# 第九章　随机算法

## 9.1 随机算法引言

## 9.2 舍伍德（Sherwood）算法

# 9.1 随机算法引言

## 确定性算法：

- 算法的每一个计算步骤都是确定的；
- 对于相同的输入，每一次执行过程都会产生相同的输出。

◆ 很多确定性的算法，其性能很坏，可用随机选择的方法来改善算法的性能；

◆ 某些方面可能不正确，对特定的输入，算法的每一次运行不一定得到相同结果。出现这种不正确的可能性很小，以致可以安全地不予理睬。

## 随机算法：

是一种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选择的算法。

- 随机算法的最坏运行时间几乎总是和非随机化算法的最坏情形运行时间相同；
- 随机算法对于相同的输入，在不同的运行过程中会得到不同的输出；
- 对于相同的输入，随机算法的执行时间也可能随不同的运行过程而不同。

## 快排——最坏情况分析

元素已按递增或递减顺序排列，处于最坏的情况。split 所得到的枢点元素位置，或者是子序列的开始位置，或者是子序列的结束位置。

| 第 i 次划分 | 子序列长度 | | 所执行的元素比较次数 |
|---|---|---|---|
| i = 1 | 0 | n – 1 | n – 1 |
| i = 2 | 1 | n – 2 | n – 2 |
| i = n | n-1 | 0 | 0 |

算法quick_sort所执行的元素比较的总次数是：

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

- 对于快排，方法A用第一个元素作为枢点元素，方法B使用随机选出的元素作为枢点元素。
  - 两种情形下，最坏情况运行时间均为$\Theta(n^2)$；
  - 存在特定的输入总能够出现在A中产生不好的运行时间；
  - 如果B以相同的输入运行两次，那么它将有两个不同的运行时间，这依赖于什么样的随机数发生。
- 通过随机化算法，特定的输入不再重要，重要的是随机数，我们可以得到一个期望的运行时间。

## 随机算法的优点：

- 执行时间和空间，小于同一问题的已知最好的确定性算法；
- 实现比较简单，容易理解。

## 随机算法的随机性：

- 对于同一实例的多次执行，效果可能完全不同；
- 时间复杂度的一个随机变量；
- 解的正确性和准确性也是随机的。

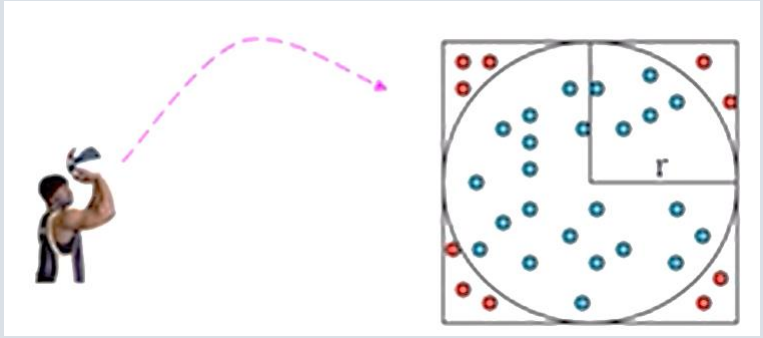
## 随机算法的类型

1）数值概率算法：

- 主要用于数值问题的求解；
- 算法的输出几乎都是近似解；
- 近似解的精度与计算时间成正比。

✓ 例：计算$\pi$值？

- 例：用随机投点法计算$\pi$值
  - 设有一个半径为$r$的圆及其外切四边形




  - 向正方形随机地投掷$n$个点，设$k$个点落入圆内；
  - 投入的点在正方形上均匀分布，因而所投掷点落入圆内的概率为 $\pi r^2 / 4r^2 = \pi/4$；
  - 当$n$足够大时，用$k/n$逼近$\pi/4$，即$k/n \approx \pi/4$，于是$\pi \approx (4k)/n$

- 例：用随机投点法计算π值

```
public double darts(int n)
{ //用随机投点法计算π值
   int k=0 ;
   for (int i=1;i<=n;i++){
       double  x=dart.fRandom();  //随机地产生四边形
       double  y=dart.fRandom();  //中的一点(x,y)
        if ((x*x+y*y)<=1)   k++;
        }
       return  4*k/(double)n;
 }
```

- 时间复杂性：$O(n)$
  - ✓ 不是输入的大小，而是随机样本的大小；
- 解的精确度时间复杂性：
  - ✓ 随着随机样本大小$n$增加而增加。

## 2）拉斯维加斯(Las Vegas)算法：

- 可能给出问题的正确解，也可能得不到问题的答案；
- 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；
- 增加对问题反复求解次数，可使求解无效的概率任意小。

## 3）蒙特卡罗(Monte Carlo)算法：

- 并不总能获得问题的正确解；
- 算法以较高的概率获得正确解；
- 主要用于求解需要准确解的问题；
- 获得精确解概率与算法执行时间成正比。

## 4）舍伍德(Sherwood)算法：

在算法中引入随机性来消除或减少算法时间复杂度在问题好坏实例之间的差别。

- 总能得到问题的一个解，且所求得的解总是正确的；
- 确定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；
- 消除最坏行为与特定实例的联系。

## 时间复杂性的衡量

1）确定性算法的时间复杂性衡量：

取其平均运行时间

2）随机算法的时间复杂性衡量：

取确定实例的期望运行时间，即反复地运行该实例的平均运行时间；

3）随机算法里讨论的是最坏情况下的期望时间，和平均情况下的期望时间。

## 随机数发生器

随机数：随机数在随机化算法设计中扮演者非常重要的角色。在现实计算机上无法产生真正的随机数，因此在随机化算法中使用的随机数都是一定程度上随机的，即伪随机数。

- **产生随机数的公式**

$$\begin{cases} d_0 = d \\ d_n = bd_{n-1} + c \quad n = 1, 2, \ldots \\ a_n = d_n / 65536 \end{cases}$$

