《算法设计与分析》

第十一章 概率算法

马丙鹏

2023年12月04日

中国科学院大学
University of Chinese Academy of Sciences

第十一章 概率算法

- 11.1 概述
- 11.2 数值概率算法
- 11.3 舍伍德型概率算法
- 11.4 拉斯维加斯型概率算法
- 11.5 蒙特卡罗型概率算法

11.1 概述

- 1. 概率算法的设计思想
  - 假设你意外地得到了一张藏宝图，
  - 可能的藏宝地点有两个，要到达其中一个地点，或者从一个地点到达另一个地点都需要 5 天的时间。
  - 你需要 4 天的时间解读藏宝图，得出确切的藏宝位置，但是一旦出发后就不允许再解读藏宝图。
  - 有另外一个人知道这个藏宝地点，每天都会拿走一部分宝藏。
  - 有一个小精灵可以告诉你如何解读藏宝图，它的条件是，需要支付给它相当于知道藏宝地点的那个人 3 天拿走的宝藏。
  - 如何做才能得到更多的宝藏呢？

11.1 概述

■ 1. 概率算法的设计思想

□假设宝藏的总价值是 x，知道藏宝地点的那个人每天拿走宝藏的价值是 y，并且 $x>9y$，可行的方案有：

① 自己解读。用 4 天的时间解读藏宝图，用 5 天的时间到达藏宝地点，可获宝藏价值：$x-9y$。

② 接受小精灵的条件。用 5 天的时间到达藏宝地点，可获宝藏价值 $x-5y$，但需付给小精灵宝藏价值 $3y$，最终可获宝藏价值：$x-8y$。

③ 碰运气。投掷硬币决定首先前往哪个地点，如果发现地点是错的，就前往另一个地点。这样，有一半的机会获得宝藏价值 $x-5y$，另一半的机会获得宝藏价值 $x-10y$，从概率的角度，最终可获宝藏价值：$x-7.5y$。

11.1 概述

■ 1. 概率算法的设计思想

- 当算法在执行过程中面临一个选择时，有时随机选择算法的执行动作可能比花费时间计算哪个是最优选择要好。
- 随机从某种角度来说就是运气，但是在算法中增加这种随机性的因素，通常可以引导算法快速地求解问题。
- 概率算法允许算法在执行过程中随机选择下一步该如何进行，同时允许结果以较小的概率出现错误，并以此为代价，获得算法运行时间的大幅度减少。

11.1 概述

■ 1. 概率算法的设计思想

□ 例：请判断表达式$f(x_1, x_2, \ldots, x_n) \equiv 0$是否正确

- 随机生成一个n元向量$(r_1, r_2, \ldots, r_n)$，
- 如果$f(r_1, r_2, \ldots, r_n)$不等于0，则$f(x_1, x_2, \ldots, x_n)$不恒等于0
- 如果$f(r_1, r_2, \ldots, r_n)$等于0，
  - ✓说明$f(x_1, x_2, \ldots, x_n) \equiv 0$
  - ✓或者$(r_1, r_2, \ldots, r_n)$比较特殊
  - ✓重复多次，如果继续$f(r_1, r_2, \ldots, r_n)$等于0，那就可以得出$f(x_1, x_2, \ldots, x_n) \equiv 0$的结论
  - ✓测试的随机向量越多，这个结果出错的概率越小。

11.1 概述

■ 1. 概率算法的设计思想

□概率算法的基本特征：

① 概率算法对于相同的输入实例，概率算法的执行时间可能不同；

② 概率算法的结果不能保证一定是正确的，但可以限定其出错概率；

③ 概率算法在不同的运行中，对于相同的输入实例可能会得到不同的结果。

11.1 概述

■ 1. 概率算法的设计思想

- 如果一个问题没有找到有效的确定性算法可以在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，概率算法也许可以快速找到这个问题的解。
- 概率算法通常分析平均情况下的期望时间复杂度，即在相同输入实例上重复执行概率算法的平均时间。

11.1 概述

■ 2. 概率算法的分类

□数值概率算法

- ➢常用于数值问题的求解。
- ➢这类算法所得到的往往是问题的近似解。
- ➢近似解的精度随着时间的增加而不断增加。
- ➢在很多情况下，求解精确解不可能或不必要，此法可得相当满意的解。

11.1 概述

■ 2. 概率算法的分类

□舍伍德（Sherwood）算法

- ➢虽然在某些步骤引入随机选择，但该算法总能求得问题的一个解，且所求得的解总是正确的。
- ➢当一个确定性算法在最坏情况下的计算复杂性与其平均情况下的计算复杂性有较大差别时，可在确定性算法中引入随机性将它改造成一个舍伍德算法，消除或减少问题的好坏实例间的差别。
- ➢精髓：消除最坏情形与特定实例之间的关联性。

