

算法设计与分析

随机算法

[见原文第 2 页 - Gregory Chaitin 名言图片]

“随机性是数学的真正基础。”

— Gregory Chaitin

[见原文第 3 页 - "Rocking Randomness" T恤图片]

目录

1 概率论预备知识

2 随机数据结构

- 字典

3 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

4 总结

有限概率空间

概率空间 (Ω, P) 是一个数学构造，为随机过程或“实验”提供形式化模型

- 样本空间 Ω ：所有可能结果的集合
 - ▶ Ω 中的每个单独元素称为基本事件
- 概率函数：为事件空间中的每个事件分配一个概率，该概率是 0 到 1 之间的实数

随机变量 X 是从样本空间 Ω 到可测空间 S （即 $[0, 1]$ ）的可测函数 $P: \Omega \rightarrow S$ 。

[见原文第 5 页 - 骰子图片]

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

基本事件 E_i ：骰子落在 i

$$P(E_i) = 1/6 \text{ 对于 } i \in [6]$$

事件的组合

\bar{E} : 事件 E 的补, 即 E 不发生。

- 定义 $\Rightarrow \Pr[\bar{E}] = 1 - \Pr[E]$

$E_1 \wedge E_2$ 表示它们的合取: E_1 和 E_2 都发生

- $0 \leq \Pr[E_1 \wedge E_2] \leq \min\{\Pr[E_1], \Pr[E_2]\}$
- E_1 和 E_2 独立 $\Rightarrow \Pr[E_1 \wedge E_2] = \Pr[E_1] \cdot \Pr[E_2]$

$E_1 \vee E_2$ 表示它们的析取: E_1 或 E_2 发生

- $\Pr[E_1] + \Pr[E_2] \geq \Pr[E_1 \vee E_2] \geq \max\{\Pr[E_1], \Pr[E_2]\}$

联合界

$$\Pr[\bigvee_{i=1}^k E_i] \leq \sum_{i=1}^k \Pr[E_i]$$

- E_1 和 E_2 互斥 $\Rightarrow \Pr[E_1 \vee E_2] = \Pr[E_1] + \Pr[E_2]$

例子。 设 E 为随机掷骰子结果为偶数的事件。显然, $E = E_2 \vee E_4 \vee E_6$, 其中 E_2, E_4, E_6 互斥, 因此 $\Pr[E] = 1/6 + 1/6 + 1/6 = 1/2$ 。

期望

设 X 为随机变量。其期望（或期望值） $\mathbb{E}(X)$ 是 $x \in \Omega$ 的概率加权平均：

$$\mathbb{E}(X) = \sum_{x \in \Omega} x \cdot \Pr[X = x] = \sum_{x \in \Omega} x \cdot P(x)$$

期望的两个有用性质

当 $\Omega = \{0, 1\}$ 时，我们称 X 为 **0-1 变量**。

$$\mathbb{E}(X) = \sum_{x \in \Omega} x \cdot P(x) = \Pr[X = 1]$$

线性性。 对于具有任意依赖关系的变量 X 和 Y ：

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y), \quad a, b \in \mathbb{N}$$

$$\mathbb{E}\left(\sum_{i=1}^k X_i\right) = \sum_{i=1}^k \mathbb{E}(X_i)$$

和的期望等于期望的和

线性性的应用

好处。 将复杂计算分解为更简单的部分。

我们接下来提供几个如何使用期望线性性的例子。在此之前，我们先描述一个简单的引理。

线性性的应用

好处。 将复杂计算分解为更简单的部分。

我们接下来提供几个如何使用期望线性性的例子。在此之前，我们先描述一个简单的引理。

引理

设 E 是一个以概率 p 发生的事件。那么平均需要进行 $1/p$ 次才能使 E 发生。

线性性的应用

好处。 将复杂计算分解为更简单的部分。

我们接下来提供几个如何使用期望线性性的例子。在此之前，我们先描述一个简单的引理。

引理

设 E 是一个以概率 p 发生的事件。那么平均需要进行 $1/p$ 次才能使 E 发生。

证明 1。 使用全概率定理：

$$N = 1 \cdot p + \cdots + i \cdot (1 - p)^{i-1} p + \cdots \Rightarrow N = 1/p$$

线性性的应用

好处。 将复杂计算分解为更简单的部分。

我们接下来提供几个如何使用期望线性性的例子。在此之前，我们先描述一个简单的引理。

引理

设 E 是一个以概率 p 发生的事件。那么平均需要进行 $1/p$ 次才能使 E 发生。

证明 1。 使用全概率定理：

$$N = 1 \cdot p + \cdots + i \cdot (1 - p)^{i-1} p + \cdots \Rightarrow N = 1/p$$

证明 2。 设 N 为事件发生前的期望次数。我们至少需要一次尝试。

- 如果发生了，我们就完成了。
- 如果没有（以概率 $1 - p$ 发生），我们需要重复。

因此： $N = 1 + [p \cdot 0 + (1 - p) \cdot N] \Rightarrow N = 1/p$

猜牌（无记忆）(1/2)

游戏。洗一副 n 张牌；逐一翻开；尝试猜测每张牌。

无记忆猜测。不能记住已经翻过的牌；从整副牌中均匀随机猜测一张。

[见原文第 9 页 - 扑克牌魔术师图片]

猜牌（无记忆）(2/2)

断言。正确猜测的期望次数为 1。

证明。[使用期望的线性性，出奇地简单]

- 设 $X_i = 1$ 如果第 i 次预测正确，否则为 0。
- 设 $X =$ 正确猜测的次数 $= X_1 + \cdots + X_n$ 。
- $\mathbb{E}[X_i] = \Pr[X_i = 1] = 1/n$ 。
- 线性性 $\Rightarrow \mathbb{E}[X] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n] = 1/n + \cdots + 1/n = 1$ 。

