随机化算法在信息学竞赛中的应用

湖南师大附中 胡泽聪

摘 要

随机化算法是一类十分有用的算法。通过引入随机函数，许多随机化算法可以在期望的意义下达到较之于确定性算法更优的时间复杂度。本文将简要介绍随机化算法的定义与分类，针对一些信息学竞赛中的例题分析随机化算法的应用与优越性，并总结出设计随机化算法的一般思路。

引 言

在信息学竞赛中，需要选手设计随机化算法的题目似乎比较少见，但是一旦出现了，往往会让不少选手束手无策。许多选手对随机化算法的了解仅限于各类骗分算法，因此在本文的前三章中，我们将简要介绍随机化算法的定义与分类。

由于随机化算法包含的范围较广，并没有一些通用的解决问题的方法，只能具体问题具体分析。故在本文的第四章中，我们将通过对于例题的分析，探究随机化算法的应用以及效果，并总结设计随机化算法解决实际问题的一般思路与分析方法。

1 定义与分类

随机化算法，也称概率算法，是这样一种算法，在算法中使用了随机函数，而且随机函数的返回值直接或者间接地影响了算法的执行流程或者执行结果。算法的一些决策可能依赖于随机的选择，并且可以通过随机因素改善算法的期望时间复杂度，或者得到可以接受的正确率。

随机化算法可以被分为以下三类：

- 数值概率算法;
- Monte Carlo算法;
- Las Vegas算法。

这三类算法都有着各自的特点与区别:

- **数值概率算法** 是通过随机选取元素从而求得在数值上的近似解。较之于传统方法，其运行速度更快，而且随着运行时间的增加，近似解的精度也会提高。在不可能或者不必要求出问题的精确解时，使用数值概率算法可以得到相当满意的近似解。常见的数值概率算法有**随机撒点法** [1]。
- Monte Carlo**算法** 总是能在确定的的运行时间内出解，但是得到的解有一定概率是错的。通常出错的概率比较小，因此我们可以通过反复运行算法来得到可以接受的正确率。常见的Monte Carlo算法有Miller-Rabin**素性测试算法** 。
- Las Vegas**算法** [2]总是能返回正确的结果，但是其运行时间不确定[3]。对于一些平均时间复杂度优秀，但是最坏情况下的复杂度较高的确定性算法，通过引入随机函数，尝试减小出现最坏情况的可能性，改造成Las Vegas算法，可以在期望意义下达到优秀的时间复杂度。常见的Las Vegas算法有**快速排序算法** [4]。

本文将侧重于介绍Monte Carlo算法与Las Vegas算法，而对于数值概率算法将不会过多提及，有兴趣的同学可以自行研究。

2 Monte Carlo算法

在信息学竞赛中，Monte Carlo算法应该是使用得相对较多的一类随机化算法。相比起Las Vegas算法，我们有时可以利用题目的性质，将一个确定性算法

[1]随机撒点法的一个主要应用是求不方便直接计算的图形的面积。

[2]在少数一些资料中，此类算法被称为Sherwood算法，并将运行时间确定但可能得不到解的随机化算法（如Pollard rho算法）称为Las Vegas算法。参考了许多其它资料后，作者决定将此类算法称为Las Vegas算法；对于前面提到的可能得不到解的算法，则直接将其归类为Monte Carlo算法，不单独分为一类。

[3]这里我们要求算法必须能在有限的时间内结束运行。运行时间有概率为无限的算法，比如低效的Bogo排序算法，不算作Las Vegas 算法。

[4]快速排序算法的随机性来源于划分过程中“基准”(pivot)的选择。

直接改造成Monte Carlo算法。

2.1 Monte Carlo算法的分类

在介绍例题之前，我们先对Monte Carlo算法进行分类。根据问题的性质，Monte Carlo算法可以分为以下两种：

- 求解最优化问题的Monte Carlo算法；
- 求解判定性问题的Monte Carlo算法。

而对于求解判定性问题的Monte Carlo算法，我们可以再进一步地分类为以下三种：

- **假倾向**(false-biased) [5]Monte Carlo算法。当这类算法的返回值为**假** 时，结果一定正确，但当返回值为**真** 时则有一定概率错误。
- **真倾向**(true-biased) Monte Carlo算法。当这类算法的返回值为**真** 时，结果一定正确，但当返回值为**假** 时则有一定概率错误。
- **产生双侧错误**(two-sided error) 的Monte Carlo算法。这类算法无论是返回真或假都有概率错误。

