第7章 随机算法（概率算法）

- 有限概率空间
  - 存在着有穷个基本事件
  - 每个基本事件发生的概率是[0,1]中的数
  - 所有基本事件的概率之和等于1

- 事件及其概率
  - 一些基本事件的组合称为事件
  - 事件发生的概率等于其中基本事件的概率之和

- 通常把事件A的概率记为 Pr[*A*] .

- 随机变量：是随机实验结果的赋值函数.
- 分布：随机变量取各种值的概率.
- 期望：随机变量的加权平均值，通常把随机变量 X 的期望记作 E[X]

例如，抛掷一枚均匀硬币n次，记录正面向上的次数，就得到一个随机变量X，其取值范围是 0, 1, 2, …, n,

X 的分布可以描述为

$$\Pr[X = i] = \binom{n}{i} 2^{-n}$$

X 的期望

$$\mathrm{E}[X] = \sum_{i=0}^{n} i \binom{n}{i} 2^{-n}$$

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（Sherwood）算法
- 拉斯维加斯（Las Vegas）算法
- 蒙特卡罗（Monte Carlo）算法
- 总结

动机和应用背景

引例1

判断表达式$f(x_1, x_2, \ldots, x_n)$是否恒等于0。

若x_i=0 或 1，布尔表达式（永假式），2^n

概率算法:

（1）生成一个随机n元向量$(r1, r2, \ldots, rn)$

（2） 计算$f(r1, r2, \ldots, rn)$的值

（3）如果$f(r1, r2, \ldots, rn)\neq 0$，则$f(x1, x2, \ldots, xn)\neq 0$；

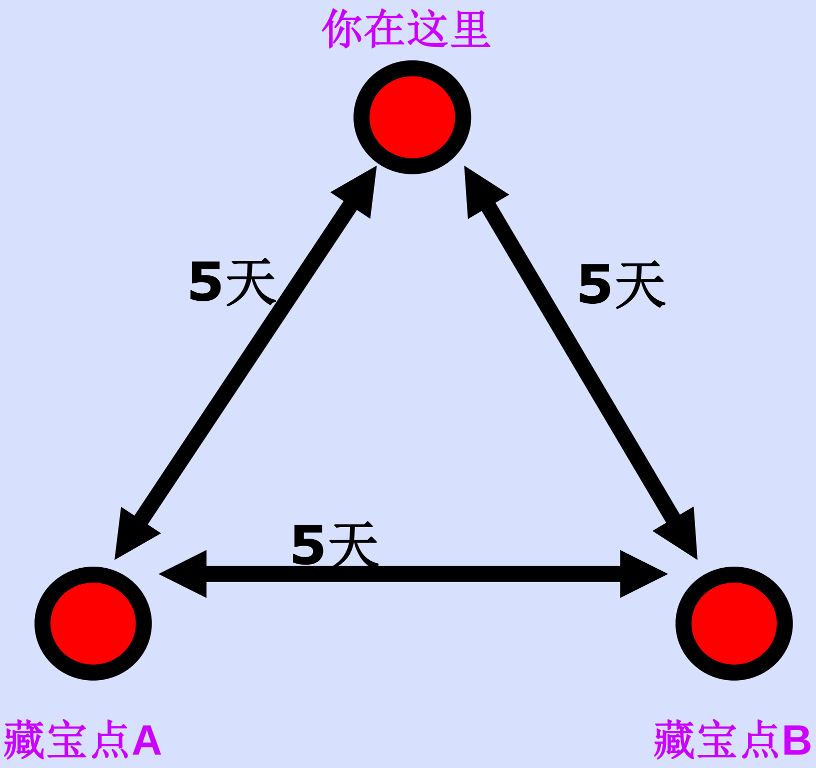
$f(r1, r2, \ldots, rn)=0$，则或者$f(x1, x2, \ldots, xn)$恒等于0，或者是$(r1, r2, \ldots, rn)$比较幸运

重复几次，有$f(r1, r2, \ldots, rn)=0$，那么就可以得出恒等于0的结论，并且测试的随机向量越多，这个结果出错的可能性就越小！！

动机和应用背景

引例2

- 一张藏宝图，宝藏数量x，两个可能的藏宝点A,B
- 需4天解读藏宝图，获知确切地点
- 另一个人每天拿走y的宝藏，x>9y
- 精灵告诉藏宝点，要求3y的报酬
- 你应该怎么做?
- 为什么?



选择:

1. 花4天解读藏宝图
2. 接受小精灵条件
3. 直接去藏宝点A
4. 直接去藏宝点B

- 选择:

