比赛讲评

刘礼瑞,魏临源,吴梓禾,谷雨阳

2025-05-13

A Calc

【2】集合 【2】位运算

- $n < 10$

首先我们知道，$S \subset T$ 等价于 $S \cap T = S$；

然后我们可以将限制转化为二进制形式：

$$(x \operatorname{xor} y) \operatorname{and} (a \operatorname{xor} b) = (x \operatorname{xor} y)$$

枚举子集应该可以做到 $O(9^n)$，不会枚举子集就是 $O(16^n)$。

跑一段时间把 $n \leq 10$ 的都算出来就行了。

【2】位运算 【7】状态压缩动态规划

- $n \leq 10^6$

没有任何提示性的一点。

源于一位验题人打了 FMT。

有兴趣的同学自行了解。

【2】加法原理 【2】乘法原理

注意到，转化为二进制形式后，限制可以拆到每一位上，并且是互相独立的。

因此可以逐位考虑，暴力检查一位的所有情况：

```
for (uint64_t a : {0, 1})
  for (uint64_t b : {0, 1})
    for (uint64_t x = 0; x <= a; ++x)
      for (uint64_t y = 0; y <= b; ++y)
        if (((x ^ y) & (a ^ b)) == (x ^ y))
          ++cnt;
```

发现结果为 7，故答案为 7^n。

当然，你也可以观察样例得到结论。

B

Chord

- $k \leq 2$

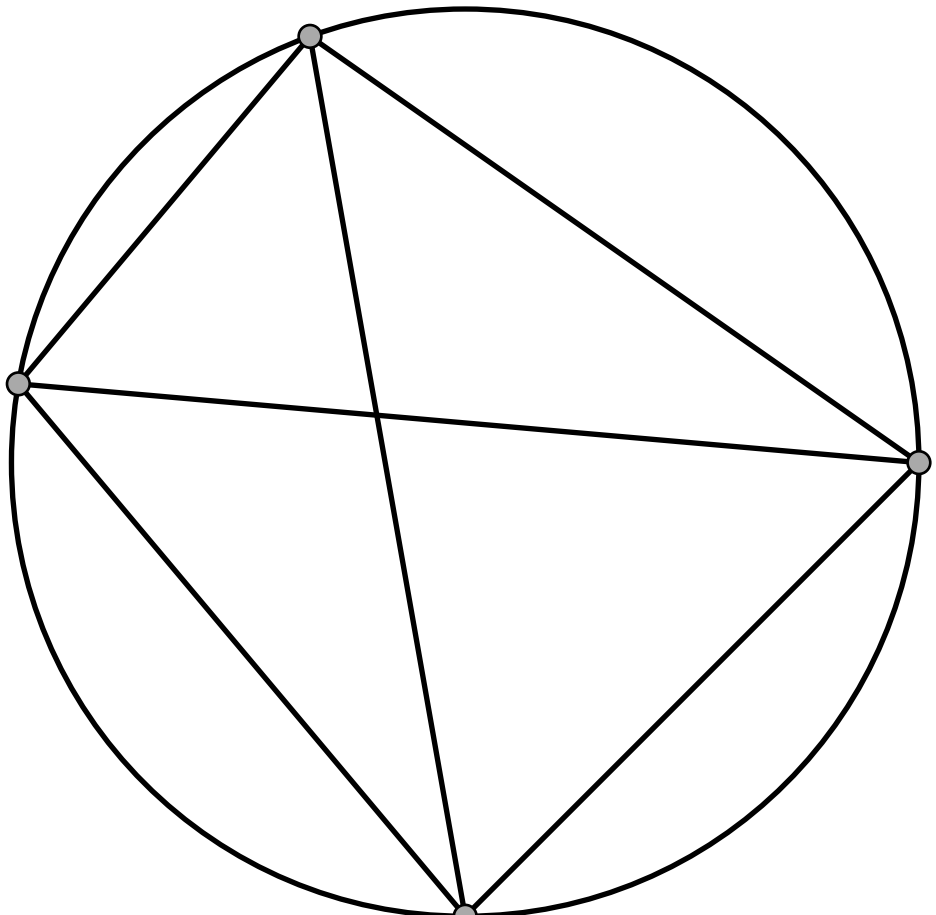
【1】几何（初中部分） 【8】概率的基本概念

怎么被你们猜到是 2 次了，不应该放样例的。

考虑几何构造，在圆上随机选 4 个点，显然其对应 3 种取弦的方案，其中只有一种相交，故在圆上随机选两条弦，他们相交的概率为 $\frac{1}{3}$。

因此 $k = 2$ 时，答案为：

$$\mathbb{E}[R] = \frac{1}{3} \times 4 + \frac{2}{3} \times 3 = \frac{10}{3}$$



- $k \leq 5$

【8】多项式函数的积分 【8】概率的基本概念

首先 $k = 2$ 有一种积分做法。

设 P 表示两条弦不相交的概率，则有：

$$\begin{aligned} P &= \int_0^1 x^2 \mathrm{d}x + \int_0^1 (1-x)^2 \mathrm{d}x \\ &= \frac{2}{3} \end{aligned}$$

类似也可以推出 $k = 2$ 时的答案。

因此剩下 $k = 3, 4, 5$ 的点，如果你的积分能力极强，可以积出来，~~但是我们都不会。~~

- $k \leq 10^{18}$

【1】几何（初中部分） 【4】组合 【8】概率的基本概念

注意到每两条弦都可能产生一个交点，而每个交点存在的概率是互相独立的（这里由几何意义显然），因此交点的期望个数为

$$\mathbb{E}[V] = \frac{\binom{k}{2}}{3} = \frac{k(k-1)}{6}$$

再注意到每个交点会贡献一个区域，因此答案为

$$\mathbb{E}[R] = k + 1 + \frac{k(k-1)}{6}$$

C

Christmas

- $n \leq 2000$

【1】枚举法 【6】最近公共祖先

给 $O(n^2 \log n)$ 或 $O(n^2)$ 留的。

暴力枚举两点 LCA。

【6】最近公共祖先

- $n \leq 2 \times 10^5$

给 $O(n \log^2 n)$ 留的。

使用树剖向上打标记和查询 LCA。

- $n \leq 10^7$
- 256 Mib

【6】最近公共祖先

考虑 LCA 的本质，发现只要一个点的子树内存在多于一个点，那么这个点就有可能成为 LCA，而每一次询问都已经限定了其中一个点是什么了，所以我们只需要从这个点开始暴力向上跳，边跳边打标记，然后如果跳到了一个之前被打过标记的点，答案就是该点了。

每个点只会被打上一次标记，时间复杂度 $O(n)$。

【6】最近公共祖先

- 4 MiB

注意到答案等于父亲序列的异或和，边读边异或，然后就不需要开任何数组了。

证明一下，显然对于每个点来说，有两种情况：

- 在该点被询问前该点**没有**被打上标记：显然当该点的孩子被打上标记时，该点就是这次询问的答案，则该点有多少个孩子就会被贡献多少次。
- 在该点被询问前该点**已经**被打上标记：显然该店第一次被打上标记时不会产生贡献，则孩子对其产生的贡献为孩子的个数 -1 次，而自己被询问的时候，由于自身已经被打标记了，则自身就是本次询问的答案，所以依旧是该点有多少个孩子就会被贡献多少次。

显然对于每个点有**多少个孩子就会被贡献多少次**等价于**父亲序列**。

时间复杂度 $O(n)$，空间复杂度 $O(1)$。

D

Coprime

- $Q \leq 6$

【1】模拟法 【2】加法原理 【2】乘法原理

手玩的分数，激励选手注意到一些东西。

D.2 30-40 pts

【1】模拟法 【2】加法原理 【5】深度优先搜索 【6】搜索的剪枝优化

首先应该注意到可以扔掉 1，这样由 $1 \sim n$ 的圆排列转为了 $2 \sim n$ 的排列。

然后可以打一个 $O(n!)$ 的暴搜，时间足够的话可能能做到 $n = 14$，这样应该有 42 分左右。

【7】状态压缩动态规划

然后你应该联想到哈密顿路径并且开始建图。

这里状压 dp 就非常典型了，时空复杂度均为 $O(n2^n)$，其中空间复杂度应该是限制选手的点。

$n = 25$ 时 $n2^n \approx 8 \times 10^8$，应该可以跑出来，这样的分数为 72 分。

【7】状态压缩动态规划 【8】动态规划的常用优化

然后你应该注意到一些数字是等价的，例如

1. 如果两个数的质因数集合相同，则这两个数显然等价；
2. 如果一个数为大于 $\frac{n}{2}$ 的质数，则这个数与 1 等价。

如果你没有注意到这一点，考虑到哈密顿路径的不可做性和数据范围，你应该考虑缩减状态数，因此你也可以考虑在图中寻找对称（等价）的点。

然后问题变成了可重集排列，可以进行 dp。

总状态数应该不超过 $79626240 \times 19 = 1512898560$（这里是从数学推理得到的等价数字，我不知道考虑对称点能否更优）。

直接开空间大概有 $5.6\mathrm{G}$，大部分同学的计算机应该是负担的起的；也可以考虑滚动数组优化；总时间应该在 $1\ \mathrm{min}$ 以内。