其中假倾向Monte Carlo算法和真倾向Monte Carlo算法并称为**产生单侧错误**(one-sided error) 的Monte Carlo算法。

我们在信息学竞赛中遇到的Monte Carlo算法基本上都是产生单侧错误的算法，因此下面的讨论中我们将忽略产生双侧错误的Monte Carlo算法。

2.2 Monte Carlo算法的正确率与复杂度

Monte Carlo算法的正确率与复杂度很好计算。假设我们有一个正确率为p，时间复杂度为$O(f(n))$的Monte Carlo算法。设我们运行这个Monte Carlo算法k次，那么显然有

- 正确率为$1-(1-p)^k$；
- 时间复杂度为$O(k \cdot f(n))$。

[5]由于没有找到对应的中文翻译，此处的中文为作者自己对括号内英文的翻译。以下带括号的词语亦同。

3 Las Vegas算法

相比起Monte Carlo算法，Las Vegas在信息学竞赛中则使用得相对较少。我们用到的Las Vegas算法大多是已有的算法，如**快速排序算法**、**随机增量算法**和Treap，需要选手自行设计Las Vegas算法的题目很少。在下面的例题中，也只有一道需要使用Las Vegas算法。

Las Vegas算法并不像Monte Carlo算法那样有各种类别，因此我们直接分析其复杂度的计算。

3.1 Las Vegas算法的复杂度

不同的Las Vegas算法之间的差别比较大，因此不存在通用的复杂度分析方法。我们将选取两个Las Vegas算法作为示范，来分析其复杂度。

3.1.1 快速排序算法

想必大家对快速排序算法都非常熟悉，并且熟知其最优复杂度为$O(n\log n)$，最差复杂度为$O(n^2)$，期望复杂度为$O(n\log n)$。这里我们尝试证明其期望复杂度。

设$T(n)$为对长度为n的序列运行快速排序算法所需的期望时间，我们有

$$T(0)=0$$

以及

$$T(n)=n+\frac{1}{n}\sum_{i=0}^{n-1}(T(i)+T(n-i-1))$$

我们通过放缩来获得对$T(n)$上界的一个估计

$$\begin{aligned}T(n) &= n+\frac{1}{n}\sum_{i=0}^{n-1}(T(i)+T(n-i-1))\\ &= n+\frac{2}{n}\sum_{i=n/2}^{n-1}(T(i)+T(n-i-1))\\ &= n+\frac{2}{n}\sum_{i=n/2}^{3n/4}(T(i)+T(n-i-1))+\frac{2}{n}\sum_{i=3n/4}^{n-1}(T(i)+T(n-i-1))\end{aligned}$$

由于$T(n) \geq n$，所以对于$\frac{n}{2} \leq i \leq j$我们显然有

$$T(i) + T(n-i) \leq T(j) + T(n-j)$$

因此

$$\begin{aligned} T(n) &\leq n + \frac{2}{n}\sum_{i=n/2}^{3n/4}(T(3n/4) + T(n/4)) + \frac{2}{n}\sum_{i=3n/4}^{n-1}(T(n-1) + T(0)) \\ &\leq n + \frac{1}{2}(T(3n/4) + T(n/4)) + \frac{1}{2}T(n-1) \end{aligned}$$

我们要证明$T(n) = O(n\log n)$，就需要证明存在常数c满足$T(n) \leq cn\log n$。我们考虑用数学归纳法证明。$n = 0$时定理显然成立。现假设对于任意$m \leq n$定理皆成立，那么

$$\begin{aligned} T(n) &\leq n + \frac{1}{2}(T(3n/4) + T(n/4)) + \frac{1}{2}T(n-1) \\ &\leq n + \frac{1}{2}(c(3n/4)\log(3n/4) + c(n/4)\log(n/4)) + \frac{1}{2}c(n-1)\log(n-1) \\ &\leq n + c\left(\frac{3n}{8}\log(n) - \frac{3n}{8}\log(4/3) + \frac{n}{8}\log(n)\right) - \frac{n}{8}\log(4) + \frac{n}{2}\log(n)\Big) \\ &= cn\log(n) + n\left(1 - \frac{3c}{8}\log(4/3) - \frac{c}{4}\right) \end{aligned}$$

