前缀和 差分 离散化

powered by **LJL**

1 前缀和

前缀和是指某序列的前 n 项和，可以把它理解为数学上数列的前 n 项和

一维前缀和

例题

题目描述

输入一个长度为 n 的整数序列。

接下来再输入 m 个询问，每个询问输入一对 l, r。

对于每个询问，输出原序列中从第 l 个数到第 r 个数的和。

输入格式

第一行包含两个整数 n 和 m。

第二行包含 n 个整数，表示整数数列。

接下来 m 行，每行包含两个整数 l 和 r，表示一个询问的区间范围。

输出格式

共 m 行，每行输出一个询问的结果。

数据范围

$1 \leq l \leq r \leq n$

$1 \leq n, m \leq 100000$

$-1000 \leq$ 数列中元素的值 ≤ 1000

输入样例

```
5 3
2 1 3 6 4
1 2
1 3
2 4
```

输出样例

```
3
6
10
```

方法一：暴力枚举

可以看到暴力枚举算法的时间复杂度为 $O(nm)$，如果 n 和 m 的值稍微大一点就可能会超时。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int a[N];
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) //输入
        cin >> a[i];
    for (int i = 1; i <= m; i++) //每次遍历
    {
        int l, r;
        cin >> l >> r;
        int sum = 0;
        for (int j = l; j <= r; j++)
            sum += a[j];
        cout << sum << endl;
    }
    return 0;
}
```

方法二：前缀和算法

前缀和思路

首先对数组进行预处理，定义一个 sum 数组，其中 $sum[i]$ 表示数组 a 中前 i 个数的和。

即

$$sum[i] = a[1] + a[2] + a[3] + \ldots + a[i]$$

对于每次查询的 l 和 r，我们只需要执行 $sum[r] - sum[l-1]$ 即可，其时间复杂度为 $O(1)$。

原理：

$$sum[r] = a[1] + a[2] + a[3] + \ldots + a[r]$$
$$sum[l-1] = a[1] + a[2] + a[3] + \ldots + a[l-1]$$
$$sum[r] - sum[l-1] = a[l] + a[l+1] + a[l+2] + \ldots + a[r]$$

n 次预处理时间复杂度为 $O(n)$，m 次查询时间复杂度为 $O(m)$，所以总时间复杂度为 $O(n+m)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n, m;
int a[N];
int sum[N]; //前缀和数组
int main()
{
    cin >> n >> m;
```

```
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
        sum[i] = sum[i - 1] + a[i]; // 预处理前缀和数组
    while (m--)
    {
        int l, r;
        cin >> l >> r;
        cout << sum[r] - sum[l - 1] << endl; // 区间和的计算
    }
    return 0;
}