1. 花4天解读藏宝图

   x-9y

2. 接受小精灵条件

   x-8y

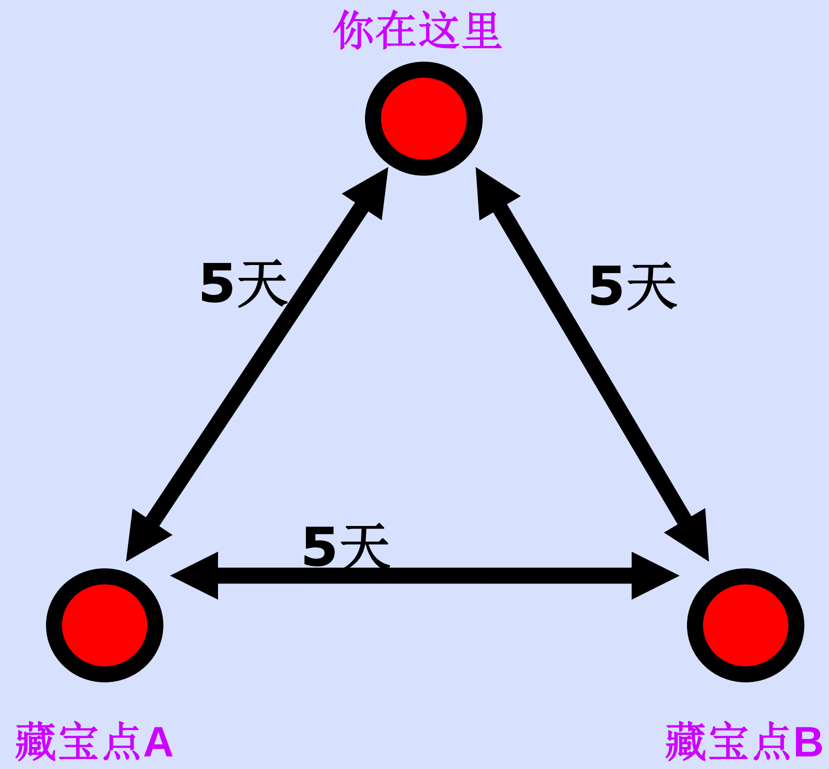
3. 直接去藏宝点A

   x-5y 或 *x-10y*

   期望：*x-7.5y*

4. 直接去藏宝点B

   同上



居然不错！

❖思 路

- 对一些问题来说，比起花费时间寻找最优选择，随机选择可能更好
- 考虑的因素:
  - 随机选择和最优选择有多大不同?
  - 作出最优选择要花多少时间?
- 随机算法对同样的问题可能获得不同的结果
- 从引例2可知:
  - 选择1和2是确定策略
  - 选择3和4是随机策略
  - 很清楚，随机策略是较好

❖基本概念

- 确定算法与随机算法

考虑某个问题的一个具体例子:

- 确定算法每次运算都相同，肯定给出正确的解.