好像还能考虑 meet-in-middle，如果我有时间可以考虑加强。

其实这个数列在 OEIS 上有收录，可以查到前 29 项。

另外，本题的模数 83456729 是一个符合要求的 $2 \sim 9$ 的排列。

原题为 PE886。

E

Departments

【1】模拟法 【4】树的表示与存储

暴力枚举或者乱搞都能过。

【1】模拟法 【4】树的表示与存储

- $P_i = Q_i$

最优情况下路径长度为边权和的两倍，读者自证不难。

那么路径的安排只要满足上面的条件，随便搞搞就行。

【3】贪心

对于一个点 u，至少需要耗费 P_u 的人。这里认为 $P_u \geq Q_u$，因为即使 $P_u < Q_u$，该节点也需要 Q_u 的人。

若只有一个节点，那么容易得到耗费的人为 P_u。

若有两个节点，不妨设为 x 和 y，且 $P_x - Q_x > P_y - Q_y$，即 x 剩下的人数比 y 多，那么感性理解肯定要先走 x，才能充分发挥剩余的人。试图证一下，先 x 后 y 的总花费为

$$\max\{P_x, P_x + P_y - (P_x - Q_x)\} = \max\{P_x, P_y + Q_x\}$$

则先 y 后 x 的总花费为

$$\max\{P_y, P_x + Q_y\}$$

将上面的式子移项可得 $P_y + Q_x < P_x + Q_y$，结论得证。

将结论推广到同一层上有 k 个节点，可以通过两两依次缩点的形式转化为两个节点。

【3】贪心 【4】树的表示与存储 【6】树形动态规划

在上面所有结论的基础上考虑更一般的情况，按照最短路径，一定是先遍历完每个子树最短。所以搜到一个点时，先处理这个点为根的所有子树，按照剩余人数排序，递归继续计算。

注意搜索时需要从 $P-Q$ 最大的点开搜。

原题：Luogu P8677。

F

Division

【1】枚举法 【3】模运算与取余 【7】模运算意义下的逆元

首先，对于有理数取模，有定义：

若既约分数 $\frac{a}{b}$ 满足 b 与 p 互质（b 没有零因子），则存在 x 使得 $a \equiv bx \pmod{p}$，且 x 即为 $\frac{a}{b}$ 在模 p 意义下取模的结果。

我们不难说明，若 $a \equiv s \pmod{p}, b \equiv t \pmod{p}$，则 $\frac{a}{b}$ 与 $\frac{s}{t}$ 取模后的结果是一致的。

因此，满足 ab 最小的 $\frac{a}{b}$ 一定至少满足 $a < p, b < p$。

所以只需要从 1 到 $p-1$ 枚举 a（或者 b），取最小的结果即可。

时间复杂度 $O(p)$。

【1】枚举法 【6】搜索的剪枝优化

首先这样的分数显然不会超过 2 个（虽然这个结论没啥用，所以留给读者自证）。

从 1 开始枚举，将每个数分别尝试作为分子和分母，得到分数并更新最优解。

假设当前最优解为 $\frac{a}{b}$，有 $l = ab$，我们证明最多枚举到 $\sqrt{l}$ 即可。

若答案的 a 或 b 大于 $\sqrt{l}$（不妨假设是 a），此时 b 不能大于 $\sqrt{l}$，否则 $ab > l$，矛盾；若 b 小于等于 $\sqrt{l}$，则此前一定被枚举过。

由于初始 $ab = x$，算法复杂度为 $O(\sqrt{x})$。

本题是 weilycoder 的原创题，使用请通知。

G

Divisor

你们怎么都在冲 Open Problem 啊。

先把我在 Luogu 上放的测试题目贴一下：U552400。

感觉随机输出能得到不少分的样子。

首先可以选择一个高复合数（另一个名字是“反素数”），例如 $720720, 1441440, 2162160$，然后随机上一些小素数，接着乘到值域上限，这个做法感觉上应该拿到 30000-40000 不成问题。

另一种做法是先选出一些因数个数较多的数字，这里可以选择搜索，但是注意不能太限制质因子的大小。

如果你的计算机内存比较充裕，可以考虑筛出 2×10^9 以内每个数的因数个数。

我选出了所有因数个数大于 400 的数字。

接着，考虑贪心的选择能使得因数个数增加最多的数字。

一个小优化是，如果在产生的答案中，一个因数被很多数字包含，我们可以认为它被选择的概率很大，从而不考虑它对答案产生的贡献。

这应该很容易接近 50000。

听说直接生成一些看起来未出现因数个数很多的数字也可以做到这个效果。

考虑模拟退火。

这里不讲具体方法；总之，经过反复调参，应该很容易突破 50300。

反复退火应该能上 50600。

到达这种地步，模拟退火随机到更优解的概率已经很小了，继续考虑贪心。

找出我们筛选的因数个数很多的数字，对于每个位置，考虑用更优的数替换，反复迭代，最终 std 达到了 50823。赛时无人超越。

```
1344863520, 1353512160, 1502701200, 1610809200, 1622864880, 1632430800, 1638501480, 1651890240,
1669187520, 1679277600, 1697295600, 1703782080, 1719998280, 1721079360, 1722520800, 1728342000,
1730907360, 1736935200, 1746763200, 1747357920, 1747746000, 1751349600, 1762160400, 1794592800,
1798851600, 1803831120, 1805403600, 1809007200, 1822700880, 1824863040, 1830628800, 1834953120,
1835673840, 1836394560, 1838319840, 1850808960, 1852250400, 1864225440, 1867924800, 1869880320,
1870268400, 1870767360, 1870989120, 1872347400, 1877614200, 1878667560, 1887815160, 1888286400,
1891643040, 1893855600, 1894898880, 1895493600, 1900538640, 1908223200, 1917946800, 1918418040,
1918425600, 1923339600, 1924322400, 1928646720, 1932084000, 1933470000, 1934357040, 1935133200,
1935632160, 1935853920, 1938736800, 1943781840, 1944502560, 1944558000, 1948106160, 1951349400,
1951876080, 1953151200, 1953982800, 1954592640, 1954789200, 1956754800, 1958816160, 1959859440,
1963241280, 1964682720, 1970211600, 1972998720, 1974772800, 1977444000, 1980316800, 1980493200,
1980538560, 1980760320, 1981150080, 1981564200, 1985256000, 1989187200, 1991349360, 1992957120,
1996394400, 1999000080, 1999277280, 1999670400
```

H

Exchange

【1】模拟法

将 solve 函数翻译为 C++ 语言，按照询问顺序调用即可。

时间复杂度为 $O((n+m)q)$。

【8】离线处理思想

- $l = 1$

因为所有的询问左端点都是 1，所以每次运行 `solve` 都是从第一项开始跑。这很浪费，其实跑一趟就够了。

使用离线的处理办法，把所有询问开桶用右端点记录下来。然后，从 1 到 n 模拟整个过程。模拟到某个被询问的位置时，直接回答就可以了。

时间复杂度：$O(n + m + q)$。

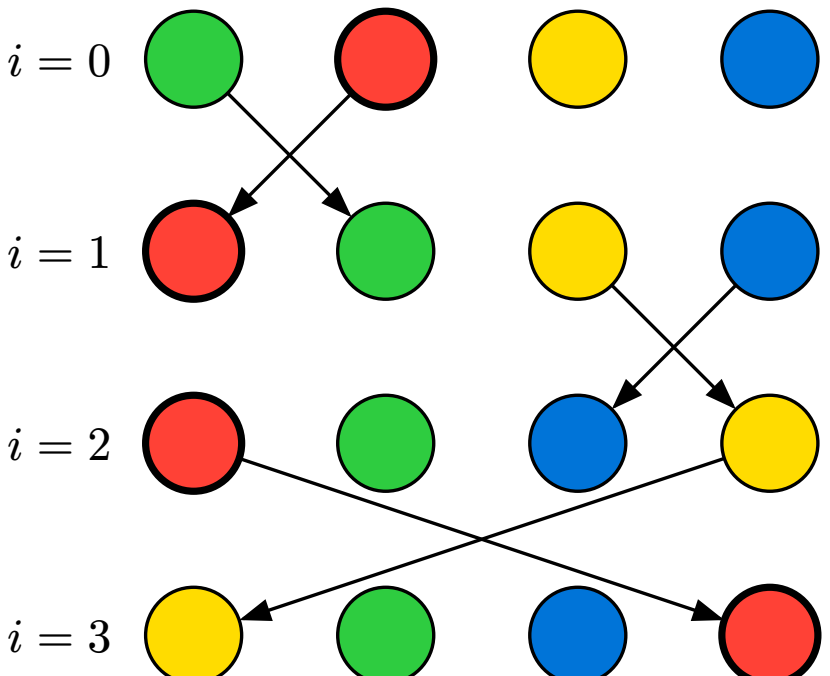
特殊性质：$r = m$

今有 N 老师云：正难则反。这个测试点启发你反过来处理操作。

为了避免歧义，这里使用颜色标记。回答 $\mathrm{SOLVE}(1, 3, 4)$ 时，考虑反过来操作，我们可以**跟踪第 4 个节点**（图中的红色节点），倒序处理为开始状态，被跟踪的节点的**位置**（第 2 个）即为答案。在此过程中，每个元素都使用元素标记，全程没有关注过某个数具体是多少。