```

二维前缀和

例题

输入一个 n 行 m 列的整数矩阵，再输入 q 个询问，每个询问包含四个整数 x_1, y_1, x_2, y_2，表示一个子矩阵的左上角坐标和右下角坐标。

对于每个询问输出子矩阵中所有数的和。

输入格式

第一行包含三个整数 n, m, q。

接下来 n 行，每行包含 m 个整数，表示整数矩阵。

接下来 q 行，每行包含四个整数 x_1, y_1, x_2, y_2，表示一组询问。

输出格式

共 q 行，每行输出一个询问的结果。

数据范围

$1 \leq n, m \leq 1000$

$1 \leq q \leq 200000$

$1 \leq x_1 \leq x_2 \leq n$

$1 \leq y_1 \leq y_2 \leq m$

$-1000 \leq$ 矩阵内元素的值 ≤ 1000

输入样例：

```
3 4 3
1 7 2 4
3 6 2 8
2 1 2 3
1 1 2 2
2 1 3 4
1 3 3 4
```

输出格式

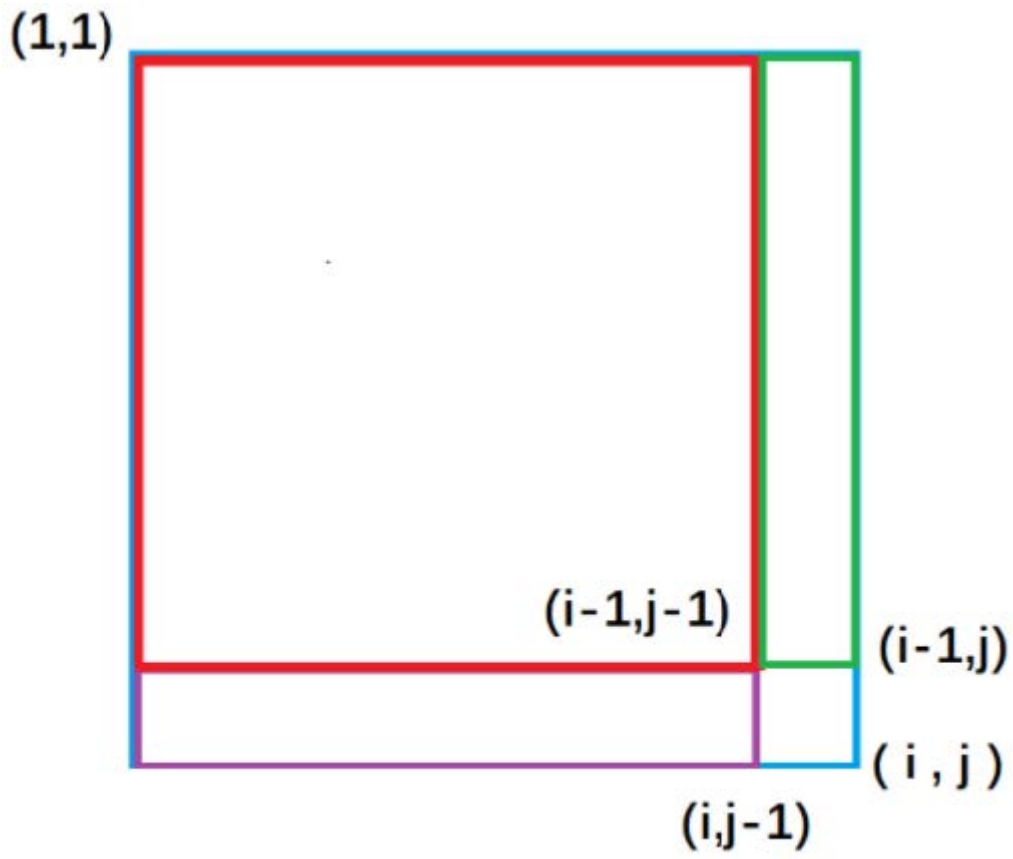
```
17
27
21
```

思路

同一维前缀和一样，我们先来定义一个二维数组 sum，其中 $sum[i][j]$ 代表表示一个左上角坐标为 $(1,1)$，右下角坐标为 (i,j) 的子矩阵

中所有元素的和。

接下来我们来推导一下二维前缀和的公式。



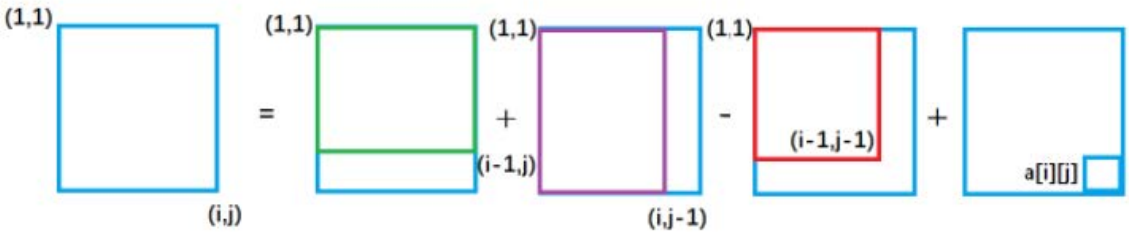
上图中，

紫色面积代表左上角为 $(1,1)$，右下角为 $(i,j-1)$ 的矩形的面积；

绿色面积代表左上角为 $(1,1)$，右下角为 $(i-1,j)$ 的矩形的面积；

红色面积代表左上角为 $(1,1)$，右下角为 $(i-1,j-1)$ 的矩形的面积。

每一个颜色的面积都代表它所包围的所有元素的和。



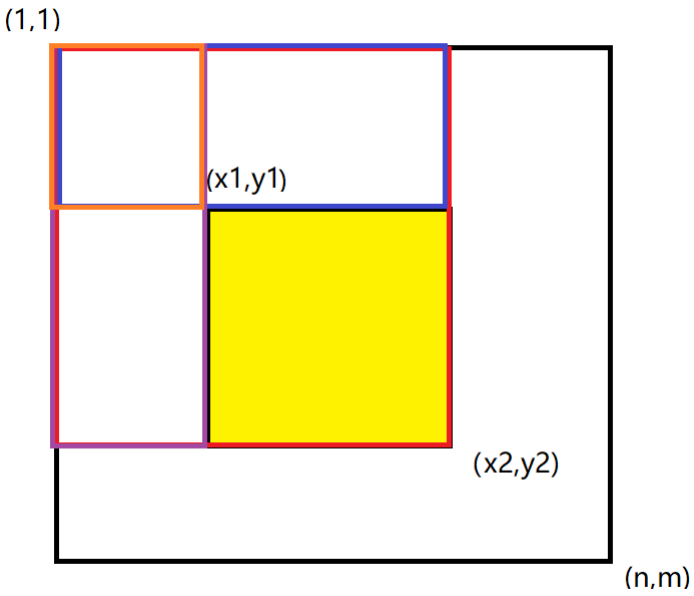
从上图中我们可以看出，整个外围蓝色矩形的面积 = 绿色面积 + 紫色面积 − 重复加过的红色面积 + 最后小蓝方块的面积 。

