



# 2020牛客暑期多校训练营（第八场）

邝斌、王冬杨、邓思钊、施毅恒、  
唐京、刘子文





# Results

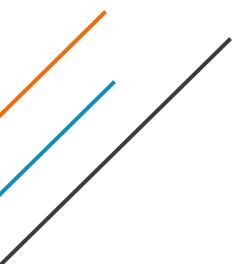
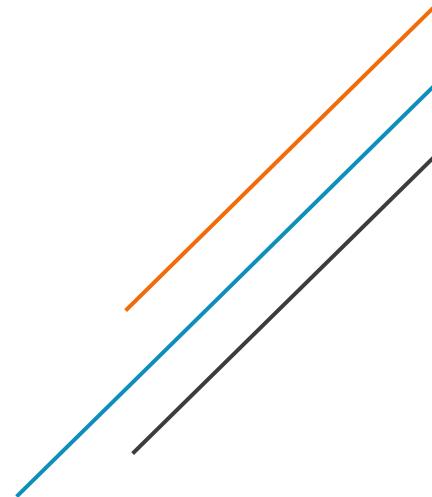
题目	通过率
A - All-Star Game	26/180
B - Bon Voyage	1/10
C - Cinema	1/144
D - Disgusting Relationship	3/17
E - Enigmatic Partition	51/316
F - Factorio	1/30
G - Game SET	365/1073
H - Hard String Problem	3/59
I - Interesting Computer Game	630/5775
J - Jumping Points	1/23
K - Kabaleo Lite	639/4596



# I – Interesting Computer Game

- 题目大意：

- 给了两个数组： $\{a_1, a_2, \dots, a_n\}, \{b_1, b_2, \dots, b_n\}$ .  $n \leq 10^5$
- 第*i*步可以从 $a_i$ 和 $b_i$ 中选择一个数。
- 求最后选出的数中，不同的数要最多。

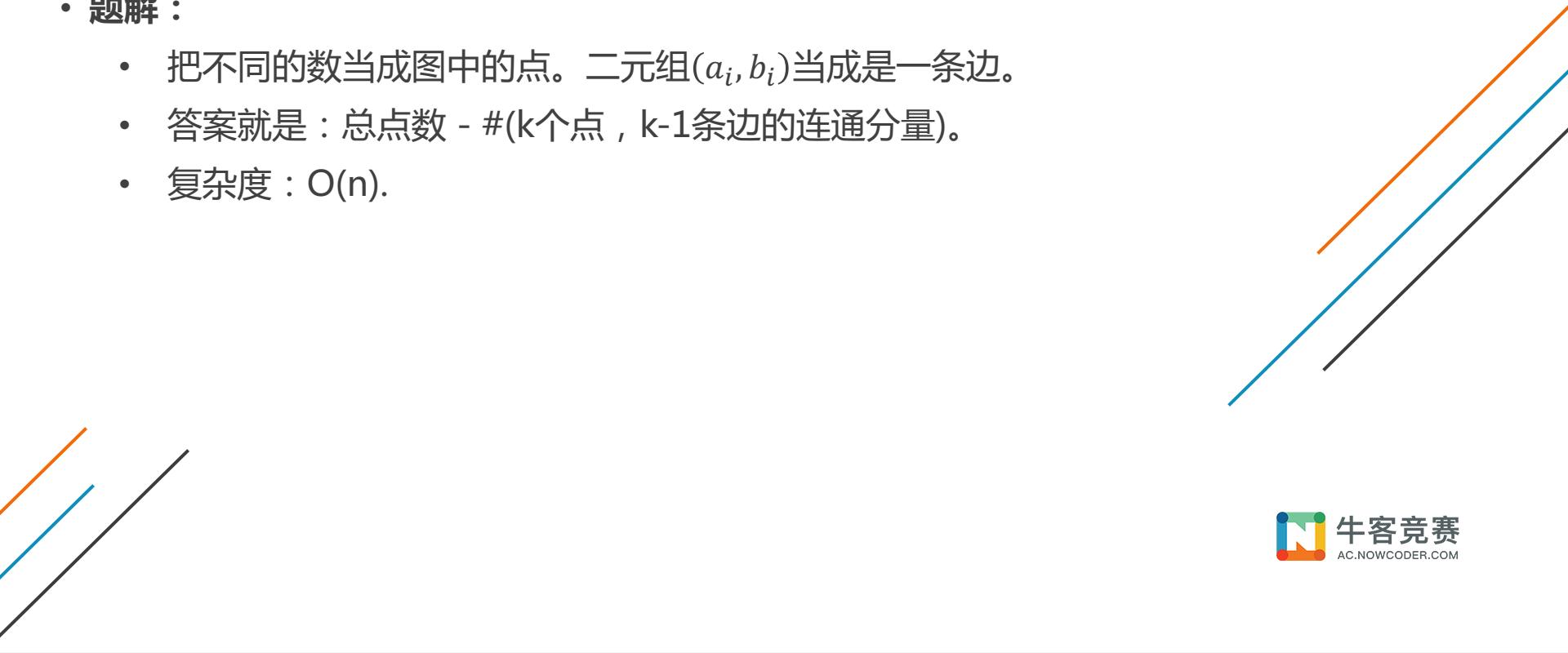




# I – Interesting Computer Game

- 题解：

- 把不同的数当成图中的点。二元组 $(a_i, b_i)$ 当成是一条边。
- 答案就是：总点数 - #(k个点， k-1条边的连通分量)。
- 复杂度 :  $O(n)$ .

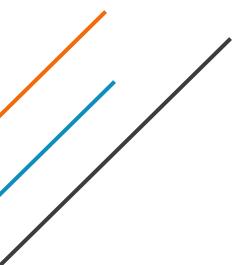
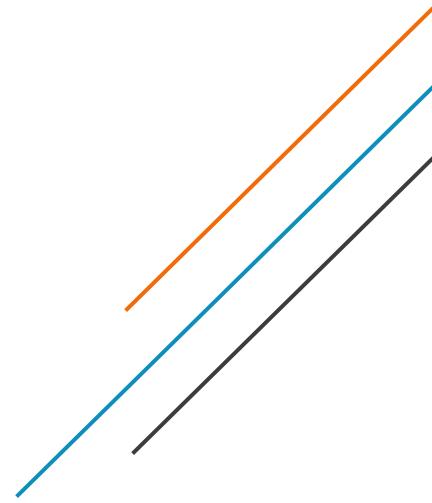




## K – Kabaleo Lite

- 题目大意：

- 有 $n$ 种菜，每种菜的数量为 $b_i$ , 每个菜的盈利为  $a_i$ .
- 每个顾客必须从第1种菜开始，连续地吃，每种吃一个。
- 保证顾客最多的情况下，盈利最大。

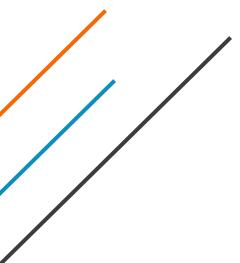
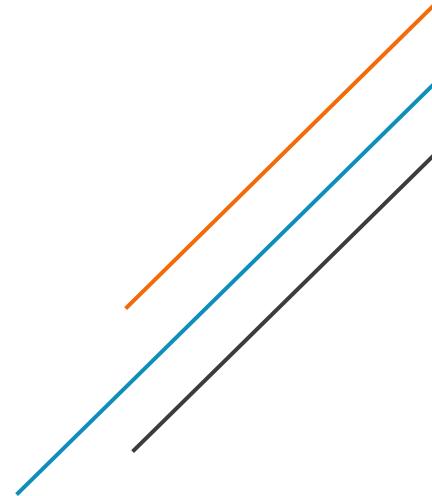




## K – Kabaleo Lite

- 题解：

- 简单题。
- $b_1$ 就是最大顾客数量。然后求盈利的前缀和，从大到小取即可。
- 注意结果会超出long long.
- 复杂度 $O(n)$  或者  $O(n\log n)$ 都可以过。

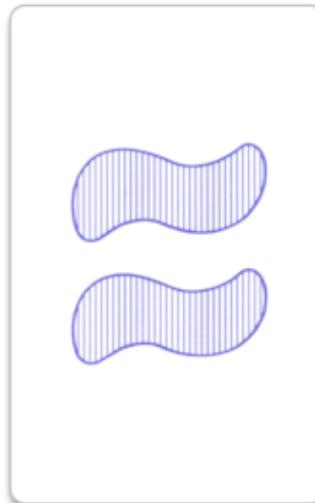
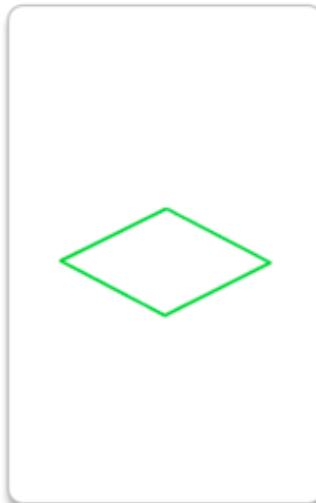




## G – Game SET

- 题目大意

- 给了n张SET牌，包含了万能牌，输出任意一种可以够成SET的方案。
- number of shapes, shape, shading, and color





## G – Game SET

- 题解

- 暴力出奇迹，直接冲！！！三层循环枚举，然后判断。
- $N \leq 256, T \leq 1000$





# G – Game SET

- 题解

- 虽然 $N \leq 256$ ，不能构成SET的集合最大就是20. 枚举21张肯定可以找到SET
- 详细证明：[here](#)
- 最大复杂度就是  $T * 21^3$ .

