基础算法串讲

zhiyangfan

2024.10.1

目录

前言

- 我是本节课的老师 zhiyangfan，可以叫我 zyf 或者帆。
- 本课程主要是对之前学过的算法进行复习和题目的扩展，希望大家可以跟上思路积极思考。
- 正如目录所示，我们会对普及组的一些基础算法进行复习。
- 如果有讲的不明白的地方欢迎随时指出。

- 贪心，通常指一种简单的，人力想出来的策略，并据此继续完成题目。有时候它是对一个问题的子问题来说的，这时候不仅要求这个策略没问题，还要保证它能够从子问题推导至原问题也没有问题。
- 不过虽然看起来贪心不太好证明，但是大部分情况下，贪心的错误可以通过一些反例来指出。如果难以指出，就可以先暂且认为它是对的，写一份简单的程序来验证。
- 作为一个简单有效的算法，它通常是题目中重要的一步转化，用简单的策略把繁琐的题目化简成我们能求解的问题。

P6568 [NOI Online #3 提高组] 水壶

有 n 个容量无穷大的水壶，它们从 $1 \sim n$ 编号，初始时 i 号水壶中装有 A_i 单位的水。

你可以进行不超过 k 次操作，每次操作需要选择一个满足 $1 \leq x \leq n-1$ 的编号 x，然后把 x 号水壶中的水全部倒入 $x+1$ 号水壶中。

最后你可以任意选择恰好一个水壶，并喝掉水壶中所有的水。现在请你求出，你最多能喝到多少单位的水。

$1 \leq n \leq 10^6$，$0 \leq k \leq n-1$，$0 \leq A_i \leq 10^3$。

- 假如我们已经确定了要喝哪个杯子里的水，策略是什么？

- 假如我们已经确定了要喝哪个杯子里的水，策略是什么？
- 那么任何不倒到这个杯子里的水都是无用的，而有可能倒进来的水只有这个杯子前 k 个。所以最多的情况一定是这些水全部进入我们喝的杯子。

- 假如我们已经确定了要喝哪个杯子里的水，策略是什么？
- 那么任何不倒到这个杯子里的水都是无用的，而有可能倒进来的水只有这个杯子前 k 个。所以最多的情况一定是这些水全部进入我们喝的杯子。
- 那么现在我们不知道要喝哪里的水，不过事实上，因为所有水杯的情况都是相同的，所以问题不难被转化为所有长度为 $k+1$ 的区间和最大值。
- 这样的区间一共只有 $\mathcal{O}(n)$ 个，按照从前至后的顺序枚举区间，每次不难计算新区间的和，取最大值即可。

```cpp
int r = k + 1, ans = 0;
for (int i = 1; i <= r; ++i) sum += a[i];
ans = sum;
while (r <= n)
{
    sum -= a[r - k], sum += a[++r];
    ans = std::max(ans, sum);
}
```

P2777 [AHOI2016 初中组] 自行车比赛

小雪非常关注自行车比赛，尤其是环滨湖自行车赛。一年一度的环滨湖自行车赛，需要选手们连续比赛数日，最终按照累计得分决出冠军。今年一共有 N 位参赛选手。每一天的比赛总会决出当日的排名，第一名的选手会获得 N 点得分，第二名会获得 $N-1$ 点得分，第三名会获得 $N-2$ 点得分，依次类推，最后一名会获得 1 点得分。保证没有选手会排名相同。

在之前的数日较量中，N 位选手已经分别累计了一些分数。现在即将开始的是最后一天的比赛。小雪希望知道有多少位选手还有可能获得最终的冠军，也就是说还有多少选手有可能通过最后一天的比赛获得累计总分第一名。

