坐标变换（其二） 202309-2

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

int main() {
    int n, m; cin >> n >> m;
    vector<double> ls(100005, 1); // 拉伸的初始值为 1
    vector<double> xz(100005); // 旋转的初始值为 0
    for(int i = 1; i <= n; i++) {
        int type; double k; cin >> type >> k;
        if(type == 1) {
            ls[i] = ls[i - 1] * k;
            xz[i] = xz[i - 1];
        }
        else {
            ls[i] = ls[i - 1];
            xz[i] = xz[i - 1] + k;
        }
    }
    for(int i = 0; i < m; i++) {
        int left, right; double x, y;
        cin >> left >> right >> x >> y;
        double lashen = ls[right] / ls[left - 1];
        double xuanzhuan = xz[right] - xz[left - 1];
        x = lashen * x; y = lashen * y;
        cout << fixed << setprecision(3) << x * cos(xuanzhuan) - y *
sin(xuanzhuan) << " " << x * sin(xuanzhuan) + y * cos(xuanzhuan) << " " <<endl;
    }
    return 0;
}
```

前缀和 前缀积

保留小数点后几位

对x操作不能影响y 对y操作不能影响x

函数传数组为参数

前缀和为 `a[right] - a[left - 1];`

矩阵运算 202305-2

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10010, D = 30;
long long tmp[D][D] = {0}, ans[N][N] = {0}; // 放入函数内导致栈帧溢出
int Q[N][D], K[N][D], V[N][D], W[N];

int main() {
    int n, d; cin >> n >> d;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < d; j++) {
```

```
            cin >> Q[i][j];
        }
    }
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < d; j++) {
            cin >> K[i][j];
        }
    }
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < d; j++) {
            cin >> V[i][j];
        }
    }
    for(int i = 0; i < n; i++) {
        cin >> W[i];
    }

    for(int i = 0; i < d; i++) {
        for(int j = 0; j < d; j++) {
            for(int k = 0; k < n; k++) {
                tmp[i][j] += K[k][i] * V[k][j];
            }
        }
    }
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < d; j++) {
            for(int k = 0; k < d; k++) {
                ans[i][j] += Q[i][k] * tmp[k][j];
            }
            ans[i][j] *= (long long) W[i];
        }
    }
    for (int i = 0; i < n; i ++) {
        for (int j = 0; j < d; j ++)
            cout << ans[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

矩阵运算方式

$$\text{例如: } A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}, B = \begin{bmatrix} g & h & i \\ j & k & l \end{bmatrix}, \text{求} C = A \times B \text{。}$$

$$A \times B = \begin{bmatrix} a \times g + b \times j & a \times h + b \times k & a \times i + b \times l \\ c \times g + d \times j & c \times h + d \times k & c \times i + d \times l \\ e \times g + f \times j & e \times h + f \times k & e \times i + f \times l \end{bmatrix}$$

放入函数内导致栈帧溢出 放在 main 函数前

```
long long tmp[D][D] = {0}, ans[N][N] = {0};
int Q[N][D], K[N][D], V[N][D], W[N];