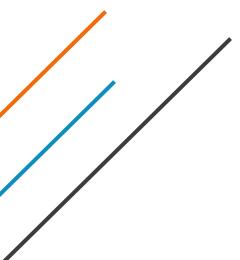
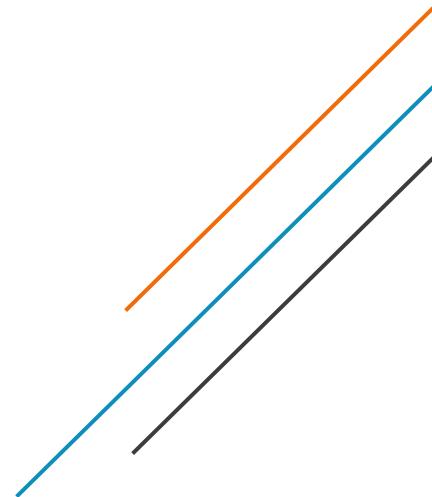
```
for (int i = 1; i <= min(n, 21); i++) {
    if (ff) break;
    for (int j = i+1; j <= min(n, 21); j++) {
        for (int k = j+1; k <= min(n, 21); k++) {
            if (isSet(i, j, k)) {
                ff = true;
                printf("Case #%d: %d %d %d\n", iCase, i, j, k);
                break;
            }
        }
        if (ff) break;
    }
}
```



## E – Enigmatic Partition

- 题目大意

- 数n的拆分，需要满足最大和最小差2，相邻两个数的差值不能超过1.
- $f(n)$ 表示这种拆分的个数。给定 $l, r$ , 求区间和。
- $T \leq 10,000. \quad 1 \leq l \leq r \leq 100,000$

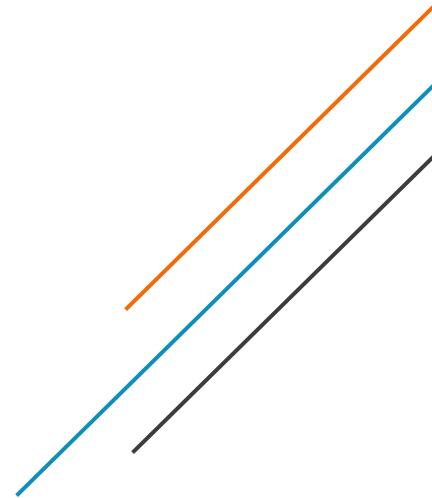




## E – Enigmatic Partition

- 题解

- 拆分肯定由三种连续的数构成，比如 $l, l + 1, l + 2$
- 可以通过枚举 $l$ 去初始化 $f(n)$ .





# E – Enigmatic Partition

- 题解

- 拆分
- 可以

```
for (int l = 1; l <= MAXN; l++) {  
    int mid = l+1;  
    int r = l+2;  
    for (int nl = 1; nl * l <= MAXN; nl++) {  
        for (int m = 3; m*mid + l*nl <= MAXN; m++) {  
            sum[m*mid+l*nl] += (m-1)/2;  
        }  
    }  
    for (int nr = 0; nr*r <= MAXN; nr++) {  
        for (int m = 3; m*mid + r*nr <= MAXN; m++) {  
            sum[m*mid+r*nr] += (m-1)/2;  
        }  
    }  
}
```



# E – Enigmatic Partition

- 题解

- 拆分
- 可以

```
for (int l = 1; l <= MAXN; l++) {  
    int mid = l+1;  
    int r = l+2;  
    for (int nl = 1; nl * l <= MAXN; nl++) {  
        for (int m = 3; m*mid + l*nl <= MAXN; m++) {  
            sum[m*mid+l*nl] += (m-1)/2;  
        }  
    }  
    for (int nr = 0; nr*r <= MAXN; nr++) {  
        for (int m = 3; m*mid + r*nr <= MAXN; m++) {  
            sum[m*mid+r*nr] += (m-1)/2;  
        }  
    }  
}
```

!小的时候会比较慢，可以用DP去优化  $l \leq 100$  的时候。



## E – Enigmatic Partition

- 有很多比标程写得优秀的做法，可以自行学习。

```
int main() {
    int T, i, j, l, r, t;
    STB(i, 2, 100000/3) {
        STB(j, 3, (100000-2)/(i-1)) {
            ++a[(i-1)*j+3];
            --a[i*j+2];
            --a[i*j+1];
            ++a[(i+1)*j-3+3];
        }
    }
    STB(i, 2, 100000) a[i] += a[i-2];
    STB(i, 1, 100000) a[i] += a[i-1];
    STB(i, 1, 100000) a[i] += a[i-1];
    scanf("%d", &T);
    STB(t, 1, T) {
        scanf("%d%d", &l, &r);
        printf("Case #%d: %lld\n", t, a[r]-a[l-1]);
    }
    return 0;
}
```



## A – All-Star Game

- 题目大意

- 有n个篮球运动员，m个球迷。
- 一个球员可能是多个球迷的粉丝。

Basketball fan  $i$  is like to watch the game of player  $j$  if one of the following conditions is met:

- Basketball fan  $i$  is a fan of basketball player  $j$ .
  - There is a fan  $i'$  and a player  $j'$ , both fan  $i$  and fan  $i'$  are like to watch the game of player  $j'$ , and fan  $i'$  is like to watch the game of player  $j$ .
- 
- 选择最少的球员进全明星赛，使得所有球迷都愿意观看（至少一个球迷想看的球员入选）。
  - 有 $q$ 个粉丝关系的修改，修改完回答询问。 $1 \leq n, m, q \leq 2 \times 10^5$



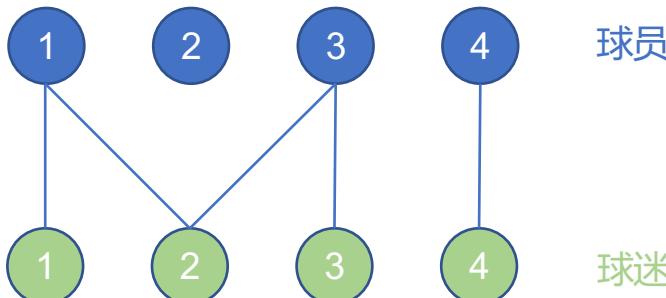
## A – All-Star Game

- 题解

- 如果把这个关系建成一个图的话。其实就是需要求n个球员的连通分量个数。

- 如果有球迷的度为0，答案就是-1。

- 否则答案就是 ( n个球员的连通分量个数 ) - (孤立球员个数)





# A – All-Star Game

- 题解

- 如果把这个关系建成一个图的话。其实就是要求n个球员的连通分量个数。  
如果有球迷的度为0，答案就是-1。  
否则答案就是 ( n个球员的连通分量个数 ) - ( 孤立球员个数 )
- 所以只需要在加边和删边的时候，维护n个球员的连通分量个数。  
 $x$ 是球员， $y$ 是球迷。  
 $x, y$ 加边时候，如果他们本来不连通，而且 $y$ 原来有边，连通分量减一。  
 $x, y$ 删边时候，如果删完他们变得不连通，而且 $y$ 还有边，连通分量加一。



# A – All-Star Game

- 题解

- 问题就变成了图中进行加边删边，查询两点连通性。

- 可以离线处理。

- 方法一：LCT

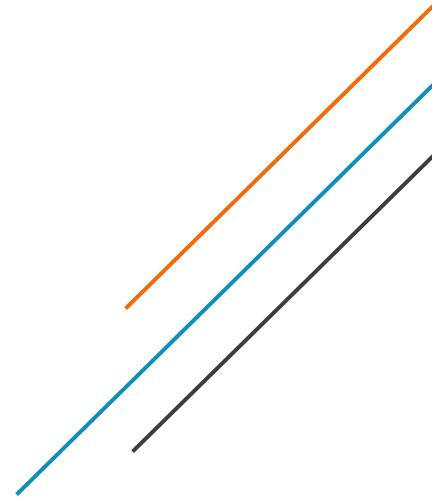
- 动态树维护一颗边删除最晚的生成树

- 加边时候，割掉时间早的边。

- 查询，判断链上删除时候和现在时间

- 方法二：线段树+可撤销并查集

- 边的存在时间作为区间，查询作为点。递归过程中并查集维护连通性。





## D – Disgusting Relationship

- 题意

有  $n$  个人，每个人都有唯一喜欢的人，且每个人喜欢的人都互不相同。我们把这样的一种关系称之为“恋情关系”。

如果有  $k$  个人的喜欢关系满足： $A_1$  喜欢  $A_2, A_2$  喜欢  $A_3, \dots, A_{k-1}$  喜欢  $A_k, A_k$  喜欢  $A_1$

