# Mục lục

# 1 Các kiến thức cần nắm

## 1.1 Mảng cộng dồn 1D

### 1.1.1 Định nghĩa

Gọi D[i] là tổng từ từ 1 đến $i$.

### 1.1.2 Ứng dụng

Cho một mảng $A$ có $n$ phần tử và $Q$ câu hỏi, mỗi câu hỏi là đoạn $[L,R]$, yêu cầu in ra tổng đoạn này. Nếu sử dụng thuật toán ngây ngô, mỗi câu hỏi ta sẽ thực hiện duyệt đoạn $[L,R]$ để tính tổng. Thuật toán này cho ta độ phức tạp $O(n*Q)$.
Tuy nhiên, với mảng cộng dồn ta có thể giảm độ phức tạp trên xuống còn $O(n+Q)$. Mục tiêu của chúng ta là tính tổng đoạn $[L,R]$, vậy ta tách đoạn ấy ra thành tổng đoạn $[1,R]$ (tức $D[R]$) và tổng đoạn $[1,L-1]$ (tức $D[L-1]$) từ đó tổng đoạn $[L,R]$=$D[R]$-$D[L-1]$.
Ví dụ minh họa

| A[i]= | 1 | 3 | 2 | 1 | 4 |
|---|---|---|---|---|---|
| D[i]= | 1 | 4 | 6 | 7 | 11 |

Tính tổng từ 2 đến 4, ta sẽ lấy $D[4]$-$D[1]$=$6=3+2+1$.

### 1.1.3 Cài đặt

Nhận thấy $D[i]$ là tổng từ từ 1 đến $i$, trong khi $D[i-1]$ tổng từ 1 đến $i-1$. Ta tính $D[i]=D[i-1]+A[i]$.

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;
int A[_size],D[_size],n;
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    for(int i=1; i<=n; i++){
        D[i]=D[i-1] + A[i];
    }
    return 0;
}
```

## 1.2 Tìm kiếm nhị phân trên mảng

### 1.2.1 Giới thiệu vấn đề

Cho một mảng $A$ có độ dài $n$ ($1 \le n \le 10^6$) đã được sắp xếp tăng dần và $Q$ ($1 \le Q \le 10^5$) câu hỏi, mỗi câu hỏi là số nguyên x, bạn cần kiểm tra $x$ có xuất hiện trong mảng hay không.

### 1.2.2 Giải pháp

Ta có giải pháp $O(Q * log(n))$ như sau, giả sử $x$ có trong mảng, nhận thấy mảng đã sắp xếp, nếu ta duyệt từ trái sang phải, phần tử $i$ có $A_i$ bé hơn $x$ thì chứng tỏ phần tử $i$ đang nằm bên trái của $x$ và ngược lại là nằm bên phải.

Từ đó, ta có thuật như sau giả sử đang tìm $x$ trên đoạn $[L,R]$, nếu vị trí ở giữa $mid = (L+R)/2$ có $A_{mid}$<x thì tức là $x$ đang nằm trong đoạn $[mid+1,R]$ ta bỏ đoạn $[L,mid]$ và ngược lại $A_{mid}$>x thì ta quan tâm đoạn $[L,mid-1]$.

Đây là ý tưởng của tìm kiếm nhị phân trên mảng nếu ta có thể xác định được phần tử $i$ nằm bên trái hay bên phải của giá trị cần tìm, ta có thể tìm nó trong $log_2$.

### 1.2.3 Cài đặt

Kiểm tra $mid$ đang nằm bên nào của $x$ và bỏ phần còn lại. độ phức tạp mỗi câu hỏi $O(log_2(n))$.

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;
int n,A[_size],l,r,mid,x,q,ans;
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    cin>>n>>q;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int query=1; query<=q; query++){
        cin>>x, l=1, r=n;
        while(l<=r){
        mid=(l+r)/2;
        if(A[mid]<=x){
            l=mid+1;
            ans=mid;
        }
        else r=mid-1;
        }
        if(A[ans]==x) cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
    }
    return 0;
}
```

## 1.3 Mảng hiệu 1D

### 1.3.1 Định nghĩa

Gọi $D[i]$= $A_i$ - $A_{i-1}$.

### 1.3.2 Ứng dụng

Mảng trên cho phép ta tăng 1 đoạn $[L,R]$ lên $k$ đơn vị trong $O(1)$.
Cụ thể hơn, khi tăng đơn vị từ $L$ đến $R$ lên $k$ đơn vị thì mảng $D$ chỉ thay đổi ở 2 vị trí $D[L]$ và $D[R+1]$, bởi vì nhận thấy khi tăng thì chênh lệch giữa 2 phần tử ngoài và trong đoạn $[L,R]$ đều không đổi, Chỉ có 2 vị trí giao giữa ngoài và trong (tức $D[L]$ và $D[R+1]$ là thay đổi thôi). Cụ thể hơn, tăng $D[L]$ lên $k$ và giảm $D[R+1]$ xuống k.
Ví dụ minh họa

| | | | | | |
|---|---|---|---|---|---|
| A= | 1 | 5 | 3 | 2 | 4 |
| D= | 1 | 4 | -2 | -1 | 2 |

Ta tăng đoạn [2,4] lên 2 đơn vị ta có mảng $D$ sau khi thay đổi là

| | | | | | |
|---|---|---|---|---|---|
| D= | 1 | 6 | -2 | -1 | 0 |

Muốn rút mảng $A$ từ mảng $D$ sau khi thay đổi, thì ta nhận thấy $D_1$ chính là $A_1$, tìm $A_2$ ta chỉ cần lấy $A_1$+$D_2$, tương tự với các phần tử còn lại.

### 1.3.3 cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;
int n,A[_size],l,r,k,q,D[_size];
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    cin>>n>>q;
    for(int i=1; i<=n; i++){
        cin>>A[i];
        D[i]=A[i]-A[i-1];
    }
    for(int i=1; i<=q; i++){
        cin>>l>>r>>k;
        D[l]=D[l]+k;
        D[r+1]=D[r+1]-k;
    }
    for(int i=1; i<=n; i++){
        A[i]=A[i-1]+D[i];
        cout<<A[i]<<' ';
    }
    return 0;
```

```
}
```

## 1.4 Mảng cộng dồn 2D

### 1.4.1 Định nghĩa

Gọi D[i][j] là tổng của hình chữ nhật có góc trái trên là $(1,1)$ và góc phải dưới là $(i,j)$.

### 1.4.2 Ứng dụng

Mảng trên cho phép ta tính tổng hình chữ nhật con bất kỳ nằm trong hình chữ nhật ban đầu.

Ví dụ:

| | | | | |
|---|---|---|---|---|
| A= | 1 | 3 | 3 | 4 |
| | 5 | 3 | 1 | 1 |
| | 4 | 2 | 1 | 1 |
| | 2 | 2 | 1 | 3 |

| | | | | |
|---|---|---|---|---|
| D= | 1 | 4 | 7 | 11 |
| | 6 | 12 | 16 | 21 |
| | 10 | 18 | 23 | 29 |
| | 12 | 22 | 28 | 37 |

Bây giờ nếu ta muốn tính tổng hình chữ nhật có góc trái trên $(2,2)$ và trái dưới là $(4,3)$, thực hiện như sau, ta lấy $D[4][3]$ thì ta cần bỏ tổng của những số đỏ.

| | | | | |
|---|---|---|---|---|
| D= | 1 | 4 | 7 | 11 |
| | 6 | 12 | 16 | 21 |
| | 10 | 18 | 23 | 29 |
| | 12 | 22 | 28 | 37 |

ta sẽ bỏ đi $D[4][1]$ và $D[1][3]$,

| | | | | |
|---|---|---|---|---|
| D= | 1 | 4 | 7 | 11 |
| | 6 | 12 | 16 | 21 |
| | 10 | 18 | 23 | 29 |
| | 12 | 22 | 28 | 37 |

| | | | | |
|---|---|---|---|---|
| D= | 1 | 4 | 7 | 11 |
| | 6 | 12 | 16 | 21 |
| | 10 | 18 | 23 | 29 |
| | 12 | 22 | 28 | 37 |

Nhận thấy bỏ đi như vậy thì vô tình $D[1][1]$ bị trừ đi 2 lần nên ta cần nạp lại 1 lượng $D[1][1]$;

| | | | | |
|---|---|---|---|---|
| D= | 1 | 4 | 7 | 11 |
| | 6 | 12 | 16 | 21 |
| | 10 | 18 | 23 | 29 |
| | 12 | 22 | 28 | 37 |

Đút kết tổng hình chữ nhật cần tìm = D[4][3]-D[4][1]-D[1][3]+D[1][1].
khái quát hơn ta có tổng của hình chữ nhật con bất kỳ có góc trái trên là $(x_1, y_1)$ góc phải dưới là $(x_2, y_2)$ sẽ bằng $D[x_2][y_2]$ - $D[x_2][y_1-1]$ - $D[x_1-1][y_2]$ + $D[x_1-1][y_1-1]$.

### 1.4.3 cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1007;
int n,m,A[_size][_size],xL,yL,xR,yR,q,D[_size][_size];
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    cin>>n>>m>>q;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D[i][j]= A[i][j] + D[i-1][j] + D[i][j-1] - D[i-1][j-1];
        }
    }
    for(int query=1; query<=q; query++){
        cin>>xL>>yL>>xR>>yR;
        cout<<D[xR][yR] - D[xR][yL-1] - D[xL-1][yR] + D[xL-1][yL-1]<<'\n';
    }
    return 0;
}
```

## 1.5 Mảng hiệu 2d

### 1.5.1 Định nghĩa

Gọi $D[i][j]$ là hiệu của $A[i][j]$ với $A[i][j-1]$.
Gọi $D2[i][j]$ là hiệu của $D[i][j]$ với $D[i-1][j]$.

### 1.5.2 Ứng dụng

Mảng hiệu 2D có công dụng là tăng một hình chữ nhật con bất kỳ lên $k$ đơn vị. khi tăng hình chữ nhật có góc trái trên là $(x_1, y_1)$ và phải dưới là $(x_2, y_2)$ lên $k$ thì quan sát thấy mảng $D2$ chỉ thay đổi giá trị ở 4 là 2 vị trí $(x_1, y_1)$, $(x_2+1, y_2+1)$ tăng lên $k$ và 2 vị trí $(x_1, y_2+1)$, $(x_2+1, y_1)$ giảm đi $k$ đơn vị.
Sau đó nếu như muốn tìm giá trị của $A[i][j]$ sau khi cập nhật thì
$A[i][j]$ = $D2[i][j]$ - $A[i-1][j]$ - $A[i][j-1]$ + $A[i-1][j-1]$.

