SOLVE LATER

Avoiding networked paths

Attempted by: **834** / Accuracy: **86%** / Maximum Score: **50** / 16 Votes

Tag(s): Algorithms, Counting and Arrangements, Dynamic Programming, Number theory

PROBLEM | EDITORIAL | MY SUBMISSIONS | ANALYTICS

Suppose there is a grid of size $n * m$ where each cell is either black or white in color. We need to find number of paths from $(1, 1)$ to (n, m)such that we travel only through white cells (i.e we cannot visit black cell at all). We can either go right or down from a cell.

Let $TotalPath(a, e)$ = Total number of paths from cell a to cell e = $(a_x - e_x + a_y - e_y)!/((a_x - e_x)! * (a_y - e_y)!)$

B = Total Number of paths having at least one black cell

C = Total number of paths having no black cell at all

Then $C = TotalPath((1, 1), (n, m)) - B$

In order to find B , let us assume we have a list of black cells $[b_1, b_2, b_3 . . . . . b_k]$ where each b_i is some cell (x, y). Lets sort the list in the increasing order of row and in case of same row , we will have the increasing order of column.

Let $BlackPath(b_j)$ = Number of paths ending at b_j without visiting any b_i such that $i < j$

Then $BlackPath(b_j) = TotalPath((1, 1), b_j) - \sum_{i=1}^{j-1} BlackPath(b_i) * TotalPath(b_i, b_j)$

If (n, m) is not a black cell , then
$C = TotalPath((1, 1), (n, m)) - \sum_{i=1}^{k} BlackPath(b_i) * TotalPath(b_i, (n, m))$

BEST SUBMISSIONS

LANGUAGE: C++ (g++ 5.4.0)

TIME (sec) 5.46748 | MEMORY (KiB) 31840

by Ayan Sheikh

VIEW BEST SUBMISSION | VIEW ALL SUBMISSION

CONTRIBUTOR

AUTHOR
Reshab Gupta

TESTER
Amirreza Poorakhavan

THIS PROBLEM WAS ASKED IN

CHALLENGE NAME
April Circuits '20

SOCIAL SHARE

LIVE EVENTS

Now let us solve the actual problem now.

$X = P_1 * P_2$ where P_i is a prime number.

Let $D =$ Number of paths having product divisible by P_1 and not by P_2

$E =$ Number of paths having product divisible by P_2 and not by P_1

$F =$ Number of paths having product divisible by neither P_1 nor P_2

$G =$ Total number of paths having product not divisible by $X = D + E + F$

We will use the above mentioned algorithm to find G .

Lets assume points having value divisible by P_2 are black cells and we can make a sorted list of black cells as $[b_1, b_2, b_3 \ldots\ldots b_k]$

If (n, m) is not a black cell , then
$C_1 = TotalPath((1,1),(n,m)) - \sum_{i=1}^{k} BlackPath(b_i) * TotalPath(b_i,(n,m)) = D + F$

Like this we can also find $C_2 = E + F$ and $C_3 = F$

Then $G = C_1 + C_2 - C_3$

Analysis:

- Precomputation of factorials and inverse factorials $= O((n+m) * log2(MOD))$
- Sorting the black cells $= O(K * log_2(K))$
- Computation of $C = O(K * K)$ by using memoization the recurrence relation mentioned above.
- Time complexity $= O((n+m) * log2(MOD) + K^2))$
- Space complexity $= O(n + m + K)$
- There is a solution which has time complexity of $= O((n+m) * log2(MOD) + K^3)$ and space complexity of $O(n + m + K^2)$, but it wont be able to get 100 points for this problem. https://ideone.com/4ZPtaw

---

IS THIS EDITORIAL HELPFUL?

Yes, it's helpful

No, it's not helpful

17 developer(s) found this editorial helpful.