11.1 概述

■ 2. 概率算法的分类

□拉斯维加斯(Las Vegas)算法

- ➢该算法不会得到不正确的解。
- ➢一旦用拉斯维加斯找到一个解，这个解一定是正确解。
- ➢但有时该算法找不到解。
- ➢找到解的概率随它所用时间的增加而提高。
- ➢对所求解的任一实例，用同一拉斯维加斯算法求解足够多次，可使求解失败概率任意小。

11.1 概述

■ 2. 概率算法的分类

□蒙特卡罗（Monte Carlo）算法

- ➢蒙特卡罗算法用于求问题的准确解。
- ➢有些问题近似解没有意义，如“y/n”的判定问题、求一个整数的因子等。
- ➢用蒙特卡罗算法总能求得一个解，但这个解未必是正确的。
- ➢求得正确解的概率随它所用的计算时间增加而提高。
- ➢一般情况下，无法有效判定所得解是否肯定正确。

11.1 概述

■ 3. 随机数生成器

- 目前，在计算机上产生随机数序列还是一个难题。
- 计算机产生随机数的方法通常采用线性同余法，产生的随机数序列为（$a_0, a_1, \ldots, a_n, \ldots$），满足：

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

- 其中，$b \geq 0$，$c \geq 0$，$m > 0$，$d \leq m$。d 称为随机种子。
- 当 b、c 和 m 的值确定后，如果随机种子相同，一个随机数生成器将会产生相同的随机数序列
- 严格地说，随机数只是一定程度上的随机（伪随机数）

11.1 概述

■ 3. 随机数生成器

- 程序设计语言一般都会提供随机数生成器，例如，C++语言提供库函数rand( )可以产生[0, 32767]之间的随机整数。

```
int Random(int a, int b)
//产生分布在任意区间[a, b]的随机数
{
   return (rand( )%(b-a) + a);
}
```

第十一章 概率算法

- 11.1 概述
- 11.2 数值概率算法
- 11.3 舍伍德型概率算法
- 11.4 拉斯维加斯型概率算法
- 11.5 蒙特卡罗型概率算法

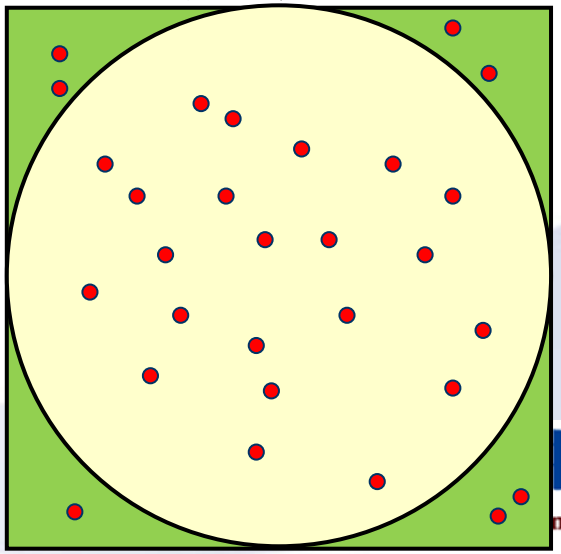
11.2 数值概率算法

- 1. 随机数生成器
  - 用于数值概率问题中求近似解的算法
  - 预期精度可随着算法可用时间的增加而提高
  - 有时答案以置信区间的形式给出

11.2 数值概率算法

■ 2. 圆周率计算

- 向一个正方形目标投掷n个飞镖，并计算落在该正方形中的圆圈内飞镖的数字k。我们假设正方形中的每个点被飞镖击中的概率完全相同。
- 内切圆的半径是r，那么它的面积就是πr^2, 而正方形的面积是$4r^2$
- 落在圆内的飞镖的平均比值是$\pi r^2/4r^2 = \pi/4$



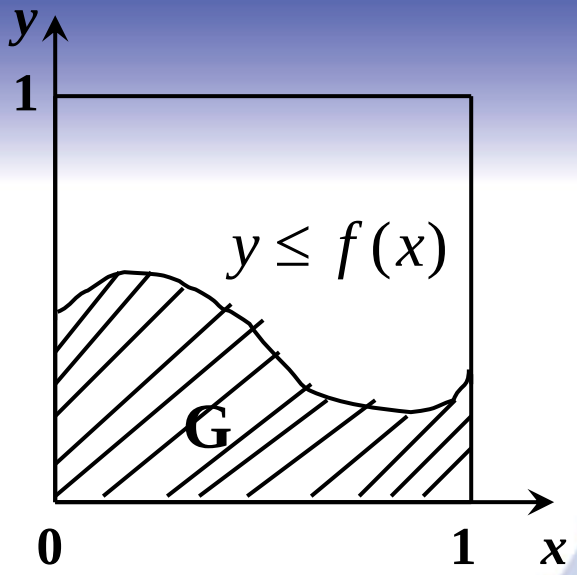
$$\pi \approx 4k/n$$

11.2 数值概率算法

■ 2. 圆周率计算

```
Function darts(n)
    k←0
    for i=-1 to n
        x ← uniform (0, 1)//按均匀分布产生0和1之间的随机数
        y ← uniform (0, 1)
        if x² + y² ≤ 1 then k ← k+1 endif
    endfor
return 4k/n
```