### 1.5.3 cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1007;
int n,m,q,A[_size][_size],D[_size][_size],D2[_size][_size],a,b,c,d,k;
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D[i][j]=A[i][j]-A[i][j-1];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D2[i][j]=D[i][j]-D[i-1][j];
        }
    }
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>a>>b>>c>>d>>k;
        D2[a][b]+=k;
        D2[c+1][d+1]+=k;
        D2[a][d+1]-=k;
        D2[c+1][b]-=k;
```

```
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            A[i][j]=D2[i][j]+A[i-1][j]+A[i][j-1]-A[i-1][j-1];
            cout<<A[i][j]<<' ';
        }
        cout<<'\n';
    }
    return 0;
}
```

## 1.6 tìm kiếm nhị phân tìm đáp án

### 1.6.1 Giới thiệu vấn đề

Cho bạn 1 số $n$ ($1 \le n \le 10^{18}$), tím số chính phương nhỏ nhất mà lớn hơn hoặc bằng $n$.

### 1.6.2 Giải pháp

Trước hết ta biết 1 số chính phương là số $x$ mà tồn tại 1 số $y$ $x = y^2$. nhận thấy ta cần duyệt các $y$, và tìm ra $x = y^2$ tìm ra $x$ nhỏ nhất bé hơn $n$, với cách làm này sẽ chậm vì $y \le 10^9$.
Nhận thấy nếu số $y$ mà $y^2 \ge n$, thì những số $z > y$ đều thỏa mãn $z^2 > n$, ngược lại nếu $y^2 < n$ thì những số $z < y$ đều thỏa mãn $z^2 < n$. Vậy ta sẽ sử dụng tư tưởng tìm kiếm nhị phân.
Ví dụ ta biết được đáp án nằm trong đoạn $[L,R]$, ta sẽ kiểm tra số $mid = \frac{(L+R)}{2}$, nếu $mid^2 < n$ thì tức là đáp án đang nằm trong đoạn $[mid+1,R]$ ngược lại $mid^2 > n$ thì đáp án nằm trong đoạn $[L,mid-1]$.
Đây là ý tưởng của tìm kiếm nhị phân đáp án thường gặp ở các dạng bài tìm đáp án nhỏ nhất, lớn nhất thỏa điều kiện đề ra và ta có thể kiểm tra $mid$ đang nằm bên trái hay phải của đáp án thì có thể thực hiện thuật toán.

### 1.6.3 cài đặt

```
#include<bits/stdc++.h>
using namespace std;
long long ans,n,l,r,mid;
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    l=1;
    r=1000'000'000;
    while(l<=r){
        mid=(l+r)/2;
        if(mid*mid>=n){
            r=mid-1;
            ans=mid*mid;
        }
        else l=mid+1;
    }
    cout<<ans;
    return 0;
}
```

# 2 Hướng dẫn và đáp án của các bài

## 2.1 A - queryone

### 2.1.1 Định nghĩa

Gọi $sum[i]$ là tổng từ 1 đến $i$.

### 2.1.2 Giải pháp

Với mỗi truy vấn $L_i$ và $R_i$, đáp án sẽ là $sum[R_i] - sum[L_i - 1]$.

### 2.1.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n,q,l,r;
long long A[_size],sum[_size];

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    for(int i=1; i<=n; i++){
        sum[i]=A[i]+sum[i-1];
    }
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>l>>r;
        cout<<sum[r]-sum[l-1]<<endl;
    }
    return 0;
}
```

## 2.2 B - Querytwo

### 2.2.1 Giải pháp

Sắp xếp lại mảng, với mỗi truy vấn ta sẽ sử dụng tìm kiếm nhị phân trên mảng, nếu $A_{mid}$ lớn hơn $x$ thì đáp án cần tìm đang nằm bên trái, nhỏ hơn hoặc bằng thì nằm bên phải của $x$.

### 2.2.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int l,r,mid,A[_size],n,q,x,ans;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    sort(A+1,A+n+1);
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>x;
        l=1;
        r=n;
        ans=-1;
        while(l<=r){
            mid=(l+r)/2;
            if(A[mid]<=x){
                l=mid+1;
                ans=A[mid];
            }
            else r=mid-1;
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

## 2.3 C - Querythree

### 2.3.1 Định nghĩa

Gọi $sum[i]$ là tổng từ 1 đến $i$.

### 2.3.2 Giải pháp

Nhận thấy $j$ càng xa $i$ thì tổng từ $j$ đến $i$ càng lớn và ngược lại, càng gần thì càng bé.
Ta sẽ sử dụng tìm kiếm nhị phân nếu tổng $mid$ đến $i$ bé hơn bằng $k$ thì $j$ cần tìm đang nằm bên trái ngược lại thì đang nằm phải. Để tính tổng của đoạn $mid$ đến $i$ cho nhanh thì ta dùng mảng cộng dồn 1D.

### 2.3.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n,k,sum[_size],A[_size],l,r,mid,pos;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>k;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++) sum[i]=sum[i-1]+A[i];
    for(int i=1; i<=n; i++){
        l=1;
        r=i;
        pos=-1;
        while(l<=r){
            mid=(l+r)/2;
            if(sum[i]-sum[mid-1]<=k){
                pos=mid;
                r=mid-1;
            }
            else l=mid+1;
        }
        cout<<pos<<endl;
    }
    return 0;
}
```

## 2.4 D - DANKIEN1

### 2.4.1 Giải pháp

Nếu chú kiến đang nằm ở tọa độ $A_i$ bé hơn $x$ thì quãng đường cần đi là $x - A_i$, lớn hơn $x$ thì quãng đường cần đi là $A_i - x$.

### 2.4.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;

int n;
long long x,A[_size],ans;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>x;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++){
        if(A[i]<=x) ans=ans+x-A[i];
        else ans=ans+A[i]-x;
    }
    cout<<ans;

    return 0;
}
```

## 2.5 E - DANKIEN2

### 2.5.1 Định nghĩa

$sum[i]$ là tổng từ 1 tới $i$.

### 2.5.2 Giải pháp

Nhận thấy những chú kiến có tọa độ $A_j$ bé hơn $A_i$ thì đi quãng đường $A_i$-$A_j$, lớn hơn thì đi quãng đường $A_j$-$A_i$, nếu đang xét vị trí $i$ thì sẽ có $i-1$ chú kiến tọa độ bé hơn và $n-i$ chú kiến có tọa độ lớn hơn. Từ đây ta có đáp án cho mỗi vị trí $i$ là bằng $(i-1)*A_i - sum[i-1] + (sum[n]-sum[i]) - (n-i)*A[i]$.

### 2.5.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;

int n;
long long A[_size],sum[_size];

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    for(int i=1; i<=n; i++){
        sum[i]=A[i]+sum[i-1];
    }
    for(int i=1; i<=n; i++){
        cout<<(A[i]*(i-1)-sum[i-1])+(sum[n]-sum[i]-(n-i)*A[i])<<' ';
    }
    return 0;
}
```

## 2.6 F - Phương trình 1

### 2.6.1 Định nghĩa

$sum[i]$ là tổng từ 1 tới $i$.

### 2.6.2 Giải pháp

Duyệt mảng, với phần tử thứ $i$ ta giả sử $y = i$, giờ ta cần tìm $x$, nhận thấy $x$ càng xa $i$ thì tổng càng tăng, ngược lại càng gần thì càng giảm. Từ đây, sử dụng tìm kiếm nhị phân và mảng cộng dồn để kiểm tra có tồn tại $x$ sao cho tổng từ $x$ đến $i$ bằng $p$.

### 2.6.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n;
long long A[_size],sum[_size],p,l,r,mid,ans;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>p;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i];
    }
    for(int i=1; i<=n; i++){
        l=1;
        r=i;
        while(l<=r){
            mid=(l+r)/2;
            if(sum[i]-sum[mid-1]<p){
                r=mid-1;
            }
            else if(sum[i]-sum[mid-1]>p){
                l=mid+1;
            }
            else if(sum[i]-sum[mid-1]==p){
                ans=ans+1;
                break;
            }
        }
    }
```

```
    cout<<ans;
    return 0;
}
```

## 2.7 G - Phương trình 2

### 2.7.1 Định nghĩa

$sum[i]$ là tổng từ 1 tới $i$.

### 2.7.2 Giải pháp

Duyệt mảng, giả sử phần tử $i$ là $w$, từ ý tưởng bài $F$ ta đi tìm $z$ trước nếu tồn tại $z$ thỏa mãn tổng từ $z$ tới $i$ bằng $p$ thì ta sẽ đi tìm $y$ và nếu tồn tại $y$ thỏa mãn tổng từ $y$ tới $i$ bằng p+q ta sẽ đi tìm $x$ thỏa mãn tổng từ $x$ tới $i$ bằng p+q+k. Nếu ba giá trị $x$, $y$, $z$ thì bộ $x$, $y$, $z$, $i$ là 1 bộ thỏa mãn.

### 2.7.3 Cài đặt

để làm gọn code, mình sẽ sử dụng hàm $cnp(l,r,i,val)$ sẽ trả về vị trí $j$ sao cho tổng từ $j$ đến $i$ bằng $val$ và $j$ thuộc đoạn $l$ đến $r$.

```
        #include<bits/stdc++.h>
using namespace std;
const int _size=100'007;


int n;
long long A[_size],sum[_size],p,q,k,ans,x,y,z;

int cnp(int l, int r, int pos, int val){
    while(l<=r){
        int mid=(l+r)/2;
        if(sum[pos]-sum[mid-1]<val){
            r=mid-1;
        }
        else if(sum[pos]-sum[mid-1]>val){
            l=mid+1;
        }
        else if(sum[pos]-sum[mid-1]==val){
            return mid;
        }
    }
    return -1;
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>p>>q>>k;
    for(int i=1; i<=n; i++){
        cin>>A[i];
```

```
    }
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i];
    }
    for(int i=1; i<=n; i++){
        z=cnp(1,i,i,k);
        y=cnp(1,i,i,q+k);
        x=cnp(1,i,i,p+q+k);
        if(x!=-1 && y!=-1 && z!=-1){
            ans=ans+1;
        }
    }
    cout<<ans;
    return 0;
}
```

## 2.8 H - Thi nấu ăn

### 2.8.1 Giải pháp

Nhận thấy nếu bạn 1 mang đến 3 món ăn, bạn 2 mang đến 4 món và bạn 3 mang đến 2 món, thì lúc này trên bàn là 9 món, từ 1 đến 3 là của bạn 1, từ 4 đến 7 là của bạn 2 và từ 7 đến 9 của bạn 3.

Từ đây, ta định nghĩa $B_i$ là số hiệu món ăn cuối cùng của bạn $i$, như ví dụ trên thì $B = 3,7,9$. Giờ với mỗi $x$, để biết đây là món bạn nào ta cần tìm ra $i$ sao cho $B_i$ là giá trị nhỏ nhất lớn hơn hoặc bằng $x$ và bạn $i$ đó chính là chủ nhân của món ăn $x$. Để tìm cho nhanh ta sẽ dùng tìm kiếm nhị phân!

### 2.8.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n,m;
long long A[_size],sum[_size],x,l,r,mid,ans;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i];
    }
    cin>>m;
    for(int query=1; query<=m; query++){
        cin>>x;
        ans=-1;
        l=1;
        r=n;
        while(l<=r){
            mid=(l+r)/2;
            if(sum[mid]>=x){
                r=mid-1;
                ans=mid;
            }
            else l=mid+1;
        }
        cout<<ans<<' ';
    }
```

```
    return 0;
}
```

## 2.9 I - Đặt trạm phủ sóng

### 2.9.1 Giới thiệu

Kiểu dữ liệu pair<kiểu dữ liệu, kiểu dữ liệu> cho phép một biến được mang 2 giá trị (ví dụ pair<int,int> x). muốn truy cập vào giá trị thứ nhất thì để đuôi .first (ví dụ x.first), giá trị thứ hai thì để đuôi .second(ví dụ x.second).

Khi bạn muốn sắp xếp lại mảng theo một điều kiện khác không phải tăng dần, thì bạn có thể chèn thêm hàm cmp, ví dụ muốn sắp xếp giảm dần có thể làm như sau

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

int A[200],n;

bool cmp(int a, int b){
    return a>b;
}
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    sort(A+1,A+n+1,cmp);
    for(int i=1; i<=n; i++) cout<<A[i]<<' ';
    return 0;
}
```

lưu ý kiểu dữ liệu của a và b phải khớp với kiểu dữ liệu của mảng bạn sắp xếp.

