RMQ 问题

范围最大值问题（Range Maximum Query,RMQ）：给定长为 n 的序列 a，有 m 次询问，对于每次询问，回答区间 $[l,r]$ 中的最大值。

ST 表概述

ST 表是用于解决 **可重复贡献问题** 的数据结构。

可重复贡献问题 是指对于运算 opt，满足 $x \text{ opt } x = x$，则对应的区间询问就是一个可重复贡献问题。例如，最大值有 $\max(x,x)=x$，gcd 有 $\gcd(x,x)=x$，所以 RMQ 和区间 gcd 就是一个可重复贡献问题。区间和就不具有这个性质，如果求区间和的时候采用的预处理区间重叠了，则会导致重叠部分被计算两次，这显然是错误的。另外，opt 还必须满足结合律才能使用 ST 表求解。

ST 算法

在 RMQ 问题中，著名的 ST 算法就是倍增的产物。给定一个长度为 N 的序列 A，ST 算法能在 $O(N\log N)$ 时间的预处理后，以 $O(1)$ 的时间复杂度在线回答 "序列 A 中下标在 l ~ r 之间的数的最大值是多少" 这样的区间最值问题。

一个序列的子区间个数显然有 $O(N^2)$ 个，根据倍增思想，我们首先在这个规模为 $O(N^2)$ 的状态空间里选择一些 2 的整数次幂的位置作为代表值。

设 $F[i,j]$ 表示序列 A 中下标在子区间 $[i,i+2^j-1]$ 里的数的最大值，也就是从 i 开始的 2^j 个数的最大值。递推边界显然是 $F[i,0]=A[i]$，即数列 A 在子区间 $[i,i]$ 里的最大值。

在递推时，我们把子区间的长度成倍增长，有公式

$$F[i,j]=\max\big(F[i,j-1],F[i+2^{j-1},j-1]\big)$$

即长度为 2^j 的子区间的最大值是左右两半长度为 2^{j-1} 的子区间的最大值中较大的一个。

```
void ST_prework()
{
    for(int i=1;i<=n;i++)
        f[i][0]=a[i];
    int t=log(n)/log(2)+1;
    for(int j=1;j<t;j++)
        for(int i=1;i<=n-(1<<j)+1;i++)
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
}
```

当询问任意区间 $[l,r]$ 的最值时，我们先计算出一个 k，满足 $2^k<r-l+1\le 2^{k+1}$，也就是使 2 的 k 次幂小于区间长度的前提下最大的 k。那么 "从 l 开始的 2^k 个数" 和 "以 r 结尾的 2^k 个数" 这两段一定覆盖了整个区间 $[l,r]$，这两段的最大值分别是 $F[l,k]$ 和 $F[r-2^k+1,k]$，二者中较大的那个就是整个区间 $[l,r]$ 的最值。因为求的是最大值，所以这两段只要覆盖区间 $[l,r]$ 即可，即使有重叠也没关系。

```
int ST_query(int l,int r)
{
    int k=log(r-l+1)/log(2);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
```

简便起见，我们在代码中使用了 cmath 库的 log 函数。该函数效率较高，一般来说对程序性能影响不大。更严格的讲，为了保证时间复杂度为 $O(1)$，应该 $O(N)$ 预处理出 1 ~ N 这 N 种区间长度各自对应的 k 值，在询问时直接使用。

P3865 【模板】ST表

给定一个长度为 N 的数列，M 次询问，求出每一次询问的区间内数字的最大值 $(1 \le N \le 10^5, 1 \le M \le 2 \times 10^6, a_i \in [0, 10^9])$。

代码

```
#include<bits/stdc++.h>
using namespace std;
const int SIZE=100100;
int a[SIZE],f[SIZE][50];
int n,m;
void ST_prework()
{
    for(int i=1;i<=n;i++)
        f[i][0]=a[i];
    int t=log(n)/log(2)+1;
    for(int j=1;j<t;j++)
        for(int i=1;i<=n-(1<<j)+1;i++)
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
}
int ST_query(int l,int r)
{
    int k=log(r-l+1)/log(2);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
int main()
{
    scanf("%d %d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    ST_prework();
    for(int i=1;i<=m;i++)
    {
        int l,r;
        scanf("%d %d",&l,&r);
        printf("%d\n",ST_query(l,r));
    }
    return 0;
}
```