猜牌（有记忆）(1/2)

游戏。洗一副 n 张牌；逐一翻开；尝试猜测每张牌。

有记忆猜测。从尚未见过的牌中均匀随机猜测一张。

[见原文第 11 页 - 电影剧照图片]

猜牌（有记忆）(2/2)

断言。正确猜测的期望次数为 $\Theta(\log n)$ 。

证明。

- 设 $X_i = 1$ 如果第 i 次预测正确，否则为 0。
- 设 $X =$ 正确猜测的次数 $= X_1 + \cdots + X_n$ 。
- $\mathbb{E}[X_i] = \Pr[X_i = 1] = 1/(n - (i - 1))$ 。
- 线性性 \Rightarrow

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n] = 1/n + \cdots + 1/2 + 1/1 = H(n) = \Theta(\log n)$$

优惠券收集者 (1/3)

优惠券收集者。 每盒麦片包含一张优惠券。共有 n 种不同类型的优惠券。假设所有盒子包含每种优惠券的可能性相等，需要多少盒才能使每种优惠券至少有一张？

[见原文第 13 页 - 小浣熊干脆面和水浒卡图片]

优惠券收集者 (2/3)

断言。期望步数为 $\Theta(n \log n)$ 。

证明。

- 定义。阶段 j : 从拥有 j 种到 $j+1$ 种不同优惠券之间的时期。
- 设 X_j = 在阶段 j 花费的步数。
- 设 X = 总步数 $= X_0 + X_1 + \cdots + X_{n-1}$ 。

$$\mathbb{E}(X) = \sum_{j=1}^{n-1} \mathbb{E}(X_j) = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^n \frac{1}{i} = nH(n)$$

$$\text{成功概率} = (n-j)/n \Rightarrow \text{期望步数} = n/(n-j)$$

优惠券收集者 (3/3)

如果 X_i 的分布不是均匀的, $\mathbb{E}(X)$ 会更高。

优惠券收集者 (3/3)

如果 X_i 的分布不是均匀的, $\mathbb{E}(X)$ 会更高。

[见原文第 15 页 - 表情包图片]

优惠券收集者 (3/3)

如果 X_i 的分布不是均匀的, $\mathbb{E}(X)$ 会更高。

[见原文第 15 页 - 表情包图片]

长大之后我才知道, 水浒卡是一代人共同的记忆, 可能有数千万人卷入其中。据说, 小浣熊公司挣到几个亿, 用利润建起一栋办公大楼。
—某知乎网友

马尔可夫不等式

马尔可夫不等式。 设 X 为非负随机变量, $v > 0$ 。 则

$$\Pr[X \geq v] \leq \mathbb{E}(X)/v$$

证明。 设 X 取值于 Ω 。 我们有:

$$\begin{aligned}\mathbb{E}(X) &= \sum_{x \in \Omega} x \cdot \Pr[X = x] \\ &\geq \sum_{x \in \Omega, x < v} 0 \cdot \Pr[X = x] + \sum_{x \in \Omega, x \geq v} v \cdot \Pr[X = x] \\ &= v \cdot \Pr[X \geq v]\end{aligned}$$

马尔可夫不等式在对 X 知之甚少时很有用。

Chernoff 界（高于均值）(1/2)

定理

假设 X_1, \dots, X_n 是独立的 0-1 随机变量。设 $X = X_1 + \dots + X_n$ 。则对于任意 $\mu \geq \mathbb{E}[X]$ 和任意 $\delta > 0$, 有:

$$\Pr[X > (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu$$

独立 0-1 随机变量之和紧密集中在均值附近

证明。 我们应用一系列简单变换。 $\forall t > 0$,

$$\Pr[X > (1 + \delta)\mu] = \Pr[e^{tX} > e^{t(1+\delta)\mu}] \leq e^{-t(1+\delta)\mu} \cdot \mathbb{E}(e^{tX})$$

e^{tx} 关于 x 单调

马尔可夫不等式: $\Pr[X > a] \leq \mathbb{E}(X)/a$

$$\mathbb{E}(e^{tX}) = \mathbb{E}(e^{t \sum_{i=1}^n X_i}) = \prod_i \mathbb{E}(e^{tX_i})$$

X 的定义

X_i 之间的独立性

Chernoff 界（高于均值）(2/2)

设 $p_i = \Pr[X_i = 1]$ 。由于 $\forall \alpha \geq 0, 1 + \alpha \leq e^\alpha$ ，我们有：

$$\mathbb{E}(e^{tX_i}) = p_i e^t + (1 - p_i) e^0 = 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)}$$

综合所有内容：

$$\begin{aligned} \Pr[X > (1 + \delta)\mu] &\leq e^{-t(1+\delta)\mu} \prod_i \mathbb{E}(e^{tX_i}) && // \text{上一页} \\ &\leq e^{-t(1+\delta)\mu} \prod_i e^{p_i(e^t - 1)} && // \text{上述不等式} \\ &\leq e^{-t(1+\delta)\mu} e^{\mu(e^t - 1)} && // \sum_i p_i = \mathbb{E}(X) \leq \mu \end{aligned}$$

最后，选择 $t = \ln(1 + \delta)$ 。

Chernoff 界（低于均值）

定理

假设 X_1, \dots, X_n 是独立的 0-1 随机变量。设 $X = X_1 + \dots + X_n$ 。则对于任意 $\mu \leq \mathbb{E}(X)$ 和任意 $0 < \delta < 1$ ，有：

$$\Pr[X < (1 - \delta)\mu] < e^{-\delta^2\mu/2}$$

证明思路类似。

注记。 不完全对称，因为只考虑 $\delta < 1$ 才有意义。

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

动机：字典数据结构

字典。 给定一个全集 U ，维护一个子集 $S \subseteq U$ ，使得在 S 中的插入、删除和查找操作是高效的。

- $S \subseteq U$ 是我们真正关心的，可能是静态的或动态的。通常 $|S| \ll |U|$ ，例如， S 是这门课学生姓名的集合 $\ll 5000^{80}$

