奇思妙学 信息学竞赛

CSP-S 提高组

常见优化技巧

双指针

双指针概述

基本概念

双指针：使用两个指针在序列中按照某种规则移动，从而高效解决问题

核心思想：通过指针的单调移动，避免重复计算，降低时间复杂度

时间复杂度：通常将 $O(n^2)$ 优化到 $O(n)$

应用场景：

- 有序数组的两数之和
- 滑动窗口问题
- 链表操作
- 区间合并
- 去重问题

双指针基本类型

1. 同向双指针

两个指针从同一侧开始，都向同一方向移动

特点：

- 指针移动方向相同
- 通常用于滑动窗口问题（即将学到的单调队列）
- 时间复杂度 $O(n)$

同向双指针模板

```
// 同向双指针通用模板
void sameDirectionTwoPointers(int arr[], int n) {
    int left = 0; // 左指针

    for (int right = 0; right < n; right++) {
        // 将 arr[right] 加入当前考虑范围

        while (left <= right && /* 不满足条件 */)
            // 移动左指针，缩小范围
            left++;

        // 此时 [left, right] 满足条件，更新答案
    }
}
```

2. 相向双指针

两个指针分别从序列两端向中间移动

特点：

- 指针移动方向相对（贪心算法常用，例如配对问题）
- 通常用于有序数组的搜索
- 时间复杂度 $O(n)$

相向双指针模板

```
// 相向双指针通用模板
void oppositeDirectionTwoPointers(int arr[], int n) {
    int left = 0, right = n - 1;

    while (left < right) {
        int sum = arr[left] + arr[right]; // 或其他计算

        if (sum == target)
            // 找到目标解
            left++, right--;
        else if (sum < target)
            left++; // 和太小，左指针右移
        else
            right--; // 和太大，右指针左移
    }
}
```

例题一：P1147 连续自然数和

题目描述

给定正整数 M，求有多少种连续自然数段，使得这些自然数的和为 M

数据范围：$1 \leq M \leq 2 \times 10^6$

解题思路

使用同向双指针维护滑动窗口：

- 左指针 `l`，右指针 `r`
- 当前和 `sum = l + (l+1) + ... + r`
- 当 `sum < M` 时，`r++` 扩大窗口
- 当 `sum > M` 时，`l++` 缩小窗口
- 当 `sum == M` 时，记录答案并移动指针

参考代码

```cpp
int main() {
    int M;
    cin >> M;

    int l = 1, r = 1;  // 左右指针
    int sum = 0;       // 当前窗口和
    int count = 0;     // 方案数

    while (l <= M / 2 + 1) {  // 至少两个数，l最大到M/2
        if (sum < M) {
            sum += r;  // 扩大窗口
            r++;
        } else if (sum > M) {
            sum -= l;  // 缩小窗口
            l++;
        } else {
            // 找到满足条件的区间 [l, r-1]
            cout << l << " " << r - 1 << endl;
            count++;
            sum -= l;  // 移动左指针继续寻找
            l++;
        }
    }

    return 0;
}
```

例题二：P1102 A-B 数对

题目描述

给出一串正整数序列和一个正整数 C，要求计算出所有 $A - B = C$ 的数对的个数

数据范围：$1 \leq n \leq 2 \times 10^5$，$1 \leq C \leq 2 \times 10^9$

解题思路

1. 对数组排序
2. 对于每个数 A，使用双指针寻找 $B = A - C$
3. 注意可能有重复数字，需要统计相同数字的个数

参考代码

```cpp
const int N = 200010;
int arr[N];
int main() {
    int n, C;
    cin >> n >> C;
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    sort(arr, arr + n);

    long long ans = 0;
    int l = 0, r = 0; // 双指针

    for (int i = 0; i < n; i++) {
        int target = arr[i] - C;

        // 寻找第一个等于target的位置
        while (l < n && arr[l] < target)
            l++;
        // 寻找最后一个等于target的位置
        while (r < n && arr[r] <= target)
            r++;

        if (l < n && arr[l] == target)
            ans += (r - l); // 区间 [l, r-1] 都是target
    }

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

例题三：P1638 逛画展

题意

有 n 幅画，每幅画有一个画家编号，求包含所有画家编号的最短区间，如果存在多个最短区间，输出最开始出现的。

数据范围：数组长度 n 满足 $1 \leq n \leq 10^5$，数组元素为整数。

分析

双指针进行区间伸缩（滑动窗口）算法通过两个指针 l 和 r 遍历数组：

- 初始化 $l = 0, r = 0$，窗口为空
- 当区间 $[l, r]$ 不满足条件时，扩展右边界 $r++$
- 当区间满足条件时，记录当前解，并收缩左边界 $l++$
- 通过调整左右边界，确保每个元素最多被访问两次

时间复杂度：每个元素被左指针和右指针各访问一次，因此总时间复杂度为 $O(n)$。

参考代码

