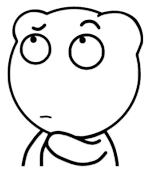
# 2019 Summer Newbie Selection Contest 5

## Presentation of solutions

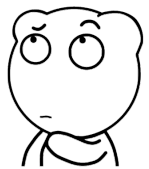
**打完这场就脱单队**

August 22, 2019



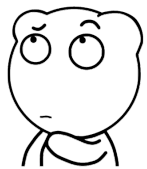
# Difficulty

- Very Easy: I
- Easy: A,C
- Medium: B,D
- Medium Hard:H,F
- Hard:E



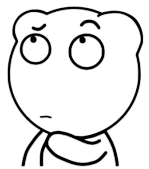
# Statistics

| | Solved | Attempted | First Solved |
|---|---|---|---|
| A | 21 | 64 | 00:24 by 王如嫣 |
| B | 3 | 10 | 01:42 by 葛鹏飞 |
| C | 7 | 41 | 01:55 by 葛鹏飞 |
| D | 1 | 33 | 03:18 by 陈浚毅 |
| E | 0 | 3 | |
| F | 2 | 3 | 03:06 by 黄铖 |
| G | 0 | 18 | |
| H | 2 | 11 | 03:41 by 黄铖 |
| I | 26 | 43 | 00:06 by 胡雅琪 |



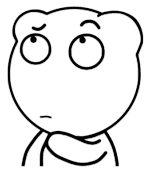
# Problem I. Ei

- 把 Ei 替换 8，把 8 替换成 Ei
- 无须改动原来的字符串，遇到需要改的部分直接输出就好
- 复杂度 $O(n)$



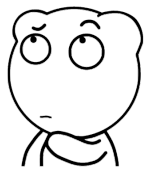
# Problem I. Ei

- 把 Ei 替换 8，把 8 替换成 Ei
- 无须改动原来的字符串，遇到需要改的部分直接输出就好
- 复杂度 $O(n)$



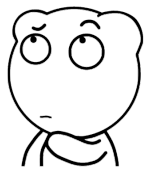
# Problem I. Ei

- 把 Ei 替换 8，把 8 替换成 Ei
- 无须改动原来的字符串，遇到需要改的部分直接输出就好
- 复杂度 $O(n)$



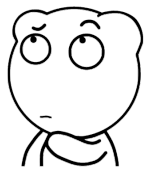
# Problem A. 远哥迟到的七夕礼物

- **求最小生成树即可**
- 但是不能用 Prim 或者 Kruskal 直接求，因为 $n$ 的范围太大了，存不下
- 我们先手玩一下，考虑一下用逻辑与的情况，发现好像能玩出来点东西
- 对于所有的编号为偶数的点，直接和 1 号点连边，那么边权直接就为 0 了
- 对于奇数点，除了二进制位全为 1 且等于 n 的情况，其余数一定能找到一个偶数点和它连边，边权为 0



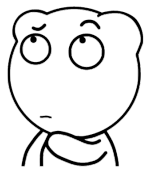
# Problem A. 远哥迟到的七夕礼物

- 求最小生成树即可
- 但是不能用 Prim 或者 Kruskal 直接求，因为 $n$ 的范围太大了，存不下
- 我们先手玩一下，考虑一下用逻辑与的情况，发现好像能玩出来点东西
- 对于所有的编号为偶数的点，直接和 1 号点连边，那么边权直接就为 0 了
- 对于奇数点，除了二进制位全为 1 且等于 n 的情况，其余数一定能找到一个偶数点和它连边，边权为 0



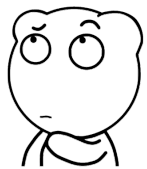
# Problem A. 远哥迟到的七夕礼物

- 求最小生成树即可
- 但是不能用 Prim 或者 Kruskal 直接求，因为 $n$ 的范围太大了，存不下
- 我们先手玩一下，考虑一下用逻辑与的情况，发现好像能玩出来点东西
- 对于所有的编号为偶数的点，直接和 1 号点连边，那么边权直接就为 0 了
- 对于奇数点，除了二进制位全为 1 且等于 n 的情况，其余数一定能找到一个偶数点和它连边，边权为 0