本例操作序列：

```
((2, 1), (3, 4), (1, 4))
```



让我们放眼到多个询问。持续跟踪每个节点的位置是非常容易的，开个数组维护每个元素的位置，每次交换再顺手交换一下所对应的元素位置即可。对于每个询问，回答其对应的 k 在现在哪里即可。

时间复杂度为 $O(n+m+q)$。

【8】离线处理思想

最后的做法和 subtask 3 非常接近了！

在那个例子当中我们都是对用颜色标记的节点操作，并没有关心具体的数字是多少，只关注这个节点最后去到了哪里。

采用离线的方法，从后往前扫，遇到一个询问的右端点就开始跟踪对应的点，遇到左端点就读取被跟踪的节点的位置作为答案即可。

最后的时间复杂度为：$O(n+m+q)$。

I

Graph

【3】排序的基本概念

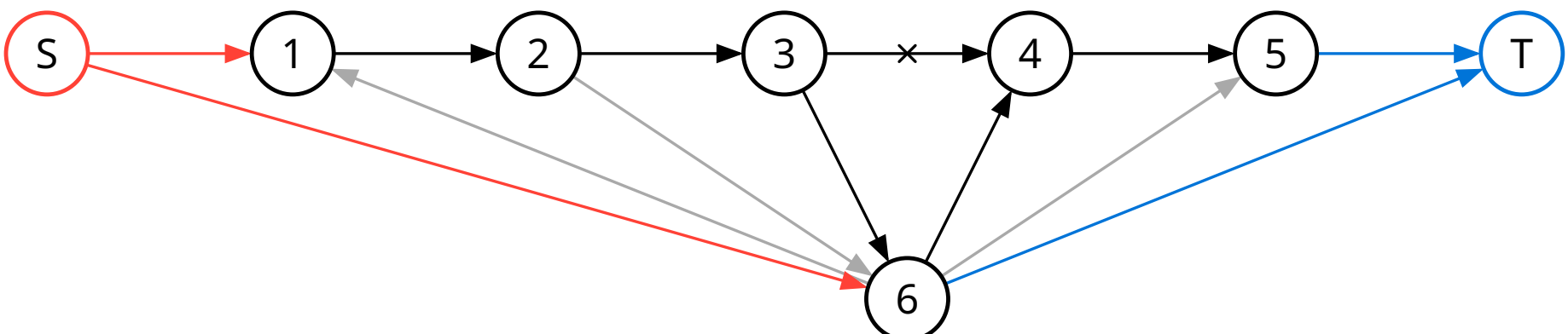
由于图是一张 DAG，各个点存在**全序**关系。因此，可以使用一种比较次数严格 $n \log_2 n$ 次的排序算法，例如堆排序、二分优化的插入排序、归并排序等。

快速排序的复杂度基于期望，按理说是过不了的。赛时数据较水，不过没人用这个方法。

【1】枚举法

假设我们已经找到了一条链，希望在链中插入一个新的节点 x。如果我们从链上找到了一个点 p，满足存在边 $p \to x$，且 p 在链上的后继节点 q 满足存在边 $x \to q$，即可将 x 插入在 p 和 q 之间。

此外，假如存在一条 x 指向链首节点的边，可以直接将 x 接在链首前，链尾同理。对于处理这种情况，我们可以通过建立超级源点 S 和超级汇点 T 的方式规约到一般情况。