```
int n, m, a[1000005], b[2005], k, ans, l, r, ll, rr;
// b[i] 表示当前区间画家i的图画数

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);

    l = 1, r = 1, k = 1, b[a[1]] = 1, ans = 1000005;
    // k 记录当前区间中有多少画家的图画
    while (l <= r && r <= n) {
        if (k == m) // 判断是否符合要求
        {
            if (ans > r - l + 1) {
                ans = r - l + 1; // ans记录最小区间长度
                ll = l, rr = r;
                // ll记录最小区间的左端点,rr记录最小区间的右端点
            }
            b[a[l]]--;
            if (b[a[l]] == 0)
                k--;
            l++;
        } else {
            r++, b[a[r]]++;
            if (b[a[r]] == 1)
                k++;
        }
    }

    printf("%d %d", ll, rr);
    return 0;
}
```

例题四：P6465 [传智杯 #2 决赛] 课程安排

题意

给定 n 节课程的序列，每节课程有一个类型编号。需要选择连续的一段课程作为一周的学习任务，要求：

1. 选定的课程序列长度至少为 len（$len \geq 2$）
2. 序列中不能有连续两节相同类型的课程
3. 序列的第一节和最后一节课程类型不能相同

求满足条件的选课方案数量。

数据范围：$n \leq 5 \times 10^5$，测试数据组数 T 不超过 5。

分析

40pts 做法，$O(n \times m)$

使用双指针：

1. **预处理**：计算每个位置 i 向右最远能扩展的位置 $R[i]$，使得区间 $[i, R[i]]$ 内没有连续相同的课程
2. **方案计算**：对于每个起始位置 i，计算以 i 开头且满足条件的方案数
   - 有效区间为 $[i + \max(l, 2) - 1, R[i]]$
   - 总方案数为区间长度：$\max(0, R[i] - (i + \max(l, 2) - 1) + 1)$
   - 需要减去首尾课程类型相同的方案数

时间复杂度：预处理 $O(n)$，方案计算 $O(nm)$，总时间复杂度 $O(n \times m)$。

参考代码

```
const int N = 500010;
int T, n, l, a[N], R[N];

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &l);
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);

        // 预处理R数组：计算每个位置能扩展的最远位置
        int r = 0;
        for (int i = 1; i <= n; i++) {
            if (r < i)
                r = i;
            while (r < n && a[r] != a[r + 1])
                r++;
            R[i] = r;
        }
```

```
        long long ans = 0;
        int min_len = max(l, 2); // 最小长度为max(l,2)

        for (int i = 1; i <= n; i++) {
            int left = i + min_len - 1; // 右端点最小值
            if (left > R[i])
                continue; // 无有效区间

            // 计算总方案数
            long long total = R[i] - left + 1;

            // 减去首尾课程类型相同的方案数
            // 在区间[left, R[i]]中查找课程类型与a[i]相同的位置
            int cnt = 0;
            for (int j = left; j <= R[i]; j++)
                if (a[j] == a[i])
                    cnt++;

            ans += total - cnt;
        }

        printf("%lld\n", ans);
    }
    return 0;
}
```

100pts 做法 $O(n)$

本题需要统计所有满足条件的连续子序列数量。

刚刚的做法是固定左端点，找到最远不出现相邻课程的区间右端点 R，

减去长度至少为 m 情况下的右端点 r，贡献为 $R-r+1$。

然后再去减去每个区间长度为 m 且满足不相邻课程的区间情况下，

所有头尾相等的区间数量。

这么做显然是 $O(nm)$。

优化：

- 条件 1 要求序列中相邻元素不同，这意味着序列本身是"相邻不同"的。
- 条件 2 要求首尾元素不同，这可以通过排除首尾相同的子序列来满足。
- 条件 3 要求序列长度至少为 l，可以通过计算长度不小于 l 的子序列数量来满足。

思路：

1. **计算 res1**：所有满足"相邻不同"且"首尾不同"的子序列数量（包括长度小于 l 的子序列）。
   - 使用双指针维护以 i 为左端点的最长"相邻不同"序列 $[i, r]$。
   - 对于每个 i，计算以 i 为左端点的子序列中，结尾不等于 $a[i]$ 的数量，即 $(r - i + 1) - cnt[a[i]]$，其中 $cnt[a[i]]$ 是序列 $[i, r]$ 中 $a[i]$ 的出现次数（不包括左端点 i）。

2. **计算 res2**：所有满足"相邻不同"且"首尾不同"但长度小于 l 的子序列数量。
    - 同样使用双指针，但限制序列长度小于 l。
    - 计算方式与 res1 类似。
3. **最终答案**：res1 - res2，即长度至少为 l 的满足条件的子序列数量。

时间复杂度：每个元素被左右指针各访问一次，因此时间复杂度为 $O(n)$。

参考代码

