Lecture 8 随机算法

绳伟光

上海交通大学微纳电子学系

2021-10-28

提纲

提纲

确定性算法

之前所研究的算法都是确定性的，所谓确定性算法，其确定性体现在：

1) 算法每个步骤是确定的，即在某一步到底该做什么是确定的
2) 算法的结果是确定的，即输出的东西是什么事先就知道
3) 步骤的数量是确定的，即算法的时间复杂性是确定的

随机算法简介

随机算法

随机算法使用概率和统计方法在其执行过程中对于下一计算步骤作出随机性的选择。随机算法的随机性主要体现在三个方面：

1) 算法在问题实例上的操作过程是随机的，每次执行的操作可能不同
2) 算法在问题的同一实例上的计算复杂度是一随机变量
3) 算法的输出结果是随机的，可能是正确的，也可能是错误的

随机算法的优势：

- 对于有些问题：算法简单
- 对于有些问题：时间复杂性低
- 对于有些问题：同时兼有简单和时间复杂性低

随机算法的历史及应用

随机算法起源于 20 世纪 40 年代，在曼哈顿工程的研究过程中提出可以通过统计抽样的方法获得问题的近似解；20 世纪 70 年代，随机算法逐渐被引入非数值计算过程中

Michael Rabin

某些问题，如果仅使用确定算法而不采用随机算法，则在可以容忍的时间内无法获得所需的计算结果

随机算法在分布式计算、通信、信息检索、计算几何和密码学领域应用广泛，比如著名的公钥密码 RSA 算法中

随机算法分类

- Monte Carlo 算法
  - 并不总能获得问题的正确 (精确) 解
  - 算法以较高的概率获得正确 (精确) 解，此概率与算法执行时间成正比
  - 不存在有效过程判定算法输出的解是否为问题的正确 (精确) 解
- Las Vegas 算法
  - 一旦找到一个解, 该解一定是正确的
  - 找到解的概率与算法执行时间成正比，增加求解次数可降低求解无效的概率
- Sherwood 算法
  - 一定能够求得一个正确解
  - 确定型算法的最坏与平均复杂度差别大时，加入随机性即得到 Sherwood 算法，从而消除最坏行为与特定实例的联系

随机算法的性能分析及典型算法实例

- 随机算法分析的特征
  - 仅依赖于随机选择，不依赖于输入的分布
  - (*) 确定算法的平均复杂性分析时需要考虑输入的分布
- 随机算法分析的目标
  - 平均时间复杂性：时间复杂性随机变量的均值
  - 获得正确解的概率
  - 获得优化解的概率
  - 解的精确度估计
- 典型算法实例
  - 数值算法：计算 π 值、积分
  - 素数测试算法、最小割
  - N 皇后问题、随机选择
  - 随机排序算法、随机最近点对算法

提纲

Monte Carlo 算法介绍

- 以概率和统计理论方法为基础的一种计算方法，将所求解的问题同一定的概率模型相联系，用计算机实现统计模拟或抽样，以获得问题的近似解
- 由 John von Nermann、Stanislaw Ulam、Nicholas Metropolis、Enrico Fermi 在参与曼哈顿工程的过程中提出 (Ulam 的叔叔总在蒙特卡罗输钱)
- Monte Carlo 算法可分为两类
  - 所求解的问题本身具有内在的随机性，借助计算机的运算能力可以直接模拟这种随机的过程。例如在核物理研究中，分析中子在反应堆中的传输过程
  - 所求解问题可以转化为某种随机分布的特征，比如随机事件的概率，或者随机变量的期望。通过随机抽样的方法，以随机事件出现的频率估计其概率，或者以抽样的数字特征估算随机变量的数字特征，并将其作为问题的解

随机投点法

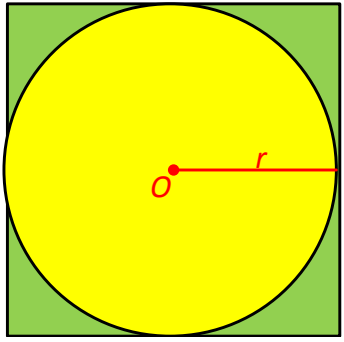
随机投点法利用大数定律完成数值的随机计算，先将待计算的数值与某个随机事件的概率相关联；然后依据大数定律，通过 n 次随机实验测得该随机事件发生的频率，以该频率作为概率得到待计算数值的一个随机近似值

大数定律

设随机事件 A 在每次实验中发生的概率为 p，且 μ_n 是 n 次独立试验中事件 A 发生的次数，则对任意正数 ε 有 $\lim_{n\to+\infty} Pr\left(\left|\frac{\mu_n}{n} - p\right| < \varepsilon\right) = 1$

随机投点采用的随机试验次数越多，所得计算结果的可靠性越高

π 值计算问题



- 上图中内切圆与外接矩形的面积比例为 $\pi r^2/(2r)^2 = \pi r^2/4r^2 = \pi/4$
- 如果向上图的矩形内随机投入 n 个点，则点落入黄色圆形内的概率即为两部分的面积比 $\pi/4$，进而可求出 π 值
- 所求 π 值精度只与投入点数多少相关

