中国科学技术大学 计算机科学与技术系
University of Science and Technology of China
Department of Computer Science and Technology

算法设计与分析

Design and Analysis of Algorithms

主讲人　徐云

Fall 2012, USTC

第4章(补充) 随机算法

4.1 概述

4.2 时间复杂性度量

4.3 一般设计风范

4.4 随机取样

4.5 串匹配算法

4.6 格点逼近问题

4.1 方法概述

- 随机算法的定义
- 背景和历史
- 研究意义
- 随机算法的分类
- 重复性定律
- 示例：Quick Sort, Min Cut

随机算法的定义

- 定义：是一个概率图灵机. 也就是在算法中引入随机因素，即通过随机数选择算法的下一步操作。
- 三要素：输入实例、随机源和停止准则。
- 特点：简单、快速和易于并行化。
- 一种平衡：随机算法可以理解为在时间、空间和随机三大计算资源中的平衡(*Lu C.J. PhD Thesis 1999*)
- 重要文献： *Motwani R. and Raghavan P., Randomized Algorithms. Cambridge University Press, New York, 1995*

背景和历史

- 著名的例子
  - Monte Carlo求定积分法
  - 随机k-选择算法
  - 随机快排序
  - 素性判定的随机算法
  - 二阶段随机路由算法
- 重要人物和工作
  - de Leeuw等人提出了概率图灵机(1955)
  - John Gill的随机算法复杂性理论(1977)
  - Rabin的数论和计算几何领域的工作(1976)
  - Karp的算法概率分析方法(1985)
  - Shor的素因子分解量子算法(1994)

研究意义（1）

- 求解问题的一种重要让步策略
- 有效的随机算法
- 实际可行的随机算法
- 可转化为确定算法
- 易于并行化
- 促进智能计算的发展

研究意义（2）

一种重要让步策略：

- 算法
  - 确定的
    - 传统算法：以100%成功概率求解问题的最优解
    - 近似算法：一种对解质量的让步
  - 非确定的
    - 随机算法：一种对求解成功概率和解质量的让步
    - 智能算法：遗传算法、模拟退火法、拟人拟物算法等

研究意义（3）

有效的随机算法：

- 素性测试问题：提出的第一个多项式时间的算法是随机算法；
- 匹配问题：存在NC类的随机并行算法，而没有NC类的确定并行算法；
- 无向图可达问题：存在logspace的随机算法，而没有logspace的确定算法，

随机算法的分类

常见的两类随机算法：

- Las Vegas算法运行结束时总能给出正确的解，但其运行时间每次有所不同。
- Monte Carlo算法可能得到不正确的结果，但这种概率是小的且有界的。

重复性定律

- 设ε是一次随机试验的成功概率，则N次独立随机试验的成功概率为

$$P = 1-(1-\varepsilon)^N \approx N\varepsilon$$

- 例如：$\varepsilon = 0.05$, $N=20$, $P \approx 1$

Quick Sort

- 传统的快排序算法

  -总是取第一个元素作为划分元；

  -算法的最坏运行时间是$O(n^2)$，平均时间是$O(n\log n)$；

  -因此存在一些输入实例，使得算法达到最坏运行时间，如：降序的序列；

- 随机快排序算法

  -随机选择一个元素作为划分元；

  -任何一个输入的期望时间是$O(n\log n)$；

  -是一个Las Vegas算法；

Min Cut (1)

- 最小截问题定义：

  给定一个无向图$G(V,E)$，找一个截(V_1,V_2)使得V_1和V_2间的连边数最小。

- 注：

  该问题可以用确定性算法（max-flow min-cut algorithm）在$O(n^2)$时间内完成。

Min Cut (2)

