250702数组搜索（题解）

> 题目链接：
> https://www.matiji.net/exam/brushquestion/22/4497/C2CBD34082148550EF198C50D10DBDC7?from=1

#以下题目可能有错别字等，完整题目请访问上述链接
现在有一个长度为n的非严格单调递增的数组，小码哥希望你能根据他的m条指令来进行相应的搜索查找，你能够帮他完成吗。如果小码哥说‘L’和一个数字X，那么请你找出这列数组中第一次出现X的数组下标，如果小码哥说‘R’和一个数字X，那么请你找出这列数组中最后一次出现这个XX他的数组下标。如果没有这个数，请你输出‘-1’；
注意：下标从0开始！

输入格式：
第一行输入一个n，m，n表示数组长度，m表示小码哥的指令条数；
第二行输入一个长度为n的数组，所有数字均不大于100000；
之后m行输入一个字符ch，和一个数字x表示要找的数。

输出格式：
输出m行，x被要求的下标或者“-1”。

输入：
```
6 3
1 2 3 4 4 5
L 4
R 4
R 0
```
输出：
```
3
4
-1
```

其中：1≤n,m≤100000

暴力（超时TLE）

对于每条指令：
'L'：从头到尾遍历，遇到第一个等于 X 的位置返回下标
'R'：从尾到头遍历，遇到第一个等于 X 的位置返回下标
没有找到，输出 -1

```cpp
//文档最后附C语言代码

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

int main() {
    int n, m;
    cin >> n >> m; // 读入数组长度 n 和指令条数 m

    vector<int> num(n); // 定义长度为 n 的数组 num
    for (int i = 0; i < n; ++i) cin >> num[i]; // 读入数组元素

    for (int i = 0; i < m; ++i) {
        char ch;
        int x;
        cin >> ch >> x; // 读入指令类型 ch 和要查找的数字 x

        if (ch == 'L') {
            // find 从左到右查找，返回迭代器指向第一个等于 x 的位置
            auto it = find(num.begin(), num.end(), x);
            if (it != num.end())
                cout << (it - num.begin()) << "\n"; // 输出下标
            else
                cout << -1 << "\n"; // 未找到输出 -1
        } else {
            // find 从右到左查找，返回逆序迭代器指向第一个等于 x 的位置
            auto it = find(num.rbegin(), num.rend(), x);
            if (it != num.rend())
                cout << (n - 1 - (it - num.rbegin())) << "\n"; // 计算真实下
标
            else
                cout << -1 << "\n";
        }
    }
    return 0;
}
```

从左往右查找（`ch == 'L'`）

```cpp
auto it = find(num.begin(), num.end(), x);
```

- `find` 函数在 `[num.begin(), num.end())` 区间内查找第一个等于 `x` 的元素。
- 如果找到，`it` 就指向那个元素；否则 `it == num.end()`。

下面计算并输出元素的索引：

```
cout << (it - num.begin()) << "\n";
```

迭代器之间可以做减法，结果是两个迭代器之间的距离（元素个数），这里就是找到元素的下标。

从右往左查找（`ch != 'L'`）

```
auto it = find(num.rbegin(), num.rend(), x);
```

- 这里用的是逆序迭代器，`num.rbegin()` 指向最后一个元素，`num.rend()` 指向第一个元素之前。
- `find` 在逆序区间中查找第一个等于 `x` 的元素。

然后输出真实索引：

```
cout << (n - 1 - (it - num.rbegin())) << "\n";
```

因为是逆序迭代器，`it - num.rbegin()` 是从最后一个元素开始数到找到元素的距离，距离越大下标越小。
通过 `n - 1 - 距离` 得到正常的数组下标。

概念	作用	举例代码
迭代器	容器的“指针”，用来遍历元素	`num.begin()`，`num.end()`，`num.rbegin()`
`auto`	自动推断变量类型，简化代码	`auto it = find(...);`
`find()`	在指定区间查找某个元素，返回迭代器	`find(num.begin(), num.end(), x)`
迭代器减法	计算两个迭代器之间元素的距离，类似下标差	`it - num.begin()`