$3 \leq N \leq 3 \times 10^5$ 且 $0 \leq B_i \leq 2 \times 10^6$。

- 问题显然聚焦于如何分配最后一次比赛的得分情况。

- 问题显然聚焦于如何分配最后一次比赛的得分情况。
- 想要它成为第一名的人应该分配 n 分，剩下的人，分数越高的分配的越低。在这种情况下，如果还不能成为第一名，那么便不可能了。
- 接下来的问题就变为把这个东西从平方做法简化为可以通过的复杂度。

- 不难发现大部分情况下所有人的得分是类似的，只有少数的人因为主角的不同而得分稍有变化。
- 不妨先不考虑主角，而是给所有人按照分数越高分配的分数越少的原则分配分数，得到一个需要越过的最大值。
- 接下来考虑一个人是否能成为第一，此时给它分配 n 的分数，考察剩下的人之前分配的分数是否合理。
- 对于比它大的，分配的分数完全没有错。对于比它小的，分配的分数多了 1。

- 这多的 1 有没有影响呢？如果有影响，只会在特定情况下，设当前考虑的人分数为 x，某个分数比它小的人分数为 y，额外得分为 z。
- 则如果 $x+n \geq y+z$ 但 $x+n < y+z+1$ 就会出现问题。但这里可以推导出 $x=y$，这与分数互不相同矛盾。所以不会有问题。

```cpp
std::sort(b + 1, b + n + 1);
for (int i = 1; i <= n; ++i)
    b[i] += n - i + 1, mx = std::max(mx, b[i]);
for (int i = 1; i <= n; ++i)
    if (c[i] + n >= mx) ++cnt;
```

P8732 [蓝桥杯 2020 国 ABC] 答疑

有 n 位同学同时找老师答疑。每位同学都预先估计了自己答疑的时间。
老师可以安排答疑的顺序，同学们要依次进入老师办公室答疑。
一位同学答疑的过程如下:

- 首先进入办公室，编号为 i 的同学需要 s_i 毫秒的时间。
- 然后同学问问题老师解答，编号为 i 的同学需要 a_i 毫秒的时间。
- 答疑完成后，同学很高兴，会在课程群里面发一条消息，需要的时间可以忽略。
- 最后同学收拾东西离开办公室，需要 e_i 毫秒的时间。

一位同学离开办公室后，紧接着下一位同学就可以进入办公室了。
答疑从 0 时刻开始。老师想合理的安排答疑的顺序，使得同学们在课程群里面发消息的时刻之和最小。

- 这题是一种非常经典的贪心思路，即对于某个序列重排使得它的某个性质取得最值。
- 有时候我们可以总结出一个排序的 cmp 函数，使得按照它排序即符合条件。

- cmp 函数需要完成的任务是，判断两个元素谁放在前更好。注意到本题中，两位相邻同学的顺序，并不影响其他人，因为他们的总时间一定。
- 既然如此，如果我们能比较相邻位置的优秀与否，由排序的性质（交换相邻两个元素）我们即可得到最优的序列。
- 假设 x, y 相邻，我们需要比较：

$$s_x + a_x + +(s_x + a_x + e_x + s_y + a_y), s_y + a_y + +(s_y + a_y + e_y + s_x + a_x)$$

- 不难发现实质上只是在比较 $s_x + a_x + e_x$。以它为标准排序计算即可。

```cpp
bool cmp(int x, int y)
{
    return s[x] + a[x] + e[x] < s[y] + a[y] + e[y];
}
//...
std::sort(id + 1, id + n + 1, cmp);
long long now = 0, sum = 0;
for (int j = 1, i = id[j]; j <= n; ++j, i = id[j])
{
    sum += now + s[i] + a[i];
    now += s[i] + a[i] + e[i];
}
printf("%lld\n", sum);
```

P9344 去年天气旧亭台

共有 n 块地板，地板分为两类，第 i 块地板的类别用 c_i 表示，积灰程度用 a_i 表示。** 注意 c_i 为 0 或 1。**

现在要清理这些地板上的灰尘。每次操作中，你可以：

- 选择两个下标 i, j，满足 $1 \le i \le j \le n$，$c_i = c_j$，且第 i 块和第 j 块地板上的灰尘均未被清理过；
- 花费 $a_i + a_j$ 的能量清理第 i 块到第 j 块所有地板上的灰尘。