### 2.9.2 Giải pháp

Sắp xếp lại mảng $A$ theo $X$ tăng dần, nhận thấy x tăng dần nên nếu như đặt $i$ là nơi phủ sóng và $j$ được phủ ($j < i$) thì $j+1$ cũng được phủ, cũng như nếu $j$ chưa được phủ thì $j-1$ cũng chưa được phủ.

Nếu muốn phủ đoạn $(j,i)$ thì cách tốt nhất là đặt máy phát sóng tại vị trí ở giữa $\frac{X_i - X_j + 1}{2}$, từ đây ta thấy nếu từ $j$ đến $i$ mà có khoảng cách bé hơn hoặc bằng $2 * k$ thì khi đặt máy phát sóng ở giữa $i$ và $j$ thì quãng đường xa nhất mà máy phát có thể phát tới trong đoạn từ $j$ đến $i$ sẽ bé hơn hoặc bằng $k$.

Đúc kết, với mỗi $i$ ta cần tìm $j$ xa $i$ nhất sao cho khoảng cách từ $j$ đến $i$ bé hơn hoặc bằng $2 * k$. Vậy nếu $i$ là khu được chọn để phát sóng cuối cùng bên phải, thì đoạn $[j,i]$ là số lượng tối đa người dân có sóng sử dụng mảng cộng dồn để tính nhanh hơn. Ta lấy giá trị lớn nhất trong tất cả các $i$ sẽ là đáp án.

### 2.9.3 Cài đặt

```
#include<bits/stdc++.h>
```

```
using namespace std;
const int _size=1000'007;

int n,k,l,r,mid,pos;
pair<int,long long> A[_size];
long long sum[_size],ans;

bool cmp(pair<int,long long> a,pair<int,long long> b){
    return a.first<b.first;
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>k;
    for(int i=1; i<=n; i++){
        cin>>A[i].first>>A[i].second;
    }
    sort(A+1,A+n+1,cmp);
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i].second;
    }
    for(int i=1; i<=n; i++){
        l=1;
        r=i;
        pos=-1;
        while(l<=r){
            mid=(l+r)/2;
            if(A[i].first-A[mid].first<=2*k){
                pos=mid;
                r=mid-1;
            }
            else l=mid+1;
        }
        ans=max(ans,sum[i]-sum[pos-1]);
    }
    cout<<ans;


    return 0;
}
```

## 2.10 J - Truyền hình

### 2.10.1 Giải pháp

Ta sắp xếp lại mảng theo thời điểm kết thúc của các phần tử (bạn có thể đọc bài I để rõ hơn cách sắp xếp cũng như cách lưu mảng).

Ta duyệt từ trái sang phải, khi xét tới $i$ nhận thấy một phần tử $j$ ($j < i$) sao cho $(i, j)$ là cặp không phù hợp khi và chỉ khi $s_i < t_j \leq t_i$. Vì các phần tử sắp xếp theo thời điểm kết thúc nên nhận thấy nếu $j$ và $i$ là cặp không hợp thì $j+1$ và $i$ cũng là cặp không hợp, nếu $j$ và $i$ là cặp hợp thì $j-1$ và $i$ cũng là cặp hợp. Từ đây với mỗi $i$ ta tìm $j$ xa $i$ nhất sao cho $s_i$ bé hơn $t_j$ (không cần kiểm tra $t_j \leq t_i$ vì mảng đã được sắp xếp theo $t$) đoạn từ $j$ đến $i-1$ là số lượng phần tử không phù hợp với phần tử $i$.

### 2.10.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=500'007;

int n,l,r,mid,pos;
pair<int,int> A[_size];
long long ans;

bool cmp(pair<int,int> a,pair<int,int> b){
    return a.second<b.second;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>A[i].first;
        cin>>A[i].second;
    }
    sort(A+1,A+n+1,cmp);
    for(int i=1; i<=n; i++){
        l=1;
        r=i;
        while(l<=r){
            mid=(l+r)/2;
            if(A[mid].second>A[i].first){
                pos=mid;
                r=mid-1;
            }
            else l=mid+1;
        }
        ans=ans+i-pos;
```

```
    }
    cout<<ans;
    return 0;
}
```

## 2.11 K - Cặp khán giả may mắn

### 2.11.1 Định nghĩa

$p_i$ là giá trị nhỏ nhất trong đoạn 1 đến $i$.

### 2.11.2 Giải pháp

Nhận thấy mảng $p_i$ là mảng giảm dần. nến ta duyệt mảng $A$, với mỗi $i$ ta sẽ sử dụng tìm kiếm nhị phân trên mảng $p$ để tìm ra $j$ xa $i$ nhất ($j < i$) thỏa mãn $A_i - p_j \geq P$, ta với mỗi cặp $i, j$ tìm được ta chọn ra $i - j$ lớn nhất là đáp án.

### 2.11.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size = 1e6 + 5;

int n,a[_size],p[_size],i,j,q,ma=0,mi=1e18;

int bns(int x){
    int ans = n + 1;
    int l=1;
    int r=n;
    while(l<=r){
        int mid = (l+r)/2;
        if(p[mid] <= x){
            ans=mid;
            r=mid-1;
        }
        else l=mid+1;
    }
    return ans;
}

int main(){
    ios::sync_with_stdio(0);cin.tie();
    cin >> n >> q;
    for(i = 1 ; i <= n ; i++) cin >> a[i];
    p[1] = a[1];
    for(i = 2 ; i <= n ; i++)
        p[i] = min(p[i-1] , a[i]);

    for(i = 1 ; i <= n ; i++)
        ma=max(ma , i - bns(a[i]-q));

    for(i = 1 ; i <= n ; i++){
        int t = bns(a[i]-q);
```

```
        if(i-t == ma){
            cout << t<< " " << i;
            return 0;
        }
    }
    cout << 0;

    return 0;

}
```

## 2.12 L - Vận chuyển

### 2.12.1 Giải pháp

Trường hợp $k >= n$ hoặc $k >= m$ tức là bạn có thể hủy nhiều hơn $n$ hoặc $m$ chuyến thì coi như món hàng không thể vận chuyển tới nơi (hủy $n$ chuyến $A_i$ hoặc $m$ chuyến $B_i$).
Ta duyệt mảng $A$ từ 1 đến $k+1$, với chuyến thứ $i$ ta coi như đã hủy $i-1$ chuyến trước đó và ta có thể hủy thêm $k-(i-1)$ chuyến $B$, vậy đầu tiên với mỗi $i$ trong mảng $A$, ta cần dùng tìm kiếm nhị phân để tìm chuyến $B_j$ nhỏ nhất ($>= T_a + A_i$), đây sẽ là chuyến đáng ra món hàng sẽ chuyển đi tuy nhiên vì vẫn đang còn có thể hủy thêm $k-(i-1)$ chuyến, vậy chuyến $B_{j+k-(i-1)}$ mới là chuyến món quà được chuyển, lưu ý $j+k-(i-1) > m$ thì cũng như món hàng không thể chuyển tới tay người nhận.

### 2.12.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size = 1e6 + 5;

int n,m,i,j,ta,tb,a[_size],b[_size],ma=0,k;
int ans;

int bns(int x){
      ans = k+1;
        int l=1;
        int r=m;
        while(l<=r){
            int mid = (l+r)/2;
            if(b[mid] < x){
                 l=mid+1;
            }
            else if(b[mid]>=x){
                ans = mid;
               r=mid - 1;
            }
        }
    return ans;
}

int main(){
    ios_base::sync_with_stdio(0);cin.tie();
    cin>>n>>m>>ta>>tb>>k;
    for(i=1;i<=n;i++) cin>>a[i];
    for(i=1;i<=m;i++) cin>>b[i];
    if(k>=n){
        cout<<"-1";
        return 0;
```

```
    }
    for(i=1;i<=k+1;i++){
        int t = bns(a[i]+ta);
        if(t+k-i+1 > m ){
            cout<<"-1";
            return 0;
        }
        ma=max(ma,b[t+k-i+1]+tb);
    }
    cout<<ma;
    return 0;
}
```

## 2.13 M - Chênh lệch

### 2.13.1 Định nghĩa

$sum_i$ là tổng từ 1 đến $i$.

### 2.13.2 Giải pháp

Với truy vấn $i$ gồm hai số $u$ và $v$, ta nhận thấy nếu $x$ sao cho $u \le x \le v$, để chọn để chia ra 2 dãy 1 đến $x$ và $x+1$ đến $n$.

Ta có nhận xét sau, nếu tổng từ $x+1$ đến $n$ lớn hơn tổng từ 1 đến $x$ thì $x$ là vị trí tối ưu chia dãy làm hai có chênh lệch nhỏ nhất trong tất cả các vị trí từ 1 đến $x$. Bởi vì muốn chênh lệch nhỏ thì 2 số càng gần nhau càng tốt, nhận thấy dù chọn vị trí $i$ nào trong $x$ vị trí trên thì, tổng của dãy sau luôn lớn hơn tổng trước, vậy ta càng làm cho tổng trước càng lớn càng tốt vì nó sẽ luôn bé hơn tổng sau và làm chênh lệch 2 dãy giảm dần vậy chọn $x$ là vị trí tối ưu nhất. Tương tự, ta cũng tìm một $x$ sao cho tổng dãy trước lớn hơn tổng dãy sau, thì vị trí $x$ là có chênh lệch nhỏ nhất trong các vị trí $x$ đến $n$ chứng minh như trên.

Từ đây, ta chặt nhị phân tìm vị trí xa nhất từ trái qua phải sao cho thỏa mãn tổng sau lớn hơn tổng trước đặt là $d$, vậy $d+1$ sẽ là vị trí xa nhất thỏa mãn tổng trước lớn hơn tổng sau tình từ phải qua trái.

Ta lấy chênh lệch nhỏ nhất trong hai vị trí $d$ và $d+1$ là đáp án của truy vấn.

### 2.13.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1e5+10;
const long long oo=1000'000'000'000'000'000;

int n,u,v,q,l,r,mid,pos;
long long A[_size],sum[_size],ans;

long cnp(int a, int b){
    l=a;
    r=b;
    pos=a-1;
    ans=oo;
    while(l<=r){
        mid=(l+r)>>1;
        if(sum[mid]-sum[a-1]<=sum[b]-sum[mid]){
            pos=mid;
            l=mid+1;
        }
        else r=mid-1;
    }
    if(pos) ans=min(ans,sum[b]-sum[pos]-(sum[pos]-sum[a-1]));
```

```
    if(pos!=b) ans=min(ans,sum[pos+1]-sum[a-1]-(sum[b]-sum[pos+1]));
    return ans;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>q;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i];
    }
    for(int query=1; query<=q; query++){
        cin>>u>>v;
        cout<<cnp(u,v)<<endl;
    }
    return 0;
}
```

## 2.14 N - GCD lớn nhất

### 2.14.1 Định nghĩa

$L_i$ là gcd của dãy từ 1 đến $i$.
$R_i$ là gcd của dãy từ $i$ đến $n$.

