

主题

2020—2025 的第一题。

T1 的特点

- 题面短、模型直白、代码几十行
- 六年的 T1 全落在三类形态里：模拟与枚举、贪心与排序、图论与枚举优化
- 分差不在算法上，而在边界与实现细节

T1 (2020—2025)

题目	难度	通过率	知识点
2024 T1 决斗	普及-	48.14%	贪心、排序
2023 T1 密码锁	普及-	41.20%	枚举、模拟
2025 T1 社团招新	普及/提高-	38.09%	贪心、排序
2021 T1 廊桥分配	普及+/提高	24.07%	贪心、有序表、前缀和
2020 T1 儒略日	普及+/提高	19.21%	模拟、二分定位、日期计算
2022 T1 假期计划	提高+/省选-	19.28%	BFS、枚举优化

2024 T1 决斗

知识点：贪心、排序

难度：普及-

题目

今天是小 Q 的生日，他得到了 n 张卡牌作为礼物。这些卡牌属于火爆的「决斗怪兽」，其中第 i 张卡代表一只攻击力为 r_i 、防御力也为 r_i 的怪兽。

一场游戏分为若干回合。每回合，小 Q 会选择某只怪兽 i 以及另一只怪兽 j ($i \neq j$)，并让怪兽 i 向怪兽 j 发起攻击。此时，若怪兽 i 的攻击力小于等于怪兽 j 的防御力，则无事发生；否则，怪兽 j 的防御被打破，怪兽 j 退出游戏不再参与到剩下的游戏中。一只怪兽在整场游戏中至多只能发起一次攻击。当未退出游戏的怪兽都已发起过攻击时，游戏结束。

小 Q 希望决定一组攻击顺序，使得在游戏结束时，未退出游戏的怪兽数量尽可能少。

限制

- $1 \leq n \leq 10^5$
- $1 \leq r_i \leq 10^5$

化为配对

一只怪兽的两种身份互不干扰：它至多发起一次攻击，也只可能被消灭一次。于是「安排攻击顺序」这件事可以看成在怪兽之间做一次**配对**：每一对里，攻击力大的那只消灭攻击力小的那只。

引理 1

游戏结束时未退出的怪兽数量至少是 $n - M$ ，其中 M 是满足下列条件的最大配对数：每只怪兽在所有配对中至多出现一次，且每对中两只怪兽的攻击力**严格**一大一小。

证明：每次成功消灭都消耗掉一只攻击者（它此后再也不能攻击）与一只被消灭者，两者都是原本的怪兽；同一只怪兽重复使用不可能，所以全部成功消灭对应一组配对。反过来，给定一组配对，让每对中攻击力较大的那只去攻击较小的那只即可实现这组配对，中间不会互相干扰。

贪心：排序加双指针

把怪兽按攻击力升序排列： $r_0 \leq r_1 \leq \dots \leq r_{n-1}$ 。

贪心

用两个指针： i 从 0 开始，指向「还没被消灭的最小怪兽」； j 从 1 开始，用来找攻击者。

- 先把 j 往后推，跳过所有与 r_i 相等的怪兽，直到 $r_j > r_i$ 或 $j = n$ ；
- 若 $j < n$ ，就让 j 消灭 i ，并把 j 右移一位（它已经用过一次攻击）；
- 若 $j = n$ ，说明剩下的 $n - i$ 只怪兽再也找不到严格大于自己的攻击者，游戏结束。