当$1 - \frac{3c}{8}\log(4/3) - \frac{c}{4} \leq 0$时，也即约$c \geq \frac{5}{2}$时，我们有

$$T(n) \leq cn\log n$$

故归纳成立，$T(n) = O(n\log n)$。

3.1.2 一类由Monte Carlo算法改造而成的算法

对于一类一定有解的构造性问题，假设我们有一个正确率为p，时间复杂度为$O(f(n))$的产生单侧错误的Monte Carlo算法，我们可以通过不断运行该算法，直到找到解为止，来将其改造为Las Vegas算法。我们关注的是该算法的期望运行次数k。

我们可以列出k的计算式：

$$k = \sum_{i=1}^{\infty} p(1-p)^{i-1}i \tag{1}$$

两边乘上$1-p$得

$$(1-p)k = \sum_{i=1}^{\infty} p(1-p)^i i = \sum_{i=2}^{\infty} p(1-p)^{i-1}(i-1) \tag{2}$$

令式(1)减去式(2)得

$$p \cdot k = p + \sum_{i=2}^{\infty} p(1-p)^{i-1} = p \sum_{i=0}^{\infty} (1-p)^i \tag{3}$$

而右侧的合式是等比数列的求和，因此可以得出

$$k = \sum_{i=0}^{\infty} (1-p)^i = \frac{1-(1-p)^{\infty}}{1-(1-p)} = \frac{1}{p} \tag{4}$$

结果十分简洁。

因此，这个算法的期望运行时间为$O(p^{-1}f(n))$。

4 例题与分析

下面我们通过一些例题来体会如何使用随机化算法解决信息学竞赛中的问题。

例题一：MSTONE[6]

平面上有n个互不重合的点，已知存在不超过7条直线可以覆盖全部的点，问在平面上作一条直线，最多能覆盖多少个点。

$n \leq 10000$。

题目分析

我们不妨先考虑一个最朴素的算法：枚举两个点，确定一条直线，然后判断有多少个点在这条直线上。这个算法的复杂度为$O(n^3)$，无法在时间限制内求出问题的解。

我们考虑向这个朴素算法中加入随机，将其改造成Monte Carlo算法。问题就在于，在哪一个部分随机?

[6]**题目来源：** CodeChef。

注意到题目有一个很关键的条件：存在不超过7条直线可以覆盖全部的点。我们先来分析一下这个条件能给我们来带什么。

引理1.1. 覆盖了最多点的直线覆盖了至少$\left\lceil \frac{n}{7} \right\rceil$个点。

从而引出下面的定理

定理1.2. 从给定的n个点中随机选择两个点，过这两个点的直线与覆盖了最多点的直线重合的概率为$\frac{1}{49}$。

证明比较显然。

那么我们将“枚举两个点”的步骤改为“随机选择两个点”，从而将算法改造成Monte Carlo算法。运行这个算法一次的复杂度为$O(n)$，正确率为$\frac{1}{49}$。设算法运行k次，取$k = 1000$，可以求出算法的正确率为

$$1 - \left(1 - \frac{1}{49}\right)^k \approx 1 - 10^{-9}$$

这个正确率相当令人满意，而且运行时间方面也可以接受。 □

例题二： Ghd[7]

令一个集合S的“GHD”为

$$\max\left\{GCD(S') \middle| |S'| \geq \frac{1}{2}|S|\right\}$$

给定长度为n的序列a，求$GHD(\{a_1, a_2, \ldots, a_n\})$。

$n \leq 10^6$，$1 \leq a_i \leq 10^{12}$。时间限制为4秒。

题目分析

我们依然先考虑不加入随机的算法。一个朴素的算法是枚举GHD，然后检验是枚举的GHD的倍数的数有多少个。一个稍微优化一些的算法是枚举每个数的每个因子，当然这样也是没法通过这道题的。

