Giải thuật: Nguyên lý bù trừ và quy hoạch động theo lát cắt .

Giải thuật I

Với **n** và **m** đủ nhỏ (**n** , **m** ≤ 1000) bài toán tìm số lượng đường đi dễ dàng giải quyết bằng *phương pháp quy hoạch động* với độ phức tạp $O(n \times m)$.

Gọi $\mathbf{dp}_{i,j}$ – số lượng đường đi hợp lệ từ ô (1,1) tới ô (**i**,**j**), $\mathcal{K}$ – tập các ô có vật cản.

Ban đầu $\mathbf{dp}_{i,j}$ = -1 nếu $(i, j) \in \mathcal{K}$.

Công thức lặp:

$$\mathbf{dp}_{i,j} = \begin{cases} 1 \text{ với } i = 1 \text{ hoặc } j = 1, (i, j) \notin \mathcal{K}, \\ 0 \text{ nếu } (i, j) \in \mathcal{K}, \\ (\mathbf{dp}_{i-1,j} + \mathbf{dp}_{i,j-1}) \bmod \mathbf{p} \text{ với } (i, j) \notin \mathcal{K}. \end{cases}$$

Tổng số đường đi cần tìm là giá trị $\mathbf{dp}_{n,m}$.

Chương trình I:

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.out");

int n,m,k,p,x,y;
vector<vector<int>> dp;

int main()
{
    fi>>n>>m>>k>>p;
    dp.resize(n+1,vector<int> (m+1,0));

    for(int i=0; i<k; ++i)
    {
        fi>>x>>y;
        dp[x][y]=-1;
    }
    dp[0][1]=1;
    for(int i=1; i<=n; ++i)
```

```
        for(int j=1; j<=m; ++j)
            if(dp[i][j]==-1) dp[i][j]=0;
                    else dp[i][j]=(dp[i-1][j]+dp[i][j-1])%p;

    fo<<dp[n][m];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Giải thuật II

Với **n**, **m** lớn hơn 10^3, việc sử dụng mảng **dp** 2 chiều đòi hỏi bộ nhớ sử dụng lớn, vượt quá khả năng phục vụ của hệ thống lập trình.

Dòng thứ **i** của bảng **dp** được tính dựa trên dòng **i-1**. Vì vậy chỉ cần giữ 2 dòng của bảng và dùng kỹ thuật con lắc để từ dòng cũ tính dòng mới.

Trong trường hợp này cần lưu các ô chứa vật cản và sắp xếp thứ tự từ điển tăng dần của các tọa độ.

Độ phức tạp của giải thuật vẫn là $O(n \times m)$, nhưng bộ nhớ sử dụng chỉ thuộc bậc $O(m)$.

Chương trình II:

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.out");
typedef pair<int,int> pii;
int n,m,k,p,x,y,u=1,v=0,ib=0;
int main()
{
    fi>>n>>m>>k>>p;
    vector<vector<int>> dp(2,vector<int> (m+1,0));
    vector<pii>ob(k);                // Lưu các ô cấm
    for(int i=0; i<k; ++i)
    {
        fi>>x>>y;
        ob[i]={x,y};
    }
    sort(ob.begin(),ob.end());
    ob.push_back({n+1,m+1});              // Phần tử hàng rào
    dp[0][1]=1;
```

```
    for(int i=1; i<=n; ++i)
    {
        u^=1; v^=1;                     // Tạo con lắc
        for(int j=1; j<=m; ++j)
            if(ob[ib]==make_pair(i,j))dp[v][j]=0,++ib;
                else dp[v][j]=(dp[v][j-1]+dp[u][j])%p;
    }
    fo<<dp[v][m];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";