那么我们就称这  $k$  个人构成一段“ $k$ 角恋”的关系。

假如一种恋情关系会造成  $a_1$  段“1角恋”， $a_2$  段“2角恋”， $\dots, a_n$  段“ $n$ 角恋”，那么我们就称这样的恋情关系是一种型为  $(a_1, a_2, \dots, a_n)$  的恋情关系。

求解：有多少种不同的型，满足“能构成这种型的恋情关系的种类数不是  $p$  的倍数”。



## D – Disgusting Relationship

首先， $n$ 个人的恋情关系其实就是一个 $n$ 元置换。而 $k$ 角恋关系其实就是一个 $k$ 阶轮换。

有 $a_1$ 个1阶轮换， $a_2$ 个2阶轮换，…， $a_n$ 个 $n$ 阶轮换的 $n$ 元置换（下记为：型为 $(a_1, a_2, \dots, a_n)$ 的 $n$ 元置换）的种类数是 $f(a_1, a_2, \dots, a_n) = \frac{n!}{1^{a_1} 2^{a_2} \dots n^{a_n} \cdot a_1! \cdot a_2! \cdots a_n!}$ ，并且显然我们有 $\sum_{i=1}^n i * a_i = n$ 。

（以上内容如果学过 polya 定理，应该非常熟悉，在此证明略过）

对于一个型为 $(a_1, a_2, \dots, a_n)$ 的 $n$ 元置换

$f(a_1, a_2, \dots, a_n)$ 中分子部分是固定的，因此我们只要考虑让分母中各部分对 $p$ 的贡献充分大。



## D – Disgusting Relationship

- 考虑分母中包含  $a_k$  的项： $k^{a_k} \cdot a_k!$

1. 如果  $k$  不是  $p$  的倍数并且不是 1, 且  $a_k \geq p$  , 那么它对  $p$  的贡献就是： $\sum_{\alpha \geq 1} \left\lfloor \frac{a_k}{p^\alpha} \right\rfloor$

而由  $\sum_{i=1}^n i * a_i = n$  可知，如果我们把  $a_k$  个  $k$  阶轮换换成  $k * a_k$  个 1 阶轮换，也将是一种可行的型。并且此时的贡献为  $\sum_{\alpha \geq 1} \left\lfloor \frac{k * a_k}{p^\alpha} \right\rfloor \geq \sum_{\alpha \geq 1} k * \left\lfloor \frac{a_k}{p^\alpha} \right\rfloor > \sum_{\alpha \geq 1} \left\lfloor \frac{a_k}{p^\alpha} \right\rfloor$ 。这说明 “如果  $k$  不是  $p$  的倍数并且不是 1, 且  $a_k$  对  $p$  有贡献，那么这样的贡献一定不是充分大的”。



## D – Disgusting Relationship

- 考虑分母中包含  $a_k$  的项 :  $k^{a_k} \cdot a_k!$

2. 如果  $k = t * p, t > 1$  , 那么它对p的贡献就是 :  $\sum_{\alpha \geq 1} \left\lfloor \frac{a_k}{p^\alpha} \right\rfloor + a_k * (1 + \operatorname{argmax}_i(p^i|t))$

由  $\sum_{i=1}^n i * a_i = n$  可知 , 如果我们把  $a_k$  个k阶轮换换成  $t * a_k$  个p阶轮换 , 也将是一种可行的型。

并且此时的贡献为  $\sum_{\alpha \geq 1} \left\lfloor \frac{t * a_k}{p^\alpha} \right\rfloor + t * a_k > \sum_{\alpha \geq 1} \left\lfloor \frac{a_k}{p^\alpha} \right\rfloor + a_k * (1 + \operatorname{argmax}_i(p^i|t))$  。

这说明 “如果k是p的倍数并且  $k \neq p$ , 且  $a_k$  对p有贡献 , 那么这样的贡献一定不是充分大的” 。



## D – Disgusting Relationship

- 考虑分母中包含  $a_k$  的项： $k^{a_k} \cdot a_k!$

经过我们之前的分析，我们得到了一个结论：当  $k$  不为 1 或  $p$  时，如果  $a_k$  会对  $p$  造成贡献，或者可以通过调整  $a_k$  的分配使某个  $a_k$  对  $p$  造成贡献，那么这样的贡献一定不是充分大的。

接下来我们考虑  $a_1$  和  $a_p$ ：

记  $n \% p = a_0$ ,  $n = a_0 + n_1 * p$

假设某种型有  $t$  个  $p$  阶轮换，很容易想到， $a_1$  不能小于  $n_1 * p - t * p$ ，否则贡献必然不是充分大。

并且此时分母处所有贡献来自  $a_1$  和  $a_p$ ，因此可以计算出分子和分母中  $p$  的个数应该相等：

$$\sum_{\alpha} \left\lfloor \frac{(n_1 - a_p) * p + \varepsilon}{p^{\alpha}} \right\rfloor + \sum_{\alpha} \left\lfloor \frac{a_p}{p^{\alpha}} \right\rfloor + a_p = \sum_{\alpha} \left\lfloor \frac{n_1 * p + a_0}{p^{\alpha}} \right\rfloor$$



## D – Disgusting Relationship

- 化简一下式子：

$$\sum_{\alpha} \left\lfloor \frac{(n_1 - a_p) * p + \varepsilon}{p^{\alpha}} \right\rfloor + \sum_{\alpha} \left\lfloor \frac{a_p}{p^{\alpha}} \right\rfloor + a_p = \sum_{\alpha} \left\lfloor \frac{n_1 * p + a_0}{p^{\alpha}} \right\rfloor$$

$$\sum_{\alpha} \left\lfloor \frac{(n_1 - a_p)}{p^{\alpha}} \right\rfloor + \sum_{\alpha} \left\lfloor \frac{a_p}{p^{\alpha}} \right\rfloor + (n_1 - a_p) + a_p = \sum_{\alpha} \left\lfloor \frac{n_1}{p^{\alpha}} \right\rfloor + n_1$$

$$\sum_{\alpha} \left\lfloor \frac{(n_1 - a_p)}{p^{\alpha}} \right\rfloor + \sum_{\alpha} \left\lfloor \frac{a_p}{p^{\alpha}} \right\rfloor = \sum_{\alpha} \left\lfloor \frac{n_1}{p^{\alpha}} \right\rfloor$$

这即是在说： $C_{n_1}^{a_p}$  中不含  $p$  这个因子。

于是我们得到了最后的结论：型为  $(a_1, a_2, \dots, a_n)$  的  $n$  元置换种类数不是  $p$  的倍数，当且仅当：

$a_1 \geq (n_1 - a_p) * p$  并且  $p \nmid C_{n_1}^{a_p}$ ，其中  $n \% p = a_0$ ,  $n = a_0 + n_1 * p$



## D – Disgusting Relationship

型为 $(a_1, a_2, \dots, a_n)$ 的n元置换种类数不是p的倍数，当且仅当： $a_1 \geq (n_1 - a_p) * p$  并且  $p \nmid C_{n_1}^{a_p}$ ，

其中  $n \% p = a_0$ ,  $n = a_0 + n_1 * p$

$a_1 - (n_1 - a_p) * p$  的部分，以及  $a_i (i \neq 1, p)$  的部分，显然是可以随意分配的，因此这部分的方案数是  $\rho(a_0)$ ，即  $a_0$  的整数分拆方案数。

$p \nmid C_{n_1}^{a_p}$  的部分的方案数，由卢卡斯定理容易推导出：记  $n_1$  在p进制下表示为  $(N_0 N_1 \dots)_p$ ，那么恰有  $\prod_{i \geq 0} (N_i + 1)$  种这样的组合数。

整数分拆方案数可以通过五边形数定理  $O(n^{1.5})$  预处理，分解p进制复杂度为  $O(\log(n))$ 。

因此总复杂度为  $O(p^{1.5} + T * \log_p(n))$ 。

（事实上，这个结果还有一种 high level 但也更具普适性的证明方法，需要较深一些的近世代数+高等代数的基础，在此不做赘述，想了解的小伙伴可以赛后找我要资料XD）