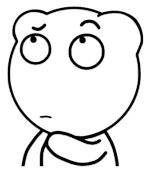
# Problem A. 远哥迟到的七夕礼物

- 求最小生成树即可
- 但是不能用 Prim 或者 Kruskal 直接求，因为 $n$ 的范围太大了，存不下
- 我们先手玩一下，考虑一下用逻辑与的情况，发现好像能玩出来点东西
- 对于所有的编号为偶数的点，直接和 1 号点连边，那么边权直接就为 0 了
- 对于奇数点，除了二进制位全为 1 且等于 n 的情况，其余数一定能找到一个偶数点和它连边，边权为 0



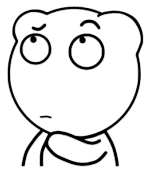
# Problem A. 远哥迟到的七夕礼物

- 求最小生成树即可
- 但是不能用 Prim 或者 Kruskal 直接求，因为 $n$ 的范围太大了，存不下
- 我们先手玩一下，考虑一下用逻辑与的情况，发现好像能玩出来点东西
- 对于所有的编号为偶数的点，直接和 1 号点连边，那么边权直接就为 0 了
- 对于奇数点，除了二进制位全为 1 且等于 n 的情况，其余数一定能找到一个偶数点和它连边，边权为 0



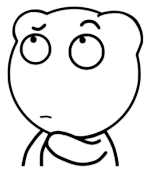
# Problem A. 远哥迟到的七夕礼物

- 对于二进制位全为 1 且等于 n 的这个点，可以找到一个点和它连边使得边权为 1
- 所以如果只用逻辑与连边的话，答案最大是 1，用逻辑异或一定大于等于它
- 所以答案是当 n 的二进制位全为 1 时，输出 1，其余输出 0
- 也可以小数据写个 Prim 或者 Kruskal，找个规律
- 复杂度 $O(\log n)$



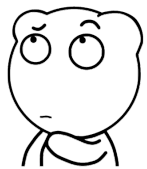
# Problem A. 远哥迟到的七夕礼物

- 对于二进制位全为 1 且等于 n 的这个点，可以找到一个点和它连边使得边权为 1
- 所以如果只用逻辑与连边的话，答案最大是 1，用逻辑异或一定大于等于它
- 所以答案是当 n 的二进制位全为 1 时，输出 1，其余输出 0
- 也可以小数据写个 Prim 或者 Kruskal，找个规律
- 复杂度 $O(\log n)$



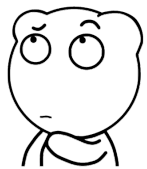
# Problem A. 远哥迟到的七夕礼物

- 对于二进制位全为 1 且等于 n 的这个点，可以找到一个点和它连边使得边权为 1
- 所以如果只用逻辑与连边的话，答案最大是 1，用逻辑异或一定大于等于它
- 所以答案是当 n 的二进制位全为 1 时，输出 1，其余输出 0
- 也可以小数据写个 Prim 或者 Kruskal，找个规律
- 复杂度 $O(\log n)$



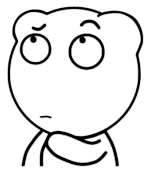
# Problem A. 远哥迟到的七夕礼物

- 对于二进制位全为 1 且等于 n 的这个点，可以找到一个点和它连边使得边权为 1
- 所以如果只用逻辑与连边的话，答案最大是 1，用逻辑异或一定大于等于它
- 所以答案是当 n 的二进制位全为 1 时，输出 1，其余输出 0
- 也可以小数据写个 Prim 或者 Kruskal，找个规律
- 复杂度 $O(\log n)$



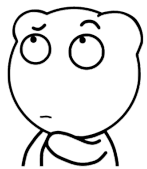
# Problem A. 远哥迟到的七夕礼物

- 对于二进制位全为 1 且等于 n 的这个点，可以找到一个点和它连边使得边权为 1
- 所以如果只用逻辑与连边的话，答案最大是 1，用逻辑异或一定大于等于它
- 所以答案是当 n 的二进制位全为 1 时，输出 1，其余输出 0
- 也可以小数据写个 Prim 或者 Kruskal，找个规律
- 复杂度 $O(\log n)$