### 2.14.2 Giải pháp

Ước chung lớn nhất có tính "giao hoán" và "kết hợp", nôm na ta có mảng $A_1, A_2, A_3, A_4$, thì để tìm gcd của cả mảng bằng cách tìm $gcd(A_1, A_2)$ đặt là $a$ xong tìm $gcd(a, A_3)$ đặt $b$, và tìm $gcd(b, A_4)$ ngoài cách này ra tìm có thể tìm $gcd(A_1, A_2)$ và $gcd(A_3, A_4)$ sau đó tìm gcd của hai số mới cũng sẽ ra gcd của dãy.
Vậy khi xóa $k$ số liên tiếp từ $i$ đến $i+k$ và tìm gcd của dãy mới sau khi xóa. Thực chất là ta tìm gcd từ 1 đến $i-1$ và tìm gcd từ $i+k+1$ đến $n$, rồi sau đó ta tìm gcd của 2 số vừa tìm được là sẽ ra gcd của dãy sau khi xóa các phần tử từ $i$ đến $i+k$.
Vậy trước hết ta xóa $k$ số cuối hoặc xóa $k$ số đầu, $ans = max(L_{n-k}, R_{k+1})$, sau đó ta duyệt từ 1 đến $n-k-1$, với $i$ ta xóa các số từ $i$ đến $i+k$ vậy, ta cập nhật $ans = max(ans, gcd(L_i, R_{i+k+1}))$.

### 2.14.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'008;

int n,A[_size],ans,L[_size],R[_size],k;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>k;
    for(int i=1; i<=n; i++) cin>>A[i];
    L[1]=A[1];
    R[n]=A[n];
    for(int i=2; i<=n; i++) L[i]=__gcd(A[i],L[i-1]);
    for(int i=n-1; i>=1; i--) R[i]=__gcd(A[i],R[i+1]);
    ans=max({ans,L[n-k],R[k+1]});
    for(int i=1; i<n-k; i++){
        ans=max(ans,__gcd(L[i],R[i+k+1]));
    }
    cout<<ans<<' ';


    return 0;
}
```

## 2.15 O - Tạo dãy

### 2.15.1 Định nghĩa

$ma_i$ là giá trị lớn nhất $A_j$ sao cho $j$ thuộc đoạn từ 1 đến $i$.

### 2.15.2 Giải pháp

Ta duyệt mảng $B$, nhận thấy với mỗi $B_i$ cần tìm $A_j$ lớn nhất với $j <= i$, vậy với mỗi $i$ ta cập nhật $ma_i = max(ma_i, A_{i-1})$ sau đó, ta chỉ cần cập nhật $ans = max(ans, ma_i * B_i)$ là được!

### 2.15.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n;
long long ma[_size],A[_size],B[_size],ans;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++) cin>>B[i];
    for(int i=1; i<=n; i++){
        ma[i]=max(ma[i-1],A[i]);
    }
    for(int i=1; i<=n; i++){
        ans=max(ans,ma[i]*B[i]);
        cout<<ans<<'\n';
    }
    return 0;
}
```

## 2.16 P - 1DA

### 2.16.1 Định nghĩa

$D_i$ là hiệu của $A_i$ với $A_{i-1}$.

### 2.16.2 Giải pháp

Đây là ứng dụng cơ bản của mảng hiệu, như đã nêu ở trên: khi gặp truy vấn $l,r,x$ thì chỉ có chênh lệch của $A_l$ và $A_{l-1}$ và chênh lệch của $A_r$ và $A_{r+1}$ là thay đổi, cụ thể hơn các giá trị $D_l$ sẽ tăng lên $x$ đơn vị và $D_{r+1}$ sẽ giảm đi $x$ đơn vị.

### 2.16.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n,q,l,r;
long long A[_size],D[_size],k;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++) D[i]=A[i]-A[i-1];
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>l>>r>>k;
        D[l]+=k;
        D[r+1]-=k;
    }
    for(int i=1; i<=n; i++){
        A[i]=D[i]+A[i-1];
        cout<<A[i]<<' ';
    }
    return 0;
}
```

## 2.17 Q - Nhân mảng

### 2.17.1 Giải pháp

Với phần tử $i$, mình cần xác định em sau truy vấn, phần tử $i$ đã nhân với $-1$ bao nhiêu lần, nếu số lần là chẵn thì $A_i$ vẫn giữ nguyên (do ta biết rằng cứ chẵn số $-1$ nhân lại với nhau sẽ bằng 1), và ngược lại nếu số lần lẻ thì $A_i$ sẽ bị đổi dấu. Để đếm số lần, thì với mỗi truy vấn $i$ ta sẽ tăng đoạn $l$ đến $r$ lên một đơn vị, Vậy ta có thể ứng dụng mảng hiệu như bài $P$ để đếm số lần.

### 2.17.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n,q,l,r,A[_size],D[_size],B[_size];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>l>>r;
        D[l]+=1;
        D[r+1]-=1;
    }
    for(int i=1; i<=n; i++){
        B[i]=D[i]+B[i-1];
        if(B[i]%2==1) cout<<-A[i]<<' ';
        else cout<<A[i]<<' ';
    }
    return 0;
}
```

## 2.18 R - Du hành

### 2.18.1 Giải pháp

Để thực hiện yêu cầu bài toán, trước hết cần xác định năm thứ $i$ có bao nhiêu người sinh sống. Để làm điều đó, với người thứ $j$ ta biết rằng họ sống từ năm $L_j$ đến năm $R_j - 1$ vậy ta cần tăng số người sống từ năm $L_j$ đến $R_j - 1$, ta có thể ứng dụng mảng hiệu để tăng đoạn $L_j$ đến $R_j - 1$ lên một. Sau đó, ta duyệt vòng for đầu để tìm ra năm có nhiều người sinh sống nhất. Duyệt thêm vòng for để đếm số năm có nhiều người sinh sống bằng với năm có nhiều người sinh sống nhất.

### 2.18.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n,q,l,r,ans,A[_size],D[_size];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>l>>r;
        D[l]+=1;
        D[r]-=1;
    }
    for(int i=1; i<=1000'000; i++){
        A[i]=A[i-1]+D[i];
        ans=max(ans,A[i]);
    }
    for(int i=1; i<=1000'000; i++){
        if(ans==A[i]){
            cout<<i<<' '<<ans;
            return 0;
        }
    }
    return 0;
}
```

## 2.19 S - Thời điểm có nhiều đèn sáng nhất

### 2.19.1 Giải pháp

Với giây thứ $i$ ta cần xác định có bao nhiêu bóng đèn sáng trong thời điểm này. Vậy với mỗi bóng đèn $j$ ta cần tăng số bóng đèn phát sáng từ giấy $L_j$ đến $R_j$ lên một, Ta sẽ ứng dụng mảng hiệu để tăng đoạn $L_j$ đến $R_j$ lên một. Sau đó ta duyệt giây $i$ từ 1 đến $10^6$, tìm ra giây có số bóng đèn sáng cùng lúc nhiều nhất, sau đó duyệt tiếp $i$ từ 1 đến $10^6$ để đếm số lượng giây có số bóng đèn phát sáng bằng với giây có số lượng bóng đèn sáng cùng lúc nhiều nhất đã tìm ở trên.

### 2.19.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n,q,l,r,ans,A[_size],D[_size],dem;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>l>>r;
        D[l]+=1;
        D[r+1]-=1;
    }
    for(int i=1; i<=1000'000; i++){
        A[i]=A[i-1]+D[i];
        ans=max(ans,A[i]);
    }
    for(int i=1; i<=1000'000; i++){
        if(ans==A[i]){
            dem+=1;
        }
    }
    cout<<ans<<endl;
    cout<<dem;
    return 0;
}
```

## 2.20 T - Cấp số cộng 1

### 2.20.1 Giải pháp

‘ Trước hết, ta nhận thấy nếu đang ở phần tử $i$ đang được tăng bởi $x$ là số lượng cấp số cộng tác động lên $i$ (Ví dụ mảng $A$ có 5 phần tử có hai truy vấn [$l$=2,$r$=5,$x$=1] và [$l$=4,$r$=5,$x$=3] thì tại $i$=3 đang có 1 cấp số cộng tác động, tại $i$ =4 có 2 cấp số cộng tác động, tại $i$ bằng 5 cũng có 2 cấp số cộng tác động) và phần tử $i-1$ được nhận thêm một giá trị là $y$, vậy phần tử $i$ sẽ được tăng lên một giá trị là $y$+$x$, (Ví dụ tại $i-1$ đang nhận thêm một lượng là $3+4+5$ và có 3 cấp số động tác động lên $i$, vậy $i$ sẽ được nhận thêm giá trị là $4+5+6=(3+1)+(4+1)+(5+1)=3+4+5+3=y+x$. Khi ta kết thúc cấp số cộng $i$ ta cần trừ đi $y$ một lượng bằng $(r_i-l_i+1)$ và giảm $x$ đi 1. Vậy, trước hết ta cần biết phần tử $i$ đang bị bao nhiêu cấp số cộng tác động, với cấp số cộng $i$ ta tăng đoạn $L_i$ đến $R_i$ lên một, sử dụng mảng hiệu để tăng nhanh hơn, ta dùng mảng $H$ với $H_j$ là giá trị mà $y$ cần giảm khi duyệt $i=j$ , vậy truy vấn thứ $i$ ta tăng $H_{r_i+1}$ lên $r-l+1$ và giảm $y$=y-$H_i$.

### 2.20.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;

int n,q,l,r;
long long A[_size],D[_size],sum,H[_size],cnt;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>l>>r;
        D[l]+=1;
        D[r+1]-=1;
        H[r+1]+=(r-l+1);
    }
    for(int i=1; i<=n; i++){
        cnt=cnt+D[i];
        sum=sum+cnt-H[i];
        cout<<A[i]+sum<<' ';
    }
    return 0;
}
```

## 2.21 U - Cấp số cộng 2

### 2.21.1 Giải pháp

Ta xây dựng ý tưởng giống bài $T$, xác định với phần tử $i$ đang có bao nhiêu cấp số cộng tác động lên (với bạn này ta sẽ đặt là dem) và phần tử $i-1$ đang nhận thêm một lượng là $y$ và $i$ cũng nhận một lượng bằng $y$+$dem$. Tuy nhiên, với mỗi truy vấn $j$ ta cần tăng $y$ khi phần tử đang xét hiện tại $i = L_j$ một lượng bằng $x_i$ và giảm y khi $i = R_j + 1$ một lượng bằng $(r-l+1)+x_i$, phần còn lại ta xử lý như bài $T$.

### 2.21.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'007;

int n,q,l,r;
long long A[_size],D[_size],sum,H[_size],cnt,x;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>l>>r>>x;
        D[l+1]+=1;
        D[r+1]-=1;
        H[r+1]=H[r+1]+(r-l)+x;
        H[l]-=x;
    }
    for(int i=1; i<=n; i++){
        cnt=cnt+D[i];
        sum=sum+cnt-H[i];
        cout<<A[i]+sum<<' ';
    }
    return 0;
}
```

## 2.22 V - 2DQ

### 2.22.1 Định nghĩa

$sum_{ij}$ là tổng các phần tử thuộc hình chữ nhật có góc trái trên là 1,1 và phải dưới là $i,j$.