二维RMQ

【例题】HDU-2888 Check Corners

给定一个 $n \times m$ 的矩阵，Q 次询问，每次询问以 (r_1, c_1) 为左上角，(r_2, c_2) 为右下角的最大值，并判断这个最大值是否出现在了这个子矩阵的四个角之中$(1 \le n, m \le 300, 1 \le Q \le 10^6)$。

暴力

每次花子矩阵大小的时间复杂度去查询，最坏时间复杂度为 $O(Q \times n \times m)$。

改进

用 n 棵线段树或者树状数组来维护每行区间最大值，最坏时间复杂度为 $O(n \times m \log m + Q \times n \log m)$。

分析

既然是静态的询问，没有修改操作，很容易想到处理 RMQ 问题中的 ST 表。

暴力的 ST 表和线段树的做法类似，预处理 n 个 ST 表，每次在多个 ST 表中查询最大值，最坏时间复杂度为 $O(n \times m \log m + Q \times n)$，仍然无法通过。

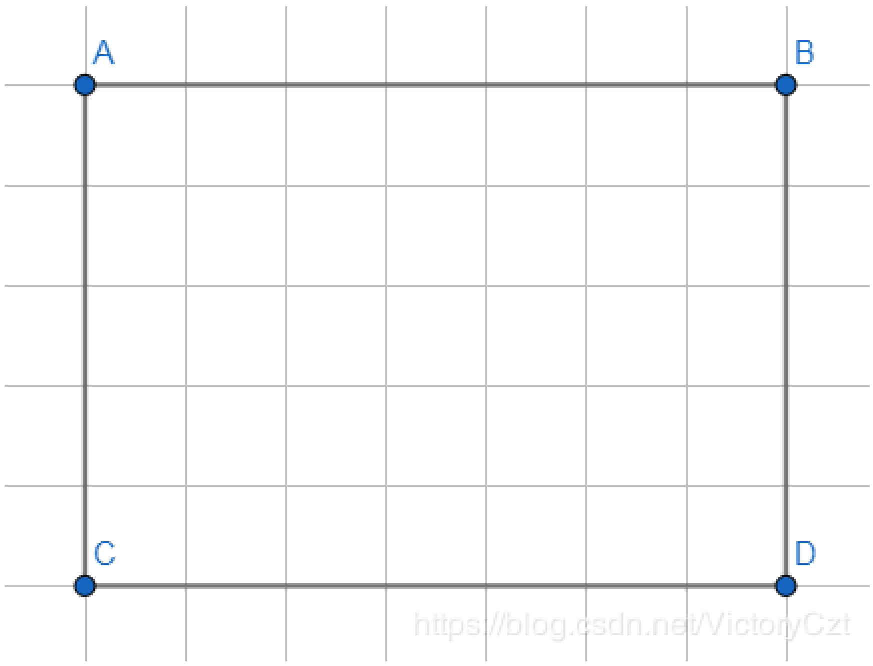
二维 ST 表

既然查询对象是一个二维矩阵，可以考虑维护一个二维 ST 表。

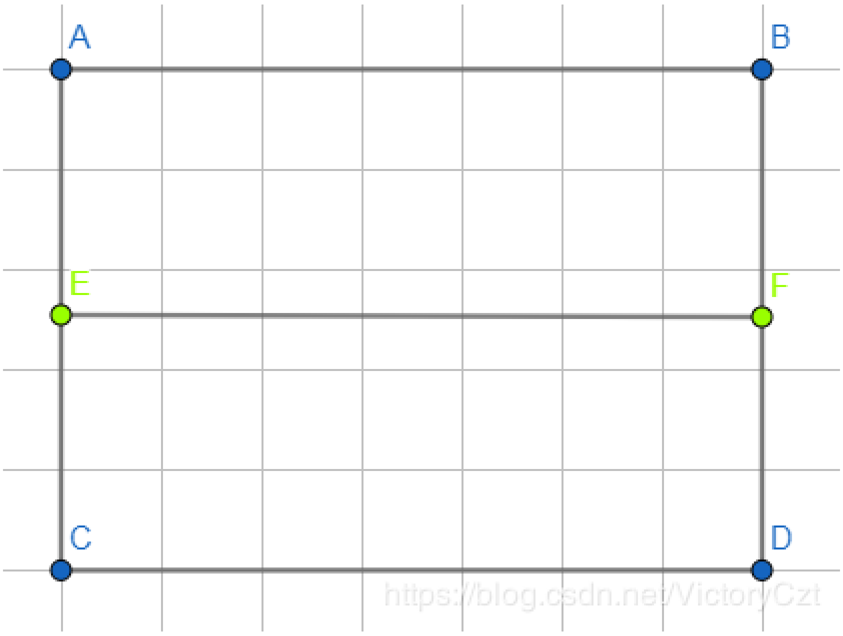
预处理

设 $f[i][j][k][l]$ 为以 (i, j) 为左上角，$(i + 2^k - 1, j + 2^l - 1)$ 的矩阵中的最大值。预处理的时间复杂度为 $O(n \times m \times \log n \times \log m)$，所以对询问数较少的问题时间复杂度为 $O(n \times m \log m)$ 的预处理更好一些。

