NOIP 杂项选讲

炼石计划

2023 年 11 月 12 日

前言

对于信息学竞赛中的一些题目，其并不能直接被归类于某一类算法，包括随机化、组合构造或其他被称为乱搞的方法。

前言

对于信息学竞赛中的一些题目，其并不能直接被归类于某一类算法，包括随机化、组合构造或其他被称为乱搞的方法。

对于很多同学来说，这一类题目往往看上去很难想到，但其解法和一些核心的思路、tricks 并非无迹可寻。

前言

对于信息学竞赛中的一些题目，其并不能直接被归类于某一类算法，包括随机化、组合构造或其他被称为乱搞的方法。

对于很多同学来说，这一类题目往往看上去很难想到，但其解法和一些核心的思路、tricks 并非无迹可寻。

对于另一些题目来说，也许其正解较难，但其可以通过随机化等乱搞方法获得较高的部分分甚至满分的分数。

前言

对于信息学竞赛中的一些题目，其并不能直接被归类于某一类算法，包括随机化、组合构造或其他被称为乱搞的方法。

对于很多同学来说，这一类题目往往看上去很难想到，但其解法和一些核心的思路、tricks 并非无迹可寻。

对于另一些题目来说，也许其正解较难，但其可以通过随机化等乱搞方法获得较高的部分分甚至满分的分数。

在本 ppt 中，我们将主要借助一些题目逐渐探索、了解这一类方法，并努力掌握里面的一些核心思路。

目录

随机化算法

随机化算法与确定性算法相对，通常并不一定保证方法的正确性（包括：算法正确性，时间空间的正确性），而是只需在极大概率下方法能够正确运行。

换一句话来说，这些算法的正确性与时空复杂度通常依赖于某些随机事件发生的概率很小且这类随机事件无法被特殊构造这一前提。

随机化算法

随机化算法与确定性算法相对，通常并不一定保证方法的正确性（包括：算法正确性，时间空间的正确性），而是只需在极大概率下方法能够正确运行。

换一句话来说，这些算法的正确性与时空复杂度通常依赖于某些随机事件发生的概率很小且这类随机事件无法被特殊构造这一前提。

随机化算法通常依赖于概率和随机数，因此在介绍具体的随机化算法前，我们先简单介绍概率和随机数的知识。

概率

概率反映了随机事件出现的可能性大小，对于某个事件 A，我们将其概率称为 $P(A)$，$P(A)$ 的大小一般在 $[0,1]$ 之间。

当 $P(A)=0$ 时，我们称事件 A 为不可能事件。

当 $P(A)=1$ 时，我们称事件 A 为必然事件。

对于 $P(A)\neq 0$ 或 1 的情况，我们称事件 A 为可能事件，这一类事件通常可能发生，可能不发生；$P(A)$ 的大小越接近 1，事件 A 就越可能发生。

当用 A 代表我们使用的随机化算法时，我们的目标即是让 $P(A)$ 尽可能大，从而代表着我们正确的概率越高。

概率和统计上的频率有着对应的关系，这也是我们使用随机化算法的基础。

随机数生成

随机化算法中另一个重要的前置知识是随机数的产生。在 oi 中常用的与随机相关的函数有以下两个：

rand()

调用 *rand*() 函数会返回一个 $[0, RAND\_MAX]$ 中的随机非负整数，其中 $RAND\_MAX$ 是标准库中的一个宏，在 *Linux* 系统下 $RAND\_MAX$ 等于 $2^{31}-1$。

通常，我们可以用取模来限制所生成的数的大小。如果我们想要生成 $[L, R]$ 中的一个随机数 $(1 \le R-L+1 \le 2^{31})$，我们可以使用 $rand()\%(R-L+1)+L$ 进行生成。

random_shuffle(*first*, *last*)

调用 *rand_shuffle*(*first*, *last*) 函数会将指针 $[first, last)$ 之间的序列随机打乱（注意为左闭右开，与 *sort* 等函数相同）。

例如，如果我们需要将数组 $a[1]$ 到 $a[n]$ 的数重新打乱，语句写为 $random\_shuffle(a+1, a+n+1)$。

一个简单的例子

通过随机撒点估算圆周率（Markov 方法）

考虑下列估计圆周率 π 的精确值的算法：
在正方形区域 $[-1,1]^2$ 内随机生成 n 个点，记其中落入单位圆盘 $x^2+y^2\leq 1$ 的点数为 m，则可以取 $\dfrac{4m}{n}$ 为 π 的近似值。

一个简单的例子

通过随机撒点估算圆周率（Markov 方法）

考虑下列估计圆周率 π 的精确值的算法：
在正方形区域 $[-1,1]^2$ 内随机生成 n 个点，记其中落入单位圆盘 $x^2+y^2\leq 1$ 的点数为 m，则可以取 $\dfrac{4m}{n}$ 为 π 的近似值。

为什么可以这样计算？
概率和频率之间有着天然的对应关系！

一个简单的例子

通过随机撒点估算圆周率（Markov 方法）

考虑下列估计圆周率 π 的精确值的算法：
在正方形区域 $[-1,1]^2$ 内随机生成 n 个点，记其中落入单位圆盘 $x^2+y^2\leq 1$ 的点数为 m，则可以取 $\dfrac{4m}{n}$ 为 π 的近似值。

为什么可以这样计算？

概率和频率之间有着天然的对应关系！

利用这一关系，我们利用成功概率较大的方法，就可以获得较高的正确率，从而可以在题目中获得更高的分数。

一个简单的例子

通过随机撒点估算圆周率（Markov 方法）

考虑下列估计圆周率 π 的精确值的算法：
在正方形区域 $[-1,1]^2$ 内随机生成 n 个点，记其中落入单位圆盘 $x^2+y^2\le 1$ 的点数为 m，则可以取 $\dfrac{4m}{n}$ 为 π 的近似值。

为什么可以这样计算？

概率和频率之间有着天然的对应关系！

利用这一关系，我们利用成功概率较大的方法，就可以获得较高的正确率，从而可以在题目中获得更高的分数。

接下来，我们将通过一些题目来了解随机化的方法和一些技巧。

最大团问题[1]

当 G' 是图 G 的子图，且 G' 是关于 V 的完全图时，称子图 G' 为图 G 的团。

当 G' 是团，且不是其他团的子集时，G' 为图 G 的极大团。

当 G' 是极大团，且点数最多时，称 G' 为图 G 的最大团。

给定一个 n 个点 m 条边的图，求这个图的最大团大小。、

$n, m \leq 1000$

[1]经典问题

最大团问题

最大团从本质上来说是 NP 问题，目前常用搜索的方法。

最大团问题

最大团从本质上来说是 NP 问题，目前常用搜索的方法。

但搜索的剪枝往往并不容易想到，有没有什么更容易的方法呢？

一个简单的思路是一个一个点加入，每加入一个点判断是否与之前在团里的所有点有边，若有边，则加入团中；否则不加入。不断这样重复直至 n 个点都做完。

最大团问题

最大团从本质上来说是 NP 问题，目前常用搜索的方法。

但搜索的剪枝往往并不容易想到，有没有什么更容易的方法呢？

一个简单的思路是一个一个点加入，每加入一个点判断是否与之前在团里的所有点有边，若有边，则加入团中；否则不加入。不断这样重复直至 n 个点都做完。

观察到团的要求实际上非常苛刻，最大团的大小也较小。

最大团问题

最大团从本质上来说是 NP 问题，目前常用搜索的方法。

但搜索的剪枝往往并不容易想到，有没有什么更容易的方法呢？

一个简单的思路是一个一个点加入，每加入一个点判断是否与之前在团里的所有点有边，若有边，则加入团中；否则不加入。不断这样重复直至 n 个点都做完。

观察到团的要求实际上非常苛刻，最大团的大小也较小。

考虑对于最大团内的点而言，其加入的顺序并不影响最大团的构造。

最大团问题

最大团从本质上来说是 NP 问题，目前常用搜索的方法。

但搜索的剪枝往往并不容易想到，有没有什么更容易的方法呢？

一个简单的思路是一个一个点加入，每加入一个点判断是否与之前在团里的所有点有边，若有边，则加入团中；否则不加入。不断这样重复直至 n 个点都做完。

观察到团的要求实际上非常苛刻，最大团的大小也较小。

考虑对于最大团内的点而言，其加入的顺序并不影响最大团的构造。

对于最大团外的点而言，其能够满足加入条件的概率较低，

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

取多次计算中大小最大的一次作为答案。

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

取多次计算中大小最大的一次作为答案。

这样的正确性在于：根据我们之前的分析，在一定的概率下，序列的顺序并不会对最大团的构造造成影响。

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

取多次计算中大小最大的一次作为答案。

这样的正确性在于：根据我们之前的分析，在一定的概率下，序列的顺序并不会对最大团的构造造成影响。

假设这个概率为 p, 我们重复计算 k 次，那么最终答案正确的概率为 $1-(1-p)^k$。

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

取多次计算中大小最大的一次作为答案。

这样的正确性在于：根据我们之前的分析，在一定的概率下，序列的顺序并不会对最大团的构造造成影响。

假设这个概率为 p, 我们重复计算 k 次，那么最终答案正确的概率为 $1-(1-p)^k$。

当 $k=10, p=0.3$ 时，这个值就到了 0.97。

最大团问题

我们考虑多次 *random_shuffle* 一个 1 的序列，按照上面的方法按序列顺序加入构造最大团。

取多次计算中大小最大的一次作为答案。

这样的正确性在于：根据我们之前的分析，在一定的概率下，序列的顺序并不会对最大团的构造造成影响。

假设这个概率为 p, 我们重复计算 k 次，那么最终答案正确的概率为 $1-(1-p)^k$。

当 $k=10, p=0.3$ 时，这个值就到了 0.97。事实上，我们可以在时间范围内尽可能多的增加随机的次数，从而可以达到最高的正确率。

[TJOI2015] 线性代数[2]