由此我们可以得出二维前缀和的预处理公式：

$$sum[i][j] = sum[i-1][j] + sum[[i][j-1] - sum[i-1][j-1] + a[i][j]$$

所以由上述推导可以得到二位前缀和的结论：

以 (x_1, y_1) 为左上角，(x_2, y_2) 为右下角的子矩阵元素的和为：



$$rns = sum[x_2][y_2] - sum[[x_1-1][y_2] - sum[x_2][y_1-1] + sum[x_1-1][y_1-1]$$

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3 + 5;
int a[N][N];
int sum[N][N]; //二维前缀和数组
int n, m, q;
int main()
{
    cin.tie(0);
    cin.sync_with_stdio(false);//输入同步关闭
    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++) //预处理二维前缀和数组
    {
        for (int j = 1; j <= m; j++)
        {
            sum[i][j] = sum[i - 1][j] + sum[i][j - 1] + a[i][j] - sum[i - 1][j - 1];
        }
    }
    while (q--)
    {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2; //二维前缀和结论求子矩阵中元素的和
```

```
        cout << sum[x2][y2] - sum[x2][y1 - 1] - sum[x1 - 1][y2] + sum[x1 - 1][y1 - 1] << "\n";
    }
    return 0;
}
```

2 差分

差分可以看成前缀和的逆运算。

接下来我们来介绍一下差分数组：

首先给定一个原数组 a，根据 a 构造差分数组 b，使得 $a[i] = b[1] + b[2] + \ldots + b[i]$。

也就是说，数组 a 是 b 数组的前缀和数组，所以反过来我们把数组 b 叫做数组 a 的差分数组。

接下来我们考虑该如何构造差分数组：

$$a[i] = b[1] + b[2] + \ldots + b[i]$$
$$a[i-1] = b[1] + b[2] + \ldots + b[i-1]$$
$$b[i] = a[i] - a[i-1]$$

特别的，$a[0] = 0$。

例题

输入一个长度为 n 的整数序列。

接下来输入 m 个操作，每个操作包含三个整数 l, r, c，表示将序列中 $[l, r]$ 之间的每个数加上 c。

请你输出进行完所有操作后的序列。

输入格式

第一行包含两个整数 n 和 m。

第二行包含 n 个整数，表示整数序列。

接下来 m 行，每行包含三个整数 l, r, c，表示一个操作。

输出格式

共一行，包含 n 个整数，表示最终序列。

数据范围

$1 \leq n, m \leq 100000$

$1 \leq l \leq r \leq n$

$-1000 \leq c \leq 1000$

$-1000 \leq$ 整数序列中元素的值 ≤ 1000

输入样例

```
6 3
1 2 2 1 2 1
1 3 1
3 5 1
1 6 1
```

输出样例

```
3 4 5 3 4 2
```

思路

我们可以构造数组 a 的差分数组 b，要始终记住 a 数组是 b 数组的前缀和数组，若我们对 b 数组中的元素 $b[i]$ 进行修改，这将会影响到 数组中 $a[i]$ 后面的每一个元素。

所以我们首先让 b 数组中的 $b[l]$ 加上 c，通过前缀和运算，a 数组变成了

$b[l]+c$ 效果是让 a 数组 $a[l]$ 及以后元素值都加上了 c。

$a[1],a[2],a[3],\ldots,a[l-1],a[l]+c,a[l+1]+c,\ldots,a[n]+c$

然后我们再让 b 数组中的 $b[r+1]$ 减去 c，通过前缀和运算， 数组变成了

$b[r+1]-c$ 效果是让 a 数组 $a[l+1]$ 及以后元素值都减去了 c。

$a[1],a[2],a[3],\ldots,a[l-1],a[l]+c,a[l+1]+c,\ldots,a[r]+c,a[r],a[r+1],\ldots,a[n]$

以上两种效果同时作用时就使得数组 a 在区间 l,r 元素值都加上了 c。

最后我们可以由此得出一维差分的结论：

给数组 a 中的 $[l,r]$ 区间中的每个数都加上 c，只需要对其差分数组 b 做 $b[l]+=c$ 和 $b[r+1]-=c$，再重新利用数组 b 求一遍前缀和即可得数组 a。

时间复杂度分析：m 次操作时间复杂度为 $O(m)$，最后构造前缀和数组的时间复杂度为 $O(n)$，总的时间复杂度为 $O(m+n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int a[N];
int b[N]; //定义差分数组
int n, m;
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        b[i] = a[i] - a[i - 1]; //构建差分数组
    }
    while (m--)
    {
        int l, r, c;
        cin >> l >> r >> c;
        b[l] += c;
        b[r + 1] -= c;
    }
    for (int i = 1; i <= n; i++)
        a[i] = a[i - 1] + b[i]; //利用差分数组重新建立前缀和数组
    for (int i = 1; i <= n; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
```

```
28 }
```