证明： 设某个最优解第一次与贪心不一致时，贪心用 j 消灭了当前最小的怪兽 i ，而它让某只怪兽 i' 消灭了别的目标。把这一对换成 j 消灭 i ： r_j 是当前能消灭 i 的最小攻击力，替换后不违反条件，而 i' 被空出来，还能继续与后面尚未被消灭的怪兽配对。于是总存在一个包含贪心选择的最优解，归纳可得贪心最优。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++)
9         cin >> a[i];
10    sort(a.begin(), a.end());
11    for (int i = 0, j = 1; i < n; i++) {
12        while (j < n && a[j] == a[i])
13            j++;
14        if (j == n) {
15            cout << n - i << '\n';
16            break;
17        }
18        // j打死i
19        j++;
20    }
21 }
```

实现要点

- 输出的是「未被消灭的怪兽数」 $n - i$ ，与题目要求的方向一致；不需要像另一种写法那样先求消灭数再相减。
- `while (j < n && a[j] == a[i]) j++;` 这一句是把「严格大于」落实成代码的关键：攻击力相同的怪兽互相打不动，所以 j 要跳过整整一段相等的值。
- j 只会向右移动，全程 $O(n)$ ；加上排序是 $O(n \log n)$ 。
- 循环靠 `break` 结束：一旦 j 到末尾，后面更大的 i 同样找不到攻击者，答案已经确定。

部分分

测试点	n	涉及知识点
1—4	10	枚举、DFS 与 BFS
5—10	20	DFS 与 BFS
11—15	300	搜索的剪枝优化
16—20	10^5	贪心法、排序

官方报告的口径是：前 15 个测试点可以用搜索与剪枝拿到 75 分，满分做法「需要一定的思维能力，但相对容易想到」。

2023 T1 密码锁

知识点：枚举、模拟

难度：普及-

题目

小 Y 有一把五个拨圈的密码锁，每个拨圈上是从 0 到 9 的数字，并且从 0 到 9 循环。

她的锁车方式是从正确密码开始随机转动仅一次：每次以某个幅度仅转动一个拨圈，或者同时转动两个相邻的拨圈，此时两个拨圈转动的幅度相同——可以把 0 0 1 1 5 转成 1 1 1 1 5，但不会转成 1 2 1 1 5。

小 Y 记下了锁车后密码锁的 n 个状态，这 n 个状态都不是正确密码。求可能的正确密码个数：每个这样的密码都要能按上述方式产生给出的全部 n 个状态。

限制

- $1 \leq n \leq 8$
- 每个状态给出五个 0 到 9 之间的整数

转动关系对称

设 s 是一个给出的状态， c 是一个候选密码。把小 Y 的动作看成状态之间的一次转动关系 $s \sim c$ 。

引理 1

若 c 能一次转动得到 s ，则 s 也能一次转动得到 c ；即一次转动关系是对称的。

证明：一次转动只做两件事之一，都是在若干位置上把数字加上同一个幅度 j （模 10）：

- 转一个拨圈：位置集 $\{i\}$ ；
- 转两个相邻拨圈：位置集 $\{i, i + 1\}$ 。

反过来从 s 出发，把同样的位置集上的数字加上 $(10 - j) \bmod 10$ ，就回到 c ；当 $1 \leq j \leq 9$ 时 $(10 - j) \bmod 10$ 也落在 1 到 9 之间，所以这也是一次合法的转动。

正向难，反向容易

要求的是「对每个给出的状态 s ， c 都能一次转动得到 s 」。直接判定一个候选 c 是否满足，需要逐个状态比较不同的位数，还要分类讨论不同的形态。

利用引理 1 把方向倒过来：

做法

对每个给出的状态 s ，枚举从 s 出发转一次能到达的全部状态，把每个到达的状态计数一次。

一个状态 c 被数到的次数，正好是「能一次转动得到 c 的给出状态个数」。于是

$$c \text{ 是可能的正确密码} \iff \text{cnt}[c] = n.$$

给出的状态都不是正确密码这一条不需要单独处理：从 s 出发转一次一定改变至少一个拨圈（幅度 $j \neq 0$ ），所以 s 自己不会被数到。

枚举规模

每个给出状态 s 要枚举多少种转动？

计数

- 幅度 j 取 1 到 9，共 9 种；
- 转一个拨圈：5 种位置；
- 转两个相邻拨圈：4 种位置。

每个状态共 $9 \times (5 + 4) = 81$ 种转动， $n \leq 8$ 时总转动数不超过 648 次。

状态只有五位十进制数，直接编码成一个整数：

$$x = (((((t_0 \cdot 10 + t_1) \cdot 10 + t_2) \cdot 10 + t_3) \cdot 10 + t_4), \quad 0 \leq x < 10^5.$$

用一个长度为 10^5 的数组 `cnt` 计数，最后统计 `cnt[x] = n` 的 x 的个数。总复杂度 $O(81n)$ 。

参考实现（一）

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int cnt[100000];
4 int s[5], t[5];
5 void count() {
6     // {0, 0, 1, 1, 5} => 115
7     int x = 0;
8     for (int i = 0; i < 5; i++)
9         x = x * 10 + t[i];
10    cnt[x]++;
11 }
12
13 int main() {
14     int n;
15     cin >> n;
16     for (int _ = 0; _ < n; _++) {
17         for (int i = 0; i < 5; i++)
18             cin >> s[i];
19         for (int j = 1; j < 10; j++) { //转动幅度
20             // 转一个轮
21             for (int i = 0; i < 5; i++) {
22                 for (int k = 0; k < 5; k++) {
23                     if (k == i)
24                         t[k] = (s[k] + j) % 10;
25                     else
```

参考实现（二）

```
1         t[k] = s[k];
2     }
3     count();
4 }
5 //转两个相邻的轮
6 for (int i = 0; i < 4; i++) {
7     for (int k = 0; k < 5; k++) {
8         if (k == i || k == i + 1)
9             t[k] = (s[k] + j) % 10;
10        else
11            t[k] = s[k];
12    }
13    count();
14 }
15 }
16 }
17
18 int ans = 0;
19 for (int i = 0; i < 100000; i++)
20     if (cnt[i] == n)
21         ans++;
22 cout << ans << '\n';
23
24 return 0;
25 }
```