给定一个 $N \times N$ 的矩阵 B 和一个 $1 \times N$ 的矩阵 C。求 $1 \times N$ 的 01 矩阵 A 中，使得 $D = (A \times B - C) \times A^T$ 最大对应的 D。其中矩阵中的数都是非负整数。

其中 A^T 为 A 的转置。

$N \leq 1000$

[2] luogu P3973

[TJOI2015] 线性代数

考虑题目可以转化为给定 N 个数，在其中选取任意个数。当选取 i 时，会有 $-c[i]$ 的贡献，而对于同时选取两个数 i,j 时，会有 $b[i][j]+b[j][i]$ 的贡献。题意即求这个贡献的最大值。

[TJOI2015] 线性代数

考虑题目可以转化为给定 N 个数，在其中选取任意个数。当选取 i 时，会有 $-c[i]$ 的贡献，而对于同时选取两个数 i,j 时，会有 $b[i][j]+b[j][i]$ 的贡献。题意即求这个贡献的最大值。

如果熟悉网络流的同学很容易发现可以通过网络流建图求解这个最大值，但在 NOIP 的赛场上，从想出一个网络流到写完一个网络流显然不是一个简单的过程。

[TJOI2015] 线性代数

考虑题目可以转化为给定 N 个数，在其中选取任意个数。当选取 i 时，会有 $-c[i]$ 的贡献，而对于同时选取两个数 i,j 时，会有 $b[i][j]+b[j][i]$ 的贡献。题意即求这个贡献的最大值。

如果熟悉网络流的同学很容易发现可以通过网络流建图求解这个最大值，但在 NOIP 的赛场上，从想出一个网络流到写完一个网络流显然不是一个简单的过程。

考虑我们如何使用随机化解决这个问题，一个简单的随机思路是每次随机一个 A，去计算对应的贡献，这样的计算复杂度是 $O(N^2)$ 的。

[TJOI2015] 线性代数

但这样并没有用到题目中的性质，且能随机的次数很少，我们考虑怎样是一个比较好的选择方式。

一个比较直观的想法是选的数目应该较多，因为选的数量越多，所能提供的第二部分的贡献就会越多。

[TJOI2015] 线性代数

但这样并没有用到题目中的性质，且能随机的次数很少，我们考虑怎样是一个比较好的选择方式。

一个比较直观的想法是选的数目应该较多，因为选的数量越多，所能提供的第二部分的贡献就会越多。

那我们就可以从全 1 的 A 矩阵出发，每次随机修改其中一位，不断计算结果。这样能够保证其选择的数量是较多的，理论上来说取得的贡献也会更优。

[TJOI2015] 线性代数

但这样并没有用到题目中的性质，且能随机的次数很少，我们考虑怎样是一个比较好的选择方式。

一个比较直观的想法是选的数目应该较多，因为选的数量越多，所能提供的第二部分的贡献就会越多。

那我们就可以从全 1 的 A 矩阵出发，每次随机修改其中一位，不断计算结果。这样能够保证其选择的数量是较多的，理论上来说取得的贡献也会更优。

更好的是我们发现，这样进行修改，我们每次可以只计算与修改的哪一位相关的贡献，每次重新计算的复杂度编程 $O(n)$ 的了！

[TJOI2015] 线性代数

但这样并没有用到题目中的性质，且能随机的次数很少，我们考虑怎样是一个比较好的选择方式。

一个比较直观的想法是选的数目应该较多，因为选的数量越多，所能提供的第二部分的贡献就会越多。

那我们就可以从全 1 的 A 矩阵出发，每次随机修改其中一位，不断计算结果。这样能够保证其选择的数量是较多的，理论上来说取得的贡献也会更优。

更好的是我们发现，这样进行修改，我们每次可以只计算与修改的哪一位相关的贡献，每次重新计算的复杂度编程 $O(n)$ 的了！

这样我们可以大幅度的提高随机的次数。在实际情况下，1500 次左右可以通过此题。

一个小结

诸如这一类的题目还有很多，使用随机化 + 贪心的方法可以通过很多正解并非随机化的题目。

一个小结

诸如这一类的题目还有很多，使用随机化 + 贪心的方法可以通过很多正解并非随机化的题目。

这些题目往往是一些最优化的题目，可能正解较难（如 [WC2018] 通道），但通过这样的方法能简单地取得高分

就算无法通过正解，随机化往往也能帮助你拿到不错的分数，帮助你获得一些标准暴力之外的分。

一个小结

诸如这一类的题目还有很多，使用随机化 + 贪心的方法可以通过很多正解并非随机化的题目。

这些题目往往是一些最优化的题目，可能正解较难（如 [WC2018] 通道），但通过这样的方法能简单地取得高分

就算无法通过正解，随机化往往也能帮助你拿到不错的分数，帮助你获得一些标准暴力之外的分。

但有一些题目正解本身就是随机化，接下来介绍一些以随机化思路作为正解的题目。

[NOIP2014 提高组] 解方程 [3]

已知多项式方程：

$$a_0 + a_1 x + a_2 x^2 + ... + a_n x^n = 0$$

求这个方程在 $[1, m]$ 内的整数解（n 和 m 均为正整数）。
数据范围：$0 < n \leq 100, |a_i| \leq 10^{10000}, a_n \neq 0, m \leq 1000000$

[3] Luogu P2312

[NOIP2014 提高组] 解方程

观察到数据范围非常大，直接对 $[1, m]$ 里的每个数计算显然行不通

[NOIP2014 提高组] 解方程

观察到数据范围非常大，直接对 $[1, m]$ 里的每个数计算显然行不通
考虑令：

$$f[x] = \sum_{i=0}^{n} a_i * x^i$$

即我们需要求 $f[x] = 0$ 对应的解。

[NOIP2014 提高组] 解方程

观察到数据范围非常大，直接对 $[1, m]$ 里的每个数计算显然行不通
考虑令：

$$f[x] = \sum_{i=0}^{n} a_i * x^i$$

即我们需要求 $f[x] = 0$ 对应的解。
而当 $f[x] = 0$ 时有对于任意 p，$f[x]\ mod\ p \equiv 0$

[NOIP2014 提高组] 解方程

观察到数据范围非常大，直接对 $[1, m]$ 里的每个数计算显然行不通
考虑令：

$$f[x] = \sum_{i=0}^{n} a_i * x^i$$

即我们需要求 $f[x] = 0$ 对应的解。
而当 $f[x] = 0$ 时有对于任意 p，$f[x]\ mod\ p \equiv 0$
因此我们可以考虑随机多个模数 p，计算数 x 对应的 $f[x]$ 在 $mod\ p$ 意义下是否为 0。

[NOIP2014 提高组] 解方程

观察到数据范围非常大，直接对 $[1, m]$ 里的每个数计算显然行不通

考虑令：

$$f[x] = \sum_{i=0}^{n} a_i * x^i$$

即我们需要求 $f[x] = 0$ 对应的解。

而当 $f[x] = 0$ 时有对于任意 p，$f[x]\ mod\ p \equiv 0$

因此我们可以考虑随机多个模数 p，计算数 x 对应的 $f[x]$ 在 $mod\ p$ 意义下是否为 0。

如果均为 0，则可视为其即为对应的解。

最小圆覆盖 [4]

