position, what are the variables on which my answer depends? Here, we need the row and column number to uniquely identify a position. For more details on how to decide the state of a dynamic programming solution, see this : How can one start solving Dynamic Programming problems? Therefore, let NumWays(i,j) be the number of ways to reach position (i,j). As stated above, number of ways to reach cell (i,j) will be equal to the sum of number of ways of reaching (i-1,j) and number of ways of reaching (i,j-1). Thus, we have our recurrence relation as :

```
numWays(i,j) = numWays(i-1,j) + numWays(i,j-1)
```

Now, all you need to do is take care of the base cases and the recurrence relation will calculate the rest for you. :)

The base case, as in the previous question, are the topmost row and leftmost column. Here, each cell in topmost row can be visited in only one way, i.e. from the left cell. Similar is the case for the leftmost column. Hence the code is:

```cpp
#include <bits/stdc++.h>
    using namespace std;
    #define F(i,a,b) for(int i = (int)(a); i <= (int)(b); i++)
    #define RF(i,a,b) for(int i = (int)(a); i >= (int)(b); i--)
    int main()
    {
        int X,Y; //X:number of rows, Y: number of columns
        X = Y = 10; //assuming 10X10 matrix

        int NumWays[X][Y]; //declare the NumWays matrix
        NumWays[0][0] = 1;
        // initialize first row of NumWays matrix
        F(j,1,Y-1)
            NumWays[0][j] = 1;
        //Initialize first column of NumWays Matrix
        F(i,1,X-1)
            NumWays[i][0] = 1;
        //This bottom-up approach ensures that all the sub-problems needed
        // have already been calculated.

        F(i,1,X-1)
        {
```

```
        F(j,1,Y-1)
        {
            //Calculate number of ways visiting (i,j) using the
            //recurrence relation discussed above
            NumWays[i][j] = NumWays[i-1][j] + NumWays[i][j-1];
        }
    }

    cout<<"Number of ways from(0,0) to (X,Y) is "<<NumWays[X-1][Y-1];
    return 0;
}
```

3. Finding the number of ways to reach a particular position in a grid from a starting position (given some cells which are blocked)

Problem Statement : A robot is designed to move on a rectangular grid of M rows and N columns. The robot is initially positioned at (1, 1), i.e., the top-left cell. The robot has to reach the (M, N) grid cell. In a single step, robot can move only to the cells to its immediate east and south directions. That means if the robot is currently at (i, j), it can move to either (i + 1, j) or (i, j + 1) cell, provided the robot does not leave the grid. Now somebody has placed several obstacles in random positions on the grid, through which the robot cannot pass. Given the positions of the blocked cells, your task is to count the number of paths that the robot can take to move from (1, 1) to (M, N).

Input is three integers M, N and P denoting the number of rows, number of columns and number of blocked cells respectively. In the next P lines, each line has exactly 2 integers i and j denoting that the cell (i, j) is blocked.

Solution : The code below explains how to proceed with the solution. The problem is same as the previous one, except for few extra checks(due to blocked cells.)

```cpp
#include <bits/stdc++.h>
 using namespace std;

 typedef long long int ll;
 #define F(i,a,b) for(int i = (int)(a); i <= (int)(b); i++)
 #define RF(i,a,b) for(int i = (int)(a); i >= (int)(b); i--)
 #define MOD 1000000007
 int main()
 {
     int M,N,P,_i,_j;

     //Take input the number of rows, columns and blocked cells
     cin>>M>>N>>P;

     //declaring a Grid array which stores the number of paths
```

```
            //declaring a Grid array which stores the number of paths
            ll Grid[M+1][N+1];

            //Note that we'll be using 1-based indexing here.
            //initialize all paths initially as 0
            memset(Grid, 0, sizeof(Grid));

            F(i,0,P-1)
            {
                //Take in the blocked cells and mark them with a special
value(-1 here)
                cin>>_i>>_j;
                Grid[_i][_j] = -1;
            }


            // If the initial cell is blocked, there is no way of moving
anywhere
            if(Grid[1][1] == -1)
            {
                printf("0");
                return 0;
            }



            // Initializing the leftmost column
            //Here, If we encounter a blocked cell, there is no way of
visiting any cell
            //directly below it.(therefore the break)
            F(i,1,M)
            {
                if(Grid[i][1] == 0) Grid[i][1] = 1LL;
                else break;
            }



            //Similarly initialize the topmost row.

            F(i,2,N)
            {
                if(Grid[1][i] == 0) Grid[1][i] = 1LL;
                else break;
            }



            //Now the recurrence part
            //The only difference is that if a cell has been marked as -1,
            //simply ignore it and continue to the next iteration.
            F(i,2,M)
```

```
            {
                F(j,2,N)
                {
                    if(Grid[i][j] == -1) continue;

                    //While adding the number of ways from the left and top
    cells,
                    //check that they are reachable,i.e. they aren't blocked

                    if(Grid[i-1][j] > 0) Grid[i][j] = (Grid[i][j] + Grid[i-
    1][j] + MOD)%MOD;
                    if(Grid[i][j-1] > 0) Grid[i][j] = (Grid[i][j] + Grid[i]
    [j-1] + MOD)%MOD;
                }
            }


            //If the final cell is blocked, output 0, otherwise the answer
            printf("%lld",(Grid[M][N] >= 0 ? Grid[M][N] : 0));
            return 0;
        }
```