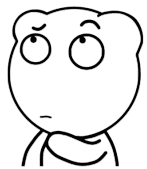
# Problem C. Petrichor_x 爱打牌

- 由于 $n$ 比较小，故我们可以考虑采取暴力枚举的方式。
- 对于每一张卡牌，我们只有选和不选两种选择，故我们可以扫一遍 $a_1 ... a_n$，用回溯法枚举。
- 我们可以通过一个 $std :: vector$ 来表示已经选择了哪些卡牌。
- 每次枚举到新的卡牌时，我们可以判断该卡牌加入后是否合法，如果合法我们可以将其 $push\_back()$，在回溯的过程中 $pop\_back()$。
- 复杂度 $O(2^n)$



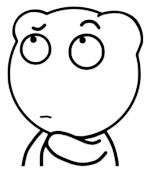
# Problem C. Petrichor_x 爱打牌

- 由于 $n$ 比较小，故我们可以考虑采取暴力枚举的方式。
- 对于每一张卡牌，我们只有选和不选两种选择，故我们可以扫一遍 $a_1 ... a_n$，用回溯法枚举。
- 我们可以通过一个 $std :: vector$ 来表示已经选择了哪些卡牌。
- 每次枚举到新的卡牌时，我们可以判断该卡牌加入后是否合法，如果合法我们可以将其 $push\_back()$，在回溯的过程中 $pop\_back()$。
- 复杂度 $O(2^n)$



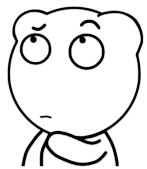
# Problem C. Petrichor_x 爱打牌

- 由于 $n$ 比较小，故我们可以考虑采取暴力枚举的方式。
- 对于每一张卡牌，我们只有选和不选两种选择，故我们可以扫一遍 $a_1 ... a_n$，用回溯法枚举。
- 我们可以通过一个 $std :: vector$ 来表示已经选择了哪些卡牌。
- 每次枚举到新的卡牌时，我们可以判断该卡牌加入后是否合法，如果合法我们可以将其 $push\_back()$，在回溯的过程中 $pop\_back()$。
- 复杂度 $O(2^n)$



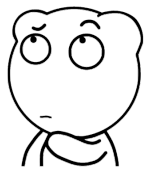
# Problem C. Petrichor_x 爱打牌

- 由于 $n$ 比较小，故我们可以考虑采取暴力枚举的方式。
- 对于每一张卡牌，我们只有选和不选两种选择，故我们可以扫一遍 $a_1 ... a_n$，用回溯法枚举。
- 我们可以通过一个 $std :: vector$ 来表示已经选择了哪些卡牌。
- 每次枚举到新的卡牌时，我们可以判断该卡牌加入后是否合法，如果合法我们可以将其 $push\_back()$，在回溯的过程中 $pop\_back()$。
- 复杂度 $O(2^n)$



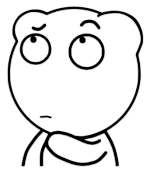
# Problem C. Petrichor_x 爱打牌

- 由于 $n$ 比较小，故我们可以考虑采取暴力枚举的方式。
- 对于每一张卡牌，我们只有选和不选两种选择，故我们可以扫一遍 $a_1 ... a_n$，用回溯法枚举。
- 我们可以通过一个 $std :: vector$ 来表示已经选择了哪些卡牌。
- 每次枚举到新的卡牌时，我们可以判断该卡牌加入后是否合法，如果合法我们可以将其 $push\_back()$，在回溯的过程中 $pop\_back()$。
- 复杂度 $O(2^n)$



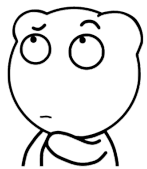
# Problem B. 罗老师修车

- 若要使某个区间内任何三个数都能组成一个三角形，只要区间内最小的两个数和最大的一个数能够组成一个三角形即可。
- 使用线段树维护区间内的最大值，最小值，次小值。
- 对于最小值和次小值，两端区间合并的时候可以把两个最小值和两个次小值放进一个数组里进行从小到大排序，新的区间取前两个即可。
- 复杂度 $O(n \log n + q \log n)$



