第7章　概率算法

本章主要内容

- 7.1 随机数
- 7.2数值随机化算法
- 7.3舍伍得算法
- 7.4拉斯维加斯算法
- 7.5蒙特卡洛算法

7.1 随机数

随机数在概率算法设计中十分重要。

计算机上无法产生真正的随机数。是一定程度上随机的，是**伪随机数**

线性同余法是产生伪随机数的最常用方法。

随机序列$a_0,a_1,\ldots,a_n$满足：

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

其中b≥0，c≥0，d≤m。d称为该随机序列的种子。

一般取**m为机器大数**，要求 **gcd(m,b)=1**，因此可取b为一素数。

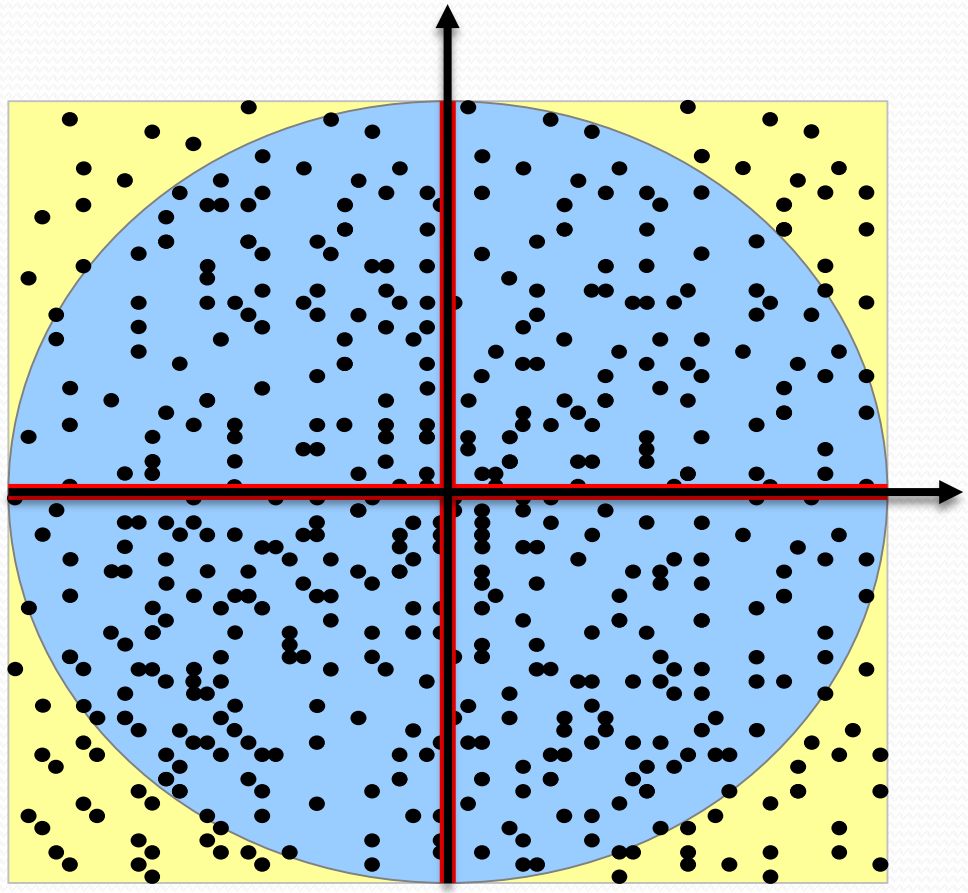
7. 2数值随机化算法

- 7.2.1 用随机投点法计算π值

设计一个半径r为1的圆外接正方形，在该正方形内产生n随机点，统计落在圆内的随机点的个数k，

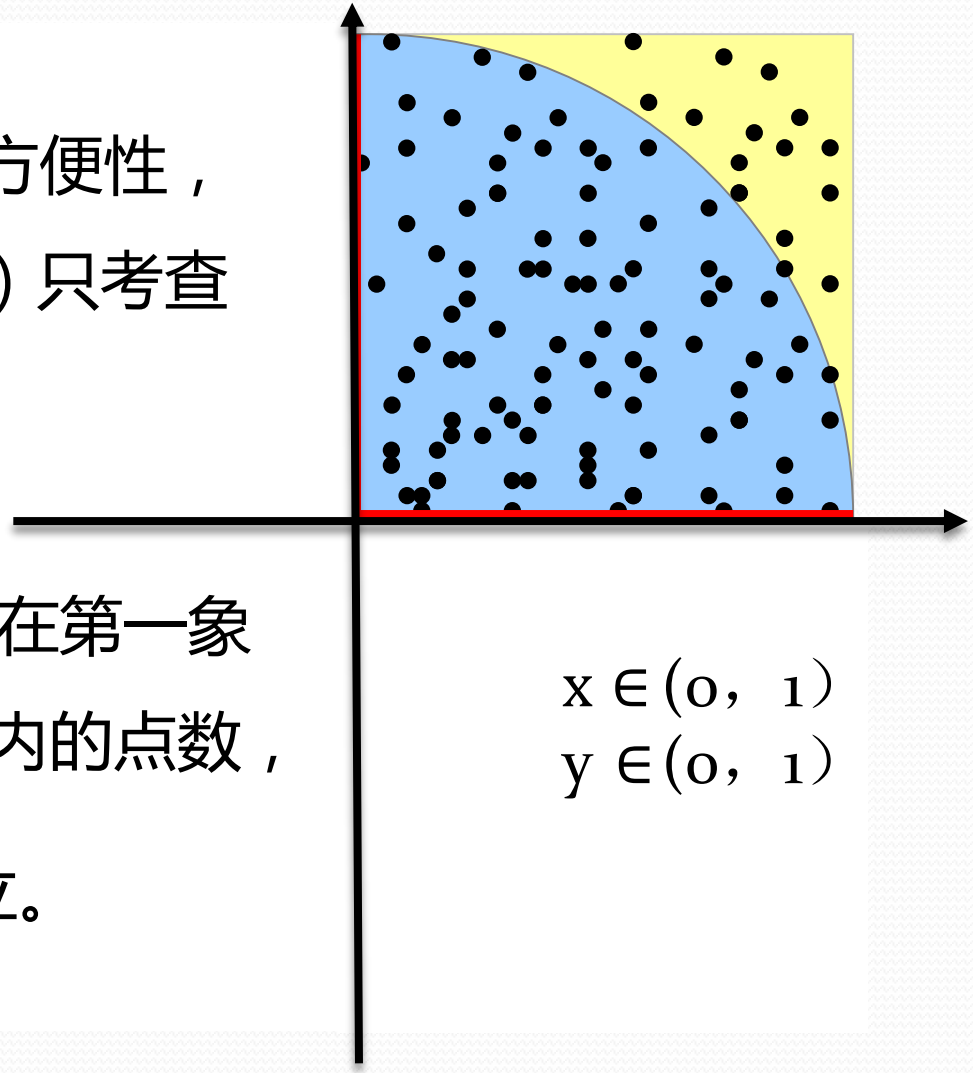
则有：$\frac{\pi r^2}{4r^2} \approx \frac{k}{n}$，