3 离散化

离散化，就是把无限空间中有限的个体映射到有限的空间中去，以此提高算法的时空效率。

通俗的说，离散化是在不改变数据相对大小的条件下，对数据进行相应的缩小。

例如：

原数据：1，999，100000，15；离散化处理后：1，3，4，2。

原数据：{100，200}，{20，50000}，{1，400}；离散化处理后：{3，4}，{2，6}，{1，5}。

离散化的原理和实现都很简单。为了确保不出错且尽可能地提高效率，我们希望离散化能实现以下几种功能：

1. 保证离散化后的数据非负且尽可能的小。
2. 离散化后各数据项之间的大小关系不变，原本相等的也要保持相等。

离散化一共有两种方法：一种方法是重复的元素经过离散化后数字相同，另一种方法是重复的元素经过离散化后数字不同。（这里我们介绍第一种方法）

重复元素离散化后数字相同

例如：对于序列 {105，35，35，79，-7} 经过离散化后变为 {4，3，2，1}。

基本步骤

1. 用一个辅助数组把所有要离散的数据存下来。
2. 辅助数组排序，排序是为了后面的二分。
3. 去重，因为我们要保证相同的元素离散化后数字相同。
4. 索引，再用二分把离散化后的数字放回原数组。

例题

假定有一个无限长的数轴，数轴上每个坐标上的数都是 0。

现在，我们首先进行 n 次操作，每次操作将某一位置 x 上的数加 c。

接下来，进行 m 次询问，每个询问包含两个整数 l 和 r，你需要求出在区间 $[l,r]$ 之间的所有数的和。

输入格式

第一行包含两个整数 n 和 m。

接下来 n 行，每行包含两个整数 x 和 c。

再接下来 m 行，每行包含两个整数 l 和 r。

输出格式

共 m 行，每行输出一个询问中所求的区间内数字和。

数据范围

$-10^9 \leq x \leq 10^9$

$1 \leq n, m \leq 10^5$

$-10^9 \leq l \leq r \leq 10^9$

$-10000 \leq c \leq 10000$

输入样例

```
3 3
1 2
3 6
7 5
1 3
4 6
7 8
```

输出样例

```
8
0
5
```

思路

由题意首先想到前缀和，但 x 的数据范围太大，开数组存储肯定不行。但是 n 和 m 的数据范围较小。n 次操作，每次使用一个下标，m 次查询，每次使用两个下标，也就是说使用的下标总数不会超过 $3 * 10^5$ ，所以我们开 $3 * 10^5$ 的数组。可以考虑先将要用到的所有数据下标存起来然后通过离散化进行处理。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 5;
int a[N], b[N]; //数组a存原式的值，数组b存对应的加数
int lsh[N];     //辅助数组
int sum[N];     //前缀和数组
int n, m;
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i] >> b[i];
        lsh[i] = a[i];
    }
    sort(lsh + 1, lsh + n + 1);
    int cnt = unique(lsh + 1, lsh + n + 1) - (lsh + 1); //去重
    for (int i = 1; i <= n; i++)
        sum[lower_bound(lsh + 1, lsh + cnt + 1, a[i]) - lsh] += b[i];
    //初始化sum数组
    for (int i = 1; i <= cnt; i++)
        sum[i] += sum[i - 1]; //初始化前缀和数组
    while (m--)
    {
        int l, r;
        cin >> l >> r;
        l = lower_bound(lsh + 1, lsh + cnt + 1, l) - lsh;
        r = upper_bound(lsh + 1, lsh + cnt + 1, r) - lsh - 1;
        cout << sum[r] - sum[l - 1] << endl;
```

```
    }
    return 0;
}
```

PS：有任何问题（包括文档错误、不懂的地方）可以联系**LJL**，**QQ：2834535844**