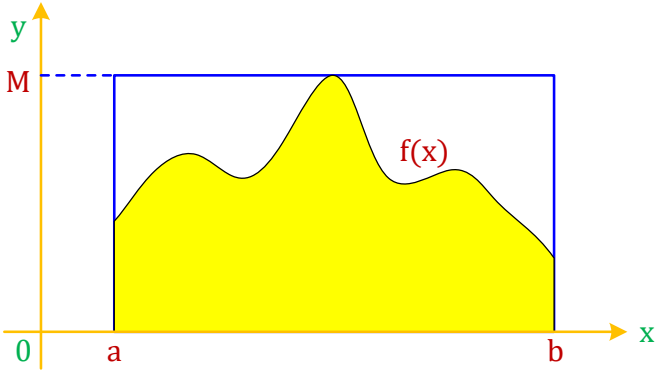
π 值求解的随机算法

PI(n)

1: $k = 0$
2: **for** $i \leftarrow 1$ to n **do**
3: 随机产生矩形内一点 (x, y)
4: **if** $x^2 + y^2 \leq 1$ **then**
5: $k = k + 1$
6: **return** $4k/n$

- 时间复杂性 $O(n)$，其中 n 是随机样本大小
- 解的精确度随着随机样本数增加而增加

定积分计算问题



- 求 $[a,b]$ 区间上非负连续函数 $f(x)$(其中 $0 \le f(x) \le M$) 的定积分 I
- 在长为 $b-a$，宽为 M 的矩形中产生随机点 P，P 位于曲线 $(x,f(x))$ 下方的概率为 $I/[M(b-a)]$，从而可反推出积分 I
- 由大数定律，该概率可通过随机试验中 “P 位于曲线 $(x,f(x))$ 下方的频率来近似”

随机投点定积分计算算法

Calculus-Rand1($f(x), a, b, M, n$)

1: $k \leftarrow 0$
2: **for** $i \leftarrow 1$ **to** n **do**
3: 均匀地随机产生点 (x, y)，其中 $x \in [a, b], y \in [0, M]$
4: **if** $y \leq f(x)$ **then**
5: $k \leftarrow k + 1$
6: **return** $M(b-a) \cdot (k/n)$

- 时间复杂性 $O(n)$，其中 n 是随机样本大小
- 解的精确度随着随机样本数增加而增加

素数测试问题建模

费马小定理

假如 a 是一个整数，p 是一个素数，且 a, p 互素，即 $gcd(a, p) = 1$，则 $a^{p-1} \equiv 1 \pmod{p}$ 成立

- 素数必满足费马小定理，不满足 $a^{p-1} \equiv 1 \pmod{p}$ 的数必不为素数
- 逆命题大多数时候成立，即满足费马小定理的数是素数的可能性大于 1/2(不做证明)
- 令 $a = 2$ 即是一个很好的检测标准，误判的概率非常低，对于随机选取的一个 1024 位数，其是伪素数的概率低于 $1/10^{41}$，比如 341 即是一个伪素数，$2^{341-1} \equiv 1 \pmod{341}$，但 $341 = 11 \times 31$
- 另有一类数对所有与 p 互素的 a 都满足费马小定理，但仍然是合数，这样的数被称为卡迈克尔 (Carmichael) 数，但这样的数很少，比如 561

素数测试的随机算法

- 给定一个数 n，测试 n 是否为素数?
- **利用费马小定理，如果不满足，一定是合数；如果满足，可能是将合数误判为素数，误判概率小于 1/2**
- 用随机算法求解素数测试问题，流程如下：
  - 对 n 进行 m 次随机测试
  - 如果有一次测试不成立, 则回答 n 是合数
  - 如果 m 次测试均成立, 则回答 n 是素数
  - 回答 n 是合数时, 答案百分之百正确
  - 回答 n 是素数时, 答案正确的概率至少是 $1 - 2^{-m}$

素数测试的随机算法伪代码

ISPRIME(n)

1: 随机选择 m 个数 $\{b_1, b_2, \cdots, b_m\}$，$b_1 \le b_2 \le \cdots \le b_m$
2: **for** $i \leftarrow 1$ to m **do**
3: **if** $b_i^{n-1} \neq 1 \pmod{n}$ **then**
4: **return** n 是合数
5: **return** n 是素数

例 1 (测试 $n = 12$，选定测试数集 $\{2, 3, 7\}$)

测试 2：$2^{12-1} = 2048 \neq 1 \ mod\ 12$，所以 n 是合数

例 2 (测试 $n = 11$，选定测试数集 $\{2, 3, 7\}$)

测试 2：$2^{11-1} = 1024 = 1 \ mod\ 11$
测试 3：$3^{11-1} = 59049 = 1 \ mod\ 11$
测试 7：$7^{11-1} = 282475249 = 1 \ mod\ 11$
结论：11 可能是素数，答案正确的概率至少为 $1 - 2^{-3}$

