基础算法-快速排序

一、AcWing 785. 快速排序

【题目描述】

给定你一个长度为n的整数数列。

请你使用快速排序对这个数列按照从小到大进行排序。

并将排好序的数列按顺序输出。

【输入格式】

输入共两行，第一行包含整数n。

第二行包含n个整数（所有整数均在$1 \sim 10^9$范围内），表示整个数列。

【输出格式】

输出共一行，包含n个整数，表示排好序的数列。

【数据范围】

$1 \leq n \leq 100000$

【输入样例】

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

【输出样例】

```
1 2 3 4 5
```

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 100010;

int a[N];
```

```cpp
int n;

void quick_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int x = a[(l + r) >> 1], i = l - 1, j = r + 1;
    while (i < j)
    {
        do i++; while (a[i] < x);
        do j--; while (a[j] > x);
        if (i < j) swap(a[i], a[j]);
    }
    quick_sort(a, l, j);
    quick_sort(a, j + 1, r);
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    quick_sort(a, 0, n - 1);
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}
```

二、AcWing 786. 第k个数

【题目描述】

给定一个长度为n的整数数列，以及一个整数k，请用快速选择算法求出数列从小到大排序后的第k个数。

【输入格式】

第一行包含两个整数n和k。

第二行包含n个整数（所有整数均在$1 \sim 10^9$范围内），表示整数数列。

【输出格式】

输出一个整数，表示数列的第k小数。

【数据范围】

$1 \leq n \leq 100000$

$1 \le k \le n$

【输入样例】

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

【输出样例】

```
3
```

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 100010;
int a[N];
int n, k;

void quick_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int x = a[(l + r) >> 1], i = l - 1, j = r + 1;
    while (i < j)
    {
        do i++; while (a[i] < x);
        do j--; while (a[j] > x);
        if (i < j) swap(a[i], a[j]);
    }
    quick_sort(a, l, j);
    quick_sort(a, j + 1, r);
}

int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    quick_sort(a, 0, n - 1);
    printf("%d\n", a[k - 1]);

    return 0;
```

```
}
```

基础算法-归并排序

一、AcWing 787. 归并排序

【题目描述】

给定你一个长度为n的整数数列。

请你使用归并排序对这个数列按照从小到大进行排序。

并将排好序的数列按顺序输出。

【输入格式】

输入共两行，第一行包含整数n。

第二行包含n个整数（所有整数均在$1 \sim 10^9$范围内），表示整个数列。

【输出格式】

输出共一行，包含n个整数，表示排好序的数列。

【数据范围】

$1 \leq n \leq 100000$

【输入样例】

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

【输出样例】

```
1 2 3 4 5
```

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 100010;
int a[N], tmp[N];
```

```cpp
int n;

void merge_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(a, l, mid);
    merge_sort(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (a[i] <= a[j]) tmp[k++] = a[i++];
        else tmp[k++] = a[j++];
    while (i <= mid) tmp[k++] = a[i++];
    while (j <= r) tmp[k++] = a[j++];
    for (int i = l, j = 0; i <= r; i++, j++)
        a[i] = tmp[j];
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    merge_sort(a, 0, n - 1);
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}
```

二、AcWing 788. 逆序对的数量

【题目描述】

给定一个长度为n的整数数列，请你计算数列中的逆序对的数量。

逆序对的定义如下：对于数列的第i个和第j个元素，如果满足$i < j$且$a[i] > a[j]$，则其为一个逆序对；否则不是。

【输入格式】

第一行包含整数n，表示数列的长度。

第二行包含n个整数，表示整个数列。

【输出格式】

输出一个整数，表示逆序对的个数。

【数据范围】

$1 \leq n \leq 100000$

数列中的元素的取值范围$[1, 10^9]$。

【输入样例】

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

【输出样例】