如图，可以将节点 6 插入到 3 与 4 之间或 1 之前。

我们需要找到这样的节点。由于每条边的方向都随机生成，这样的节点应该非常常见。

使用暴力枚举的方法挨个询问，直到找到符合要求的点后直接退出即可。

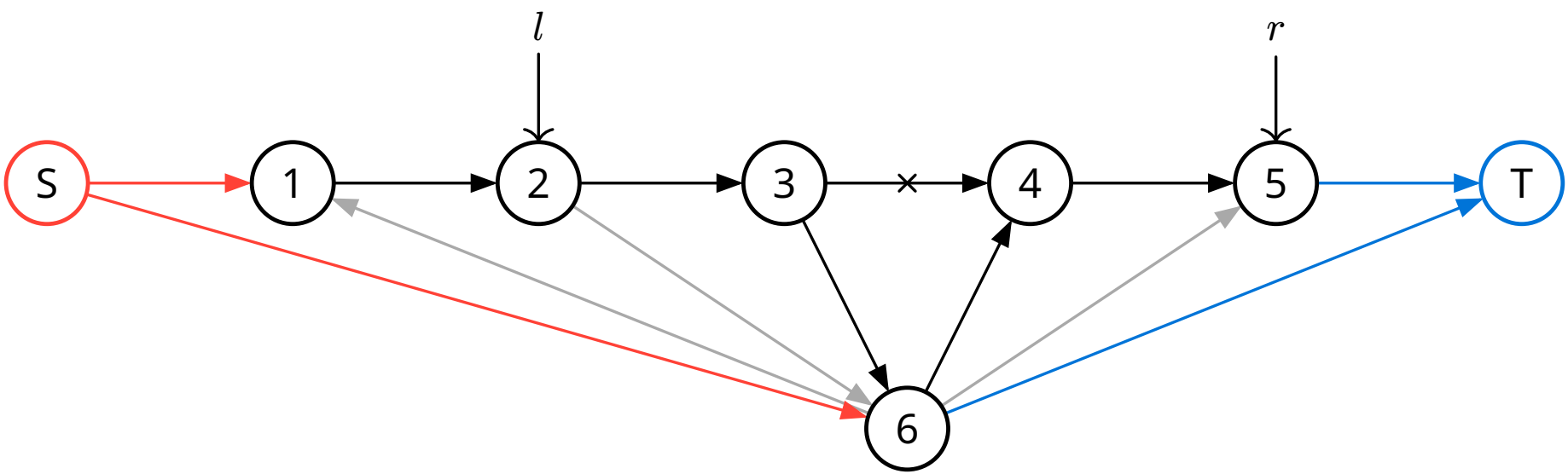
对于 Subtask 1，每插入一个节点，最差询问 $O(n)$ 次，足够了。

对于 Subtask 2，每插入一个节点，期望询问次数为 2 次。

【4】二分法

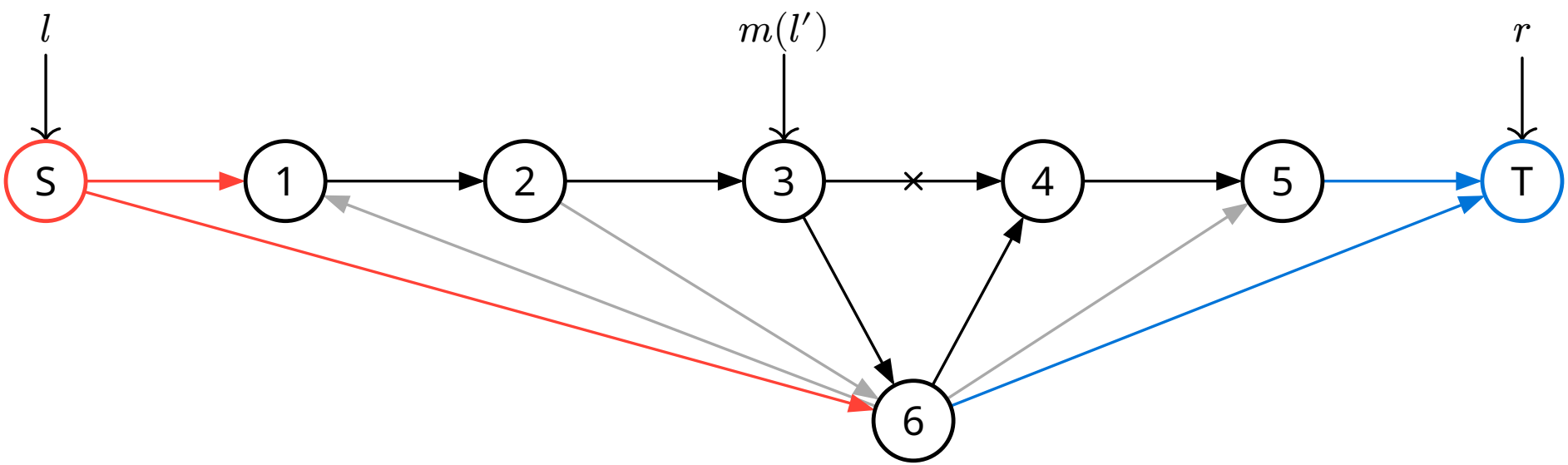
这是本题最天才的一步。

将左指针放在任意一个指向 x 的节点，右指针放在任意一个被 x 指向的节点。那么我们可以断言：l 和 r 之间必然存在满足之前提到的要求的点 p。



初始时让 l 和 r 分别指向 S 和 T。使用乌姆尼克算法二分法询问链中 l 和 r 中间的点 m 与 x 之间连边的方向。倘若 m 有指向 x 的边，则 $l \leftarrow m$，否则 $r \leftarrow m$。这样，一定可以保证 l 存在边指向 x，r 存在边指向 y。根据上述结论，l 与 r 之间一定存在合适的节点 p。

当 l 和 r 相距 1 时，我们便找到了答案，即 l 是要求的 p，r 是要求的 q。



示例代码如下。

```cpp
bool ask(size_t a, size_t b) {
  fout << "? " << a + 1 << ' '
    << b + 1 << endl;
  bool t; fin >> t; return t;
}
int main(void) {
  size_t n; fin >> n;
  vector<size_t> ans(n);
  iota(ans.begin(), ans.end(), 0);
  ranges::sort(ans, ask);
  fout << "! ";
  for (size_t i : ans)
    fout << i + 1 << ' ';
}
```

写完之后你将会发现，整个过程其实就是二分优化的插入排序。

此外，归并排序、快速排序对于本题也是正确的，留给你证明。不过快速排序的比较次数非严格 $\log_2 n$ 次，因此只能通过本题的第一个测试点。

本题原题是 Luogu P10451，蓝书上也有。

J Hack

```
for (size_t i = 1; i < n; ++i) {
  int x;
  fin >> x;
  if ((li)x * s.back() > 0)
    s.back() += x;
  else
    s.emplace_back(x);
}
```

这份代码通过把两个数乘起来看符号的方式判断两个数的符号是否相同。不过，`s.back()` 为 `long long int` 类型，与 `int` 相乘将会导致整型溢出的问题。

所以，构造若干个极大的正数就可以卡掉了。

```
while (!q.empty()) {
  ...
  for (pair<ui, vector<size_t>> const &i : sn[p])
    if (i.first != c)
      for (size_t j : i.second)
        if (!dis[j].count(i.first) || dis[j][i.first] > d + 1)
          dis[j][i.first] = d + 1, q.emplace_back(j, i.first);
}
```

代码的这一部分尝试在每次循环到当前节点时都遍历一遍当前节点所有的边。当图是一张菊花图时，菊花心会重复遍历与之相连的所有边，运行时间爆炸。

这份代码非常暴力地把该点所有可能出现的值都塞进去了。需要想办法让每个点都塞进去尽可能多的数。

720720 有非常多的因子（240 个）。可以构造一种方案，有许多条路径都经过 720720 与他的一 个因子，这样靠后的每个单元格都会塞满他的 240 的因子。例如：

```
720720  720720  720720  720720  720720  720720  720720  720720  ...
720720  1       720720  2       720720  3       720720  4       ...
720720  5       720720  6       720720  7       720720  8       ...
720720  9       720720  10      720720  11      720720  12      ...
720720  13      720720  14      720720  15      720720  16      ...
720720  18      720720  20      720720  21      720720  22      ...
720720  24      720720  26      720720  28      720720  30      ...
720720  33      720720  35      720720  36      720720  39      ...
720720  40      720720  42      720720  44      720720  45      ...
...
```

K

Hide

【1】枚举法

暴力枚举值域内每个数并对每个限制 Check 即可。

时间复杂度 $O(nm)$。

【2】集合

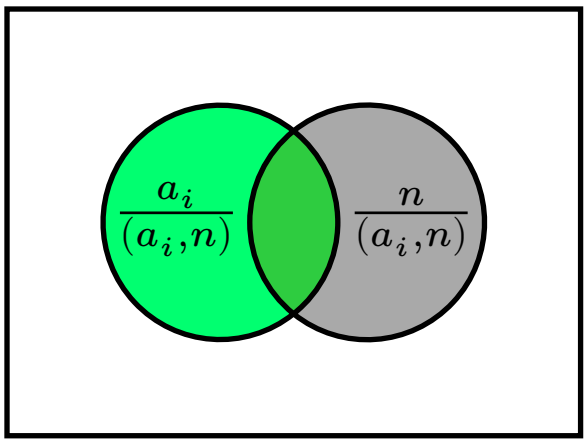
考虑一对 a_i, b_i 和 n 的因数集合的关系：

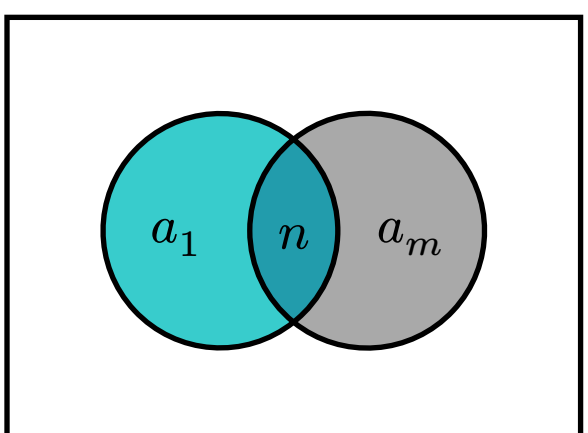
右上图中深绿色和浅绿色表示 a_i，灰色和深绿色表示 n，灰色部分为 b_i（用题中定义表示）。

注意到保证了 $(a_1, a_m) = 1$，也就是说 a_1 和 a_m 的因数集合不交，则 $a_1 b_1$ 与 $a_m b_m$ 相交的部分即为 n。

接下来只需要确定求出的 n 是否满足所有约束条件即可。

赛时题中没有给出无解的数据；这个可以认为是 MrPython 的锅。





L

Judge

【1】枚举法 【2】位运算

暴力。

对于每个分组枚举两两之间是否满足要求。

时间复杂度 $O(n^2)$，无任何技术含量。

【1】枚举法 【2】位运算

对于每个分组，考虑如何快速验证。

注意到给定限制等价于任意两个数表示的集合互不包含（具体说明见 Pack 题题解）。

对于每个组，都可以采用子集枚举的方法验证每个是否存在包含关系。对 2^X 以内的数分别进行子集枚举的时间复杂度为 $O(3^X)$。所以，进行 k 次子集枚举的复杂度不超过 $O\left(k3^{\log_2 v}\right)$。

示例如下。

```
for (size_t d = 0; d < k; ++d)
  for (uint32_t s = 0; s < (1u << 18); ++s)
    if (b[d][s])
      for (uint32_t j = s & (s - 1); j; j = (j - 1) & s)
        if (b[d][j]) { /** NO **/ }
```

这样还是不够优秀。实际上我们不需要进行 k 轮子集枚举。首先变换一下循环的顺序，将遍历所有分组的集合提到里面去。

```
for (uint32_t s = 0; s < (1u << 18); ++s)
  for (uint32_t j = s & (s - 1); j; j = (j - 1) & s)
    for (size_t d = 0; d < k; ++d)
      if (b[d][s] && b[d][j]) { /** NO **/ }
```

复杂度 $O\left(3^{\log_2 v}\right)$

里面这一坨完全是按位与运算的逻辑。我们直接把所有 d 相同的项压到一个数字去，然后用位与运算判断，这样就可以把复杂度里的 k 省掉了。

```
for (uint32_t s = 0; s < (1u << 18); ++s)
  for (size_t d = 0; d < k; ++d)
    c[s] |= uint32_t(b[d][s]) << d;
for (uint32_t s = 0; s < (1u << 18); ++s)
  for (uint32_t j = s & (s - 1); j; j = (j - 1) & s)
    if (c[s] & c[j]) { /** NO **/ }
```