实现要点

- 计数数组开 10^5 ，下标就是五位状态的十进制编码；`x = x * 10 + t[i]` 这一句保留前导零，例如 `0 0 1 1 5` 编成 115。
- 幅度循环写成 `j = 1..9`，不需要 `j = 0`（那是没转动），也不需要 `j ≥ 10`（模 10 后与 1 到 9 重复）。
- 答案要统计的是 `cnt[x] = n` 而不是 `cnt[x] > 0`：只有能被**全部** n 个状态转到的候选才算正确密码。
- 换方向枚举的好处是判定被换成了计数，不需要判断「不同的位数是 0、1 还是 2」，也就不用讨论那些边界。

2025 T1 社团招新

知识点：贪心、排序

难度：普及/提高-

题目

算法协会招收了 n 个新成员 (n 为偶数)，要把他们分到三个部门。第 i 个成员对第 j 个部门的满意度是 $a_{i,j}$ ，一个分配方案的满意度是所有人对自己部门的满意度之和。

如果第 i 个成员被分到第 $d_i \in \{1, 2, 3\}$ 个部门，则方案的满意度为

$$\sum_{i=1}^n a_{i,d_i}.$$

小 L 不希望某个部门人数过多：方案中不能有部门被分配多于 $\frac{n}{2}$ 个成员。求满足要求的方案里满意度的最大值。

限制

- $1 \leq t \leq 5$
- $2 \leq n \leq 10^5$, n 为偶数
- $0 \leq a_{i,j} \leq 2 \times 10^4$

无约束最优解

先不考虑人数限制：让每个人都选自己最满意的部门，得到的满意度之和记为 S 。这个 S 是一切合法方案的上界，因为每个人在合法方案里的满意度都不超过他的最大值。

问题只剩下：为了满足人数限制，最少要损失多少满意度。

引理 1

在「每人选最优」的方案里，至多只有一个部门的人数超过 $n/2$ 。

证明：若有两个部门的人数都超过 $n/2$ ，则这两个部门的人数之和超过 n ，而所有部门的人数之和恰好是 n ，矛盾。

于是只需处理唯一超标的那个部门。

调出多少人

设超标部门的人数为 c ，需要调出 $k = c - \frac{n}{2}$ 人。

引理 2

恰好调出 k 人一定可行：超标部门之外的部门，人数都不会因为这次调整而超过 $\frac{n}{2}$ 。

证明： 超标部门之外的总人数是 $n - c = \frac{n}{2} - k$ ，所以其中任何部门的人数都不超过 $\frac{n}{2} - k$ 。从超标部门调出 k 人后，任何其他部门至多增加 k 人，于是至多达到

$$\left(\frac{n}{2} - k\right) + k = \frac{n}{2}.$$

调哪些人

一个人从最优部门改到次优部门，满意度会下降

$$\ell_i = a_{i,d_i^*} - \max_{j \neq d_i^*} a_{i,j},$$

称为这个人的**损失**。因为超标部门之外的人数不会超标，所以每个人只需要考虑去次优部门，不必考虑第三选择。

结论

答案为 S 减去超标部门中损失最小的 k 个人的损失之和。

证明：任何合法方案都必须把超标部门的至少 k 个人调出去，而每调出一个人至少损失 ℓ_i ，所以任何方案的总满意度不超过 S 减去「 k 个最小损失之和」。另一方面，取损失最小的 k 个人调出，由引理 2 是合法的，恰好达到这个界。

把每个人的三个满意度排序，取最小的 k 个损失只需要一次排序，总复杂度 $O(n \log n)$ 。

参考实现（一）

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 1e5 + 5;
5 struct S {
6     int myd;
7     int id;
8 };
9 S a[maxn][3];
10
11 bool cmp(S x, S y) {
12     return x.myd > y.myd;
13 }
14
15 void solve() {
16     int n;
17     cin >> n;
18     vector<int> cnt(3);
19     int ans = 0;
20     for (int i = 0; i < n; i++) {
21         for (int j = 0; j < 3; j++) {
22             cin >> a[i][j].myd;
23             a[i][j].id = j;
24         }
```

参考实现（二）

