双指针、单调栈、单调队列

dbywsc

2025/7

目录

介绍

双指针(two-pointer) 是一种简单而又灵活的技巧和思想，单独使用可以用来解决一些特定的问题，也可以配合其他算法实现不同的功能。

顾名思义，双指针即遍历时同时使用两个指针指向不同的位置，通过移动两个指针来维护不同的信息。双指针通常有以下两种形式：

快慢指针，即两个指针从一个方向出发，但是移动速度不同。

首尾指针，即两个指针从两端出发，向中间靠拢。

快慢指针 – I

P1102 A-B 数对

先将数组排序。由于排序后相同的数字一定是连续的，所以我们可以维护快慢指针 r_1 和 r_2，r_1 找最后一个 $a_{r_1} - a_i \leq c$ 的下标，r_2 找最后一个 $a_{r_2} - a_i < c$ 的下标，那么中间的 $r_1 - r_2$ 个数一定是 $a_x - a_i = c$ 的个数，因此 $ans += r_1 - r_2$ 。

快慢指针 – II

```
void solve(void) {
    int n, c; std::cin >> n >> c;
    i64 ans = 0;
    std::vector<int> a(n);
    for(int i = 0; i < n; i++) std::cin >> a[i];
    std::sort(a.begin(), a.end());
    int r1 = 0, r2 = 0;
    for(int i = 0; i < n; i++) {
        while(r1 < n && a[r1] - a[i] <= c) r1++;
        while(r2 < n && a[r2] - a[i] < c) r2++;
        ans += r1 - r2;
    }
    std::cout << ans << std::endl;
}
```

由于在循环中，r_1 和 r_2 自始至终都是递增的，也就是说总共只会执行 n 次，因此上面代码双指针部分的时间复杂度是 $O(n)$ 。

首尾指针 – I

P8708 [蓝桥杯 2020 省 A1] 整数小拼接

先对数组排序，然后维护首尾指针 l 和 r，此时我们发现：

1. 如果 $a_l + a_r < k$ [1]，那么 l 到 r 之间的所有数字和 a_l 拼接的结果也必然**小于** k，因此 ans 可以直接加上 $r - l$，之后移动 l 到下一位。
2. 如果 $a_l + a_r = k$，那么 l 到 r 之间的所有数字和 a_l 拼接的结果也必然**小于等于** k，因此 ans 可以直接加上 $r - l$，之后移动 l 到下一位，移动 r 到前一位。
3. 如果 $a_l + a_r > k$，那么 l 到 r 之间的所有数字和 a_l 拼接的结果也必然**大于** k，没有合适的答案，所以移动 r 到前一位。

需要注意的是由于 $sort()$ 默认对字符串的排序使用的是 **字典序**，而在本题中，字符串长度的优先级大于字典序的优先级，因此我们要自定义排序规则。

[1] 此处的 $+$ 指的是字符串拼接

首尾指针 – II

另外，由于本题中 $a_l + a_r$ 和 $a_r + a_l$ 是两种不同的方案，因此我们需要跑两次双指针，一次判断 $a_l + a_r$ ，一次判断 $a_r + a_l$ 。
由于指针自始至终是单调的，所以本题的时间复杂度仍然是线性的。

首尾指针 – III

```cpp
bool cmp(std::string x, std::string y) {
    if(x.size() == y.size()) return x < y;
    return x.size() < y.size();
}
void solve(void) {
    int n; i64 k, ans = 0; std::cin >> n >> k;
    std::vector<std::string> a(n);
    for(int i = 0; i < n; i++) std::cin >> a[i];
    std::sort(a.begin(), a.end(), cmp);
    int l = 0, r = n - 1;
    while(l <= r)
        if(std::stol(a[l] + a[r]) < k) ans += r - l, l++;
        else if(std::stol(a[l] + a[r]) == k)  ans += r - l, l++, r--;
        else r--;
    l = 0, r = n - 1;
    while(l <= r)
        if(std::stol(a[r] + a[l]) < k) ans += r - l, l++;
        else if(std::stol(a[r] + a[l]) == k) ans += r - l, l++, r--;
        else r--;
    std::cout << ans << std::endl;
}
```