在一个平面上有 n 个点，求一个半径最小的圆，能覆盖所有的点。
$n \leq 10^5$

[4] Luogu P1742

最小圆覆盖

性质 1

最小圆覆盖具有唯一性.
若不唯一，假设有两个最小圆覆盖，则所有点都在对应的交集内。取交集的弦作为新圆的直径即可获得一个更小的圆覆盖。与最小圆覆盖矛盾

最小圆覆盖

性质 2

若圆 O 是点集 S 的最小圆覆盖，则再新加一点 p。若其在圆 O 的外部，则新点集的最小覆盖圆一定过点 p，即 p 一定在最小圆覆盖上。
若不经过点 p，则一定可以取一个更小的经过点 p 的圆。

最小圆覆盖

性质 3

最终的最小圆覆盖一定只有两种情况：
要么为圆上至少有三个点，则三个点不共线的确定了这个圆。
要么为圆上只有两个点，则该圆以两点的连线作为直径。
若不然，则一定可以找到一个更小的圆（对于一个点或不以两点连线为直径的情况）

最小圆覆盖

算法过程：

假设圆 O 是前 $i-1$ 个点的最小覆盖圆，加入第 i 个点，如果在已经包括在圆内则什么也不做，否则新得到的最小覆盖圆肯定经过第 i 个点。

最小圆覆盖

算法过程：

假设圆 O 是前 $i-1$ 个点的最小覆盖圆，加入第 i 个点，如果在已经包括在圆内则什么也不做，否则新得到的最小覆盖圆肯定经过第 i 个点。

我们知道三个点可以确定一个圆，因此我们在前 $i-1$ 个点中继续寻找剩下的两个点，首先我们以第 i 个点为基础（半径为 0），依次加入第 j 个点，若第 j 个点在圆外，则最小覆盖圆必经过第 j 个点。

最小圆覆盖

算法过程：

假设圆 O 是前 $i-1$ 个点的最小覆盖圆，加入第 i 个点，如果在已经包括在圆内则什么也不做，否则新得到的最小覆盖圆肯定经过第 i 个点。

我们知道三个点可以确定一个圆，因此我们在前 $i-1$ 个点中继续寻找剩下的两个点，首先我们以第 i 个点为基础（半径为 0），依次加入第 j 个点，若第 j 个点在圆外，则最小覆盖圆必经过第 j 个点。

我们再按照上面的步骤，以 i，j 作为直径，寻找第三个点 k，最终确定前 i 个点的最小圆覆盖。

最小圆覆盖

算法过程：

假设圆 O 是前 $i-1$ 个点的最小覆盖圆，加入第 i 个点，如果在已经包括在圆内则什么也不做，否则新得到的最小覆盖圆肯定经过第 i 个点。

我们知道三个点可以确定一个圆，因此我们在前 $i-1$ 个点中继续寻找剩下的两个点，首先我们以第 i 个点为基础（半径为 0），依次加入第 j 个点，若第 j 个点在圆外，则最小覆盖圆必经过第 j 个点。

我们再按照上面的步骤，以 i，j 作为直径，寻找第三个点 k，最终确定前 i 个点的最小圆覆盖。

遍历完所有点之后，所得到的圆就是覆盖所有点的最小圆覆盖。

最小圆覆盖

这个算法看上去是 $O(n^3)$，实际上当我们考虑概率时可以发现每一次进入内层循环的条件为上一层循环为之前最小圆覆盖圆外的点，也即确定新的最小圆覆盖的三个点之一。

最小圆覆盖

这个算法看上去是 $O(n^3)$，实际上当我们考虑概率时可以发现每一次进入内层循环的条件为上一层循环为之前最小圆覆盖圆外的点，也即确定新的最小圆覆盖的三个点之一。

这样的概率为 $\frac{3}{n}$，因此最终的复杂度为：

$$O(n) \times (O(1) + \frac{3}{n} \times O(n) \times (O(1) + \frac{3}{n} \times O(n))) = O(n)$$

最小圆覆盖

这个算法看上去是 $O(n^3)$，实际上当我们考虑概率时可以发现每一次进入内层循环的条件为上一层循环为之前最小圆覆盖圆外的点，也即确定新的最小圆覆盖的三个点之一。

这样的概率为 $\frac{3}{n}$，因此最终的复杂度为：

$$O(n) \times (O(1) + \frac{3}{n} \times O(n) \times (O(1) + \frac{3}{n} \times O(n))) = O(n)$$

在输入时需要随机化所有点 $(random\_shuffle)$，这样可以防止特殊构造数据导致复杂度错误。

最小圆覆盖

这个算法看上去是 $O(n^3)$，实际上当我们考虑概率时可以发现每一次进入内层循环的条件为上一层循环为之前最小圆覆盖圆外的点，也即确定新的最小圆覆盖的三个点之一。

这样的概率为 $\frac{3}{n}$，因此最终的复杂度为：

$$O(n) \times (O(1) + \frac{3}{n} \times O(n) \times (O(1) + \frac{3}{n} \times O(n))) = O(n)$$

在输入时需要随机化所有点 $(random\_shuffle)$，这样可以防止特殊构造数据导致复杂度错误。

随机化是一个好习惯！

Tourism [5]

给定一个 n 个顶点的无向带权完全图，求从点 1 出发，经过 k 条边后回到点 1，且不经过奇环的最短花费。(可以经过重复的点和边)

其中奇环指对于某个顶点 v，经过 v 后，走了奇数条边又回到 v。

$n \leq 80, k \leq 10$

[5] Codeforces 1310D

Tourism

考虑不存在奇环的图实际上为二分图，而二分图可以黑白染色。

Tourism

考虑不存在奇环的图实际上为二分图，而二分图可以黑白染色。

我们考虑将这幅图的 n 个点二分染色，如果染色正确，则我们相当于去除了奇环的条件，很容易通过一个 dp 求解。

Tourism

考虑不存在奇环的图实际上为二分图，而二分图可以黑白染色。

我们考虑将这幅图的 n 个点二分染色，如果染色正确，则我们相当于去除了奇环的条件，很容易通过一个 dp 求解。

而如果我们随机二分染色，对于正确答案对应的路恰好被染成黑白交替颜色的概率为 $p = \frac{1}{2^{k-1}}$，最小为 $\frac{1}{512}$。

Tourism

考虑不存在奇环的图实际上为二分图，而二分图可以黑白染色。

我们考虑将这幅图的 n 个点二分染色，如果染色正确，则我们相当于去除了奇环的条件，很容易通过一个 dp 求解。

而如果我们随机二分染色，对于正确答案对应的路恰好被染成黑白交替颜色的概率为 $p = \frac{1}{2^{k-1}}$，最小为 $\frac{1}{512}$。

我们只需要取足够多的次数 T，即可使 $1 - (1-p)^T$ 很小。

Tourism

考虑不存在奇环的图实际上为二分图，而二分图可以黑白染色。

我们考虑将这幅图的 n 个点二分染色，如果染色正确，则我们相当于去除了奇环的条件，很容易通过一个 dp 求解。

而如果我们随机二分染色，对于正确答案对应的路恰好被染成黑白交替颜色的概率为 $p = \frac{1}{2^{k-1}}$，最小为 $\frac{1}{512}$。

我们只需要取足够多的次数 T，即可使 $1 - (1-p)^T$ 很小。

在实际做题的过程中，取 $T = 1500$ 即可通过此题。

Andy and Maze [6]

在 n 个点和 m 条边的无向图中，每个点有一个宝石，你要收集 k 个宝石才能逃脱，被访问的点不能再访问第二次，求任选起点可能经过的最大权值是多少？

$n, m \leq 10^4, 2 \leq k \leq 6$

[6] hdu6664

Andy and Maze

和上题类似，我们不方便处理不经过重点的条件。

Andy and Maze

和上题类似，我们不方便处理不经过重点的条件。

我们考虑随机染色，每个点染上 $[1,k]$ 之间的颜色，要求每一步走到颜色恰好多 1 的点上。

Andy and Maze

和上题类似，我们不方便处理不经过重点的条件。

我们考虑随机染色，每个点染上 $[1,k]$ 之间的颜色，要求每一步走到颜色恰好多 1 的点上。

这样求解就可以通过增加一个虚拟源点的最短路解决。

Andy and Maze

和上题类似，我们不方便处理不经过重点的条件。

我们考虑随机染色，每个点染上 $[1,k]$ 之间的颜色，要求每一步走到颜色恰好多 1 的点上。

这样求解就可以通过增加一个虚拟源点的最短路解决。

每次随机染色求解，正确率为 $\frac{k!}{k^k}$，重复多次即可。

Andy and Maze

和上题类似，我们不方便处理不经过重点的条件。

我们考虑随机染色，每个点染上 $[1,k]$ 之间的颜色，要求每一步走到颜色恰好多 1 的点上。

这样求解就可以通过增加一个虚拟源点的最短路解决。

每次随机染色求解，正确率为 $\frac{k!}{k^k}$，重复多次即可。

这道题展现了一个经典的套路，当遇到某个条件不好处理时，可以考虑通过随机染色，构造一个更强的条件解决。

Kuroni and the Punishment [7]