字典接口。

- `create()`：初始化一个字典， $S = \emptyset$ 。
- `lookup(u)`： u 在 S 中吗？
- `insert(u)`：将元素 $u \in U$ 添加到 S 。
- `delete(u)`：从 S 中删除 u （如果 s 当前在 S 中）。

应用。 文件系统、数据库、编译器、校验和、P2P 网络、关联数组、密码学、网页缓存、AI 搜索算法等。

简单情况：静态字典

对于静态字典，我们可以使用带二分查找的有序数组进行查找。
 $|S| \ll |U|$ 且一旦确定就固定不变。

- create $\rightsquigarrow O(|S| \log |S|)$ (排序)
- lookup $\rightsquigarrow O(\log |S|)$
- 没有删除和插入

动态字典怎么办？

动态字典的挑战

全集 U 可能非常大, S 可能不断增长

- ① 尝试 1: 使用有序数组作为静态字典, 查找复杂度为 $O(\log |S|)$, 插入和删除复杂度很高 $\rightsquigarrow O(|S|)$
- ② 尝试 2: 定义一个大小为 $|U|$ 的数组 A , 当且仅当 $u \in S$ 时设置 $A[u] = 1$.
 - ▶ U 可能不是 \mathbb{N} 的子集 \rightsquigarrow 元素不能直接用作索引
 - ▶ U 可能很大 $\rightsquigarrow A$ 会很大 \rightsquigarrow 太浪费空间

动态字典的正确数据结构

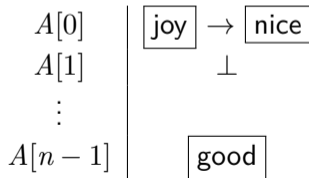
- 平衡搜索树。
- 哈希提供了另一种方法: 通常是最快和最方便的方式

哈希

哈希函数。 $h: U \rightarrow \{0, 1, \dots, n-1\}$ 。

哈希方法的思想： 数组 + 分离链接

- 创建数组 $A[n]$ ，每个 $A[i]$ 存储满足 $h(u) = i$ 的元素 u 的链表。
- 要插入元素 u ，将 u 放在 $A[h(u)]$ 处链表的顶部。
- 要查找元素 u ，只需计算索引 $i = h(u)$ ，然后沿着 $A[i]$ 处的链表向下走直到找到它（或走出链表）。
- 要删除，只需在相关链表上执行删除操作。



思考

哈希的作用是双重的

- 将大的全集 U (可能是稀疏的) 压缩到大小为 n 的小全集
- 小全集是 \mathbb{N} 的子集 \rightsquigarrow 可以用作索引

哈希的一个重要性质是所有字典操作都非常容易实现。

我们现在要讨论的问题是：

哈希方案需要什么才能获得良好的性能？

好哈希函数的理想性质

高效可计算。 h 计算速度快。

- h 不应该太复杂，因为这会影响整体运行时间。我们将计算 $h(x)$ 的时间视为常数。

碰撞少。 碰撞 $\leftrightarrow h(u) = h(v)$ 但 $u \neq v$

- 碰撞影响查找和删除的时间。
- 在 $\Theta(\sqrt{n})$ 次随机插入后碰撞不可避免（生日悖论），但我们希望碰撞少，使得键能够很好地分散。
- 注意，无论链的长度如何，插入都需要 $O(1)$ 时间，而查找和删除需要的时间与链的长度成线性关系。给定良好分散的性质，当 $m = |S| = O(n)$ 时，查找项 x 的时间为 $O(1)$ 。

低复杂度确定性哈希函数

我们将低复杂度且确定性的哈希函数（特别是那些没有经过良好分析的）称为临时哈希函数。

```
int hash(String s, int n) {  
    int hash = 0;  
    for (int i = 0; i < s.length(); i++)  
        hash = (31 * hash) + s[i];  
    return hash % n;  
}
```

Java 字符串库的哈希函数

很多临时函数在实践中通常工作良好，但无法提供良好的最坏情况保证。

临时哈希的坏消息

如果 $|U| \geq mn$ (其中 $|S| = m$)，临时哈希在对抗性设置下无法提供任何保证。

- 鸽巢原理 + 确定性 \Rightarrow 至少有一个桶有 m 个元素
- 低复杂度 \Rightarrow 容易求逆：对手可能选择 S 恰好是这样的 m 个元素（一个桶的原像）
- 因此，所有数据键都落在同一个桶中，使哈希变得无用 \rightsquigarrow 查找、删除复杂度为 $O(|S|)$

什么时候我们不能接受临时哈希函数？

明显的情况：飞机控制、核反应堆、心脏起搏器

令人惊讶的情况：拒绝服务攻击

- 恶意对手了解你的临时哈希函数（例如，通过阅读 Java API）并导致单个槽中的大量堆积，使性能停滞

精炼我们的目标

对于关键用途，我们期望无条件/信息论保证，而不是依赖于计算假设的保证。

理想哈希性能

理想哈希函数。 将任意 m 个元素（可能是动态的）**均匀随机**映射到 n 个哈希槽。

- 这种要求保证了期望碰撞数较低，即使数据是由对手选择的。
-
- 所有操作的运行时间取决于链的长度
 - 链的平均长度 $= \alpha = m/n$
 - 选择 $n \approx m \Rightarrow$ 期望每次插入、查找或删除为 $O(1)$