11.2 数值概率算法



■ 3. 计算定积分

- □f(x)是[0,1]上的连续函数，且 0≤f(x) ≤ 1
- □试计算积分值 $I = \int_0^1 f(x)dx$
- □向单位正方形内均匀地作随机投点试验，则随机点落在曲线 y=f(x)下面的概率为

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

- □向单位正方形内随机投入n 个点$(x_i, y_j), i = 1,2,\cdots,n$，若随机点 (x_i, y_i) 落入G内，则 $y_i \le f(x_i)$。如果有m个点落入G内，则m/n近似等于随机点落入G 内的概率，即 I ≈m/n, 据此可以设计出计算积分的概率算法。

11.2 数值概率算法

■ 3. 计算定积分

```
double Darts(int n)
{                                    //计算积分的数值概率算法
    static RandomNumber dart;
    int k = 0;
    for (int i = 1; i <= n; i + +) {
        double x = dart.fRandom( );
        double y = dart.fRandom( );
     if ( y <= f(x) )  k + +;
     }
    return k/double(n);
}
```

11.2 数值概率算法

■ 4. 解非线性方程组

□设要求解非线性方程组：

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \cdots\cdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

□其中x_i是实变量，f_i是非线性实函数。

□数值方法有线性化方法、求函数极小值方法等。但有时会遇到一些麻烦，甚至方法失效得不到一个近似解。

□概率算法思想简单，易于实现，实际使用中比较有效。当然，概率法往往耗费较多时间、精度不高等问题。对精度要求较高的问题，可提供一个较好的初值。

$$f(\mathrm{X}, n) = \sum_{i=1}^{n} f_i^{\,2}(X)$$

11.2 数值概率算法

■ 4. 解非线性方程组

- □解法：构造函数 $f(X,n)=\sum_{i=1}^{n} f_i^2(X)$，其中$X=(x_1, x_2, \ldots, x_n)$
- □该函数的零点即为一组解。
- □用随机搜索法。
  - ➢选定一个随机点X_0作为出发点，
  - ➢设第j步的搜索点为X_j，
  - ➢先随机计算搜索方向r，再根据目标值修改搜索步长a，得到随机搜索增量Δx_j，得第j+1步搜索点$x_{j+1}=x_j+\Delta x_j$。
  - ➢当$fx(X_{j+1})<\varepsilon$时，取X_{j+1}为近似解，否则继续搜索。

第十一章 概率算法

- 11.1 概述
- 11.2 数值概率算法
- 11.3 舍伍德型概率算法
- 11.4 拉斯维加斯型概率算法
- 11.5 蒙特卡罗型概率算法

11.3 舍伍德型概率算法

■ 1. 设计思想

- 很多算法对于不同的输入实例，运行时间差别很大。此时，可采用舍伍德型概率算法来消除算法的时间复杂度与输入实例间的依赖关系。
- 舍伍德型概率算法的有两种应用方式：

① 在确定性算法的某些步骤引入随机因素，将确定性算法改造成舍伍德型概率算法；

② 借助于随机预处理技术，不改变原有的确定性算法，仅对输入实例进行随机处理（称为洗牌），然后再执行确定性算法。

- 舍伍德型概率算法设法消除了算法的不同输入实例对算法时间性能的影响，对于任何输入实例，舍伍德型概率算法能够以较高的概率与原有的确定性算法在平均情况下的时间复杂度相同。

11.3 舍伍德型概率算法

■ 1. 设计思想

□一个线性时间的洗牌算法，实现对输入实例进行随机处理。

```
void RandomShuffle(int r[ ], int n)
{
    int i, j, k = n/2, temp;
    for (i = 0; i < k; i++)
    {
        j = Random(0, n-1); //随机选择一个元素
        temp = r[i]; r[i] = r[j]; r[j] = temp; //交换r[i]和r[j]
    }
}
```

11.3 舍伍德型概率算法

■ 2. 快速排序

□ 问题描述

- ➢ 设计快速排序的舍伍德型概率算法

□ 求解思路

- ➢ 快速排序算法的关键是在一次划分中选择合适的划分元素作为划分的基准，
- ➢ 如果划分元素是序列中最小（或最大）元素，则一次划分后，得到的两个子序列不均衡，使得快速排序的时间性能降低。