- 随机算法每次运算可能不同，可能返回正确的解。

①概率算法中，包括一处或若干处随机选择，根据随机值来决定算法的运行

② 每次运算结果可能不同，可能需要运行算法多次

③可能返回错误的解，但可以限定其出错概率

④对于相同的输入实例，概率算法的执行时间可能不同。

随机算法一直是可用和适合的吗？

- 和问题相关

  - 一些问题可能需要正确的解
  - 一些问题可能只需要近似解
  -一些问题要求多样性的解

- 如果一个问题没有有效的确定性算法可以在一个合理的时间内给出解答，但是，该问题能接受小概率的错误，那么，采用快速的概率算法就是一个合理的选择。
- 期望时间

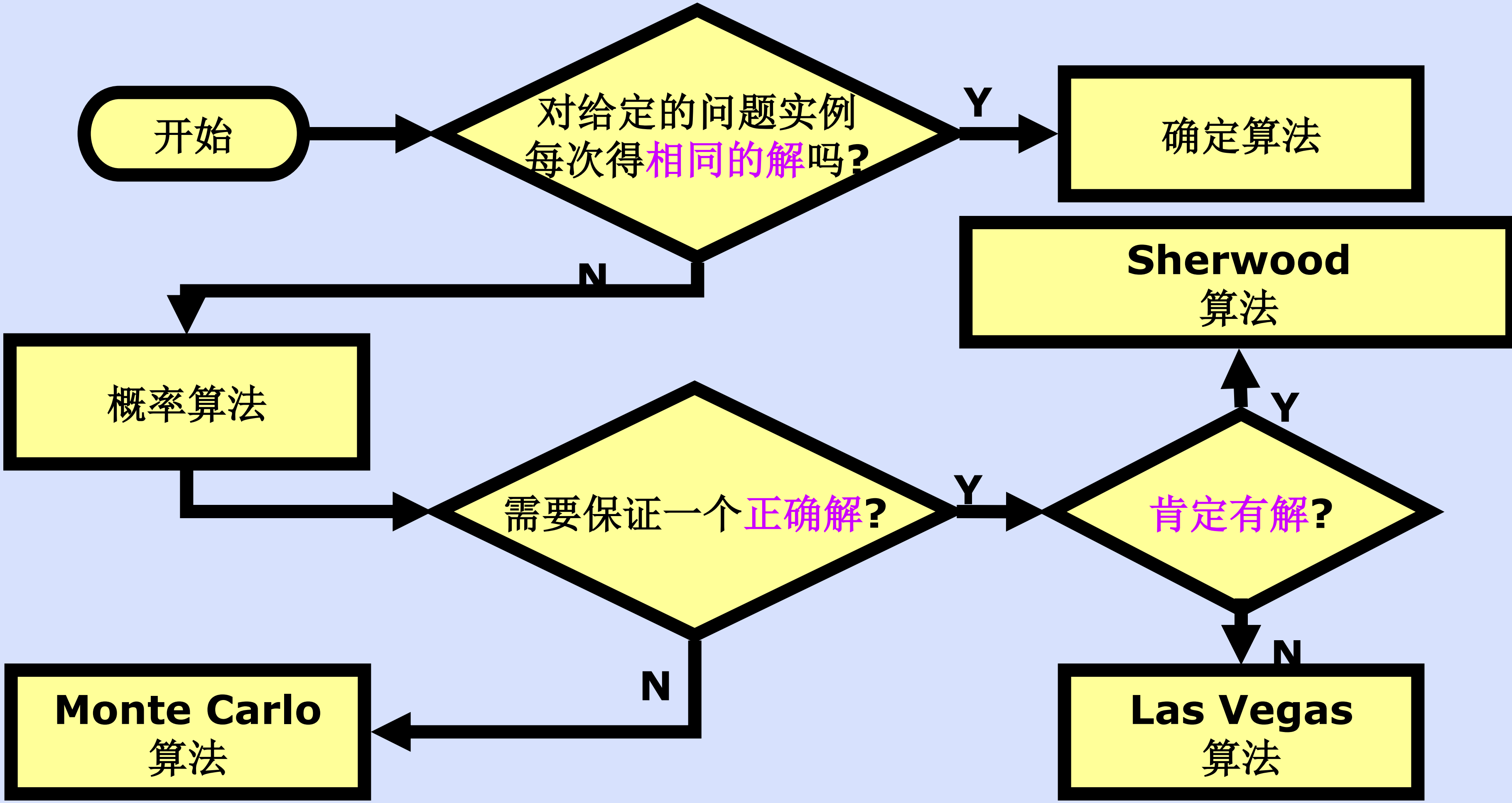
  -概率算法反复运行同一输入实例所得的平均运行时间

- 优点

  - 较之解决同一问题最好的确定性算法，概率算法所需的运行时间或空间通常小一些

  - 就迄今已发明的概率算法来看，它总是无一例外地易于理解和实现

概率算法的分类



开始
对给定的问题实例
每次得相同的解吗?
Y
确定算法
N
Sherwood
算法
概率算法
Y
需要保证一个正确解?
Y
肯定有解?
N
N
Monte Carlo
算法
Las Vegas
算法

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（**Sherwood**）算法
- 拉斯维加斯（**Las Vegas**）算法
- 蒙特卡罗（**Monte Carlo**）算法
- 总结

随机数

❖问题的提出

– 在概率算法中，需要进行随机选择，再对其进行测试或计算

– 随机选择的基础：生成某一范围内的随机数

❖线性同余法

$$\begin{cases} a_0 = d \\ a_n = (ba_{n-1} + c) \bmod m \qquad n = 1,2,\cdots \end{cases}$$

其中，***b***≥**0**，***c***≥**0**，***m***＞**0**，***d***≤***m***，***d***称为随机数发生器的随机种子

❖随机数算法的调用

- **random(*n*):** 产生**0 — *n*–1**之间的一个随机整数

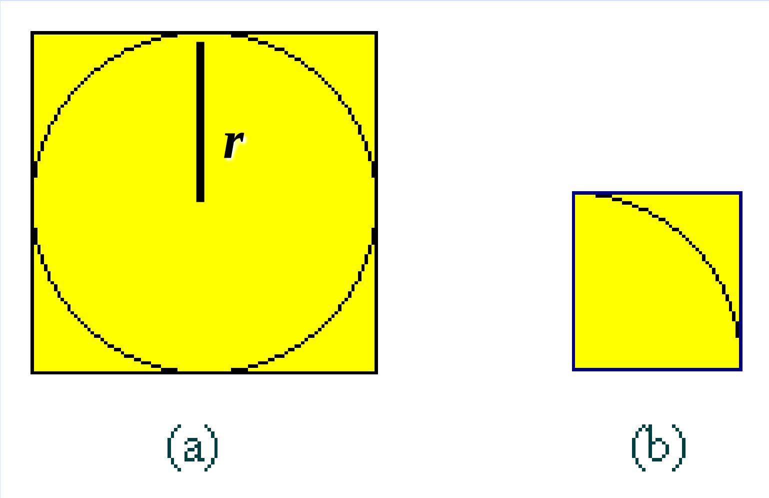
- **fRandom():** 产生**0 — 1**之间的一个随机实数

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（**Sherwood**）算法
- 拉斯维加斯（**Las Vegas**）算法
- 蒙特卡罗（**Monte Carlo**）算法
- 总结

数值概率算法

问题：计算π值



算法：

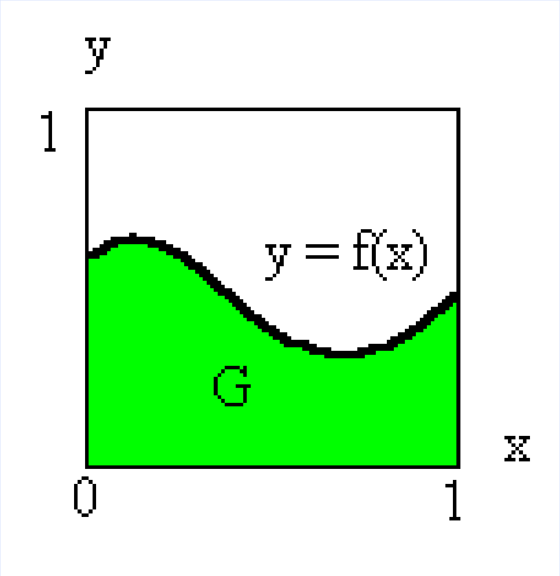
```
darts(int n) {
   int k=0;
   for (i=1;i <=n;i++){
      x=dart.fRandom();
      y=dart.fRandom();
      if ((x*x+y*y)<=1) k++;
   }
   return 4*k/(double)n;
}
```