### 2.22.2 Giải pháp

Đây là ứng dụng cơ bản của mảng cộng dồn $2d$ đã nêu ở trên: ta xây dựng sum[i][j]=sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+A[i][j]. Với mỗi truy vấn $(a,b,c,d)$ ta in ra sum[c][d]-sum[a-1][d]-sum[c][b-1]+sum[a-1][b-1]. Để hiểu rõ vì sao các bạn có thể đọc lại lý thuyết đã ghi ở trên !!

### 2.22.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=507;


int n,m,q,x_1,y_1,x_2,y_2;
long long A[_size][_size],S[_size][_size];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m>>q;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            S[i][j]=A[i][j]+S[i-1][j]+S[i][j-1]-S[i-1][j-1];
        }
    }
    for(int query=1; query<=q; query++){
        cin>>x_1>>y_1>>x_2>>y_2;
        cout<<S[x_2][y_2] - S[x_2][y_1-1] - S[x_1-1][y_2] + S[x_1-1][y_1-1]<<'\n';
    }
    return 0;
}
```

## 2.23 W - 2DA

### 2.23.1 Giải pháp

Đây là ứng dụng cơ bản của mảng hiệu 2d đã nếu ở trên, bạn có thể đọc ở trên và áp dụng vào phần này nhé !!

### 2.23.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1007;

int n,m,q;
long long A[_size][_size],D[_size][_size],D2[_size][_size],a,b,c,d,k;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m>>q;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D[i][j]=A[i][j]-A[i][j-1];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D2[i][j]=D[i][j]-D[i-1][j];
        }
    }
    for(int query=1; query<=q; query++){
        cin>>a>>b>>c>>d>>k;
        D2[a][b]+=k, D2[c+1][d+1]+=k;
        D2[a][d+1]-=k, D2[c+1][b]-=k;
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            A[i][j]=D2[i][j]+A[i-1][j]+A[i][j-1]-A[i-1][j-1];
            cout<<A[i][j]<<' ';
        }
        cout<<'\n';
    }
    return 0;
}
```

## 2.24 X - Bonus

### 2.24.1 Định nghĩa

$S_{ij}$ là tổng các phần tử thuộc hình chữ nhật có góc trái trên là 1,1 và phải dưới là $i,j$.

### 2.24.2 Giải pháp

Ta duyệt hết toàn bộ các hình vuồng $k*k$, để làm điều đó ta sẽ duyệt $i,j$ là điểm trái trên của hình vuồng vậy $i+k-1,j+k-1$ sẽ là điểm phải dưới của hình vuông này !!,ta sẽ ứng dựng mảng cộng dồn 2d để tính tổng của hình vuông nhanh chóng.

### 2.24.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1007;


int n,k;
long long A[_size][_size],S[_size][_size],ans,sum;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>k;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            S[i][j]=A[i][j]+S[i-1][j]+S[i][j-1]-S[i-1][j-1];
        }
    }
    for(int i=1; i<=n-k+1; i++){
        for(int j=1; j<=n-k+1; j++){
            sum=S[i+k-1][j+k-1]-S[i-1][j+k-1]-S[i+k-1][j-1]+S[i-1][j-1];
            ans=max(ans,sum);
        }
    }
    cout<<ans;
    return 0;
}
```

## 2.25 Y - Chia đất

### 2.25.1 Định nghĩa

gọi $sum_{i,j}$ là tổng của hình chữ nhật có gốc trái trên là 1,1 và phải dưới là $i,j$.

### 2.25.2 Giải pháp

Ta duyệt mảng $A$ với $i$ từ 1 đến $n-1$ và $j$ từ 1 đến $n-1$, với cặp $(i,j)$ ta giả sử chia làm 4 phần cho 4 người con, người con 1 nhận hình chữ nhật có trái trên (1,1) và phải dưới $(i,j)$, người con 2 nhận hình chữ nhật có trái trên là $(1,j+1)$ và phải dưới $(i,n)$, người con 3 nhận hình chữ nhật có trái trên là $(i+1,1)$ và phải dưới là $(n,j)$, người con 4 nhận hình chữ nhật có trái trên là $(i+1,j+1)$ và phải dưới là $(n,n)$. Ta dùng mảng cộng dồn $2d$ để tính tổng của bốn hình chữ nhật này, sau đó cập nhật đáp án bằng chênh lệch của hình chữ nhật có tổng lớn nhất và bé nhất trong 4 hình vừa tìm được.

### 2.25.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=507;

int n,A[_size][_size],S[_size][_size],s1,s2,s3,s4,ans;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            S[i][j]=A[i][j]+S[i-1][j]+S[i][j-1]-S[i-1][j-1];
        }
    }
    ans=n*n;
    for(int i=1; i<n; i++){
        for(int j=1; j<n; j++){
            s1=S[i][j], s2=S[n][j]-S[i][j];
            s3=S[i][n]-S[i][j], s4=S[n][n]-(s1+s2+s3);
            ans=min(ans,max({s1,s2,s3,s4})-min({s1,s2,s3,s4}) );
        }
    }
    cout<<ans;
    return 0;}
```

## 2.26 Z - Checkin

### 2.26.1 Giải pháp

Ta sử dụng thuật tìm kiếm nhị phân để tìm đáp án, ta nhận thấy nếu thời gian là $x$ có thể giải quyết xong $m$ người thì $x+1$ cũng sẽ giải quyết được, ngược lại $x$ không thể giải quyết xong $m$ người thì $x-1$ cũng không giải quyết được. Vậy, hiện tại đang xét đoạn $L$ đến $R$ ta biết đáp án nằm trong phần này, ta sẽ kiểm tra $mid = \frac{R-l+1}{2}$ liệu có giải quyết cho $m$ người không, nếu có thì đáp án nằm trong đoạn $L$ đến $mid$ ngược lại đáp án nằm trong đoạn từ $mid$ đến $R$.

### 2.26.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n,m;
long long ans,A[_size],l,r,mid,cnt;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>A[i];
    l=1;
    r=1000'000'000'000'000'000;
    ans=r;
    while(l<=r){
        mid=(l+r)/2;
        cnt=0;
        for(int i=1; i<=n; i++){
            cnt+=mid/A[i];
        }
        if(cnt>=m){
            ans=mid;
            r=mid-1;
        }
        else l=mid+1;
    }
    cout<<ans;
    return 0;
}
```

## 2.27 ZA - Lồng đèn

### 2.27.1 Giải pháp

Sử dụng tìm kiếm nhị phân để tìm đáp án, ta nhận thấy nếu thời gian hoàn thành càng lớn thì số lượng robot cần dùng càng ít lại, vậy ta sẽ kiểm tra với thời gian hoàn thành là $mid$ thì cần bao nhiêu con robot để làm ra đủ lồng đèn, với lồng đèn loại $i$ có số lượng $a_i$, thì sẽ cần $\frac{a_i}{mid}$ robot, nếu không chia hết thì cần thêm 1 robot nữa. Vậy với mỗi $mid$ ta có thể đếm số lượng robot cần dùng. Ta sẽ sử dụng tìm kiếm nhị phân để giảm độ phức tạp. Nếu đáp án đang nằm trong đoạn $L$ đến $R$, ta kiểm tra thời gian $mid = \frac{L+R}{2}$ cần bao nhiêu robot, số robot lớn hơn hoặc bằng $n$ thì đáp án nằm trong đoạn $L$ đến $mid$ ngược lại, bé hơn thì nằm trong $mid+1$ đến $R$.

### 2.27.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n,m;
long long ans,A[_size],l,r,mid,cnt;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1; i<=m; i++) cin>>A[i];
    l=1;
    r=1000'000'000'000'000'000;
    ans=r;
    while(l<=r){
        mid=(l+r)/2;
        cnt=0;
        for(int i=1; i<=m; i++){
            cnt+=A[i]/mid;
            if(A[i]%mid>0) cnt+=1;
        }
        if(cnt<=n){
            ans=mid;
            r=mid-1;
        }
        else l=mid+1;
    }
    cout<<ans;
    return 0;
}
```

## 2.28 ZB - Bảng nhân 2

### 2.28.1 Giải pháp

Bài này ta dùng tìm kiếm nhị phân trên đáp án, ta cần viết một hàm để kiếm tra số $x$ đang đứng ở vị trí thứ bao nhiêu trong dãy, dễ dàng nhận thấy vị trí của nó đúng sẽ bằng số lượng số <=$x$.

Vậy đút kết ta cần đếm tồn tại bao nhiêu cặp $i * j <= x$, ta biến đổi lại thành $j <= \frac{x}{i}$, vậy ta for $i$ từ 1 đến $n$, với mỗi $i$ sẽ có $\frac{x}{i}$ số $j$ thỏa mãn $i * j <= x$, **Lưu ý:** vì $j$ chỉ nhận giá trị <= $m$ nên số cặp $j$ thỏa mãn là $min(\frac{x}{i}, m)$.

Vậy ta đã xây dựng xong hàm tìm vị trí của số $x$ trong mảng, ta thực hiện tìm kiếm nhị phân, nếu $mid$ có vị trí < k thì đáp án nằm phải của $mid$ ngược lại nằm bên trái.

### 2.28.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n,m;
long long k,cnt,ans,l,r,mid;
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m>>k;
    l=1;
    r=1000'000'000'000;
    while(l<=r){
        mid=(l+r)/2;
        cnt=0;
        for(int i=1; i<=n; i++){
            cnt+=min(1ll*m,mid/i);
        }
        if(cnt>=k){
            ans=mid;
            r=mid-1;
        }
        else l=mid+1;
    }
    cout<<ans;
    return 0;
}
```

## 2.29 ZC - Tấn và rượu

### 2.29.1 Định nghĩa

$sum_i$ là tổng các phần tử từ 1 đến $i$.

### 2.29.2 Giải pháp

Ta sắp xếp mảng tăng dần, ta ưu tiên đập vỡ các bình có $A_i$ lớn vì như thế ta sẽ hạn chế số lần đập bình (dung tích càng lớn ta càng dễ đổ vào các bình kia cho nó bằng nhau). Vậy ta duyệt i từ n->1, ta giả sử đã đập các bình $i, i+1, ..., n$ vậy hiện tại ta có lượng rượu bằng tổng các phần tử từ $i$ đến $n$, để kiểm tra lượng rượu này có rót đều các bình còn lại không ta cần tính toán xem lượng rượu tối thiểu để rót đều và kiểm tra lượng rượu đang có nếu lớn hơn hoặc bằng lượng tối thiểu thì $n-i+1$ là đáp án bài. Về cách tính lượng tối thiểu, ta nhân thấy tốt nhất rót sao cho các bình từ 1 đến $i-1$ có dung tích bằng $A_{i-1}$ vậy lượng tối thiểu bằng $A_{i-1}*(i-1)-$ (tổng từ $1->i-1$).

### 2.29.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=100'007;

int n;
long long A[_size],sum[_size],s,need;
bool OK=true;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=2; i<=n; i++) if(A[i] != A[i-1]) OK=false;
    if(OK){ cout<<0; return 0; }
    sort(A+1,A+n+1);
    for(int i=1; i<=n; i++) sum[i] = sum[i-1] + A[i];
    for(int i=n; i>1; i--){
        s=sum[n]-sum[i-1];
        need=A[i-1]*(i-1)-sum[i-1];
        if(s>=need){
            cout<<n-i+1;
            return 0;
        }
    }
    return 0;
}
```

## 2.30 ZD - DANKIEN

### 2.30.1 Định nghĩa

$sum_i$ là tổng của $A_j * B_j$ với $j$ từ 1 đến $i$.
$sl_i$ là số lượng chú kiến đừng ở vị trí $B_j$ ($B_j \leq B_i$).