11.3 舍伍德型概率算法

■ 2. 快速排序

□求解思路

初始序列	6	35	19	23	30	12	58
一次划分结果	6	35	19	23	30	12	58

(a) 以最小值6为划分元素，划分不均衡

初始序列	6	35	19	23	30	12	58
随机选择划分元素	23	35	19	6	30	12	58
一次划分结果	12	6	19	23	30	35	58

(b) 随机选择划分元素，划分均衡

11.3 舍伍德型概率算法

■ 2. 快速排序

□求解思路

➢可以采用如下舍伍德型概率算法：

① 在一次划分之前，在待排序序列中随机确定一个元素作为划分元素，并与第一个元素交换，则一次划分后得到期望均衡的两个子序列。

② 在执行快速排序之前调用洗牌函数**RandomShuffle**，将待排序序列随机排列。

➢这两种方法都能够以较高的概率避免快速排序的最坏情况。

11.3 舍伍德型概率算法

■ 2. 快速排序

□算法实现

在快速排序算法执行一次划分之前引入随机选择，得到舍伍德型概率算法进行快速排序，程序如下：

```
void RandQuickSort(int r[ ], int low, int high)
{
   int i, k, temp;
   if (low < high)
   {
      i = Random(low, high);                    //随机选择r[i]作为轴值
      temp = r[low]; r[low] = r[i]; r[i] = temp;
      k = Partition(r, low, high);
      RandQuickSort(r, low, k-1);
      RandQuickSort(r, k+1, high);
   }
}
```

