二分与前缀和

一、AcWing 789. 数的范围

【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】

$1 \le n \le 100000$

$1 \le q \le 10000$

$1 \le k \le 10000$

【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，直接上代码。

【代码】

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

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

int main()
{
    cin >> n >> q;
    for (int i = 0; i < n; i++) cin >> a[i];
    while (q--)
    {
        cin >> k;
        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[r] != k) cout << -1 << ' ' << -1 << endl;
        else
        {
            cout << r << ' ';
            l = 0, r = n - 1;
            while (l < r)
            {
                int mid = l + r + 1 >> 1;
                if (a[mid] <= k) l = mid;
                else r = mid - 1;
            }
```

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

二、AcWing 790. 数的三次方根

【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】

$-10000 \le n \le 10000$

【输入样例】

```
1000.00
```

【输出样例】

```
10.000000
```

【分析】

模板题，直接上代码。

【代码】

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

```

```cpp
int main()
{
    double n;
    cin >> n;
    double l = -1e4, r = 1e4;
    while (r - l > 1e-8)
    {
        double mid = (l + r) / 2;
        if (pow(mid, 3) < n) l = mid;
        else r = mid;
    }
    printf("%lf\n", r);
    return 0;
}
```

三、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】

$1 \le l \le r \le n$

$1 \le n, m \le 100000$

$-1000 \le \text{数列中元素的值} \le 1000$

【输入样例】

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

【输出样例】

```
3
6
10
```

【分析】

模板题，直接上代码。

【代码】

```cpp
#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;
}
```

四、AcWing 796. 子矩阵的和

【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】

$1 \le n, m \le 1000$

$1 \le q \le 200000$

$1 \le x_1 \le x_2 \le n$

$1 \le y_1 \le y_2 \le m$

$-1000 \le$ 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】

模板题，直接上代码。

【代码】

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

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

int main()
{
    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] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];
    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;
}
```

五、AcWing 730. 机器人跳跃问题

【题目描述】

机器人正在玩一个古老的基于DOS的游戏。

游戏中有$N+1$座建筑，从$0 \sim N$编号，从左到右排列。

编号为0的建筑高度为0个单位，编号为i的建筑高度为H_i个单位。

起初，机器人在编号为0的建筑处。

每一步，它跳到下一个（右边）建筑。

假设机器人在第k个建筑，且它现在的能量值是E，下一步它将跳到第$k+1$个建筑。

如果$H_{k+1} > E$，那么机器人就失去$H_{k+1} - E$的能量值，否则它将得到$E - H_{k+1}$的能量值。

游戏目标是到达第N个建筑，在这个过程中能量值不能为负数个单位。

现在的问题是机器人至少以多少能量值开始游戏，才可以保证成功完成游戏？

【输入格式】

第一行输入整数N。

第二行是N个空格分隔的整数，$H_1, H_2, \ldots, H_N$代表建筑物的高度。

【输出格式】

输出一个整数，表示所需的最少单位的初始能量值上取整后的结果。

【数据范围】

$1 \le N, H_i \le 10^5$

【输入样例1】

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

【输出样例1】