## C – Cinema

- 题目大意

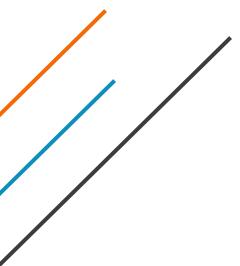
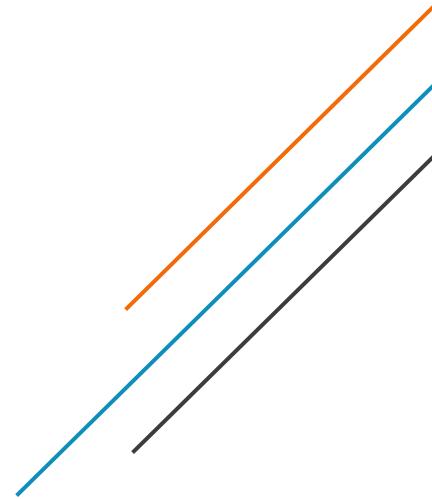
- $n*m$ 的电影院座位。两个人不能前后左右相邻地做。问最坏情况下，坐多少人就坐满了。
- 输出方案。 $1 \leq n \leq 1000, 1 \leq m \leq 15, T \leq 1000$



## C – Cinema

- 题解

- 其实就在 $n*m$ 的网格中，用最少的十字覆盖掉所有格子。
- 贪心构造？





# C – Cinema

- 题解

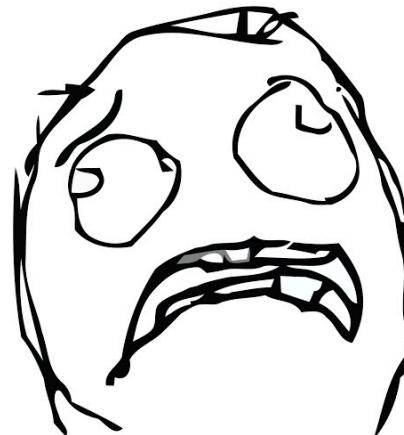
- 状压DP.
- 三进制表示行的状态：0表示未覆盖，1表示被覆盖了的，2表示放了十字的。



# C – Cinema

- 题解

- 状压DP.
- 三进制表示行的状态：0表示未覆盖，1表示被覆盖了的，2表示放了十字的。
- 状态数： $3^{15} = 14,348,907$  ?





## C – Cinema

- 题解

- 状压DP.
- 三进制表示行的状态：0表示未覆盖，1表示被覆盖了的，2表示放了十字的。
- 去除非法状态：

两个0相邻，两个2相邻，0和2相邻，两个相邻的1边上没有2

$m = 15$ , 状态数~6531,  $m = 14$ 状态数~3721,  $m = 10$ 状态数~392



- 题解

- 状压DP.
- 三进制表示行的状态：0表示未覆盖，1表示被覆盖了的，2表示放了十字的。
- 去除非法状态：
  - 两个0相邻，两个2相邻，0和2相邻，两个相邻的1边上没有2
- $m = 15$ , 状态数~6531,  $m = 14$ 状态数~3721,  $m = 10$ 状态数~392
- 状态转移：
  - 0的下一行一定要2，2的下一行是1，



## C – Cinema

- 题解

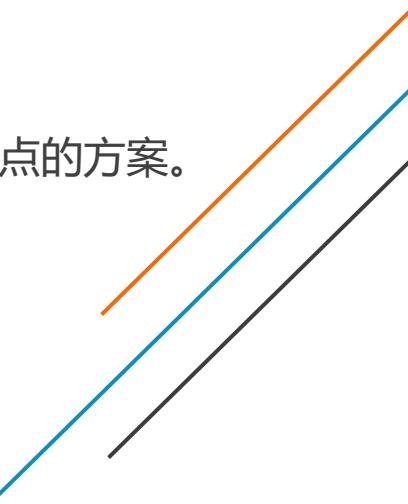
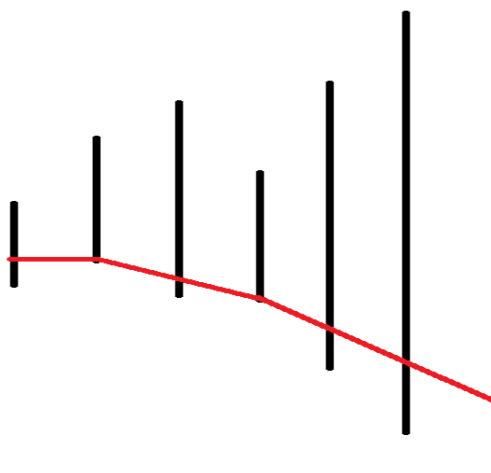
- 因为T最大有1000，可以离线所有test case, 然后按照m相同的一起dp求解。
- 记录状态转移，输出方案。



## J – Jumping Points

- 题目大意：

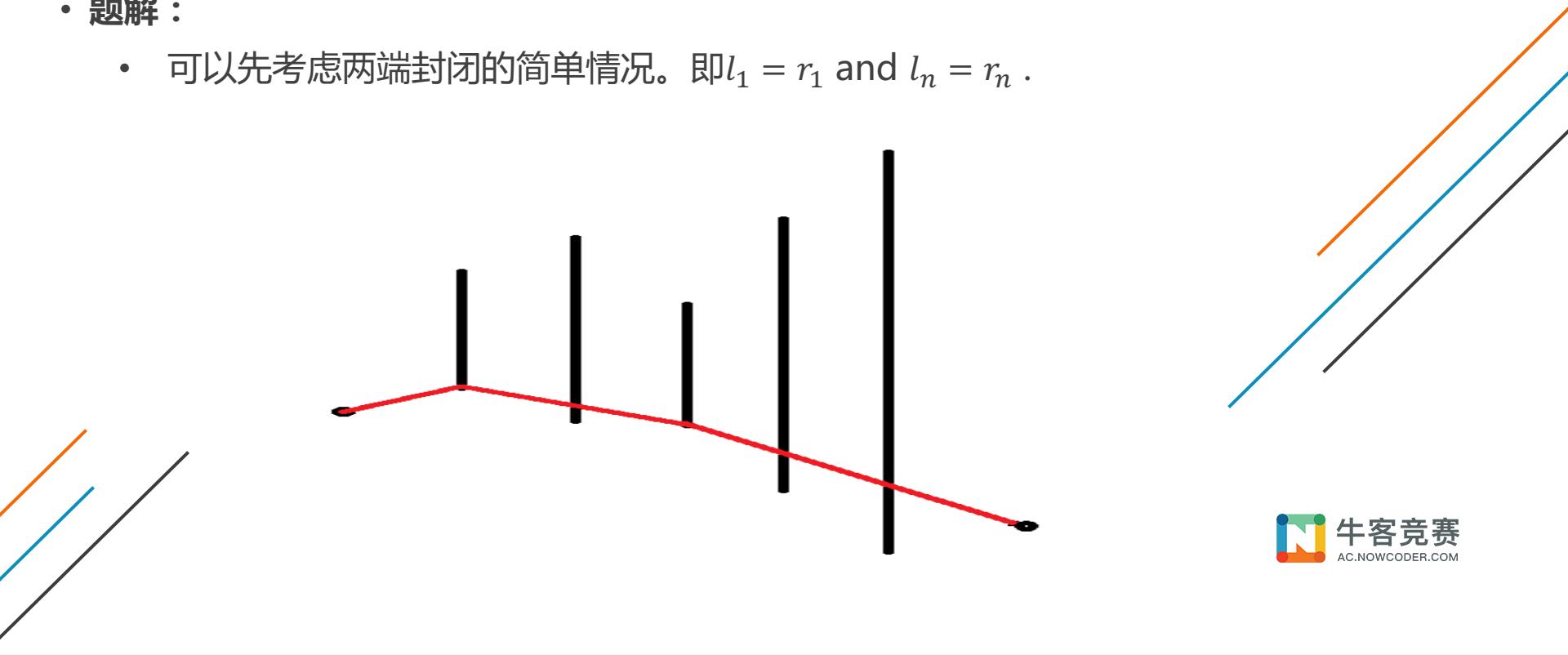
- 给了 $n$ 条线段： $(i, l_i) \sim (i, r_i)$ .
- 要在每个线段上选择一个点，使得这 $n$ 个点组成的折线最短。输出一种点的方案。
- $2 \leq n \leq 10^5$ .





## J – Jumping Points

- 题解：
  - 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .

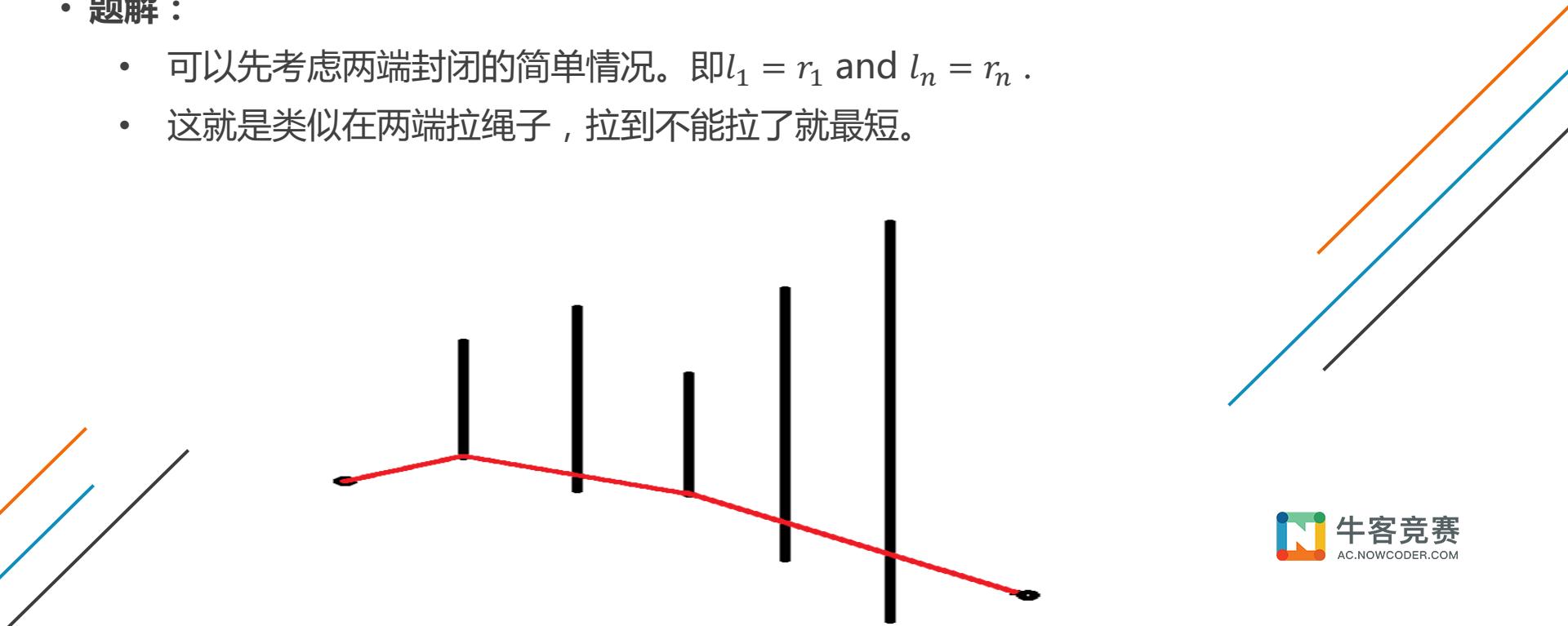




## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .
- 这就是类似在两端拉绳子，拉到不能拉了就最短。

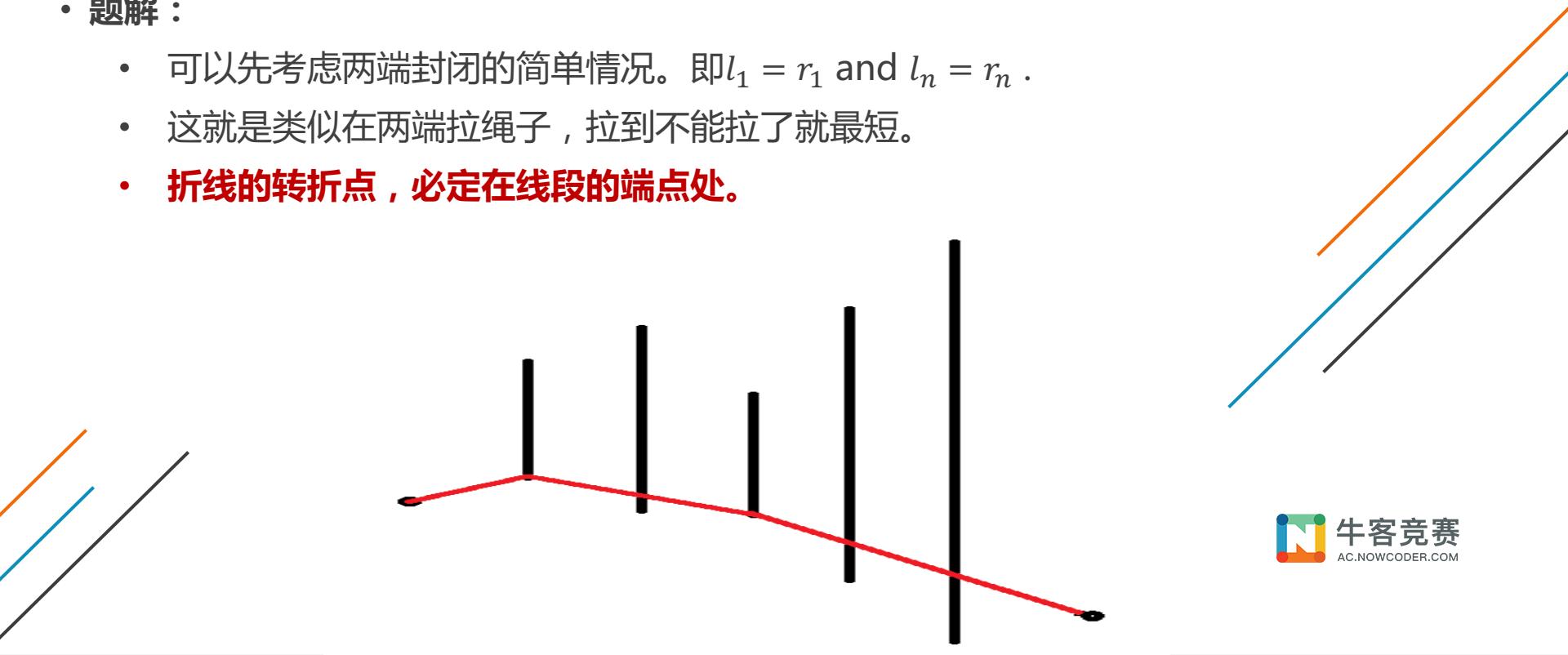




## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .
- 这就是类似在两端拉绳子，拉到不能拉了就最短。
- **折线的转折点，必定在线段的端点处。**

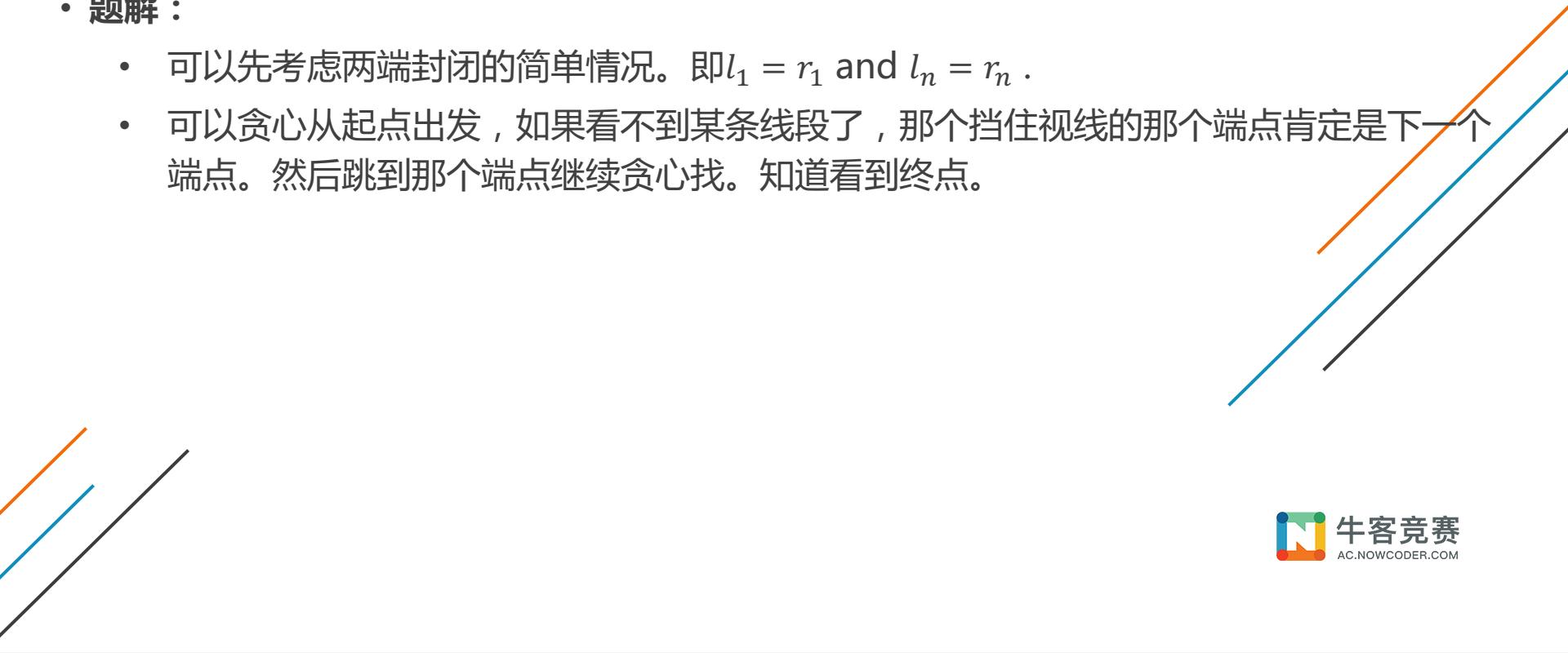




## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .
- 可以贪心从起点出发，如果看不到某条线段了，那个挡住视线的那个端点肯定是下一个端点。然后跳到那个端点继续贪心找。知道看到终点。