- ✓ 产生 0~65535 的随机数序列 $a_1$，$a_2$ ,…
- ✓ $b$、$c$、$d$ 为正整数;
- ✓ $d$ 称为所产生的随机序列的种子;
- ✓ 常数 $b$、$c$，对所产生的随机序列的随机性能有很大的关系，$b$ 通常取一素数。

## 随机数发生器

1）函数 random_seed：产生随机数的种子

```
static unsigned long  seed;
void random_seed(unsigned long d)
{
    if (d==0)
        seed = time(0);          // 取系统时间作为随机数种子
    else
        seed = d;
}
```

## 2）随机数发生器

在给定种子的基础上，计算新的种子，并产生一个范围为low~high的新的随机数。

```
#define         MULTIPLIER    0x015A4E35L;
#define         INCREMENT             1;
unsigned  int  random(unsigned long low,
                              unsigned long high)
{
    seed = MULTIPLIER * seed + INCREMENT;
    return ((seed >> 16) % (high – low) + low);
}
```

# 9.2 舍伍德（Sherwood）算法

一　舍伍德算法的基本思想

二　随机快速排序算法

三　随机选择算法

# 9.2.1 舍伍德算法的基本思想

## 1. 确定性算法的平均运行时间

- $T_A(x)$：确定性算法 $A$ 对输入实例 $x$ 的运行时间
- $X_n$：规模为 $n$ 的所有输入实例全体
- 算法的平均运行时间：$\overline{T}_A(n) = \sum_{x \in X_n} T_A(x)/|X_n|$
- 存在个别实例：$x \in X_n, T_A(x) \gg \overline{T}_A(n)$
- 例：快速排序算法：
  - ✓ 当输入数据均匀分布时，运行时间是 $\Theta(n\log n)$
  - ✓ 当输入数据按递增或递减顺序排列时，算法的运行时间 $O(n^2)$

## 2. 舍伍德算法的基本思想

消除不同输入实例对算法性能的影响，使随机算法 $B$ 对规模为 $n$ 的每一个实例，都有：

$$T_B(x) = \overline{T}_A(n) + s(n)$$

## 3. 舍伍德算法的期望运行时间

$$\overline{T}_B(x) = \frac{\sum_{x \in X_n} T_B(x)}{|X_n|} = \overline{T}_A(n) + s(n)$$

- 当 $s(n)$ 与 $\overline{T}_A(n)$ 相比很小可以忽略时，舍伍德算法有很好的性能；
- 对所有输入实例而言，运行时间相对均匀。
- 时间复杂性与确定性算法的时间复杂性相当。

# 9.2.2 随机快速排序算法

## 1. 思想方法

随机选择枢点的快速排序算法

## 2. 算法描述

算法9.1 随机选择枢点元素的快速排序算法

输入：数组A，数组元素的起始位置low，终止位置high

输出：按非降顺序排序的数组A

```
2. void quicksort_random(Type A[ ], int low,
        int high)
3. {
4.     random_seed(0);// 选择系统当前时间作为随机数种子
5.     r_quicksort(A,low,high);// 递归调用随机快速排序算法
6. }
```

# 算法的期望运行时间是 $\Theta(n\log n)$

```
1. void r_quicksort(Type A[ ], int low, int high)
2. {
3.     int k;
4.     if (low<high) {
5.         k = random(low,high);// 产生low到high之间的随机数 k
6.         swap(A[low],A[k]); // 把元素A[k]交换到数组的第一个位置。
7.         k = split(A,low,high); // 按元素A[low]把数组划分为两个
8.         r_quicksort(A,low,k-1);// 排序第一个子数组
9.         r_quicksort(A,k+1,high);    // 排序第二个子数组
10.    }
11. }
```

# 9.2.3 随机选择算法

## 1. 思想方法

1）问题提出：从数组 $A$ 中选取第 $k$ 小元素

2）思想方法：随机选取枢点，把数组 $A$ 划分为两个，丢弃其中一个子数组，在另一个子数组中递归使用本算法选取该元素。

# 2. 算法描述

算法9.2 随机选择算法

输入：数组A及其第一个元素下标low，最后一个元素下标high，所选择第k小元素的序号k

输出：所选择的元素

```
2. Type select_random(Type A[ ], int low,
                      int high, int k)
3. {
4.     random_seed(0); // 产生随机数种子
5.     k = k – 1;          // 使 k 从第 low 元素开始计算
5.     return r_select(A[ ], low, high, k);       // 递归调用
6. }
```

```
1. Type r_select(Type A[ ], int low, int high, int k)
2. {
3.     int i;
4.     if (high-low <= k)
5.         return A[high];                  // 直接返回最高端元素
6.     else {
7.         i = random(low, high);       // 产生随机数 i
8.         swap(A[low], A[ i ]);        // 元素交换位置
9.         i = split(A, low, high);     // 按元素 A[low] 划分
10.        if ((i-low) == k)            // 元素 A[ i ] 就是第k小元素
11.            return  A[ i ];
12.        else if ((i-low) > k)          // 从第一个子数组寻找
13.            return r_select(A, low, i-1, k);
14.        else                                    // 从第二个子数组寻找
15.            return r_select(A, i+1, high, k-i-1);
16.    }
17. }
```

## 3. 算法分析

- 算法对 $n$ 个元素的数组所执行的元素比较的期望次数—

  —数学归纳法证明

- 元素比较的期望次数小于 $4n$

- 期望运行时间是 $\Theta(n)$

# THE END