我们不妨直接枚举每个数，然后直接考虑当这个数在选出的集合S'中时，可以取到的最大的GHD。设这个数为a_x，我们求出a_x和其它每个数a_i的GCD，记

[7]**题目来源：** Codeforces Round #213 (Div. 1) - Problem D。

为g_i。如果a_i也在集合S'中，能成为GHD的只能是g_i的因子。因此我们对于a_x的每个因子d记录cnt_d，代表有多少个数加入集合S'后GHD可以为d。那么对于每个g_i，枚举其所有因子p，并令cnt_p加1。

设$\tau(n)$为n的因子个数，这个算法的复杂度为$O(n(\sqrt{a_x}+\tau^2(a_x)+n\log n))$。在不超过$10^{12}$的所有数中，因子最多的数为$963,761,198,400$，有6720个因子，如果去掉枚举的部分，复杂度是可以承受的。

现在我们有了一个去掉枚举后时间复杂度可以接受的算法。我们仿照上一题的做法，考虑用随机来替代枚举。我们有如下定理：

定理2.1. 从给定的n个数中随机选择一个数，这个数在最终选出的集合中的概率为$\frac{1}{2}$。

证明十分显然。

那么我们将“枚举一个数”改成“随机选择一个数”，从而将算法改造成Monte Carlo算法。这个算法的正确率为$\frac{1}{2}$，可以说是相当之高，只需运行数十次即可得到满意的正确率。 □

算法设计思路1

我们可以从这两道例题中总结出一个大致的算法设计思路，与这两道题类似的题都可以使用下面的思路来设计一个可行的Monte Carlo 算法：

1. 设计一个能解决问题的确定性的算法。
   - 这个算法需要枚举一些元素；
   - 设这个算法的复杂度为$O(f(n)g(n))$，其中$f(n)$为枚举部分的复杂度，$g(n)$为单次枚举中计算所需的复杂度。应当保证运行若干次$O(g(n))$的算法不会导致超时。
2. 向算法中引入随机。将算法的枚举部分改成随机选择。
   - 你需要证明随机算法的正确率，并计算为了达到一个可以接受的正确率需要运行该算法多少次；
   - 通常情况下，需要使用随机算法的题目都有较强的约束条件，可以保证在随机选择的步骤中，选到所需要的元素的概率不会太低；

- 即使正确率不尽人意，我们也可以使用这个随机算法获得一定的部分分。在遇到提交答案题的时候尤为有效。 □

我们尝试使用上面的思路来解决下面两道例题：

例题三：Couriers[8]

给定长度为n的序列a，有m次询问，每次给定l和r，问$a[l..r]$中是否存在一个元素出现次数严格大于$\frac{1}{2}(r-l+1)$，如果有则输出该元素，否则输出-1。

$n, m \le 500000$，$1 \le a_i \le n$。

题目分析

我们还是先设计一个朴素的算法。我们可以枚举区间内的每个数，然后统计这个数在区间内出现的次数，并判断是否是答案。直接这么做的复杂度是$O(mn^2)$。

我们可以预处理每种数字出现的位置的序列，统计区间中出现次数的时候在序列中二分，这样把复杂度降到了$O(mn\log n)$，但还是会超时。

现在我们应用设计思路1，用随机来代替枚举。我们将“枚举区间内的每个数”改为“随机选择区间内的一个数”，从而将算法改造为随机化算法。我们有如下定理：

定理3.1. 如果询问的答案不是-1，那么一次随机选到答案的概率不小于$\frac{1}{2}$；否则无论怎么随机都无法找到答案。

故这个算法是产生单边错误的Monte Carlo算法，且正确率为$\frac{1}{2}$。

经实测，我们只需对每个询问运行这个算法20次即可通过这道题。当然为了保险起见，可以在时间允许的范围内运行更多次。算法的复杂度为$O(km\log n)$，其中$k = 20$。 □

例题四：TKCONVEX[9]