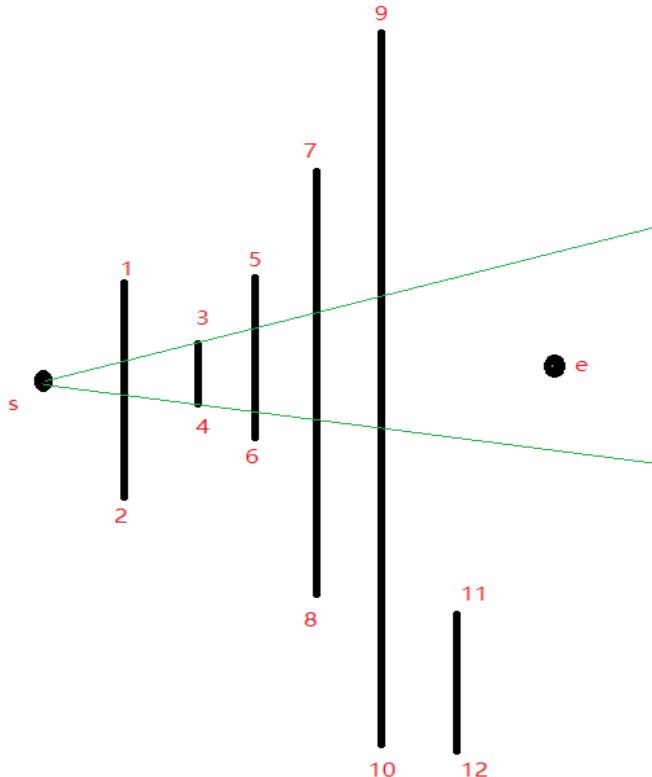




# J – Jumping Points

- 题解：

- 可以先考虑两
- 可以贪心从起
- 端点。然后跳



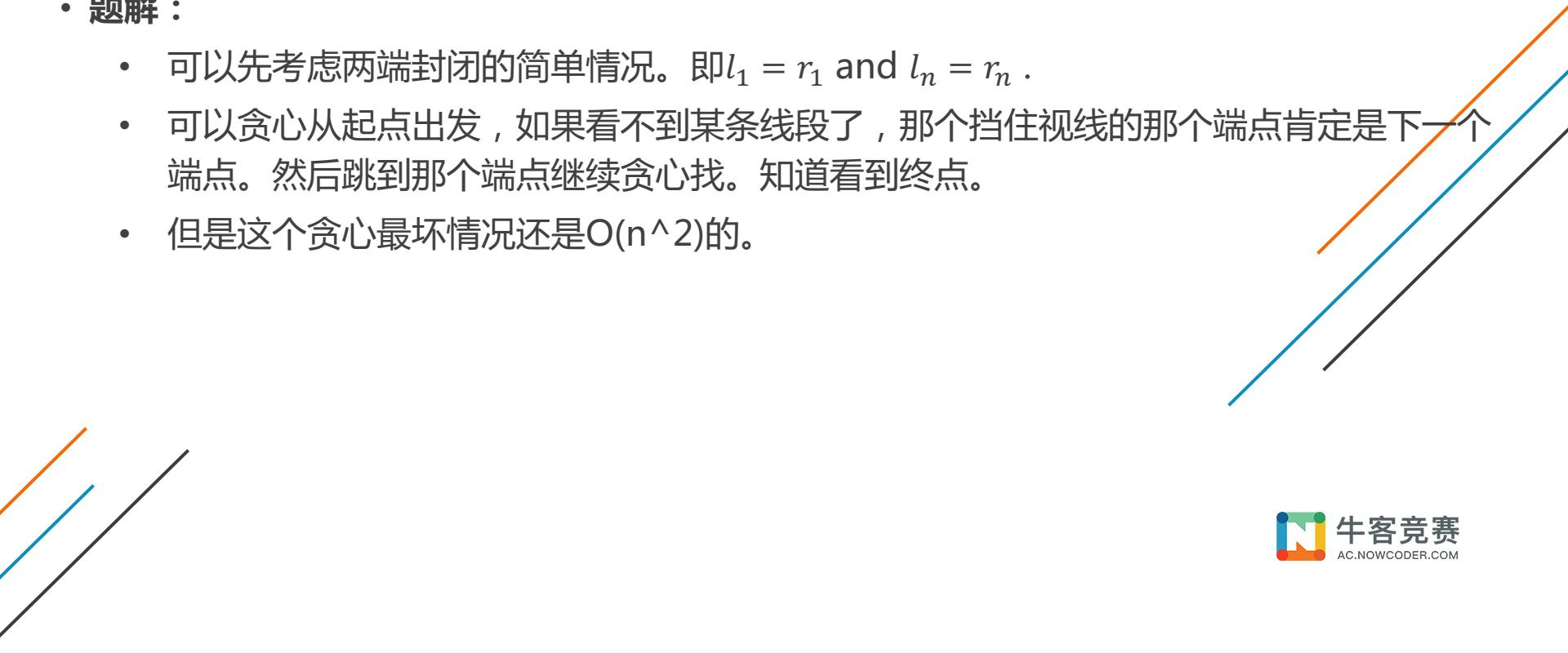
那个端点肯定是下一个



## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .
- 可以贪心从起点出发，如果看不到某条线段了，那个挡住视线的那个端点肯定是下一个端点。然后跳到那个端点继续贪心找。知道看到终点。
- 但是这个贪心最坏情况还是 $O(n^2)$ 的。

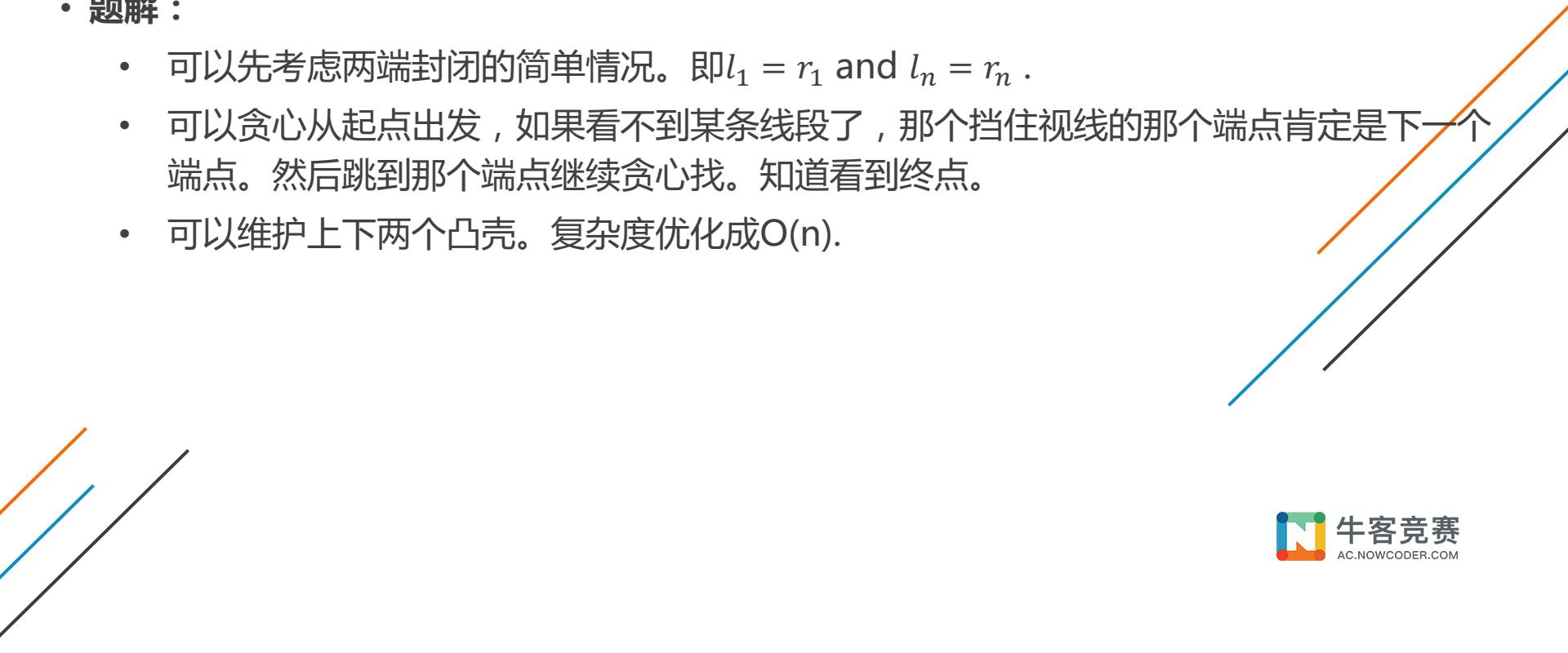




## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。即 $l_1 = r_1$  and  $l_n = r_n$  .
- 可以贪心从起点出发，如果看不到某条线段了，那个挡住视线的那个端点肯定是下一个端点。然后跳到那个端点继续贪心找。知道看到终点。
- 可以维护上下两个凸壳。复杂度优化成 $O(n)$ .