给定 n 个正整数 a_i，每次操作可以选择某一个数 $+1$ 或 -1（需要保证操作后仍为正整数）。

求使得所有数的 gcd 不等于 1 的最小操作次数。

$n \leq 2 \times 10^5, a_i \leq 10^{12}$

[7] Codeforces 1305F

Kuroni and the Punishment

假设我们知道了最终的 gcd，那我们可以 $O(n)$ 扫一遍获得需要操作的次数。

Kuroni and the Punishment

假设我们知道了最终的 gcd，那我们可以 $O(n)$ 扫一遍获得需要操作的次数。

另外，我们也可以发现答案至多为 n。因为当我们取 $gcd = 2$ 时，要改的数的数量就是序列中奇数的数量。

Kuroni and the Punishment

假设我们知道了最终的 gcd，那我们可以 $O(n)$ 扫一遍获得需要操作的次数。

另外，我们也可以发现答案至多为 n。因为当我们取 $gcd = 2$ 时，要改的数的数量就是序列中奇数的数量。

因此，紧接着可以推出操作次数 ≥ 2 的数的数量一定 $\leq \frac{n}{2}$。

Kuroni and the Punishment

假设我们知道了最终的 gcd，那我们可以 $O(n)$ 扫一遍获得需要操作的次数。

另外，我们也可以发现答案至多为 n。因为当我们取 $gcd = 2$ 时，要改的数的数量就是序列中奇数的数量。

因此，紧接着可以推出操作次数 ≥ 2 的数的数量一定 $\leq \frac{n}{2}$。

那么，我们随机一个数 a_i，则 $a_i, a_i + 1, a_i - 1$ 三个数的某个质因子在最终答案的 gcd 内的概率是 $\geq \frac{1}{2}$ 的。

Kuroni and the Punishment

假设我们知道了最终的 gcd，那我们可以 $O(n)$ 扫一遍获得需要操作的次数。

另外，我们也可以发现答案至多为 n。因为当我们取 $gcd = 2$ 时，要改的数的数量就是序列中奇数的数量。

因此，紧接着可以推出操作次数 ≥ 2 的数的数量一定 $\leq \frac{n}{2}$。

那么，我们随机一个数 a_i，则 $a_i, a_i + 1, a_i - 1$ 三个数的某个质因子在最终答案的 gcd 内的概率是 $\geq \frac{1}{2}$ 的。

我们重复选取 a_i，枚举这三个数的质因子，更新答案即可解决。

[BJOI2014] 想法 [8]

小强要出一套题目。他为了出题，一共攒了 M 个本质不同的想法，每个想法形成了一个题目。不过，他觉得拿这些题目去考察选手会把比赛搞的太过变态，所以，想请阿米巴来帮忙调整一下他的题目。

阿米巴指出，为了让一场考试的题目的考察点尽量全面，有一个通用的做法叫做「组合」。如果把两个题目 A 和 B 组合在一起，那么组合而成的题目涉及到的想法的集合就是 A 涉及到的想法的集合和 B 涉及到的想法的集合的并。并且，题目是可以反复组合的。

例如，小强现在有三个想法 $1, 2, 3$，分别对应了题目 P_1, P_2, P_3。

现在，小强把 P_1 和 P_2 组合得到 P_4。P_4 涉及的想法的集合是 $\{1, 2\}$。

之后，小强把 P_2 和 P_3 组合得到 P_5。P_5 涉及的想法的集合是 $\{2, 3\}$。

最后，小强把 P_4 和 P_5 组合得到 P_6。P_6 涉及的想法的集合是 $\{1, 2, 3\}$。‘

现在，小强告诉你每个题目都是如何组合而来的。你要回答的就是，每个题目涉及的想法的集合有多大。

$M \leq 10^5$，$N \leq 10^6$

评分方式：对于每个输出文件，如果其中你有 95% 以上的行的答案和正确答案的误差不超过 25%，那么你就可以得到分数。所谓误差不超过 25%，即，如果正确答案是 X，那么你的答案在 $[0.8X, 1.25X]$ 这个闭区间内。

[8] luogu P4581

[BJOI2014] 想法

对于每个想法 Rand 一个随机数代表它，每次合并的时候保留前 T 小的数，并记录其出现次数。

[BJOI2014] 想法

对于每个想法 Rand 一个随机数代表它，每次合并的时候保留前 T 小的数，并记录其出现次数。

最后算答案时，如果出现种类少于 T 就直接加上去，否则我们估计他的大小为 $\frac{T\times RAND\_MAX}{RAND\_T}$，其中 $RAND\_T$ 是第 T 小的 RAND 出来的数。

[BJOI2014] 想法

对于每个想法 Rand 一个随机数代表它，每次合并的时候保留前 T 小的数，并记录其出现次数。

最后算答案时，如果出现种类少于 T 就直接加上去，否则我们估计他的大小为 $\frac{T\times RAND\_MAX}{RAND\_T}$，其中 $RAND\_T$ 是第 T 小的 RAND 出来的数。

为什么这样是正确的？

[BJOI2014] 想法

对于每个想法 Rand 一个随机数代表它，每次合并的时候保留前 T 小的数，并记录其出现次数。

最后算答案时，如果出现种类少于 T 就直接加上去，否则我们估计他的大小为 $\frac{T\times RAND\_MAX}{RAND\_T}$，其中 $RAND\_T$ 是第 T 小的 RAND 出来的数。

为什么这样是正确的？

设包含的总想法数为 L，在 L 中取到 T 的期望为 $\frac{T}{L}$，而随机数是均匀分布的，所以 $\frac{T}{L}$ 会等于 $\frac{RAND\_T}{RAND\_MAX}$，所以 $\frac{T\times RAND\_MAX}{RAND\_T}$

Problem with Random Tests [9]

给出一个 01 串 s，请你任意选择两个子串 $s1, s2$，输出将他们向右对齐按位或的最大值的二进制表示。

$n \leq 10^6$

保证数据随机生成

[9] Codeforces 1743D

Problem with Random Tests

由于长度越长，01 串就越大，因此两个子串 $s1$，$s2$ 中必定有一个为 s 串本身。

Problem with Random Tests

由于长度越长，01 串就越大，因此两个子串 $s1$，$s2$ 中必定有一个为 s 串本身。

那么考虑去掉 s 串的前导 0，在剩下的串中，考虑贪心。

Problem with Random Tests

由于长度越长，01 串就越大，因此两个子串 $s1$，$s2$ 中必定有一个为 s 串本身。

那么考虑去掉 s 串的前导 0，在剩下的串中，考虑贪心。

我们考虑如何能把从左到右第一个 0 变为 1，当且仅当选择的串长度大于从这个 0 到串末尾的长度。

Problem with Random Tests

由于长度越长，01 串就越大，因此两个子串 $s1$，$s2$ 中必定有一个为 s 串本身。

那么考虑去掉 s 串的前导 0，在剩下的串中，考虑贪心。

我们考虑如何能把从左到右第一个 0 变为 1，当且仅当选择的串长度大于从这个 0 到串末尾的长度。

满足这样条件的子串左端点一定在第一个 0 的左边，考虑枚举第一个 0 对应的在 $s2$ 中的位置，暴力进行判断即可

Problem with Random Tests

由于长度越长，01 串就越大，因此两个子串 $s1$，$s2$ 中必定有一个为 s 串本身。

那么考虑去掉 s 串的前导 0，在剩下的串中，考虑贪心。

我们考虑如何能把从左到右第一个 0 变为 1，当且仅当选择的串长度大于从这个 0 到串末尾的长度。

满足这样条件的子串左端点一定在第一个 0 的左边，考虑枚举第一个 0 对应的在 $s2$ 中的位置，暴力进行判断即可

为什么复杂度正确？数据随机，不会存在长串连续的 1。

Problem with Random Tests

数据随机生成有什么性质？

树–深度几乎是 $O(\log)$ 级别

图–每个点的度数较为平均

其他序列–序列分布平均，不会出现长期对某一段的操作，不会出现很长一段相同

对于数据随机需要敏感起来！这也是题目提供的条件之一。

爬山算法

如果有一个盲人拄着拐杖想要爬到山顶 (或回到山脚)，他只能通过拐杖去探测当前他所在的位置的局部信息，他该如何做？

爬山算法

如果有一个盲人拄着拐杖想要爬到山顶 (或回到山脚)，他只能通过拐杖去探测当前他所在的位置的局部信息，他该如何做？

盲人的做法：不断探测周围的信息，从而决定当前最优的行走方向，循环往复，直到达到他的目的地。

爬山算法

如果有一个盲人拄着拐杖想要爬到山顶 (或回到山脚)，他只能通过拐杖去探测当前他所在的位置的局部信息，他该如何做？

盲人的做法：不断探测周围的信息，从而决定当前最优的行走方向，循环往复，直到达到他的目的地。

根据这种思路，我们出现了爬山算法。

爬山算法

爬山算法 [10]