- 随机算法

  随机选一条边，将两顶点合一并除去顶点上的环；

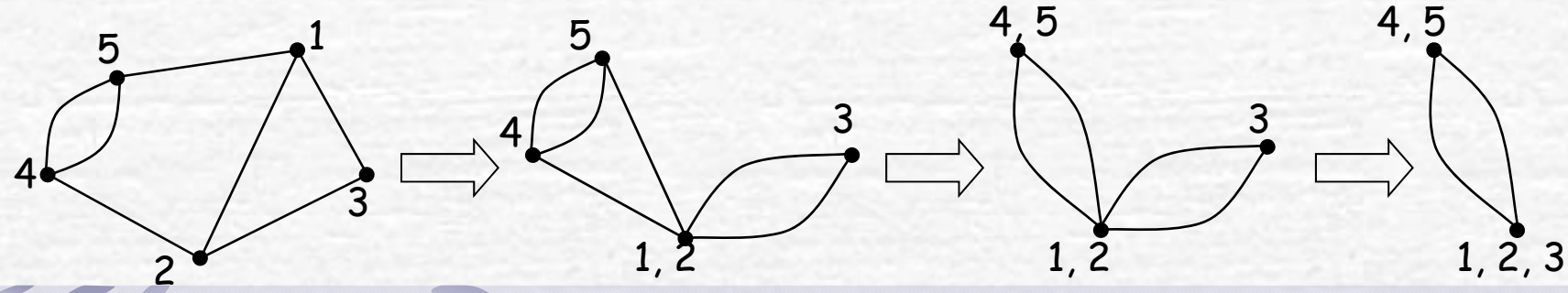
  直到图中只剩下两个顶点；

  返回剩下两顶点间的连边数；

- 重复版本

  重复执行算法k次，取返回连边数最小数作为解。

- 示例：#cut=2



Min Cut（3）

- 出错概率

  重复k次出错概率为$\left(1-\frac{2}{n(n-1)}\right)^k \le e^{-\frac{2k}{n(n-1)}}$

- 本算法是一个Monte Carlo型算法

第4章(补充) 随机算法

4.1 概述
4.2 时间复杂性度量
4.3 一般设计风范
4.4 随机取样
4.5 串匹配算法
4.6 格点逼近问题

4.2 时间复杂性度量

- 运行时间的期望和方差

  (1)实例的运行时间期望

  对固定实例X，设随机算法A的运行时间 ξ_x^A 是一个[0,+∞)上的随机变量，定义随机算法A在实例X上的运行时间期望为 $E[\xi_x^A]$，也称为随机算法A在实例X上的执行时间.

  (2)算法的运行时间期望

  如果对一个规模为n问题的所有实例是均匀选取的，则定义各个实例上的平均执行时间为随机算法在该问题上的运行时间期望，记为T(n)

  注：类似地可以定义方差.

- 随机复杂度类(参见*Motwani R. and Raghavan P., Randomized Algorithms.*)

  RP(Randomized Polynomial time), *ZPP*, *PP*, *BPP* *etc.*

第4章(补充) 随机算法

4.1 概述

4.2 时间复杂性度量

4.3 一般设计风范

4.4 随机取样

4.5 串匹配算法

4.6 格点逼近问题

4.3 一般设计风范 (1)

1.挫败对手(*Foiling the Adversary*)

将不同的算法组成算法群，根据输入实例的不同随机地或有选择地选取不同的算法，以使性能达到最佳.

2.随机采样(*Random Sampling*)

用“小”样本群的信息，指导整体的求解.

3.随机搜索(*Random Search*)

是一种简单易行的方法，可以从统计角度分析算法的平均性能，如果将搜索限制在有解或有较多解的区域，可以大大提到搜索的成功概率.

4.指纹技术(*Fingerprinting*)

利用指纹信息可以大大减少对象识别的工作量. 通过随机映射的取指方法，Karp和Rabin得到了一个快速的串匹配随机算法.

4.3 一般设计风范（2）

5.输入随机重组(*Input Randomization*)

对输入实例进行随机重组之后，可以改进算法的平均性能.

6.负载平衡(*Load Balancing*)

在并行与分布计算、网络通讯等问题中，使用随机选择技术可以达到任务的负载平衡和网络通讯的平衡.

7.快速混合Markov链(*Rapidly Mixing Markov Chain*)

使用该方法可以解决大空间中的均匀抽样问题.