```
1      sort(a[i], a[i] + 3, cmp);
2      cnt[a[i][0].id]++;
3      ans += a[i][0].myd;
4  }
5  for (int i = 0; i < 3; i++)
6      if (cnt[i] > n / 2) {
7          vector<int> loss;
8          for (int j = 0; j < n; j++) {
9              if (a[j][0].id == i) {
10                 loss.push_back(a[j][0].myd - a[j][1].myd);
11             }
12         }
13         sort(loss.begin(), loss.end());
14         ans -= accumulate(loss.begin(), loss.begin() + cnt[i] - n / 2, 0);
15     }
16     cout << ans << '\n';
17 }
18
19 int main() {
20     int T;
21     cin >> T;
22     while (T--) {
23         solve();
24     }
25 }
```

实现要点

- 三个满意度用结构体 `S{myd, id}` 保存并排序：`a[i][0]` 是最优部门，`a[i][1]` 是次优部门，损失就是两者之差；存 `id` 是为了知道最优部门是哪一个。
- `a` 与排序都开在全局 / 函数外层，每组数据都会重新赋值，不需要额外清空。
- 求和用 `int` 就够 ($n \leq 10^5$ 、 $a_{i,j} \leq 2 \times 10^4$ ，总和不超过 2×10^9)，但如果换成 `long long` 更稳妥。
- `accumulate(loss.begin(), loss.begin() + cnt[i] - n / 2, 0)` 取的是排序后最小的 k 个损失之和；`cnt[i] - n/2` 一定不超过 `loss.size()`，因为超标部门里的人数就是 `cnt[i]`。
- 损失也可以用值域计数排序做到线性（值域 2×10^4 ），但 $n \leq 10^5$ 时直接排序已经足够。

部分分

测试点	n	特殊性质
1—4	2 至 10	无
5—8	30	无
9—11	200	9 为 B
12	10^5	A
13—14	10^5	B
15—16	10^5	C
17—20	10^5	无

特殊性质

- A: 只有第 1 个部门有满意度, 其余为零
- B: 第 3 个部门全为零, 只需在两个部门间分配
- C: 满意度在 $[0, 2 \times 10^4]$ 内独立均匀随机

怎么拿分

- 小 n 的测试点允许 3^n 枚举或 DFS, 先把这 20 分拿到
- 特殊性质把三维问题降到一维或二维, 正解的化简方向就在这里

2021 T1 廊桥分配

知识点：贪心、有序表、前缀和

难度：普及+/提高

题目

机场分为国内区与国际区：国内航班只能停靠国内区的廊桥，国际航班只能停靠国际区的廊桥；没有空闲廊桥时飞机停在远机位。

L 市新建的机场一共有 n 个廊桥，廊桥的使用遵循「先到先得」：每架飞机抵达后，如果相应的区还有空闲廊桥就停靠廊桥，否则停远机位。机场只有一条跑道，因此不存在两架飞机同时抵达。

给定未来一段时间两区飞机的抵达与离开时刻，请把 n 个廊桥分配给国内区与国际区，使停靠廊桥的飞机数量最多。

限制

- $1 \leq n \leq 10^5$
- $1 \leq m_1, m_2 \leq 10^5$
- 抵达时刻 a 与离开时刻 b 都是正整数，且 $a < b$

直接做法的代价

分配方案有 $n + 1$ 种，若对每种方案都重新模拟一遍，每次模拟是 $O(m \log m)$ ，总代价 $O(nm \log m)$ ，在 $n = m = 10^5$ 时不可接受。

引理 1

国内区拿到 i 个廊桥时能停靠的飞机数 $f(i)$ ，与国际区拿到多少个廊桥无关。

证明：国内航班只停国内区，国际航班只停国际区，两区的飞机互不竞争，模拟过程完全独立。给定国内区 i 个廊桥后，国内区能停多少架只由国内航班自身的时刻决定。

于是只需要两个数组 $f(0 \dots n)$ 与 $g(0 \dots n)$ ，答案是

$$\max_{0 \leq i \leq n} (f(i) + g(n - i)).$$

逐个廊桥贪心

把「先到先得」按廊桥拆开看：第 1 个廊桥服务的飞机，是「抵达最早的飞机，接着是抵达时刻在其离开之后的最早飞机，再接着……」

引理 2

设 c_k 为「按下列方式给第 k 个廊桥安排的飞机数」：从时刻 0 开始，每次在尚未安排的飞机中取抵达时刻最早的一架，安排给它，然后把当前时刻推进到这架飞机的离开时刻；直到没有飞机的抵达时刻晚于当前时刻。则

$$f(i) = c_1 + c_2 + \cdots + c_i.$$

证明思路（对 i 归纳）：第 1 个廊桥拿到的一定是抵达最早的飞机；把它的整条链去掉之后，剩下的问题与原问题同形。而「先到先得」中，若 i 个廊桥能停靠的飞机集合固定，那么编号最小的那个廊桥必然服务这条链——否则可以把链上的飞机整体换到编号最小的空闲廊桥上，不减少总数。

有序表实现

数据结构

把所有航班存进一张以**抵达时刻**为键、离开时刻为值的 `map`。由于题目保证只有一条跑道，全机场不存在同时抵达的飞机，所以抵达时刻互不相同，可以直接作键。

每个廊桥的模拟

对第 i 个廊桥：令当前时刻 $x = 0$ ，反复用 `t.upper_bound(x)` 取「抵达时刻严格大于 x 」的最早航班，

- 取不到就结束这个廊桥；
- 取到就让它停靠（`cnt[i]++`），把 x 更新为它的离开时刻，并从 `map` 里删掉它。

最后对 `cnt` 求前缀和，得到 $f(1), f(2), \dots, f(n)$ 。

总代价 $O(n \log m)$ 次 `map` 操作加上 $O(m \log m)$ 的建表。

参考实现（一）

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int n;
5
6 vector<int> solve(int m) {
7     map<int, int> t;
8     for (int i = 0; i < m; i++) {
9         int a, b;
10         cin >> a >> b;
11         t.insert({a, b});
12     }
13     vector<int> cnt(n + 1); // cnt[i] : 停在第i个廊桥的飞机的数量
14     for (int i = 1; i <= n; i++) {
15         int x = 0; // 当前时刻
16         while (1) {
17             auto it = t.upper_bound(x);
18             if (it == t.end())
19                 break;
20             cnt[i]++;
```