蒙特卡罗算法特征

蒙特卡罗算法特征

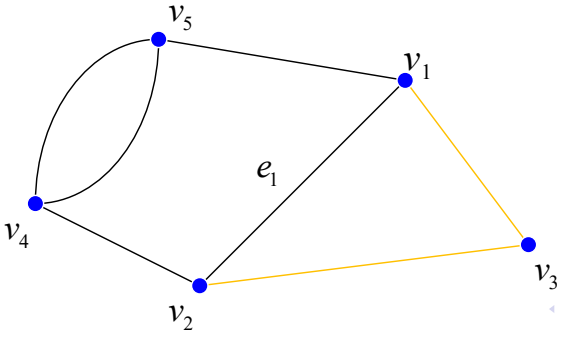
- 算法并不总能获得问题的正确解
- 算法可以较高的概率获得正确解
- 不存在有效过程判定算法输出的解是否为问题的正确解

- 蒙特卡罗算法的正确性：如果蒙特卡罗算法获得正确解的概率为 p，$p > 0$，则称蒙特卡罗算法是 p-正确的。
- 蒙特卡罗算法的一致性：如果蒙特卡罗算法在问题同一实例上的多次运行，不会输出问题的不同的正确解，则称该蒙特卡罗算法是一致的。

素数测试算法 ISPRIME 是一个一致的 $1 - 2^{-m}$-正确的蒙特卡罗算法！

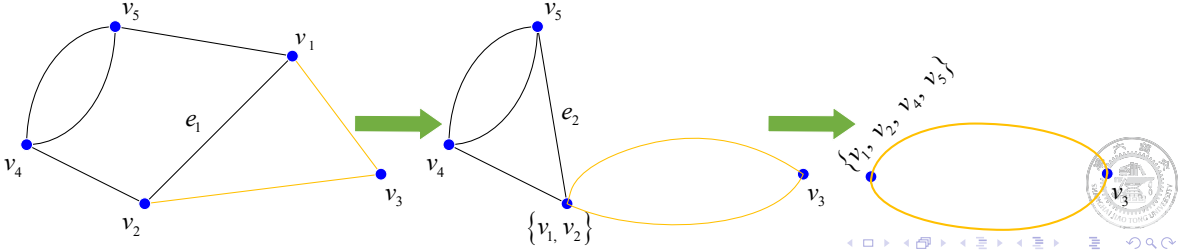
最小割问题

- 图 $G=(V,E)$ 称为多重图，如果顶点对之间允许出现多条边
- 多重图 $G=(V,E)$ 的一个割是边集 E 的一个子集 C，它使得 $G_C=(V,E-C)$ 至少存在两个连通分支
- 边的条数 $|C|$ 达到最小值的割称为最小割
- 最小割问题要求输出多重图 G=(V,E) 的最小割
- 下图的最小割为黄色的两条边 $\{(v_1,v_3),(v_2,v_3)\}$



最小割随机算法思路

- 对多重图 $G = (V, E)$，均匀随机地选择 G 中的一条边 (u, v)，并将其收缩为一个新的顶点 x_{uv}
- 收缩操作：收缩边 (u, v)，将 u, v 合并为一个新顶点 x_{uv}，删除边 (u, v)，原来与 u, v 连接的边改为与 x_{uv} 连接，收缩过程可能产生重边
- 收缩过程持续到图中只剩下两个顶点，介于此两个顶点间的边集即为最小割集



最小割随机算法

RandMinCut(G)

1: $G' \leftarrow G$
2: **while** G' 中顶点数 >2 **do**
3: 随机选取 G' 的一条边 (u, v)，收缩 (u, v)
4: G' 中剩下的两个顶点对应 G 中的顶点子集 S 和 $V - S$
5: **return** $C = \{(u, v) | u \in S, v \in V - S, (u, v) \in E\}$

复杂度分析

while 循环会执行 $n-2$ 遍，每次 while 循环的时间复杂度为 $O(n)$；最后一步计算 C 的开销需 $O(n^2)$，因此算法的时间复杂度为 $O(n^2)$。

获得最小割的概率 1

引理 1

设 $G = (V, E)$ 是一个多重图，C 是 G 的最小割且 $|C| = k$，则 $|E| \geq kn/2$

证 由 $|C| = k$ 可知，G 中任意顶点 u 的度 $d_u \geq k$。否则，与顶点 u 关联的所有边将构成一个大小小于 k 的割，这与最小割的大小为 k 矛盾。因此，$2|E| = \sum_{u \in V} d_u \geq kn$。 □

收缩是安全的

引理 2

设 $G = (V, E)$ 是一个最小割大小为 k 的多重图，$G' = (V', E')$ 是在 G 上收缩边 (u, v) 后得到的图，则 G' 的最小割至少包含 k 条边