Another variant

Finally, we discuss another variant of problems involving grids.

> **Problem Statement** : You are given a 2-D matrix A of n rows and m columns where
> A[i][j] denotes the calories burnt. Two persons, a boy and a girl, start from two corners
> of this matrix. The boy starts from cell (1,1) and needs to reach cell (n,m). On the
> other hand, the girl starts from cell (n,1) and needs to reach (1,m). The boy can move
> right and down. The girl can move right and up. As they visit a cell, the amount in the
> cell A[i][j] is added to their total of calories burnt. You have to maximize the sum of
> total calories burnt by both of them under the condition that they shall meet only in
> one cell and the cost of this cell shall not be included in either of their total.

Solution : Let us analyse this problem in steps:

> The boy can meet the girl in only one cell.

So, let us assume they meet at cell (i,j).

Boy can come in from left or the top, i.e. (i,j-1) or (i-1,j). Now he can move right or down.That is,
the sequence for the boy can be:

```
(i,j-1)-->(i,j)-->(i,j+1)
(i,j-1)-->(i,j)-->(i+1,j)
(i-1,j)-->(i,j)-->(i,j+1)
(i-1,j)-->(i,j)-->(i+1,j)
```

Similarly, the girl can come in from the left or bottom, i.e. (i,j-1) or (i+1,j) and she can go up or right. The sequence for girl's movement can be:

```
(i,j-1)-->(i,j)-->(i,j+1)
(i,j-1)-->(i,j)-->(i-1,j)
(i+1,j)-->(i,j)-->(i,j+1)
(i+1,j)-->(i,j)-->(i-1,j)
```

Comparing the 4 sequences of the boy and the girl, the boy and girl meet only at one position (i,j), iff

```
Boy: (i,j-1)-->(i,j)-->(i,j+1) and Girl: (i+1,j)-->(i,j)-->(i-1,j)
```

or

```
Boy: (i-1,j)-->(i,j)-->(i+1,j) and Girl:  (i,j-1)-->(i,j)-->(i,j+1)
```

Convince yourself that in no other case will they meet at only one position.

Now, we can solve the problem by creating 4 tables:

1. Boy's journey from start (1,1) to meeting cell (i,j)
2. Boy's journey from meeting cell (i,j) to end (n,m)
3. Girl's journey from start (n,1) to meeting cell (i,j)
4. Girl's journey from meeting cell (i,j) to end (1,n)