参考实现（二）

```
1     x = it->second;
2     t.erase(it);
3 }
4 }
5 for (int i = 1; i <= n; i++)
6     cnt[i] += cnt[i - 1];
7 return cnt;
8 }
9
10 int main() {
11     int m1, m2;
12     cin >> n >> m1 >> m2;
13     vector<int> c1 = solve(m1);
14     vector<int> c2 = solve(m2);
15
16     int ans = 0;
17     for (int i = 0; i <= n; i++)
18         ans = max(ans, c1[i] + c2[n - i]);
19     cout << ans << '\n';
20 }
```

实现要点

- `upper_bound(x)` 是「抵达时刻**严格大于** x 」：一架飞机在时刻 t 离开，则另一架在时刻 t 抵达的飞机不再复用它占用的廊桥。想清楚这一点再用，是这道题最容易写错的地方。
- `map` 以抵达时刻为键，这一点依赖题目「只有一条跑道、不存在同时抵达」的保证；若数据里出现相同抵达时刻，`insert` 会丢掉后一架飞机。
- `solve` 直接在函数里读入：两区各调用一次，各自读自己的 m 行，不需要先把数据存下来。
- 前缀和的语义是「前 i 个廊桥一共停靠的飞机数」，正是引理 2 里的 $f(i)$ ； n 可能大于 m ，此时数组后半段都是 $f(m)$ ，取用时不会越界。

2020 T1 儒略日

知识点：模拟、二分定位、日期计算

难度：普及+/提高

题目

天文学家们使用儒略日来表达时间。儒略日定义为从公元前 4713 年 1 月 1 日正午 12 点起，到此后的某一时刻间所经过的天数，不满一天者用小数表达。给定一个不含小数部分的儒略日，请计算它对应的公历日期。

输出时，公元后的年份写成 日 月 年，公元前年份写成 日 月 年 BC。

限制

- $1 \leq Q \leq 10^5$
- $0 \leq r_i \leq 10^9$ ，年份答案不超过 10^9

公历规则

- 公元 1582 年 10 月 15 日（含）以后适用格里高利历：当年份是 400 的倍数，或是 4 的倍数但不是 100 的倍数时，该年为闰年。
- 公元 1582 年 10 月 5 日（含）至 10 月 14 日（含）不存在，这些日期被删除，该年 10 月 4 日之后为 10 月 15 日。
- 公元 1582 年 10 月 4 日（含）以前适用儒略历：只要年份是 4 的倍数就是闰年。

注意公元零年并不存在，即公元前 1 年的下一年是公元 1 年。因此公元前 1 年、前 5 年、前 9 年……的年份应视为闰年。

天文纪年与闰年

用**天文纪年**表示年份：公元前 1 年记作 $y = 0$ ，公元前 2 年记作 $y = -1$ ，依此类推。这样「公元前 1、5、9……年是闰年」就等价于

$$y \leq 0 \text{ 且 } (-y) \bmod 4 = 1.$$

闰年判定 `is_run`

$$\text{is_run}(y) = \begin{cases} (-y) \bmod 4 = 1, & y < 0, \\ y \bmod 4 = 0, & 0 \leq y \leq 1582, \\ y \bmod 400 = 0 \text{ 或 } (y \bmod 4 = 0 \text{ 且 } y \bmod 100 \neq 0), & y > 1582. \end{cases}$$

年首天数

年首天数 n_day

$D(y)$ 表示从儒略日 0（公元前 4713 年 1 月 1 日）到 y 年 1 月 1 日经过的天数：

$$D(y) = (y + 4713 + [y < 0]) \times 365 + L(y) - 10 \cdot [y \geq 1582],$$

其中 $L(y)$ 是 y 年之前（不含 y 年）的闰年个数， $[y < 0]$ 用来补上不存在的公元零年，最后一项扣掉 1582 年被删除的 10 天。

n_day 就是按这个式子算的：先用「年数乘 365」把整年跳过去，再加上闰年多出来的天数。

闰年前缀

$L(y)$ 要能以 $O(1)$ 求出，否则二分里每次判定都会变慢。把 y 分成两段：

$L(y)$ 的计算

- $y \leq 1600$: 这一段的年份都在儒略历范围内（1583 到 1600 只有 18 年，按儒略历多算的闰年不影响答案，因为 1600 是 400 的倍数），直接用「每 4 年一个闰年」：

$$L(y) = \left\lfloor \frac{4713 + y + [y < 0] + 3}{4} \right\rfloor.$$

- $y > 1600$: 在 $L(1600)$ 的基础上按格里高利历修正：

$$L(y) = L(1600) + \left\lfloor \frac{y - 1600}{4} \right\rfloor - \left\lfloor \frac{y - 1600}{100} \right\rfloor + \left\lfloor \frac{y - 1600}{400} \right\rfloor.$$