求清理完所有地板上的灰尘至少要多少能量。

$1 \le n \le 2 \times 10^6$，$c_i \in \{0, 1\}$，$1 \le a_i \le 10^9$。

- 有时候贪心需要从一些看起来很简单的想法出发。比如这道题，我们不难发现，清理的范围越大越优。
- 这样，如果可行的话，一次从 1 清理到 n 肯定是最好的。否则考虑在任何的方案中肯定要涉及 $[1,i],[j,n]$，已经超过 a_1+a_n 了。
- 如果 $c_1 \neq c_n$ 呢？既然一次不行，我们能不能退而求其次，找到两次的方案呢？

- 这要求存在一个 i，满足 $c_1 = c_i, c_{i+1} = c_n, 1 \le i < n$。如果 $c_2 \neq c_1$，那么 $i = 2$ 即可。
- 所以一定有 $c_1 = c_2$，类似地分析，能一直推导到 $c_1 = c_2 = \cdots = c_{n-1}$，此时我们可以找到 $i = n - 1$。
- 故一定存在这样的 i。那取这个 i 是否更优呢？因为清理的区间不能相交，所以如果想要全部都清除，一定存在一次由 c_1 到 c_n 的转化，当然会发生在合法的 i 上面。所以所有的其他方案一定大于两次结束的方案。枚举计算即可。
- 本题我们从朴素的第一想法出发，逐步证明了朴素想法的可行性与泛用性，并具体了做法。

```
if (c[1] == c[n]) printf("%d\n", a[1] + a[n]);
else
{
    ll ans = 1e12;
    for (int i = 1; i < n; ++i)
        if (c[i] == c[1] && c[i + 1] == c[n])
            ans = std::min(ans,
                (ll)a[1] + a[i] + a[i + 1] + a[n]);
    printf("%lld\n", ans);
}
```

P7840 「C.E.L.U-03」重构

罗司机有 n 台服务器，每个服务器有一个繁忙度 v_i。罗司机将用 $n-1$ 条网络将它们连接在一起，于是每台服务器有一个连接网络数量 d_i。这个服务器网络运行的总时间是 $\sum_{i=1}^{n} d_i^2 v_i$。请你最小化这个值。

$1 \le n \le 3 \times 10^5, 1 \le v_i \le 10^3$

- 考虑给每个点分配度数。考虑每次分配度数如何让总时间的增量最小，对于一个原本度数为 d，繁忙度为 v 的服务器，给 d 加 1，会增加 $(2d+1)v$ 的时间。
- 所以我们维护一个堆，存当前所有服务器的 $(2d+1)v$ 每次取出最小值加上就好了。

- 考虑给每个点分配度数。考虑每次分配度数如何让总时间的增量最小，对于一个原本度数为 d，繁忙度为 v 的服务器，给 d 加 1，会增加 $(2d+1)v$ 的时间。
- 所以我们维护一个堆，存当前所有服务器的 $(2d+1)v$ 每次取出最小值加上就好了。
- 其实还没有做完，因为这还是一棵树，虽然总度数是 $2n-2$，但为了保证连通性，每个点至少要有 1 的度数，所以需要提前先给每个点分配 1 的度数，再做上述贪心。
- 本题就是典型的局部最优（每次增量最小）得到全局最优（最终和最小），仅根据当下情况（已经决策好了的所有 d）进行决策（分配下一个度数）

```cpp
typedef long long ll; typedef std::pair<ll, int> pii;
//...
std::priority_queue<pii> pq; int res = n - 2;
for (int i = 1; i <= n; ++i)
{
    scanf("%d", &v[i]), ans += v[i];
    pq.emplace(-3 * v[i], i), d[i] = 1;
}
while (res--)
{
    auto [x, y] = pq.top(); pq.pop();
    ans += -x; ++d[y];
    pq.emplace(-(2 * d[y] + 1) * v[y], y);
}
```