解法一：哈希表

```
暴力的时间复杂度：
        每次 find：O(n)
        总体：O(n * m)，大数据下容易超时

考虑到问题的要求，我们需要在数组中查找某个数字的位置，但每次都需要遍历整个数组，这就带来
```

了 O(n) 的时间复杂度，极易超时。哈希表能帮助我们：

预处理数组，记录每个数字：
(1)最左出现位置
(2)最右出现位置

存储首次出现和最后一次出现的位置：通过遍历数组并记录每个数字的第一次和最后一次出现的位置，我们能够在查询时直接通过哈希表得到这些位置，避免了再次遍历整个数组,因此整体效率大大提升。

使用哈希表预计时间复杂度 O(n + m)。

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

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> num(n);

    unordered_map<int, int> mpL, mpR;
    for (int i = 0; i < n; ++i) {
        cin >> num[i];
        if (!mpL.count(num[i])) {
            mpL[num[i]] = i;  // 记录最左出现位置
        }
        mpR[num[i]] = i;      // 不断更新，最后就是最右位置
    }

    for (int i = 0; i < m; ++i) {
        char ch;
        int x;
        cin >> ch >> x;

        if (ch == 'L') {
            if (mpL.count(x)) cout << mpL[x] << "\n";
            else cout << -1 << "\n";
        } else {
            if (mpR.count(x)) cout << mpR[x] << "\n";
            else cout << -1 << "\n";
        }
    }

    return 0;
}
```

解法二：数组映射（问题拓展）

数组映射前提：数字的取值范围是有限且较小（如 0~10^6 以内），如果数字范围太大（比如负数或10^9），用哈希表

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAX_VAL = 1e5 + 10;
vector<int> positions[MAX_VAL];
//注意和vector<int> positions(MAX_VAL);的区别，不了解的可以先查一下

int main() {
    int n, m;
    cin >> n >> m;
        //预处理数组，记录每个数字出现过的位置
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        positions[x].push_back(i);
    }

    for (int i = 0; i < m; i++) {
        char ch;
        int x;
        cin >> ch >> x;

        int len = (int)positions[x].size();
        if (len == 0) {
            cout << -1 << "\n";
            continue;
        }

        if (ch == 'L') {
            cout << positions[x][0] << "\n";
        } else {
            cout << positions[x][len - 1] << "\n";
        }
    }

    return 0;
}
```

最大好处 就是：
        不仅可以查询数字的第一个位置（最左）、最后一个位置（最右）
        还能直接查询该数字的第 2 个、第 n 个出现位置，非常灵活

如果遇到数据范围合适，需要找第二个第三个或第n个的问题，可以直接使用数组映射
如果要找第n个，相应的输出改成 `cout << positions[x][n-1] << "\n";` 即可

当然使用使用哈希表来优化查找第n个出现位置也可以，直接用哈希表优化“查找第n个位置”的需求可以通过存储每个数字出现位置的 vector 来实现。
unordered_map<int, vector<int>> positions;
有想了解到可以查一下相关资料

C语言代码

暴力（超时TLE）：

```c
#include <stdio.h>

int main() {
    int n, m;
    scanf("%d %d", &n, &m); // 读入数组长度 n 和指令条数 m

    int num[n]; // 定义数组
    for (int i = 0; i < n; ++i) {
        scanf("%d", &num[i]); // 读入数组元素
    }

    for (int i = 0; i < m; ++i) {
        char ch;
        int x;
        scanf(" %c %d", &ch, &x); // 读入指令类型和数字，注意前面空格吃掉换行符

        if (ch == 'L') {
            // 从左往右查找第一个等于 x 的元素下标
            int pos = -1;
            for (int j = 0; j < n; ++j) {
                if (num[j] == x) {
                    pos = j;
                    break;
                }
            }
            printf("%d\n", pos);
        } else {
            // 从右往左查找第一个等于 x 的元素下标
            int pos = -1;
            for (int j = n - 1; j >= 0; --j) {
                if (num[j] == x) {
                    pos = j;
                    break;
                }
            }
            printf("%d\n", pos);
        }
```

```
    }
    return 0;
}
```