介绍

单调栈是维护栈内元素单调性的数据结构。简单来说，如果新入栈的元素会让栈内元素不单调，就会不断的出栈，直到剩下的元素和新入栈的元素仍然满足单调性。
一般来说，单调栈存储的是当前元素在数组中的**下标** 。

原理

P5788 【模板】单调栈

显然，本题要求我们倒着维护一个**严格递减**单调栈。

实现起来其实非常简单，我们用数组模拟，也可以直接用 *stack* 容器。当新元素入栈时，如果栈内为空，那么我们直接入栈，如果栈内不为空，那么先检查栈顶是否**小于等于**要插入的元素，如果小于说明不是严格递减的，所以出栈。之后重复执行这一步，直到栈顶不小于等于当前元素或者栈空，再将当前元素入栈。

实现

```cpp
void solve(void) {
    std::stack<int> s;
    int n; std::cin >> n;
    std::vector<int> f(n + 1), a(n + 1);
    for(int i = 1; i <= n; i++) std::cin >> a[i];
    for(int i = n; i >= 1; i--) {
        while(s.size() && a[s.top()] <= a[i]) s.pop();
        if(s.size()) f[i] = s.top();
        else f[i] = 0;
        s.push(i);
    }
    for(int i = 1; i <= n; i++) std::cout << f[i] << " ";
}
```

介绍

与单调栈类似，**单调队列**是维护队内元素单调性的数据结构，通常来说队列内存储的依然是元素在数组中的下标。
不同的是当队内元素不单调时，我们既可以选择从**对首**出队，也可以选择从**队尾**出队。
因此，出了数组手动模拟之外，如果想要通过 STL 实现这一数据结构，则应该使用 *deque* （双端队列）。
双端队列有以下迭代器：
dq.front() 返回队首。
dq.back() 返回队尾。
dq.push_front() *dq.push_back*() 在对首、队尾入队。
dq.pop_front() *dq.pop_back*() 在对首、队尾出队。
dq.clear() 清空队列。

原理

P1886 滑动窗口 /【模板】单调队列

形式化的说，本题要求我们维护一段长度为 k 的连续的区间，并且求出它的最值。

对于求区间最大值和最小值，我们可以分别跑两次单调队列，第一次维护不增单调队列，如果队尾大于当前元素，就从队尾出队，知道队列为空或者满足不增的单调性，再从队尾入队。如果当前已经处理到了第 k 个以后的元素，那么每次判断新的元素时应该从队首把属于上个区间的元素出队（这个操作应该在维护单调性之前）。之后如果我们已经处理了 k 个以上的元素，此时每次判断新元素就像相当于维护新的区间，因此每次都要输出当前的最值。

第二次跑一次不减单调队列，操作和第一次一样，不同点是如果如果队尾小于当前元素才执行出队操作。

实现

```cpp
void solve(void) {
    int n, k; std::cin >> n >> k;
    std::deque<int> dq;
    std::vector<int> a(n + 1);
    for(int i = 1; i <= n; i++) std::cin >> a[i];
    for(int i = 1; i <= n; i++) {
        while(dq.size() && dq.front() + k <= i) dq.pop_front();
        while(dq.size() && a[dq.back()] > a[i]) dq.pop_back();
        dq.push_back(i);
        if(i >= k) std::cout<< a[dq.front()] << " ";
    } std::cout << std::endl;
    dq.clear();
    for(int i = 1; i <= n; i++) {
        while(dq.size() && dq.front() + k <= i) dq.pop_front();
        while(dq.size() && a[dq.back()] < a[i]) dq.pop_back();
        dq.push_back(i);
        if(i >= k) std::cout<< a[dq.front()] << " ";
    }
}
```

习题

本节学习的内容通常作为“优化技巧”出现在程序设计中，在算法竞赛中需要和一定的思考相结合，是非常实用的技巧。

P5745 【深基附 B 例】区间最大和

P1638 逛画展

CF2112C Coloring Game

P1901 发射站

P2866 [USACO06NOV] Bad Hair Day S

P2032 扫描

P2698 [USACO12MAR] Flowerpot S