给定n条线段的长度，求一个从n条线段中选出$2k$条的方案，使得能用这些线段构成两个有k条边的凸多边形，或者指出问题无解。

[8]**题目来源：** POI2014。

[9]**题目来源：** CodeChef。此处的数据范围与原题相比有所不同。

$n \le 1000$，$3 \le k \le 6$。

题目分析

我们先考虑一个基础的问题：给定n条线段，判断这些线段是否能构成一个凸多边形。我们有如下引理：

引理4.1. 假设n条边的边长为$a_1, \ldots, a_n$，这些边能构成一个简单多边形当且仅当对于任意$1 \le i \le n$有

$$a_i \le \frac{1}{2}(a_1 + \cdots + a_n)$$

这个条件也等价于

$$\max(a_1, \ldots, a_n) \le \frac{1}{2}(a_1 + \cdots + a_n)$$

这个引理的证明与本文主题无关，故在这里略去。有兴趣的读者可以自行思考。

注意这个引理只能判断边是否能构成一个简单多边形，而并非凸多边形。不过，我们还有一个引理：

引理4.2. 任意非凸多边形都可以通过对其边的平移和旋转得到一个凸多边形。

出于与上面相同的理由，这里略去该引理的证明。那么我们可以用引理4.1来判断给定的线段是否能构成凸多边形。

在本题中，可以选择的集合有$\binom{n}{2k}\binom{2k}{k}$个之多，我们当然不能一个一个地去检验。我们从上面的引理出发，继续探究问题的性质，尝试缩小可选集合的范围。我们可以得出下面的定理：

定理4.3. 假设n条线段中存在可以构成凸多边形的大小为k的子集，那么一定存在一个这样的子集满足，对所有线段按长度排序后，这个子集中的元素对应排序后序列中的一个区间。

证明： 任选一个可以构成凸多边形的大小为k的子集，记为S。设排序后的序列为p，假如S中的元素对应到p上并非一个区间，那么一定存在至少两个$i(i < n)$，满足

$$p_i \in S \text{ 且} p_{i+1} \notin S$$

我们找到满足条件的第二大的i，并从S中删去p_i，再加入p_{i+1}，由引理4.1可知此时集合S中的线段仍然可以构成凸多边形。重复上述操作直到找不到满足条件的i，此时S中的元素对应到p中一定是一个区间。 □

注意到我们的定理只是用来处理选出一个大小为k的集合的情况的。那么我们怎么将其应用到原问题上呢?

一个很直接的想法就是，把线段长度的序列分成两部分，然后在两部分中各找一个长度为k的区间，使得区间中的线段可以构成凸多边形。但是如果要枚举子集，无疑会超时。

既然这里是“枚举”，我们尝试套用上面的设计思路，将其改造为Monte Carlo算法。设两个集合分别为S和$\bar{S}$，对于每条线段，我们有0.5的概率将其放入S中，有0.5的概率将其放入$\bar{S}$中。这样我们选出任意一个子集的概率是相同的。接下来我们对S和$\bar{S}$分别运行上面的算法。运行一次的复杂度可以做到$O(n)$。

我们来分析这个算法的正确率。划分的总方案数为2^n。考虑一个极端的情况，有且仅有两个不相交集合X和Y可以构成凸多边形。那么一个划分方案有解，当且仅当$S \cap X = X$且$S \cap Y = \varnothing$，或者$S \cap X = \varnothing$且$S \cap Y = Y$。这样的方案数为$2^{n-2k} \times 2$。因此在最坏情况下，选出一个合法划分的概率，也即算法的正确率，约为$\frac{1}{2^{2k-1}}$。

设我们运行这个算法t次，则其复杂度为$O(n \log n + tn)$。在不超时的情况下，t最多可以取到约20000，此时算法的正确率在$k = 6$时约为

$$1 - \left(1 - \frac{1}{2^{11}}\right)^t \approx 1 - 10^{-5}$$

已经相当令人满意了。 □

事实上，例题一、例题三与例题四均存在确定性算法[10]。这两道题的确定性算法需要进一步分析题目的性质，与这里介绍的随机化算法相比，思维难度更高。

由此可见，引入随机化可以在一定程度上降低思维难度、简化算法，同时对于一些题目来说，随机化算法还可以达到更优的复杂度。

[10]例题四的确定性算法的大致思路是，证明对于任意大小不小于约70的集合均有解，从而缩小数据范围。接着再证明定理4.3的一个加强，即选出的集合对应一个长度为$2k$的区间，或者两个长度为k的不相交区间。至此就可以直接枚举枚举区间然后用确定性算法解决了。

例题五：向量内积[11]

定义两个d维向量$A = [a_1\ a_2\ \ldots\ a_d]$和$B = [b_1\ b_2\ \ldots\ b_d]$的内积为其对应维度的权值的乘积和，即

$$(A, B) = \sum_{i=1}^{d} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_d b_d$$