11.3 舍伍德型概率算法

■ 3. 二叉查找树

□ 问题描述

- 二叉查找树（binary search trees）是具有下列性质的二叉树：

① 若它的左子树不空，则左子树上所有结点的值均小于根结点的值；

② 若它的右子树不空，则右子树上所有结点的值均大于根结点的值；

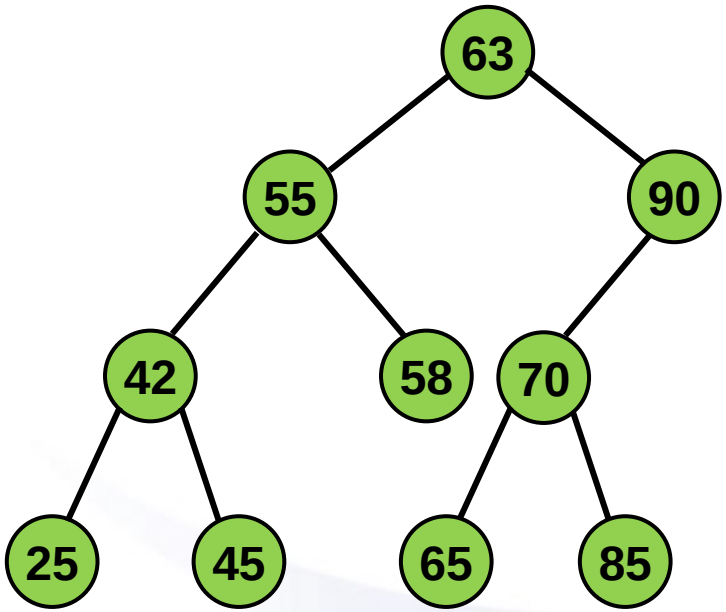
③ 它的左右子树也都是二叉排序树。

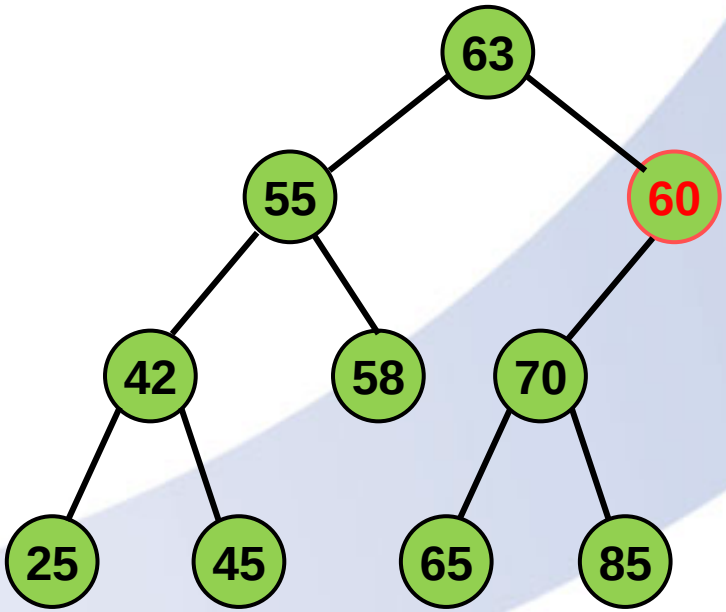
11.3 舍伍德型概率算法

■ 3. 二叉查找树

□ 问题描述

- ➢ 设计舍伍德型概率算法构造二叉查找树，使得二叉查找树左右子树的结点数大致相等。



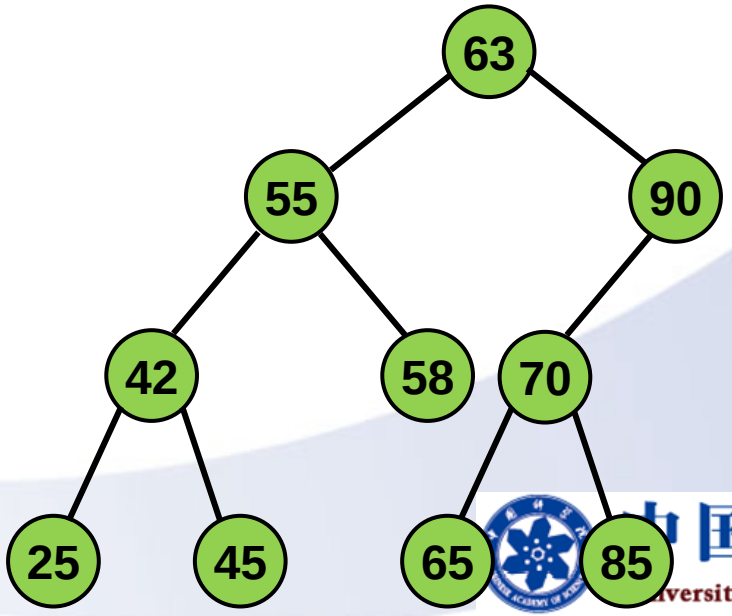


11.3 舍伍德型概率算法

■ 3. 二叉查找树

□求解思路

- ➢构造二叉查找树的过程是从空的二叉查找树开始，依次插入一个个结点。
- ➢例如，给定查找集合{63, 55, 42, 45, 58, 90, 70, 25, 85, 65}，构造二叉排序树的过程如下：



11.3 舍伍德型概率算法

■ 3. 二叉查找树

□求解思路

- ➢在二叉查找树的构造过程中，插入结点的次序不同，二叉查找树的形状就不同，而不同形状的二叉查找树可能具有不同的深度。构造二叉查找树的舍伍德型概率算法：

① 在插入每一个结点时，在查找集合中随机选定一个元素；

② 在执行构造算法之前调用洗牌函数**RandomShuffle**，将查找集合随机排列。

11.3 舍伍德型概率算法

■ 3. 二叉查找树

□算法实现

二叉查找树采用二叉链表存储，设root为指向二叉链表的根指针，舍伍德型概率算法采用洗牌方法，程序如下：

```
struct BiNode
{
    int data;
    BiNode *lchild, *rchild;
};
BiNode *InsertBST(BiNode *root, BiNode *s) {
    if (root == NULL) root = s;
    else if (s->data < root->data)
        root->lchild = InsertBST(root->lchild, s);
    else
        root->rchild = InsertBST(root->rchild, s);
    return root;
}
```

11.3 舍伍德型概率算法

```
BiNode *Creat(BiNode *root, int r[ ], int n)
{
   int i, j, temp;   BiNode *s = NULL;
   for (i = 0; i < n/2; i++)                    //执行洗牌操作
   {
      j = rand( ) % n;
      temp = r[i]; r[i] = r[j]; r[j] = temp;
   }
   for (i = 0; i < n; i++)
   {
      s = new BiNode; s->data = r[i];
      s->lchild = s->rchild = NULL;
      root = InsertBST(root, s);
   }
   return root;
}
```

11.3 舍伍德型概率算法

■ 3. 二叉查找树

□算法分析

- ➢洗牌操作的时间开销是$O(n)$，
- ➢插入第 i 个结点时，查找插入位置的操作不超过二叉查找树的期望深度$O(\log_2 i)$，
- ➢因此，算法的期望时间复杂度是$O(n\log_2 n)$。

第十一章 概率算法

- 11.1 概述
- 11.2 数值概率算法
- 11.3 舍伍德型概率算法
- 11.4 拉斯维加斯型概率算法
- 11.5 蒙特卡罗型概率算法

11.4 拉斯维加斯型概率算法

■ 1. 设计思想

- 拉斯维加斯型（Las Vegas）概率算法对同一个输入实例反复多次运行算法，直至运行成功，获得问题的解。
  如果运行失败，在相同的输入实例上再次运行算法。
- 拉斯维加斯型概率算法的基本特征：