基本思想：

- 向正方形随机投掷n个点，落入圆内k个；点均匀分布
- 点落入圆内的概率为 $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$
- n足够大时， $\pi \approx \frac{4k}{n}$

问题：计算定积分

***f*(*x*)在[0, 1]上连续，0≤ *f*(*x*) ≤1，需要计算积分** $I=\int_0^1 f(x)dx$



基本思想：

- 向正方形随机投掷***n***个点，点落在曲线下面的概率为：

$$P_r\{y\le f(x)\}=\int_0^1\int_0^{f(x)}dydx=\int_0^1 f(x)dx$$

- 如果有***m***个点落入***G***内，则随机点落入***G***内的概率为：

$$I\approx\frac{m}{n}$$

数值概率算法小结：

– 得到的往往是问题的近似解，但总能得到问题的一个答案

– 投掷的点数越多→ 运行时间越长→ 答案越精确

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（Sherwood）算法
- 拉斯维加斯（Las Vegas）算法
- 蒙特卡罗（Monte Carlo）算法
- 总结

舍伍德算法

引例：随机快速排序算法

❖ 问题的提出

- 快速排序算法在输入数据是均匀分布时平均时间为$O(n\log n)$

  最坏情况为$O(n^2)$

- 问题：如何保证对任意输入实例平均运行时间为$O(n\log n)$？

❖ 基本思想

- 引入随机元素：在*low*和*high*之间随机选择一个位置*v*，*A*[*v*]为*pivot*，并与*A*[*low*]交换，即符合partition划分方法

算法

输入：包含n个元素的数组

输出：经过排序的n个元素的数组

1. 若数组包含0或1个元素则返回
2. 从数组中随机选择一个元素作为枢轴元素
3. 把数组元素分为三个子数组，按照A,B,C顺序排列

   A：包含比枢轴元素小的元素；

   B：包含与枢轴元素相等的元素；

   C：包含比枢轴元素大的元素.
4. 对A和C递归地执行上述步骤.

算法分析

定理 设数组含n个不同元素，随机快速排序算法的**期望**比较次数 $T(n) \le 2\,n \ln n$.

解为$T(n)=O(n\log n)$，与确定型的快速排序算法**平均时间复杂度**一样.

证明：选定枢轴元素后，其余n-1个元素每个都要与枢轴元素比较一次，以决定在子数组A,B,C 中的归属。

枢轴元素在排序后的数组中可能位于第i个位置上(i=1,2,…,n)，A和C中的元素个数分别可能是i个和n-i-1个。

由于枢轴元素是随机选取的，枢轴元素在排序后的数组中位于第i个位置上(i=1,2,…,n)的概率都是1/n，所以有递推式

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

$T(n) \le 2\, n \ln n$

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

子问题的大小从0到n-1,共n种情况

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1}T(i)$$

$$T(n) \le (n-1) + \frac{2}{n}\sum_{i=1}^{n-1}(2i\ln i)$$

归纳假设，假设小于n时上式成立，下面证明T(n)时上式也成立

$$\le (n-1) + \frac{2}{n}\int_1^n (2i\ln i)di$$

用积分作为上界（放大）

$$\le (n-1) + \frac{2}{n}(n^2\ln n - n^2/2 + 1/2)$$

$$\le 2n\ln n$$

随机选择算法

算法RandSelect(A, p, r, k)

1. if $p=r$ then return $A[p]$
2. $i \leftarrow$ Random(p, r)
3. 以$A[i]$ 为标准划分A
4. $j \leftarrow$小于等于$A[i]$的数的个数
5. if $k \le j$
6. then return RandSelect($A,p,p+j-1,k$)
7. else return RandSelect($A,p+j,r,k-j$)

与Select算法的区别：

划分元素由确定性选择变成随机选择

时间期望值估计

假设 1, ..., n 中每个数被选概率相等,第k个数出现在划分后较大数组（较坏的情况）

若n为偶数，期望时间为

$$T(n) \le \frac{1}{n}\left[T(n-1) + T(n-2) + \cdots + T\left(\frac{n}{2}+1\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}+1\right) + \cdots + T(n-1)\right] + O(n)$$

$$T(n) \le \frac{2}{n}\sum_{i=n/2}^{n-1} T(i) + O(n)$$

合并化简

$T(n)\le cn$的证明

$$T(n) \le \frac{2}{n}\sum_{i=n/2}^{n-1} T(i) + O(n)$$

归纳步骤：假设对$k<n$命题为真

$$T(n) \le \frac{2}{n}[c\frac{n}{2} + c\left(\frac{n}{2}+1\right) + \cdots + c(n-1)] + tn$$

$$= \frac{2c}{n} \times \frac{(\frac{n}{2}+n-1)\frac{n}{2}}{2} + tn$$

$$= \frac{3cn}{4} - \frac{c}{2} + tn \le \frac{3cn}{4} + \frac{c}{c}tn$$

$$= \left(\frac{3}{4} + \frac{t}{c}\right)cn \le cn \quad \text{取}c \ge 4t\text{即可}$$

归纳假设等差数列求和

n为奇数，可类似证明T(n)≤cn.