获得最小割的概率 2

证 反证法，设 G' 的最小割 C' 且 $|C'| < k$；记收缩边 (u, v) 产生的新顶点为 x。

1) 如 C' 中任意边不以 x 为端点，则 $C' \subseteq E$。同时在 G 和 G' 中删除边集 C' 将产生相同数量的连通分支，其中边 (u, v) 所在连通分支与 x 所在连通分支对应，其余连通分支相同，意味着 C' 也是 G 的一个割，与 G 的最小割大小为 k 矛盾。
2) 如 C' 中存在以 x 为端点的边。由于收缩过程保留了边的重数，执行收缩过程的逆过程，将 C' 中以 x 为端点的任意 (重边)(x, y) 替换为 (u, y)(和) 或 (v, y)，得到 G 的边集 C，满足 $|C| = |C'|$。此时，在 G 中删除边集 C 和在 G' 中删除边集 C' 将产生相同数量的连通分支，其中边 (u, v) 所在的连通分支与 x 所在连通分支对应，其余连通分支相同。这意味着 C 是 G 的一个割，与 G 的最小割大小为 k 矛盾。

获得最小割的概率 3

引理 3

RandMinCut 算法输出多重图 $G=(V,E)$ 最小割的概率大于 $2/n^2$，$n=|V|$

证 设 C 是图 $G=(V,E)$ 的最小割且 $|C|=k$。由 RandMinCut 算法的操作过程可知，算法输出 C 当且仅当 C 中的边在算法执行中没有被收缩过。因此，仅需计算 C 中所有边都没有被收缩的概率。为此，定义随机事件 X_i 表示“算法第 i 次执行收缩操作时未选择 C 中的边”，其中 $1 \le i \le n-2$。记第 i 次收缩操作后得到的图为 $G_i=(V_i,E_i),\ i \in [1,n-3]$。则由前述引理，

$$Pr(X_1)=1-|C|/|E| \ge 1-2/n \quad (\text{已知} |E| \ge kn/2)$$

$$Pr(X_2|X_1)=1-|C|/|E_1| \ge 1-2/(n-1)$$

$$Pr(X_{i+1}|X_1 \cap \cdots \cap X_i)=1-|C|/|E_{i+1}| \ge 1-2/(n-i)$$

获得最小割的概率 4

证 (续) 反复运用条件概率公式 $Pr(A \cap B) = Pr(A|B) \cdot Pr(B)$，可得：

$$
\begin{aligned}
Pr(X_1 \cap \cdots \cap X_{n-2}) &= Pr(X_{n-2}|X_1 \cap \cdots \cap X_{n-3}) \\
&\cdot Pr(X_{n-3}|X_1 \cap \cdots \cap X_{n-4}) \cdot \cdots \cdot Pr(X_2|X_1) \cdot Pr(X_1) \\
&\geq \frac{1}{3} \cdot \frac{2}{4} \cdot \frac{3}{5} \cdot \cdots \cdot \frac{n-4}{n-2} \cdot \frac{n-3}{n-1} \cdot \frac{n-2}{n} \\
&= \frac{2}{n(n-1)} > \frac{2}{n^2}
\end{aligned}
$$

RandMinCut 是一个 $2/n^2$-正确的蒙特卡罗算法。假如运行算法 $n^2/2$ 遍，输出发现的最小割，则该割仍然不是问题正确解的概率不超过 $(1-2/n^2)^{n^2/2} < e^{-1}$。该算法效率较低！

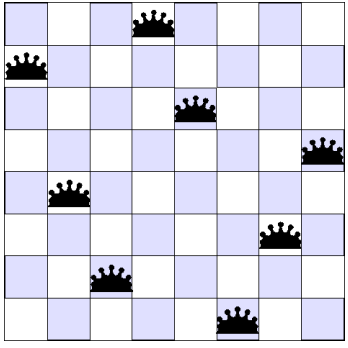
提纲

Las Vegas 算法 vsMonte Carlo 算法

- Las Vegas 算法：通常采用 bool 型变量来表示拉斯维加斯算法的返回值。当算法找到一个解时返回 true，否则 false。当返回 false 时，说明未得到解，那么可再次独立调用该算法，在时间允许的情况一直运算到得出解为止
- 拉斯维加斯算法的一个显著特征是它所作的随机性决策有可能导致算法找不到所需的解，但只要找到一个解，那么该解一定是正确的
- Monte Carlo 算法一定会找到一个解，但该解可能精度成问题，但只要花足够的时间，一定可以找到足够精确的解

N 皇后问题简介

- N 皇后问题是八皇后问题的推广，八皇后问题则最早由国际西洋棋棋手马克斯·贝瑟尔于 1848 年提出。之后高斯和康托尔将其推广为更一般的 N 皇后问题：如何能够在 $n \times n$ 的国际象棋棋盘上放置 n 个皇后，使得任两个皇后都不能处于同一条横行、纵行或斜线上？



- Las Vegas 算法求解 N 皇后问题：在棋盘上相继的各行中随机地放置皇后，并注意使新放置的皇后与已放置的皇后互不攻击，直至 n 个皇后已相容地放置好，或已没有下一个皇后的可放置位置时为止。注意这里解决的是找到其中一个解，求不是求出 N 皇后的全部解。

