← Notes

Dynamic Programming - Problems involving Grids

24 Dynamic Programming Competitive Programming Grid

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)

---

2. 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.

```
#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 ne
    // 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]) -
        }
    }

    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))
```

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:

```
#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 ne
    // 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-
    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 : You can read the problem statement here: Robots and Paths 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.)

```
#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
    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
        cin>>_i>>_j;
        Grid[_i][_j] = -1;
    }

    // If the initial cell is blocked, there is no way of moving
    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 vi
    //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 t
            //check that they are reachable,i.e. they aren't bloc

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

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

Another variant

Finally, we discuss another variant of problems involving grids. You can see the problem here. Working Out A short description of the problem is:

> **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:

```
#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

    //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

    //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]) + Work
```

```
    //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]) + Work

    //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] + G

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

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

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

Thanks for reading. :) I hope it was helpful!

AUTHOR

Shubham Goswami
Bahadurgarh
1 note

Drafts

TRENDING NOTES

Strings And String Functions
written by Vinay Singh

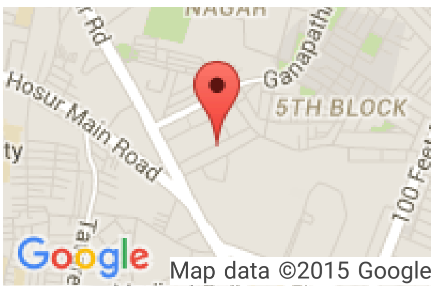
Segment Tree and Lazy Propagation
written by Akash Sharma

Number Theory - II
written by Tanmay Chaudhari

Matrix exponentiation
written by Mike Koltsov

Graph Theory - Part II
written by Pawel Kacprzak

more ...



Annual Report

IIIrd Floor, Salarpuria Business Center,
4th B Cross Road, 5th A Block,
Koramangala Industrial Layout,
Bangalore, Karnataka 560095, India.

contact@hackerearth.com
+91-80-4155-4695
+1-650-461-4192