❖舍伍德算法小结

- 总能得到问题的一个解，且求出的答案总是正确的
- 适用情况：

问题的确定算法在最坏情况和平均情况下计算复杂性差异较大，而又试图得到对任意输入均有较好计算性能的算法

- 基本出发点：

通过随机选择消除最坏情况与特定输入实例之间的依赖

（注意：不是避免最坏情况的发生）

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（**Sherwood**）算法
- 拉斯维加斯（**Las Vegas**）算法
- 蒙特卡罗（**Monte Carlo**）算法
- 总结

拉斯维加斯算法

- 引例

对n后问题的任何一个解而言，每一个皇后在棋盘上的位置无任何规律, 不具有系统性，而更象是随机放置的

设计思想

确定型算法：回溯搜索——深度优先

随机n后搜索算法的设计思想

每一步对搜索方向的确定型选择改成随机选择

问题

是否保证找到解？

*n*后问题的拉斯维加斯算法

-在棋盘上相继的各行中随机地放置皇后，并使新放置的皇后与已放置的皇后互不攻击。

结果：

成功：**n**个皇后均已相容地放置好，返回一个解

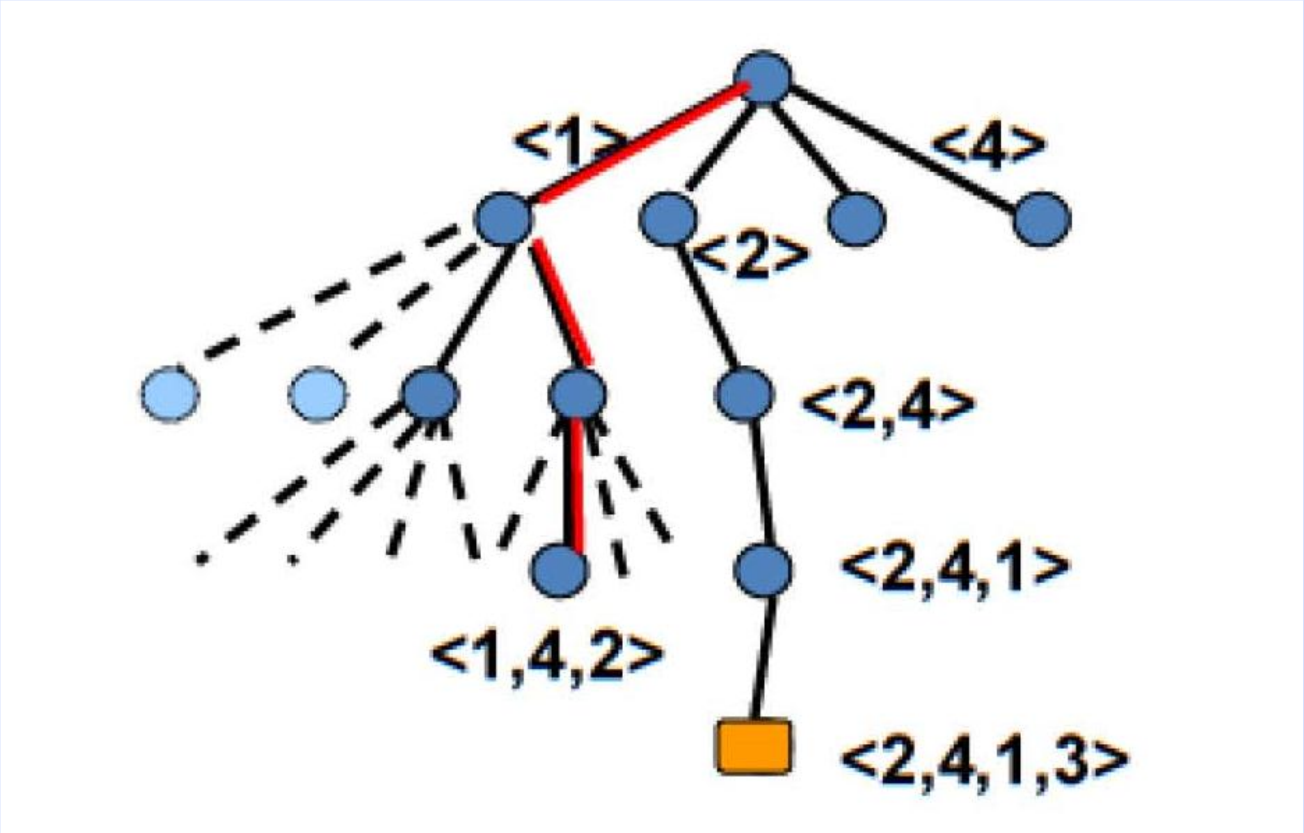
失败：下一个皇后没有可放置的位置，返回一个失败信息

while循环结束：
k>n或者count=0

```
BoolQueen
k=1, count=1
While (k<=n and count>0){  \\考察第k个皇后的位置
  count=0, j=0            \\count是合法位置计数器
  for(i=1;i<=n;i++){      \\考察所有列位置
   x[k]=i
   if(place(k))
    if (random(++count)==0) j=i  \\多个合法位置中随机选择一个
  }
  if(count>0) x[k++]=j
}
return (count>0)
```

- 为皇后*k*随机放置（列）
- random(n): 产生0 — n–1之间的一个随机整数

算法分析（4皇后为例）



正确性: 算法如果找到解，这个解就是正确的. 但一次搜索（红色路径）不一定找到解.

n后问题的随机算法

解决方案:

多次重复调用随机算法BoolQueen

直到找到解为止.