① 拉斯维加斯型概率算法的随机性选择有可能导致算法找不到问题的解，即算法运行一次，或者得到一个正确的解，或者无解。

② 只要出现失败的概率不占多数，当算法运行失败时，在相同的输入实例上再次运行概率算法，就又有成功的可能。

11.4 拉斯维加斯型概率算法

1. 设计思想

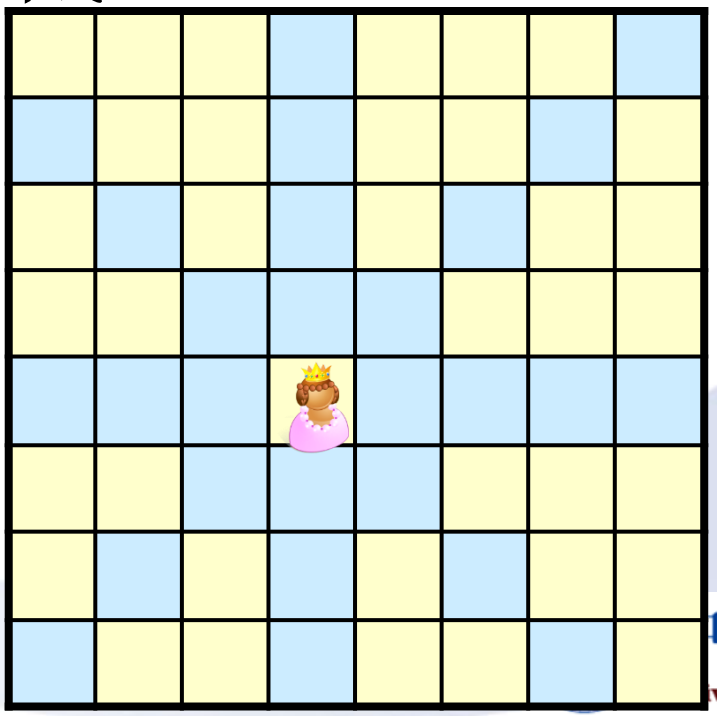
- 拉斯维加斯型概率算法一定能够找到解吗？
- 设 $p(x)$ 是对输入实例 x 调用拉斯维加斯型概率算法获得问题的一个解的概率，则一个正确的拉斯维加斯型概率算法应该对于所有的输入实例 x 均有$p(x) > 0$。
- 在更强的意义下，存在一个正的常数 δ，使得对于所有的输入实例 x 均有$p(x)>\delta$。
  由于 $p(x)>\delta$，所以，只要对算法运行的次数足够多，对任何输入实例 x，拉斯维加斯型概率算法总能找到问题的一个解。
- 拉斯维加斯型概率算法找到正确解的概率随着运行次数的增加而提高

11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□ 问题描述

- ➢ 在 8×8 的棋盘上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上。



11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□求解思路

- ➢对于八皇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，更像是随机放置的。
- ➢由此得到拉斯维加斯型概率算法：
  - ✓在棋盘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，直至八个皇后均已相容地放置好，或下一个皇后没有可放置的位置。

11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□算法实现

设n皇后问题的可能解用向量$X=(x_1, x_2, \ldots, x_n)$表示，其中，$1 \le x_i \le n$并且$1 \le i \le n$，即第 i 个皇后放置在第 i 行第 x_i 列，算法如下：

算法：八皇后问题拉斯维加斯型概率算法

输入：皇后的个数 n

输出：满足约束条件解向量 X

1. 试探次数 count 初始化为 0；
2. 循环变量 i 从 1~n 放置第 i 个皇后：
   2.1 j = [1, n]的随机数；
   2.2 count = count+1，进行第 count 次试探；
   2.3 若皇后 i 放置在位置 j 不发生冲突，则$x_i = j$；count=0；转步骤 2 放置下一个皇后；
   2.4 若(count == n)，则无法放置皇后 i，算法运行失败，结束算法；
   否则，转步骤 2.1 重新放置皇后 i；
3. 输出解向量$(x_1, x_2, \ldots, x_n)$；

11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□算法实现

设数组x[n]表示皇后的列位置，其中x[i]表示皇后 i 摆放在x[i]的位置：

```
int Queue(int x[ ], int n)
{
   int i, j, k, count = 0;
   for (i = 0; i < n; )
   {
      j = Random(0, n-1);                          //注意数组下标从0开始
      x[i] = j; count++;
      for (k = 0; k < i; k++)                      //检测约束条件
         if (x[i] == x[k] || abs(i - k) == abs(x[i] - x[k])) break;
      if (k == i)  { count = 0; i++; }             //不发生冲突，摆放下一个
      else if (count == n)  return 0;              //无法摆放皇后i
   }
   return 1;
}
```

11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□算法分析

- 随着棋盘上放置皇后数量的增多，算法试探的次数也随之增加。
- 如果将上述随机放置策略与回溯法相结合，则会获得更好的效果。

（1）先在棋盘的若干行随机地放置相容的皇后，其他皇后用回溯法继续放置，直至找到一个解或宣告失败。

（2）在棋盘中随机放置的皇后越多，回溯法搜索所需的时间就越少，但失败的概率也就越大。

11.4 拉斯维加斯型概率算法

■ 2. 八皇后问题

□算法分析

（3）例如八皇后问题，实验表明：

① 随机地放置两个皇后再采用回溯法比完全采用回溯法快大约两倍；

② 随机地放置三个皇后再采用回溯法比完全采用回溯法快大约一倍；

③ 所有皇后都随机放置比完全采用回溯法慢大约一倍。

11.4 拉斯维加斯型概率算法

■ 3. 整数因子划分问题

□问题描述

- ➢如果 *n* 是一个合数，则 *n* 必有一个非平凡因子*m*（即*m*≠1且*m*≠*n*），使得 *m* 可以整除 *n*。
- ➢给定一个合数 *n*，求 *n* 的一个非平凡因子的问题称为整数因子划分问题。