```
5
```

【分析】

在归并排序的合并过程中，两个数组内的元素都为有序状态，因此若左半部分数组中的第一个元素$a[i]$大于右半部分数组的第一个元素$a[j]$，说明$a[i]$以及其后面的所有元素都是$a[j]$的逆序对（因为左半部分数组内部是有序的，$a[i]$之后的元素一定大于等于$a[i]$），故可以利用此性质在归并排序的过程中求解逆序对的数量。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = 100010;
int n, a[N], tmp[N];
LL res;

void merge_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(a, l, mid);
    merge_sort(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;

    while (i <= mid && j <= r)
```

```
        if (a[i] <= a[j]) tmp[k++] = a[i++];
        else
        {
            res += mid - i + 1;
            tmp[k++] = a[j++];
        }
    while (i <= mid) tmp[k++] = a[i++];
    while (j <= r) tmp[k++] = a[j++];
    for (int i = l, j = 0; i <= r; i++, j++)
        a[i] = tmp[j];
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    merge_sort(a, 0, n - 1);
    printf("%lld", res);
    return 0;
}
```

基础算法-二分

一、AcWing 789. 数的范围（整数二分）

【题目描述】

给定一个按照升序排列的长度为n的整数数组，以及q个查询。

对于每个查询，返回一个元素k的起始位置和终止位置（位置从0开始计数）。

如果数组中不存在该元素，则返回`-1 -1`。

【输入格式】

第一行包含整数n和q，表示数组长度和询问个数。

第二行包含n个整数（均在$1 \sim 10000$范围内），表示完整数组。

接下来q行，每行包含一个整数k，表示一个询问元素。

【输出格式】

共q行，每行包含两个整数，表示所求元素的起始位置和终止位置。

如果数组中不存在该元素，则返回`-1 -1`。

【数据范围】

$1 \leq n \leq 100000$

$1 \leq q \leq 10000$

$1 \leq k \leq 10000$

【输入样例】

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

【输出样例】

```
3 4
5 5
-1 -1
```

【分析】

二分模板题，首先数组a是有序的，某个数k出现的第一个位置为序列中大于等于这个数的第一个数，即二分查找这个位置mid，如果$a[mid] \geq k$，说明答案在mid的左半部分且包括mid，则$r = mid$，否则$l = mid + 1$。如果这个数不等于k，那么不存在这个数；k出现的最后一个位置为序列中大于等于这个数的最后一个数，即二分查找这个位置mid，如果$a[mid] \leq k$，说明答案在mid的右半部分且包括mid，则$l = mid$，否则$r = mid - 1$。

二分时首先无脑`mid = l + r >> 1`，然后如果`check`完后是`r = mid`则之前定义的`mid`不变且`else`中无脑`l = mid + 1`；若`check`完后是`l = mid`，则将之前的定义改为`mid = l + r + 1 >> 1`，且`else`中无脑`r = mid - 1`。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

```

```cpp
const int N = 100010;
int a[N];
int n, q, k;

int main()
{
    scanf("%d%d", &n, &q);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    while (q--)
    {
        scanf("%d", &k);
        //二分左端点，左端点的右边均大于等于端点值
        int l = 0, r = n - 1;
        while (l < r)
        {
            int mid = l + r >> 1;
            if (a[mid] >= k) r = mid;
            else l = mid + 1;
        }
        if (a[l] != k) printf("-1 -1\n");
        else
        {
            printf("%d ", l);
            //二分右端点，右端点的左边均小于等于端点值
            l = 0, r = n - 1;
            while (l < r)
            {
                int mid = l + r + 1 >> 1;
                if (a[mid] <= k) l = mid;
                else r = mid - 1;
            }
            printf("%d\n", l);
        }
    }
    return 0;
}
```

二、AcWing 790. 数的三次方根（浮点数二分）

【题目描述】

给定一个浮点数n，求它的三次方根。

【输入格式】

共一行，包含一个浮点数n。

【输出格式】

共一行，包含一个浮点数，表示问题的解。

注意，结果保留6位小数。

【数据范围】

$-10000 \leq n \leq 10000$

【输入样例】

```
1000.00
```

【输出样例】

```
10.000000
```

【分析】

二分查找答案，如果$mid^3 \geq res$，那么答案在左半区间，即$r = mid$，否则答案在右半区间，即$l = mid$。浮点数二分相比整数二分可以不需要注意那么多边界问题，只需要注意精度问题，一般来说二分的精度比答案的精度多两位，本题要求保留6位小数，那么我们二分的精度就为8位小数，即二分到$r - l < 10^{-8}$为止。

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

int main()
{
    double n;
    scanf("%lf", &n);
    double l = -10000, r = 10000;
    while (r - l > 1e-8)
    {
        double mid = (l + r) / 2;
        if (mid * mid * mid >= n) r = mid;
        else l = mid;
```

```
    }
    printf("%lf\n", l);
    return 0;
}
```