下面来分析对于每个 $f[i][j][k][l]$ 可以由哪些状态更新。



设左上角的点 A 的坐标为为 (i, j)，右下角的点 D 的坐标为 $(i + 2^k - 1, j + 2^l - 1)$，那么可以分成两部分，如下图：



设点 E 的坐标为 $(i, j+2^{l-1})$，其实原来的大矩阵可以由分成的两个小矩阵更新得到得到，转移如下：

$$f[i][j][k][l] = \max\Big(f[i][j][k][l-1], f[i][j+2^{l-1}][k][l-1]\Big)$$

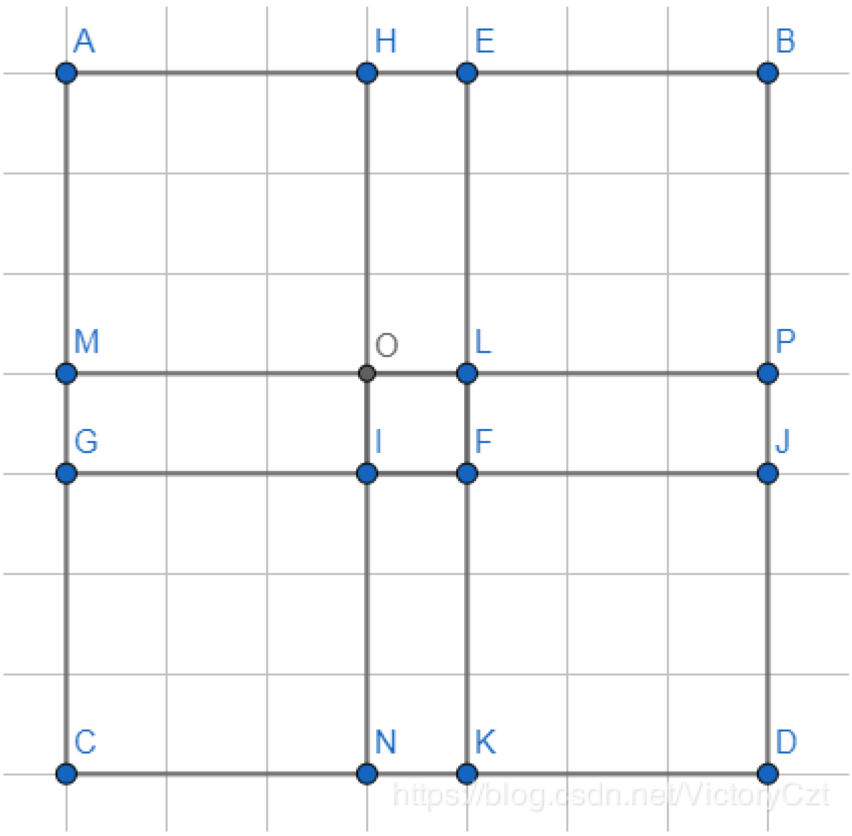
若 $l=0$，可以将矩形竖起剖成两部分，转移如下：

$$f[i][j][k][l] = \max\Big(f[i][j][k-1][l], f[i+2^{k-1}][j][k-1][l]\Big)$$

最后按照 k, l 从小到大更新即可。

查询

对于一个子矩阵，设左上角坐标为 (r_1, c_1)，右下角为 (r_2, c_2)，可以通过预处理的二维 ST 表，将其分成四部分查询，如下图：



四个部分分别为图中 $W_1(A,E,F,G), W_2(H,B,J,I), W_3(M,L,K,C), W_4(O,P,D,N)$。

令 $p=\log(r_2-r_1+1), q=\log(c_2-c_1+1)$，则:

$$ans=\max\begin{cases} f[r_1][c_1][p][q] & W_1 \\ f[r_2-2^p+1][c_1][p][q] & W_2 \\ f[r_1][c_2-2^q+1][p][q] & W_3 \\ f[r_2-2^p+1][c_2-2^q+1][p][q] & W_4 \end{cases}$$

总时间复杂度为 $O(n\times m\times\log n\times\log m+Q)$。

代码

```
#include<bits/stdc++.h>
using namespace std;
const int _log=9,N=305,INF=0x3f3f3f3f;
int n,m,Q;
int f[N][N][_log][_log],_log2[N],a[N][N];
init()
{
    _log2[2]=_log2[3]=1;
    for(int i=4;i<=301;i++)
    _log2[i]=_log2[i>>1]+1;
}
void RMQ()
{
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            f[i][j][0][0]=a[i][j];
    for(int k=0;(1<<k)<=n;k++)
    {
        for(int l=0;(1<<l)<=m;l++)
        {
            if(k+l!=0)
            {
                for(int i=1;i+(1<<k)-1<=n;i++)