元素选择问题

元素选择问题

输入： 含有 n 个互异实数的集合 S 及正整数 k，$k \in [1, n]$

输出： S 中第 k 小的元素 $\min(S, k)$

随机算法思路

- 基本思想是利用排序操作来找出 $\min(S, k)$
- 为了使排序操作可以在线性时间内完成，只排序 S 中部分元素，即采用抽样的方法

$$1 \cdots l = x - \sqrt{n} \cdots x = kn^{-1/4} \cdots h = x + \sqrt{n} \cdots n^{3/4}$$

随机选择算法的操作过程

- 第一阶段：均匀、独立、可放回地执行 $n^{3/4}$ 次抽样，从 S 中抽取元素存入 R，然后将 R 在 $O(n)$ 时间内排序
- 第二阶段：若 $\min(S,k) \in R$(可能不成立)，则其大致位于排序后 R 的第 $x = (k/n)n^{3/4}$ 位置。由于 R 系由随机抽样所得，上述位置存在误差，因此扩大查找区间，令 $l = \max\{\lfloor x - \sqrt{n} \rfloor, 1\}$，$h = \min\{\lfloor x + \sqrt{n} \rfloor, n^{3/4}\}$。最后，抽取 R 中第 l 个元素作为 L，抽取 R 中第 h 个元素作为 H，希望 $L \le \min(S,k) \le H$，从而可以用 L 和 H 作为选择范围在下一阶段从 S 中真正将 $\min(S,k)$ 选出来。
- 第三阶段：从 S 中抽取介于 L 和 H 之间的元素构成子集 P，即 $P = \{y \in S | L \le y \le H\}$。如果 $\min(S,k) \in P$ 且 $|P| \le 4n^{3/4} + 1$，则可以在 $O(n)$ 时间内将 P 排序并找到目标元素。计算 $L_p = rank(S,L)$ 和 $H_p = rand(S,H)$ 找到 L 和 H 在 S 中的位置。于是，$\min(S,k) \in P$ 当且仅当 $L_p \le k \le H_p$

随机选择算法

RandSelect(S, k)

1: 独立、均匀、可放回地从 S 中随机选择 $n^{3/4}$ 个元素存入 R
2: 在 $O(n^{3/4} \log n^{3/4}) = O(n)$ 时间内排序 R
3: $x \leftarrow (k/n)n^{3/4}$
4: $l \leftarrow \max\{\lfloor x - \sqrt{n} \rfloor, 1\}$, $h \leftarrow \min\{\lfloor x + \sqrt{n} \rfloor, n^{3/4}\}$
5: $L \leftarrow \min(R, l)$, $H \leftarrow \min(R, h)$ ▷ 找第 l，h 小元素
6: $L_p = rank(S, L)$, $H_p = rand(S, H)$ ▷ 将 L 和 H 与 S 的每个元素比较
7: $P = \{y \in S | L \leq y \leq H\}$
8: **if** $\min(S, k) \in P$(即 $L_p \leq k \leq H_p$) 且 $|P| \leq 4n^{3/4} + 1$ **then**
9: 递增排序 P，输出 P 中第 $k - L_p$ 个元素

随机选择算法分析

- 第 1 步开销 $O(n^{3/4}) = O(n)$；第 2 步开销 $O(n)$；第 3–5 步需要 $O(1)$ 时间
- 第 6 步需要 $2n$ 次比较；第 7 步需要 $O(n)$ 时间；第 8 步需要 $O(1)$ 时间，第 9 步排序需要 $O(n)$ 时间
- 随机选择算法的时间复杂度 $O(n)$

定理

算法 RandSelect 第 1–9 步执行一遍 ($O(n)$ 时间) 就可以求出 $\min(S, k)$ 的概率是 $1 - O(n^{-1/4})$

证 证明过程过于复杂，从略!

拉斯维加斯算法总结

- 拉斯维加斯算法获得的解一定是问题的正确解，但算法也有可能不能得到问题的解，获得正确解的概率为 p，其中 $0 < p < 1$，此事称该算法为 p-正确的拉斯维加斯算法
- 可通过重复运行提高找到正确解的概率，对于 p-正确的拉斯维加斯算法，第一遍找到正确解的概率为 p，第二遍为 $(1-p)p$，依此类推，算法在运行 i 遍时才找到正确解的概率为 $(1-p)^{i-1} \cdot p$
- 算法找出问题正确解需要运行的遍数 N 的数学期望 $E(N)$ 为 $E(N) = \sum_{i=1}^{+\infty} i \cdot [(1-p)^{i-1}p] = 1/p$，因此反复运行时间复杂度为 $T(n)$ 的 p-正确拉斯维加斯算法，找出问题正确解需要的期望时间复杂度为 $T(n)/p$
- 随机选择算法的期望运行时间为 $O(n)/[1-O(n^{-1/4})] = O(n)$

提纲

快速排序算法 (Quick Sort)