typedef long long LL;
```

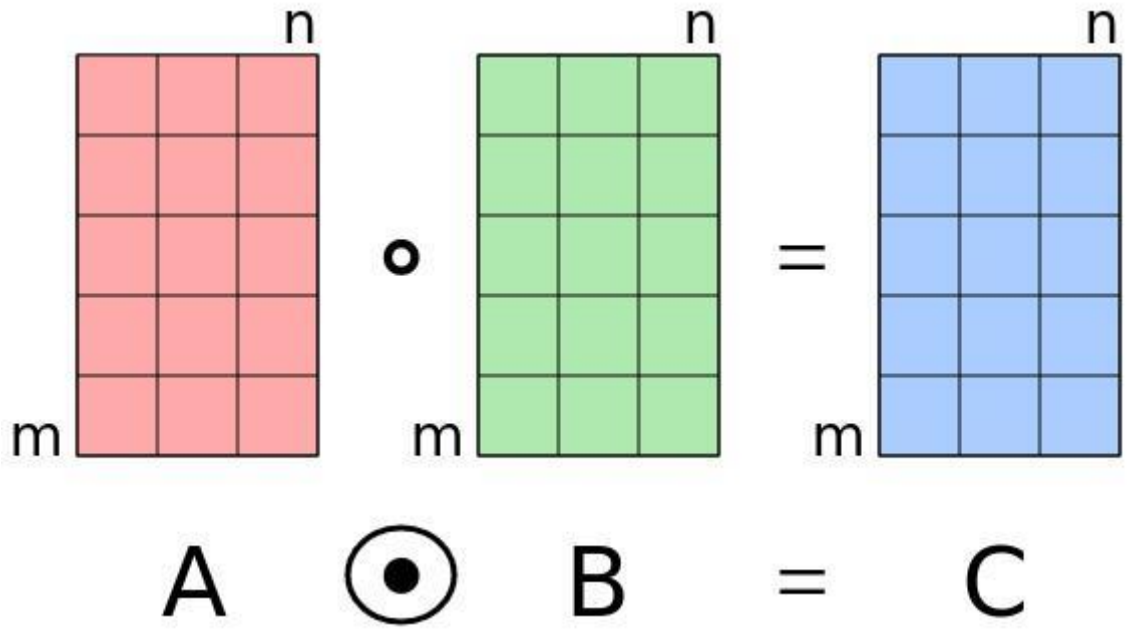
矩阵点乘——各个矩阵对应元素相乘

$$y = wx = \begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} = \begin{bmatrix} w_{11}x_{11} & w_{11}x_{12} & w_{11}x_{13} \\ w_{21}x_{21} & w_{21}x_{22} & w_{21}x_{23} \end{bmatrix}$$

or:

$$y = wx = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} = \begin{bmatrix} w_{11}x_{11} & w_{12}x_{12} & w_{13}x_{13} \\ w_{21}x_{21} & w_{22}x_{22} & w_{23}x_{23} \end{bmatrix}$$

当矩阵A和矩阵B的维度相同时，矩阵点乘即为**哈达玛积**（Hadamard Product/Point-wise Product/Element-wise Product/Element-wise Multiplication），如下图所示：



矩阵叉乘——矩阵乘法规则运算

