[4] 基础算法的灵活运用

洛谷 基础组秋令营
V 2022-10

洛谷
www.luogu.com.cn

课前声明

- 请大家加入课程 QQ 群，投屏看不到网页上的实时提问，提问请在群里进行。
- 我们已经默认了同学们已经掌握了以下基础算法：贪心、二分、倍增、基础内容（递推、递归、排序、前缀和、差分等），因此这节课不会介绍这些算法的概念、模板等等，而是会直接以具体的题目为主。
- 本次课程的题目按照算法分类，来自 CSP、NOIP 真题以及洛谷月赛真题及洛谷原创题目，具有一定的思维量。
- 授课过程中有不懂的请随时在 QQ 群提问。
- 总结：这节课会带领各位从具体题目入手，体会一下基础算法是如何应用于题目中的。

课前声明

请同学们将设备音量调整至目前的 50%。

目录

贪心

例题：三国游戏、魔法、旅行家的预算

思路和实现流程

使用贪心算法解题一定需要保证从不同的路径走到一个共同状态，后续的状态变迁都是一样的，和之前采用何种路径到这个状态没有关系，即无后效性。同时，一定要确保自己能证明其正确性。

从问题的某一初始解出发。

当能朝给定总目标前进一步时，求出可行解的一个解元素；

最后，由所有解元素组合成问题的一个可行解。

证明正确性

大胆假设，小心求证。

验证贪心算法是否正确称为贪心算法的证明，贪心算法的证明主要有以下两种方法。

1. 反证法：如果交换方案中任意两个元素/相邻的两个元素后，答案不会变得更好，那么可以推定目前的解已经是最优解了。

2. 归纳法：先算得出边界情况（例如 $n = 1$）的最优解 F_1，然后再证明：对于每个 F_n，都可以由 F_{n-1} 推导出结果。

证伪贪心算法往往采用构造出一个反例的方式。

例题 - [NOIP2010 普及组] 三国游戏

题目链接 https://www.luogu.com.cn/problem/P1199

N 位武将，任意两个武将之间有特定的“默契值”。

你和计算机轮流选择武将，你是先手。选择完成后比较双方武将默契值最大值大小。

计算机采取的策略：任何时刻，尝试将对手军队中的每个武将与当前每个自由武将进行一一配对，找出默契值最高的一对，选择这一对中未被选中的武将。

询问在最优策略下，你是否有可能战胜计算机，如果有可能，你的阵容中的默契值最高是多少。

$N \leq 500$

例题 - [NOIP2010 普及组] 三国游戏

思路

考虑这样一个问题。我们能够取到的最大“默契值”一定来自给定的表格。因此，我们想要在表格中取到尽可能大的值。

考虑能否取到最大值。

考虑一下一般情况，假设我们选择了第 i 号武将，那么这位武将所能产生的贡献全部在表格的第 i 行中。

在这种情况下，计算机会优先破坏第 i 行的最大值。

因此无论如何，每一行的最大值都无法取到。

既然最大值无法取到，不妨考虑一下是否能够取到这种情况下的次大值。

例题 - [NOIP2010 普及组] 三国游戏

思路

考虑在计算机破坏了最大值组合后的选择。

不难发现，当我们起初就选择了第 i 个武将后，计算机破坏了第 i 行的最大值。这种情况下选择权轮到了我们。无论如何，计算机都不能够破坏这一行的第二大值，因此我们总是可以选择这个第二大值。

得到一个很重要的结论：对于每一行，无论如何，我们都无法选择这一行的最大值，并且我们都能够选择这一行的第二大的值。

因此将给定的表格按行考虑，不难发现，在上面的结论的基础上，每一行能够提供的最大贡献就是这一行的第二大的值。

因此答案只要取每一行第二大的值中的最大值即可。

例题 - [NOIP2010 普及组] 三国游戏

思路

考虑这种情况下的胜负问题。

对于第一次操作，我们取走了第 i 个武将后，即使计算机破坏了第 i 行的最大值，由于我们已经取走的这个最大值的构成之一（第 i 个武将），计算机也得不到这个最大值。

因此，不难发现，行中的最大值我们与计算机都无法取到，而行中的第二大值我们可以取到。

同时，我们在第二轮取得了次大值后，对于计算机做出的任何操作，我们也都可以破坏它所选择的武将所在的那一行的最大值，因此计算机同样无法取到任何一行的最大值。

因此，在这种情况下，我们取得了所有行第二大的值的最大值，因此我们必定胜利，故不需要考虑 0 的情况。

例题 - [NOIP2010 普及组] 三国游戏

```cpp
int n, a[505][505]; // a[i][j] (i, j) 的默契最大值
int solution() {
    int ans = 0;
    for (int i = 1; i <= n; ++i) {
        sort(a[i] + 1, a[i] + n + 1);
        ans = max(ans, a[i][n - 1]);
    }
    return ans;
}
int main() {
    cin >> n;
    for (int i = 1; i < n; ++i)
        for (int j = i + 1; j <= n; ++j) {
            cin >> a[i][j];
            a[j][i] = a[i][j];
        }
    cout << 1 << endl << solution() << endl;
    return 0;
}
```

例题 -魔法

题目链接 https://www.luogu.com.cn/problem/P3619

多测，Z 组数据。

对于每组数据，初始有一个属性值 T，有 n 个任务，每个任务有两个参数 t_i, b_i。如果想要完成第 i 个任务，需要 $T > t_i$。完成后，T 会加上 b_i。询问是否存在一种完成任务的顺序，能够完成所有的 n 个任务。

如果可以，输出 $+1s$，否则输出 $-1s$。

$n \le 10^5, Z \le 10, t_i \le 10^5, T \le 10^5, -10^5 \le b_i \le 10^5$

例题 -魔法

思路

首先想到，可以按 b_i 是否 ≥ 0 将所有任务分为两类。A 类为 $b_i \geq 0$ 的任务，B 类为 $b_i < 0$ 的任务。

显然我们应该先尽可能执行全部的 A 类任务，将 T 值尽可能拉高，便于后面执行 B 类任务。

对 A 类任务，显然最优决策为优先完成 t_i 最小的任务。如果期间遇到某个任务 $t_i >$ 目前的 T，由于不可能存在一个未完成的任务可以完成且可以增大 T 值，所以显然无法完成所有任务，输出 $-1s$ 即可。