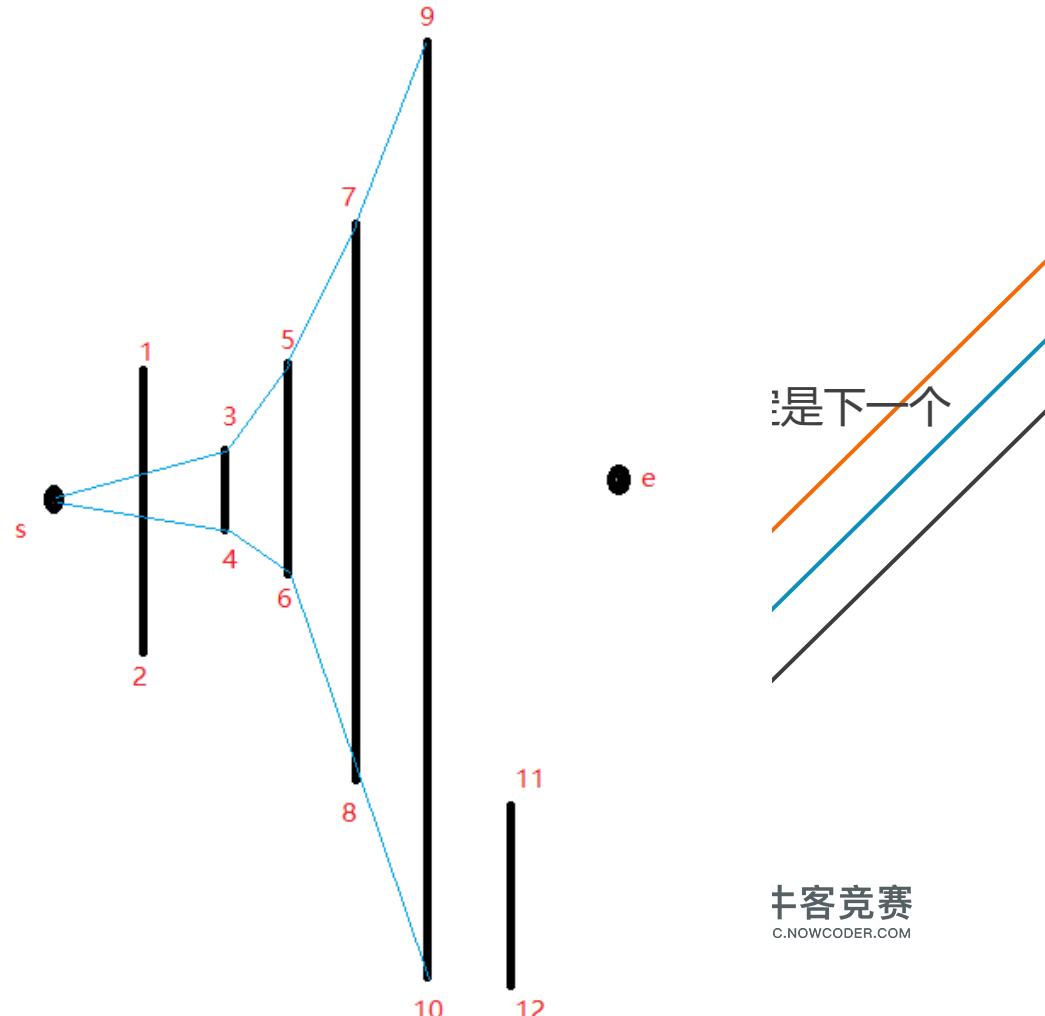




## J – Jumping Points

- 题解：

- 可以先考虑两端封闭的简单情况。
- 可以贪心从起点出发，如果看不到端点。然后跳到那个端点继续贪心。
- 可以维护上下两个凸壳。复杂度优

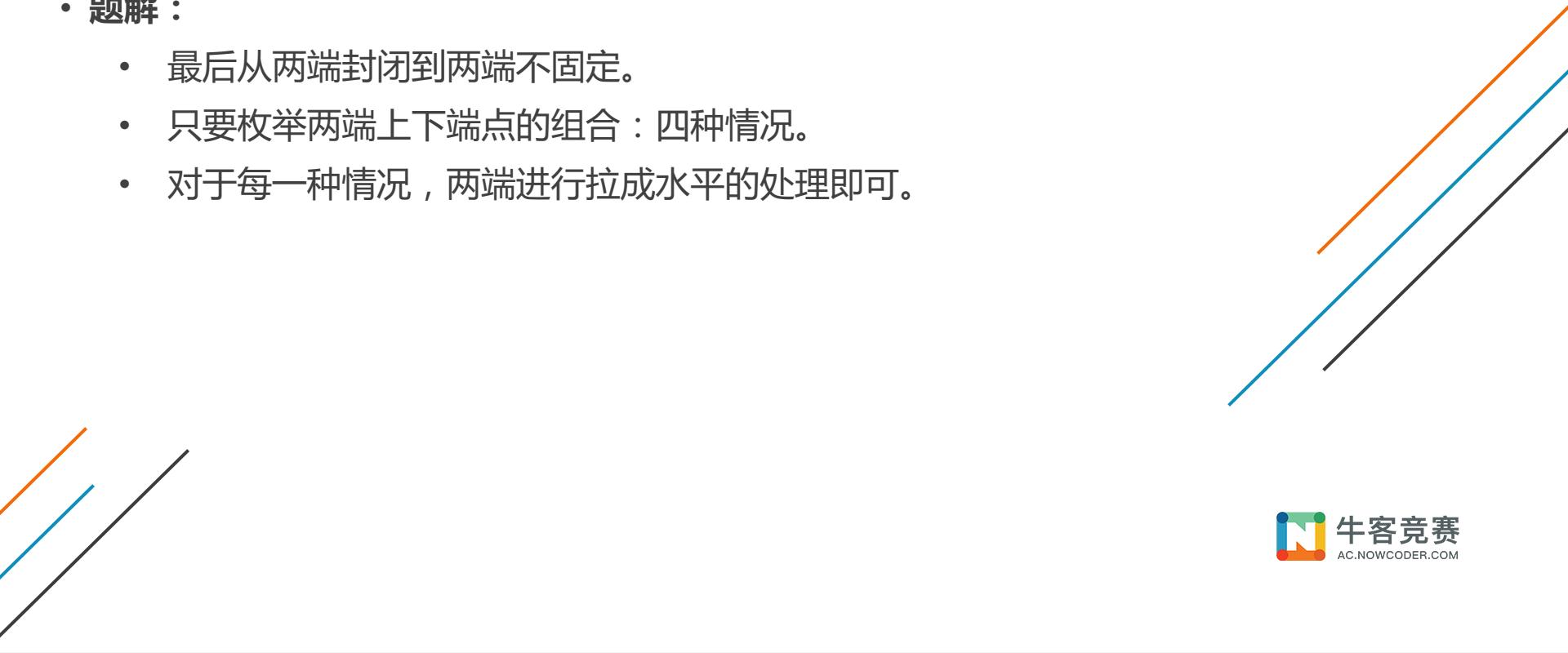




## J – Jumping Points

- 题解：

- 最后从两端封闭到两端不固定。
- 只要枚举两端上下端点的组合：四种情况。
- 对于每一种情况，两端进行拉成水平的处理即可。





## B - Bon Voyage

- 题意：

给一棵 $n$ 个点的树，树上每个点有权值。对于一个距离系数 $k$ ，我们定义在这个距离系数下每个点的  $l_i l_i$  值为以  $i$  为结尾的到根路径上长度为  $k$  的区间里所有  $l_i l_i$  序列的最大长度。然后给出一个目标值  $W$ ，你需要回答对于  $k=1 \dots n$  的每个状态下， $l_i l_i$  值大于等于  $W$  的节点有多少个。