□求解思路

- ➢对一个正整数 n 进行因子划分，最自然的想法是试除，即 m 从 2 ~ $\sqrt{n}$
- ➢依次试除，如果 n mod m = 0，则 n 可以划分为 m 和 n/m;
- ➢如果余数均不为 0，则 n 是一个素数。

11.4 拉斯维加斯型概率算法

■ 3. 整数因子划分问题

□求解思路

- ➢显然，这个算法的时间复杂度是$O(\sqrt{n})$。
- ➢这是一个伪多项式时间算法，对于一个正整数 n，其位数为 $m=\lceil \log_{10}(n+1) \rceil$，时间复杂度是$O(10^{m/2})$。

```
int Split(int n)
{
  int m = floor(sqrt(double(n)));
  for (int i=2; i<=m; i++)
     if (n%i==0) return i;
  return 1;
}
```

11.4 拉斯维加斯型概率算法

■ 3. 整数因子划分问题

□ Pollard算法（Las Vegas 算法）

- 在开始时选取0～n-1范围内的随机数，然后递归地由 $x_i = (x_{i-1}^2 - 1) \bmod n$ 产生无穷序列 $x_1, x_2, \cdots, x_k, \cdots$
- 对于$i=2^k$，以及$2^k<j \le 2^{k+1}$，算法计算出x_j-x_i与n的最大公因子$d=gcd(x_j-x_i, n)$。
- 如果d是n的非平凡因子，则实现对n的一次分割，算法输出n的因子d。

11.4 拉斯维加斯型概率算法

■ 3. 整数因子划分问题

```
void Pollard(int n)
{// 求整数n因子分割的拉斯维加斯算法
  RandomNumber rnd;
  int i=1;
  int x=rnd.Random(n); // 产生随机整数
  int y=x; int k=2;
  while (true) {
      i++;
      x=(x*x-1)%n;    //
      int d=gcd(y-x, n);  // 求n的非平凡因子
      if ((d>1) && (d<n)) cout<<d<<endl;
      if (i==k) {y=x;  k*=2;}
   } }
```

```
int gcd (int a, int b)
  { // 求整数 a  和 b 的最大公因子的 Euclid 算法
     if ( b = = 0 ) return a ;
     else return gcd (b, a % b ) ;
  }
```

11.4 拉斯维加斯型概率算法

■ 3. 整数因子划分问题

□ Pollard算法（Las Vegas 算法）

- 对Pollard算法更深入的分析可知，执行算法的while循环约$\sqrt{p}$次后，Pollard算法会输出n的一个因子p。由于n的最小素因子p $\leq\sqrt{n}$ ，故Pollard算法可在$O(n^{1/4})$时间内找到n的一个素因子。

第十一章 概率算法

- 11.1 概述
- 11.2 数值概率算法
- 11.3 舍伍德型概率算法
- 11.4 拉斯维加斯型概率算法
- 11.5 蒙特卡罗型概率算法

11.5 蒙特卡罗型概率算法

■ 1. 设计思想

□对于许多问题来说，近似解毫无意义。例如：

① 判定问题的解为“是”或“否”，二者必居其一，不存在任何近似解。

② 整数因子划分问题，一个整数的近似因子没有任何意义。

11.5 蒙特卡罗型概率算法

■ 1. 设计思想

□蒙特卡罗型（Monte Carlo）概率算法的基本特征：

① 蒙特卡罗型概率算法用于求问题的准确解。

② 蒙特卡罗型概率算法偶尔会出错，但无论任何输入实例，总能以很高的概率找到一个正确解。

③ 蒙特卡罗型概率算法总是给出解，但是，这个解偶尔可能是不正确的，一般情况下，也无法有效地判定得到的解是否正确。

④ 蒙特卡罗型概率算法求得正确解的概率依赖于算法的运行次数，算法运行的次数越多，得到正确解的概率就越高。

11.5 蒙特卡罗型概率算法

■ 1. 设计思想

□蒙特卡罗型概率算法一定能够找到解吗？

➢设 p 是一个实数，且 $1/2<p<1$。

① 如果一个蒙特卡罗型概率算法对于问题的任一输入实例得到正确解的概率不小于 p，则称该蒙特卡罗型概率算法是 p正确的。

② 如果对于同一输入实例，蒙特卡罗型概率算法不会给出两个不同的正确解，则称该蒙特卡罗型概率算法是一致的。

③ 如果重复运行一致的 p正确的蒙特卡罗型概率算法，每一次运行都独立地进行随机选择，就可以使产生不正确解的概率变得任意小。

11.5 蒙特卡罗型概率算法

■ 2. 主元素问题

□ 问题描述：

- 设A[n]是含有 *n* 个元素的数组，*x* 是数组A[n]的一个元素，如果数组有一半以上的元素与 *x* 相同，则称元素 *x* 是数组A[n]的主元素。
- 例如，在数组A[7]={3, 2, 3, 2, 3, 3, 5}中，元素 3 就是主元素。