例题 -魔法

思路

考虑 B 类任务应当用什么策略来完成。

一种比较容易想到的决策是为了使 T 下降尽可能慢，优先完成 b_i 绝对值最小的任务。

但是显然，我们可以轻松构造一些反例否定这个决策。

例如目前 $T = 100000$，有两个 B 类任务。$t_1 = 1000, b_1 = -1, t_2 = 100000, b_2 = -32767$。

显然我们可以先做第二个任务，后做第一个任务来完成所有任务。但是如果按照上面的贪心策略，第一个任务完成后 $T = 99999$，不能完成第二个任务。

例题 -魔法

思路

考虑这一种决策哪里出现了问题。

有可能对于某个 b 绝对值相对较小的任务，需求的 t 也很小。然而某个 b 绝对值略大的任务，需求的 t 很大，比较极限。

那不妨再考虑一下优先完成 t_i 较大的任务。然而容易发现，这种决策也是很容易被举反例否定的。

$T = 100000$，两组任务 $t_1 = 90000, b_1 = -100000, t_2 = 10, b_2 = -10000$。

不难发现应该先完成 t_2 再完成 t_1。

例题 -魔法

思路

考虑这两种决策哪里出了问题。

他们都只考虑了某一种参数，而不考虑另一种参数。

又，对于每个任务只有两种参数。

显然可以考虑到，应该基于这两个参数来决策。

考虑一下这种情况下该如何进行决策。

例题 -魔法

思路

不妨依旧考虑两个任务 $t_1, b_1; t_2, b_2$。

如果当且仅当先完成 2 后完成 1 两个任务才可以全部完成，则会存在：

$$T + b_1 < t_2, T + b_2 > t_1$$

移项得 $T < t_2 - b_1, T > t_1 - b_2$。

故一定存在 $t_1 - b_2 < t_2 - b_1$。

移项得 $t_1 + b_1 < t_2 + b_2$。

所以我们得到了正确的决策：优先完成 $t_i + b_i$ 最大的任务。

如果在完成某个任务时，$T \le t_i$，由上面的推导可知，一定不存在一种方式能够在走到这一步时 T 比目前更大。

例题 -魔法

思路

总结一下思路:

首先按 b_i 是否 ≥ 0 将所有任务分为两类。A 类为 $b_i \geq 0$ 的任务，B 类为 $b_i < 0$ 的任务。

将 A 类任务按照 t_i 由小到大进行排序，逐个完成，并判断能否完成。

将 B 类任务按照 $t_i + b_i$ 由大到小排序，逐个完成，并判断能否完成。

例题 - 魔法

```
#define lint long long
#define pii pair<int, int>

bool cmp(pii x, pii y) {
    return x.first + x.second > y.first + y.second;
}

int solution(int T, int n, int* t, int* b) {
    vector<pii> plus, minus;
    for (int i = 1; i <= n; ++i)
        if (b[i] >= 0)
            plus.push_back(make_pair(t[i], b[i]));
        else
            minus.push_back(make_pair(t[i], b[i]));
```

例题 - 魔法

```
    sort(plus.begin(), plus.end());
    lint cur = T;
    for (int i = 0; i < plus.size(); ++i) {
        if (cur <= plus[i].first)
            return -1;
        cur += plus[i].second;
    }
    sort(minus.begin(), minus.end(), cmp);
    for (int i = 0; i < minus.size(); ++i) {
        if (cur <= minus[i].first)
            return -1;
        cur += minus[i].second;
        if (cur <= 0)
            return -1;
    }
    return 1;
}
```

例题 - [NOIP1999 提高组] 旅行家的预算

题目链接 https://www.luogu.com.cn/problem/P1016

你想要从一座城市到达另一座城市，两座城市距离为 d_1，城市中有 N 个加油站，每个加油站与起点间的距离 d_i 和油单价 p_i 已知。

你的汽车油箱容积为 c 升，每升汽油可以供汽车行驶 d_2，起点站的油价为 p。

求解你是否能够到达目的地。如果可以，输出最小费用，否则输出 “No Solution”。

$N \leq 6$，其余数字 ≤ 500

例题 - [NOIP1999 提高组] 旅行家的预算

思路

首先考虑有无解情况。可以注意到，情况是无解的，当且仅当 $\exists\, i \in [1,n], d_i - d_{i-1} > dist$ $(dist = c \times d_2$，定义起始位置 $d_0 = 0)$。

其次考虑最小费用情况。

由路程下手，显然我们希望对于每段路程 $d_i - d_{i-1}$，这段路程的代价是最小的。

考虑这段路程消耗汽油的来源。显然要想使路程代价最小，汽油一定尽可能地来自于该段路程起点及其之前的油价最低的加油站。

考虑如何计算，发现由于有油箱容量限制，每段路程汽油来源可能十分复杂，计算较难。

例题 - [NOIP1999 提高组] 旅行家的预算

思路

既然想到汽油来源，不妨考虑在每个加油站的加油情况。

对于某个加油站 i，显然可以计算出“在加满油的情况下，由该加油站所能到达的加油站/目的地”。因此考虑能否基于此做出决策。

考虑一下不难发现，只有这两种情况会影响决策：

1. 能够到达的加油站中没有油价更低的加油站
2. 能够到达的加油站中存在比当前加油站油价更低的加油站

对这两种情况分别考虑这两个问题：

1. 在此站加多少油
2. 此站之后去往哪一站

例题 - [NOIP1999 提高组] 旅行家的预算

思路

对于第一种情况，我们知道，至少在“在加满油的情况下，由该加油站所能到达的最远距离”内，使用该加油站的汽油是最合适的。因此这种情况下，对第一个问题，我们直接在此站将油箱加满即可。

考虑最远距离之后的距离，我们一定希望这段距离的费用最少。又，显然我们可以按顺序前往能够到达的任何加油站。因此对第二个问题，我们直接去往这些加油站中油价最低的加油站即可。

例题 - [NOIP1999 提高组] 旅行家的预算

思路

对于第二种情况。假设加油站 i 是油价比当前加油站低的加油站中距离当前加油站最近的点。

显然我们将油加到恰好能够到达 i 的量，然后前往 i 加油站是最优的。

因此这种情况下，第一个问题答案如上，第二个问题答案为 i 加油站。

例题 - [NOIP1999 提高组] 旅行家的预算

思路

总结一下思路。对于一个加油站：

- 如果从该加油站出发，能够到达的加油站中没有油价更低的加油站，则在该加油站加满油箱，然后前往这些加油站中油价最低的加油站。
- 如果从该加油站出发，能够到达的加油站中存在比当前加油站油价更低的加油站，则设加油站 i 是油价比当前加油站低的加油站中距离当前加油站最近的点。将油加到恰好能够到达 i 的量，然后前往 i 加油站。