### 2.30.2 Giải pháp

**Nhận xét:** vị trí cần chọn sẽ là 1 trong các số $B_i$. Với những vị trí $B_i$< x <$B_{i+1}$, ta có thể nháp sẽ nhận thấy tổng quãng đường đi cũng bằng các bạn chọn $x = B_i$ để đặt viên đường.

Vậy, ta duyệt $i$ từ 1 đến $n$, coi $B_i$ là vị trí đặt viên đường, ta cần tính tổng số quãng đường đi với độ phức tạp $O(1)$. Nhận thấy với những chú kiến vị trí $B_j$ ($j \leq i$) thì sẽ đi quãng đường là $B_i - B_j$, trong khi những chú kiến ở vị trí $B_j$ ($i < j$) sẽ đi quảng đường là $B_j - B_i$. Vậy với những $j \leq i$ tổng quãng đường đi được bằng $B_i * sl[i]$ - $sum(1->i)$, với những $j > i$ tổng quãng đường đi được là tổng quảng đường $sum(i+1->n) - B_i * (sl[n] - sl[i])$.

### 2.30.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n;
long long A[_size],B[_size],sum[_size],sl[_size],ans;


int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++) cin>>B[i];
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+A[i]*B[i];
        sl[i]=sl[i-1]+A[i];
    }
    ans=sum[n];
    for(int i=1; i<=n; i++){
        ans=min(ans, B[i]*sl[i]-sum[i] + (sum[n]-sum[i]) - B[i]* (sl[n]-sl[i]));
    }
    cout<<ans;
    return 0;
}
```

## 2.31 ZE - Solbin

### 2.31.1 Định nghĩa

$sum_{i,j}$ là tổng các phần tử trong hàng $i$ từ cột 1 đến cột $j$.

### 2.31.2 Giải pháp

**Nhận xét:** dù dịch sang phải hoặc từ trên xuống như nào thì tổng của 1 hàng bất kỳ trước và sau đều không đổi. Hay nói cách khác các phần tử trong cùng một hàng chỉ tráo vị trí cho nhau chớ không "lạc" sang hàng khác.

Nhận thấy, nếu mảng đã dịch sang phải $x$ lần, nếu $x$ là bội của $m$ thì coi như về lại lúc đầu, $x\%m = 1$ thì giống với việc ta dịch sang phải 1 lần, $x\%m = 2$ thì giống việc ta dịch sang phải 2 lần,... điều này cũng đúng với việc ta dịch mảng xuống $y$ lần, $y$ bội của $n$ thì coi như về lại mảng lúc đầu. Vậy ta gọi $D$ và $R$ là 2 giá trị xác định hàng $i$ trong mảng đầu là hàng $(i+D)\%n$ (nếu $= 0$ thì tức là hàng i lúc đầu là hàng $n$) của mảng hiện tại, tương tự với $R$ xác định cột $j$ trong mảng đầu là cột $(j+R)\%m$ (nếu $= 0$ thì tức là cột j lúc đầu là cột $m$) của mảng hiện tại.
Với mỗi truy vấn loại "0 $x$ $y$" thì $R = (R+x)\%m$ và $D = (D+y)\%n$.
Với truy vấn tính tổng ta duyệt $i$ từ $x_1$ đến $x_2$ ta sẽ tính tổng của từng hàng sau đó nạp vào đáp án. với hàng $i$ đầu tiên cần xác định hàng của nó trong mảng hiện tại, sau đó xác định cột $y_1$ và $y_2$ là cột nào trong mảng hiện tại, gọi hai cột tìm được lần lượt là $a$ và $b$, nếu $a <= b$ vậy tổng của hàng này sẽ bằng $sum_b - sum_{a-1}$ vì lúc này cột $y_1$ đứng trước $y_2$ trong mảng hiện tại. Ngược lại, nếu $a > b$ nhận thấy tổng sẽ bằng $(sum_n - sum_{a-1})$ + $sum_b$.

### 2.31.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=508;

int n,q,m,x_1,y_1,x_2,y_2,type,D,R,ans;
long long A[_size][_size],sum[_size][_size];

long long get(int i, int a, int b){
    i=(i+D)%n;
    if(i==0) i=n;
    a=(a+R)%m;
    if(a==0) a=m;
    b=(b+R)%m;
    if(b==0) b=m;
    if(b<a) return (sum[i][m] - sum[i][a-1] + sum[i][b]);
    return sum[i][b] - sum[i][a-1];
}
```

```
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            sum[i][j]=sum[i][j-1]+A[i][j];
        }
    }
    cin>>q;
    for(int query=1; query<=q; query++){
        cin>>type;
        if(type==0){
            cin>>x_1>>y_1;
            R=(R+x_1)%m;
            D=(D+y_1)%n;
        }
        else{
            cin>>x_1>>y_1>>x_2>>y_2;
            ans=0;
            for(int i=x_1; i<=x_2; i++){
                ans+=get(i,y_1,y_2);
            }
            cout<<ans<<'\n';
        }
    }
    return 0;
}
```

## 2.32 ZF - trapping

### 2.32.1 Định nghĩa

$L_i$ là giá trị lớn nhất trong các phần tử từ 1 đến $i$.
$R_i$ là giá trị lớn nhất trong các phần tử từ $i$ đến $n$.

### 2.32.2 Giải pháp

Sẽ khó nếu ta đi xác định từng vũng nước để cập nhật đáp án, với bài này ta cần xác định với mỗi cột sẽ chứa bao nhiêu nước.
Quan sát, nhận thấy thực ra lượng nước cột $i$ đựng sẽ bằng $min(L_i,R_i)-A_i$ hay nói cách khác cột $j$ sẽ nằm trong vũng nước có cột cao bên trái có độ cao là $L_i$ và cột cao bên phải độ cao là $R_i$. Để rút ra nhận xét, ta nhận thấy, nước mưa thì luôn dâng lên vậy nó sẽ phải dâng lên cột cao nhất có thể, nếu cột bên phải là $x < R_i$ thì nước mưa sẽ ngập được $x$ và dâng tới $R_i$ tương tự với $L_i$. và để nước mưa không trào ra, thì nước mưa chỉ dâng lên bằng $min(L_i,R_i)$.

### 2.32.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int n;
long long L[_size],R[_size],A[_size],ans;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++){
        L[i]=max(L[i-1],A[i]);
    }
    for(int i=n; i>=1; i--){
        R[i]=max(R[i+1],A[i]);
    }
    for(int i=1; i<=n; i++){
        ans+=min(L[i],R[i])-A[i];
    }
    cout<<ans;
    return 0;
}
```

## 2.33 ZG - Đào vàng

### 2.33.1 Giải pháp

Sử dụng tìm kiệm nhị phân đáp án, ta sẽ kiểm tra với bán kính là $x$ thì sẽ cần bao nhiêu lần đào, để đào hết $n$ thỏi vàng. Ta duyệt tọa độ của thỏi vàng từ 1 đến $n$, nhận thấy thỏi 1 sẽ là thỏi bị đào đầu tiên và vị trí tốt nhất để khoan đó là $A_1 + x$, ta càng kéo dài khoảng cách đào so với viên mình đào thì mình sẽ đào được nhiều vàng hơn, vậy khi đặt khoan tại $A_1 + x$ thì ta sẽ đào được các thỏi vàng trong đoạn từ $A_1$ đến $A_1 + 2 * x$, ta sử dụng tìm kiếm nhị phân tiếp để tìm ra vị trí $j$ xa nhất sao cho $A_j <= A_1 + 2 * x$, coi như thỏi vàng từ $1 -> j$ đã được đào, ta xét tiếp $j + 1$ lập lại thuật toán trên đào ở vị trí $A_{j+1} + x$, ăn các thỏi vàng từ $A_{j+1}$ đến $A_{j+1} + 2 * x$, rồi lặp lại như trên cho đến khi thu hoạch đủ $n$ thỏi. Vậy ta sẽ tính được số lần khoan tối thiểu với bán kính là $x$.
Vậy, ta sẽ sử dụng tìm kiếm nhị phân $mid = \frac{l+r}{2}$ và kiểm tra số lần đào với bán kính là mid, nếu $\leq k$ thì đáp án nằm bên trái $mid$, $> k$ thì đáp án nằm bên phải $mid$.

### 2.33.2 Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
const int _size = 1000'007;

int n , k;
long long a[_size], x[_size], pre[_size], prea[_size];

long long cnp(int coordinate){
    int l = 1 , r = n;
    long long result = -1;
    while(l <= r){
        int mid = (l + r) / 2;
        if(x[mid] <= coordinate){
            result = mid;
            l = mid + 1;
        }
        else r = mid - 1;
    }
    return result;
}


long long f(int R){
    int i = 1;
    long long cnt = 0;
    while(i <= n){
        int Right = x[i] + 2 * R;
        long long ind = cnp(Right);
        i = ind + 1;
```

```
        cnt++;
    }
    return cnt;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin >> n >> k;
    for(int i = 1 ; i <= n ; i++)
        cin >> x[i];
    x[n + 1] = 1000'000'007;
    sort(x + 1 , x + 1 + n);
    int l = 0 , r = 1000'000'000;
    long long  result = -1;
    while(l <= r){
        int mid = (l + r) / 2;
        if(f(mid) <= k){
            r = mid - 1;
            result = mid;
        }else l = mid + 1;
    }

    cout << result << endl;
    return 0;
}
```

## 2.34 ZH - Anh thợ điện may mắn

### 2.34.1 Giải pháp

Ta thực hiện tìm kiếm nhị phân kết quả, ta xây dựng hàm kiểm giá trị $x$ liệu có phải là sức mạnh tối thiểu của $n$ thành phố không. Để kiểm tra ta duyệt từ trái sang phải, nếu $A_i < x$, ta nhận thấy cần thêm $A_i - x$ nhà máy điện, vậy ta sẽ đặt $A_i - x$ này ở đâu là tối ưu, câu trả lời đó là ta sẽ đặt máy ở vị trí $i + r$ là tốt nhất. Bởi vì ta nhận thấy vì duyệt từ trái sang phải và $A_i < x$ đồng nghĩa với việc các $A_j >= x$ $(j < i)$ vậy ta cần đặt $x - A_i$ máy trạm điện càng xa càng tốt để các thành phố chưa xét được hưởng lợi nhiều nhất. Vậy khi đặt $x - A_i$ trạm tại vị trí $i + r$, thì các thành phố từ $i$ đến $i + 2 * r$ được tăng sức mạnh bằng $x - A_i$ để tăng đoạn $i$ đến $i + 2 * r$ một lượng $x - A_i$ ta sẽ ứng dụng mảng hiệu vào đây.
Vậy sau khi duyệt xong ta sẽ có được số lượng máy trạm cần thêm vào nếu như số lượng $<= k$ thì đáp án nằm bên phải $x$ ngược lại nằm bên trái $x$.