尝试。 从所有将 U 映射到 \mathbb{Z}_n 的哈希函数中随机选择一个函数（真随机）。对手知道所有哈希函数的集合，但不知道你做出的随机选择。

挑战。 所有函数的集合太大 \rightsquigarrow 甚至无法高效采样

通用哈希 (Carter-Wegman 1980s)

真随机是过度的。较弱的性质可能就足够了，并且允许较小的哈希函数集。

通用哈希函数族是一组将全集 U 映射到集合 $\{0, 1, \dots, n-1\}$ 的哈希函数 H ，满足：

- 对于任意一对元素 $u \neq v$:

$$\Pr_{h \leftarrow H} [h(u) = h(v)] \leq 1/n$$

- 可以高效地随机采样 h 。
 - 可以高效地计算 $h(u)$ 。
-

直觉。当 $h \leftarrow H$ 时，全集中的任意两个键碰撞的概率至多为 $1/n$ 。

为什么不用其他定义？

替代定义 1

$$\forall u \in U, \forall y \in \mathbb{Z}_n, \quad \Pr_{h \leftarrow H} [h(u) = y] \leq 1/n$$

为什么不用其他定义？

替代定义 1

$$\forall u \in U, \forall y \in \mathbb{Z}_n, \quad \Pr_{h \leftarrow H} [h(u) = y] \leq 1/n$$

问题

- 未能捕捉理想随机函数直觉背后的**抗碰撞性**
- 病态例子：** 容易构建满足上述定义的 H ，但每个 $h_i \in H$ 将所有输入映射到相同的值 i
 \rightsquigarrow 完全无用

为什么不用其他定义？

替代定义 1

$$\forall u \in U, \forall y \in \mathbb{Z}_n, \quad \Pr_{h \leftarrow H} [h(u) = y] \leq 1/n$$

问题

- 未能捕捉理想随机函数直觉背后的**抗碰撞性**
- 病态例子：** 容易构建满足上述定义的 H ，但每个 $h_i \in H$ 将所有输入映射到相同的值 i
 \rightsquigarrow 完全无用

替代定义 2：正则性

$$\forall y \in \mathbb{Z}_n, |h^{-1}(y)| \text{ 大致相同}$$

为什么不用其他定义？

替代定义 1

$$\forall u \in U, \forall y \in \mathbb{Z}_n, \quad \Pr_{h \leftarrow H} [h(u) = y] \leq 1/n$$

问题

- 未能捕捉理想随机函数直觉背后的**抗碰撞性**
- 病态例子：** 容易构建满足上述定义的 H ，但每个 $h_i \in H$ 将所有输入映射到相同的值 i
 \rightsquigarrow 完全无用

替代定义 2：正则性

$$\forall y \in \mathbb{Z}_n, |h^{-1}(y)| \text{ 大致相同}$$

问题

- 正则性是相对于整个（且固定的）定义域定义的，而不是动态的。

(非) 通用哈希的例子

$$U = \{a, b, c, d, e, f\}, \quad n = 2$$

非通用哈希	H	a	b	c	d	e	f
	$h_1(x)$	0	1	0	1	0	1
	$h_2(x)$	0	0	0	1	1	1

$$H = \{h_1, h_2\}, \quad h \stackrel{R}{\leftarrow} H$$

$$\Pr[h(a) = h(b)] = 1/2$$

$$\Pr[h(a) = h(c)] = 1$$

$$\Pr[h(a) = h(d)] = 0$$

...

通用哈希	H	a	b	c	d	e	f
	$h_1(x)$	0	1	0	1	0	1
	$h_2(x)$	0	0	0	1	1	1
	$h_3(x)$	0	0	1	0	1	1
	$h_4(x)$	1	0	0	1	1	0

$$H = \{h_1, h_2, h_3, h_4\}, \quad h \stackrel{R}{\leftarrow} H$$

$$\Pr[h(a) = h(b)] = 1/2$$

$$\Pr[h(a) = h(c)] = 1/2$$

$$\Pr[h(a) = h(d)] = 1/2$$

$$\Pr[h(a) = h(e)] = 1/2$$

$$\Pr[h(a) = h(f)] = 0$$

关键引理

定理

设 H 为将全集 U 映射到集合 \mathbb{Z}_n 的通用哈希函数族, 设 $h \xleftarrow{R} H$; 则对于任意大小为 m 的集合 $S \subseteq U$, 对于任意 $u \in U$ (例如, 我们可能想要查找的), u 与 S 中其他元素碰撞的期望次数至多为 $|S|/n$ 。

证明。 设 C_u 为一个随机变量, 计数由 $h \xleftarrow{R} H$ 引起的与 u 的碰撞总数。对于任意 $s \in S$, 定义随机变量 $C_{us} = 1$ 如果 $h(s) = h(u)$, 否则为 0。

$$\begin{aligned}\mathbb{E}[C_u] &= \mathbb{E}\left[\sum_{s \in S} C_{us}\right] && // C_u, C_{us} \text{ 的定义} \\ &= \sum_{s \in S} \mathbb{E}[C_{us}] && // \text{期望的线性性} \\ &= \sum_{s \in S} \Pr[C_{us} = 1] && // C_{us} \text{ 是 0-1 变量} \\ &\leq \sum_{s \in S} \frac{1}{n} = \frac{|S|}{n} && // \text{通用性}\end{aligned}$$

字典构造的应用

我们现在立即得到以下推论。

推论

如果 H 是通用的，则对于任意元素 u 的插入、查找和删除操作，在系统中任何时刻最多有 m 个元素的情况下，对于随机 $h \xleftarrow{R} H$ ，操作的期望总复杂度仅为 $O(m/n)$ （将计算 h 的时间视为常数）。

证明。 根据上述定理，对于任意给定的 u ，期望碰撞次数以 m/n 为界，链的平均长度也是如此 \Rightarrow 所有操作的期望复杂度以 $O(m/n)$ 为界。