8.孤立和破对称技术(*Isolation and Symmetry Breaking*)

使用该技术可以解决处理的并行性，避免分布式系统的死锁等问题. 如:图着色算法，部分独立集问题

4.3 一般设计风范（3）

9.概率存在性证明(*Probabilistic Methods and Existence Proofs*)

如果随机选取的物体具有某种特性的概率大于0，则具有该特性的物体一定存在.

10.消除随机性(*Derandomization*)

许多优秀的确定性算法是由随机算法转换而来的.

第4章(补充) 随机算法

4.1 概述

4.2 时间复杂性度量

4.3 一般设计风范

4.4 随机取样

4.5 串匹配算法

4.6 格点逼近问题

4.4 随机取样

- 问题定义
- 随机取样算法
- 期望运行时间分析

问题定义和随机取样算法

- 问题定义

  从n个元素的集合中随机选取m个元素的样本，m<n

- 随机取样算法

```
RandomSampling(n,m)
{ //从集合{1,2,…,n}中随机选择m个不同的正整数组成的数组A[1..m]
    for i=1 to n do  s[i]=false;  //s表示一个整数是否被选择
    k=0;
    while k<m do
    {   r=random(1,n);
        if not s[r] then
        {    k++; A[k]=r; s[r]=true;   }
    }
 }
```

注：一般我们可以假设m≤n/2。

期望运行时间分析（1）

- 预备知识：几何分布

设c是一枚硬币，出现正面的概率是p，令q=1-p。令X是一个表示直到正面第一次出现时总的抛掷次数的随机变量，则X满足分布和期望分别为：

$$\Pr[X=k]=\begin{cases} pq^{k-1} & \text{若} k\geq 1 \\ 0 & \text{若} k=0 \end{cases}, \quad E(X)=\sum_{k=0}^{\infty} kpq^{k-1}=\frac{1}{p}$$

- 期望运行时间分析

设p_k为生成第k个样本整数的成功概率，k=1~m。显然有

$$p_k=(n-k+1)/n$$

用随机变量X_k表示为了选定第k个整数而生成的整数的个数，随机变量Y表示从n个整数中选出m个样本整数而生成的总的整数个数，则

$$E(Y)=E(X_1)+E(X_2)+\ldots+E(X_m)$$

期望运行时间分析（2）

- 期望运行时间分析(续)

$$E(Y)=\sum_{k=1}^{m}E(X_k)=\sum_{k=1}^{m}\frac{n}{n-k+1}=n\sum_{k=1}^{n}\frac{1}{n-k+1}-n\sum_{k=m+1}^{n}\frac{1}{n-k+1}$$

$$=n\sum_{k=1}^{n}\frac{1}{k}-n\sum_{k=1}^{n-m}\frac{1}{k}$$

$\because$ $\sum_{k=1}^{n}\frac{1}{k}\le \ln n+1$ 和 $\sum_{k=1}^{n-m}\frac{1}{k}\ge \ln(n-m+1)$

$$\Rightarrow E(Y)\le n(\ln n+1-\ln(n-m+1))\le n(\ln n+1-\ln(n/2)) \quad // \because m\le n/2$$

$$=n(\ln 2+1)=n\ln 2e\approx 1.69n$$

所以，$T(n)=\Theta(n)$

第4章(补充) 随机算法

4.1 概述

4.2 时间复杂性度量

4.3 一般设计风范

4.4 随机取样

4.5 串匹配算法

4.6 格点逼近问题

4.5 串匹配算法

- 问题定义
- 测试串的相等性
- 串匹配算法

问题定义

- **问题：**

  给定两个字符串$X=x_1,\cdots,x_n$，$Y=y_1,\cdots,y_m$，判断Y是否为X的子串？(即Y是否为X中的一段)

- **蛮力算法：** $O((n-m+1)m)$

  记$X(j)=x_jx_{j+1}\cdots x_{j+m-1}$(从X的第j位开始、长度与Y一样的子串)，算法从起始位置j=1开始到j=n-m+1，对串X(j)和串Y进行相等性比较。