- 二分是对于一个范围内的数和一个性质，这些数对于这个性质具有单调性，从而我们可以快速找到想要的数。
- 所谓的单调性，可以类比有序性。换句话说，通过区间中点的情况，我们能快速确定想要的数到底在左边还是在右边。
- 比如当前的中点不符合条件，我们就能确定比它更大的数一定也不符合条件，以此一次排除一半的数，做到 $\mathcal{O}(\log n)$ 的查找速度。
- 二分答案相当于把原问题转化成了判断性问题，后者一般会更好做（知道的信息更多）

P8647 [蓝桥杯 2017 省 AB] 分巧克力

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边长为整数的正方形且大小相同
当然小朋友们都希望得到的巧克力尽可能大，你能帮小 H_i 计算出最大的边长是多少？
$1 \leq N, K, H_i, W_i \leq 10^5$。

- 考虑如果边长 x 能切出来，那么 $1..x$ 一定都能切出来，反之亦然。而在此基础上，我们想要最大化边长，不难发现是否可以切出来可以被二分。
- 即如果当前边长可行，就可以直接丢掉左边，否则直接丢掉右边。最终得到可以被切出来和不能被切出来交界处的边长。
- 剩下的任务就是如何判断一个边长是否能被切出来，只需要判断所有巧克力切出来的正方形之和是否够用即可。
- 对于边长 x，第 i 块巧克力可以切出 $\lfloor \frac{H_i}{x} \rfloor \times \lfloor \frac{W_i}{x} \rfloor$ 块，加起来即可。
- 二分的上下界为 $1, \max\{H_i, W_i\}$，再小再大都没有意义。

```
ll calc(int a)
{
    ll ret = 0;
    for (int i = 1; i <= n; ++i)
        ret += (ll)(h[i] / a) * (w[i] / a);
    return ret;
}
//...
int l = 1, r = 1e5, mid, ans = -1;
while (l <= r)
{
    mid = (l + r) >> 1;
    if (calc(mid) >= k) ans = mid, l = mid + 1;
    else r = mid - 1;
}
printf("%d\n", ans); return 0;
```

P9644 [SNCPC2019] Turn It Off

宝宝的卧室里有 n 盏灯，从 1 到 n 排成一排。每次宝宝可以选择一个整数 i，并将从第 i 盏灯到第 $(i+L-1)$ 盏灯（包括两端）之间的所有灯关掉，其中 L 是一个预定义的正整数。注意，每次操作的 L 值必须相同。给定所有灯的初始状态，请帮助宝宝确定可能的最小 L 使得他能在 k 次操作内关掉所有的灯。

$1 \leq k \leq n \leq 2 \times 10^5$。保证至少一个灯是亮着的。

- 假如我们已经知道了 L，判断是否能关掉所有的灯是简单的。我们从前至后考虑，碰到开着的灯就从这里开始执行一次关灯操作。正确性和复杂度都很正确。
- 接下来注意到，如果 L 不可行，$1..L$ 一定不可行，反之亦然。所以 L 是可以二分的。

```cpp
bool check(int len)
{
    int cnt = 0;
    for (int i = 1; i <= n; ++i) t[i] = s[i];
    for (int i = 1; i <= n; )
    {
        if (t[i] == '0') { ++i; continue; }
        for (int j = 1; j <= len && i <= n; ++j)
            t[i++] = '0';
        ++cnt;
    }
    return cnt <= k;
}
```

```
int l = 1, r = n, mid, ans = -1;
while (l <= r)
{
    mid = (l + r) >> 1;
    if (check(mid)) ans = mid, r = mid - 1;
    else l = mid + 1;
}
printf("%d\n", ans);
```