爬山算法是一种局部择优的方法，采用启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。
直白地讲，就是当目前无法直接到达最优解，但是可以判断两个解哪个更优的时候，根据一些反馈信息生成一个新的可能解。
因此，爬山算法每次在当前找到的最优方案 x 附近寻找一个新方案。如果这个新的解 x' 更优，那么转移到 x'，否则不变。

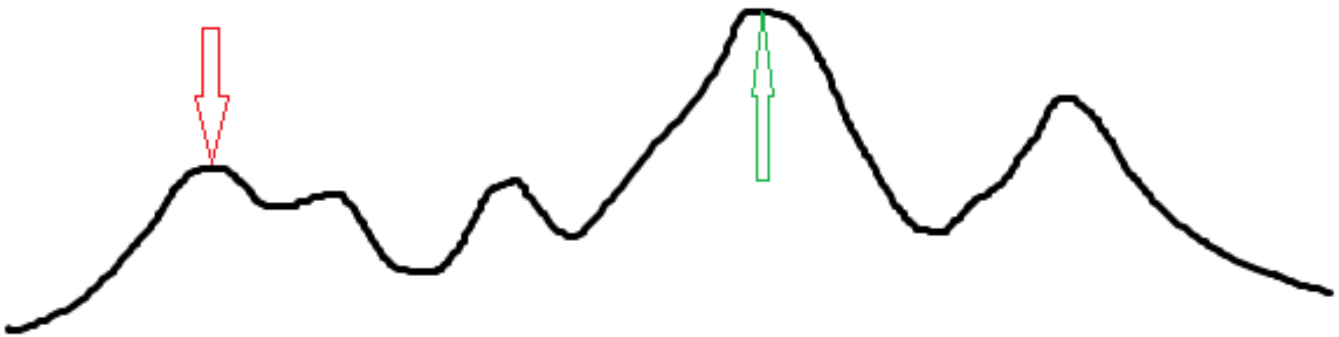
[10] from OI Wiki

爬山算法

爬山算法 [10]

爬山算法是一种局部择优的方法，采用启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。
直白地讲，就是当目前无法直接到达最优解，但是可以判断两个解哪个更优的时候，根据一些反馈信息生成一个新的可能解。
因此，爬山算法每次在当前找到的最优方案 x 附近寻找一个新方案。如果这个新的解 x' 更优，那么转移到 x'，否则不变。

爬山算法容易陷入局部最优解:



[10] from OI Wiki

退火算法

爬山容易陷入局部最优解，如何解决？
多次随机选择点出发，寻找最优解。

退火算法

爬山容易陷入局部最优解，如何解决？
多次随机选择点出发，寻找最优解。
——仍然有可能陷入局部最优解，且需要手动选择调整，不通用

退火算法

爬山容易陷入局部最优解，如何解决？
多次随机选择点出发，寻找最优解。
——仍然有可能陷入局部最优解，且需要手动选择调整，不通用
能否在迭代过程中以一定概率随机跳出局部最优解？

退火算法

爬山容易陷入局部最优解，如何解决？
多次随机选择点出发，寻找最优解。
—-仍然有可能陷入局部最优解，且需要手动选择调整，不通用
能否在迭代过程中以一定概率随机跳出局部最优解？
—-于是产生了模拟退火算法

退火算法

退火算法

退火算法优化了爬山算法可能陷入局部最优解的情况。
简而言之，退火的思路在于：随机选择周边的新状态，如果新状态的解更优则修改答案，否则以一定概率接受新状态。

退火算法

退火算法

退火算法优化了爬山算法可能陷入局部最优解的情况。
简而言之，退火的思路在于：随机选择周边的新状态，如果新状态的解更优则修改答案，否则以一定概率接受新状态。

我们定义当前温度为 T，新状态 S' 与已知状态 S（新状态由已知状态通过随机的方式得到）之间的能量（值）差为 ΔE（$\Delta E \geqslant 0$），则发生状态转移（修改最优解）的概率为

$$P(\Delta E) = \begin{cases} 1, & S' \text{ is better than } S, \\ \mathrm{e}^{\frac{-\Delta E}{T}}, & \text{otherwise.} \end{cases}$$

注意：我们有时为了使得到的解更有质量，会在模拟退火结束后，以当前温度在得到的解附近多次随机状态，尝试得到更优的解（其过程与模拟退火相似）。

退火算法

如何降温

模拟退火时我们有三个参数：初始温度 T_0，降温系数 d，终止温度 T_k。其中 T_0 是一个比较大的数，d 是一个非常接近 1 但是小于 1 的数，T_k 是一个接近 0 的正数。

首先让温度 $T = T_0$，然后按照上述步骤进行一次转移尝试，再让 $T = d \cdot T$。当 $T < T_k$ 时模拟退火过程结束，当前最优解即为最终的最优解。

注意为了使得解更为精确，我们通常不直接取当前解作为答案，而是在退火过程中维护遇到的所有解的最优值。

退火算法

如何降温

模拟退火时我们有三个参数：初始温度 T_0，降温系数 d，终止温度 T_k。其中 T_0 是一个比较大的数，d 是一个非常接近 1 但是小于 1 的数，T_k 是一个接近 0 的正数。
首先让温度 $T = T_0$，然后按照上述步骤进行一次转移尝试，再让 $T = d \cdot T$。
当 $T < T_k$ 时模拟退火过程结束，当前最优解即为最终的最优解。
注意为了使得解更为精确，我们通常不直接取当前解作为答案，而是在退火过程中维护遇到的所有解的最优值。

在模拟退火和爬山之外还有遗传算法，但由于在 OI 中使用并不多，大家可以在课下自行学习

均分数据[11]

有 n 个正整数 $a_1, \ldots, a_n$，将他们分为 m 组，使得每组数值和均方差最小. 即

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}, \overline{x} = \frac{\sum_{i}^{n} x_i}{n}$$

σ 为均方差,$\overline{x}$ 为各组数据和平均值, x_i 为第 i 组数据的数值和。

$m \leq n \leq 20, 2 \leq m \leq 6$

[11] HAOI2006

均分数据

直接使用模拟退火。

首先随机每个数在哪个组里。

每次随机选择一个数，然后把他拿出来，然后贪心地放入当前权值最小的那一组中去。

不断退火可以得到最终的答案。（基本 100 次左右就能推出最优解）

均分数据

直接使用模拟退火。

首先随机每个数在哪个组里。

每次随机选择一个数，然后把他拿出来，然后贪心地放入当前权值最小的那一组中去。

不断退火可以得到最终的答案。(基本 100 次左右就能推出最优解)

模拟退火也需要使用一些贪心或其他的方法，尝试尽量优秀的解。

Space Rescuers[12]

给定三维空间的一些点，求出一个点使得该点到其它点的最大距离最小。
$n \leq 100$

[12] Codeforces 106E

Space Rescuers[13]

观察到其确定某两维后，另一维为单峰函数。

[13] Codeforces 106E

Space Rescuers[13]

观察到其确定某两维后，另一维为单峰函数。
当然可以使用三分套三分，但为什么不使用更加简单的退火呢？

[13] Codeforces 106E

Space Rescuers[13]

观察到其确定某两维后，另一维为单峰函数。
当然可以使用三分套三分，但为什么不使用更加简单的退火呢？
随机选择初始点，使用退火寻找最优解。

[13] Codeforces 106E

Space Rescuers[13]