算法QueenLV(n)

1. $p \leftarrow$ BoolQueen(n) //放好的皇后数
2. while $p<n$ do
3. $p \leftarrow$ BoolQueen(n)

算法效率

问题:

多次重复调用随机算法，尽管每次时间为$O(n)$，但总时间复杂度可能很高.

改进算法---与回溯相结合

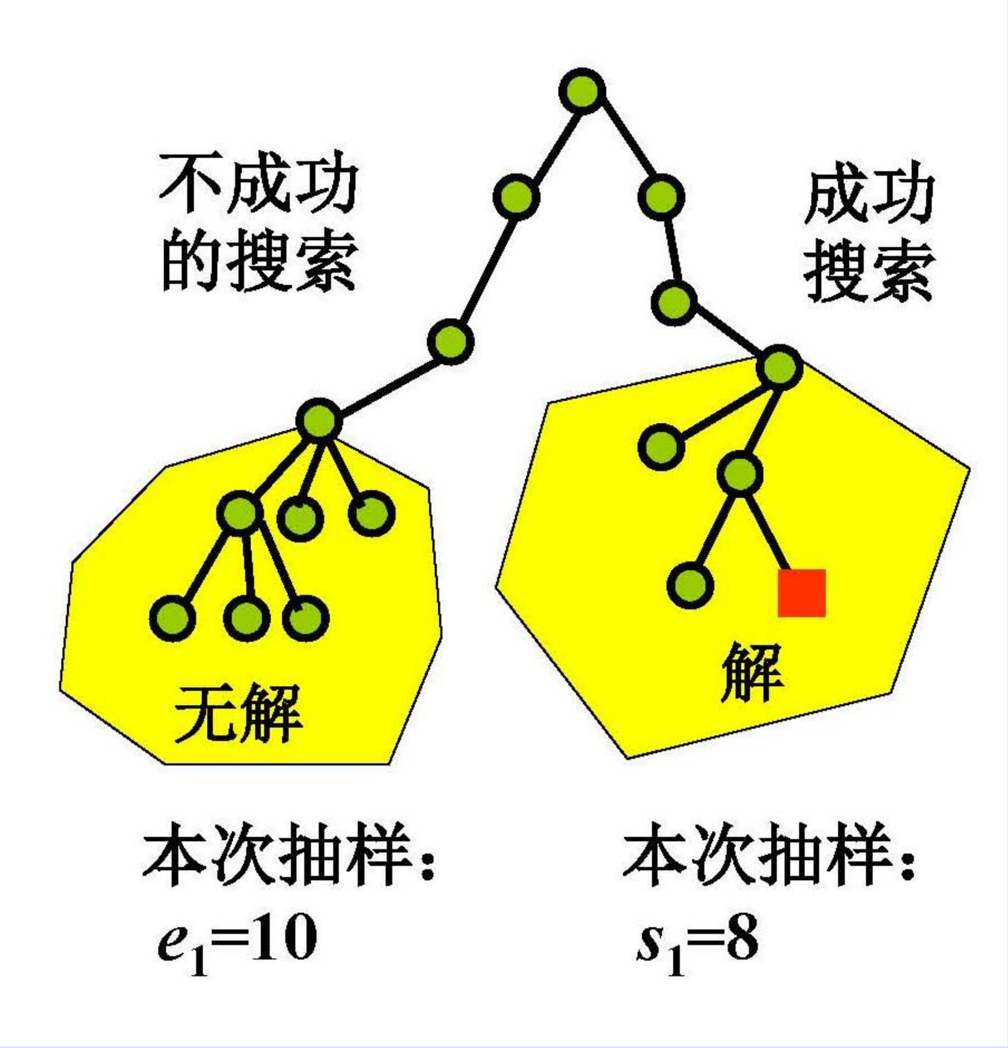
stopVegas≤*n*, 随机选择放置皇后数

n−*stopVegas*个皇后用回溯方法放置

stopVegas= 0 是完全的回溯算法

stopVegas=*n* 是完全Las Vegas算法

成功搜索与不成功搜索



***s*1**：成功搜索访问结点数

***e*1**：不成功搜索访问结点数

改进算法的分析

对于不同的*stopVegas* 值，设

p：算法一次运行成功的概率

s：一次成功搜索访问结点数平均值

e：一次不成功搜索访问结点数平均值

t：算法找到一个解的平均时间

$$\boldsymbol{t = ps + (1 - p)(e + t)}$$

$$\boldsymbol{t = s + e\frac{1 - p}{p}}$$

分析的实例

$$t = s + e\frac{1-p}{p}$$

n=12时的统计数据：

StopVegas	*p*	*s*	*e*	*t*
0	1.0000	262.00	-	262.00
5	0.5039	33.88	47.23	80.39
12	0.0465	13.00	10.20	222.11