基础算法-高精度

一、AcWing 791. 高精度加法

【题目描述】

给定两个正整数（不含前导0），计算它们的和。

【输入格式】

共两行，每行包含一个整数。

【输出格式】

共一行，包含所求的和。

【数据范围】

$1 \leq$ 整数长度 ≤ 100000

【输入样例】

```
12
23
```

【输出样例】

```
35
```

【代码】

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

vector<int> add(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    int t = 0; //进位
```

```
    for (int i = 0; i < A.size() || i < B.size(); i++)
    {
        if (i < A.size()) t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }
    if (t) C.push_back(t);
    return C;
}

int main()
{
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--)
        A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i--)
        B.push_back(b[i] - '0');
    auto C = add(A, B);
    for (int i = C.size() - 1; i >= 0; i--)
        cout << C[i];
    return 0;
}
```

二、AcWing 792. 高精度减法

【题目描述】

给定两个正整数（不含前导**0**），计算它们的差，计算结果可能为负数。

【输入格式】

共两行，每行包含一个整数。

【输出格式】

共一行，包含所求的差。

【数据范围】

1 ≤ 整数长度 ≤ 100000

【输入样例】

```
32
11
```

【输出样例】

```
21
```

【代码】

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

//判断是否有A >= B
bool cmp(vector<int> &A, vector<int> &B)
{
    if (A.size() != B.size()) return A.size() > B.size();
    for (int i = A.size() - 1; i >= 0; i--)
        if (A[i] != B[i]) return A[i] > B[i];
    return true;
}

vector<int> sub(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    int t = 0; //借位
    for (int i = 0; i < A.size(); i++)
    {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    //去掉前导0
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

int main()
{
    string a, b;
```

```
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--)
        A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i--)
        B.push_back(b[i] - '0');
    if (cmp(A, B))
    {
        auto C = sub(A, B);
        for (int i = C.size() - 1; i >= 0; i--)
            cout << C[i];
    }
    else
    {
        auto C = sub(B, A);
        cout << '-';
        for (int i = C.size() - 1; i >= 0; i--)
            cout << C[i];
    }
    return 0;
}
```

三、AcWing 793. 高精度乘法

【题目描述】

给定两个非负整数（不含前导0）A和B，请你计算$A \times B$的值。

【输入格式】

共两行，第一行包含整数A，第二行包含整数B。

【输出格式】

共一行，包含$A \times B$的值。

【数据范围】

$1 \leq A$的长度 ≤ 100000

$0 \leq B \leq 10000$

【输入样例】

```
2

3
```

【输出样例】

```
6
```

【代码】

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

vector<int> mul(vector<int> &A, int b)
{
    vector<int> C;
    int t = 0; //进位
    for (int i = 0; i < A.size() || t; i++)
    {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    //去掉前导0
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

int main()
{
    string a;
    int b;
    vector<int> A;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--)
        A.push_back(a[i] - '0');
    auto C = mul(A, b);
    for (int i = C.size() - 1; i >= 0; i--)
        cout << C[i];
    return 0;
}
```

【双大数相乘代码】

```cpp
#include <iostream>
#include <string>
using namespace std;

const int N = 4010;
int num1[N], num2[N], res[N];
string str1, str2;
int len1, len2;

int main()
{
    cin >> str1 >> str2;
    for (int i = str1.size() - 1; i >= 0; i--)
        num1[len1++] = str1[i] - '0';
    for (int i = str2.size() - 1; i >= 0; i--)
        num2[len2++] = str2[i] - '0';
    for (int i = 0; i < len1; i++)
        for (int j = 0; j < len2; j++)
            res[i + j] += num1[i] * num2[j];
    for (int i = 0; i < len1 + len2; i++)
    {
        res[i + 1] += res[i] / 10;
        res[i] %= 10;
    }
    int idx = N - 1;
    while (idx > 0 && !res[idx]) idx--;//去前导0
    while (idx >= 0) cout << res[idx--];
    return 0;
}
```

四、AcWing 794. 高精度除法

【题目描述】

给定两个非负整数（不含前导0）A,B，请你计算A/B的商和余数。

【输入格式】

共两行，第一行包含整数A，第二行包含整数B。

【输出格式】

共两行，第一行输出所求的商，第二行输出所求余数。

【数据范围】

$1 \leq A$的长度 ≤ 100000

$1 \leq B \leq 10000$

B一定不为0

【输入样例】