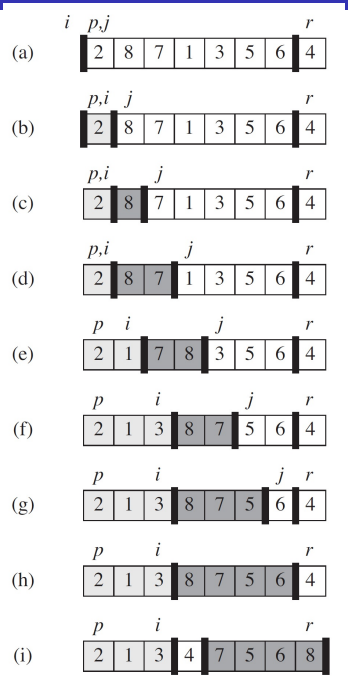
QUICKSORT(A, p, r)

```
1: if p < r then
2:     q = PARTITION(A, p, r)
3:     QUICKSORT(A, p, q − 1)
4:     QUICKSORT(A, q + 1, r)
```

PARTITION(A, p, r)

```
1: x = A[r]
2: i = p − 1
3: for j ← p to r − 1 do
4:     if A[j] ≤ x then
5:         i = i + 1
6:         SWAP(A[i], A[j])
7: SWAP(A[i+1],A[r])
8: return i + 1
```

快速排序分区过程示意


i p,j r
(a) 2 8 7 1 3 5 6 4
p,i j r
(b) 2 8 7 1 3 5 6 4
p,i j r
(c) 2 8 7 1 3 5 6 4
p,i j r
(d) 2 8 7 1 3 5 6 4
p i j r
(e) 2 1 7 8 3 5 6 4
p i j r
(f) 2 1 3 8 7 5 6 4
p i j r
(g) 2 1 3 8 7 5 6 4
p i r
(h) 2 1 3 8 7 5 6 4
p i r
(i) 2 1 3 4 7 5 6 8

快速排序算法的性能分析

- 最坏情况划分
  - 划分的子问题分别包含了 $n-1$ 个元素和 0 个元素时产生；
  - 划分操作时间复杂度 $\Theta(n)$；
  - 此时递归方程：$T(n)=T(n-1)+T(0)+\Theta(n)=T(n-1)+\Theta(n)=\Theta(n^2)$。
- 最好情况划分
  - 划分得到的两个子问题规模都小于 $n/2$，平衡划分；
  - 递归方程：$T(n)=2T(n/2)+\Theta(n)=\Theta(n \lg n)$。
- 可知快速排序算法最好情况和最坏情况性能差距非常大。

快排序算法的平均复杂度

1) 假设排序问题的输入服从均匀分布且输入数组 $A[p:r]$ 中元素各不相同，因此 $A[p:r]$ 中小于划分元素 $x = A[r]$ 的元素个数均匀等可能地为 $0, 1, 2, \cdots, n-1$ 个，每种情况出现的概率为 $1/n$

2) 运行算法 Partition 时得到的划分结果中 $A[p:q-1]$ 等概率地含有 $0, 1, 2, \cdots$ $n-1$ 个元素，而 $A[q:r]$ 中相应地含有 $n-1, n-2, \cdots, 1, 0$ 个元素

3) 故此，快排序算法平均复杂度满足递归方程:$T(1) = \Theta(1)$

$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-i) + \Theta(n)] \qquad n > 1$$

$$= \frac{2}{n}[T(n-1) + T(n-2) + \cdots + T(1) + T(0)] + \Theta(n)$$

$$\Rightarrow T(n) = \Theta(n \log n)$$

随机排序算法建模

- 问题：输入序列 $S = \{s_1, s_2, \cdots, s_n\}$，给出排序后的 S。
- 基本思想：
  - 以快速排序算法 QuickSort 和分治算法为基础
  - 采用随机抽样的方法确定序列的划分点
  - 把集合划分为两个子集合
  - 分别递归地在每个子集合上使用随机排序算法
- 随机排序算法是 Sherwood 算法的应用，消除固定划分点导致的算法性能波动

RAND-SORT(n)

1: 按照均匀分布从 S 中随机选择一个样本 y
2: 将 S 划分为如下两个集合：$S_1 = \{x | x \in S, x < y\}$，$S_2 = \{x | x \in S, x \geq y\}$
3: 递归排序 S_1 和 S_2
4: 按顺序输出 S_1, y, S_2

随机排序算法的性能分析 — 1

- z_i 表示待排序元素中第 i 小的元素，Z_{ij} 表示 z_i 与 z_j 之间的元素集合 (注意每次划分后的子数组都对应某个区间 Z_{ij})
- 定义随机变量 $X_{ij} = I\{z_i$ 与 z_j 进行比较$\}$
- 随机排序中几乎所有的计算量都花在了两个元素互相比较
- 随机排序算法的计算量可由总的比较次数来刻画：