例题 - [NOIP1999 提高组] 旅行家的预算

```
double d1, c, d2, p;
int n;
double arr_d[15], arr_p[15];
double solution() {
    arr_p[0] = p;
    arr_d[n + 1] = d1;
    double max_dist = c * d2, ans = 0;
    double cur_c = 0;
    int cur_position = 0;
    while (cur_position <= n) {
        int to = 0;
        int alter_to = 0, alter_val = 114514.0;
        for (int i = cur_position + 1; i <= n + 1 &&
            arr_d[i] - arr_d[cur_position] <= max_dist; ++i)
            if (arr_p[i] < arr_p[cur_position]) {
                to = i;
                break;
            } else if (arr_p[i] < alter_val) {
                alter_to = i;
                alter_val = arr_p[i];
            }
```

例题 - [NOIP1999 提高组] 旅行家的预算

```
        if (to) {
            double dis = arr_d[to] - arr_d[cur_position];
            if (cur_c > (dis / d2)) {
                cur_c -= dis / d2;
            } else {
                ans += arr_p[cur_position] * ((dis / d2) - cur_c);
                cur_c = 0;
            }
            cur_position = to;
        } else if (alter_to) {
            double dis = arr_d[alter_to] - arr_d[cur_position];
            ans += arr_p[cur_position] * (c - cur_c);
            cur_position = alter_to;
            cur_c = c - (dis / d2);
        } else {
            return -1;
        }
    }
    return ans;
}
```

总结

我们发现上面的三道题，在解题的过程中，我们总是将题目抽丝剥茧，由大到小考虑。首先将一些决策显然的情况优先处理掉，然后将剩下的核心问题进行考虑。

对于核心问题，我们首先由大局入手，其次基于我们已经得到的问题/情况，层层递进思考，将核心问题分解为一个个子问题。对于子问题，我们逐个攻破，最后得到最终的答案。

什么是已经得到的问题/情况？

- 计算机一定会破坏当前行的最大值，那么这种情况下该怎么办？
- 按 b_i, t_i 中的任何单一种参数排序的策略是错误的，为什么会错误，这种情况下应该怎么办？
- 由于有油箱容量限制，每段路程汽油来源可能十分复杂，这种情况下应该考虑哪一方面？

总结

在解决这些问题的过程中，我们就可以尝试使用贪心做出决策，并提出一些新的问题以供思考。

比如：

- 我们想要在表格中取到尽可能大的值，能否取到？
- 显然要想使路程代价最小，汽油一定尽可能地来自于该段路程起点及其之前的油价最低的加油站。这种情况下如何进行计算？
- 为了使 T 下降尽可能慢，优先完成 b_i 绝对值最小的任务。这种策略是否可行？有没有漏洞？

我们这节课的内容是基础算法的灵活应用。实际上，上面的内容就是将贪心思想融入到我们提出、分析、解决问题的过程当中。

希望各位同学们仔细回味一下我们处理问题的过程。

二分

例题：KC 喝咖啡、聪明的质监员、借教室

概念

二分查找：在一个有序数组中查找某一元素。

在指定的区域之间尝试中间值。如果中间值是答案则直接输出答案；如果中间值太大，则处理左区间；如果中间值太小，则处理右区间。

每次可以将范围缩小至一半，平均时间复杂度 $O(\log n)$。

一般来说，二分答案只能在**有序**数组中查找某一元素。

概念

二分答案与二分查找类似。

有一类题，答案具有有界性、单调性，直接求解很难，但是如果验证某解是否符合题意相对容易。那这类题便可采用二分答案求解。

单调性：

设答案区间 $[1,n]$，存在某一合法解 x，使得 $x+1,x+2,\ldots,n$ 都符合要求，而 $1,2,\ldots,x-1$ 都不符合要求。则 x 为答案，且可以说答案具有单调性。

二分答案本质：每次缩小答案的规模，在答案合法条件下不断逼近最优，得到答案。

概念

一般而言，可用二分答案解决的题目具有以下特征：

- 求最小的最大值
- 求最大的最小值
- 求满足条件的最小（大）值
- 求最接近某值的一个值
- 求最小的能满足条件的代价

例题 – KC 喝咖啡

题目链接 https://www.luogu.com.cn/problem/P1570

给定 n 份调料，每份调料有 v_i, c_i 两个属性。

现要求从这 n 份调料中选择 m 份，使得 $\frac{\sum v_i}{\sum c_i}$ 最大。

$1 \le n \le 200, 1 \le m \le n, 1 \le c[i], v[i] \le 1 \times 10^4$，数据保证答案不超过 1000。

例题 – KC 喝咖啡

思路

发现由于分子分母问题，通过原有的条件推算 m 种选择的调料不太现实。因此可以逆向思考一下，考虑能否通过最终的答案推算是否可以选择 m 种调料以满足要求。

考虑一下式子 $\frac{\sum v_i}{\sum c_i}$。假设最终的答案为 ans。

在一开始，我们由于分母对答案有约束，因此这里考虑能否对分母做一定的处理。

对 $\frac{\sum v_i}{\sum c_i} = ans$ 变形，得到 $\sum v_i = ans \cdot \sum c_i$，进一步推算得 $\sum v_i - ans \cdot \sum c_i = 0$。

貌似一定程度上减弱了分母的影响!

例题 – KC 喝咖啡

思路

考虑 $\sum v_i - ans \cdot \sum c_i = 0$。

将该式拆解为 $(v_1 - ans \cdot c_1) + (v_2 - ans \cdot c_2) + \cdots + (v_m - ans \cdot c_m)$

我们希望 ans 尽可能大。不难发现当 ans 很大时，$v_i - ans \cdot c_i < 0$，这时是不符合要求的。当 ans 很小时，$v_i - ans \cdot c_i > 0$，这时是符合要求的。

因此，我们对于验证 ans 是否成立，只需要判断能否找到 m 份调料，使得 $\sum v_i - ans \cdot \sum c_i \geq 0$ 即可。

不难发现 ans 是符合单调性的，因此直接套二分答案即可。

由于本题在实数域上二分，所以需要注意一下二分的写法。

例题 – KC 喝咖啡

```
struct item {
    double v, c, f;
    item(double var_v = 0, double var_c = 0, double var_f = 0) {
        v = var_v;
        c = var_c;
        f = var_f;
    }
    bool operator< (const item& x) const {
        return f > x.f;
    }
};

int check(double x, int n, int m, item* a) {
    for (int i = 1; i <= n; ++i)
        a[i].f = a[i].v - a[i].c * x;
    sort(a + 1, a + n + 1);
    double cur = 0;
    for (int i = 1; i <= m; ++i)
        cur += a[i].f;
    return cur >= 0;
}
```

例题 – KC 喝咖啡

```
double solution(int n, int m, item* a) {
    double l = 0, r = 1000.0; // 题目给出了答案最大范围
    while (r - l > 1e-5) { // 注意一下特殊二分格式
        double mid = (l + r) / 2;
        if (check(mid, n, m, a))
            l = mid;
        else
            r = mid;
    }
    return l;
}
```