### 2.34.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=3000'007;
const long long oo=1000'000'000'000;

int n;
long long ans,k,v,l,r,mid,A[_size],B[_size],S[_size],sz,cnt,val;

bool check(long long a){
    for(int i=1; i<=n+1; i++) B[i]=0;
    cnt=0;
    sz=k;
    for(long long i=1; i<=n; i++){
        cnt+=B[i];
        if(S[min(i+v,1ll*n)]-S[max(0ll,i-v-1ll)]+cnt<a){
            val=a-(S[min(i+v,1ll*n)]-S[max(0ll,i-v-1)]+cnt);
            if(val>sz) return false;
            sz-=val;
            cnt+=val;
            B[min(i+2*v+1,1ll*n+1)]-=val;
        }
    }
    return true;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>v>>k;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int i=1; i<=n; i++) S[i]=A[i]+S[i-1];
```

```
    l=1;
    r=oo;
    while(l<=r){
        mid=(l+r)>>1;
        if(check(mid)){
            l=mid+1;
            ans=mid;
        }
        else{
            r=mid-1;
        }
    }
    cout<<ans;
    return 0;
}
```

## 2.35 Ma trận 0 1

### 2.35.1 Định nghĩa

$sum_{i,j}$ là số lượng ô bị cấm của hình chữ nhất trái trên là $(1,1)$ và phải dưới là $(i,j)$.

### 2.35.2 Giải pháp

Ta duyệt tất cả các hình chữ nhật $w*h$ từ trái qua phải từ trên xuống dưới, kiểm tra nếu như tất cả các ô trong hình chữ nhật này không có ô nào bị chiếm đóng thì ta sẽ đánh dấu toàn bộ hình chữ nhật. Sau đó ta duyệt lại và tìm xem nếu tồn tại một ô trống chưa đánh dấu thì đáp án là $NO$ ngược lại nếu tất cả ô trắng đều đánh dấu thì đáp án là $YES$.
Để kiểm tra coi hình chữ nhật có tồn tại ô trống không ta tính tổng của hình chữ nhật đó nếu tổng $> 0$ thì tức là có ô cấm. Ngược lại $= 0$ ta sẽ đánh dấu toàn bộ ô trong hình chữ nhật đó, để đánh dấu ta có thể sử dụng mảng hiệu $2d$ hoặc mảng cộng dồn $2d$. Với mảng cộng dồn $2d$, giả sử ta muốn đánh dấu hình chữ nhật có ô trái trên là $(x_1,y_1)$ và phải dưới là $(x_2,y_2)$, Ta tăng $D_{x_1,y_1}$ và $D_{x_2+1,y_2+1}$ lên 1 và giảm $D_{x_2+1,y_1}$ và $D_{x_1,y_2+1}$ đi 1. Sau đó, để biết ô $(i,j)$ có được đánh dấu không ta kiểm tra $D_{i,j} = D_{i-1,j} + D_{i,j-1} - Di-1,j-1 + D_{i,j} > 0$ thì tức ô này được đánh dấu.

### 2.35.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=1000'008;

int t,n,m,w,h,s,cur;
vector<int> A[_size],S[_size],B[_size];
bool OK;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>t;
    while(t--){
        OK=true;
        cin>>n>>m>>h>>w;
        for(int i=0; i<=n+1; i++){
            A[i].clear();
            S[i].clear();
            B[i].clear();
            for(int j=0; j<=m+1; j++){
                A[i].push_back(0);
                S[i].push_back(0);
                B[i].push_back(0);
            }
        }
```

```
        for(int i=1; i<=n; i++){
            for(int j=1; j<=m; j++){
                cin>>A[i][j];
            }
        }
        for(int i=1; i<=n; i++){
            for(int j=1; j<=m; j++){
                S[i][j]=A[i][j]+S[i-1][j]+S[i][j-1]-S[i-1][j-1];
            }
        }
        for(int i=h; i<=n; i++){
            for(int j=w; j<=m; j++){
                if(S[i][j]-S[i-h][j]-S[i][j-w]+S[i-h][j-w]==0){
                    B[i-h+1][j-w+1]++;
                    B[i+1][j-w+1]--;
                    B[i-h+1][j+1]--;
                    B[i+1][j+1]++;
                }
            }
        }
        for(int i=1; i<=n; i++){
            for(int j=1; j<=m; j++){
                B[i][j]+=B[i-1][j]+B[i][j-1]-B[i-1][j-1];
                if(!B[i][j] && !A[i][j]){
                    OK=false;
                    break;
                }
            }
        }
      if(OK) cout<<"YES";
      else cout<<"NO";
      cout<<'\n';
    }
    return 0;
}
```

## 2.36 ZJ - Cut cake

### 2.36.1 Định nghĩa

$sum_{i,j}$ là tổng của hình chữ nhật có góc trái trên là (1,1) và phải dưới là $(i,j)$.

### 2.36.2 Giải pháp

Ta duyệt $x_1$ từ 1 đến $n$, $x_2$ từ $x_1$ đến $n$, ta sẽ xét những hình chữ nhật có gốc trái trên có hàng là $x_1$ và phải dưới có hàng là $x_2$, ta duyệt j là cột từ 1 đến $m$, vơi mỗi cột $j$ ta tính tổng của hình chữ nhật có trái trên là $(x_1,1)$ và phải dưới là $(x_2,j)$. Bây giờ ta sẽ tìm một cột $z$ ($z <= j$) trước đó mà có tổng là âm, nhận thấy nếu ta bỏ phần âm này tức là lấy hình chữ nhật có trái trên là $(x_1,z+1)$ và phải dưới là $(x_2,j)$ thì tổng của ta sẽ được tăng lên, vậy ta cần chọn ra cột $z$ trước đó có tổng âm bé nhất có thể để tổng được tăng tối đa.
Đúc kết, cố định 2 hàng của hình chữ nhật, với mỗi cột $j$ tìm $z < j$ có tổng âm bé nhất, việc tìm tổng âm bé nhất ta có thể duy trì biến $mi$ để lưu min của các tổng ta đã chạy qua và bỏ phần tổng này đi để tổng ta có là lớn nhất nếu ko tồn tại tổng âm trước đó ta sẽ lấy tổng của hình chữ nhật hiện tại đang xét luôn, với các tổng tìm được ta lấy số lớn nhất chính là đáp án của bài.

### 2.36.3 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int _size=508;
const long long oo=1000'000'000'000'000;

int n,m;
long long A[_size][_size],sum[_size][_size],mi,ans,val;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
        }
    }
    for(int j=1; j<=m; j++){
        for(int i=1; i<=n; i++){
            sum[i][j]=A[i][j]+sum[i-1][j];
        }
    }
    ans=-oo;
    for(int x1=1; x1<=n; x1++){
        for(int x2=x1; x2<=n; x2++){
```

```
            val=0;
            mi=0;
            for(int y2=1; y2<=m; y2++){
                val=val+sum[x2][y2]-sum[x1-1][y2];
                ans=max(ans,val-mi);
                mi=min(mi,val);
            }
        }
    }
    cout<<ans;
    return 0;
}
```

## 2.37 ZK - Google

### 2.37.1 Giải pháp

Nhận thấy $n \leq 1000$, $k \leq 1000$, vậy với mỗi $k$ ta sẽ duyệt toàn bộ $i$ từ 1 đến $n$ với mỗi $i$ ta xác định đoạn nào sẽ được tăng lên một lượng bằng $b$, về vấn đề tăng một đoạn lên $b$ đơn vị ta sẽ sử dụng mảng hiệu để tăng nhanh. Về vấn đề xác định đoạn nào sẽ được tăng nhận thấy tại $i$=$x$ thì đoạn từ $y-r$ đến $y+r$ sẽ được tăng lên $b$, giờ nếu ta giảm $i$=$x-1$ ta nhận thấy đoạn cập nhật nhỏ lại và nằm trong đoạn $y-r$ đến $y+r$ giả sử đoạn đó là $l_1$ và $r_1$, tiếp tục giảm $i$=$x-2$ ta nhận thấy đoạn cần cập nhật cũng nhỏ lại và nằm trong đoạn $l_1$ và $r_1$ hay nói cách khác càng xa $x$ đoạn cập nhật càng nhỏ và nằm trong đoạn trước đó. Từ đó ta có ý tưởng sau: giả sử $i = x$ $L = y-r$ và $R = y+r$, giờ để tìm đoạn cập nhật cho $i-1$, ta sẽ hỏi khoảng cách từ $(y,x)$ đến $(R,i)$ nếu lớn hơn $r$ thì ta giảm $R$ đi 1 giảm đến khi nào khoảng cách $\leq r$ tương tự ta hỏi khoảng cách từ $(y,x)$ đến $(L,i)$ nếu lớn hơn $r$ ta tăng $L$ đến khi nào khoảng cách $\leq r$, vậy từ đó ta xác định được đoạn cập nhật cho $i-1$, ta sẽ dùng màng hiệu để tăng đoạn cho nhanh. Tương tự, ta cũng duyệt $i$ từ $x+1$ đến $n$ và làm với nhận xét như trên, $i$ càng xa $x$ thì đoạn cập nhật sẽ giảm và nằm trong đoạn trước đó.

Sau đó ta duyệt lại để cập nhật giá trị của mỗi ô $i,j$ và tìm ô có giá trị lớn nhất lẫn đếm số ô có giá trị như vậy.

### 2.37.2 Cài đặt

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

int D[30003][1008],n,m,k,x,y,r,b,ans,dem,d,L,R;
int cnt(int a, int b,int c,int d){
    return (c-a)*(c-a)+(d-b)*(d-b);
}
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>m>>n>>k;
    while(k--){
        cin>>x>>y>>r>>b;
        d=r;
        D[max(1,y-d)][x]+=b;
        D[min(m+1,y+d+1)][x]-=b;
        L=max(1,y-d);
        R=min(m,y+d);
        for(int i=x-1; i>=1; i--){
            while(L<=R && cnt(R,i,y,x)>r*r){
                R-=1;
            }
            while(L<=R && cnt(L,i,y,x)>r*r){
                L+=1;
```

```
            }
            if(L>R) break;
            D[R+1][i]-=b;
            D[L][i]+=b;
        }
        L=max(1,y-d);
        R=min(m,y+d);
        for(int i=x+1; i<=n; i++){
            while(L<=R && cnt(R,i,y,x)>r*r){
                R-=1;
            }
            while(L<=R && cnt(L,i,y,x)>r*r){
                L+=1;
            }
            if(L>R) break;
            D[R+1][i]-=b;
            D[L][i]+=b;
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            D[j][i]+=D[j-1][i];
        }
    }
    for(int i=1; i<=m; i++){
        for(int j=1; j<=n; j++){
            ans=max(ans,D[i][j]);
        }
    }
    for(int i=1; i<=m; i++){
        for(int j=1; j<=n; j++){
            if(D[i][j]==ans) dem++;
        }
    }
    cout<<ans<<'\n';
    cout<<dem;

    return 0;
}
```