```
4
```

【输入样例2】

```
3
4 4 4
```

【输出样例2】

```
4
```

【输入样例3】

```
3
1 6 4
```

【输出样例3】

```
3
```

【分析】

假设当前能量为E_k，首先分析两种情况：

- 如果$E_k < H_{k+1}$，则$E_{k+1} = E_k - (H_{k+1} - E_k) = 2E_k - H_{k+1}$
- 如果$E_k \geq H_{k+1}$，则$E_{k+1} = E_k + (E_k - H_{k+1}) = 2E_k - H_{k+1}$

因此无论如何，每跳一步的计算公式都为$E_{k+1} = 2E_k - H_{k+1}$。

接着再分析其性质，假设E_0为初始能量值，$E_0' \geq E_0$，那么$E_1' = 2E_0' - H_1 \geq E_1 = 2E_0 - H_1$，以此类推之后的每个$E_k'$都会大于等于$E_k$，因此如果初始能量值取$E_0$能够满足条件那么取$E_0'$也一定能满足条件。那么我们就可以二分找出这个最小的E_0。

二分E_0的值mid，我们判断当初始能量值为mid时是否能够跳到N号建筑且中途能量值没有小于0，如果满足则返回$true$，否则返回$false$。那么递推的过程中可能会爆int，如何进行优化呢？假设建筑的最高高度为$H_{max} \leq 10^5$，如果在跳跃过程中有一个点的能量值已经大于等于最高高度即$E_k \geq H_{max}$，那么之后不管怎么样能量值都不会再下降了，因为对于跳过最高建筑的时候，$E_{max} = 2E_k - H_{max} \geq E_k$，而易知对于其它高度低于$H_{max}$的建筑，跳完之后能量一定也是上升的，因此递推的时候如果发现当前能量已经大于等于10^5那么就可以返回$true$了。

【代码】

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

const int N = 100010;
int h[N];
int n;

bool check(int mid)
{
    for (int i = 0; i < n; i++)
    {
        mid = mid * 2 - h[i];


        if (mid >= 1e5) return true;//如果当前能量已经大于最大的h那么能量就
```

```
一定不会再下降了
        if (mid < 0) return false;//如果能量小于0则当前的mid不合法
    }
    return true;
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> h[i];
    int l = 0, r = 1e5;//最大值取1e5已经一定大于h的最大值了,因此一定合法
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    cout << r << endl;
    return 0;
}
```

六、AcWing 1221. 四平方和

【题目描述】

四平方和定理，又称为拉格朗日定理：

每个正整数都可以表示为至多4个正整数的平方和。

如果把0包括进去，就正好可以表示为4个数的平方和。

比如：

$5 = 0^2 + 0^2 + 1^2 + 2^2$

$7 = 1^2 + 1^2 + 1^2 + 2^2$

对于一个给定的正整数，可能存在多种平方和的表示法。

要求你对4个数排序：

$0 \le a \le b \le c \le d$

并对所有的可能表示法按a, b, c, d为联合主键升序排列，最后输出第一个表示法。

【输入格式】

输入一个正整数N。

【输出格式】

输出4个非负整数，按从小到大排序，中间用空格分开。

【数据范围】

$0 < N < 5 * 10^6$

【输入样例】

```
5
```

【输出样例】

```
0 0 1 2
```

【分析】

第一个思路是暴力，可以按字典序从小到大枚举a, b, c，然后计算$d = \sqrt{n - a^2 - b^2 - c^2}$是否合法，第一次合法的序列即为满足条件的字典序最小的序列。时间复杂度为$O(n^3)$，会超时。

第二个思路是二分，先按字典序从小到大枚举c, d，将$s = c^2 + d^2$保存下来，同时记录c, d的值，然后对于s从小到大进行排序，s相同时按c, d字典序从小到大排序。然后按字典序从小到大枚举a, b，对于$t = n - a^2 - b^2$，如果存在t等于之前记录的某个$c^2 + d^2$，那么就找到了一个合法的序列，由于s已经从小到大排序了，因此可以二分找出第一个大于等于t的s，如果$s == t$，那么s所记录的c, d与当前枚举的a, b就产生了字典序最小的合法序列。时间复杂度为$O(n^2 logn)$。

第三个思路是哈希表，整体思路与二分差不多，使用哈希表存下$c^2 + d^2$的序列c, d，由于是按字典序枚举c, d的，因此在第一次出现$c^2 + d^2$这个值时记录下c, d即可，然后枚举a, b，判断$t = n - a^2 - b^2$的值是否已经在哈希表中存在，如果存在，取出哈希表该值对应的c, d即为答案。