问题：我们能否真正构造一个通用的 H ？

通用哈希函数族的构造和应用

答案是肯定的。

许多通用族是已知的（用于哈希整数、向量、字符串），它们的求值通常非常高效，并且它们的安全性是**无条件的**（不依赖于任何假设）。

通用哈希在计算机科学中有众多用途：

- 哈希表的实现
- 随机算法
- 密码学

设计通用哈希函数族

模数。 我们使用素数 p 作为哈希表的大小。

输入编码。 假设 $U \subseteq [0, p^\ell - 1]$ 对于某个整数 ℓ 。我们可以将每个元素 $u \in U$ 识别为 ℓ 位的基 p 整数: $u = \mathbf{x} = (x_1, x_2, \dots, x_\ell)$ 。

或者, 我们将每个元素 u 视为 \mathbb{Z}_p 上的 ℓ 维向量。

哈希函数族。 设 $\mathbf{a} = (a_1, a_2, \dots, a_\ell)$ 为函数索引或密钥, 设 $A = \mathbb{Z}_p^\ell$ 。

定义 $H = \{h_{\mathbf{a}} : \mathbf{a} \in A\}$, 其中 $h_{\mathbf{a}}$ 定义为:

$$h_{\mathbf{a}}(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod p = \left(\sum_{i=1}^{\ell} a_i x_i \right) \mod p$$

将全集 U 映射到集合 $\{0, 1, \dots, p-1\}$

构造的证明

定理

$H = \{h_{\mathbf{a}} : \mathbf{a} \in A\}$ 是一个通用哈希函数族。

证明。 设 $\mathbf{x} = (x_1, x_2, \dots, x_\ell)$ 和 $\mathbf{y} = (y_1, y_2, \dots, y_\ell)$ 是 U 的两个不同元素。证明

$\Pr[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})] \leq 1/p$, 其中 $\mathbf{a} \xleftarrow{R} \mathbb{Z}_p^\ell$ 。

由于 $\mathbf{x} \neq \mathbf{y}$, 存在至少一个索引 j 使得 $x_j \neq y_j$ 。

我们有 $h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})$ 当且仅当

$$\underbrace{a_j(y_j - x_j)}_z \equiv \underbrace{\sum_{i \neq j} a_i(x_i - y_i)}_m \pmod{p}$$

- 假设 \mathbf{a} 是通过先选择所有 $i \neq j$ 的 a_i , 然后随机选择 a_j 来均匀随机选择的。
- 由于 p 是素数, $z \neq 0$ 在乘法群 \mathbb{Z}_p^* 中可逆 $\Rightarrow a_j z \equiv m \pmod{p}$ 在 p 种可能性中恰好有一个解
- 因此 $\Pr[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})] \leq 1/p$ 。

通用哈希：总结

目标。 给定全集 U ，维护子集 $S \subseteq U$ ，使得插入、删除和查找是高效的。

使用通用哈希构建哈希表，选择 p 使得 $m \leq p \leq 2m$ ，其中 $m = |S|$

- 事实：在 m 和 $2m$ 之间至少存在一个素数，可以通过另一个随机算法找到。

结果

- 使用空间： $\Theta(m)$ 。
- 每次操作的期望碰撞次数 $\leq 1 \Rightarrow$ 每次插入、删除或查找 $O(1)$ 时间

讨论

问：为什么不直接使用密码学哈希函数？

- 实际哈希如 SHA 和基于 AES 的哈希本质上不是随机化的 \rightsquigarrow 任何形式的均匀性都是启发式的
- 输出长度通常至少 128 位 \rightsquigarrow 对于应用可能太大；截断输出可能会损害抗碰撞性

问：为什么不使用伪随机函数？

- 完整的伪随机性对于均匀哈希目的是过度的
- 太强 \rightsquigarrow 使用起来很重

密码学哈希和伪随机函数都不提供信息论安全性。

总结

为什么通用哈希给我们均匀性保证？

$$(h, h(u)) \approx_s (h, y \stackrel{R}{\leftarrow} \mathbb{Z}_n), \text{ 其中 } h \stackrel{R}{\leftarrow} H, u \stackrel{R}{\leftarrow} S$$

- 这正是剩余哈希引理告诉我们的：当输入在 S 上均匀随机分布（具有高最小熵）时，输出在 \mathbb{Z}_n 上均匀随机分布。

通用哈希函数不捕捉任何形式的独立性，成对独立哈希捕捉相互独立性。

通用哈希函数只捕捉弱版本的抗碰撞性：小定义域上的抗碰撞性（通过联合界）

- Bazinga：我们可以使用有损函数将通用哈希函数编译为抗碰撞哈希函数
- 这种弱形式的抗碰撞性意味着在任何集合（如 S ）上定义的均匀性 \rightsquigarrow 足以应用于字典

澄清

我们强调基于通用哈希函数的字典适用于关键用途，因为它提供统计保证

- **警告：** 在期望意义/平均情况下

对于非关键用途，我们可以使用实际的密码学哈希代替通用哈希函数

当输入不是对抗性选择时，我们甚至可以使用轻量级哈希（可能是非密码学的）代替

- 例如，C++ 中的 `unordered set` 由 Murmurhash 驱动

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

随机化. 允许公平抛掷硬币

- 在实际中，访问伪随机数生成器（PRG）

随机化的好处：可以产生简单、快速的算法，或者是某些问题的唯一已知算法。

例子. 图算法、快速排序、快速选择、哈希、密码学

BPP 与 ZPP

双边误差.

- BPP . [蒙特卡洛] 可以在多项式时间内以双边误差求解的问题类。(名称来源于著名的赌场度假胜地)

单边误差.