【2】位运算 【7】状态压缩动态规划

可以考虑使用 DP 解决子集的问题。对于每个分组分别考虑，设 $f_{d,S}$ 表示第 d 组中有没有元素为 S 的子集，则

$$f_{d,s} = \bigvee_{x \in S} f_{d,S \setminus \{x\}}$$

这是一个朴素的状压 DP，直接做就好。这样总的复杂度是 $O(kv \log v)$ 的。

```
for (size_t d = 0; d < k; ++d) {
  for (uint32_t s = 1; s < (1u << 18); ++s) {
    for (uint32_t j = 0; j < n; ++j)
      if ((1u << j) & s) f[d][s] |= f[d][s ^ (1u << j)];
    if (b[d][s] && f[d][s]) { /** NO **/ }
    f[d][s] |= b[d][s];
  }
}
```

【2】位运算 【7】状态压缩动态规划

采用和第 2 节一样的优化，对第三节的 DP 进行数组进行压缩，把复杂度里的 k 压掉。复杂度只有 $O(v \log v)$！

```
for (uint32_t s = 1; s < (1u << 18); ++s) {
  for (uint32_t j = 0; j < n; ++j)
    if ((1u << j) & s)
      f[s] |= f[s ^ (1u << j)];
  if (c[s] & f[s]) { /** NO **/ }
  f[s] |= b[s];
}
```

M

Locate

【1】枚举法

使用 $n \times m$ 次询问把整个数组问出来就好了。

没啥脑子。

【3】二分法

每行和每列都是单调的，可以分别对每行或每列进行二分。

总的询问次数约 $n \log_2 m$ 或 $m \log_2 n$。

由于每行每列都是单调的，所以我们考虑从右上角那一个数开始问：

- 如果比目标数字大：直接排除一列。
- 如果比目标数字小：直接排除一行。
- 如果就是目标数字：你找完了呀。

你一直问右上角的数，就可以在 $n+m$ 次询问内解决了。

【10】熵、互信息、条件熵、相对熵

本题的 SPJ 将所有解都卡到了最劣，因此思考本题 SPJ 的写法其实很有提示性的。

注意到询问一个点与目标点的关系后，由于左上角的点均小于等于这个点，右下角的点均大于等于这个点：

- 若该点小于所求点，则右下角的所有点均小于所求点，因此右下角的所有点均被排掉；
- 若该点大于所求点，则左上角的所有点均大于所求点，因此左上角的所有点均被排掉。

所以，SPJ 可以计算出：

1. 当前哪些点被确定；
2. 若回答 `<`，有多少点会被排除；
3. 若回答 `>`，有多少点会被排除。

因此，

1. 如果当前询问的点已经被排除，SPJ 返回确定的结果；
2. 如果**只有**询问的点未被确定，SPJ 返回 `=`；
3. 否则，SPJ 返回能排除点的数量**最少**的回答（对的，单次询问 SPJ 的时间复杂度为 $O(n^2)$）。

N Matrix

【1】模拟法 【6】矩阵的运算

按题意模拟即可。

【5】数值哈希函数构造

希望快速矩阵乘法卡不过去。但是卡过去就卡过了罢。

若 $A \times B = C + D$，显然有

$$V \times (A \times B) = V \times (C + D) \quad (1)$$

矩阵乘法有结合律，故

$$V \times A \times B = V \times (C + D) \quad (2)$$

令 V 是 $t \times N$ 的随机矩阵，这里 t 为小常数，检查上式是否成立即可。

在式 (1) 不成立的情况下，可以认为式 (2) 两侧 $t \times M$ 个数均为 $\mathbb{Z}_{998244353}$ 中随机取的点，因此我们认为这个做法的错误概率小于 $\left(\frac{1}{998244353}\right)^{t \cdot M}$。

实际上取 $t = 1$ 完全够用，复杂度 $O(N^2)$，这里认为 N, M, K 同阶。

原题：POJ 3318，Luogu P10102。

O

Pack

【1】枚举法 【4】深度优先遍历

前 3 个测试点中，n, k 较小，因此考虑暴力枚举每个点的编号。

复杂度 $O(n^n)$。

【7】状态压缩动态规划

前八个测试点都满足 $n \leq 1$，很容易联想到状态压缩动态规划。首先暴力枚举所有行李集合的每个子集，判断是否存在两个元素违反条件，求出所有可行的行李集合，这一步的复杂度是 $O\left(2^n \times n^2\right)$ 的。

然后考虑 dp，设 f_S 表示行李集合 S 所需的最少盒子，则有：

$$f_S = \min_{U \in S, U \text{ is good}} f_{S-U} + 1$$

这一步的复杂度为 $O(3^n)$，总的复杂度为 $O\left(3^n + 2^n \times n^2\right)$。

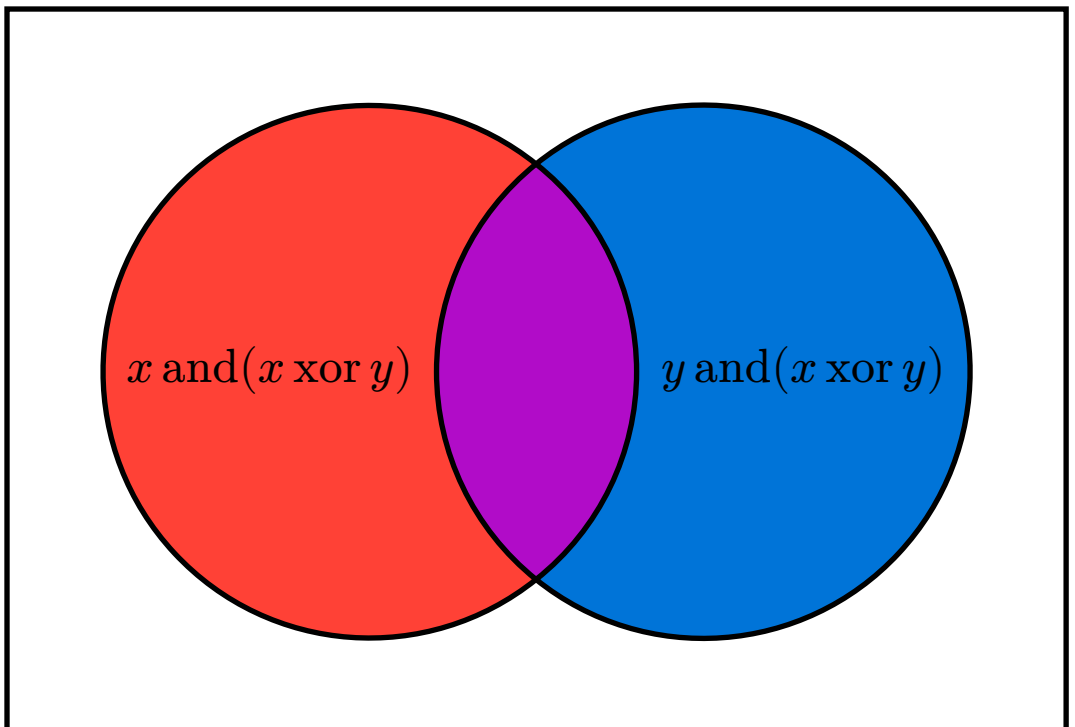
【2】集合 【2】位运算

【6】有向无环图的拓扑排序

先分析限制是什么牛鬼蛇神。

$$x \operatorname{and}(x \operatorname{xor} y) \neq 0$$
$$y \operatorname{and}(x \operatorname{xor} y) \neq 0$$

把 Venn 图画一下。



所以限制就是说，如果把数的二进制表示看成集合，**那么任意两个放在一个组内的行李的值不能是包含关系**。

由于集合包含是偏序关系，因此考虑当 $A \subset B$ 时从 A 向 B 连边。

考虑这张 DAG 上的最长链，设其长为 L，由于这条链上任意两个点都不能在同一个组，因此有 $X \geq L$，这里 X 表示答案。

考虑对图进行拓扑排序，在拓扑排序的过程中，每次将所有入度为 0 的点删掉，显然这些点两两之间不包含，因此这些点可以放进一组。由于最长链的长为 L，因此 L 次可以将图取完，故 $X = L$。

因此 $O(n^2)$ 建图后跑拓扑排序即可。

顺便，我们发现，答案至多为 $\max(\log_2 a_i) + 1$。

设计上这个做法是 50 分，实际可能可以卡到 60 pts。

【2】集合 【2】位运算 【7】状态压缩动态规划

方便表述，设 $\log_2 a_i < k$。

考虑优化上页的建图方法。

使用枚举子集的方法，可以做到 $O(3^k)$ 建图。

因此总的复杂度为 $O(3^k)$。

【2】集合 【2】位运算 【7】状态压缩动态规划

考虑将建图的复杂度优化到 $O(nk)$。

只需要对每一个数建一条连向其超集的边即可。

但是一个数的超集可能有很多，其实我们只需要连 popcount 比该数的 popcount 大 1 的数即可（换句话说，元素数恰好比给定集合大 1 超集），显然对于任意一个数 x 来说，这样的数至多只有 $\log_2 x$ 个，于是时间复杂度就是 $O(n \log V)$ 的。