现有n个d维向量$x_1, x_2, \ldots, x_n$，问是否存在两个向量的内积为k的倍数。如果存在，输出任意一组解。

数据分为两类：

- $k = 2$，$n \le 10000$，$d \le 100$；
- $k = 3$，$n \le 100000$，$d \le 30$。

题目分析

$k = 3$的情况处理起来较为复杂，而且其难点与本文主题无关，故下面只讨论$k = 2$的情况。

我们将这个n个向量组成一个$n \times d$的矩阵，记这个矩阵为A。令A^T为A的转置，设$B = A \times A^T$，那么$B_{i,j}$的值实际上就是第i个向量和第j的向量的内积。而我们实际上要判断的就是，在B中是否存在一个不在主对角线上的位置(i, j)，满足$B_{i,j} \equiv 0 \pmod 2$。直接计算矩阵乘法与暴力无异，我们需要想别的方法。

为了处理方便，我们把所有向量的每一维都对2取模，计算中也全程对2取模。我们构造一个$n \times n$的矩阵C，满足C的主对角线与B的主对角线相同，而且主对角线以外的元素都是1。那么问题就变成了判断$A \times A^T = C$是否成立，如果不成立则找到一个不同的位置。B的主对角线可以在$O(nd)$的时间内求出来，故构造矩阵C是没有问题的。

分析到了现在，我们似乎无法继续前进了。确定性算法应该是无法避免矩阵乘法的，而矩阵乘法的复杂度无法接受。我们考虑引入随机化，但是这个问题似乎也无法套用思路1，因为可以构造出矩阵B只有很少几个元素不为1的数据。我们需要一种新的随机化思路。

假设$A \times A^T = C$成立，那么对于任意$m \times n$的矩阵X都应满足$X \times A \times A^T = X \times C$，其中$m$为任意正整数。如果取$m = 1$，那么$X$就是一个向量。我们能不能随机生

[11]**题目来源：** NOI2013 Day 1

成X，然后用这个随机的X来检验?

对于$X \times A \times A^T$，计算一次的复杂度为$O(nd)$。对于$X \times C$，由于矩阵C具有很大的特殊性，我们可以直接推出其乘出来的向量。设$Y = X \times C$，那么有

$$Y_i = (C_{i,i} - 1)X_i + \sum_{j=1}^{n} X_j$$

因此我们可以在$O(n)$的时间内求出$X \times C$。在时间复杂度方面是没有问题的。

而在算法正确率方面，我们有如下定理:

定理5.1. 如果$A \times A^T = C$，一定会返回相等。而如果$A \times A^T \neq C$，算法返回相等的概率不超过$\frac{1}{2}$。

证明： 由矩阵乘法运算的结合律可知，定理的第一部分成立。我们考虑如何证明定理的第二部分。

令$D = A \times A^T - C$，由假设$A \times A^T \neq C$得，D中存在至少一个不为0的元素，设其中一个为$D_{i,j}$。我们假设算法返回了相等，现在我们需要求出X的哪些取值会导致这一错误，并求出X取到这样的值的概率。

令$E = X \times D$，考虑E_j。算法会返回相等，当且仅当E的每个元素均为0，因此

$$E_j = \sum_k X_k D_{k,j} = 0$$

由于我们假设$D_{i,j} \neq 0$，因此$E_j = 0$当且仅当

$$X_i = -\frac{1}{D_{i,j}} \sum_{k \neq i} X_k D_{k,j}$$

因此，当X中其它元素都已确定时，要使得E的每个元素均为0，X_i只有一种取值。而$X_i \in \{0, 1\}$，所以X_i取到会导致错误的值的概率为$\frac{1}{2}$。

由此可知，算法返回相等的概率不超过$\frac{1}{2}$，定理的第二部分得证。故定理得证。 □

我们求出的正确率十分的可观，这意味着我们只需要运行算法数十次即可判断相等，在时间上完全可以接受。

现在我们还需找到一个为0的元素的位置。我们找到一个不为0的E_i，那么一定存在一组解包含了第i个向量，我们只需枚举另一个向量并计算检验即可。这一部分的复杂度为$O(nd)$。