- RP . 若正确答案是是, 则以 $\geq 2/3$ 的概率返回是。若正确答案是否, 则总是返回否。
- $co-RP$. 若正确答案是否, 则以 $\geq 2/3$ 的概率返回否。若正确答案是是, 则总是返回是。

可以将错误概率降低到可忽略

零边误差.

- ZPP . [拉斯维加斯] 可以在期望多项式时间内求解的问题类。

与其他复杂性类的关系

定理

$$\mathcal{P} \subseteq \mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{NP}$$

核心开放问题

- $\mathcal{P} = \mathcal{BPP}$? 这个问题涉及随机化能在多大程度上提供帮助。
 - ▶ 许多复杂性理论学家实际上相信 $\mathcal{P} = \mathcal{BPP}$ 。
 - ▶ 换言之，存在一种方法可以将每个概率算法转换为确定性算法（不抛掷任何硬币），同时仅导致多项式级别的减速。

随机性的作用远远超出对随机算法和 \mathcal{BPP} 等复杂性类的研究。

- 整个密码学领域，包括加密、签名、交互式证明和概率可检验证明，都以本质性的方式依赖于随机性

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

随机快速排序

标准快速排序.

总是选择第一个元素作为枢轴。其结果是最坏情况复杂度（当输入数组已经有序时）为 $O(n^2)$

$$W(n) = W(0) + W(n-1) + O(n) \Rightarrow W(n) = O(n^2)$$

随机快速排序. 每次随机选择一个元素 x 作为枢轴。显然， x 是第 i 个元素的概率为 $1/n$ ，从而将数组分成三部分： $[0, i-1]$, $[i]$, $[i+1, n]$ ：

$$\begin{aligned} T(n) &= \frac{1}{n} \left(\sum_{i=0}^{n-1} (T(i) + T(n-1-i)) \right) + (n-1) \\ &= \frac{2}{n} \sum_{i=0}^{n-1} T(i) + (n-1) \\ &= \Theta(n \ln n) \end{aligned}$$

随机快速选择 (1/2)

标准快速选择.

首先将所有元素分成大小为 5 的组，选出它们的中位数，然后从这些中位数中选出中位数 m^* ，使用 m^* 作为枢轴来划分数组。

$$W(n) = W(7n/10) + W(n/5) + O(n) \Rightarrow W(n) = \Theta(n)$$

随机快速选择. 随机选择一个元素作为枢轴。

- **最好情况:** 持续选中中位数，得到理想情况 $|S_L|, |S_R| \approx |S|/2$
 $\rightsquigarrow T(n) = T(n/2) + \Theta(n)$ 。
- **最坏情况:** 我们持续选择 m^* 为最大（或最小）元素，从而每次只将数组缩小一个元素
 $\rightsquigarrow T(n) = \Theta(n^2)$ 。

在从 $\Theta(n)$ 到 $\Theta(n^2)$ 的这个范围中，平均运行时间处于什么位置？
幸运的是，它非常接近最好情况的时间。

随机快速选择 (2/2)

运行时间取决于 m^* 的随机选择。

- 为了区分 m^* 的幸运和不幸运的选择，如果 m^* 落在数组的第 25 到第 75 百分位数之间，我们称其为**好的**。我们喜欢这些 m^* 的选择，因为它们确保子列表 S_L 和 S_R 的大小最多是 S 的 $3/4$ ，从而数组显著缩小。
- 好的 m^* 很丰富：任何列表中有一半的元素必定落在第 25 到第 75 百分位数之间。

差

好

差

随机选择并检查：平均两次尝试我们将获得一个好的 m^* ：

$$T(n) \leq T(3n/4) + \Theta(n) \Rightarrow T(n) = \Theta(n)$$

- 期望多项式时间**。对于任何输入，随机算法平均在线性步数后返回正确答案。

简要总结

随机快速排序实际上使用随机性来均摊最坏情况。现在平均复杂度不仅依赖于输入实例的分布，而且可以借助随机性仅与任何固定实例相关联。

随机快速选择使用随机性来设计一种更简单的枢轴选择方法。

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

素性检验

素性检验.

给定一个整数 N ，判断它是否是素数。

素性检验算法早在计算机出现之前就被追求，因为数学家需要它们来验证各种猜想。

- 理想情况下，我们希望得到在 N 的表示大小的多项式时间内运行的高效算法，即 $\text{poly}(\log N)$ 。
- 几个世纪以来，数学家们对这个问题不知道任何这样的高效算法。
- 在 1970 年代，高效的概率素性检验算法被发现，这是概率算法威力的首批证明之一。
- 2004 年，Agrawal、Kayal 和 Saxena 给出了素性检验的确定性多项式时间算法（突破性进展）

Miller-Rabin 素性检验

Miller-Rabin 算法的关键是找到一个区分素数和合数的性质。

事实 1

若 N 是素数，则 $\forall a \in \{1, \dots, N-1\}$ 有 $a^{N-1} \equiv 1 \pmod{N}$

这建议通过选择一个均匀随机元素 a 并检验 $a^{N-1} \stackrel{?}{=} 1 \pmod{N}$ 来测试 N 是否是素数。

- 若 $a^{N-1} \not\equiv 1 \pmod{N}$ ，则 N 不可能是素数。我们称任何这样的 a 为 N 是合数的证据。
- 反过来，我们可能希望如果 N 不是素数，那么一个随机选择的 a 成为合数证据的概率是合理的，因此通过多次重复这个测试，我们可以高置信度地确定 N 是否是素数。