- **确定型算法：**

Algorithm	Preprocessing Time	Matching Time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite Automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m)$	$\Theta(n)$

测试串的相等性（1）

- 长串的“指纹”测试问题

  给定两个长字符串x和y，由x和y的“指纹” 判断x和y是否相等？

  注：“指纹”是一种特征信息，比原对象容量小得多。一般地，“指纹”不同则两个串一定不同，“指纹”相同时会出现误判。

- 一种“指纹”技术：长度为n的串可以压缩为$O(\log_2 n)$

  令I(x)为x的编码，n是I(x)的二进制表示的长度，则有 $I(x) < 2^n$。

  取$I_p(x)=(I(x) \bmod p)$作为x的指纹，这里p是一个小于M的素数，M可根据具体需要进行调整，一般取$M=2n^2$

  ∵ $0 \le I_p(x) < p$ ∴ $I_p(x)$的二进制长度不超过$\log_2 p$

  由于$p < M=2n^2$，故有$\log_2 p \le \lfloor \log_2(2n^2) \rfloor + 1 = O(\log_2 n)$。

测试串的相等性（2）

- 一种“指纹”技术（续）：

例：设I(x)是10^6位二进制的数，即n=10^6，则
$M=2\times10^{12}\approx2^{40.8631}$

∴ $I_p(x)$的位数不超过41位

因此如果是网络传输，一次可节省约2.5万倍。

- 测试算法

首先由A发一个x的长度给B，

若长度不等，则x≠y。

若长度相等，则采用上面取“指纹”的方法：

A对x进行处理，取出x的“指纹”，然后将x的“指纹”发给B，由B检查x的“指纹”是否等于y的“指纹”。

如果“指纹”不等，则x≠y；

否则，x=y；　//此时可能会误判

测试串的相等性（3）

- 错判问题

  如果$I_p(x)=I_p(y)$而$x \neq y$，则此种情况为一个误匹配。问题是误匹配的概率有多大？

- 两个数论定理
  - 定理1：设π(a)是小于a的素数个数，则π(a)≈a/ln a
  - 定理2：如果$a < 2^n$，则能够整除a的素数个数不超过π(n)个

- 错判率分析

  设n是x和y的二进制表示形式的位数，只有p整除$I_p(x)-I_p(y)$时才有可能误判，则失败的概率是

$$\frac{\left|\{p \mid \text{素数} p < M \text{且} p \mid (I_p(x)-I_p(y))\}\right|}{\pi(M)} \leq \frac{\pi(n)}{\pi(M)} \approx \frac{n/\ln n}{2n^2/\ln(2n^2)} \approx \frac{1}{n}$$

串匹配算法（1）

- 算法的基本思想
  - 记$X(j)=x_jx_{j+1}\cdots x_{j+m-1}$(从X的第j位开始、长度与Y一样的子串)，
  - 从起始位置j=1开始到j=n-m+1，我们不去逐一比较X(j)与Y，而仅逐一比较X(j)的指纹$I_p(X(j))$与Y的指纹$I_p(Y)$。
  - 由于$I_p(X(j+1))$可以很方便地根据$I_p(X(j))$计算出来，故算法可以加速。

串匹配算法（2）

- 由$I_p(X(j))$计算$I_p(X(j+1))$的方法

不失一般性,设 $x_i(1\leqslant i\leqslant n)$ 和 $y_j(1\leqslant j\leqslant m)\in\{0,1\}$,即 X, Y 都是 0-1 串。

$$Ip(X(j+1)) = (x_{j+1}\cdots x_{j+m})(\text{mod } p)$$

$$=(2(x_{j+1}\cdots x_{j+m-1})+x_{j+m})(\text{mod } p)$$

$$=(2(x_{j+1}\cdots x_{j+m-1})+2^m x_j-2^m x_j+x_{j+m})(\text{mod } p)$$

$$=(2(x_j x_{j+1}\cdots x_{j+m-1})-2^m x_j+x_{j+m})(\text{mod } p)$$

（$\because (xy+z)(\text{mod } p)=(x(y \text{ mod } p)+z)(\text{mod } p)$）

$$=(2\ (\ (x_j x_{j+1}\cdots x_{j+m-1})\text{mod } p\ )-2^m x_j+x_{j+m})(\text{mod } p)$$

$$=(2*Ip(X(j))-\underbrace{(2^m \bmod p)}_{Wp} x_j+x_{j+m})(\text{mod } p) \quad (*)$$