例题 – [NOIP2011 提高组] 聪明的质监员

题目链接 https://www.luogu.com.cn/problem/P1314

给定 n 种矿石，每种矿石有 w_i, v_i 两种属性。给定一个参数值 s 和 m 个区间 $[l_i, r_i]$。要求选出一个参数 W 使得 $|\sum_{i=1}^{m}(\sum_{j=l_i}^{r_i}[w_j \geq W] \times \sum_{j=l_i}^{r_i}[w_j \geq W]\, v_j) - s|$ 最小。求这个最小值。

$$[w_j \geq W] = \begin{cases} 0 & w_j < W \\ 1 & w_j \geq W \end{cases}$$

$$1 \leq n, m \leq 2 \times 10^5, 0 < w_i, v_i \leq 10^6, 0 < s \leq 10^{12}, 1 \leq l_i \leq r_i \leq n$$

例题 – [NOIP2011 提高组] 聪明的质监员

思路

令 $t = \sum_{i=1}^{n}(\sum_{j=l_i}^{r_i}[w_j \geq W] \times \sum_{j=l_i}^{r_i}[w_j \geq W]\, v_j)$

容易发现直接通过计算所需的最小值和对应的 W 不太现实。

但是我们逆向思考一下，如果我们知道了 W，那么我们可以很容易地通过 W 计算出 t，进而计算出 $|t - s|$。

同时考虑这样一个问题：随着 W 增大，t 的变化情况。

不难发现，随着 W 增大，满足 $w_i \geq W$ 的矿石数量单调不增。

那么显然，t 也会单调不增。W 具有单调性。

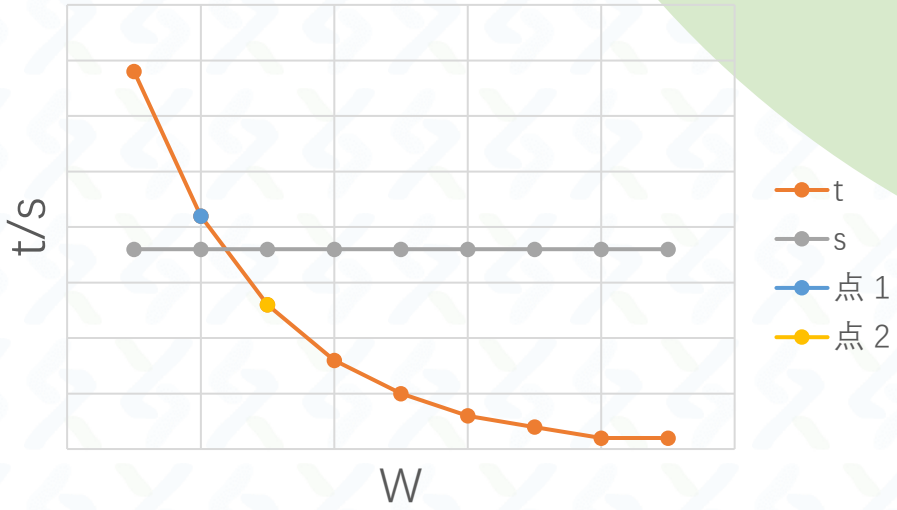
因此我们对 W 二分答案即可。

例题 – [NOIP2011 提高组] 聪明的质监员

思路

考虑一下实现细节。

我们可以粗略地将 W, t, s 的关系绘制成以下图表。



不难发现，所有点中只有点 1，点 2 是有可能满足 $|t - s|$ 最小的点。

例题 – [NOIP2011 提高组] 聪明的质监员

思路

考虑点 1，点 2 的特征。假设两点都不能使得 $t = s$，那么点 1 是最后一个使得 $t > s$ 的点，点 2 是第一个使得 $t < s$ 的点。

显然，我们只需要找出点 $1, 2$ 对应的 t，进行比较判断即可。

所以，我们可以二分找到最后一个使得 $t > s$ 的 W，或第一个使得 $t < s$ 的 W，进而可以找到对应的另一个 W。

那么这道题的思路就清晰了：

二分 W，找到最后一个使得 $t > s$ 的 W 或第一个使得 $t < s$ 的 W，由此找到另外一个，对二者计算 $|t - s|$，取最小值即可。

例题 – [NOIP2011 提高组] 聪明的质监员

思路

但是，如果我们直接暴力计算 t 的话，会得到 70 分的好成绩。

考虑一下如何优化。分析一下我们的算法，二分过程显然不能再优化，那只能考虑对暴力计算 t 优化的过程。

考虑一下过程慢在哪里。显然，对于每个 W，$[w_j \geq W]$ 的情况都是不连续且不确定的，所以统计 $[w_j \geq W]$ 的过程不能优化。

那么考虑剩下的过程。我们知道，一旦 W 确定，那么整个序列的情况也已经确定了，不再变化。这时，如果我们再次暴力查询 m 次 $[l_i, r_i]$，那显然有很多运算是不需要的。

例题 – [NOIP2011 提高组] 聪明的质监员

思路

考虑使用前缀和优化。

使用 $f[i]$ 记录 $\sum_{j=1}^{i}[w_j \geq W]$，$g[i]$ 记录 $\sum_{j=1}^{i}[w_j \geq W] \cdot v_j$。

显然，$f[i] = f[i-1] + [w_i \geq W], g[i] = g[i-1] + [w_i \geq W] \times v_i$。

通过上面的式子，将 i 从 1 到 n 枚举，我们可以在线性的时间求出 $f[1] \sim f[n], g[i] \sim g[n]$。

那么对于每次查询 $[l_i, r_i]$，我们对前半部分，只需要使用 $f[r_i] - f[l_i - 1]$ 即可得到；对后半部分，只需要使用 $g[r_i] - g[l_I - 1]$ 即可得到。各位可以自行推算一下。

那么，对于这 m 次查询，只需要逐个计算 $f[r_i] - f[l_i - 1] \times g[r_i] - g[l_I - 1]$ 即可。

例题 – [NOIP2011 提高组] 聪明的质监员

```
#define lint long long
lint f[200005], g[200005];
lint check(int x, int n, int m, int* w, int* v, int*
var_l, int *var_r) {
    for (int i = 1; i <= n; ++i) {
        if (w[i] >= x)
            f[i] = f[i - 1] + 1, g[i] = g[i - 1] + v[i];
        else
            f[i] = f[i - 1], g[i] = g[i - 1];
    }
    lint ans = 0;
    for (int i = 1; i <= m; ++i) {
        ans += (f[var_r[i]] - f[var_l[i] - 1]) *
(g[var_r[i]] - g[var_l[i] - 1]);
    }
    return ans;
}
```