证据的分布

回顾：模 N 的指数运算和从 $[N - 1]$ 中选择均匀随机元素都可以在多项式时间内完成。看起来问题已经解决了。

证据的分布

回顾：模 N 的指数运算和从 $[N-1]$ 中选择均匀随机元素都可以在多项式时间内完成。看起来问题已经解决了。

如果 N 是合数， $\{1, \dots, N-1\}$ 中有多少个证据？

- 若 $a \notin \mathbb{Z}_N^*$ ，则 $a^{N-1} \neq 1 \pmod N$ （若 $\gcd(a, N) \neq 1$ 则 $\gcd(a^{N-1}, N) \neq 1$ ） $\Rightarrow a$ 是 N 为合数的证据。但是， \mathbb{Z}_N^* 之外的证据很少。实际上，如果你找到一个 $a \notin \mathbb{Z}_N^*$ ，你可以分解 N 。
- 因此我们将注意力限制在 \mathbb{Z}_N^* 中的证据 a 。

证据的分布

回顾：模 N 的指数运算和从 $[N-1]$ 中选择均匀随机元素都可以在多项式时间内完成。看起来问题已经解决了。

如果 N 是合数， $\{1, \dots, N-1\}$ 中有多少个证据？

- 若 $a \notin \mathbb{Z}_N^*$ ，则 $a^{N-1} \not\equiv 1 \pmod{N}$ （若 $\gcd(a, N) \neq 1$ 则 $\gcd(a^{N-1}, N) \neq 1$ ） $\Rightarrow a$ 是 N 为合数的证据。但是， \mathbb{Z}_N^* 之外的证据很少。实际上，如果你找到一个 $a \notin \mathbb{Z}_N^*$ ，你可以分解 N 。
- 因此我们将注意力限制在 \mathbb{Z}_N^* 中的证据 a 。

通过两个简单的群论引理，我们得出：

事实 2

如果存在一个证据表明 N 是合数，则 \mathbb{Z}_N^* 中至少一半的元素是 N 为合数的证据。

然而，上述结论没有给出完整的解决方案，因为存在无穷多个没有任何证据的合数 N 。这样的 N 被称为 *Carmichael* 数。

对上述检验的改进

幸运的是，上述检验的一个改进可以对所有 N 有效。

- **关键思想：**放宽证据的标准 \rightsquigarrow 规避“无证据”问题。
- 具体地，令 $a^{N-1} \equiv 1 \pmod N$ 可以作为 N 是合数的证据，同时施加更细粒度的要求

令 $N-1 = 2^r u$ ，其中 u 是奇数且 $r > 1$ 。之前展示的算法只测试 $a^{N-1} = a^{2^r u} \equiv 1 \pmod N$ 。

让我们观察 $r+1$ 个值的序列：

$$a^u, a^{2u}, \dots, a^{2^r u} \quad \text{全部模 } N$$

序列中的每一项都是前一项的平方。如果某个值等于 ± 1 ，则所有后续值都将等于 1。

强证据

合数的强证据.

$a \in \mathbb{Z}_N^*$ 是 N 为合数的强证据, 如果

(1) $a^u \not\equiv \pm 1 \pmod{N}$ 且 (2) $a^{2^i u} \not\equiv -1 \pmod{N}$ 对所有 $i \in \{1, \dots, r-1\}$

换言之, 当元素 a 不是强证据时, 序列 $(a^u, a^{2u}, \dots, a^{2^r u})$ 具有以下形式之一:

$$(1, 1, \dots, 1, 1) \quad \text{或} \quad (\star, \dots, \star, -1, 1, \dots, 1, 1)$$

回顾: $f(x) = x^2 \pmod{N}$ 在 \mathbb{Z}_N^* 上是一个 4 对 1 的映射, 其中 $N = p \cdot q$.

整合所有内容

定理

设 N 是一个奇数且不是素数幂。则 \mathbb{Z}_N^* 中至少一半的元素是 N 为合数的强证据。

Algorithm 1 Miller-Rabin 素性检验

输入 $N > 2$

- 1: **if** N 是偶数 **then**
- 2: **return** "合数"
- 3: **end**
- 4: **if** N 是完全幂 **then**
- 5: **return** "合数"
- 6: **end**
- 7: 计算 $r > 1$ 和奇数 u 使得 $N - 1 = 2^r u$
- 8: **for** $j = 1$ **to** t **do**
- 9: $a \xleftarrow{R} \{1, \dots, N - 1\}$

目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- **Schwartz-Zippel 引理及其应用**
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

Schwartz-Zippel 引理

DeMillo-Lipton-Schwartz-Zippel 引理

设 $P \in \mathbb{F}[x_1, \dots, x_n]$ 是一个总次数 $d \geq 0$ 的非零多项式, 设 $\alpha_1, \dots, \alpha_n$ 独立且均匀随机地从 $S \subset \mathbb{F}$ 中选取。则:

$$\Pr[P(\alpha_1, \dots, \alpha_n) = 0] \leq \frac{d}{|S|}$$

在单变量情形下, 这直接来自于 d 次多项式至多有 d 个根这一事实。

单项式的次数是单项式中变量指数之和; 多项式的次数是其中任何单项式的最大次数。

Schwartz-Zippel 引理的应用

- 概率多项式恒等检验 \Rightarrow SNARK
- 全域哈希的分析
- 素性检验

两个多项式的比较

多项式恒等检验：给定有限域 \mathbb{F} 上的两个多项式 $C(x)$ 和 $D(x)$ ，判断它们是否相等。

应用：分支程序、概率可检验证明

为什么这个问题不平凡？

看起来我们可以逐一检查系数 \Rightarrow 复杂度为 $O(n)$ ，其中 $n = \max\{\deg(C), \deg(D)\}$ 。

问题的细化

上述检查仅当系数以明文形式给出时才有效。我们需要考虑以下两种情况。

情况 1: 多项式的表示很复杂，展开可能代价过高

- 例 1. $D(x)$ 的形式是 $A(x) \cdot B(x)$ ，要得出 $D(x)$ 的系数需要多项式乘法
 - ▶ 朴素算法：通过卷积计算系数 $\Theta(n^2)$
 - ▶ FFT: $\Theta(n \log n)$