```
7
2
```

【输出样例】

```
3
1
```

【代码】

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
using namespace std;

vector<int> div(vector<int> &A, int b, int &r)
{
    vector<int> C;
    r = 0; //余数
    for (int i = A.size() - 1; i >= 0; i--)
    {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    //由于除法从高位开始做，因此结果需要翻转
    reverse(C.begin(), C.end());
    //去掉前导0
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

int main()
{

    string a;
```

```
    int b;
    vector<int> A;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--)
        A.push_back(a[i] - '0');
    int r;
    auto C = div(A, b, r);
    for (int i = C.size() - 1; i >= 0; i--)
        cout << C[i];
    cout << endl << r;
    return 0;
}
```

基础算法-前缀和与差分

一、AcWing 795. 前缀和

【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
```

【代码】

```
#include <iostream>
using namespace std;

const int N = 100010;
int s[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) { cin >> s[i]; s[i] += s[i - 1]; }
    while (m--)
    {
        int l, r;
        cin >> l >> r;
        cout << s[r] - s[l - 1] << endl;
    }
    return 0;
}
```

二、AcWing 796. 子矩阵的和（二维前缀和）

【题目描述】

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】

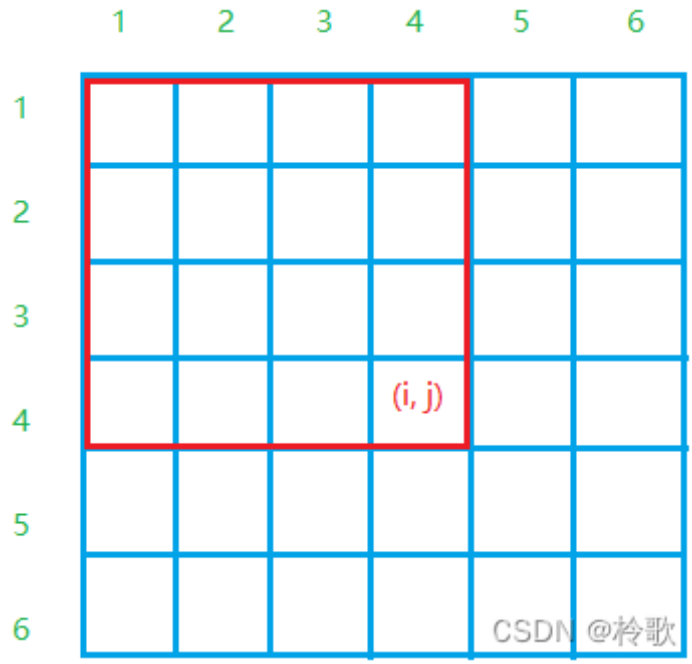
```
3 4 3
1 7 2 4
3 6 2 8
2 1 2 3
1 1 2 2
2 1 3 4
1 3 3 4
```

【输出样例】

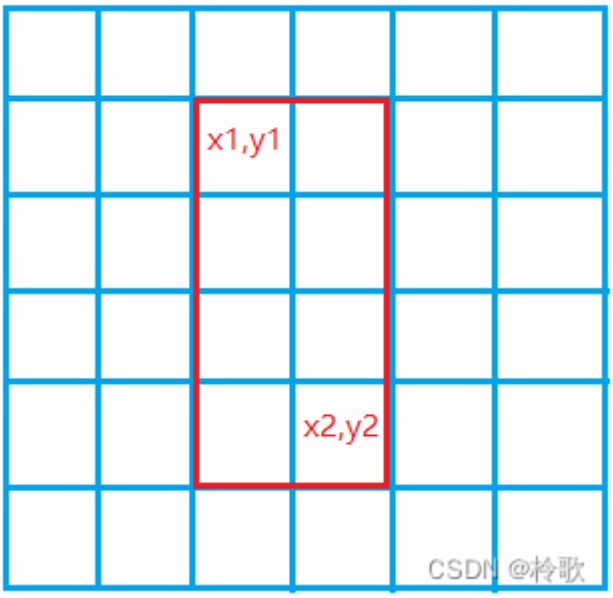
```
17
27
21
```

【分析】

设$s[i][j]$表示方块(i, j)及其左上角全部方块的数值之和，如下图所示，根据图片分析我们可以推出：$s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + a[i][j]$。



在查询的时候，同样画出示意图如下，根据图片分析我们可以推出：$res = s[x_2][y_2] - s[x_1 - 1][y_2] - s[x_2][y_1 - 1] + s[x_1 - 1][y_1 - 1]$。



【代码】