观察到其确定某两维后，另一维为单峰函数。

当然可以使用三分套三分，但为什么不使用更加简单的退火呢？

随机选择初始点，使用退火寻找最优解。

当遇到存在一定单调性，但并不方便利用其单调性时，可以使用退火进行处理。

[13] Codeforces 106E

组合构造

构造题是 OI 比赛中一种常见的题型，但其比较特别的是，其并不能被总结为某一种知识点，相比较而言，其更像是“人类智慧题”。

组合构造

构造题是 OI 比赛中一种常见的题型，但其比较特别的是，其并不能被总结为某一种知识点，相比较而言，其更像是“人类智慧题”。

一般而言，构造题常常需要选手较为敏锐的洞察力、对题目较为详细的分析和对数论、图论中的一些知识较为深刻地理解。

组合构造

构造题是 OI 比赛中一种常见的题型，但其比较特别的是，其并不能被总结为某一种知识点，相比较而言，其更像是“人类智慧题”。

一般而言，构造题常常需要选手较为敏锐的洞察力、对题目较为详细的分析和对数论、图论中的一些知识较为深刻地理解。

构造题也需要选手大胆地做出假设和猜想。

组合构造

构造题是 OI 比赛中一种常见的题型，但其比较特别的是，其并不能被总结为某一种知识点，相比较而言，其更像是“人类智慧题”。

一般而言，构造题常常需要选手较为敏锐的洞察力、对题目较为详细的分析和对数论、图论中的一些知识较为深刻地理解。

构造题也需要选手大胆地做出假设和猜想。

当然，构造题也存在一些常见的构造思路，我们将在以下的例题中进行展示。

平方

请构造一个长度不超过 100000 的序列 $a_1, \ldots, a_n$，其中恰有 k 对 $1 \leq i < j \leq n$ 满足 $a_i + a_j$ 为平方数，$1 \leq a_i \leq 100000$ $1 \leq k \leq 10^9$。

平方

并没有要求不能重复，实际上往往构造可以使用大量重复的元素

平方

并没有要求不能重复，实际上往往构造可以使用大量重复的元素

在这个题中，我们首先可以选取两对和为平方数数字，如：$5+11=16, 1+3=4$，然后构造一个由 $5, 11, 1, 3$ 组成的序列。

平方

并没有要求不能重复，实际上往往构造可以使用大量重复的元素

在这个题中，我们首先可以选取两对和为平方数数字，如：$5+11=16, 1+3=4$，然后构造一个由 $5, 11, 1, 3$ 组成的序列。

假设有 a 个 5，b 个 11，c 个 1，d 个 3，就恰好有 $ab+cd$ 对。

平方

并没有要求不能重复，实际上往往构造可以使用大量重复的元素

在这个题中，我们首先可以选取两对和为平方数数字，如：$5+11=16, 1+3=4$，然后构造一个由 $5, 11, 1, 3$ 组成的序列。

假设有 a 个 5，b 个 11，c 个 1，d 个 3，就恰好有 $ab+cd$ 对。

构造这样的 $abcd$ 是很容易的，选取 $a=[sqrt(k)], d=k\ mod\ [sqrt(k)], c=1, b=(k-d)/a$ 即可。

Errich Tac Toe[14]

你有一个 $n \times n$ 的棋盘，上面有一些棋子。你需要使一些 X 变成 O，使一些 O 变成 X，从而使棋盘上没有连续的 3 颗相同的棋子（只考虑横竖，不考虑斜线）。

你改变的棋子数量不能超过总棋子数量的 $\frac{1}{3}$

$n \leq 100, t \leq 100$（t 为数据组数）

[14] Codeforces 1450C2

Errich Tac Toe

抽屉原理

将 n 个物品放入 k 个抽屉，则其中有一个抽屉至少包含大于等于 $\frac{n}{k}$ 个物品，也至少有一个抽屉包含小于等于 $\frac{n}{k}$ 个物品。

Errich Tac Toe

抽屉原理

将 n 个物品放入 k 个抽屉，则其中有一个抽屉至少包含大于等于 $\frac{n}{k}$ 个物品，也至少有一个抽屉包含小于等于 $\frac{n}{k}$ 个物品。

抽屉原理是常用的进行构造的方法。

在很多构造题中，当出现要求至少（至多）的情况时，可以考虑使用抽屉原理。

Errich Tac Toe

考虑将棋盘上的每个点 (r, c) 按照 $r + c$ 模 3 的余数进行分类，则我们可以发现连续的 3 颗棋子模 3 的余数一定分别为 $0, 1, 2$。

Errich Tac Toe

考虑将棋盘上的每个点 (r, c) 按照 $r + c$ 模 3 的余数进行分类，则我们可以发现连续的 3 颗棋子模 3 的余数一定分别为 $0, 1, 2$。

按照这样一个分类，我们很容易想到，只要我们把两类分别设置为 O 和 X，则一定能满足连续三颗棋子不同的条件。

Errich Tac Toe

考虑将棋盘上的每个点 (r, c) 按照 $r + c$ 模 3 的余数进行分类，则我们可以发现连续的 3 颗棋子模 3 的余数一定分别为 $0, 1, 2$。

按照这样一个分类，我们很容易想到，只要我们把两类分别设置为 O 和 X，则一定能满足连续三颗棋子不同的条件。

如何满足改变棋子数量不超过 $\frac{1}{3}$ 的条件呢，这时候我们就可以考虑抽屉原理了：

Errich Tac Toe

考虑将棋盘上的每个点 (r, c) 按照 $r + c$ 模 3 的余数进行分类，则我们可以发现连续的 3 颗棋子模 3 的余数一定分别为 $0, 1, 2$。

按照这样一个分类，我们很容易想到，只要我们把两类分别设置为 O 和 X，则一定能满足连续三颗棋子不同的条件。

如何满足改变棋子数量不超过 $\frac{1}{3}$ 的条件呢，这时候我们就可以考虑抽屉原理了：

考虑有三种方案：将第 0 类棋子的 X 改成 O，第 1 类棋子的 O 改成 X；将第 1 类棋子的 X 改成 O，第 2 类棋子的 O 改成 X；将第 2 类棋子的 X 改成 O，第 3 类棋子的 O 改成 X。

Errich Tac Toe

考虑将棋盘上的每个点 (r, c) 按照 $r + c$ 模 3 的余数进行分类，则我们可以发现连续的 3 颗棋子模 3 的余数一定分别为 $0, 1, 2$。

按照这样一个分类，我们很容易想到，只要我们把两类分别设置为 O 和 X，则一定能满足连续三颗棋子不同的条件。

如何满足改变棋子数量不超过 $\frac{1}{3}$ 的条件呢，这时候我们就可以考虑抽屉原理了：

考虑有三种方案：将第 0 类棋子的 X 改成 O，第 1 类棋子的 O 改成 X；将第 1 类棋子的 X 改成 O，第 2 类棋子的 O 改成 X；将第 2 类棋子的 X 改成 O，第 3 类棋子的 O 改成 X。

这三种方案架加起来的改变次数恰好为 k，则根据抽屉原理，一定有至少一种方案改变次数小于等于 $\frac{1}{3}$。

Ehab's Last Corollary[15]

给出一个无向图，要找到一个恰好为 $\lceil \frac{k}{2} \rceil$ 个点的独立集或找到一个长度不超过 k 的简单环。

$3 \leq k \leq n \leq 10^5, n-1 \leq m \leq 2 \times 10^5$

[15] Codeforces 649D

Ehab's Last Corollary

在图上考虑构造问题时，有时会尝试使用 DFS 树的方法。

DFS 树即通过 DFS 构建出来的树，在无向图中具有只有返祖边，没有横叉边的性质。

在这题中，我们考虑该图的 DFS 树。

对于树上的每条非树边（即返祖边）(u, v)，若 u, v 的深度之差小于 k，则能构成一个长度不超过 k 的简单环。

Ehab's Last Corollary

否则，只有可能有两种情况：

一种情况为没有非树边，即为一棵树，则直接二分图染色，取其中较大的哪个集合即为独立集。

一种情况为有非树边但没有深度差在 $k-1$ 以内的非树边，则取深度最大的点和其 $2, 4, \ldots, 2\lceil \frac{k}{2} \rceil - 2$ 级祖先即可

这样构建出来的均为满足条件的独立集。

Strange Housing[16]

给出一个无向图，要选择一个点集满足这个点集内的点之间没有边，且通过至少有一个点在点集内的边可以联通整张图。

$n \leq 3 \times 10^5, m \leq 3 \times 10^5$

[16] Codeforces 694D

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。
假设我们已经考虑了前 $n-1$ 个点且满足条件，下面考虑第 n 个点。

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。

假设我们已经考虑了前 $n-1$ 个点且满足条件，下面考虑第 n 个点。

如果第 n 个点与之前点集中的点有连边，则不用加入点集；否则直接加入点集满足条件。

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。

假设我们已经考虑了前 $n-1$ 个点且满足条件，下面考虑第 n 个点。

如果第 n 个点与之前点集中的点有连边，则不用加入点集；否则直接加入点集满足条件。

这种方法是否有问题？

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。

假设我们已经考虑了前 $n-1$ 个点且满足条件，下面考虑第 n 个点。

如果第 n 个点与之前点集中的点有连边，则不用加入点集；否则直接加入点集满足条件。

这种方法是否有问题？

没有考虑前 n 个点的连通性！

Strange Housing

对于一部分题，我们可以考虑使用递归的方法进行构造。

假设我们已经考虑了前 $n-1$ 个点且满足条件，下面考虑第 n 个点。

如果第 n 个点与之前点集中的点有连边，则不用加入点集；否则直接加入点集满足条件。

这种方法是否有问题？

没有考虑前 n 个点的连通性！

我们按照 DFS 序进行这个过程即可。

Boring Game

有一个长度为 n 的序列 $a[1]\dots a[n]$，你想把它变成 $b[1]\dots b[n]$。

你可以进行一种操作：选取一个 $1<t<n$，把 $(a[t-1],a[t],a[t+1])$ 变成 $(a[t-1]+a[t],-a[t],a[t]+a[t+1])$