11.5 蒙特卡罗型概率算法

■ 2. 主元素问题

□求解思路：

- 蒙特卡罗型概率算法求解主元素问题可以随机地选择数组的一个元素A[i]进行统计，
- 如果该元素出现的次数大于 *n*/2，则该元素就是数组的主元素，算法返回 1；
- 否则随机选择的元素 A[i]不是主元素，算法返回 0，
- 本次运行结果表明，或者数组 A[n] 没有主元素，或者数组 A[n] 有主元素但不是元素 A[i]。
- 再次运行蒙特卡罗型概率算法，直至算法返回 1，或者达到给定的错误概率。

11.5 蒙特卡罗型概率算法

■ 2. 主元素问题

□算法实现：

➢求解主元素问题的蒙特卡罗型概率程序如下：

```
int MajorityMC(int A[ ], int n)
{
   int i, j, count = 0;
   i = Random(0, n-1);                    //随机选择一个数组元素
   for (j = 0; j < n; j++)
      if (A[j] == A[i]) count++;
   if (count > n/2) return A[i];          //A[i]是主元素
   else return 0;
  }
```

11.5 蒙特卡罗型概率算法

■ 2. 主元素问题

□算法分析：

- 如果数组存在主元素，算法MajorityMC将以大于1/2 的概率返回 1，即算法出现错误的概率小于 1/2。
- 重复运行算法 k次，算法返回 0的概率将减少为2^{-k}，则算法发生错误的概率为 2^{-k}。
- 对于任何给定错误概率 $e>0$，算法MajorityMC重复调用 $\lceil \log_2(1/e) \rceil$ 次，时间复杂度是$O(n\log_2(1/e))$。

11.5 蒙特卡罗型概率算法

■ 3. 素数测试

□问题描述：

- ➢素数的研究和密码学有很大的关系，素数测试是素数研究的一个重要课题。
- ➢给定一个正整数 n，素数测试要求判定 n 是否是素数。

□求解思路

- ➢采用概率算法进行素数测试的理论基础来自费尔马定理。
- ➢定理11-2（费尔马定理） 如果正整数 n 是一个素数，取任一正整数 a 且$1<a<n$，则 $a^{n-1} \bmod n \equiv 1$。

11.5 蒙特卡罗型概率算法

■ 3. 素数测试

□求解思路

- ➢例如 7 是一个素数，则$2^6 \bmod 7 = 1$，$3^6 \bmod 7 = 1$，$4^6 \bmod 7 = 1$，$5^6 \bmod 7 = 1$，$6^6 \bmod 7 = 1$
- ➢费尔马定理表明，如果存在一个小于 n 的正整数 a，使得 $a^{n-1} \bmod n \neq 1$，则 n 肯定不是素数。
- ➢费尔马定理只是素数判定的一个必要条件。
- ➢Carmichael数（卡迈克尔数）：
  - ✓满足费尔马定理的合数。
  - ✓当一个合数 n 对于整数 a 满足费尔马定理时，称整数 a 为合数 n 的伪证据。
  - ✓例如，341 是合数，取 $a=2$，$2^{340} \bmod 341 = 1$。

11.5 蒙特卡罗型概率算法

■ 3. 素数测试

□求解思路

- ➢为了提高素数测试的准确性，可以多次随机选取小于 n 的正整数 a，重复计算 $a^{n-1} \bmod n$ 来判定n是否是素数。
- ➢例如，对于 341，取 a=3，$3^{340} \bmod 341 = 56$，从而判定 341 不是素数。

11.5 蒙特卡罗型概率算法

■ 3. 素数测试

□算法实现

➢为了避免 a^{n-1} 超出 int 型的表示范围，每做一次乘法之后对 n 取模，而不是先计算 a^{n-1} 再对 n 取模。程序如下：

```
int FermatPrime(int n)
{
   int i, a, b = 1;
   a = Random(2, n-1);          //产生[2, n-1]之间的一个随机整数
   for (i = 1; i < n; i++)
      b = (b * a) % n;
   if (b == 1) return 1;        //可能是素数或Carmichael数
   else return 0;               //一定不是素数
}
```

11.5 蒙特卡罗型概率算法

■ 3. 素数测试

□算法分析

- ➢函数FermatPrime返回 0 时，整数 *n* 一定是合数；
- ➢如果返回 1，说明整数 *n* 可能是素数，还可能是Carmichael数。

End