P9559 [SDCPC2023] Fast and Fat

您正在参加一场团体越野比赛。您的队伍共有 n 名队员，其中第 i 名队员的速度为 v_i，体重为 w_i。

比赛允许每名队员独立行动，也允许一名队员背着另一名队员一起行动。当队员 i 背着队员 j 时，如果队员 i 的体重大于等于队员 j，则队员 i 的移动速度不会变化，仍然为 v_i；如果队员 i 的体重小于队员 j，则队员 i 的移动速度会减去两者的体重差值，即变为 $v_i-(w_j-w_i)$。如果队员 i 的移动速度将变为负数，则队员 i 无法背起队员 j。每名队员最多只能背负另一名队员，被背负的队员无法同时背负其他队员。

所有未被背负的队员中，最慢的队员的速度，即为整个队伍的速度。求整个队伍能达到的最大速度。

$1\le n\le 10^5, 1\le v_i, w_i\le 10^9$

- 本题的答案为最大值最小，发现这样的最大值是可以二分的。即如果一个最大值可行，那么比它大的最大值都可行。所以通常的转化是二分一个最大值，并以它作为限制判断是否可行。
- 现在我们已经知道最大值了，对于一些速度较慢的人，一定是需要找人背的，但问题是找谁呢？
- 直观上，肯定是找一个速度足够且体重刚刚好的，如果体重太大，会浪费，如果体重不够，速度会慢。

- 在此基础上我们考虑贪心地去匹配，把可以背人的人能背的最大重量，和需要背的人重量分别排序后从大至小一一对应。
- 证明考虑从最大值开始，如果不能满足条件，显然能被的最大重量更小的人一定也不行。否则后面重量更小的也一定用不上这个（如果后面的人需要使用这个最大值才能达到速度，这与排好序不符）
- 所以扔掉两个的最大值并不改变可行性。递归考虑即可。

```cpp
bool check(int x)
{
    int t1 = 0, t2 = 0;
    for (int i = 1; i <= n; ++i)
        if (vi[i] >= x) s1[++t1] = vi[i] + wi[i] - x;
        else s2[++t2] = wi[i];
    if (t1 < t2) return false;
    sort(s1 + 1, s1 + t1 + 1, greater<int>());
    sort(s2 + 1, s2 + t2 + 1, greater<int>());
    for (int i = 1; i <= t2; ++i)
        if (s1[i] < s2[i]) return false;
    return true;
}
```

P10091 [ROIR 2022 Day 2] 分数排序

有两个由 n 个不同整数组成的序列 $A = [a_1, a_2, \ldots, a_n]$ 和 $B = [b_1, b_2, \ldots, b_n]$。将它们组合成 n^2 个分数，形式为 $\frac{a_i}{b_j}$，并将每个分数约分后按递增顺序排序。

给定一个数字 q 和 q 个整数 $c_1, c_2, \ldots, c_q$。对于每个 c_i，请输出上面所说的 n^2 个分数中第 c_i 小的分数。

$1 \le n \le 10^5$，$1 \le q \le 10^5$ 且 $q \le n^2, n \times q \le 10^5$（所以实际上 $q \le 1000\sqrt[3]{10} \approx 2154$），$1 \le a_i, b_i \le 10^6$，$1 \le c_i \le n^2$。

- 二分也可以用来查找第 k 大。对于一个 x，只需要知道有多少个比它小的数，即可确定 x 是大了还是小了。
- 比如本题，我们需要解决两个问题，用二分判断分数第 k 大是多少，以及用它去找到一对合法的 a_i, b_i。
- 对于第一个问题，考虑将 a_i, b_i 先排序，之后可以通过一些单调性快速回答。