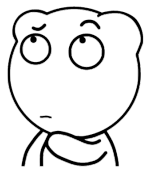
# Problem B. 罗老师修车

- 若要使某个区间内任何三个数都能组成一个三角形，只要区间内最小的两个数和最大的一个数能够组成一个三角形即可。
- 使用线段树维护区间内的最大值，最小值，次小值。
- 对于最小值和次小值，两端区间合并的时候可以把两个最小值和两个次小值放进一个数组里进行从小到大排序，新的区间取前两个即可。
- 复杂度 $O(n \log n + q \log n)$



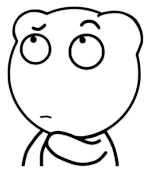
# Problem B. 罗老师修车

- 若要使某个区间内任何三个数都能组成一个三角形，只要区间内最小的两个数和最大的一个数能够组成一个三角形即可。
- 使用线段树维护区间内的最大值，最小值，次小值。
- 对于最小值和次小值，两端区间合并的时候可以把两个最小值和两个次小值放进一个数组里进行从小到大排序，新的区间取前两个即可。
- 复杂度 $O(n \log n + q \log n)$



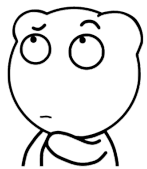
# Problem B. 罗老师修车

- 若要使某个区间内任何三个数都能组成一个三角形，只要区间内最小的两个数和最大的一个数能够组成一个三角形即可。
- 使用线段树维护区间内的最大值，最小值，次小值。
- 对于最小值和次小值，两端区间合并的时候可以把两个最小值和两个次小值放进一个数组里进行从小到大排序，新的区间取前两个即可。
- 复杂度 $O(n \log n + q \log n)$



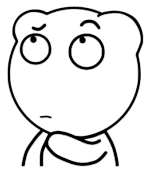
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



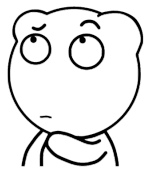
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



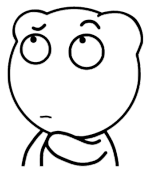
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



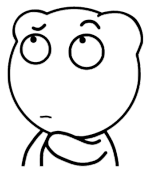
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



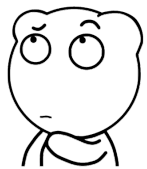
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



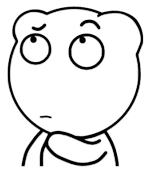
# Problem D. 996

- 对于每一个不为 0 的数 $a_i$ 记下标为 $i$，记录这个数前一次出现的位置 $pre[a_i]$，查询这段区间内有没有和它匹配的一个 0，且只能匹配一次
- 考虑用 $std::set$ 维护所有 0 出现的位置，然后用 $std::set::lower\_bound()$ 查询有无大于 $pre[a_i]$ 的 0 出现，若出现则使用 $std::earse()$ 删除，因为只能匹配一次
- 不出现即方案无法实现
- 用 $std::vector$ 也可以，但它的删除复杂度是 $O(n)$ 的，所以会导致 TLE
- 用 $std::lower\_bound$ 在理论上查询复杂度和 $std::set::lower\_bound()$ 是相同的，但是会由于一些原因，实际上复杂度会较高，所以也会导致 TLE
- 复杂度 $O(n \log n)$



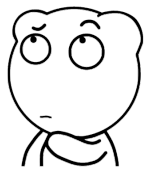
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



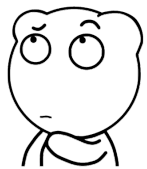
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



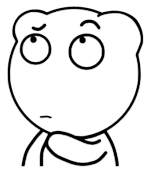
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



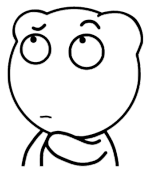
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



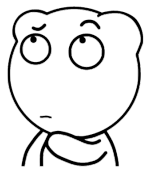
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



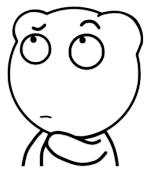
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- 令 $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (根据 wallis 公式)
- 得到化简后的式子，我们可以 $O(n)$ 处理完分子分母即可，
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