结论：*stopVegas* = 5时算法效率高

❖一般成功求解的时间

设x是一个输入实例

boolean ***LV(x,y)***是一个拉斯维加斯算法

$p(x) > 0$是调用一次LV获得问题一个解的概率，则$1-p(x)$未获得解的概率

$s(x)$和$e(x)$分别是算法LV对于实例x求解成功或失败所需的平均时间

考虑下面算法

```
obstinate(x,y){ \\反复调用LV，直到找到一个解
 boolean success=false;
 while(!success) success=LV(x,y)
}
```

设$t(x)$是算法obstinate找到实例x一个解的平均时间，则有

$t(x)=p(x)*s(x)+(1-p(x))(e(x)+t(x))$ 解方程有：

$$t(x) = s(x) + \frac{1-p(x)}{p(x)} e(x)$$

❖确定算法 vs. Las Vegas

可能的输出

- 正确解

可能的输出

- 正确解
- 没有解

运行时间

- 对少数实例快
- 对大多数实例慢

运行时间

- 慢的实例有一定概率变快
- 快的实例有一定概率变慢

❖拉斯维加斯算法小结

- 修改确定性算法得到,一般将算法某步确定型选择变成随机选择
- 要么无解，要么得到正确解
- 算法找到正确解的概率随所用计算时间的增加而提高
- 改进途径：与确定型算法结合，先随机选择，然后确定型搜索
- 提高LV算法效率的原则：尽快报告错误，即减少处理失败的时间

（八后问题 92 16777216 0.000548%）

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（Sherwood）算法
- 拉斯维加斯（Las Vegas）算法
- 蒙特卡罗（Monte Carlo）算法
- 总结

蒙特卡罗算法

主元素

-设T[n]是一个含有n个元素的数组，x是数组T的一个元素，如果数组中有一半以上的元素与x相同，则称元素x是数组T的主元素（Major Element）。

问题

输入：n 个元素的数组T

输出：如果存在主元素则输出“true”，

否则“false”

最明显的方法:

统计每个元素在数组中出现的次数, $O(n^2)$

确定型的算法:

先选中位数(平均$O(n)$，最坏$O(n^2)$)，然后计数这个数出现的次数($O(n)$).

随机算法

设计思想：

随机选择一个元素，测试出现次数

算法Majority(T,n)

1. $i \leftarrow$ Random(1,n)

2. $x \leftarrow T(i)$

3. 计数x 在T 中出现的次数k

4. if $k>n/2$ then return true

5. else return false

算法的正确性

回答**true:** 则T 存在主元素,算法正确;

回答**false:** T 仍可能存在主元素,算法可能出错.

如果主元素存在，回答正确概率大于**1/2**.

改进思路: 增加测试次数

BoolMajority(T,n)

1. if Majority(T,n)

2. then return true

3. else return Majority(T,n)

正确性的概率估计

设p为一次调用回答true的概率，BoolMajority 算法正确的概率为

$$\boldsymbol{p+(1-p)p=2p-p^2=1-(1-p)^2>\frac{3}{4}}$$

调用k 次Majority算法正确的概率为

$$\boldsymbol{p+(1-p)p+(1-p)^2p+\cdots++(1-p)^{k-1}p}$$

$$\boldsymbol{=1-(1-p)^k>1-2^{-k}}$$

调用次数k	1	2	3	4	5	6
正确概率>	0.5	0.75	0.875	0.938	0.969	0.985

改进算法

对于任意给定的ε>0, 如果要使出错概率

不超过ε，则调用次数k 满足

$$\mathbf{1-2^{-k} \geq 1-\varepsilon \Rightarrow (\frac{1}{2})^{k} \leq \varepsilon}$$

$$\Rightarrow \frac{1}{2^{k}} \leq \varepsilon$$

$$\Rightarrow 2^{k} \geq \frac{1}{\varepsilon}$$

$$\Rightarrow k \geq \left\lceil log \frac{1}{\varepsilon} \right\rceil$$

改进算法MCMajority

出错概率不超过ε

算法MCMajority(T,n,ε)

1. $k \leftarrow \log(1/\varepsilon)$
2. for $i \leftarrow 1$ to k
3. if Majority(T,n) then return true
4. return false

注：上述算法的时间复杂性为$O(n\log_2(1/\varepsilon))$，算法的时间复杂性通常由问题规模以及错误可接受概率确定！！

蒙特卡洛算法小结

- 蒙特卡洛算法偶然会产生一个错误解
- 对于任何实例，能以较高的概率获得一个正确解
- 通常情况下，无法判断解是否正确
- 设p是一个实数，$1/2 < p < 1$, 一个 p – 正确的蒙特卡罗算法以至少 p 的概率返回一个正确解。
- p 可能和问题实例的规模有关，但是和实例本身无关
- 改进结果 ——增加结果正确的概率

  - 多次调用，选择出现频率最高的解

素数的判定

测试一个整数n是否是素数?