至于分组方案，显然也是简单的，只需要把距离 0 相等的那些数分到一组里即可，用 `vector` 即可简单维护。

原题为：Luogu P4934。

P

Remove

【6】记忆化搜索 【6】时间复杂度分析

考虑 $D(n)$，显然有一个递推

$$D(n)=D\left(n-\left\lfloor\sqrt[3]{n}\right\rfloor^3\right)+1$$

注意到求和的时候可以将大于 $\left\lfloor\sqrt[3]{n-1}\right\rfloor$ 的部分统一处理，因此有

$$S(n)=S(p^3)+S(n-p^3)+(n-p^3)$$

这里 $p=\left\lfloor\sqrt[3]{n-1}\right\rfloor$。

上一个记忆化搜索就完了。

分析时间复杂度，递推式的 $S(p^3)$ 一项显然只有 $O(\sqrt[3]{n})$ 个，不妨先假定已经全部计算完毕，那么单次计算的时间复杂度满足

$$T(n)=T(n-p^3)+O(1)$$

由于 $n-p^3<(p+1)^3-p^3=3p^2+3p+1$，因此有 $n-p^3=O\left(n^{\frac{2}{3}}\right)$，显然最多递归 $O(\log\log n)$ 层。

综上所述，这个记忆化搜索的时间复杂度是 $O((\sqrt[3]{n}+T)\log\log n)$ 的。

原题是 PE884。

Q

Set

【2】集合

四种运算可以任意组合，太过复杂。尝试将有解的充要条件写出来。

- 考虑对于一个元素 $x \in U$。倘若存在另一个元素 $y \in U$，满足：
  对于任意的 $1 \le i \le n$，S_i 要么同时包含 x 和 y，要么同时不包含 x 和 y。
  此时，我们没有任何办法将 x 和 y 区分开。因此构造出的任何集合，要么同时包含 x 和 y，要么同时不包含 x 和 y，这样就不可能得到集合 $\{x\}$。
- 若不存在这样的 y，便可以考虑如下构造过程：设

$$T_i = \begin{cases} S_i \text{ if } i \in S_i \\ \complement_U S_i \text{ if } i \notin S_i \end{cases}$$

  计算 $\bigcap_{i=1}^{n} T_i$，即可得到想要的集合 $\{x\}$。

按照上述过程模拟，对于每一个 x 都判断是否存在这样的 y。时间复杂度 $O(n^2 m)$。

【2】集合 【6】字符串哈希函数构造

将 Subtask 1 的做法优化一下。

对于每个元素，可以使用一个长度为 m 的 01 序列表示其在各个集合的情况，第 i 位表示该元素是否存在于 S_i 中。

把 n 个长度为 m 的 01 序列求出来，然后对每个 01 序列都做一次**字符串哈希**，得到 n 个数字。判断一下这 n 个数字中有多少个只出现了 1 次就好了！

由于本题的每个串只用来判断整个串是否相等，不需要考虑添加 / 删除字符的问题，因此哈希时各个位的权值可以图省事直接随。权都是随的了，那对着卡都很难，直接用 `unsigned long long` 自然溢出也问题不大。

【2】集合 【4】差分 【6】字符串哈希函数构造

接着 40 pts 的做法再优化。

输入以区间形式给出。我们需要在由字符串哈希构成的数组中做区间加操作，因此使用差分数组就好了。

原题来自信友队。有人赛时对外求助，应当谴责，但这个题我觉得还蛮不错的。

至于为什么这个题有 20 pts 的卡常点，那是因为原题出题人没素质，我直接搬来了！

R

Sort

【3】冒泡排序

由于你会冒泡排序，因此 $O(n^2)$ 次比较的算法是显然的。

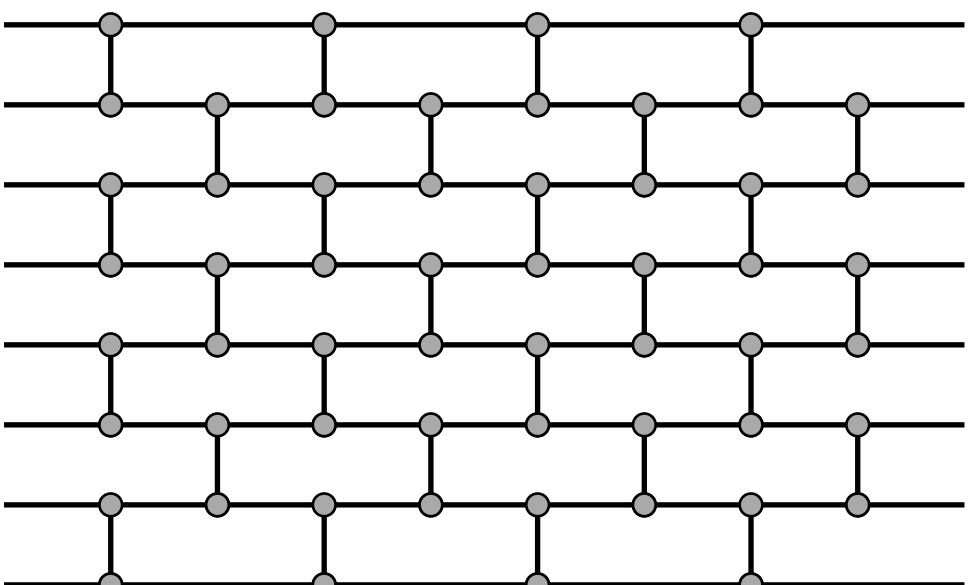
【3】冒泡排序

其次，你注意到冒泡排序的流程中，有一些 CheckAndSwap 操作是不冲突的，你大概可以将排序次数优化到 $2n$ 次左右。

【3】冒泡排序

再次，作为一名合格的 OI 选手，你应该快速意识到，可以对奇偶位置分别操作，将次数优化到 n 次。

大概像这样：



以上内容应该在 10 分钟内结束思考，在继续之前，我们需要先考虑一个问题：

如何更简便地判断一个排序流程是正确的?

我们可以注意到，如果我们的流程是正确的（可以对任意的序列排序），当且仅当我们可以对任意的 01 序列排序。

考虑证明其逆否命题：基于任意一个**不能**被正确排序的序列，我们可以构造出不能被正确排序的 01 序列。

细节留作练习。

通过这个引理，我们可以利用数学归纳法证明上一页中的 n 次排序流程是正确的，细节留给读者。

【4】递归法

该考虑正解了。

作为一名合格的 OI 选手，你应该联想到归并排序，并将题目的数据范围由 100 扩展到 128 考虑。

假如你还可以从 30 联想到 $28 = 1 + 2 + 3 + 4 + 5 + 6 + 7$，你可以意识到我们应该寻找一种可以在 $\log n$ 次操作内完成归并的算法。

这个过程并不简单，并可能是整道题中难度最大的部分。

不过如果你在做题过程中注意到了上述引理，应该也不算过于困难。

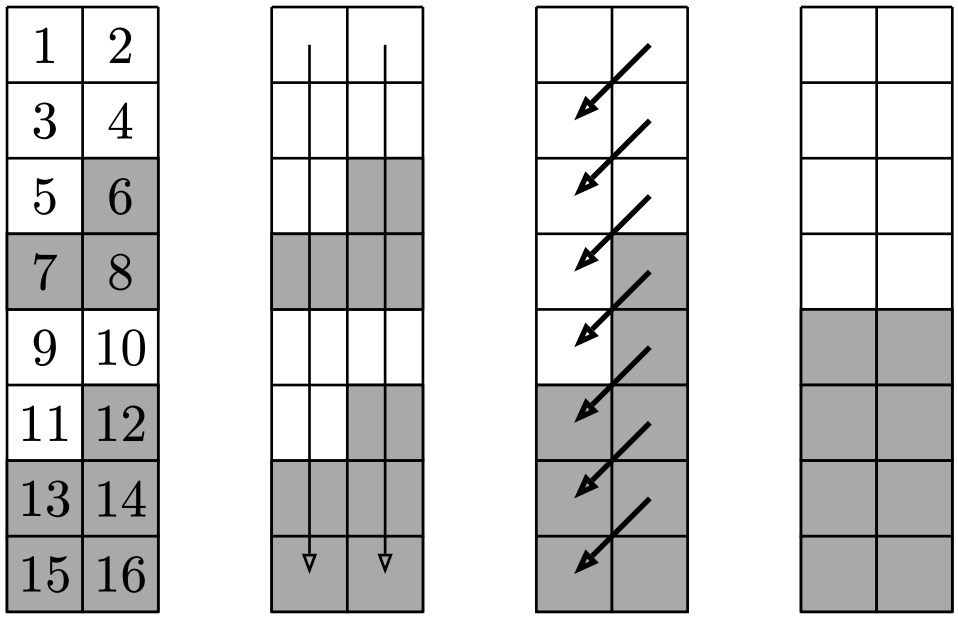
我们只考虑 01 序列（下标从 1 开始）。

首先，假定序列的前半部分和后半部分均已经完成排序（图中灰色表示 1）。

然后，我们考虑**递归地**归并**奇数下标的子序列**和**偶数下标的子序列**。

接下来，由于偶数下标的子序列可能会比奇数下标子序列 1 的数量多 $0 \sim 2$ 个，我们需要一次操作处理这种情况。

我感觉这类算法挺有意思（如果不作为题目），你可以查询“排序网络”获得更多信息。



S

Stairs

对题意进行等价转化：

定义函数 $f(x)$ 为将 x 拆分成若干个 1 到 m 的数的和的方案数（考虑顺序）。

将 x 这个数字串分割成若干个数字（允许前导 0），设他们的和为 S，则令 $g(x)$ 为所有情况下的 $f(S)$ 之和。比如 $g(123)=f(1+2+3)+f(1+23)+f(12+3)+f(123)$。