例题 – [NOIP2011 提高组] 聪明的质监员

```
lint solution(int n, int m, lint s, int* w, int* v, int*
var_l, int *var_r) {
    int l = 1, r = 1000000;
    int ans = 0;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (check(mid, n, m, w, v, var_l, var_r) >= s) {
            ans = mid;
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    lint var1 = abs(check(ans, n, m, w, v, var_l, var_r)
- s);
    lint var2 = abs(check(ans + 1, n, m, w, v, var_l,
var_r) - s);
    return min(var1, var2);
}
```

例题 – [NOIP2012 提高组] 借教室

题目链接 https://www.luogu.com.cn/problem/P1083

共 n 天，每天有 r_i 个教室可用。

有 m 条订单，每条订单要在 $s_i \sim t_i$ 天每天都占用 d_i 个教室。

订单需要按照给出的顺序（即按 $1 \sim m$）依次满足。

计算是否可以满足所有订单。如果不能满足，计算第一个不能满足的订单的序号。

例题 – [NOIP2012 提高组] 借教室

思路

此题与之前两道题不太一样。前面两道题不容易直接计算得到问题的解，但是对于这道题，我们容易想到一个暴力思路进行求解：

枚举 $1 \sim m$，逐个从 $[s_i, t_i]$ 枚举记录每条订单占用的教室数量。当记录到某条订单，使得某一天占用的总教室数量超过 r_i，那么该订单即为所求答案。

显然，我们思考的重心就放到了优化这个暴力过程上。

考虑一下最有可能优化的过程是哪一个。显然，如果按照这个思路，枚举 $1 \sim m$ 不可避免。那么最有可能优化的过程就放在了枚举 $[s_i, t_i]$ 上。

例题 – [NOIP2012 提高组] 借教室

思路

逐个修改是整个过程中最慢的过程，考虑这个过程是否可以优化。

注意到，对于某个特定的订单数，我们的查询只有一次。

对于这种多次逐个修改和单一次查询的过程，特征很明显，我们可以联想到差分。我们将这多个修改转换为标记，最后统一遍历 $1 \sim n$ 即可极大地优化这个过程。

下一个问题是，我们如何进行 $1 \sim m$ 的处理。

注意到，前面我们提到过这个变量具有单调性。并且，根据上面的思路，通过这个变量可以直接验证答案。

二分即可。

例题 – [NOIP2012 提高组] 借教室

思路

总结一下思路:

我们以 $1 \sim m$ 为边界，二分一个变量 x，代表不能够完成的第一份任务的序号。

对于每个 x，我们使用差分计算序列，最后与 $r[i]$ 逐个比较即可。

如果可行，那么 $l = mid + 1$，否则 $r = mid - 1$。

过程中同时统计答案即可。

例题 – [NOIP2012 提高组] 借教室

```
#define pii pair<int, int>
#define lint long long
lint f[1000005];
int check(int x, int n, int* r, int* d, int* s, int* t)
{
    memset(f, 0, sizeof(f));
    for (int i = 1; i <= x; ++i) {
        f[s[i]] += d[i];
        f[t[i] + 1] -= d[i];
    }
    lint cur = 0;
    for (int i = 1; i <= n; ++i) {
        cur += f[i];
        if (cur > r[i])
            return 0;
    }
    return 1;
}
```

例题 – [NOIP2012 提高组] 借教室

```
pii solution(int n, int m, int* r, int* d, int* s, int* t) {
    pii ans;
    int var_l = 1, var_r = m, cur;
    if (check(m, n, r, d, s, t)) {
        ans = make_pair(0, 0);
        return ans;
    }
    while (var_l <= var_r) {
        int mid = (var_l +var_r) / 2;
        if (check(mid, n, r, d, s, t)) {
            var_l = mid + 1;
        } else {
            var_r = mid - 1;
            cur = mid;
        }
    }
    ans = make_pair(-1, cur);
    return ans;
}
```

总结

我们发现上面的三道题，在解题的过程中，与贪心类似，我们总是将题目抽丝剥茧，由大到小考虑。首先将一些决策显然的情况优先处理掉，然后将剩下的核心问题进行考虑。

对于核心问题，我们首先由大局入手，其次基于我们已经得到的问题/情况，层层递进思考，将核心问题分解为一个个子问题。对于子问题，我们逐个攻破，最后得到最终的答案。

这个是一个大致的解题思路。

总结

特别地，对于二分，我们有一些特殊的解题流程。

可以使用二分解决的题目大多具有以下特征：

- 对答案直接求解较难或较繁琐，但是如果知道了答案，逆向验证答案相对简单。
- 答案具有单调性，例如答案上下界为 $1, n$。如果问题的答案是 x，那么 $1 \sim x-1$ 都可行，$x+1 \sim n$ 都不可行（或者 $1 \sim x-1$ 都不可行，$x+1 \sim n$ 都可行）。

上面的例题就是将二分算法融入解题的过程。

仍然是，希望各位同学仔细回味一下我们处理问题的过程。

倍增

例题：「Wdoi-2」死亡之后愈发愉悦

概念

倍增算法，顾名思义，就是不断地翻倍。

具体而言，倍增本质可以表述为，对于一种操作 $f(x)$，通过计算 $f^1(x), f^2(x), f^4(x), \cdots, f^{2^k}(x)$ 来加速求解 $f^n(x)$。

一般来讲，它可以将 $O(n)$ 的计算加速到 $O(\log n)$ 。

例题 -「Wdoi-2」死亡之后愈发愉悦

题目链接 https://www.luogu.com.cn/problem/P8541

定义 $f(x)$ 为严格大于 x 的最小的完全平方数，定义 $g(x)$ 为小于等于 x 的最大完全平方数。

定义一个正整数是“可爱”的，当且仅当 $x - g(x) < f(x) - x$。

现在你需要猜出一个数 a。

你可以做出询问。对于每次询问，输出一个正整数 x，你会收到一个回答，内容是 $a + x$ 是否为“可爱”的正整数。

请通过适当的方式找出 a。

可以暂时不考虑交互次数限制，但是要求使用倍增。

这道题目不要求同学们能够完整地实现代码，只需要体会思想即可。

例题 -「Wdoi-2」死亡之后愈发愉悦

思路

首先我们考虑 $x - g(x) < f(x) - x$ 这个式子。

对一个区间 $[i^2, (i+1)^2)$，容易知道，有且仅有 $2i+1$ 个数 b 满足 $f(b) = (i+1)^2, g(b) = i^2$。

在这 $2i+1$ 个数中，容易知道，前 $i+1$ 个数为可爱数，后 i 个数为非可爱数（$b = i^2 + i$ 是最后一个满足 $b - i^2 < (i+1)^2 - b$ 的可爱数）。