情况 2: 多项式的表示是隐藏的（可能出于隐私考虑），算法只能访问求值预言机

我们能否不用朴素的比较方法来解决这个问题？

答：是的。随机性可以帮忙！

多项式恒等检验

多项式恒等检验[出人意料地简单]

- 随机选取 $\alpha \xleftarrow{R} \mathbb{F}$, 如果 $C(\alpha) = A(\alpha) \cdot B(\alpha)$ 则输出“是”, 否则输出“否”。

分析

- 复杂度: 给定系数表示, 多项式求值的代价是 $3 \cdot \Theta(n)$, 比较的代价是 $O(1) \rightsquigarrow$ 总复杂度为 $\Theta(n)$
- 当 $C(x) = A(x) \cdot B(x)$ 时: 算法的输出总是“是”
- 当 $C(x) \neq A(x) \cdot B(x)$ 时: Schwartz-Zippel 引理 \Rightarrow 算法的输出以至少 $1 - n/|\mathbb{F}|$ 的概率为“否”

矩阵恒等检验

问题. 给定 \mathbb{F}_p 上的三个 $n \times n$ 矩阵 A 、 B 和 C , 其中 $p > n^2$ 是素数。检验 $A \times B = C$ 是否成立。

朴素解法. 计算 $A \times B$ 然后将结果与 C 比较。复杂度归结为矩阵乘法, 即 $O(n^{2.372})$ 。

思路. 使用随机化检验。主要工具是全域哈希。直观地, 它可以将大对象压缩成小指纹。但是如何做? 让我们首先回顾全域哈希的构造。

全域哈希，再访

我们的目标：构建一族从 $\mathbb{F}_p^n \rightarrow \mathbb{F}_p$ 的全域哈希。

之前的构造：

$$h_{\mathbf{a}}(\mathbf{x}) = a_0x_0 + \cdots + a_{n-1}x_{n-1}, \quad \mathbf{a}, \mathbf{x} \in \mathbb{F}_p^n$$

- 将 \mathbf{x} 解释为系数向量， \mathbf{a} 解释为变量向量，Schwartz-Zippel 引理（多变量情形） \Rightarrow 碰撞概率小于 $1/|\mathbb{F}_p|$ ；

一个更简单的构造（这正是 Reed-Solomon 码）

$$h_a(x) = a^1x_0 + \cdots + a^nx_{n-1}, \quad a \in \mathbb{F}_p, \mathbf{x} \in \mathbb{F}_p^n$$

- 将 \mathbf{x} 解释为系数向量（消息）， a 解释为单变量，Schwartz-Zippel 引理（单变量情形） \Rightarrow 碰撞概率小于 $n/|\mathbb{F}_p|$ 。

Freivalds 算法

[Freivalds 算法 1977] 时间复杂度 $O(n^2)$ 的随机算法

- ① 选取 $r \xleftarrow{R} \mathbb{F}_p$, 设 $\mathbf{x} = (r, r^2, \dots, r^n)$
- ② 计算 $\mathbf{y} = C\mathbf{x}$ 和 $\mathbf{z} = A \cdot B\mathbf{x}$
- ③ 如果 $\mathbf{y} = \mathbf{z}$ 则输出“真”, 否则输出“假”

复杂度分析: 总共 $\Theta(n^2)$

- 生成向量 \mathbf{x} : $\Theta(n)$
- 矩阵-向量乘积: $3 \times \Theta(n^2)$

正确性分析

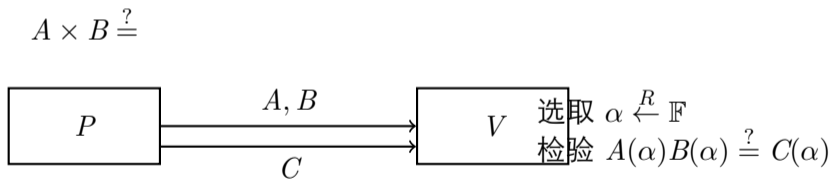
- 当 $C = A \cdot B$ 时: 算法的输出总是“是”
- 当 $C \neq A \cdot B$ 时: 算法的输出以至少 $1 - n/|\mathbb{F}_p|$ 的概率为“否” \Leftarrow Schwartz-Zippel 引理
 - ▶ 至少存在一个 (i, j) 使得 $C_{ij} \neq D_{ij}$, 然后考虑行向量的差作为系数向量
 - ▶ $\langle \mathbf{a}, \mathbf{x} \rangle$ 应该被视为消息 \mathbf{x} 的 Reed-Solomon 编码的随机部分

Freivalds 算法的高层视角

多项式恒等检验和矩阵乘积检验可以通过概率可检验技术推广到可验证计算

随机性的威力

通过随机化检查来验证计算的正确性，而非重新执行



目录

① 概率论预备知识

② 随机数据结构

- 字典

③ 随机算法

- 随机快速排序与快速选择
- 概率素性检验
- Schwartz-Zippel 引理及其应用
 - 多项式恒等检验
 - 矩阵恒等检验

④ 总结

蒙特卡洛与拉斯维加斯算法

基本概率论

从全域哈希构建的随机数据结构

随机算法

- ① **Las Vegas** \leftrightarrow **ZPP**: 保证找到正确答案，很可能在多项式时间内运行（实际上是期望多项式时间）。
 - ▶ 例子：随机快速排序和快速选择
- ② **Monte Carlo** \leftrightarrow **BPP**: 保证在多项式时间内运行，很可能找到正确答案。
 - ▶ 例子：多项式恒等检验、矩阵恒等检验、素性检验

备注. 可以总是将 Las Vegas 算法转换为 Monte Carlo 算法，但没有已知的一般方法将其反向转换。