【暴力写法$O(n^3)$超时】

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


int n;
```

```

int main()
{
    scanf("%d", &n);
    for (int a = 0; a * a <= n; a++)
        for (int b = a; a * a + b * b <= n; b++)
            for (int c = b; a * a + b * b + c * c <= n; c++)
            {
                int t = n - a * a - b * b - c * c;
                int d = sqrt(t);
                if (d * d == t) { printf("%d %d %d %d\n", a, b, c, d); return 0; }
            }
    return 0;
}
```

【二分写法$O(n^2logn)2755ms$】

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

const int N = 2500010;
int n, m;

struct Sum
{
    int s, c, d;
    bool operator< (const Sum& t) const
    {
        if (s != t.s) return s < t.s;
        else if (c != t.c) return c < t.c;
        else if (d != t.d) return d < t.d;
    }
}sum[N];

int main()
{
    scanf("%d", &n);
    for (int c = 0; c * c <= n; c++)
        for (int d = c; c * c + d * d <= n; d++)

            sum[m++] = { c * c + d * d, c, d };
```

```
    sort(sum, sum + m);
    for (int a = 0; a * a <= n; a++)
        for (int b = a; a * a + b * b <= n; b++)
        {
            int l = 0, r = m - 1, t = n - a * a - b * b;
            while (l < r)
            {
                int mid = l + r >> 1;
                if (sum[mid].s >= t) r = mid;
                else l = mid + 1;
            }
            if (sum[l].s == t)
            {
                printf("%d %d %d %d\n", a, b, sum[l].c, sum[l].d);
                return 0;
            }
        }
    return 0;
}
```

【哈希表写法$O(n^2)506ms$】

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

const int N = 5000010;
typedef pair<int, int> PII;
PII ids[N];
bool st[N];
int n;

int main()
{
    scanf("%d", &n);
    for (int c = 0; c * c <= n; c++)
        for (int d = c; c * c + d * d <= n; d++)
        {
            int t = c * c + d * d;
            if (!st[t]) ids[t] = { c, d }, st[t] = true;
        }
    for (int a = 0; a * a <= n; a++)

        for (int b = a; a * a + b * b <= n; b++)
```

```
        {
            int t = n - a * a - b * b;
            if (st[t])
            {
                printf("%d %d %d %d\n", a, b, ids[t].first, ids[t].second);
                return 0;
            }
        }
    return 0;
}
```

七、AcWing 1227. 分巧克力

【题目描述】

儿童节那天有K位小朋友到小明家做客。

小明拿出了珍藏的巧克力招待小朋友们。

小明一共有N块巧克力，其中第i块是$H_i \times W_i$的方格组成的长方形。

为了公平起见，小明需要从这N块巧克力中切出K块巧克力分给小朋友们。

切出的巧克力需要满足：

- 形状是正方形，边长是整数
- 大小相同

例如一块6×5的巧克力可以切出6块2×2的巧克力或者2块3×3的巧克力。

当然小朋友们都希望得到的巧克力尽可能大，你能帮小明计算出最大的边长是多少么？

【输入格式】

第一行包含两个整数N和K。

以下N行每行包含两个整数H_i和W_i。

输入保证每位小朋友至少能获得一块1×1的巧克力。

【输出格式】

输出切出的正方形巧克力最大可能的边长。

【数据范围】

$1 \le N, K \le 10^5$

$1 \le H_i, W_i \le 10^5$

【输入样例】

```
2 10
6 5
5 6
```

【输出样例】

```
2
```

【分析】

当正方形的边长越大时，所能切分出的块数cnt就越小。我们需要找出满足$cnt \ge K$的最大边长，因此可以使用二分进行查找。$check(mid)$判断当边长为mid时切分出的块数是否大于等于K，如果满足则$l = mid$，否则$r = mid - 1$。

如何快速计算每块巧克力所能切分的块数呢？当边长为mid时，$H_i \times W_i$的巧克力可以切成$(H_i/mid) * (W_i/mid)$块。

【代码】

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

typedef long long LL;
const int N = 100010;
int h[N], w[N];
int n, k;

bool check(int mid)
{
    LL cnt = 0;
    for (int i = 0; i < n; i++) cnt += (LL)(h[i] / mid) * (w[i] / mid);
    return cnt >= k;
}

int main()
{
    cin >> n >> k;

    for (int i = 0; i < n; i++) cin >> h[i] >> w[i];
```

```cpp
    int l = 1, r = 1e5;
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    cout << r << endl;
    return 0;
}
```