至此$k = 2$的情况已经被我们完美地解决了。 □

算法设计思路2

我们试着从上一道题中总结一个具有一般性的思路。对于类似上一题的一些题，可以使用下面这一思路来设计一个可行的Monte Carlo 算法：

1. 针对问题设计一个确定性的算法，这个算法需要用到一个传入的向量。
   - 算法接受的向量中的值不应当依赖于输入数据。换句话说，我们并非从输入数据中抽出一个向量传入到算法的函数中——我们应当将这样的算法归类到设计思路1；
   - 通常这一类算法都直接是为了随机化而设计的，因此比起设计思路1，拥有更高的思考难度。
2. 向算法中引入随机。随机生成一个向量，并传入到上面算法的函数中。
   - 可以使用这一类算法通过，或者是标准算法就是这一类算法的题目不多。但这并不代表这一类算法没有意义；
   - 对于一些有多维代价的最优化问题，可以用这一类算法获得一定的部分分。具体做法是随机一个向量代表权重，并重新定义代价为原代价向量与权重向量的内积。以新的代价求一遍只有一维代价的最优化问题，并构造方案，计算出实际代价；
   - 而对于一些和计算几何有关的题目，也可以使用这个方法。 □

例题六：Graph Reconstruction[12]

给定一个含有n个顶点和m条边的无向图，其中每个顶点的度数不超过2，且图中无自环与重边。构造一个新图，满足以下条件：

- 新图含有n个顶点与m条边；
- 新图中每个顶点的度数不超过2，且不含有自环与重边；
- 假设在原图中顶点u和v之间存在一条边，则在新图中顶点u和v之间不得有边。

输出任意一个满足条件的新图，或指出无解。

$1 \le m \le n \le 100000$。

[12]**题目来源：** Codeforces Round #192 (Div. 1) - Problem C

题目分析

我们先分析题目中给出的图的性质。由于每个顶点的度数不超过2，因此原图应该是一些环和链，而且在原图的补图中，每个顶点的度数不小于$n-3$。问题实际上就是在原图的补图中找出m条边，满足给出的条件。

先考虑，在什么情况下问题会无解。我们考虑一个较强的约束：求出的新图应为一个哈密尔顿回路。对于哈密尔顿回路的存在性，我们有如下定理：

定理6.1 (Ore's Theorem)**.** 对于一个含有n个顶点的无向图，其中存在哈密尔顿回路的充分条件是，对于任意两个不相邻的顶点u和v，满足

$$\deg u + \deg v \geq n$$

其中$\deg u$代表顶点u的度数。

由上面的分析可知，$\deg u \geq n-3$对于每个顶点都成立，故定理6.1的条件实际上就是

$$2(n-3) \geq n$$

也即$n \geq 6$。因此，对于任意$n \geq 6$的情况，问题始终有解。那么在下面的分析中，我们将只考虑$n \geq 6$的情况。

如果我们求出了原图的补图的一个哈密尔顿回路，那么我们只需在哈密尔顿回路上任选m条边作为新图即可，易知这样得出的新图满足题目的要求。现在问题就在于如何求出哈密尔顿回路了。

众所周知，在给定的图中求一个哈密尔顿回路是NPC问题，不存在高效的算法。而且我们似乎也不好设计一个可以按照设计思路1改造成Monte Carlo算法的确定性算法。这里我们介绍一个Las Vegas算法，并尝试求出其期望时间复杂度。

算法流程如下：

1. 随机一个$1 \sim n$的排列$p_1, \ldots, p_n$；
2. 对于$1 \leq i < n$，检查无向边(p_i, p_{i+1})是否存在于原图中，同时检查无向边(p_n, p_1)是否存在于原图中；
3. 如果检查的所有边都不存在于原图中，那么我们成功地找到了一个哈密尔顿回路；否则则返回第一步。

这个算法的步骤一和二的时间复杂度可以做到$O(n\log n)$或者$O(n)$，现在我们关注的就是期望意义下，算法需要运行几遍?