- 具体地，我们有 $\frac{a}{b} > \frac{a}{b+1}$，所以对于一个 b_i，我们求出 $a_{1..j}$ 满足 $\frac{a}{b} < x$。
- 那么对于 b_{i+1}，得到的 $a_{1..j'}$ 一定满足 $j' \geq j$。根据 j 的单调性，我们在枚举 b_i 的同时求 j，由于 j 递增，所以复杂度是均摊 $\mathcal{O}(n)$ 的。之后把所有 j 加起来即可。
- 对于第二个问题，我们枚举分母是多少，乘上二分的答案后去 a_i 中找分子即可。
- 本题对精度要求比较高，有时候需要高一点以保证答案准确，有时候需要小一点以保证能找到分子。可能需要一定的调试。
- 且以本题为例给大家展示一下实数二分的写法。

```
ll check(db x)
{
    ll sum = 0; int j = 0;
    for (int i = 1; i <= n; ++i)
    {
        while (j < n && a[j + 1] < x * b[i]) ++j;
        sum += j;
    }
    return sum;
}
```

```
ll c; scanf("%lld", &c);
db l = (db)a[1] / b[n], r = (db)a[n] / b[1], mid;
while (r - l > 1e-13)
{
    mid = (l + r) / 2;
    if (check(mid) >= c) r = mid;
    else l = mid;
}
```

```cpp
for (int j = 1; j <= n; ++j)
{
    int ta = std::round(l * b[j]); // 四舍五入
    if (ta < N && vis[ta]
        && fabsl(ta - l * b[j]) < 1e-7)
    {
        int g = std::__gcd(ta, b[j]);
        printf("%d %d\n", ta / g, b[j] / g);
        break;
    }
}
```

前缀和与差分

- 对于一个序列 $\{a_i\}$，其前缀和 $\{s_i\}$ 与差分 $\{d_i\}$，形如：

$$s_i = \sum_{j=1}^{i} a_j, d_i = a_i - a_{i-1}$$

不难发现，差分做一次前缀和就是原序列。前缀和做一次差分亦是原序列。我们可以根据这些性质，选择好维护的东西，并在回答询问时还原。

- 且前缀和可以快速回答区间询问，$s_r - s_{l-1}$ 就是 $a_{l..r}$ 的和。
- 不仅如此，只要满足上述性质的运算（存在逆元，结合律等）都可以用前缀和维护区间询问。

P8649 [蓝桥杯 2017 省 B] k 倍区间

给定一个长度为 N 的数列，$A_1, A_2, \cdots A_N$，如果其中一段连续的子序列 $A_i, A_{i+1}, \cdots A_j (i \leq j)$ 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？

$(1 \leq N, K, A_i \leq 10^5)$

- 对于区间和，我们可以表示为 $a_r - a_{l-1}$，我们直接要求它模 k 余 0。
- 枚举 r，则要求 $a_r \equiv a_{l-1} \pmod{k}$。可以方便的用桶维护这个东西。
- 对于区间和的问题，有时候拆成前缀和相减会更加清晰，更加容易入手（毕竟涉及到的东西很少）

```
for (int i = 1, x; i <= n; ++i)
scanf("%d", &x), sum[i] = (sum[i - 1] + x) % k;
cnt[0] = 1; long long ans = 0;
for (int i = 1; i <= n; ++i) ans += (cnt[sum[i]]++);
printf("%lld\n", ans);
```

P9533 [YsOI2023] 区间翻转区间异或和

符卡有一个长度为 n 的整数数组 a，符卡认为一个区间 $[l,r]$ 是灵异区间当且仅当 $\bigoplus_{i=l}^{r} a_i = 0$，或者说这个区间内所有数字异或起来刚好等于 0。

符卡有特殊的魔法，可以把任意一个灵异区间翻转。具体来说，如果 $[l,r]$ 区间是灵异区间，那么符卡就可以对这个区间使用魔法，整个数组就会变成 $a_1, a_2, \ldots, a_{l-1}, a_r, a_{r-1}, \ldots, a_l, a_{r+1}, a_{r+2} \ldots, a_n$。