给定 s_0,m，求 $g(s_0)$。

$s_0<10^{601},1\le m\le 5$。

- $n < 10^6$

【1】模拟法 【4】简单一维动态规划

求 $f(x)$ 是一个经典 $O(n)$ dp 问题（上楼梯），暴力枚举拆分方案即可。

$$f(x) = \sum_{i=1}^{m} f(x-i)$$

【6】矩阵的运算 【8】动态规划的常用优化

由于计算 $f(x)$ 是常系数递推，而 $m \leq 5$，启发我们使用矩阵，我们能够得到如下转移矩阵：

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 1 \\ 0 & 1 & 0 & \dots & 0 & 1 \\ 0 & 0 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \end{bmatrix}$$

依然考虑暴力拆分，将计算 $f(x)$ 的过程用矩阵快速幂优化到 $O(\log n)$。

- $m = 1$

【4】简单一维动态规划

注意到此时 $f(x) = 1$，因此只需要计算拆分方案数即可。

设 g_n 为拆分 s_0 的前 n 位的方案数，则

$$g_n = \sum_{i=1}^{n-1} g_i$$

$O(n^2)$ 或 $O(n)$ 计算。

【4】简单一维动态规划 【6】矩阵的运算

若设 $F = [0 \quad 0 \quad \dots \quad 1]$，则可以通过 $F \times G^k$ 计算得到 $f(k)$。

然后怎么计算 g 呢?

Subtask 3 启发我们考虑 dp，设状态 g_i 表示取了 s_0 的前 i 位，可以得到一个关于矩阵的另类递推式：

$$g_i = \sum_{j=1}^{i} g_{i-j} \times G^{\overline{s_0[i-j+1:i]}}$$

最后 g_n 即为答案。

实现时需要预处理出 $G^{a \times 10^b}$，这一部分时间复杂度为 $O(n^2m^3)$，最后的 dp 部分的时间复杂度为 $O(n^2m^3)$，空间复杂度是 $O(nm^2)$ 的。

原题为：Luogu P3176。

String

【1】模拟法

- $n \leq 1000$

$O(n^2)$ 暴力；大家应该都会。

【5】位集合（bitset）

典型的 bitset 乱搞字符串匹配。

首先，我们使用 26 个 bitset 存储每种字符出现在原串中的位置，具体来说，字符串 S 的第 i 个位置为 t，当且仅当 $c_t[i]$ 为 true。

```
const size_t N = 1e5 + 3;
bitset<N> c[26], ans;
// ... ...
for (size_t i = 0; i < s.size(); ++i)
  c[s[i] - 'a'].set(i);
```

那么，对于 S 的单点修改是容易维护的。

```
size_t pos; char ch;
cin >> pos >> ch, --pos;
c[s[pos] - 'a'].reset(pos);
c[(s[pos] = ch) - 'a'].set(pos);
```

考虑如何利用以上信息匹配。

先考虑如何全局查找子串 t。

考虑将每个 t 出现的位置记录在另一个 `bitset` 中。

- 首先，我们统计出所有 $t[0]$ 出现的位置，即为 $c_{t[0]}$；
- 接着，我们统计出所有 $t[1]$ 出现的位置，即为 $c_{t[1]}$；
- 接着，我们统计出所有 $t[2]$ 出现的位置，即为 $c_{t[2]}$；
- …… ……
- 我们发现要满足 $t[0]$ 出现后紧接着 $t[1]$ 出现；$t[1]$ 出现后紧接着 $t[2]$ 出现 ……
- 因此只需要检查 $\left(c_{t[0]}\right) \operatorname{and} \left(c_{t[1]} \operatorname{lshift} 1\right) \operatorname{and} \left(c_{t[2]} \operatorname{lshift} 2\right) \ldots$ 中存在几个 1。

扩展到区间是容易的，只需要通过左移和右移将区间外的数据丢掉即可。

使用 `count` 函数快速获得 1 的个数。

这里 ans 数组的含义与上文例子中有差异，请自行区分。

```
size_t l, r; string t;
cin >> l >> r >> t, --l;
size_t m = t.size();
ans.set();
for (size_t i = 0; i < m; ++i)
  ans &= c[t[i] - 'a'] << (m - i - 1);
size_t ansr = (ans >> (l + m - 1)).count();
size_t ansl = (ans >> r).count();
if (ansr >= ansl)
  cout << ansr - ansl << '\n';
else
  cout << 0 << '\n';
```

原题：Codeforces 914F。

原题正解是分块 SAM，但是本题不一定能过。

U

Substring

【1】程序设计语言以及程序编译和运行的基本概念

希望有人注意到了这个测试点。

某人在山东夏令营中因某个 1 分的 subtask，达到了挂分的最小正值。大家引以为戒。

4 操作是唯一的输出操作，也就是子任务 4 没有输出操作。提交任何不输出任何东西且能正常结束运行的代码可以获得 1 分。

【1】枚举法 【2】位运算

这个点为朴素暴力准备。

对于每次操作 4，使用二进制枚举每一位是否出现在序列当中，得到 S 的所有子序列，然后插入到 set 中去除重复内容。最后输出该 set 的大小。

时间复杂度为 $O(2^n m \log n)$。

【4】简单一维动态规划

考虑一个单次询问复杂度为 $O(n)$ 的做法。

U.3.1 Algo 1

考虑令 p_c 表示字符 c 上一次出现的位置，f_i 是前 i 个数构成的本质不同子序列数量。

若 s_i 未出现过，则有

$$f_i = 2f_{i-1} + 1$$

第一个 f_{i-1} 表示前 $i-1$ 个数的不同子序列数，第二个 f_{i-1} 是将这些不同子序列后面都加一个 s_i。

若 s_i 出现过，则有重复统计的部分：

$$f_i = 2f_{i-1} - f_{p_{s_i}} - 1$$

U.3.2 Algo 2

或者，还有另一种思路。

设 $f_{c,i}$ 表示前 i 个字符，以 c 结尾的本质不同子序列数量。

先说明一个很显然的性质：若 s_i 恰好为 c，则计数 c 结尾的本质不同子序列数量时，不妨钦定 s_i 为结尾。

然后可以看到一个比较显然的递推：

$$f_{c,i} = \begin{cases} f_{c,i-1} & \text{if } c \neq s_i \\ 1 + \sum_t f_{t,i-1} & \text{if } c = s_i \end{cases}$$

式中 t 枚举字符集中所有字符，本题中取值范围为 $\{0, 1\}$。

dp 的不重由 $f_{t,i-1}$ 的定义保证；dp 的不漏由上述性质保证。

【4】动态规划的基本思路 【6】矩阵的初等变换 【6】线段树 【6】矩阵的运算

考虑使用矩阵加速递推。字符集为 $\{0,1\}$，设状态为三维向量

$$M_i = \begin{bmatrix} f_{0,i} \\ f_{1,i} \\ 1 \end{bmatrix}$$

则 dp 转移变为：

$$M_i = \begin{cases} M_{i-1} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \text{if } s_i = 0 \\ M_{i-1} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} & \text{if } s_i = 1 \end{cases}$$