我们可以发现，这是一个由Monte Carlo算法改造而成的Las Vegas算法，因此计算期望复杂度的关键在于求出算法的正确率。不过对于这个算法而言，求出精确的正确率相当麻烦，因此我们考虑求出近似的正确率。

我们假设原图是一个长度为n的环。考虑逐一确定排列的每一位，令f_i^n表示，排列长度为n且确定了前i位时，尚未产生错误[13]的概率。特别地，我们直接用f^n表示f_n^n。由于只是近似，我们不考虑具体哪些点已经加入排列，只考虑一次选择中不产生错误的概率。

显然我们有$f_1^n = 1$，对于任意$i(i > 1)$，我们可以做如下的估计

$$\begin{aligned} f_i^n &= f_{i-1}^n \frac{1}{\binom{n-1}{2}} \left(\binom{i-2}{2} + (i-2)(n-i+1)\frac{n-i}{n-i+1} + \binom{n-i+1}{2}\frac{n-i-1}{n-i+1} \right) \\ &= \frac{n-3}{n-1} f_{i-1}^n \\ &= \left(\frac{n-3}{n-1}\right)^{i-1} \end{aligned}$$

结果十分简洁。那么对于f^n有

$$f^n = \left(\frac{n-3}{n-1}\right)^{n-1}$$

对其求极限可知

$$\lim_{n\to\infty} f^n = \frac{1}{e^2} \approx 0.135335$$

而当$n \geq 9$时，已经有$f^n \geq 0.1$。根据3.1.2小节中的分析可知，这个算法的期望运行次数$k = p^{-1} = (f^n)^{-1}$。对于足够大的n，我们可以认为$k \approx 10$，在运行时间上可以说是绰绰有余。至于$n < 9$的情况，我们可以直接用暴力算法求解。 □

由此我们总结出本文的最后一个设计思路：

算法设计思路3

我们尝试从上一道题中总结一个相对通用的思路。对于一些操作上类似的题，都可以使用下面的思路设计一个可行的Las Vegas算法：

[13]此处的“产生错误”即发现了一条属于原图的边。

1. 设计一个能解决问题的确定性算法。
    - 这个算法需要枚举所有元素的一个排列；
    - 通常能够使用这个思路解决的问题，要么是只求一个可行方案而不要求最优，要么是最优方案特别多。总之，需要保证“有用”的排列个数不会太少。
2. 向算法中引入随机。将算法枚举排列的部分改为随机一个排列。
    - 这一部分的分析往往是问题的关键。正如你在上一道题中所看到的一样，这一类随机算法的复杂度分析一般比较复杂，通常情况下无法做出精确的计算；
    - 如果你在比赛的时候想到了这样一个算法，与其花时间给出严格的证明，不如通过实践来检验真理；
    - 值得一提的是，可以通过限制运行次数（俗称“卡时”）来将这个算法改造成Monte Carlo算法。对于根据解的优劣程度给部分分的题目来说，即使算法正确率不高也可以出奇迹；
    - 或许对于有些题目而言，这样的算法真的很不靠谱。但如果你坚信这道题的数据很不好出，而且一时半会找不到别的更好的确定性算法，不妨就用这个方法。 □

5 总结

通过对两类算法的简要介绍以及对五道例题的分析，可以看出，随机化算法可以运用在各种类型的题目中。不过，随机化算法本身涵盖的范围就很广，因而不存在一些通用的处理问题的方法，只能具体问题具体分析。

随机化算法相比起确定性算法有如下的优势：

- 不需要对问题的性质做过多分析；
- 编程复杂度较低；
- 一些情况下，可以达到更优的时间复杂度；
- 对于有些题目而言，随机化算法是唯一的正确算法。

当然，也有一些劣势：

- 需要比较严谨的复杂度与正确率的证明，有些时候证明会相当复杂；

- 并非所有问题都存在随机化的做法，大多数时候随机化只能作为获得部分分的工具。

本文针对具有不同操作的一些随机化算法进行了分类，并提出了3个相对通用的设计算法的思路。当然这些并不是设计随机化算法仅有的几个思路。我们在面对需要设计随机化算法的题目时，应当发挥创造新思维，并力求严谨。希望本文能对大家有所启发。

6 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢我的教练李淑平老师对我的指导与关心。
感谢父母对我学习信息学竞赛的支持与鼓励。
感谢一路陪我走来的许许多多的同学的帮助。

参考文献

[1] Wikipedia. “*Randomized algorithm*”.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms (Second Edition)*. Massachusetts, USA: The MIT Press.

[3] Motwani Rajeev (1995). *Randomized algorithms*. Cambridge, UK: Cambridge University Press.