现在符卡可以使用任意次数的魔法，符卡希望最后得到的数组的灵异区间数量能够尽可能多，你能告诉她最后最多有多少个灵异区间吗？

$1 \le n \le 10^5$，$0 \le a_i < 2^{20}$

- 考虑反转一个灵异区间会带来什么改变。显然当前的灵异区间仍然灵异。所有与它成包含或相离关系的区间仍然保持灵异。
- 所以只需要考虑剩余相交的区间。假设为 $[l_1, r_1], [l_2, r_2]$ 且 $l_1 \le l_2 \le r_1 \le r_2$。
- 显然有 $\bigoplus_{i=l_1}^{l_2-1} a_i = \bigoplus_{i=l_2}^{r_1} a_i = \bigoplus_{i=r_1+1}^{r_2} a_i$。
- 而反转 $[l_1, r_2]$ 后，$[l_1, l_2 - 1]$ 被反转到了后面，但与 $[r_1 + 1, r_2]$ 相邻，所以它们有组成了一个新灵异区间。换句话说，灵异区间总数不变。
- 所以问题变为求出原序列的总灵异区间个数。考虑异或的性质很好，具有逆元。$\bigoplus_{i=l}^{r} a_i = s_r \oplus s_{l-1}$，其中 $s_i \bigoplus_{j=1}^{i} a_j$，即针对异或的前缀和。
- 所以统计 $s_i = s_j$ 的个数即可。写法与上题类似，但本题值域较大，需要使用 std::map 统计。

```cpp
const int N = 1e5 + 10; int sum[N];
std::map<int, int> cnt;
int main()
{
    int n; scanf("%d", &n);
    for (int i = 1, x; i <= n; ++i)
        scanf("%d", &x), sum[i] = sum[i - 1] ^ x;
    cnt[0] = 1; long long ans = 0;
    for (int i = 1; i <= n; ++i)
        ans += (cnt[sum[i]]++);
    printf("%lld\n", ans); return 0;
}
```

P10903 [蓝桥杯 2024 省 C] 商品库存管理

在库存管理系统中，跟踪和调节商品库存量是关键任务之一。小蓝经营的仓库中存有多种商品，这些商品根据类别和规格被有序地分类并编号，编号范围从 1 至 n。初始时，每种商品的库存量均为 0。

为了高效地监控和调整库存量，小蓝的管理团队设计了 m 个操作，每个操作涉及到一个特定的商品区间，即一段连续的商品编号范围（例如区间 $[L, R]$）。执行这些操作时，区间内每种商品的库存量都将增加 1。

然而，在某些情况下，管理团队可能会决定不执行某些操作，使得这些操作涉及的商品区间内的库存量不会发生改变，维持原有的状态。

现在，管理团队需要一个评估机制，来确定如果某个操作未被执行，那么最终会有多少种商品的库存量为 0。对此，请你为管理团队计算出，对于每个操作，如果不执行该操作而执行其它操作，库存量为 0 的商品的种类数。

$1 \leq n, m \leq 3 \times 10^5$，$1 \leq L \leq R \leq n$

- 考虑如果我们知道最终的商品库存，则询问相当于询问当前区间内 1 的个数和区间外 0 的个数之和。这个可以通过前缀和求出（相当于一个值只有 $0,1$ 表示这一位是不是 0 的序列的前缀和，1 是类似的）
- 所以问题变为若干区间加，求出最终结果。考虑这样一次加会对差分数组带来什么改变。
- 具体来讲，如果给区间 $[l,r]$ 加上 w，则在差分数组 d 上，相当于给 d_l 加上 w，而 $dr+1$ 减去 w，这个容易验证。
- 所以维护好差分数组，求一次前缀和得到原数组，求出 01 数量的前缀和，再依次回答询问即可。