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

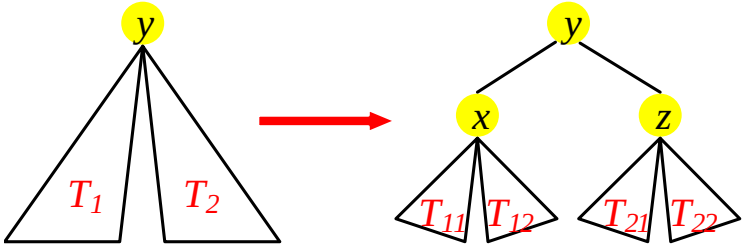
- 对上式取期望即可得到随机排序算法的性能：

$$E(X) = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr(z_i \text{ 与 } z_j \text{ 进行比较})$$

随机排序算法的性能分析 — 2

用树来表示随机排序的计算过程：



- 一个子树的根必须与其子树的所有节点比较
- 不同子树中的节点不可能比较
- 任意两个节点至多比较一次
- 只有两个结点处于同一子树下才有可能比较
- 对任意一对元素 z_i 与 z_j 对应的子树，只有当 z_i 或 z_j 被选为划分点才会产生二者比较的可能，这一概率即为

$$Pr(z_i \text{ 与 } z_j \text{ 进行比较}) = \frac{2}{j-i+1}$$

随机排序算法的性能分析 — 3

性能分析：

$$
\begin{aligned}
E(X) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr(z_i \text{ 与 } z_j \text{ 进行比较}) \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n)
\end{aligned}
$$

随机排序算法的时间复杂度为 $O(n \lg n)$，消除了快速排序最坏情况下 $O(n^2)$ 的复杂度！

舍伍德算法总结

- 普通快排序算法是确定性的，针对同一问题实例操作序列是确定的，复杂度平均值为 $\Theta(n \log n)$，但仍难以避免在特定实例上复杂度为 $\Theta(n^2)$
- 随机快排序算法的复杂度数学期望为 $\Theta(n \log n)$，它在任意输入序列上的操作序列是随机的，算法的复杂度是一个随机变量，意味着随机快排序多次排序同一实例，时间复杂度均值为 $\Theta(n \log n)$，达到最坏复杂度 $\Theta(n^2)$ 的概率几乎为 0
- 随机算法时间复杂度的数学期望不同于算法的平均复杂度，其随机性是由算法执行的操作步骤的随机性引起的，不受问题实例的影响
- 当一个确定型算法的最坏和最好复杂度差别较大时，可在算法中引入随机性消除好坏实例间复杂度的差别，这就是舍伍德算法的核心思想。该算法总能求的一个解且解是正确的

最近点对问题

最近点对问题

- 输入：Euclidean 空间上的 n 个点的集合 S
- 输出：点 $p_1, p_2 \in S$, $dis(p1, p2) = \min\{dis(x, y) | x, y \in S\}$

- 分治算法求解最近点对问题时间复杂度为 $O(n \log n)$
- 可否构造复杂度为 $O(n)$ 的最近点对算法?

充分利用 Hash 数据结构，随机最近点对算法复杂度期望为 $O(n)$

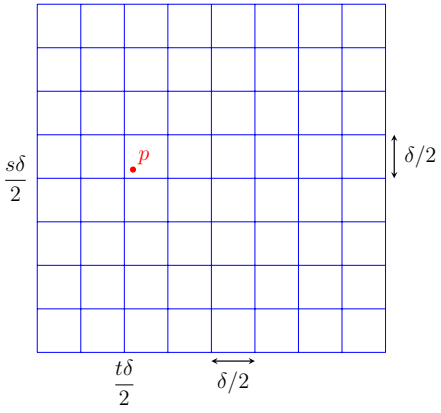
随机最近点对问题求解思路

1. 为简化问题，认为所有点的坐标分布于单位矩形中，即 $\forall i = 1, \ldots, n$，$0 \leq x_i, y_i < 1$
2. 对于一般性问题，可以对点的坐标先进行缩放，使之分布于单位矩形内
3. 首先对所有点随机排序，然后依次扫描各点，在扫描的过程中记录已找到的最近点对及其距离 δ
4. 假设当前扫描点为 p，快速查看 p 周围的点，判断其与 p 距离是否小于 δ。如是，更新 δ 值。接下来继续扫描下一点

算法的难点在于如何“快速”扫描点 p 附近的点？有没有办法在常数时间内完成扫描？如可以则上述算法的复杂度可达到 $O(n)$

快速扫描临近点

- 假设当前待扫描点为 p，之前已扫描过的点的最近点对距离为 δ
- 将单位矩形的空间划分为宽度为 $\delta/2$ 的网格，这样的网格有 N^2 个，$N = \lceil 1/(\delta/2) \rceil$
- N^2 个网格依照 x, y 两个维度的坐标引用，比如右图 p 位于网格 S_{st} 内



$$S_{st} = \{(x, y) : s\delta/2 \le x < (s+1)\delta/2; t\delta/2 \le y < (t+1)\delta/2\}$$

快速扫描临界点 (续)

命题 1

如果两个点 p, q 属于同一个网格 S_{st}，则 $d(p, q) < \delta$

证 任一网格 S_{st} 内最长距离为其对角线长度 $\sqrt{(\delta/2)^2 + (\delta/2)^2} = \delta/\sqrt{2} < \delta$ □