The meeting cell can range from 2<= i <= n-1 and 2 <= j <= m-1

See the code below for more details:

```cpp
#include <bits/stdc++.h>
    #define F(i,a,b) for(int i = (int)(a); i <= (int)(b); i++)
    #define RF(i,a,b) for(int i = (int)(a); i >= (int)(b); i--)
    #define MAX 1005
    int Boy1[MAX][MAX];
    int Boy2[MAX][MAX];
    int Girl1[MAX][MAX];
    int Girl2[MAX][MAX];

    using namespace std;
    int main()
    {
        int N,M,ans,op1,op2;
        scanf("%d%d",&N,&M);
        int Workout[MAX][MAX];
        ans = 0;
```

```
            //Take input the calories burnt matrix
            F(i,1,N)
                F(j,1,M)
                    scanf("%d",&Workout[i][j]);

            //Table for Boy's journey from start to meeting cell
            F(i,1,N)
                F(j,1,M)
                    Boy1[i][j] = max(Boy1[i-1][j],Boy1[i][j-1]) + Workout[i]
[j];

            //Table for boy's journey from end to meet cell
            RF(i,N,1)
                RF(j,M,1)
                    Boy2[i][j] = max(Boy2[i+1][j],Boy2[i][j+1]) + Workout[i]
[j];


            //Table for girl's journey from start to meeting cell
            RF(i,N,1)
                F(j,1,M)
                    Girl1[i][j] = max(Girl1[i+1][j],Girl1[i][j-1]) +
Workout[i][j];


            //Table for girl's journey from end to meeting cell
            F(i,1,N)
                RF(j,M,1)
                    Girl2[i][j] = max(Girl2[i-1][j],Girl2[i][j+1]) +
Workout[i][j];


            //Now iterate over all meeting positions (i,j)
            F(i,2,N-1)
            {
                F(j,2,M-1)
                {
                    //For the option 1
                    op1 = Boy1[i][j-1] + Boy2[i][j+1] + Girl1[i+1][j] +
Girl2[i-1][j];

                    //For the option 2
                    op2 = Boy1[i-1][j] + Boy2[i+1][j] + Girl1[i][j-1] +
Girl2[i][j+1];

                    //Take the maximum of two options at each position
                    ans = max(ans,max(op1,op2));
```

```
                    }
            }

            printf("%d",ans);
            return 0;
    }
```

4. Edit Distance

Edit distance is a way of quantifying how dissimilar two strings are, i.e., how many operations (add, delete or replace character) it would take to transform one string to the other. This is one of the most common variants of edit distance, also called Levenshtein distance, named after Soviet computer scientist, Vladimir Levenshtein. There are 3 operations which can be applied to either string, namely: insertion, deletion and replacement.

```cpp
int editDistance(string s1, string s2) {
        int m = s1.size();
        int n = s2.size();
        // for all i, j, dp[i][j] will hold the edit distance between the
  first
        // i characters of source string and first j characters of target
string
        int dp[m + 1][n + 1];
        memset(dp, 0, sizeof(dp));
        // source can be transformed into target prefix by inserting
        // all of the characters in the prefix
        for (int i = 1; i <= n; i++) {
            dp[0][i] = i;
        }
        // source prefixes can be transformed into empty string by
        // by deleting all of the characters
        for (int i = 1; i <= m; i++) {
            dp[i][0] = i;
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (s1[i - 1] == s2[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1]; // no operation
  required as characters are the same
                }
                else {
                    dp[i][j] = 1 + min(dp[i - 1][j - 1],    //
  substitution
                                    min(dp[i][j - 1],    // insertion
                                        dp[i - 1][j]));    // deletion
                }
```

```
                }
            }
            return dp[m][n];
        }
```

Let's look at the DP table when s1 = "sitting" (source string)
s2 = "kitten" (target string)

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Dynamic Programming is not an algorithm or data-structure. It is a technique and it is applied to a certain class of problems. The key to figure, if a problem can be solved by DP, comes by practice.

Contributed by: Prateek Garg

Did you find this tutorial helpful?   👍 YES   👎 NO

TEST YOUR UNDERSTANDING

Matrix Sum

You are given a $N * M$ matrix. You need to print the sum of all the numbers in the rectangle which has $(1, 1)$ as the top left corner and (X, Y) as the bottom right corner.

Input:
First line contains two integers, N and M, number of rows and number of columns in the matrix.
Next N lines contains M space separated integers, elements of the matrix.
Next line contains an integer, Q, number of queries.
Each query contains two space separated integers, X and Y.

Output:
For each query, print the sum of all the numbers in the rectangle which has $(1, 1)$ as the top left

corner and (X, Y) as the bottom right corner.