八、AcWing 99. 激光炸弹

【题目描述】

地图上有N个目标，用整数X_i, Y_i表示目标在地图上的位置，每个目标都有一个价值W_i。

注意：不同目标可能在同一位置。

现在有一种新型的激光炸弹，可以摧毁一个包含$R \times R$个位置的正方形内的所有目标。

激光炸弹的投放是通过卫星定位的，但其有一个缺点，就是其爆炸范围，即那个正方形的边必须和x, y轴平行。

求一颗炸弹最多能炸掉地图上总价值为多少的目标。

【输入格式】

第一行输入正整数N和R，分别代表地图上的目标数目和正方形的边长，数据用空格隔开。

接下来N行，每行输入一组数据，每组数据包括三个整数X_i, Y_i, W_i，分别代表目标的x坐标，y坐标和价值，数据用空格隔开。

【输出格式】

输出一个正整数，代表一颗炸弹最多能炸掉地图上目标的总价值数目。

【数据范围】

$0 \le R \le 10^9$

$0 < N \le 10000$

$0 \le X_i, Y_i \le 5000$

$0 \le W_i \le 1000$

【输入样例】

```
2 1
0 0 1
1 1 1
```

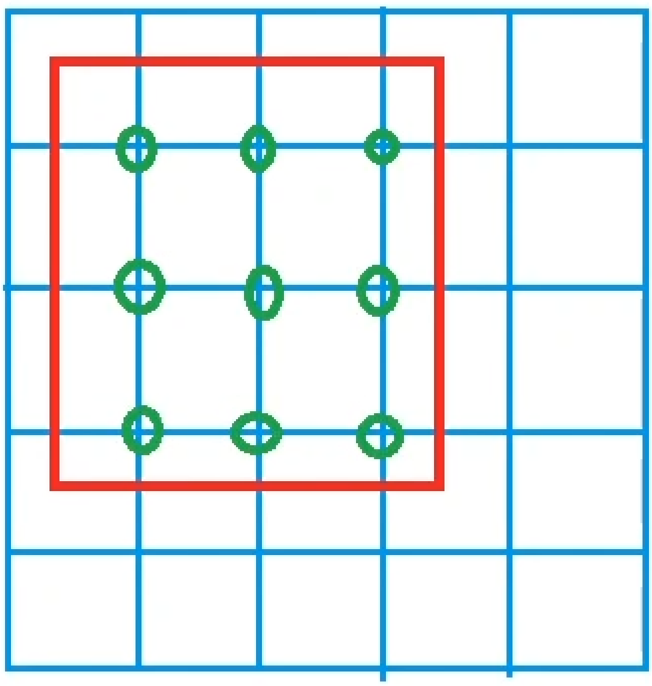
【输出样例】

```
1
```

【分析】

首先所有目标一定都在一个边长为5000的正方形区域内，因此R最大等于5001时（在爆炸正方形区域的边长上的目标不会受影响）即可将所有目标炸了，因此题中所给的R值可能过大，可以$R = min(R, 5001)$。