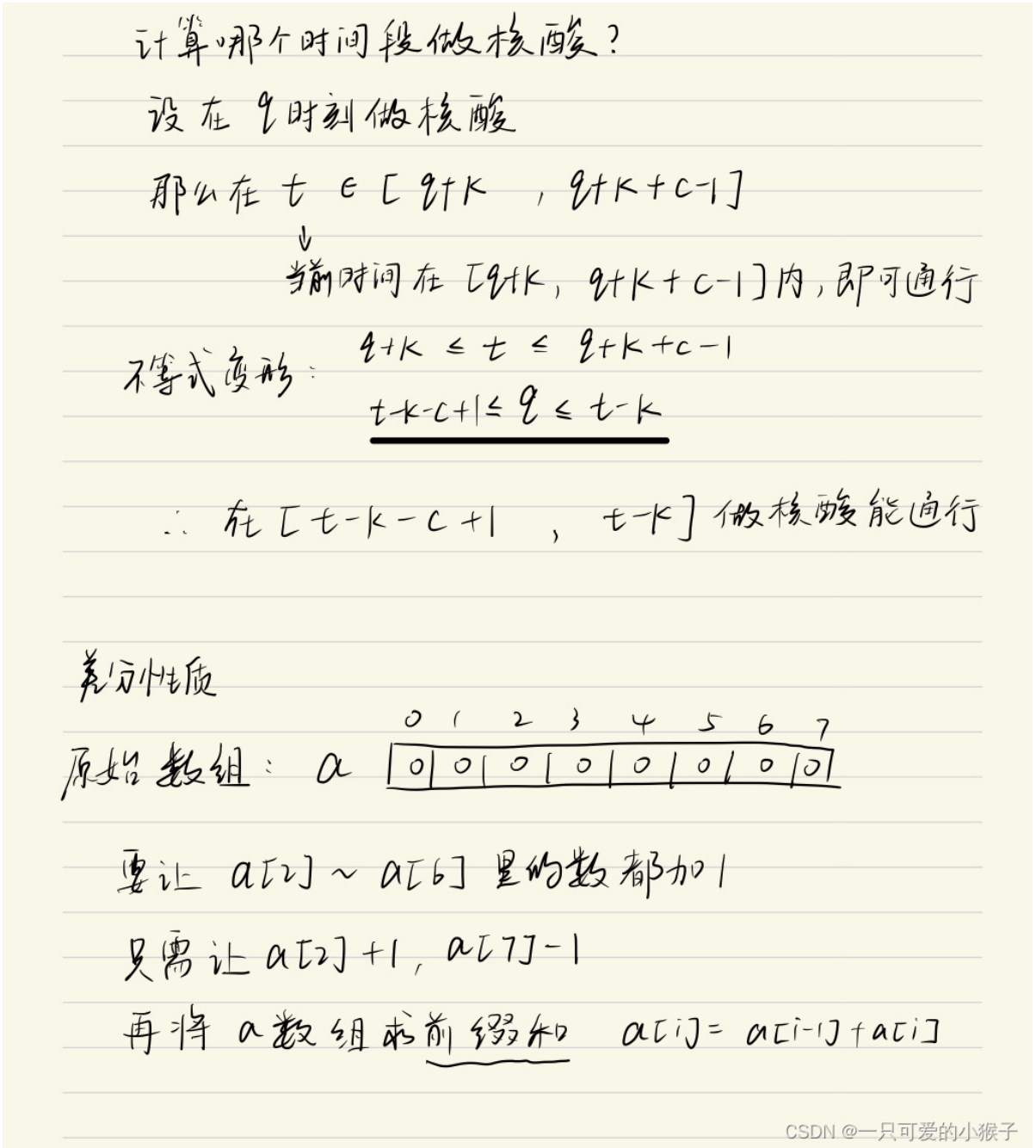
$$y = wx = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \times \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} w_{11}x_{11} + w_{12}x_{21} & w_{11}x_{12} + w_{12}x_{22} \\ w_{21}x_{11} + w_{22}x_{21} & w_{21}x_{12} + w_{22}x_{22} \\ w_{31}x_{11} + w_{32}x_{21} & w_{31}x_{12} + w_{32}x_{22} \end{bmatrix}$$

出行计划

解题思路

- 题意就是给你一个做核酸的时间，计算能出行的计划个数
- 有n个计划，m个时间询问，如果暴力求解，对于每一个询问时间，遍历每一个计划看能否出行，则时间复杂度是10亿，会超时，只能拿70分

- 然后我的解题思路是利用差分，首先对于每一个计划，计算应在哪个时间段内做核酸使得该计划能成功通行，让该时间段上的通行数都加一
- 所以我们需要开辟一个数组来记录每个时间能通行的数量
- 当遍历完所有通行计划后，对差分数组进行还原成原始数组
- 就可以利用数组下标直接得到询问的结果