前缀和与差分

```
for (int i = 1; i <= m; ++i)
    scanf("%d%d", &l[i], &r[i]),
    ++d[l[i]], --d[r[i] + 1];
for (int i = 1; i <= n; ++i) d[i] += d[i - 1];
for (int i = 1; i <= n; ++i)
    for (int j = 0; j < 2; ++j)
        s[j][i] = s[j][i - 1] + (d[i] == j);
for (int i = 1; i <= m; ++i)
    printf("%d\n", s[0][l[i] - 1] + s[0][n]
        - s[0][r[i]] + s[1][r[i]] - s[1][l[i] - 1]);
```

P9681 幽默的世界。

给定一个长为 n 的序列 $a_1, a_2, \cdots, a_n$，定义 a 的一个连续子序列 $a_l, a_{l+1}, \cdots, a_r$ 是幽默的，当且仅当：

- $\sum_{i=l}^{r} a_i > 0$；
- 对于所有 $l \le x \le y < r$，满足 $\sum_{i=x}^{y} a_i \le 0$。

q 次询问，每次给定两个整数 l, r，查询满足以下条件的数对 (l', r') 个数：

- $l \le l' \le r' \le r$；
- 连续子序列 $a_{l'}, a_{l'+1}, \cdots a_{r'}$ 是幽默的。

$1 \le n, q \le 2 \times 10^5$，$1 \le l \le r \le n$，$|a_i| \le 10^9$

- 题目中的幽默序列显然可以转化为，最后一个是整数，其余全是负数，且总和为正数。
- 先考虑 $l=1, r=n$ 的询问。对于所有正数，我们向前找到最多的能加进来的负数，比如为 a 个。则它的贡献为 $a+1$。
- 这一步可以通过二分查找，每次通过前缀和判断当前找到的左端点到这里是否全为负数，和是否是正数。

- 题目中的幽默序列显然可以转化为，最后一个是整数，其余全是负数，且总和为正数。
- 先考虑 $l=1, r=n$ 的询问。对于所有正数，我们向前找到最多的能加进来的负数，比如为 a 个。则它的贡献为 $a+1$。
- 这一步可以通过二分查找，每次通过前缀和判断当前找到的左端点到这里是否全为负数，和是否是正数。
- 接下来考虑区间询问，记 a_i 表示这一位的贡献。求出前缀和差分。
- 但这样无法处理区间最开头的那个区间可能会被截断。这也很简单，找到 l 后第一个正数，单独处理它的贡献即可。

```cpp
int n, q; scanf("%d%d", &n, &q);
for (int i = 1; i <= n; ++i)
{
    scanf("%d", &a[i]);
    s[i] = s[i - 1] + a[i];
    cnt[i] = cnt[i - 1] + (a[i] <= 0);
}
```

```
for (int i = 1; i <= n; ++i)
{
    if (a[i] <= 0) { sc[i] = sc[i - 1]; continue; }
    int l = 1, r = i, mid, ans = 0;
    while (l <= r)
    {
        mid = (l + r) >> 1;
        if (s[i] - s[mid - 1] > 0 &&
            cnt[i - 1] - cnt[mid - 1] == i - mid)
            ans = mid, r = mid - 1;
        else l = mid + 1;
    }
    c[i] = i - ans + 1; sc[i] = sc[i - 1] + c[i];
}
```

```cpp
for (int i = 1, l, r; i <= q; ++i)
{
    scanf("%d%d", &l, &r);
    ll ans = sc[r] - sc[l - 1];
    int L = l, R = r, mid, t = l;
    while (L <= R)
    {
        mid = (L + R) >> 1;
        if (cnt[mid] - cnt[l - 1] < mid - l + 1)
            t = mid, R = mid - 1;
        else L = mid + 1;
    }
    ans += std::min(t - l + 1, c[t]) - c[t];
    printf("%lld\n", ans);
}
```

本次课程到这里就结束了

谢谢大家！如果有疑问或错误欢迎课下交流指正！