你可以进行任意次操作，问是否可行（可以将 a 变成 b）

Boring Game

常见套路：考察数列的前缀和。

Boring Game

常见套路：考察数列的前缀和。

$(a[t-1], a[t], a[t+1])$ 的前缀和为

$(s+a[t-1], s+a[t-1]+a[t], s+a[t-1]+a[t]+a[t+1])$

Boring Game

常见套路：考察数列的前缀和。

$(a[t-1], a[t], a[t+1])$ 的前缀和为

$(s+a[t-1], s+a[t-1]+a[t], s+a[t-1]+a[t]+a[t+1])$

$(a[t-1]+a[t], -a[t], a[t]+a[t+1])$ 的前缀和为

$(s+a[t-1]+a[t], s+a[t-1], s+a[t-1]+a[t]+a[t+1])$

Boring Game

常见套路：考察数列的前缀和。

$(a[t-1], a[t], a[t+1])$ 的前缀和为

$(s+a[t-1], s+a[t-1]+a[t], s+a[t-1]+a[t]+a[t+1])$

$(a[t-1]+a[t], -a[t], a[t]+a[t+1])$ 的前缀和为

$(s+a[t-1]+a[t], s+a[t-1], s+a[t-1]+a[t]+a[t+1])$

即我们实际上可以交换除了最后一个位置外的任意两个前缀和。

Boring Game

常见套路：考察数列的前缀和。

$(a[t-1], a[t], a[t+1])$ 的前缀和为

$(s+a[t-1], s+a[t-1]+a[t], s+a[t-1]+a[t]+a[t+1])$

$(a[t-1]+a[t], -a[t], a[t]+a[t+1])$ 的前缀和为

$(s+a[t-1]+a[t], s+a[t-1], s+a[t-1]+a[t]+a[t+1])$

即我们实际上可以交换除了最后一个位置外的任意两个前缀和。

所以只需要将前缀和数组除最后一位外排序比较即可。

绯色 IOI（抵达）[17]

有一个 n 个点的完全图。第 i 个点有点权 $v[i]$，连接第 i 个点和第 j 个的边边权为 $(v[i]-v[j])^2$。求一个边权和最小的哈密顿回路（即经过每个点恰好一次的回路）。

$n \leq 100000$。

[17] loj 521

绯色 IOI（抵达）

首先将 v 排序，设 $v[1] \leq v[2] \leq \cdots \leq v[n]$。

考虑将回路分为两部分，1 到 n 的和 n 到 1 的，显然可以发现，1 到 n 走的一定是编号递增的路径，n 到 1 的一定是编号递减的路径。

如果走了 $a->c->b->d(a<b<c<d)$，则

$$(v[a]-v[c])^2+(v[c]-v[b])^2+(v[b]-v[d])^2 >= (v[a]-v[b])^2+(v[b]-v[c])^2+(v[c]-c[d])^2$$

所以调整为 $a->b->c->d$ 不会变劣。

绯色 IOI（抵达）

设 1 到 n 的路径上的点为 A，n 到 1 路径上的点为 B。考虑证明一个更强的性质，对于每个 $i\in[2,n-2]$，i 和 $i+1$ 不全在 A 中，也不全在 B 中。

若 $i,i+1$ 均属于 A，设 p,q 满足 $p<i$，$i+1<q$ 且 p、q 在 B 中（即跨过 i,$i+1$ 这段的 B 边），我们将 (p,q),$(i,i+1)$ 改为 $(p,i+1)$,(i,q)，可以发现答案不会变劣。

证明：设 $v[i]-v[p]=a$,$v[i+1]-v[i]=b$,$v[q]-v[i+1]=c$，那么原来和为 $(a+b+c)^2+b^2$，现在为 $(a+b)^2+(b+c)^2$，容易看出和变小了。

所以我们可以发现，边 $(1,2),(n-1,n),(1,3),(2,4),(3,5)\dots(n-2,n)$ 即为最优解。

Modulo Matrix [18]

给定 n，要求构造一个 $n \times n$ 的矩阵，矩阵内的元素两两不同，且任意相邻的两个元素 x, y，满足 $max(x, y)\ mod\ min(x, y)$ 等于一个非零常数。

$n \leq 500$,$1 \leq$ 矩阵中的元素 $\leq 10^{15}$。

[18] AGC 027D

Modulo Matrix

如何使得 $max(x, y)\ mod\ min(x, y)$ 为常数？

Modulo Matrix

如何使得 $max(x,y)\ mod\ min(x,y)$ 为常数？

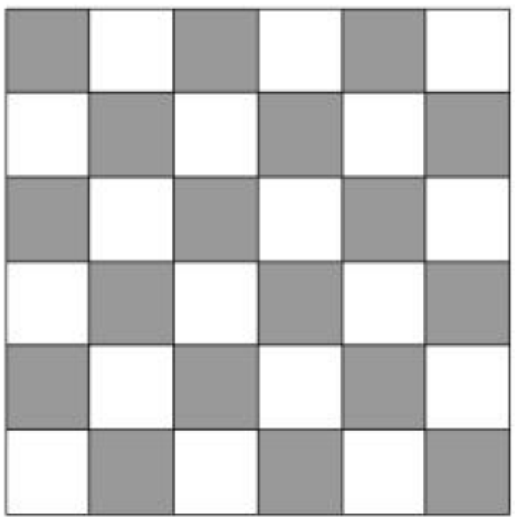
考虑将棋盘黑白染色，在黑格上填上一些不同的数，在白格上填上周围的数的 lcm 的一个倍数 $+1$。

Modulo Matrix

如何使得 $max(x, y)\ mod\ min(x, y)$ 为常数？

考虑将棋盘黑白染色，在黑格上填上一些不同的数，在白格上填上周围的数的 lcm 的一个倍数 $+1$。

一种简单的填黑格方法是全填不同质数，但是这样显然超出了 10^{15}。



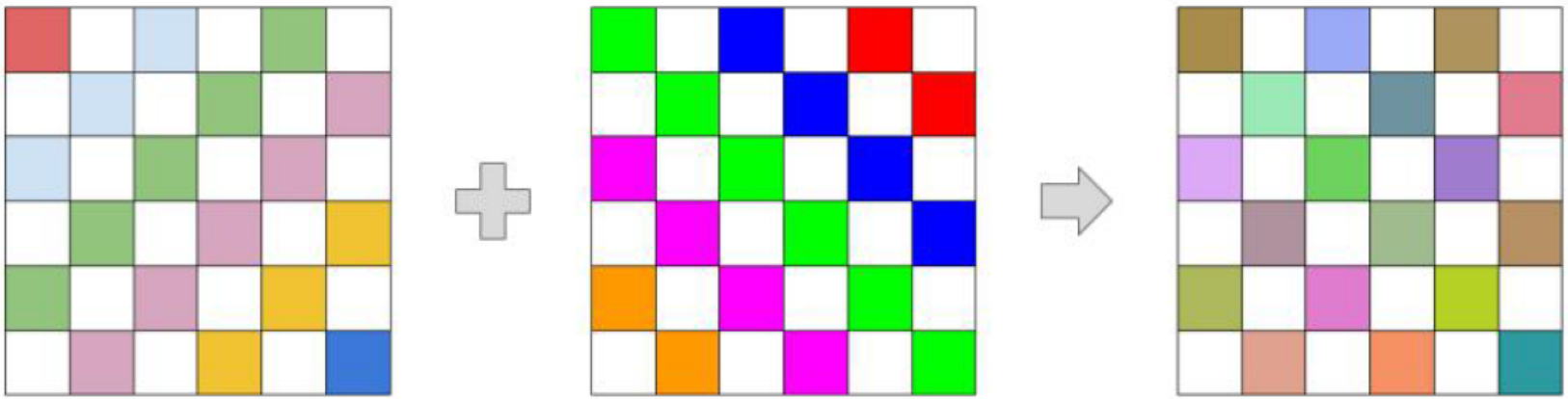
Modulo Matrix

考虑在每个黑格上填上两个质数之积。

Modulo Matrix

考虑在每个黑格上填上两个质数之积。

如图，在每条对角线上各选取一个质数，黑格上填上两种方向对角线上质数的乘积。这样每个白格的值就是四个小质数的乘积 $+1$，可以通过。



Misaka Network 与测试 [19]