命题 2

任意两点 p, q 间距小于 δ，则该两点要么处于同一网格，要么分处相近的两个网格 *(*两个网格 S_{st} 和 $S_{s't'}$ 相近指 $|s - s'| \leq 2$ 且 $|t - t'| \leq 2$*)*

证 p, q 如处于同一网格，则其间距小于 δ；如处于不同网格，假设 $|s - s'| > 2$ 或 $|t - t'| > 2$，则必然在某个维度 p, q 的间距大于 δ，从而不满足相近的定义 □

随机点对算法的设计考量

- 由上页两个命题，对于每个当前待处理的点 p，仅需扫描其附近 (x, y 坐标差距不超过 2) 的 25 个网格并计算 p 与这些网格内的点之间的距离，并判断是否找到了更小的 δ
- 为快速找到某个网格所包含的点，可将这些点与所在网格通过某种数据结构 bind 在一起
- 由于 δ 可能很小，导致网格数 N^2 非常大，如果用数组存储全部网格会占据非常多的空间，可考虑仅存储用到的网格，用 hash 表实现快速的查询

随机最近点对算法

RAND_CLOSEST_PAIR(A)

1: 构造 A 中所有点的一个随机序列 $p_1, p_2, \ldots, p_n$
2: $\delta = d(p_1, p_2)$ ▷ δ 代表当前最近点对距离
3: 初始化一个空 hash 结构 H，用于存储网格
4: **for** $i \leftarrow 1$ to n **do** ▷ 依次处理 n 个点
5: 计算点 p_i 所在的网格 S_{st}
6: 找到 p_i 附近的 25 个网格 (含 S_{st})
7: 从 H 搜索这 25 个网格，并计算这些网格所绑定的点与 p_i 的距离
8: **if** 存在某个点 $p_j (j < i)$ 使得 $\delta' = d(p_j, p_i) < \delta$ **then**
9: 清空 H，以 $\delta'/2$ 为边长重新划分网格
10: **for all** $p_1, p_2, \ldots, p_i$ **do**
11: 计算该点所在网格 (边长 $\delta'/2$) 并将其与该点绑定，Insert 到 H 中
12: **else**
13: 将 S_{st} 与 p_i 绑定并 Insert 到 H 中
14: **return** δ' 及其对应点对

对 hash 表插入操作次数的分析

定义随机变量 X_i，如果算法第 8 行条件为真，即当前点引起了最短点距 δ 的变化，则 $X_i = 1$；否则，$X_i = 0$

引理 4

算法所执行的对 H 的 *Insert* 操作的数量至多为 $n + \sum_i iX_i$

证 观察算法，所有点在首次遇到时被插入 H，然后在处理点 p_i 的迭代过程中被重新插入 H，二者合计至多为 $n + \sum_i iX_i$ 次 □

引理 5 ($Pr[X_i = 1] \leq 2/i$)

证 假设序列 $p_1, p_2, \ldots, p_i$ 中的最近点对为 (p, q)，$X_i = 1$ 表示 p_i 要么是 p，要么是 q，二者必居其一，此概率至多为 $2/i$，因此 $Pr[X_i = 1] \leq 2/i$ □

推论 1

对 H 总的插入操作的次数期望值为 $E[X] = n + \sum_i iE[X_i] \leq n + 2n = 3n$

随机最近点对算法的性能粗略分析

引理 6

随机最近点对算法的时间复杂度为 $O(n)$，另外需额外进行 $O(n)$ 次 *hash* 表操作

- 算法第 1 行将 n 个点随机排列，复杂度 $O(n)$；第 2–3 行仅需常数时间
- 第 4 行的 for 循环遍历 n 个点，需迭代 n 次；第 5–6 行都只需要常数时间
- 算法的 hash 表采用全域哈希，即哈希函数的选择为从一族函数中随机选择一种，因此其发生碰撞的概率很低，单次哈希操作的复杂度期望值为 $O(1)$
- 考虑到网格以已扫描点的最近点对距离为准进行划分，因此已扫描过的点不会有两个及以上处于同一网格中，否则其距离将小于 δ。因此，第 7 行对 H 进行 25 次搜索仅需常数时间，25 个网格所包含的已扫描点至多不超过 25 个，第 7 行时间复杂度为 $O(1)$
- 第 8–13 行的 if 语句，由上页的推论 1，在整个算法的执行期间，至多进行 $3n$ 次对哈希表的插入操作，可认为时间复杂度也为 $O(n)$
- 因此，随机最近点对算法的时间复杂度为 $O(n)$

随机性对最近点对算法的影响

随机最近点对算法的随机性体现在两点：

1. 算法第 1 步，对点的处理顺序是随机的，从而避免了出现 δ 每次都缩小的情况
2. 对 hash 表采用全域哈希结构，随机从一族函数中选择一种作为哈希函数，降低了碰撞的概率，保证了算法性能

从随机最近点对算法的特点可知，该算法属于舍伍德类型算法

小结

小结：

- 随机算法的相关概念 (了解)
- 随机算法的设计应用 (重点)