简单的方法:

n依次除以2～$\sqrt{n}$ 之间的数

如果某次余数为0，则n是一个合数

否则，n是一个素数

算法的时间复杂度是O($\sqrt{n}$)

对于一个正整数n，n的位数为m=$\lceil \log_{10}(n+1) \rceil$,则$\sqrt{n} = 10^{m/2}$，算法的时间复杂性是O($10^{m/2}$)，时间复杂性是指数阶

费尔马（Fermat）小定理：

如果n是一个素数，且$0<a<n$，则$a^{n-1} \bmod n \equiv 1$。

例：7是一个素数，取a=5，则$a^{n-1} \bmod n = 5^6 \bmod 7 = 1$

素数测试：检测，设a=2

$2^{n-1} \bmod n \equiv 1$

如是，输出“素数”

否则，输出“合数”

测试算法　$a^{n-1} \bmod n \equiv 1$

算法Ptest1(*n*)

输入：奇整数*n*, *n* > 5

输出:“prime” 或者“composite”

1. if power (2,*n*−1,*n*) =1

then return“prime”

2. else return“composite”

问题：只对a=2进行测试，会出错.

如果*n*为合数且算法输出“素数”，则称*n* 为基2的伪素数. 例如341.

$$2^{340} (\bmod 341) = 1$$

改进算法

改进算法：随机选取2~n-1中的数，多次测试. 如n=341, 取a=3，则

$$3^{340} (\text{mod } 341) = 56$$

341不是素数.

算法Ptest2(n)

1. a←Random(2, n-1)

2. if power(a, n-1, n)=1

   then return “prime”

3. else return “composite”

改进算法的问题

Fermat小定理是必要条件，不是充分条件，满足该条件的也可能是合数.

对所有与n互素的正整数a都满足条件的合数n称为Carmichael数，如561,1105,1729,2465等.

例：651 693 055 693 681是一个合数　99.9965%

Carmichael数非常少，小于10^8的只有255个

二次探测定理：

如果n是一个素数，且$0<x<n$，则方程$x^2 \bmod n \equiv 1$的根只有$x=1$或$x=n-1$。

即：如果$x^2 \bmod p \equiv 1$，$x \neq 1$，$x \neq n-1$，则n不是素数

例：$x^2\ mod\ 12 \equiv 1$

$x=1$ 或$x=11$或$x=5$ 或$x=7$

结论：5 和7是非平凡的根, 12是合数

二次探测定理的使用

1. 确定一个x;
2. result=(x*x)%n; 计算 $x^2 \bmod n$
3. 如果 result==1， 并有x!=1， x!=n-1
   则n是合数

基本思想

找出素数的证据：

- 满足**Fermat**的数未必是素数，但不满足**Fermat**的数肯定不是素数

二次探测定理用于检测满足**Fermat**定理的合数

- 不满足**Fermat** 或者 是满足**Fermat**的合数，则“不是素数”

测试方法

1. 对一个n，随机选择一个a, 且$0<a<n$ ，求$a^{n-1} \bmod n \equiv 1$

   设p=n-1，有$a^{n-1} \bmod n = ((a^{p/2} \bmod n)*(a^{p/2} \bmod n)) \bmod n$

2. 递归分治，递归求$a^{p/2} \bmod n$

3. 递归过程中，$x=a^{p/2} \bmod n (0<x<n)$, result=(x*x) mod n

   （1）result=1（$x^2 \bmod n$ 利用二次探测原理）

   （2）result=$((a^{p/2} \bmod n)*(a^{p/2} \bmod n)) \bmod n$（若p是偶数）

   result=(result*a)%n （若p是奇数）

4. 如果$a^{n-1} \bmod n=1$, 则n是素数，否则n是合数（利用费马小定理）

素数测试的随机算法

```
 int power(int a,int p,int n)
{
  if (p==0) result=1;
  else {
    x=power(a,p/2,n);
    result=(x*x)%n;
    if ((result==1)&&(x!=1)
        &&(x!=n-1)){
      composite=true;
      return;
    }
    if ((p%2)==1)
      result=(result*a)%n;
   }
 return result;
}
```

```
boolean prime(int n)
  composite=false;
  a=random(2,n-1);
  result=power(a,n-1,n);
  if (composite||(result!=1))
    return false;
  else return true;
}
```

p(=3/4) – 正确的蒙特卡洛算法

提纲

- 动机和应用背景
- 随机数
- 数值概率算法
- 舍伍德（**Sherwood**）算法
- 拉斯维加斯（**Las Vegas**）算法
- 蒙特卡罗（**Monte Carlo**）算法
- 总结

总结

Las Vegas型随机算法

Las Vegas型随机算法

将确定型算法的某些选择变成随机选择

得到解总是正确的

运行时间本身是一个随机变量

有效的LasVegas型算法

期望运行时间是输入规模的多项式

总是给出正确答案

Monte Carlo型随机算法

- Monte Carlo型随机算法
  - 算法有时会给出错误的答案
  - 运行时间和出错概率都是随机变量
  - 通常需要分析算法出错概率
- 有效的蒙特卡洛型算法
  - 多项式时间内运行
  - 出错概率不超过1/3

随机算法评价

随机算法的局限性

错误概率有界的多项式时间随机算法不太可能解决NP完全问题

实践效果

对于许多实际问题，随机算法可能比最好的确定型算法更快


谢谢大家！