有一个 n*m 的矩阵，每个位置可能会有 0、1、2、3，你需要选取最多的互不相交的子矩阵（即每个位置最多被一个子矩阵包含），满足每个子矩阵中的数不含有 0，且每个子矩阵位置上的数平均数为 2。

$n \times m \leq 10^5$。

[19] loj 569

Misaka Network 与测试

考虑调整法。

Misaka Network 与测试

考虑调整法。
如果某个子矩形中含有 2，那么我们把该子矩形换成这个 2 不会变劣。

Misaka Network 与测试

考虑调整法。
如果某个子矩形中含有 2，那么我们把该子矩形换成这个 2 不会变劣。
考虑不含有 2 的子矩形，由于它们的平均数为 2，那么必然又有 1 又有 3。
考虑只包含 13 的子矩形，必然存在一对位置相邻的 1 和 3。

Misaka Network 与测试

考虑调整法。
如果某个子矩形中含有 2，那么我们把该子矩形换成这个 2 不会变劣。
考虑不含有 2 的子矩形，由于它们的平均数为 2，那么必然又有 1 又有 3。
考虑只包含 13 的子矩形，必然存在一对位置相邻的 1 和 3。
我们把该子矩形换成这两个相邻的不会变劣。

Misaka Network 与测试

考虑调整法。
如果某个子矩形中含有 2，那么我们把该子矩形换成这个 2 不会变劣。
考虑不含有 2 的子矩形，由于它们的平均数为 2，那么必然又有 1 又有 3。
考虑只包含 13 的子矩形，必然存在一对位置相邻的 1 和 3。
我们把该子矩形换成这两个相邻的不会变劣。
那么我们就只要考虑所有矩形要么是 2，要么是相邻的两个 13 的情况了。

Misaka Network 与测试

考虑调整法。
如果某个子矩形中含有 2，那么我们把该子矩形换成这个 2 不会变劣。
考虑不含有 2 的子矩形，由于它们的平均数为 2，那么必然又有 1 又有 3。
考虑只包含 13 的子矩形，必然存在一对位置相邻的 1 和 3。
我们把该子矩形换成这两个相邻的不会变劣。
那么我们就只要考虑所有矩形要么是 2，要么是相邻的两个 13 的情况了。
使用 dinic 二分图匹配即可。

常数优化

优化算法的常数有时能产生意想不到的效果。

常数优化

优化算法的常数有时能产生意想不到的效果。
当然，算法常数过大也可能会产生意想不到的效果。

常数优化

优化算法的常数有时能产生意想不到的效果。
当然，算法常数过大也可能会产生意想不到的效果。
一般而言常数优化可以考虑尽量访问连续的内存。

常数优化

优化算法的常数有时能产生意想不到的效果。

当然，算法常数过大也可能会产生意想不到的效果。

一般而言常数优化可以考虑尽量访问连续的内存。

调用数组其实有一个寻址的过程，但如果你连续调用连续一段下标，寻址的过程将会变的很快。

常数优化

优化算法的常数有时能产生意想不到的效果。

当然，算法常数过大也可能会产生意想不到的效果。

一般而言常数优化可以考虑尽量访问连续的内存。

调用数组其实有一个寻址的过程，但如果你连续调用连续一段下标，寻址的过程将会变的很快。

常见方法有:

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

减少递归调用，有些题目保证树上 fa[i] < i，那么我们树形 dp 时可以从编号大的向编号小的转移，或者要多次树形 dp 时，可以在 dfn 上转移。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

减少递归调用，有些题目保证树上 fa[i] < i，那么我们树形 dp 时可以从编号大的向编号小的转移，或者要多次树形 dp 时，可以在 dfn 上转移。

能用树状数组就不要用线段树。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

减少递归调用，有些题目保证树上 fa[i] < i，那么我们树形 dp 时可以从编号大的向编号小的转移，或者要多次树形 dp 时，可以在 dfn 上转移。

能用树状数组就不要用线段树。

用树链剖分求 LCA 比倍增更快。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

减少递归调用，有些题目保证树上 fa[i] < i，那么我们树形 dp 时可以从编号大的向编号小的转移，或者要多次树形 dp 时，可以在 dfn 上转移。

能用树状数组就不要用线段树。

用树链剖分求 LCA 比倍增更快。

对于模数在 int 范围内的运算，尽量不用 longlong，必要时甚至可以用 short; char 来加速。

常数优化

调整循环顺序，让最里面的循环调用最后一层下表，如矩阵乘法，状压 dp，倍增等。

重新编号，例如把一条斜线标号成连续一段。

减少递归调用，有些题目保证树上 fa[i] < i，那么我们树形 dp 时可以从编号大的向编号小的转移，或者要多次树形 dp 时，可以在 dfn 上转移。

能用树状数组就不要用线段树。

用树链剖分求 LCA 比倍增更快。

对于模数在 int 范围内的运算，尽量不用 longlong，必要时甚至可以用 short; char 来加速。

使用循环展开，展开层数一般展开 4 层（我的经验）。

打表找规律

对于某些结论题，可以考虑暴力算出答案后尝试找规律，如:[NOIP2017 提高组] 小凯的疑惑、[NOIP2018 提高组] 填数游戏

打表找规律

对于某些结论题，可以考虑暴力算出答案后尝试找规律，如:[NOIP2017 提高组] 小凯的疑惑、[NOIP2018 提高组] 填数游戏

对于某些计数题，可以暴力打出前几项找规律。

打表找规律

对于某些结论题，可以考虑暴力算出答案后尝试找规律，如:[NOIP2017 提高组] 小凯的疑惑、[NOIP2018 提高组] 填数游戏

对于某些计数题，可以暴力打出前几项找规律。

更重要的是，对于考试中某些不确定的结论，如果造数据较为方便的话，尝试造数据暴力验证。

其他乱搞方法

该题目有无解情况，且没有多组数据，需要输出-1 或 No，直接输出，一般有 10 分，可能更多。

其他乱搞方法

该题目有无解情况，且没有多组数据，需要输出-1 或 No，直接输出，一般有 10 分，可能更多。

比如 NOIP2012 文化之旅，该题目需要你输出在满足题目限制条件下的最短路径，由于存在无解情况，输出-1 即可得到 10 分。

其他乱搞方法

该题目有无解情况，且没有多组数据，需要输出-1 或 No，直接输出，一般有 10 分，可能更多。

比如 NOIP2012 文化之旅，该题目需要你输出在满足题目限制条件下的最短路径，由于存在无解情况，输出-1 即可得到 10 分。

在写暴力时使用记忆化，如果出题人重复查询某个范围较大的区间，则可能可以获得额外的分数。

其他乱搞方法

该题目有无解情况，且没有多组数据，需要输出-1 或 No，直接输出，一般有 10 分，可能更多。

比如 NOIP2012 文化之旅，该题目需要你输出在满足题目限制条件下的最短路径，由于存在无解情况，输出-1 即可得到 10 分。

在写暴力时使用记忆化，如果出题人重复查询某个范围较大的区间，则可能可以获得额外的分数。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

但在实际的考试中，无论是联赛、省选还是全国赛，真正能在考场上做出正解的题目往往并没有那么多。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

但在实际的考试中，无论是联赛、省选还是全国赛，真正能在考场上做出正解的题目往往并没有那么多。

如何拿到/多拿部分分实际上是很重要的方法。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

但在实际的考试中，无论是联赛、省选还是全国赛，真正能在考场上做出正解的题目往往并没有那么多。

如何拿到/多拿部分分实际上是很重要的方法。

在这其中，随机化就是其中很重要的方法之一。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

但在实际的考试中，无论是联赛、省选还是全国赛，真正能在考场上做出正解的题目往往并没有那么多。

如何拿到/多拿部分分实际上是很重要的方法。

在这其中，随机化就是其中很重要的方法之一。

不仅如此，随机化和组合构造也往往出现在一些题目的正解中，作为正解的一部分（甚至很多时候是一个关键的思考点）。

总结

OI 中同学往往都只在乎正解，无论是网络的刷题网站还是一些模拟考很多都只在乎正解的打法。

但在实际的考试中，无论是联赛、省选还是全国赛，真正能在考场上做出正解的题目往往并没有那么多。

如何拿到/多拿部分分实际上是很重要的方法。

在这其中，随机化就是其中很重要的方法之一。

不仅如此，随机化和组合构造也往往出现在一些题目的正解中，作为正解的一部分（甚至很多时候是一个关键的思考点）。

有时应用这些方法可以为解题打开一个新的思路。

祝大家 NOIP 考试顺利！