于是 $D(y)$ 是 y 的单调不减函数，可以对它做二分。

二分年份

要求的是第 $r + 1$ 天，即找

$$y^* = \min\{y : D(y) \geq r + 1\}.$$

二分

在 $[-4713, 10^9]$ 上二分 y ，判定写成 $D(\text{mid}) \geq r + 1$ 。注意跳过不存在的公元 0 年：

- 若 $\text{mid} = 0$ 就把它换成 1；
- 收缩边界时若 r 或 l 落到 0，就挪到 -1 或 1。

单个询问的二分代价是 $O(\log 10^9)$ 。

年内逐日

算出年份之后，从 $D(y^* - 1)$ 出发 ($y^* = 1$ 时取 $D(-1)$ ，同样跳过公元 0 年)，在这一年里按月、按日累加：

三个细节

- 闰年要把 2 月改成 29 天，用 `is_run( $y^*$ )` 判定；
- 累加到 1582 年 10 月时跳过 5 日到 14 日这 10 天；
- 计数器加到 $r + 1$ 就输出 日 月 年， $y^* < 0$ 时写成 `abs( $y^*$ )` 年 BC。

单个询问的总代价是 $O(\log 10^9 + 366)$ ， $Q = 10^5$ 时约为 3.7×10^7 次运算。

参考实现（一）

```
1  int n_run(int y) {                                // y 年之前（不含 y 年）的闰年个数
2      if (y <= 1600) {
3          return (4713 + y + (y < 0) + 3) / 4;
4      }
5      y -= 1600;
6      return n_run(1600) + y / 4 - y / 100 + y / 400;
7  }
8
9  bool is_run(int y) {
10     if (y < 0)
11         return -y % 4 == 1;
12     if (y <= 1582)
13         return y % 4 == 0;
14     return y % 400 == 0 or (y % 4 == 0 and y % 100 != 0);
15 }
16
17 int days[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
18
19 long long n_day(int y) {                            // 儒略日 0 到 y 年 1 月 1 日的天数
20     long long ans = (y + 4713 + (y < 0)) * 365LL + n_run(y);
21     if (y >= 1582)
22         ans -= 10;
23     return ans;
24 }
25
```

参考实现（二）

```
1 void solve(long long x) {
2     int l = -4713, r = 1000000000;
3     // 最小的 y 使 n_day(y) >= x + 1
4     while (l <= r) {
5         int y = (l + r) / 2;
6         if (y == 0)
7             y = 1;
8         if (n_day(y) >= x + 1) {
9             r = y - 1;
10            if (r == 0)
11                r = -1;
12        } else {
13            l = y + 1;
14            if (l == 0)
15                l = 1;
16        }
17    }
18    int y = l;
19    int py = y == 1 ? -1 : y - 1;
20    long long c = n_day(py);
```

参考实现 (三)

```
1  for (int m = 1; m <= 12; m++) {
2      int t = days[m];
3      t += m == 2 and is_run(y);
4      for (int d = 1; d <= t; ++d) {
5          if (y == 1582 and m == 10 and d >= 5 and d < 15)
6              continue; // 1582-10-05 至 10-14 不存在
7          if (++c == x + 1) {
8              cout << d << ' ' << m << ' ' << abs(y) << (y < 0 ? " BC\n" : "\n");
9              return;
10         }
11     }
12 }
13 }
14
15 int main() {
16     int n;
17     cin >> n;
18     for (int i = 0; i < n; ++i) {
19         long long x;
20         cin >> x;
21         solve(x);
22     }
23 }
```

两种做法

二分加年内逐日（本页代码）

- 一个「年首天数」函数加一次二分，思路直白
- 年内最多枚举 366 天，边界集中在两处：跳过公元 0 年、跳过 1582 年缺失的 10 天
- 每个询问 $O(\log 10^9 + 366)$

周期定位

- 用 4 年 1461 天与 400 年 146097 天两个周期整体跳
- 每个询问 $O(1)$ ，但常量多（78、287、6287）
- 代码见 `代码/02_julian.cpp`

两份实现在 2 万组随机 r （含 0、2299160、2299161、 10^9 等边界）上逐项比对，结果一致。

实现要点

- 边界自查： $r = 2299160$ 应得 `4 10 1582`， $r = 2299161$ 应得 `15 10 1582`。
- 天文纪年与输出格式的换算：内部 $y = 0$ 是公元前 1 年，输出时写 `abs(y)` 并补 `BC`； $y < 0$ 时闰年判据是 `-y % 4 == 1`。
- 年份用 `int`、天数用 `long long`：年份答案可到 10^9 ，天数可到 3.6×10^{11} 。
- 二分里的公元 0 年处理容易被忽略：`mid = 0` 要当成 1，收缩边界时也要避开 0，否则二分可能停在不存在的那一年上。