```
#include <iostream>
using namespace std;

const int N = 1010;
int s[N][N];
int n, m, q;

```

```cpp
int main()
{
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            cin >> s[i][j];
            s[i][j] += s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
        }
    while (q--)
    {
        int x1, x2, y1, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        cout << s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1 - 1] << endl;
    }
    return 0;
}
```

三、AcWing 797. 差分

【题目描述】

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
```

【代码】

```
#include <iostream>
using namespace std;

const int N = 100010;
int a[N], b[N];
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]; }
    while (m--)
    {
        int l, r, c;
        cin >> l >> r >> c;
        b[l] += c, b[r + 1] -= c;
    }
    for (int i = 1; i <= n; i++) { b[i] += b[i - 1]; cout << b[i] << ' '; }
    return 0;
}
```

四、AcWing 798. 差分矩阵（二维差分）

【题目描述】

输入一个n行m列的整数矩阵，再输入q个操作，每个操作包含五个整数x_1, y_1, x_2, y_2, c，其中(x_1, y_1)和(x_2, y_2)表示一个子矩阵的左上角坐标和右下角坐标。

每个操作都要将选中的子矩阵中的每个元素的值加上c。

请你将进行完所有操作后的矩阵输出。

【输入格式】

第一行包含整数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5个整数x_1, y_1, x_2, y_2, c，表示一个操作。

【输出格式】

共n行，每行m个整数，表示所有操作进行完毕后的最终矩阵。

【数据范围】

$1 \leq n, m \leq 1000$

$1 \leq q \leq 100000$

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

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

$-1000 \leq c \leq 1000$

$-1000 \leq$ 矩阵内元素的值 ≤ 1000

【输入样例】

```
3 4 3
1 2 2 1
3 2 2 1
1 1 1 1
1 1 2 2 1
1 3 2 3 2
3 1 3 4 1
```

【输出样例】

```
2 3 4 1
4 3 4 1
2 2 2 2
```

【代码】

```cpp
#include <iostream>
using namespace std;

const int N = 1010;
int b[N][N];
int n, m, q;

void add(int x1, int y1, int x2, int y2, int c)
{
    b[x1][y1] += c;
    b[x2 + 1][y1] -= c;
    b[x1][y2 + 1] -= c;
    b[x2 + 1][y2 + 1] += c;
}

int main()
{
    scanf("%d%d%d", &n, &m, &q);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            int x; scanf("%d", &x);
            add(i, j, i, j, x);
        }
    while (q--)
    {
        int x1, y1, x2, y2, c;
        scanf("%d%d%d%d%d", &x1, &y1, &x2, &y2, &c);
        add(x1, y1, x2, y2, c);
    }
    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];
            printf("%d ", b[i][j]);
        }
        puts("");
    }
    return 0;

}
```