那么，我们将这些数按照是否为可爱数划分为一些区间。对于一个待猜的数 a，我们只要找到 a 在其所在区间中的位置及所在的区间的下一个区间，根据这个区间的长度及区间内元素是否为可爱数即可判断出 a。

例题 -「Wdoi-2」死亡之后愈发愉悦

思路

考虑如何找到上面我们想要的信息。

不难发现，我们只需要找到 a 所在区间的下一个区间的第一个数和最后一个数。

比较显然的，我们需要知道下一个区间内是否是可爱数。所以我们的第一次询问要询问 a 自己是否是可爱数。接下来查找对其取反的值（即后一个区间的值）即可。

重心转移到了如何查找这些值。

一种思路是使用二分，但是在边界未知的情况下使用二分无法保证单调性。

本题开始的时候，要求大家使用倍增。

例题 - 「Wdoi-2」死亡之后愈发愉悦

思路

我们假设 $h(a)=a$ 是否是可爱数。

首先考虑一下：按照 $1,2,4,8,\cdots,2^k$ 的步长向后跳，查询 $a+1,a+2,\cdots,a+2^k$，直到对于某个 $k,h(a)\neq h(a+2^k)$，这时的 2^{k-1} 即为查询的上边界。

确定了上边界后，按照 $2^{k-2},2^{k-3},\cdots,4,2,1$ 的方式倍增查询，可以查询到 a 所在的区间的最后一个数，此时对其 $+1$ 即为下一区间的第一个数。

同样的，我们将下一区间的第一个数设为 l，从 l 开始按照上面的方法查询到 l 所在的区间的最后一个数标记为 r 即可。

这样，我们就确定了 a 所在的区间下一区间的第一个数和最后一个数。

例题 -「Wdoi-2」死亡之后愈发愉悦

思路

现在我们已经知道了下一区间的长度以及是否为可爱数，考虑如何进行推算。

首先，我们知道，对一个区间 $[i^2,(i+1)^2)$，前 $i+1$ 个数为可爱数，后 i 个数为非可爱数。

那么，我们通过判断下一区间是否为可爱数，即可推算出这个 i，进而推算出下一区间的实际起始位置。

在这之后，我们对这个实际起始位置减去 l，即为最终答案位置。

这里画图为大家举例解释整个过程。

例题 -「Wdoi-2」死亡之后愈发愉悦

思路

但是如果按照这种方式写，会获得一个不是满分的成绩。

以下的内容思维难度会较大，各位听不懂的话也没有关系。

这是紫题！能够做到之前那种程度已经很不错了！

我们考虑一下

同样的，我们将下一区间的第一个数设为 l，从 l 开始按照上面的方法查询到 l 所在的区间的最后一个数标记为 r 即可。

在这种情况下，步长是否可以不从 $1, 2, 4, 8 \cdots$ 这样开始？

显然是可以的！我们注意到在 $(r - l)$ 一定 $\geq (l - a)$。那么我们知道，在上一过程的 2^k 内，是一定没有 r 的。那么我们为什么不直接将 2^k 作为初始步长呢？

例题 - 「Wdoi-2」死亡之后愈发愉悦

```c
#define lint long long

int get_val(lint x) {
    printf("? %lld\n", x);
    fflush(stdout);
    int v;
    scanf("%d", &v);
    if (v == -1)
        exit(0);
    return v;
}
```

例题 -「Wdoi-2」死亡之后愈发愉悦

```
void solution() {
    lint v = get_val(0);
    lint step = 0, l = 0, r = 0;
    for ( ; ; ++step) {
        if (get_val(1ll << step) != v)
            break;
        l = (1ll << step);
    }
    lint next_step = step;
    step--;
    for (step--; step >= 0; --step) {
        if (get_val(l + (1ll << step)) != v)
            continue;
        l += (1 << step);
    }
    l++;
```

例题 -「Wdoi-2」死亡之后愈发愉悦

```
r = l; step = next_step;
v = !v;
for ( ; ; ++step) {
    if (get_val(l + (1ll << step)) != v)
        break;
    r = l + (1ll << step);
}
for (--step; step >= 0; --step) {
    if (get_val(r + (1ll << step)) != v)
        continue;
    r += (1 << step);
}
```

例题 - 「Wdoi-2」死亡之后愈发愉悦

```
    lint len = (r - l + 1);
    if (v) {
        printf("! %lld\n", (len - 1) * (len - 1) - l);
        fflush(stdout);
    }
    else {
        printf("! %lld\n", len * len + len + 1 - l);
        fflush(stdout);
    }
}
```

杂题选讲

这部分我留了一些冗余的内容，咱们讲多少算多少
剩余的内容同学们可以课后练习一下
有不懂的问题可以在 QQ 群中讨论提问

以下内容全部来自于洛谷月赛

例题 -做不完的作业

题目链接 https://www.luogu.com.cn/problem/P8508

n 个任务，第 i 个任务需要 t_i 时间。你要在若干天内**依次**完成这些任务。完成任务的时间必须**连续**。

一天有 x 的时间，具体的：

- 每天必须睡觉
- 每天睡觉的时间是连续的，且睡觉结束后第二天恰好开始
- 前 i 天的睡觉总时间不能少于 $r \cdot x \cdot i$，$r = \frac{p}{q}$，会在输入数据给出。

询问：至少需要多少天才能完成全部任务

例题 -做不完的作业

思路

这里提供一种绝妙的思路。

首先注意到一个事实：每段睡觉时间会对其后面的所有天数产生贡献。

那么我们将睡觉时间尽可能放到前面，在后面加班赶工，这样会让睡觉时间发挥最大的价值。

也就是说，我们需要将写作业的时间尽可能往后放。

这是一个显然正确的贪心策略。

例题 -做不完的作业

思路

考虑一下如何实现这个流程。

注意到，如果我们想让作业时间尽可能向前放，那么我们会枚举 $1 \sim n$，从第一天开始，如果当天有剩余的空间就放入，否则就睡觉进入下一天。

类比的，这里我们枚举 $n \sim 1$，从最后一天开始，如果当天有剩余的空间就放入，否则就睡觉进入上一天即可。

至于最后一天如何确定，实际上我们只需要设立一个数组，从下标 1 开始，即可摆脱最后一天的限制，因为最后可以反向枚举。

这里给大家画图演示一下。

例题 -做不完的作业

思路

下一个问题是，我们如何计算所有任务前方需要睡几天觉。

在上面的数组中，我们记录了从最后一天之前的每天的任务量。因此，我们可以进行逐个计算需要的最少睡眠天数。下面考虑如何进行计算。

设睡觉天数为 ans，考虑在睡觉天数后的第 i 天，之前不计算整天睡觉一共休息了 t 单位时间，由题意应满足 $\frac{p}{q}(ans+i)x \leq (t+x\cdot ans)$ 。