Constraints:
$1 \leq N, M \leq 10^3$
$1 \leq$ elements of matrix $\leq 10^3$
$1 \leq X \leq N$
$1 \leq Y \leq M$
$1 \leq Q \leq 10^5$

SAMPLE INPUT

```
3 3
1 2 3
4 5 6
7 8 9
2
3 3
2 3
```

SAMPLE OUTPUT

```
45
21
```

Enter your code or Upload your code as file.   Save   C (gcc 5.4.0) ▾

```
1    /*
2    // Sample code to perform I/O:
3
4    scanf("%s", name);            // Reading input from STDIN
5    printf("Hi, %s.\n", name);    // Writing output to STDOUT
6
7    // Warning: Printing unwanted or ill-formatted data to output will cause the test case
8    */
9
10   // Write your code here
11
```

1:1

▼ Provide custom input   💡 *Press Ctrl-space for autocomplete suggestions.*

COMPILE & TEST   SUBMIT

COMMENTS (24) ↻

SORT BY: Relevance ▾

Login/Signup to Comment

Akhilesh Soni 7 months ago
Hint : Formula : dp[i][j]=(a[i][j]+dp[i][j-1]+dp[i-1][j]-dp[i-1][j-1]);
▲ 11 votes ● Reply ● Message ● Permalink

> **Remi Kassoua** ✎ Edited 4 months ago
> can you please explain why did you subtract dp[i-1][j-1] form this formula?
> ▲ 0 votes ● Reply ● Message ● Permalink
>
>> **bpr10ry** 3 months ago
>> wen u add dp[i][j-1] and dp[i-1][j], the dp[i-1][j-1] is added two times. cause dp[i-1][j-1] is common in both dp[i][j-1] and dp[i-1][j].
>> ▲ 3 votes ● Reply ● Message ● Permalink

akshat sharma 7 months ago
this article is badly written
▲ 9 votes ● Reply ● Message ● Permalink

Neha Mota 4 months ago
In 3, one more condition needs to be added. if the boy goes from (i-1, j) -> (i, j) -> (i+1, j) and girl goes from (i+1, j) -> (i, j) -> (i-1, j).
▲ 2 votes ● Reply ● Message ● Permalink

> **Attyuttam Saha** 2 months ago
> even i have this doubt. a clear explanation of the problem is required
> ▲ 1 vote ● Reply ● Message ● Permalink

Vishwas Tyagi a year ago
What will be the output of query 2 2
▲ 0 votes ● Reply ● Message ● Permalink

> **Abhishek Kumar** a year ago
> 12
> ▲ 1 vote ● Reply ● Message ● Permalink

xarilaos 8 months ago
http://stackoverflow.com/questions/2277749/calculate-the-sum-of-elements-in-a-matrix-efficiently
▲ 1 vote ● Reply ● Message ● Permalink

mohammadmahdi abdollahpoor 5 months ago
I exceed the time limit why?
import java.util.Scanner;
/**
* Created by ASUS on 12/07/2017.
*/
public class MatrixSum {
static int[][] sum;
static int[][] mat;
static int findSum(int x, int y) {
if (x < 0 || y < 0) {

```
return 0;
}
if (sum[y][x] != -1) {
return sum[y][x];
}


sum[y][x]= findSum(x - 1, y) + findSum(x, y - 1) + mat[y][x]-findSum(x-1,y-1);
return sum[y][x];
}
public static void main(String[] args) {
Scanner in = new Scanner(System.in);
int n = in.nextInt();
int m = in.nextInt();
sum = new int[m][n];
mat = new int[m][n];
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
mat[j][i]=in.nextInt();
sum[j][i] = -1;
}
}
sum[0][0]=mat[0][0];
int q = in.nextInt();
for (int i = 0; i < q; i++) {
int x = in.nextInt();//n
int y = in.nextInt();//m
x--;
y--;
System.out.println(findSum(x, y));
}
}
}
```

▲ 1 vote ● Reply ● Message ● Permalink

Ashish Nimbalkar 7 months ago
https://ideone.com/ONz9oV
▲ 0 votes ● Reply ● Message ● Permalink

Anindya Pal 6 months ago
How to generate the actual paths taken to reach a particular position in a grid from a starting
position? Also, I need to have the paths available somewhere so that I can sort them
lexicographically.
▲ 0 votes ● Reply ● Message ● Permalink

Gowtham 6 months ago
You shouldn't ask answers during the contest.
▲ 0 votes ● Reply ● Message ● Permalink

Saksham Sharma 6 months ago
https://ideone.com/5LqdXH
what is problem with this solution,
it showing wrong on only first query and correct for all other .
▲ 0 votes ● Reply ● Message ● Permalink