```cpp
// 差分 前缀和
#include <bits/stdc++.h>
using namespace std;
int res[200010];

int main() {
    int n, m, k; cin >> n >> m >> k;
    int t, c;
    for(int i = 0; i < n; i++) {
        cin >> t >> c;
        int l = max(t - k - c + 1, 0); // 最早核酸时间
        int r = max(0, t - k); // 最晚核酸时间
```

```
        //在【l, r】时间段内能出行的计划个数加一
        res[l] += 1;
        res[r + 1] -= 1; // 边界
    }
    for (int i = 1; i < 200001; i++) {
        res[i] += res[i - 1]; // 前缀和
    }

    for (int i = 0; i < m; i++){
        int q;
        cin >> q;
        cout << res[q] << endl;
    }
    return 0;
}
```

非零段划分

差分

```
#include<bits/stdc++.h>
using namespace std;
int a[500005], b[10005];

int main()
{
    int n; cin >> n;
    int t = 1;
    for(int i = 1; i <= n; i++) {
        int input; cin >> input;
        if(input != a[t - 1]) {
            a[t] = input;
            t++;
        }
    }
    a[t] = 0;
    n = t - 1;
    for(int i = 1; i <= n; i++) {
        if(a[i] > a[i - 1] && a[i] > a[i + 1]) b[a[i]]++;
        if(a[i] < a[i - 1] && a[i] < a[i + 1]) b[a[i]]--;
    }
    int ans = 0, sum = 0;
    for(int i = 10001; i >= 0; i--) {
        sum += b[i];
        ans = max(ans, sum);
    }
    cout << ans;
}
```

邻域均值 —— 二位前缀和

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

```
int a[605][605], qzh[605][605];
int ans = 0;
int main() {
    int n, l, r, t; cin >> n >> l >> r >> t;

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            cin >> a[i][j];
            qzh[i][j] = qzh[i - 1][j] + qzh[i][j - 1] - qzh[i - 1][j - 1] + a[i]
[j];
        }
    }

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            int left = max(1, j - r);
            int right = min(j + r, n);
            int upper = max(i - r, 1);
            int down = min(i + r, n);
            int res = qzh[down][right] - qzh[down][left-1] - qzh[upper-1][right]
+ qzh[upper-1][left-1];
            if(res <= (down - upper + 1) * (right - left + 1) * t) ans ++;
        }
    }
    cout << ans;
    return 0;
}
```

unordered_set存储pair类型

```
struct pair_hash {
    template <class T1, class T2>
    size_t operator() (const pair<T1, T2> &p) const {
        auto hash1 = hash<T1>{}(p.first);
        auto hash2 = hash<T2>{}(p.second);
        return hash1 ^ hash2;
    }
};
unordered_set<pair<int, int>, pair_hash> s;
```

例:

```
#include <bits/stdc++.h>
using namespace std;
struct pair_hash {
    template <class T1, class T2>
    size_t operator() (const pair<T1, T2> &p) const {
        auto hash1 = hash<T1>{}(p.first);
        auto hash2 = hash<T2>{}(p.second);
        return hash1 ^ hash2;
    }
};
unordered_set<pair<int, int>, pair_hash> s;
int point[5] = {0};
```

```
int main() {
    int n; cin >> n;
    int x[n], y[n];
    for(int i = 0; i < n; i++) {
        cin >> x[i] >> y[i];
        s.insert(make_pair(x[i], y[i]));
    }
    for(int i = 0; i < n; i++) {
        if(s.find(make_pair(x[i], y[i] - 1)) != s.end() &&
           s.find(make_pair(x[i], y[i] + 1)) != s.end() &&
           s.find(make_pair(x[i] + 1, y[i])) != s.end() &&
           s.find(make_pair(x[i] - 1, y[i])) != s.end()) {
            int cnt = 0;
            if(s.find(make_pair(x[i] - 1, y[i] - 1)) != s.end()) cnt++;
            if(s.find(make_pair(x[i] - 1, y[i] + 1)) != s.end()) cnt++;
            if(s.find(make_pair(x[i] + 1, y[i] - 1)) != s.end()) cnt++;
            if(s.find(make_pair(x[i] + 1, y[i] + 1)) != s.end()) cnt++;
            point[cnt]++;
        }
    }
    for(int i = 0; i < 5; i++) cout << point[i] << endl;
    return 0;
}
```