```
const int N = 5e5 + 10;
int T, n, l, a[N], cnt[N];
void init(){
    scanf("%d%d", &n, &l);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        cnt[i] = 0; // 初始化计数数组
    }
    ll res1 = 0, res2 = 0;
    int r = 0;
}
int main() {
    scanf("%d", &T);
    while (T--) init(), solve();
    return 0;
}
```

```
void solve() {
    // 计算 res1：所有满足相邻不同且首尾不同的子序列数量（包括长度小于l的）
    for (int i = 1; i <= n; i++) {
        // 扩展右指针 r，确保 [i, r] 相邻不同
        if (i > r) r++, cnt[a[r]]++;
        while (r < n && a[r] != a[r + 1]) r++, cnt[a[r]]++;

        // 减少左指针 i 的计数，此时 cnt[a[i]] 表示 [i+1, r] 中 a[i] 的出现次数
        cnt[a[i]]--;

        // 计算以 i 为左端点的子序列中，结尾不等于 a[i] 的数量
        res1 += (r - i + 1) - cnt[a[i]];
    }

    r = 0;
    // 计算 res2：长度小于 l 的满足相邻不同且首尾不同的子序列数量
    for (int i = 1; i <= n; i++) {
        // 扩展右指针 r，确保 [i, r] 相邻不同且长度小于 l
        if (i > r) r++, cnt[a[r]]++;
        while (r < n && a[r] != a[r + 1] && (r + 1 - i + 1) < l) r++, cnt[a[r]]++;

        // 减少左指针 i 的计数
        cnt[a[i]]--;

        // 计算以 i 为左端点的子序列中，结尾不等于 a[i] 的数量
        res2 += (r - i + 1) - cnt[a[i]];
    }
    printf("%lld\n", res1 - res2); // 长度至少为l的满足条件的子序列数量
}
```

说明：

- res1 计算的是所有满足相邻不同且首尾不同的子序列数量（包括长度小于 l 的）
- res2 计算的是长度小于 l 的满足相邻不同且首尾不同的子序列数量
- res1 - res2 得到的就是长度至少为 l 的满足条件的子序列数量

双指针技巧总结

适用问题特征

1. 有序或可排序序列
2. 寻找满足某种条件的区间
3. 需要维护某种单调性
4. 暴力解法为 $O(n^2)$

算法复杂度

问题类型	暴力复杂度	双指针复杂度	优化倍数
连续自然数和	$O(n^2)$	$O(n)$	n 倍
A-B 数对	$O(n^2)$	$O(n \log n)$	$n / \log n$ 倍
逛画展	$O(n^2)$	$O(n)$	n 倍

使用技巧与注意事项

1. **指针初始化**：根据问题类型选择合适的初始位置
2. **移动条件**：明确指针移动的条件和方向
3. **边界处理**：注意数组越界和空序列情况
4. **重复元素**：需要时跳过重复元素避免重复计算
5. **循环终止**：确保循环能够正常终止

```
// 安全的双指针实现示例
void TwoPointers(int arr[], int n) {
    int left = 0;

    for (int right = 0; right < n; right++) {
        // 处理当前右指针指向的元素

        // 移动左指针的条件
        while (left <= right && /* 不满足条件 */)
            // 处理左指针移动
            left++;

        // 此时 [left, right] 满足条件
        if (left <= right) {
            // 更新答案
        }
    }
}
```

复习要点

1. **理解双指针的核心思想**：通过指针的单调移动避免重复计算
2. **掌握同向和相向双指针的应用场景**
3. **熟练处理边界条件和指针移动**
4. **学会将复杂问题转化为双指针模型**
5. **注意时间复杂度的优化效果**

掌握双指针技巧，高效解决序列操作问题！

扩展练习（课下完成）

- P11231 [CSP-S 2024] 决斗
- P12218 [蓝桥杯 2023 国 Java B] 玩具
- P3143 [USACO16OPEN] Diamond Collector S