根据题意，边长为R的爆炸范围一共可以炸掉$R \times R$个目标，如下图所示：



CSDN @柃歌

因此我们可以枚举所有边长为R的子矩阵，求出所有子矩阵和中的最大值，因此可以使用二维前缀和快速求解。我们可以枚举子矩阵的右下角(i, j)，其对应的子矩阵左上角的坐标为$(i - R + 1, j - R + 1)$，根据二维前缀和公式：$res = s[x_2][y_2] - s[x_1 - 1][y_2] - s[x_2][y_1 - 1] + s[x_1 - 1][y_1 - 1] \iff s[i][j] - s[i - R][j] - s[i][j - R] + s[i - R][j - R]$。

注意我们求前缀和时数组下标从1开始，因此需要将题中所给的X_i, Y_i先加一，将其转换成$1 \sim 5001$后再求解。

【代码】

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

const int N = 5010;
int s[N][N];
int cnt, R;

int main()
{
    cin >> cnt >> R;
    R = min(R, 5001);
    while (cnt--)
    {
        int x, y, z;
        cin >> x >> y >> z;
        s[++x][++y] += z;
    }
    for (int i = 1; i < N; i++)
        for (int j = 1; j < N; j++)
            s[i][j] += s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
    int res = 0;
    for (int i = R; i < N; i++)
        for (int j = R; j < N; j++)//正方形左上角顶点坐标为(i-R+1,j-R+1)
            res = max(res, s[i][j] - s[i - R][j] - s[i][j - R] + s[i - R][j - R]);
    cout << res << endl;
    return 0;
}
```

九、AcWing 1230. K倍区间

【题目描述】

给定一个长度为N的数列，$A_1, A_2, \dots, A_N$，如果其中一段连续的子序列$A_i, A_{i+1}, \dots, A_j$之和是K的倍数，我们就称这个区间$[i, j]$是K倍区间。

你能求出数列中总共有多少个K倍区间吗？

【输入格式】

第一行包含两个整数N和K。

以下N行每行包含一个整数A_i。

【输出格式】

输出一个整数，代表K倍区间的数目。

【数据范围】

$1 \le N, K \le 10^5$

$1 \le A_i \le 10^5$

【输入样例】

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

【输出样例】

```
6
```

【分析】

首先我们应该会想到暴力做法，先预处理出数列的前缀和s，然后分别枚举区间的右端点r与左端点l，如果区间$[l, r]$的和模k等于0，即$(s[r]-s[l-1])\%k == 0$，则答案加一。这种做法的时间复杂度为$O(n^2)$，会超时。

我们枚举左端点l的含义可以理解为：当右端点r固定时，在$1 \sim r$之间找出有多少个l满足$(s[r]-s[l-1])\%k == 0$，即在$0 \sim r-1$之间找出有多少个l满足$(s[r]-s[l])\%k == 0$，也就是$s[r], s[l]$模k的余数相等。因此我们可以开一个数组$cnt[x]$记录r之前出现过的$s[i]\%k == x$的个数，如果当前$s[r]\%k == x$，那么答案就加上$cnt[x]$，然后将$s[r]$加入计数，即$cnt[s[r]\%k]++$。这种做法的时间复杂度为$O(n)$。

注意初始化时$s[0]=0$，因此余数为0的数量初始化为1个，即$cnt[0]=1$。可以理解为如果存在某一个i，使得$s[i]\%k=0$，那么$s[i]$自成一个k倍区间，所以需要一开始将$cnt[0]$置成1；相应的，其他$s[i]\%k \ne 0$，这种$s[i]$自己不能单独构成k倍区间，故不能提前置1（同余非零的$s[i]$出现一个以上，方可构成k倍区间）。

【代码】

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

typedef long long LL;
const int N = 100010;
LL s[N], cnt[N];//cnt[i]表示%k余数为i的个数
int n, k;

int main()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++) { cin >> s[i]; s[i] += s[i - 1]; }
    cnt[0] = 1;//s[0]%k==0,即初始化余数为0的数量是1
    LL res = 0;
    for (int i = 1; i <= n; i++)
    {
        res += cnt[s[i] % k];
        cnt[s[i] % k]++;
    }
    cout << res << endl;
    return 0;
}
```