变形后得 $(pix-qt) \leq (qx-px)ans$，即 $ans \geq \frac{pix-qt}{qx-px}$。

逐个计算取最大值即可。

下面结合代码解释一下实现细节。

例题 - 做不完的作业

```
#define lint long long
lint a[100005];
lint solution(int n, lint x, lint p, lint q, lint* t) {
    lint tot_days = 1, cur = x, tot_sleep_time = 0;
    for (int i = n; i; --i) {
        if (cur <= t[i])
            ++tot_days, cur = x;
        cur -= t[i];
        a[tot_days] += t[i];
    }
    for (int i = 1; i <= tot_days; ++i)
        tot_sleep_time += x - a[i];
    lint ans = 0;
    for (int i = 1; i <= tot_days; ++i) {
        ans = max(ans, (lint) ceil(1.0 * (p * x * (tot_days - i
+ 1) - q * tot_sleep_time) / (q * x - p * x)));
        tot_sleep_time -= x - a[i];
    }
    return ans + tot_days;
}
```

例题 - 「GLR-R3」雨水

题目链接 https://www.luogu.com.cn/problem/P8475

给定一个长为 n 的整数序列 A，下标从 1 开始。

你可以任取一个自然数 k 以及一个序列 $\{1,2,\ldots,n\}$ 的，长度为 $2k\sim(k\in N)$ 的子序列 P，并对于所有 $i=1,2,\ldots,k$，交换 $A_{P_{2i-1}}$ 与 $A_{P_{2i}}$的值。

求在所有可能得到的新序列 A' 中，字典序最小的序列。

很有意思的贪心构造题。

例题 - 「GLR-R3」雨水

思路

这里提供一种绝妙的思路。

首先我们注意到一个字眼，“要求输出字典序最小的序列”。

提到字典序最小，我们考虑的是优先满足排在序列第一位的元素最小，其次保证第二位最小，如此向后延伸。

那么，从第一位考虑，我们贪心地将后续的最小的元素中最靠后的一个与其交换，这样一定是最优的。

为什么？

如果元素不是最小的，交换后第一位一定不是最优的，字典序一定不是最小的。

如果元素不是最靠后的，单独挑出来这些最小的元素构成序列，则将第一位原有的元素放到最靠后一定比不是最靠后更优。

例题 - 「GLR-R3」雨水

思路

还是刚才的第一位，我们设被交换的元素在 i 位，那么 $2 \sim i-1$ 位的元素在第一次交换后便不可交换。

但是没有关系！由于要求字典序最小，我们已经保证了第一位最小！且在这种情况下，我们保证了后续序列最优！

因此，下一步，我们直接以与第一位相同的方式考虑 $i+1$ 位即可，更下一步亦是如此，直至达到 n。

这种贪心策略，保证了我们构造出的序列一定是字典序最小的系列！

例题 - 「GLR-R3」雨水

思路

考虑一下实现细节，这里为大家提供一种思路。

我们使用两个数组 f, g 分别记录两项数据。$f[i]$ 记录 $i \sim n$ 中最小的元素的值，$g[i]$ 记录 $i \sim n$ 中最小的元素中的最靠后的一个的位置。

不难发现，上面的数组可以 $O(n)$ 预处理好。

处理好后从 1 开始枚举，每次查找 $f[i]$ 是否小于 $a[i]$。如果是，则交换 i 与 $g[i]$ 位置的值，同时下一步让 $i = g[i] + 1$ 继续枚举，直至 n。

最后输出 $(\sum_{i=1}^{n} i \times A'_i) \bmod 2^{64}$ 即可。可以使用 unsigned long long 的自然溢出实现。

例题 - 「GLR-R3」雨水

```
int min_value[MAXN + 5], min_pos[MAXN + 5];
int main() {
    // 读入流程已省略
    min_value[n] = a[n];
    min_pos[n] = n;
    for (int i = n - 1; i; --i)
        if (a[i] < min_value[i + 1])
            min_value[i] = a[i], min_pos[i] = i;
        else
            min_value[i] = min_value[i + 1], min_pos[i] =
min_pos[i + 1];
    for (int i = 1; i <= n; ++i)
        if (a[i] > min_value[i]) {
            swap(a[i], a[min_pos[i]]); i = min_pos[i];
        }
    unsigned long long ans = 0;
    for (int i = 1; i <= n; ++i)
        ans += 1llu * i * a[i];
    printf("%llu\n", ans);
    return 0;
}
```

例题 - 「SvR-1」Problem

题目链接 https://www.luogu.com.cn/problem/P8411

n 道题，第 i 道题有趣程度为 a_i。

对 $i \in [2,n]$，必须先做 fa_i 题才能做 i 题，fa 数组是给出的，且 $1 \leq fa_i < i$。

如果在做一道题之前高兴程度为 k，则他做完第 i 题后，高兴程度会变为 $\min\{k, a_i\}$。在做题前高兴程度为无穷大。

在必须先做第 1 题且不能重复做某一道题的情况下，全过程中每做完一道题后高兴程度总和的最大值。

例题 - 「SvR-1」Problem

思路

注意到 $n \leq 10^7$，所以出题人一定想让我们使用线性的算法。

考虑一个贪心策略：我们每次做当前能够做的题中有趣程度最大的一个，以此类推。

这个贪心策略正确性是显然的。容易发现其余的做法只会使当前的答案更劣。

在这种情况下，我们注意到，能够限制做第 i 题后高兴程度的因素有且仅有它自己以及 fa_i。具体的，我们设做第 i 题后的高兴程度为 f_i，则 $f_i = \min\{f_{fa_i}, a_i\}$。

同时，注意到一个声明：$1 \leq fa_i < i$。那么我们按照 $1 \sim n$ 遍历，即可保证当 f_i 被求解时，f_{fa_i} 一定已被求解。

因此，按照 $1 \sim n$ 遍历求解 f_i，累加即可。

例题 - 「SvR-1」Problem

```
int n;
uint seed;
uint a[10000005], fa[10000005], f[10000005];
#define lint long long
lint ans;
int main(){
    scanf("%d%u", &n, &seed);
    for (int i = 1; i <= n; i++)
        a[i] = get_next(seed);
    for (int i = 2; i <= n; i++)
        fa[i] = get_next(seed) % (i - 1) + 1;
    ans = f[1] = a[1];
    for (int i = 2; i <= n; ++i)
        f[i] = min(a[i], f[fa[i]]), ans += f[i];
    printf("%lld\n", ans);
    return 0;
}
```