                {
                    for(int j=1;j+(1<<l)-1<=m;j++)
                    {
                        if(k!=0)
                            f[i][j][k][l]=max(f[i][j][k-1][l],f[i+(1<<(k-
1))][j][k-1][l]);
                        else
                            f[i][j][k][l]=max(f[i][j][k][l-1],f[i][j+(1<<(l-
1))][k][l-1]);
                    }
                }
            }
        }
    }
}
int query(int r1,int c1,int r2,int c2)
{
    int p=_log2[r2-r1+1],q=_log2[c2-c1+1];
```

```
    int ans=-INF;
    ans=max(f[r1][c1][p][q],f[r1][c2-(1<<q)+1][p][q]);
    ans=max(ans,max(f[r2-(1<<p)+1][c1][p][q],f[r2-(1<<p)+1][c2-(1<<q)+1][p][q]));
    return ans;
}
int main()
{
    init();
    while(cin>>n>>m&&n&&m)
    {
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++)
                scanf("%d",&a[i][j]);
        RMQ();
        cin>>Q;
        while(Q--)
        {
            int r1,c1,r2,c2;
            scanf("%d %d %d %d",&r1,&c1,&r2,&c2);
            long long ans=query(r1,c1,r2,c2);
            bool flag=false;
            if(a[r1][c1]==ans||a[r1][c2]==ans||a[r2][c1]==ans||a[r2][c2]==ans)
                flag=true;
            printf("%d ",ans);
            if(flag)
                puts("yes");
            else
                puts("no");
        }
    }
    return 0;
}
```