Jai Prakash Thakur 5 months ago
C++ Solution:
```
#include <iostream>
using namespace std;
int sumarr[1001][1001]={0};
int ar[1001][1001];
int findsum(int ar[1001][1001],int X,int Y){
if((X<1)||(Y<1))
```

```
return 0;

if(sumarr[X][Y]!=0)
return sumarr[X][Y];
else
sumarr[X][Y]=findsum(ar,X-1,Y)+findsum(ar,X,Y-1)-findsum(ar,X-1,Y-1)+ar[X][Y];
return sumarr[X][Y];
}
int main()
{
int N,M,i,j,no,Q,X,Y,sum;
cin>>N>>M;
for(i=1;i<=N;i++){
for(j=1;j<=M;j++){
cin>>no;
ar[i][j]=no;
}
}
cin>>Q;
while(Q-->0){
cin>>X>>Y;
sum=findsum(ar,X,Y);
cout<<sum<<endl;
}
}
```

▲ 0 votes ● Reply ● Message ● Permalink

Priyanshu Varshney 5 months ago

Does the special variant above "Edit distance " mean that the boy and girl should meet at one point and also they should not traverse the cell already reached by the other player. If not then why is the outer rows and columns not visited. Also a little discussion on the calculation of matrices will be helpful for me.

▲ 0 votes ● Reply ● Message ● Permalink

Sikder 5 months ago

please submit a solution for C

▲ 0 votes ● Reply ● Message ● Permalink

Bhoomi Vaghasiya 5 months ago

how to optimize this code:
```
#include<bits/stdc++.h>
using namespace std;
int main()
{
int n,m;
cin>>n>>m;
int arr[n][m];
for(int i=0;i<n;++i)
{
for(int j=0;j<m;++j)
cin>>arr[i][j];
}
int q;
cin>>q;
while(q--)
{
int x,y;
cin>>x>>y;

int sum=0;
for(int i=0;i<x;++i)
{
```

```
for(int j=0;j<y;++j)
{
sum=sum+arr[i][j];
}
}
cout<<sum<<endl;
}
}
```
▲ 0 votes ● Reply ● Message ● Permalink

Vivek Sethi 3 months ago
Had to resort to Fast I/O (using Java)
▲ 0 votes ● Reply ● Message ● Permalink

Attyuttam Saha 2 months ago
in that boy girl problem can someone please explain why two matrices are required for the boy
and girl each. like why a matrix from start to meeting point and the other from end point to
meeting point ?
▲ 0 votes ● Reply ● Message ● Permalink

> **Chirag Mittal (LIT2016022)** ✎ Edited 2 months ago
> because we are creating a matrix for the meeting point and the two matrix will represent the
> path for reaching that point and the path which is followed after meeting at that point
> respectively.
> ▲ 0 votes ● Reply ● Message ● Permalink

Rohit 18 days ago
For those, getting TLE. please use scanf and printf and use dp[1001][1001]={}; instead of memset.
▲ 0 votes ● Reply ● Message ● Permalink

Aditya Aggarwal ✎ Edited 5 days ago
Very Well Explained.
Indeed a Nice Blog on DP.
▲ 0 votes ● Reply ● Message ● Permalink

akshive pandey 3 days ago
```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int A[1001][1001];
long int sum[1001][1001];
long int tot(int A[1001][1001], int X, int Y){
if(sum[X][Y] != -1){
return sum[X][Y];
}
return sum[X][Y] = tot(A, X-1, Y)+tot(A, X, Y-1)-tot(A, X-1, Y-1)+A[X][Y];
}
int main(){
int M, N;
cin>>M>>N;
for(int i = 0; i < M; i++){
for(int j = 0; j < N; j++){
cin>>A[i][j];
}
}
memset(sum, -1, sizeof(sum));
sum[0][0] = A[0][0];
for(int i = 1; i < M; i++){
sum[0][i] = A[0][i]+sum[0][i-1];
}
for(int i = 1; i < N; i++){
sum[i][0] = A[i][0]+sum[i-1][0];
```

```
}
int Q;
cin>>Q;
while(Q--){
int X, Y;
cin>>X>>Y;
long int res = tot(A, X-1, Y-1);
cout<<res<<endl;
}
return 0;
}
```

▲ 0 votes ● Reply ● Message ● Permalink

About Us	Innovation Management
Talent Assessment	University Program
Developers Wiki	Blog
Press	Careers

Reach Us

Site Language: English   ▼ | Terms and Conditions | Privacy |© 2017 HackerEarth