例题 - [WFOI - 01] 硬币（coin）

题目链接 https://www.luogu.com.cn/problem/P7996

n 堆硬币，同一堆硬币面值相同为 a_i。每堆硬币数量相同为 x。

已知 a_i，需要确定一个正整数 x，使得每堆硬币的总金额的方差最接近于一个正整数 k。

最接近：两数之差的绝对值最小。

方差：记数据平均数为 $\bar{x}$，方差为 s^2，数据个数为 n，则数列 a 的方差为

$$s^2 = \frac{(a_1 - \bar{x})^2 + (a_2 - \bar{x})^2 + \ldots + (a_n - \bar{x})^2}{n}$$

本题要求使用二分答案求解。

例题 - [WFOI - 01] 硬币（coin）

思路

注意到直接计算 x 相当困难，但是如果我们知道了 x，计算方差会相对简单。

又，我们将方差公式变形，会发现数列整体乘 x 时，方差乘 x^2。符合单调性。

所以很明确的，我们可以尝试使用二分答案解决本题。

考虑一下具体思路。

首先计算原序列的方差 S^2。此后二分 x，对每个 x 计算 $S^2 \times x^2$，找到第一个 $S^2 \times x^2 \geq k$ 的 x。判断 x 和 $x-1$ 孰优孰劣即可。

例题 - [WFOI - 01] 硬币（coin）

思路

本题想让大家体会一下二分答案的思维流程，但是实际上最优解不是二分答案。这里简单谈一下最优解。

如果不考虑 x 为整数的限制，对 $S^2 \times x^2 = k$ 变形，$x = \sqrt{S^2}/k$ 显然是最优解。

但是 x 为整数，所以我们判断一下对上面的 $\sqrt{S^2}/k$ 向上取整向下取整的结果即可。

这道题数据量有一些大，可以使用题面提供的快读模板。

当然，我们建议同学尝试使用二分答案写一下本题。

例题 - [WFOI - 01] 硬币（coin）

```
lint n, k; lint a[7000005];
int main() {
    n = readInt(), k = readInt();
    double cur = 0;
    for (int i = 1; i <= n; ++i)
        a[i] = readInt(), cur += a[i];
    cur /= n;
    double S = 0;
    for (int i = 1; i <= n; ++i)
        S += 1.0 * (cur - a[i]) * (cur - a[i]);
    S /= n;
    if (abs(S) <= 1e-10)
        printf("No answer!\n"), exit(0);
    lint x = floor(sqrt(k / S)), y = ceil(sqrt(k / S));
    if (!x)
        printf("%lld\n", y), exit(0);
    if (abs(S * x * x - k) <= abs(S * y * y - k))
        printf("%lld\n", x);
    else
        printf("%lld\n", y);
    return 0;
}
```

洛谷

前缀和

简单的理解为：数列的前 n 项的和

定义

前缀和可以简单理解为“数列的前 n 项的和”。

它是一种重要的预处理方式，能大大降低查询的时间复杂度。

一般来讲，我们会预处理一个数组。对数组中每个元素，我们记录从起始到该元素对应下标/状态所有数字的总和。

具体来讲，前缀和一般分为以下几种：

定义

- 一维前缀和

给定一个长度为 n 的数组 a，预处理一个 f 数组作为前缀和。那么：

$$f_i = \sum_{j=1}^{i} a_j = a_1 + a_2 + \cdots + a_i$$

- 二维前缀和

给定一个 $n \times m$ 的数组 a，预处理一个 f 二维数组作为前缀和，那么一般来讲：

$$f[i][j] = \sum_{k=1}^{i} \sum_{l=1}^{j} a[k][l] = a[1][1] + \cdots + a[1][j] + a[2][1] + \cdots + a[i][j]$$

- 多维前缀和

可以类推。

实现方式

- 一维前缀和

一般可以在线性时间内解决。枚举 $1 \sim n$，使用以下公式计算：

$$f[i] = f[i-1] + a[i]$$

- 二维前缀和

枚举 $1 \sim n, 1 \sim m$，使用以下公式计算：

$$f[i][j] = f[i-1][j] + f[i][j-1] - f[i-1][j-1] + a[i][j]$$

- 多维前缀和

可以使用容斥解决，不过这已经超出了课程范围。感兴趣的同学可以自己查找资料进行学习。

作用

主要作用：降低查询时的时间复杂度

举例：给定 n 个整数，m 个询问，每个询问求区间 $[l_i, r_i]$ 中数字的和。

如果使用暴力，需要对每个询问从 l_i 枚举到 r_i，时间复杂度会很大。

但是如果我们预处理了一维前缀和，我们可以对每个询问如下回答：

$$ans = f[r_i] - f[l_i - 1]$$

很快！

差分

简单的理解为：前缀和的逆运算

概念

差分可以简单地理解为前缀和的逆运算。

它也是一种重要的处理方式，能大大降低多次修改但最终只有一次查询的时间复杂度。

一般来讲，我们会处理一个数组。对数组中每个元素，我们记录该元素对应下标/状态代表的值与其之前一个的值的差值。

形式化的讲，对一个长度为 n 的数组 a，我们对其建立差分数组 f，结果如下：

$$f_i = \begin{cases} a_1, i = 1 \\ a_i - a_{i-1}, 2 \le i \le n \end{cases}$$

作用

主要作用：多次修改但最终只有一次查询

举例：给定 n 个整数，m 次修改，每次修改将区间 $[l_i, r_i]$ 中数字统一 $+x_i$。

如果使用暴力，需要对每个修改从 l_i 枚举到 r_i，时间复杂度会很大。

但是如果我们按照如下方式使用差分：

$$f[l_i] \mathrel{+}= x_i, f[r_i + 1] \mathrel{-}= x_i$$

在最后查询之前，我们遍历 $1 \sim n$，使用一个 cur 变量记录当前的差分值，然后进行计算即可。

很快！

作用

上面讲的比较抽象，下面我们使用代码来说明一下

假设题目如下：第一行 n, m，第二行 n 个数代表 a 数组，之后 m 行，每行三个数 l_i, r_i, x_i，代表一次修改，求最终数列。

作用

```
int n, m;
int a[maxn];
int l[maxm], r[maxm], x[maxm];
int f[maxn];
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", a + i);
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d%d", l + i, r + i, x + i);
        f[l[i]] += x[i];
        f[r[i] + 1] -= x[i];
    }
    int cur = 0;
    for (int i = 1; i <= n; ++i)
        cur += f[i], a[i] = cur;
    for (int i = 1; i <= n; ++i)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```