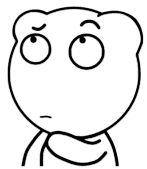
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- **令** $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ **(根据 wallis 公式)**
- **得到化简后的式子，我们可以** $O(n)$ **处理完分子分母即可，**
- 需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成 $O(nlogn)$ 导致 TLE



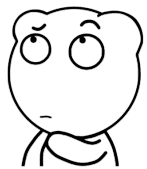
# Problem F. Petrichor_x 的课堂测验

- $\int_0^{+\infty}(\frac{1}{a^2+x^2})^b\,dx$
- **令** $x = a \times \tan y$
- $\Rightarrow \int_0^{\frac{\Pi}{2}}(\frac{1}{a^2+(a*\tan y)^2})^b\,d(a*\tan y)$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}(\frac{1}{1+\tan^2 y})^b * \sec^2 y\,dy$
- $\Rightarrow a^{1-2b}\int_0^{\frac{\Pi}{2}}\cos^{2b-2} y\,dy$
- $\Rightarrow a^{1-2b} * \frac{(2b-3)!!}{(2b-2)!!} * \frac{\Pi}{2}$ (**根据** wallis **公式**)
- **得到化简后的式子，我们可以** $O(n)$ **处理完分子分母即可，**
- **需注意分母全部乘完后，再取逆元，若对分母逐个取逆元，复杂度会退化成** $O(nlogn)$ **导致** TLE



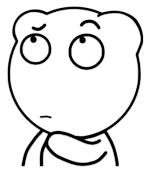
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



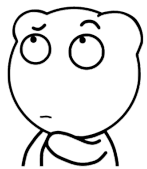
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



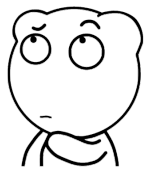
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



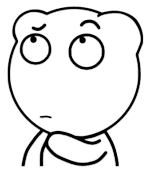
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



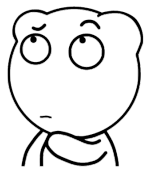
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



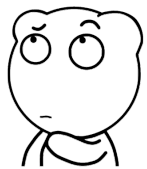
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



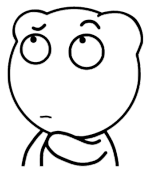
# Problem G. 魔塔

- $n \times m$ 的图，有怪兽和药，问找到公主的最短时间
- bfs()，用 $std :: priority\_queue$ 来维护
- 遇见怪兽时，一定要与其战斗，则可以经过这个点的条件是骑士当前血量大于战斗消耗的血量
- 遇见药时，可以选择喝或者不喝
- 假如喝药，则骑士恢复后血量不能超过最大血量
- 假如不喝，则骑士不会回复血量，但也不必花费回血所需时间
- 判断是否需要经过这个点的条件应该是，以当前血量值没有经过这个点，或者之前经过这个点的时间比当前时间长



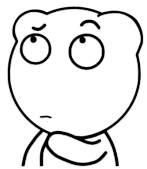
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2 \% p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2 \% p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2 \% p \neq k])$
- 时间复杂度 $O(nlogn)$



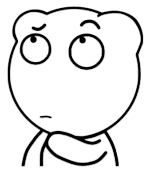
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2 \% p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2 \% p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2 \% p \neq k])$
- 时间复杂度 $O(nlogn)$



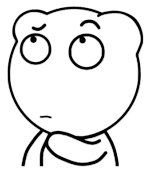
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2\%p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2\%p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2\%p \neq k])$
- 时间复杂度 $O(nlogn)$



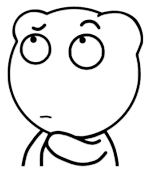
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2\%p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2\%p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2\%p \neq k])$
- 时间复杂度 $O(nlogn)$



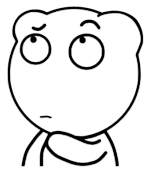
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2\%p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2\%p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2\%p \neq k])$
- 时间复杂度 $O(nlogn)$



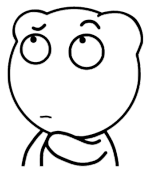
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2\%p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2\%p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2\%p \neq k])$
- 时间复杂度 $O(nlogn)$