因此可以线段树维护矩阵乘法的区间查询。这个技巧叫**动态 DP**，简称 **DDP**。

【4】动态规划的基本思路 【4】倍增法 【6】线段树 【6】矩阵的运算

考虑如何区间赋值。

显然本质就是区间赋一个转移矩阵。

你会说，这很简单，只要写一个矩阵快速幂就行了。

对的，但是只是我们忘卡矩阵快速幂了。

因为注意到一个长为 n 的各个字符都相同的序列，它的不同子序列个数只有 n 个。因此可以直接为矩阵赋值（其实没有思考含量）。

U.6.1 Algo1

【4】动态规划的基本思路 【6】矩阵的初等变换 【6】线段树 【6】矩阵的运算

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

注意到，我们的两个转移矩阵可以通过交换 $1,2$ 两行和 $1,2$ 两列相互转化。

因此区间矩阵乘法的结果也可以通过交换 $1,2$ 两行和 $1,2$ 两列实现区间反转。

这个性质可以扩展，具体来说，若你定义一个对矩阵的操作 $\Gamma(A)$ 是由**初等行变换**和**初始列变换**组合而来的，且满足对于单位矩阵 I，有 $\Gamma(I)=I$，则

$$\Gamma(A)\cdot\Gamma(B)=\Gamma(A\cdot B)$$

证明思路是初等行变换可以通过左乘变换矩阵实现，初等列变换可以通过右乘变换矩阵实现。剩下内容留给读者。

U.6.2 Algo 2

【4】动态规划的基本思路 【6】线段树 【6】矩阵的运算

或者，还有个简单的方法。

在线段树的一个节点上同时维护正常的序列和反转后的序列所对应的矩阵。

区间反转序列时，给线段树的对应节点打上 tag，同时交换维护的两个矩阵。需要下传标记时把子节点的两个矩阵也交换掉。

【4】动态规划的基本思路 【4】倍增法 【6】线段树 【6】矩阵的运算

最后，正解就是把上面两部分拼起来。

线段树标记分别用 $0, 1, 2, 3$ 表示未操作、反转区间、区间赋值为 0，区间赋值为 1。

区间赋值直接覆盖标记，区间反转相当于异或 1。

V

Triangle

这是一道签到题。

结论：不存在等边三角形，其三个顶点都在格点上。

```
print("NO\n" * int(input()), end='')
```

V.1.1 Proof 1

【1】代数（初中部分） 【1】几何（初中部分）

众所周知，在平面直角坐标系中，三角形有面积公式：

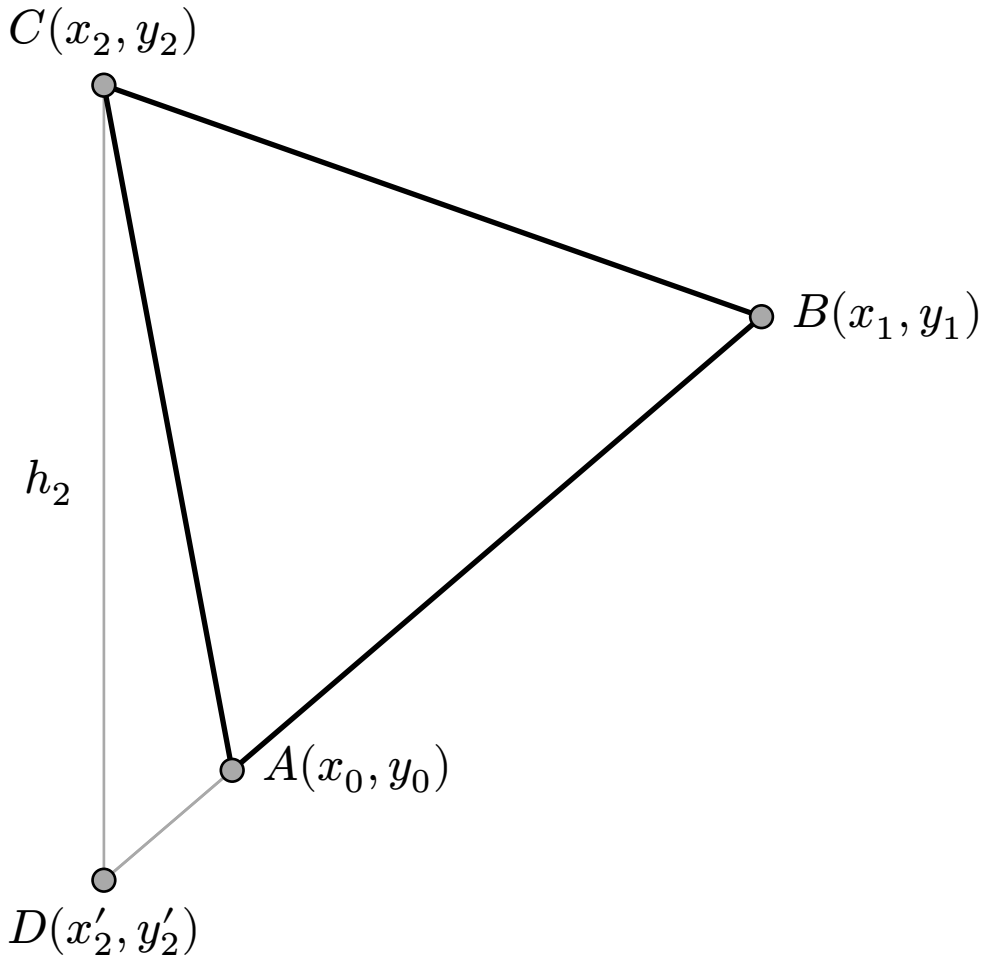
$$S = \frac{1}{2}|x_0 - x_1|h_2$$

由于 A, B, C 均为整点，D 点必为有理点，故 S 必为有理数。

又因为 a^2 必为整数，而

$$S = \frac{\sqrt{3}}{4}a^2$$

矛盾。



V.1.2 Proof 2

【5】代数（高中部分） 【6】几何（高中部分）

以等边三角形的一个顶点 A 为原点，沿格点建平面直角坐标系，将其视为复平面。设三角形的第二个顶点 $B = x + y\mathrm{i}$。由题意得 x, y 均为整数。

第三个顶点 C 为 B 绕 A 顺时针或逆时针旋转 $60°$ 后产生。以逆时针为例，顺时针同理。

考虑复数乘法的几何意义。在复平面绕原点逆时针旋转 $60°$ 相当于将复数乘 $\cos 60° + \mathrm{i} \sin 60°$，因此有：

$$
\begin{aligned}
&(x + y\mathrm{i}) \times (\cos 60° + \mathrm{i} \sin 60°) \\
&= (x + y\mathrm{i}) \times \left(\frac{1}{2} + \frac{\sqrt{3}}{2}\mathrm{i} \right) \\
&= \left(\frac{1}{2}x - \frac{\sqrt{3}}{2}y \right) + \left(\frac{\sqrt{3}}{2}x + \frac{1}{2}y \right)\mathrm{i}
\end{aligned}
$$

由于 x 和 y 均为整数，实部和虚部不可能为整数。

Q.E.D

原题来自 可达鸭编程杯，不知道是不是公开题。

W

Wythoff

Algo 1

【4】动态规划的基本思路

有一个非常基础的结论，如果一个状态可以经一步操作到达必败态，则这个状态是必胜态，否则是必败态。

因此我们最初将 $(0,0,0)$ 标记为必败态，将 $(0,0,i),(0,i,i),(i,i,i),i=1,2...N$ 均标记为必胜态。

接下来，我们遍历每一个状态，若其已经被标为必胜态，跳过，不然其为必败态，累加答案，将其能转移到的位置都标记为必胜态。

这里的总时间复杂度是 $O(N^3)$ 的，因为只有 $O(N^2)$ 个必败态。

空间复杂度 $O(N^3)$。

不能叫算法。

注意到输入只有 1500 种，考虑打表；据统计，似乎跑一个中午即可计算完毕。

但是我们的题目限制代码长度了啊。

根据反馈，对答案序列做两次差分后，写成代码时最短，这大概可以优化到 20-40 pts。

但是注意到可见的 ASCII 码字符共有 $126-32+1=95$ 个（126 对应的字符为波浪号 ~，32 对应的字符为空格），去除反斜杠 \ （92）和双引号 " （34），剩下 93 个字符可以用来编码 93 进制数。

经过反复尝试，我们认为难以通过乱搞将值域大小降到 2^{17} 以下，这样理论最优为 $1500\log_{93}2^{17}\approx 3900$ 个字节。

考虑到做到这种地步时，解压代码的占用应不能忽略，我们最终限制代码长度为 4096 个字节；有人用这个方法做到了 85 分。

Algo 2

【8】动态规划的常用优化

本题考查了空间复杂度的优化。

注意到 **Algo 1** 的标记过程可以表为 $(x+i, y+i, z+i), (x, y+i, z+i), \ldots$ 等形式，这些标记显然可以压缩到二维。以上述两个形式为例，他们分别可以被表为 $(y-x, z-x), (x, z-y)$（请注意题目已经钦定 $x \le y \le z$）。

因此空间复杂度优化到 $O(N^2)$，时间复杂度仍为 $O(N^3)$，但是时间跑不满，利用每个 (x, y) 最多对应一个 z 的性质可以进行剪枝。

上述两个算法的标记数组均需使用 `bitset` 进一步压缩空间。