}
```

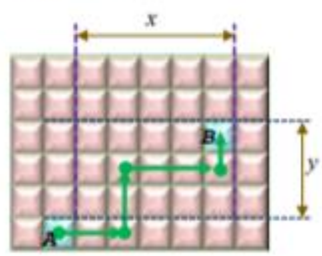
Thời gian thực hiện chương trình lớn vì độ phức tạp của giải thuật là $O(n \times m)$.

Giải thuật III

Áp dụng nguyên lý bù trừ có thể xây dựng giải thuật có *độ phức tạp chỉ phụ thuộc vào* ***k***. Như vậy sẽ xây dựng cho phép làm việc với *bảng kích thước rất lớn* (**n**, **m** $\leq 10^9$). Tuy vậy dưới đây ta chỉ xét *chi tiết cách triển khai* giải thuật cho trường hợp **n**, **m** $\leq 10^6$.

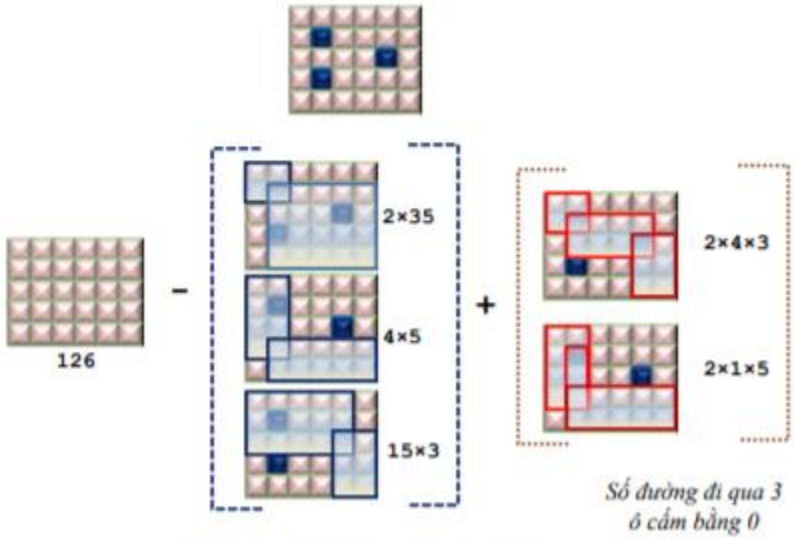
Mảng tọa độ ô cấm cần lưu dưới dạng sắp xếp theo thứ tự từ điển.

Xét số lượng đường đi từ **A** tới **B**, kể cả đi qua ô cấm (nếu có). Gọi số ô cần chuyển sang phải là **x** và sau đó phải chuyển lên trên **y** ô. Khi đó số lượng đường đi từ **A** tới **B** sẽ là C_{x+y}^{x} .



Ở ví dụ trong hình bên phải số lượng đường đi sẽ là $C_8^5 = C_8^3 = \frac{8\times7\times6}{1\times2\times3} = 56$.

Áp dụng đúng sơ đồ lý thuyết ta có nghiệm là tổng số đường đi qua mọi ô (kể cả ô cấm) trừ đi số đường đi qua từng ô cấm, cộng với số đường đi qua 2 ô cấm , trừ số đường đi qua 3 ô cấm , . . .



126 − 135 + 34 = 25

Giải thuật có độ phức tạp $O(2^k \times k)$.

Giải thuật IV

Giải thuật III có *độ phức tạp hàm mũ* vì vậy chỉ có hiệu quả khi **k** đủ nhỏ. Để giảm độ phức tạp của giải thuật cần kết hợp giữa nguyên lý bù trừ và quy hoạch động. Việc kết hợp 2 phương pháp nói trên sẽ tạo ra *giải thuật độ phức tạp đa thức đối với* ***k*** và vì vậy có thể áp dụng một cách có hiệu quả với **k** bậc 10^2, không phụ thuộc vào **n** và **m**.

Để tiện xử lý, các dòng và cột được đánh số bắt đầu từ 0.

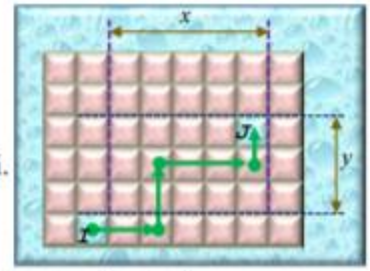
Xét $\mathfrak{X}$ – tập các ô có vật cản, ô xuất phát (**0, 0**) và ô đích (**n-1, m-1**). Tập sẽ có **k+2** phần tử, đánh số từ **0** đến **k+1**.

Sắp xếp các phần tử của $\mathfrak{X}$ theo thứ tự tăng dần.

Gọi $\mathbf{dp}_{i,j}$ – số lượng đường đi từ ô $\mathbf{i} \in \mathfrak{X}$ tới ô $\mathbf{j} \in \mathfrak{X}$, không chứa ô nào khác thuộc $\mathfrak{X}$.

Dựa vào tọa độ của các điểm **i** và **j**, như đã nêu ở trên, ta có thể dễ dàng tính được $rt_{i,j}$ – tất cả các đường đi từ **i** tới **j**, kể các đường đi qua ô cấm.

$$rt_{i,j} = \begin{cases} 0 \text{ nếu không có cách đi,} \\ C_{x+y}^{x} \text{ trong trường hợp ngược lại.} \end{cases}$$



Gọi **t** – ô cấm nằm giữa **i** và **j** trong $\mathfrak{X}$ đã sắp xếp, **i** < **t** < **j**, có

$$\boxed{dp_{i,j} = rt_{i,j} - \sum_{\forall t} dp_{i,t} \times rt_{t,j}}$$

Nghiệm của bài toán là $dp_{0,k+1}$.

Độ phức tạp của giải thuật: O(k^3).

Để giảm thời gian thực hiện cần tính sẵn bảng giá trị **i!** với **i** = 0, 1, 2, . . .

Xét *giải thuật IV_a* tính trực tiếp giá trị thực của $dp_{i,j}$ với **n**, **m** đủ nhỏ, trên cơ sở đó – cải tiến để nhận được lời giải bài toán đã nêu.

Giải thuật IV_a

Tổ chức dữ liệu

- Mảng `int64_t ft[20]` – lưu các giai thừa, ft_i = **i!**,
- Mảng `int64_t dp[100][100]` – phục vụ sơ đồ quy hoạch động,
- Mảng `vector<pii> ob` – lưu các phần tử thuộc tập $\mathfrak{X}$.

Xử lý

***Giải thuật IV_b** – Lời giải bài toán*

Hệ số C_{x+y}^{x} của nhị thức Newton tăng rất nhanh: $C_{2n}^{n} \approx \frac{2^{2n}}{\sqrt{\pi n}}$

Giải thuật đòi hỏi phải tính nhiều C_{x+y}^{x} với các **x**, **y** khác nhau,

Để giảm độ phức tạp của giải thuật:

- Lưu trữ bảng giá trị **q!** *mod* **p** với **q** = 0, 1, 2, . . ., 2×10^5,
- Tính C_{x+y}^{x} theo công thức $C_{x+y}^{x} = \frac{(x+y)!}{x!\times y!}$ và thay việc thực hiện phép chia bằng cách tính nghịch đảo theo mô đun.

Dựa vào bảng giá trị giai thừa đã lưu, có

(x+y)! = **a** (*mod* **p**),

x! = **b** (*mod* **p**),

y! = **c** (*mod* **p**).

$\frac{(x+y)!}{x!\times y!}$ = **z** (*mod* **p**) → **a** = **zXb×c** → **a=z×r**, trong đó **r** = **b×c** .