∴Ip(X(j+1))可以利用 Ip(X(j))及(*)式计算出来。

串匹配算法（3）

- 算法框架

①随机取一个小于M的素数p，置j←1；
②计算$I_p(Y)$、$I_p(X(1))$及$W_p(=2^m \bmod p)$；
③While j≤n-m+1 do
　if $I_p(X(j))=I_p(Y)$ then return j　//X(j)极有可能等于Y
　else{使用(*)式计算出$I_p(X(j+1))$；j++ }
④return 0;　//X中没有子串匹配Y

- 时间复杂度：O(m+n)
  - Y、X(1)、2^m均只有m位(二进制数)，故计算$I_p(Y)$、$I_p(X(1))$及$2^m \bmod p$的时间不超过O(m)次运算
  - $I_p(X(j+1))$的计算只需用O(1)时间
- 执行1次的错判概率≤1/n，执行k次的错判概率≤$1/n^k$。这是一个Monte Carlo算法，可以修改为Las Vegas算法。

第4章(补充) 随机算法

4.1 概述

4.2 时间复杂性度量

4.3 一般设计风范

4.4 随机取样

4.5 串匹配算法

4.6 格点逼近问题

4.6 格点逼近问题

- 问题定义
- 阈值算法和误差阶
- 随机算法和误差阶分析

问题定义和阈值算法

- 问题定义

输入：一个n×n阶的0、1矩阵A，

一个n个元素的列向量p，其元素取值于区间[0, 1]

输出：一个n个元素的0、1向量q，使得误差$||A(p-q)||_\infty$最小

- 阈值算法

```
Begin
  for i=1 to n do
  {  if  pi>=0.5  then qi=1
     else qi=0
   }
End
```

该算法可能使误差$||A(p-q)||_\infty=O(n)$

随机算法和误差阶分析（1）

- 随机算法

```
Begin
  for i=1 to n do
  {
    set qi to 1 with probability pi, and to 0 otherwise
   }
End
```

- 误差阶的证明

引理 设$X_1,\ldots,X_n$是独立的0-1随机变量，且$Pr[X_i=1]=p_i$和$Pr[X_i=0]=1-p_i$，S是整数1,…,n的子集，$s=|S|$。令$Y=\Sigma_{i\in S}X_i$，则

$$Pr[|Y-E[Y]|>\sqrt{4slns}]\leq\frac{1}{s^2}$$

随机算法和误差阶分析（2）

- 误差阶的证明（续）

定理 随机算法求得的q，使得$\|A(p-q)\|_\infty \le \sqrt{4nlnn}$，至少以概率1-1/n成立。

证明：

设A_i为A的第i行，由算法知：$E[A_iq]= A_ip$

由引理 =>

$$Pr\left[\left|A_iq - A_ip\right| > \sqrt{4nlnn}\right] = Pr\left[\left|A_iq - E[A_iq]\right| > \sqrt{4nlnn}\right] \le \frac{1}{n^2}$$

$$\therefore\ Pr\left[\|A(p-q)\|_\infty > \sqrt{4nlnn}\right] = Pr\left[\bigvee_{i=1}^{n}\left\{\left|A_iq - A_ip\right| > \sqrt{4nlnn}\right\}\right]$$

$$\le \sum_{i=1}^{n} Pr\left[\left|A_iq - A_ip\right| > \sqrt{4nlnn}\right] \le \sum_{i=1}^{n}\frac{1}{n^2} = \frac{1}{n}$$

即 $\|A(p-q)\|_\infty \le \sqrt{4nlnn}$，至少以概率1-1/n成立。

End of SCh4