## B - Bon Voyage

- 题解：

我们先简化问题，考虑序列上的问题，并且不受K限制。

我们考虑dp，对于给定的序列  $v[1], v[2], v[3] \dots v[n]$ ，我们定义  $dp[i]$  表示以  $i$  号点为开头的最长 lili 序列的长度。然后我们去考虑如何维护这个  $dp$  值。对于我们枚举到的  $i$  位置，我们去观察这个点可能贡献到哪些 lili 序列。对于数列 {1, 5, 2, 3, 4} 我们考虑第 5 个位置的值 4 能贡献到哪几个位置。显然它能对以 3 位置的 2 开头的 lili 序列，以 4 位置的 3 开头的 lili 序列，以 5 位置 4 开头的 lili 序列有贡献。那么为什么不能对前两个有贡献呢？因为对于第一个 1 来说，4 不是 1 后面第一个比它大的。对于第二个 5 来说，4 比 5 小。我们假设当前位置为  $x$ ，值为  $v[x]$ 。观察它贡献的位置，我们很容易发现其有贡献的序列就是那些开头介于  $last$  与  $x$  之间的。 $last$  为  $x$  之前的离  $x$  最近的值大于等于  $v[x]$  的位置。



## B - Bon Voyage

- 题解：

进而我们可以发现，当前一个点的值可以对一段区间产生贡献，并且贡献的形式都是区间加一，这启发我们用数据结构去维护这个转移。我们使用线段树，线段树的维护以每个位置开头的dp最大值，按从左往右的顺序枚举  $i$  点，对于每个  $i$  我们可以用单调栈找出其贡献的区间，然后在线段树上实现区间加一，最后去询问线段树上的前缀  $[1, i]$  最大值，这个值就是每个点的dp值。

那么同样，我们发现对于有  $K$  限制的版本， $i$  点的dp值就为线段树上询问区间  $[\max(i-k+1, 1), i]$  的区间最大值，其它并无区别。

此时，对于固定的  $K$ ，我们已经得到了一个效率为  $O(n \log n)$  的做法。但这题要求的是对于每个  $K$  都输出其对应的答案，这个效率显然是不够的，所以我们应该去思考如何优化这个算法。



## B - Bon Voyage

- 题解：

我们再回过头来看问题，因为其要询问的是对于每个  $K$ ， $l_i l_i$  值大于  $W$  的点数。而上述步骤中我们求出了所有点的  $l_i l_i$  值，这显然浪费了非常多的额外时间。于是我们观察，对于给定的  $W$ ，若在距离系数为  $k$  的情况下，其  $l_i l_i$  值满足了条件即大于等于  $W$ ，那么在距离系数为  $k+1$  的情况下也一定满足条件。于是我们就可以想到二分，如果对于每个点，我能求得使其满足条件的最小的  $k$  值，那么对于  $K \in [k, n]$ ，它也应该满足条件。

那么对于每个点我们可以二分使它满足条件的最小的  $k$ ，对于二分的一个值  $mid$ ，去线段树上询问区间  $[max(i-mid+1, 1), i]$  的最大值，然后根据这个值与  $W$  的关系继续二分。但是这个效率对于每个点是  $O(\log^2 n)$ ，因为有经典的做法，所以我们卡掉了这个效率。我们知道，线段树的本质其实就是在一直二分区间，那么我们就可以将二分  $k$  的这个过程和线段树结合起来，换句话说就是在线段树上做二分来找到这个最小的  $k$ ，这个效率对于每个点是  $O(\log n)$ 。



## B - Bon Voyage

- 题解：

关于线段树上二分，我们可以考虑当进入到一个节点的时候，若其右儿子维护的区间max大于等于W，那说明右儿子中一定存在满足条件的k值，且这个k值一定比左儿子中的k值来的小。如若不然，则观察其左儿子的max值，若其大于等于W，则进左儿子找答案，若左右儿子的值都小于W，则该点无论如何都不可能对答案有贡献，此时直接在线段树上直接return即可。

最后我们来考虑树上的版本与序列上的差异。序列上的从左往右按顺序就变成了树上的按dfs序，序列上的贡献区间的左端点从x前面离它最近的一个权值大于等于 $v[x]$ 的点变成了x到根路径上第一个权值大于等于 $v[x]$ 的点。 $x$ 到根路径上第一个权值大于等于 $v[x]$ 的点如何求呢？我们可以采用可撤销单调栈或者倍增，题解采用了可撤销单调栈，就是在单调栈弹栈的时候不真正的弹掉元素，只是二分一个弹完之后的栈顶位置，最后就可以直接撤销了。



## B - Bon Voyage

- 题解：

接下来我们需要考虑一下线段树上维护的信息发生了什么变化。在序列问题上，线段树很自然的可以维护每个下标位置的dp值，因为这些都是连续的。但是在树上，每个点到根路径上的点的dfs序不是连续的，所以我们不能去维护dfs序，但是我们可以发现，每个点到根路径上的点的高度是连续的！所以我们可以将线段树换成维护高度为i的点的dp值，因为在dfs的过程中，每个时刻在lili序列中高度与点是一一对应的。那么我们只需按上述方法维护答案，然后在dfs过程中进入一个点的时候加上这个点的贡献，然后计算答案，在撤出这个点的时候减去其在线段树上的贡献，并撤回单调栈即可。

此时我们已经求出了每个点可以贡献的最小k值，然后我们只需要对这些值做一遍前缀和就能得到答案了。

效率： $O(n \log n)$



## Problem F Factorio

- 题解

首先，将每种物品建一个点，将原料向产品连一条有向边，容易将题目转化为一个DAG图，其中边权为合成公式中原料的系数，点权为合成公式中产品的系数的倒数。

设 $F(i)$  = 从根到*i*所有路径长度之和。其中路径长度是指：路径上的点和边的权值的乘积。

$F(i)$ 的实际意义为在理想情况下，*i*节点与根节点的机器数量之比。

因此，题目所求的瓶颈就是 $\text{cnt}(i)/F(i)$ 最小的那些机器，其中 $F$ 需要使用高精度，或者注意到点权和边权只有1，2，1/2，因此也可以使用bitset维护超长整形。



## Problem H Hard String Problem

- 题意

有n个无限循环的字符串，给出每个循环串的循环节，求这n个串有多少个本质不同公共子串。

- 题解

首先需要将每个串处理为最简形式，用其最小循环节的最小循环同构来表示。

对于只有两个无限循环串，求公共子串的问题，我们可以得到一个结论：如果其本质循环节相同，则拥有无限多个公共子串。否则，公共子串的长度将不会超过长串长度的三倍。



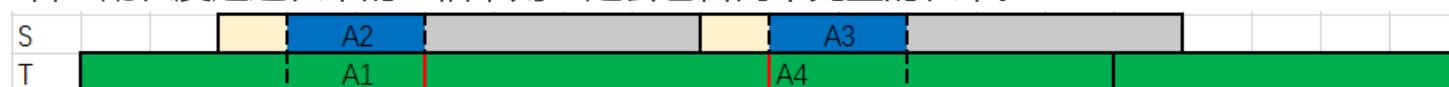
## 引理：公共子串的长度不超过长串长度的三倍

- 证明：

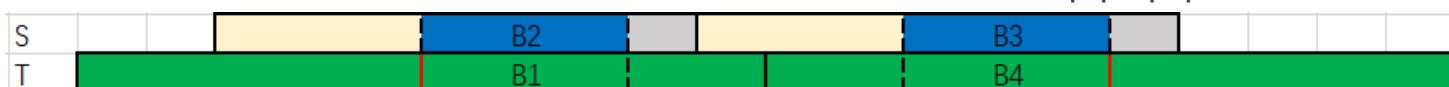
设两个无限循环串  $S, T$ 。其中  $S$  为长串， $T$  为短串。并且都已经是简形式。



若匹配长度超过长串的三倍，则一定会包含两个完整的长串。



容易发现有： $A_1 = A_2 = A_3 = A_4$ ，所以  $A$  是  $T$  的一个 border，进而  $|T| - |A|$  是  $T$  的一个周期。



同理， $B_1 = B_2 = B_3 = B_4$ ，所以  $B$  也是  $T$  的 border，进而  $|T| - |B|$  也是  $T$  的周期。注意这里的  $|B| + |A| = |T|$ 。

由弱周期定理， $C = \text{GCD}(|A|, |B|)$  一定也是  $T$  的周期。注意到  $C$  能整除  $T$ ，于是  $C$  确定为  $T$  的一个循环节。

因此  $T$  存在更短的循环节，与最简形式矛盾。



## Problem H Hard String Problem

- 题解

有了引理之后，我们思考如何将其推广到n个串，我们可以选出最短串 $S_{min}$ ，求剩余的 $n-1$ 个串 $S_i$ 各自和 $S_{min}$ 的公共子串，然后将这些子串集合求交。

在求 $S_{min}$ 与 $S_i$ 的公共子串数量时，需要将 $S_{min}$ 以及 $S_i$ 都各自翻倍到 $4|S_i|$ 长度。

因此考虑整个过程， $S_{min}$ 最长需要翻倍到 $4|S_{max}|$ ，其余每个串则需要翻倍到 $4|S_i|$ ，然后可以用广义SAM来求这些翻倍过后串的公共子串即可。

# Thanks