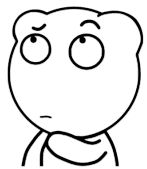
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2 \% p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2 \% p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2 \% p \neq k])$
- 时间复杂度 $O(nlogn)$



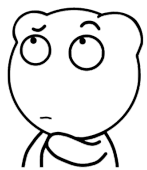
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2 \% p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2 \% p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2 \% p \neq k])$
- 时间复杂度 $O(nlogn)$



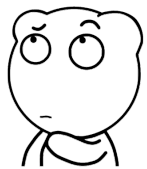
# Problem H. Petrichor_x 的简单数论

- 题目所要求的是满足 $a_i^2 + a_j^2 + a_i a_j \equiv k \ (mod \ p)$ 的 pair 个数
- 观察到 $a_i^3 - a_j^3 = (a_i - a_j)(a_i^2 + a_j^2 + a_i a_j)$
- 当 $a_i \neq a_j$ 时，由于 p 是素数，$\gcd(p, a_i - a_j) = 1$，故原式等价于 $a_i^3 - a_j^3 \equiv k * (a_i - a_j) \ (mod \ p)$
- $\Rightarrow a_i^3 - k a_i \equiv a_j^3 - a_j \ (mod \ p)$
- 故我们可以利用 map 维护对于所有 $(a_i^3 - a_i)\%p$ 值相同的个数
- 设 $(a_i^3 - a_i)\%p$ 有 $k$ 种值，每种值的个数为 $p_i$，则对于 $a_i \neq a_j$ 的情况，有 $\sum_{i=1}^{k} \binom{p_i}{2}$ 种满足条件的 pair 个数
- 当 $a_i = a_j$ 时，可发现仅当 $3 * a_i^2 \% p = k$ 时有贡献，所以在上述基础上减去 $3 * a_i^2 \% p \equiv k$ 的贡献即可
- 故 $ans = \sum_{i=1}^{k} \binom{p_i}{2} - \sum_{i=1}^{k} (\binom{p_i}{2} * [3 * a_i^2 \% p \neq k])$
- 时间复杂度 $O(nlogn)$



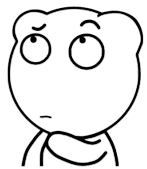
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



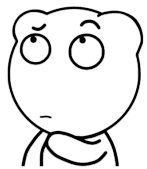
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



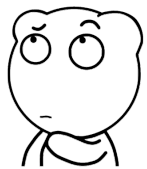
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



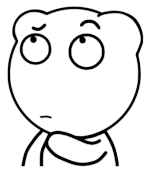
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



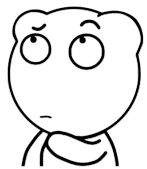
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，
  $1 \le p \le n, -n \le k \le n$
- 状态转移方程为
  $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



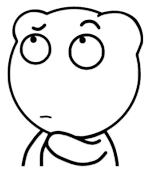
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



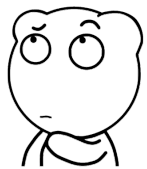
# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



# Problem E. 重修监察寮

- 输入正整数 $d$ 和 $n$ 个正整数 $a_1, a_2, .., a_n$ ，可以修改除 $a_1$ 和 $a_n$ 的其它数，要求修改后相邻两个数之差的绝对值之差不超过 $d$，且修改费用最小。修改后的费用为 $|a_1 - a'_1| + |a_2 - a'_2| + ... + |a_n - a'_n|$
- 多阶段决策，依次确定每个 $a_i$ 修改成某值时，修建成本为多少
- 但是，$d$ 的范围很大
- 通过推理可知，每个数修改后可以写成 $a_p + k \times p$ ，$1 \le p \le n, -n \le k \le n$
- 状态转移方程为 $f(i, x) = |a_i - x| + min\{f(i-1, y) | x - d \le y \le x + d\}$
- 滑动窗口用单调队列实现，在平摊 $O(1)$ 的时间复杂度内求出 $f(i, x)$
- 时间复杂度 $O(n^3)$
- UVA12170 手写单调队列



# 谢谢