基础算法-双指针算法

一、AcWing 799. 最长连续不重复子序列

【题目描述】

给定一个长度为n的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。

【输入格式】

第一行包含整数n。

第二行包含n个整数（均在$0 \sim 10^5$范围内），表示整数序列。

【输出格式】

共一行，包含一个整数，表示最长的不包含重复的数的连续区间的长度。

【数据范围】

$1 \leq n \leq 10^5$

【输入样例】

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

【输出样例】

```
3
```

【代码】

```cpp
#include <iostream>
using namespace std;

const int N = 100010;
int a[N], cnt[N];//数组cnt记录每个数出现的次数
int n;

int main()
{
    scanf("%d", &n);

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
```

```
    int res = 0;
    for (int i = 0, j = 0; i < n; i++)
    {
        cnt[a[i]]++;
        //不断移除左端点的数直到[j,i]内没有重复元素为止
        while (cnt[a[i]] > 1) cnt[a[j++]]--;
        res = max(res, i - j + 1);
    }
    printf("%d\n", res);
    return 0;
}
```

二、AcWing 800. 数组元素的目标和

【题目描述】

给定两个升序排序的有序数组A和B，以及一个目标值x。

数组下标从0开始。

请你求出满足$A[i]+B[j]=x$的数对(i,j)。

数据保证有唯一解。

【输入格式】

第一行包含三个整数n,m,x，分别表示A的长度，B的长度以及目标值x。

第二行包含n个整数，表示数组A。

第三行包含m个整数，表示数组B。

【输出格式】

共一行，包含两个整数i和j。

【数据范围】

数组长度不超过10^5。

同一数组内元素各不相同。

$1 \leq$ 数组元素 $\leq 10^9$

【输入样例】

```
4 5 6
1 2 4 7
3 4 6 8 9
```

【输出样例】

```
1 1
```

【代码】

```cpp
#include <iostream>
using namespace std;

const int N = 100010;
int a[N], b[N];
int n, m, x;

int main()
{
    scanf("%d%d%d", &n, &m, &x);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    for (int i = 0; i < m; i++) scanf("%d", &b[i]);
    for (int i = 0, j = m - 1; i < n; i++)
    {
        while (j >= 0 && a[i] + b[j] > x) j--;
        if (a[i] + b[j] == x)
        {
            printf("%d %d\n", i, j);
            break;
        }
    }
    return 0;
}
```

三、AcWing 2816. 判断子序列

【题目描述】

给定一个长度为n的整数序列$a_1, a_2, \ldots, a_n$以及一个长度为m的整数序列$b_1, b_2, \ldots, b_m$。

请你判断a序列是否为b序列的子序列。

子序列指序列的一部分项按原有次序排列而得的序列，例如序列$\{a_1, a_3, a_5\}$是序列$\{a_1, a_2, a_3, a_4, a_5\}$的一个子序列。

【输入格式】

第一行包含两个整数n, m。

第二行包含n个整数，表示$a_1, a_2, \ldots, a_n$。

第三行包含m个整数，表示$b_1, b_2, \ldots, b_m$。

【输出格式】

如果a序列是b序列的子序列，输出一行`Yes`。

否则，输出`No`。

【数据范围】

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

$-10^9 \leq a_i, b_i \leq 10^9$

【输入样例】

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

【输出样例】

```
Yes
```

【代码】

```cpp
#include <iostream>
#include <cstring>
using namespace std;

const int N = 100010;
int a[N], b[N];
int n, m;

int main()
{
    scanf("%d%d", &n, &m);

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
```

```
    for (int i = 0; i < m; i++) scanf("%d", &b[i]);
    int i = 0, j = 0;
    while (i < n && j < m)
    {
        if (a[i] == b[j]) i++;
        j++;
    }
    if (i == n) puts("Yes");
    else puts("No");
    return 0;
}
```

基础算法-位运算

一、AcWing 801. 二进制中1的个数

【题目描述】

给定一个长度为n的数列，请你求出数列中每个数的二进制表示中1的个数。

【输入格式】

第一行包含整数n。

第二行包含n个整数，表示整个数列。

【输出格式】

共一行，包含n个整数，其中的第i个数表示数列中的第i个数的二进制表示中1的个数。

【数据范围】

$1 \leq n \leq 100000$

$0 \leq$ 数列中元素的值 $\leq 10^9$

【输入样例】

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

【输出样例】

```
1 1 2 1 2
```

【分析】

我们可以写一个$lowbit(x)$函数返回x的二进制中最低位的1所表示的数，那么怎么实现这个功能呢？假设$x=(1100100)_B$，那么$-x=(0011011)_B+1=(0011100)_B$，则$x\&-x=(0000100)_B$，也就是$x$的最低位$1$所表示的数。然后回到本题，我们需要快速计算出某个数的二进制表示中有几个1，那我们就可以循环每次将这个数x减去$lowbit(x)$，也就是每次循环都把x的最低位1减去，每减一次答案加一，等这个数减为0时答案就是这个数中1的个数。