Author Solution by Reshab Gupta

```cpp
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm>

using namespace std;

#define ll long long
#define MAX 2000000
#define P1 1000000007
#define P2 100000007
#define MOD 1000000007
#define X 100000007700000049

int fact[MAX + 5], ifact[MAX + 5];

inline int mul(int x, int y) {
    ll z = 1LL * x * y;
    if (z >= MOD) {
        z %= MOD;
    }
    return z;
}

inline int add(int x, int y) {
    int z = x + y;
    if (z >= MOD) {
        z -= MOD;
```

```
    }
    return z;
}

inline int sub(int x, int y) {
    int z = x - y;
    if (z < 0) {
        z += MOD;
    }
    return z;
}

inline int fastExp(int x, int y) {
    int z = 1;
    while (y) {
        if (y & 1) {
            z = mul(z, x);
        }
        y >>= 1;
        x = mul(x, x);
    }
    return z;
}

void preCompute(int n) {
    fact[0] = ifact[0] = 1;
    for (int i = 1; i <= n; ++i) {
        fact[i] = mul(fact[i - 1], i);
        ifact[i] = mul(ifact[i - 1], fastExp(i, MOD - 2));
    }
}

int getCategory(ll no) {
    if (no % X == 0) {
        return 2;
    } else if (no % P1 == 0) {
        return 1;
    } else if (no % P2 == 0) {
        return 0;
```

```
    }
    return -1;
}

int getWays(int n, int m) {
    return mul(fact[n + m], mul(ifact[n], ifact[m]));
}

int solveTask(vector<pair<int, int>> points) {
    int i, j, x, y, xx, yy, tot, k;
    int memoize[5005];
    k = points.size();
    for (i = 0; i < k; ++i) {
        x = points[i].first;
        y = points[i].second;
        tot = getWays(x - 1, y - 1); //Total no of ways to reach point
(x,y)
        for (j = i - 1; j >= 0; --j) {
            xx = points[j].first;
            yy = points[j].second;
            if (yy <= y) {
                //special point
                tot = sub(tot, mul(memoize[j], getWays(x - xx, y - yy)));
            }
        }
        //tot = total no of ways to reach point (x,y) without visiting
special points
        memoize[i] = tot;
    }
    return memoize[k - 1];
}

int main() {

    ios_base::sync_with_stdio(0);
    cin.tie(0);
//    freopen("in.txt", "r", stdin);
//    freopen("out1.txt", "w", stdout);

```

```
    int n, m, k, w, i, x, y, cat, catDest = -1, cnt[3];
    ll v;
    vector<pair<int, int> > points[3];
    cin >> n >> m >> k >> w;
    for (i = 0; i < k; ++i) {
        cin >> x >> y >> v;
        if ((cat = getCategory(v)) != -1) {
            if (cat != 0) {
                points[0].emplace_back(x, y);
            }
            if (cat != 1) {
                points[1].emplace_back(x, y);
            }
            points[2].emplace_back(x, y);
        }
        if (x == n and y == m) {
            catDest = getCategory(v);
        }
    }
    preCompute(n + m);
    for (i = 0; i < 3; ++i) {
        if (catDest == -1 or (catDest != 2 and catDest == i)) {
            points[i].emplace_back(n, m);
        }
        sort(points[i].begin(), points[i].end());
    }
    cnt[2] = catDest == -1 ? solveTask(points[2]) : 0;
    cnt[1] = (catDest == 1 or catDest == -1) ? solveTask(points[1]) : 0;
    cnt[0] = (catDest == 0 or catDest == -1) ? solveTask(points[0]) : 0;
    cout << sub(add(cnt[0], cnt[1]), cnt[2]);

    return 0;
}
```

COMMENTS (1)

SORT BY: Relevance

Join Discussion...

Cancel Post

monu kumar 2 days ago

https://atcoder.jp/contests/dp/tasks/dp_y

▲ 5 votes ● Reply ● Message ● Permalink

+1-650-461-4192

contact@hackerearth.com