2022 T1 假期计划

知识点：BFS、枚举优化

难度：提高+/省选-

题目

地图上有 n 个点，1 号是小熊的家，其余都是景点。点对之间有双向直达线路；若 x 与 y 之间可以经过 k 条线路通达，就说它们之间可转车 k 次通达（有直达线路即可转车 0 次通达）。

小熊要从家出发游玩 4 个不同的景点再回家，行程固定为

$$\text{家} \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow \text{家},$$

每段行程最多转车 k 次。转车时经过的点没有限制，可以重复经过。每个景点有分数，求访问的四个不同景点的分数之和的最大值。

限制

- $5 \leq n \leq 2500$
- $1 \leq m \leq 10^4$
- $0 \leq k \leq 100$
- 景点分数 $1 \leq s_i \leq 10^{18}$

可达性预处理

观察

一段行程最多转车 k 次，等价于两个点之间的最短路径边数不超过 $k + 1$ 。

从每个点出发做一次 BFS，把距离存进二维数组：

$\text{dist}[i][j] = i$ 到 j 的最短路径边数， $\text{dist}[i][j] \leq k + 1$ 表示可达。

一步的代价是 $O(n(n + m))$ ，在 $n = 2500$ 、 $m = 10^4$ 时约为 3×10^7 ，可以接受。数组大小是 2500×2500 的 `int`，约 25 MB。

注意

不可达的标记不能用「 $n + 5$ 」这类哨兵值。当 k 很大（例如 $k = 100$ 而 n 很小）时所有点都可达，哨兵会被误判为可达；初值统一用一个大数（如 10^9 ）。

暴力枚举的代价

有了 dist 之后，最直接的做法是枚举四个景点：

$$\text{ans} = \max_{\substack{A,B,C,D \text{ 互不相同} \\ \text{dist}[1][A], \text{dist}[A][B], \text{dist}[B][C], \text{dist}[C][D], \text{dist}[D][1] \leq k+1}} s_A + s_B + s_C + s_D.$$

代价是 $O(n^4)$ ： $n = 20$ 时约 1.6×10^5 ，能过前 8 个测试点； $n = 2500$ 时是 4×10^{13} ，不可能。

测试点表的规模提示了目标复杂度：最后 6 个测试点 $n = 2500$ ，必须是 $O(n^2)$ 级别。

测试点	n	m	k
1—8	≤ 20	≤ 50	0 或 100
9—11	300	1000	0
12—14	100	1000	100
15—20	2500	10000	0 或 100

枚举中间两点

固定中间的两个景点 B 、 C 之后， A 与 D 的选择互相独立：

- A 要满足 $\text{dist}[1][A] \leq k + 1$ 且 $\text{dist}[A][B] \leq k + 1$ ；
- D 要满足 $\text{dist}[C][D] \leq k + 1$ 且 $\text{dist}[D][1] \leq k + 1$ 。

引理 1

对每个点 i ，把「与 1 和 i 都可达」的景点按分数从大到小排序，只保留前 3 个，记作 $\text{cand}[i]$ 。枚举 B 、 C 时， A 在 $\text{cand}[B]$ 里取、 D 在 $\text{cand}[C]$ 里取即可。

证明： 设最优解里的 A^* 不在 $\text{cand}[B]$ 中。 $\text{cand}[B]$ 里的 3 个点分数都不低于 s_{A^*} ，其中至多有 2 个与 C 、 D^* 重合（ A 只需避开这两个点），所以至少还有一个可用的候选 A' ，把 A^* 换成 A' 不降低总分。对 D 同理。

枚举顺序

枚举

外层枚举无序的中间点对 (B, C) ，只取 $B < C$ ，判定 $\text{dist}[B][C] \leq k + 1$ ；内层把 $\text{cand}[B]$ 与 $\text{cand}[C]$ 两两配对，检查 A, B, C, D 四个景点互不相同，更新答案。

只枚举 $B < C$ 不会漏解：把整条行程反过来读，家 $\rightarrow D \rightarrow C \rightarrow B \rightarrow A \rightarrow$ 家 也是一条合法行程（图是无向的，四条中间边的可达性不变），于是原行程要么本身满足 $B < C$ ，要么它的反向满足。

这一步的代价是 $O(n^2 + 9n^2)$ ，加上 BFS 预处理，总计 $O(n(n + m) + n^2)$ 。

参考实现（一）

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 2500 + 5;
5 int n, m, k;
6 vector<int> g[maxn];
7 long long p[maxn];
8 vector<int> cand[maxn];
9 int dist[maxn][maxn];
10 const int INF = 1e9;
11
12 void bfs(int s) {
13     for (int i = 1; i <= n; i++)
14         dist[s][i] = INF;
15     queue<int> q;
16     q.push(s);
17     dist[s][s] = 0;
18     while (!q.empty()) {
19         int u = q.front();
20         q.pop();
21         for (int v : g[u])
22             if (dist[s][v] == INF) {
23                 dist[s][v] = dist[s][u] + 1;
24                 q.push(v);
25             }
26     }
27 }
28
```

参考实现（二）