【代码】

```
#include <iostream>
using namespace std;

int lowbit(int x)
{
    return x & -x;
}

int main()
{
    int n;
    scanf("%d", &n);
    while (n--)
    {
        int x;
        scanf("%d", &x);
        int res = 0;
        //每次减去x的最后一位1
        while (x) x -= lowbit(x), res++;
        printf("%d ", res);
    }
    return 0;
}
```

基础算法-离散化

一、AcWing 802. 区间和

【题目描述】

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
```

【分析】

离散化的用处是将大范围的数据映射到一个小范围上，例如本题坐标的范围很大，无法使用数组表示，但是真正操作到的坐标数量不大，因此需要进行离散化。

不要求有序的离散化可以使用`unordered_map`实现，要求有序的离散化需要手写实现，主要步骤为排序与去重。

【代码】

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

typedef pair<int, int> PII;
const int N = 300010;
int a[N], s[N];
vector<int> alls;//所有需要离散化的下标
vector<PII> add, query;
int n, m;

//求x离散化后的值
int find(int x)
{
    int l = 0, r = alls.size() - 1;
    while (l < r)
    {
        int mid = l + r >> 1;
        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1;//映射到1~n
}

int main()
{
    cin >> n >> m;
    while (n--)
    {
        int x, c;
        cin >> x >> c;
        add.push_back({ x, c });
        alls.push_back(x);
    }
```

```
    while (m--)
    {
        int l, r;
        cin >> l >> r;
        query.push_back({ l, r });
        alls.push_back(l);
        alls.push_back(r);
    }
    //去重
    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());
    for (auto item : add)
    {
        int x = find(item.first);
        a[x] += item.second;
    }
    //预处理前缀和(由于映射到1~n，因此i<=alls.size())
    for (int i = 1; i <= alls.size(); i++) s[i] = s[i - 1] + a[i];
    //查询
    for (auto item : query)
    {
        int l = find(item.first), r = find(item.second);
        cout << s[r] - s[l - 1] << endl;
    }
    return 0;
}
```

基础算法-区间合并

一、AcWing 803. 区间合并

【题目描述】

给定n个区间$[l_i, r_i]$，要求合并所有有交集的区间。

注意如果在端点处相交，也算有交集。

输出合并完成后的区间个数。

例如：$[1, 3]$和$[2, 6]$可以合并为一个区间$[1, 6]$。

【输入格式】

第一行包含整数n。

接下来n行，每行包含两个整数l和r。

【输出格式】

共一行，包含一个整数，表示合并区间完成后的区间个数。

【数据范围】

$1 \leq n \leq 100000$

$-10^9 \leq l_i \leq r_i \leq 10^9$

【输入样例】

```
5
1 2
2 4
5 6
7 8
7 9
```

【输出样例】

```
3
```

【分析】

根据贪心的思想，我们将所有区间按左端点从小到大排序，然后从前往后枚举每个区间，如果当前区间的右端点在下一个区间中，那么就把下一个区间合并进来，即用下一个区间的右端点更新当前区间的右端点。

【代码】

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

typedef pair<int, int> PII;
int n;
vector<PII> segs;

vector<PII> merge(vector<PII> &segs)
{
```

```
    sort(segs.begin(), segs.end());//按左端点从小到大将所有区间排序
    vector<PII> res;
    int st = -2e9, ed = -2e9;
    for (auto seg : segs)
        if (ed < seg.first)//当前区间右端点小于下一个区间的左端点
        {
            if (ed != -2e9) res.push_back({ st, ed });
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);
    res.push_back({ st, ed });//最后一段区间
    return res;
}

int main()
{
    cin >> n;
    while (n--)
    {
        int l, r;
        cin >> l >> r;
        segs.push_back({ l, r });
    }
    vector<PII> ans = merge(segs);
    cout << ans.size() << endl;
    return 0;
}
```