## 2.38 ZL - Light

### 2.38.1 Giải pháp

Bài này có 2 trường hợp, một là đưa toàn bộ về màu đỏ, hai là toàn bộ ô về màu xanh, ta sẽ tính chi phí lần lượt cho từng trường hợp về lấy đáp án tốt nhất, nếu cả hai đều không có đáp án thì in ra -1. Giả sử cần biến tất cả các ô thành số $x$ ($x = 1$ hoặc $x = 2$), ta duyệt từ trái qua phải từ trên xuống dưới. Nhận thấy, nếu ô $(i, j)$ đang xét có giá trị khác $x$, nếu ô này có công tắc ta sẽ bật luôn nếu không coi như không có đáp án. Vậy tại sao ta không chọn những công tắc ở trên, bởi vì ta biết rằng để xét ô $(i, j)$, thì ta đã phải xét toàn bộ các ô trước $(i, j)$ tức là các ô phía trên đang bằng $x$, nên nếu ta bật công tắc ở các ô phía trên, thì sẽ có một số ô sẽ bị đổi trạng thái khác $x$ nên bắt buộc ta chỉ có thể bật công tắc tại ô $(i, j)$, khi bậc công tắt ô $(i, j)$ thì rõ ràng ta sẽ đổi trạng thái của hình chữ nhật ô công tắc đó quản lý, đổi trạng thái thực chất chỉ là tăng hình chữ nhật đó lên $b$ đơn vị nên ta có thể dùng mảng cộng dồn $2d$ để tăng nhanh hơn. Lưu ý, ta cần vừa xử lý vừa tăng hình chữ nhật chớ không phải tăng hết 1 lần rồi, mới duyệt lại để kiểm tra các ô có cùng trạng thái không.

### 2.38.2 Cài đặt

```
#include<bits/stdc++.h>
using namespace std;
const int oo=2000'000'000;

int n,m,k,a,b,c,d,x,ans,val,dem;
vector<int> A[100008],B[100008];
vector<pair<int,int>> D[100008];

int check(int a){
    for(int i=0; i<=n+1; i++){
        for(int j=0; j<=m+1; j++){
            B[i][j]=0;
        }
    }
    dem=0;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            B[i][j]+=B[i-1][j]+B[i][j-1]-B[i-1][j-1];
            if((A[i][j]+B[i][j])%3==a) continue;
            if(!D[(i-1)*m+j].empty()){
                c=D[(i-1)*m+j][0].first;
                d=D[(i-1)*m+j][0].second;
                val=(A[i][j]+B[i][j])%3;
                if(val<a){
                    val=a-val;
                }
                else{
```

```
                    val=3-val+a;
                }
                dem+=val;
                B[i][j]+=val;
                B[c+1][j]-=val;
                B[i][d+1]-=val;
                B[c+1][d+1]+=val;
            }
            else return oo;
        }
    }
    return dem;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m>>k;
    for(int i=1; i<=n; i++){
        A[i].push_back(0);
        for(int j=1; j<=m; j++){
            cin>>x;
            A[i].push_back(x);
        }
    }
    for(int i=0; i<=n+1; i++){
        for(int j=0; j<=m+1; j++){
            B[i].push_back(j);
        }
    }
    for(int i=1; i<=k; i++){
        cin>>a>>b>>c>>d;
        D[(a-1)*m+b].push_back({c,d});
    }

    ans=min(check(1), check(2));
    if(ans==oo) cout<<-1;
    else cout<<ans;


    return 0;
}
```

## 2.39 Bài ZI - MODULE

### 2.39.1 Nhận xét

Với bài toán này, ta có nhận xét sau:

+) Với giới hạn $n \le 10^3, m \le 10^3$ ta hoàn toàn có thể làm bằng Knapsack DP.

+) Với giới hạn lớn hơn, ta phải có cách làm khác.

Dựa vào việc quan sát, ta có tính chất như sau:

- Với $n \geq m$, luôn luôn tồn tại dãy con chia hết cho $m$.

**Chứng minh:**

- Ta sẽ dựa vào bài toán tìm đoạn con chia hết cho $m$ để chứng minh. Gọi $Pref[i]$ là tổng các phần tử từ $1 \rightarrow i$ chia dư cho $m$, ta nhận thấy nếu tồn tại $Pref[l] \% m = Pref[r] \% m$ thì $(l+1, r)$ là đoạn con có tổng chia hết cho $m$.

$\Rightarrow$ Tồn tại dãy con chia hết cho $m$ nếu tồn tại $Pref[l] \% m = Pref[r] \% m$.

$\Rightarrow$ nếu không tồn tại đoạn $Pref[l] \% m = Pref[r] \% m$ suy ra tất cả $Pref[i]$ phải phân biệt $\forall i$.

- Nhưng với định lý Dirichlet, với $n >= m$, sẽ luôn tồn tại ít nhất 1 cặp $Pref[l] \% m = Pref[r] \% m$.

$\Rightarrow$ Luôn tồn tại dãy con có tổng chia hết cho $m$ với $n \geq m$.

### 2.39.2 Định nghĩa DP

Gọi $DP[i][j]$ là xét tới phần tử thứ $i$ có tồn tại dãy con có tổng là $j$ chia dư cho $m$ hay không?

### 2.39.3 Suy ra công thức DP

Từ định nghĩa trên, giả sử $j$ là tổng tất cả các phần tử trong dãy con sau khi thêm $a[i]$, ta có các cách chuyển trạng thái sau:

+) Bỏ qua $a[i]$

$\Rightarrow DP[i][j] = DP[i-1][j]$

+) Thêm $a[i]$ vào dãy

$\Rightarrow DP[i][j] = DP[i-1][j - a[i]]$

Vì ta phải áp dụng các phép tính đồng dư nên có công thức QHĐ như sau:

$$DP[i][j] = DP[i-1][j] \mid DP[i-1][(j - a[i] + m) \% m]$$

### 2.39.4 Khởi gán

$DP[0][0] = true$

### 2.39.5 Đáp án

Đáp án của chúng ta được chia **2 trường hợp** như sau:

+) Nếu $n < m$, ta dùng DP với kết quả là YES nếu $DP[n][0] = true$ và ngược lại là NO.

+) Nếu $n >= m$, kết quả luôn là YES.

### 2.39.6 Code mẫu

```
#include<bits/stdc++.h>

using namespace std;

const int MAXN = 1e6 + 5 , MAXM = 1e3 + 5;

int n , m;
int a[MAXN];
bool dp[MAXM][MAXM];

int main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i] , a[i] %= m;

    if(n >= m){
        cout << "YES" << endl;
    }else{
        dp[1][a[1]] = true;
        for(int i = 2 ; i <= n ; i++){
            for(int j = 0 ; j < m ; j++)
                dp[i][j] = dp[i - 1][j] | dp[i - 1][(j - a[i] + m) % m];
            dp[i][a[i]] = true;
        }

        if(dp[n][0] == true)
            cout << "YES" << endl;
        else cout << "NO" << endl;
    }
    return 0;
}
```

## 2.40 ZN - DIVBOARD

### 2.40.1 Định nghĩa

gọi $sum_{i,j}$ là tổng của hình chữ nhật có gốc trái trên là $1,1$ và phải dưới là $i,j$.

### 2.40.2 Giải pháp

Ta duyệt mảng $A$ với $i$ từ 1 đến $n-1$ ta giả sử sẽ cặt ngang hình chữ nhật tại hàng $i$ chia làm 2 phần $A$ và $B$. Giờ mỗi phần ta cần tìm 2 cột để cắt, xét phần $A$, ta cần tìm một cột sao cho chênh lệch giữa 2 phần sau (2 phần của $A$) khi cắt cột đó sẽ là tối thiểu nhất có thể tương tư tại phần $B$ ta cũng tìm cột sao cho chênh lệch giữa 2 phần (2 phần của $B$) sau khi cắt cột đó sẽ tối thiểu.

Lúc này có 4 phần, ta sẽ lấy chênh lệch của lớn nhất với bé nhất và cập nhật vào ans và chuyển sang hàng tiếp theo.

Về vấn đề tại sao 2 cột được chọn chỉ cần đảm bảo tối thiểu chênh lệch của 2 phần sau khi cắt cột đó mà không quan tâm đến những phần còn lại.

Ta giả sử đáp án tối ưu là chọn ra 2 cột và chia làm 4 tổng là $a$, $b$, $c$, $d$ trong đó $a$ và $b$ là của phần $A$ , $c$ và $d$ là của phần $B$ sau khi cắt 2 cột ta có 2 trường hợp

Nếu $a \le c \le d \le b$ , giờ trong $A$ nó có cột khác chia $A$ thành $x$ và $y$ sao cho $a < x \le c \le d \le y < b$ thì ta sẽ lấy $x$ và $y$ thay vì $a$ và $b$ tức là ta càng giảm thiểu khoảng cách giữa 2 bạn càng nhiều càng tốt và để chọn $x$ và $y$ càng gần nhau ta cần giảm khoảng cách $c$ và $d$ về tối thiểu nhất. Tương tự nếu trong trường hợp $c \le a \le b \le d$.

Nếu $a \le c \le b \le d$ hoặc $a \le b \le c \le d$, ta nhận thấy ta sẽ phải tăng $a$ và giảm $d$ đi càng nhiều càng tốt tuy nhiên vẫn phải giữ được điều kiện $a \le b$ và $c \le d$, vậy tốt nhất đó là chọn ra 2 cột sao cho chênh lệch giữa hai $a$ với $b$ và $c$ với $d$ là tối thiểu nhất có thể.
Vậy trong hai trường hợp trên, ta đều đều thấy cần phải tối thiểu khoảng cách của $a$ và $b$, tối thiểu khoảng cách của $c$ và $d$ càng nhiều càng tốt luôn có lợi cho ta.

### 2.40.3 Cài đặt

```
        #include<bits/stdc++.h>
using namespace std;
const int _size=1007;
const int oo=1000'000'005;

int n,m,A[_size][_size],a,b,c,d,ans,diff;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
```

```
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>A[i][j];
            A[i][j]+=A[i-1][j]+A[i][j-1]-A[i-1][j-1];
        }
    }
    ans=oo;
    for(int i=1; i<n; i++){
        diff=oo;
        for(int j=1; j<m; j++){
            if(abs(A[i][m]-2*A[i][j])<diff){
                diff=abs(A[i][m]-2*A[i][j]);
                a=min(A[i][m]-A[i][j],A[i][j]);
                b=max(A[i][m]-A[i][j],A[i][j]);
            }
        }
        diff=oo;
        for(int j=1; j<m; j++){
            if(abs((A[n][m]-A[i][m])-2*(A[n][j]-A[i][j]))<diff){
                diff=abs((A[n][m]-A[i][m])-2*(A[n][j]-A[i][j]));
                c=min(A[n][j]-A[i][j],(A[n][m]-A[i][m])-(A[n][j]-A[i][j]));
                d=max(A[n][j]-A[i][j],(A[n][m]-A[i][m])-(A[n][j]-A[i][j]));
            }
        }
        ans=min(ans,max(b,d)-min(a,c));
    }
    cout<<ans;
    return 0;
}
```