```
1 bool cmp(int i, int j) {
2     return p[i] > p[j];
3 }
4
5
6 // cand[i] : 能到点1和点i的点的列表
7 // g[i] 是一个 vector<int>, 是点i的邻接表
8 int main() {
9     cin >> n >> m >> k;
10    for (int i = 2; i <= n; i++)
11        cin >> p[i];
12    for (int i = 0; i < m; i++) {
13        int u, v;
14        cin >> u >> v;
15        g[u].push_back(v);
16        g[v].push_back(u);
17    }
18    for (int i = 1; i <= n; i++)
19        bfs(i);
20
21    vector<int> a;
22    for (int i = 2; i <= n; i++)
23        if (dist[1][i] <= k + 1)
24            a.push_back(i);
```

参考实现 (三)

```
1
2  sort(a.begin(), a.end(), cmp);
3
4  for (int i = 2; i <= n; i++)
5      for (int v : a) {
6          if (v != i && dist[i][v] <= k + 1) {
7              cand[i].push_back(v);
8              if (cand[i].size() == 3) break;
9          }
10     }
11
12     long long ans = 0;
13
14     for (int c = 2; c <= n; c++)
15         for (int d = c + 1; d <= n; d++)
16             if (dist[c][d] <= k + 1)
17                 for (int b : cand[c])
18                     for (int e : cand[d])
19                         if (b != d && e != c && b != e) {
20                             // if (p[b] + p[c] + p[d] + p[e] > ans)
21                             //     cout << b << ' ' << c << ' ' << d << ' ' << e << '\n';
22                             ans = max(ans, p[b] + p[c] + p[d] + p[e]);
23                         }
24     cout << ans << '\n';
25 }
```

部分分做法

先写 $O(n^4)$ 的枚举，可以直接通过前 8 个测试点：

```
1 long long ans = 0;
2 for (int A = 2; A <= n; A++) if (can[1][A])
3     for (int B = 2; B <= n; B++) if (B != A && can[A][B])
4         for (int C = 2; C <= n; C++) if (C != A && C != B && can[B][C])
5             for (int D = 2; D <= n; D++)
6                 if (D != A && D != B && D != C && can[C][D] && can[D][1])
7                     ans = max(ans, s[A] + s[B] + s[C] + s[D]);
8 cout << ans << '\n';
```

它的代码量很小，而且顺带把可达性预处理验证了一遍；在正解写出来之后，它还可以作为小数据上的参照。

实现要点

- 结构上分三步：每个点一次 BFS 求出 `dist`；按分数从高到低筛出每个点的前 3 个候选 `cand`；枚举中间点对并配对候选。
- 候选表 `cand[i]` 里只存 3 个点，靠 `if (cand[i].size() == 3) break;` 提前结束；候选取自按分数排序的数组 `a`，顺序天然是从高到低。
- 四个景点互不相同的约束在配对时逐条判：`b != d` ($A \neq C$)、`e != c` ($D \neq B$)、`b != e` ($A \neq D$)； $A \neq B$ 、 $C \neq D$ 由候选表本身排除（候选里不含 i 自己）。
- 分数可达 10^{18} ， p 与答案都用 `long long`；`dist` 只用 `int`， 2500×2500 约 25 MB。
- 与 $O(n^4)$ 暴力在 150 组随机小图上逐一比对，结果一致。

take away

- **看数据范围定复杂度**：先判断该写 $O(n \log n)$ 还是 $O(n^2)$ ，再想做法
- 三类套路，六道题各归一类**
- **模拟**：把过程拆成周期整体跳过——儒略日
 - **枚举**：把判定换成计数（密码锁）、把枚举压成候选（假期计划）
 - **贪心**：把选择化成配对（决斗）、先取最优再按最小损失调整（社团招新）、逐个资源链式贪心加前缀和（廊桥分配）
 - **一套查错习惯**：数据类型与溢出、等号与取整、多组数据的清空

相似的题

这道题	相似题	解法核心
2024 T1 决斗	P1309 [NOIP2011 普及组] 瑞士轮	排一次序，之后用双指针线性扫，不再重排
2023 T1 密码锁	P2010 [NOIP2016 普及组] 回文日期	枚举一个维度，其余维度由约束直接推出，再逐条判定
2025 T1 社团招新	P5019 [NOIP2018 提高组] 铺设道路	先取局部最优，再用交换论证或差分说明全局最优
2021 T1 廊桥分配	P2827 [NOIP2016 提高组] 蚯蚓	用堆或单调结构维护「下一个该取出的元素」，把查找压到 $O(\log n)$
2020 T1 儒略日	P3952 [NOIP2017 提高组] 时间复杂度	规则多的大模拟：把规则写成状态与转移，边界靠清单逐条核对
2022 T1 假期计划	B4428 [CSP-X2025 山东] 勇者斗恶龙	证明最优解只需取值于常数大小的候选集合，把枚举压下来