从而有：$\pi \approx \frac{4k}{n}$



7. 2数值随机化算法

考虑到计算的方便性，随机点坐标（x，y）只考查在第一象限的情况。

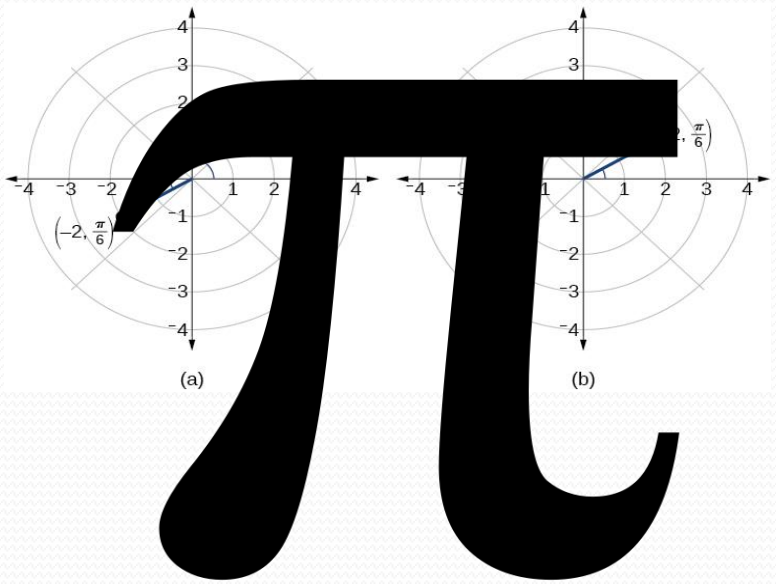


$x \in (0, 1)$
$y \in (0, 1)$

产生n个随机点在第一象限，K为落在1/4圆内的点数，公式$\pi \approx \frac{4k}{n}$仍然成立。

7.2数值随机化算法

用随机投点法计算π值算法




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

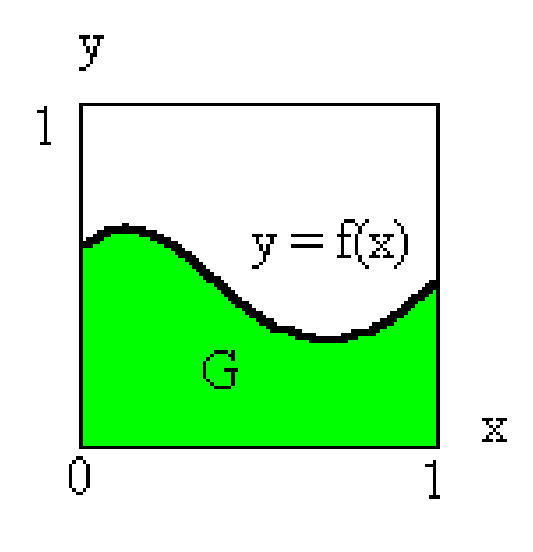
7. 2数值随机化算法

7.2.2 计算定积分

设f(x)是[0，1]上的连续函数，且0≤f(x)≤1。

$$I=\int_0^1 f(x)dx$$

设矩形面积S，$\frac{\mathrm{I}}{\mathrm{S}}\approx\frac{m}{n},S=1.\Rightarrow I=\frac{m}{n}$



随机产生n个点（x',y'），x',y'都是0~1之间的实数。判断点（x',y')落在绿色区域的条件：y'<=f(x')

7.2数值随机化算法

7.2.3 解非线性方程组

求解下面的非线性方程组

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

其中，$x_1,x_2,\ldots,x_n$是实变量，f_i是未知量$x_1,x_2,\ldots,x_n$的非线性实函数。要求确定上述方程组在指定求根范围内的一组解 $x_1^*, x_2^*, \cdots, x_n^*$

(1)首先将多目标优化转为单目标优化：$F(X)=f1^2(X)+f2^2(X)\ldots+fn^2(X)=0$

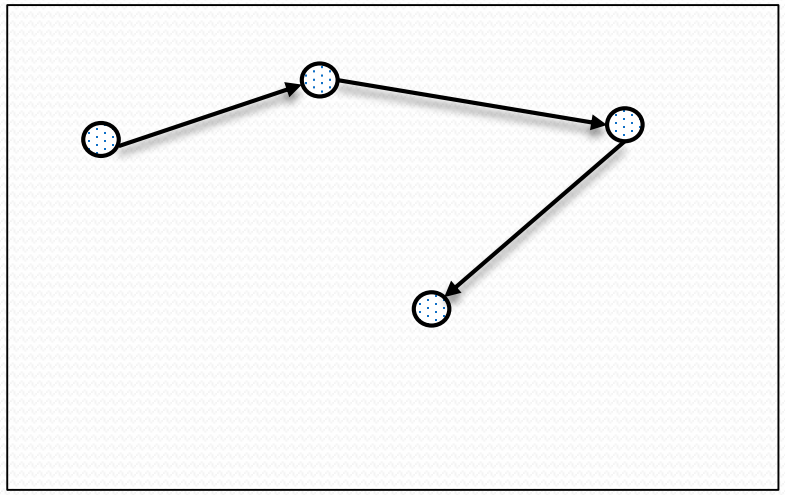
(2)此问题只能找近似解，可设定误差ε.

(3)在指定求根区域D内，随机搜素到一个点X, 当F（X）<ε时，即为所求的近似解。否则继续搜索。

7.2数值随机化算法

以二维为例说明该算法

随机产生方向向量r，依可变步长a*r得到增量ΔX，根据增量值ΔX ，从当前点移动到下一个点， 每到达一个新位置， 检查是否找到近似解



该类题目建议使用粒子群算法，或者模拟退火算法， 这两种方法求解该类问题是更有效的，也是数值随机化方法。

7.3舍伍德(Sherwood)算法

设A是一个确定性算法，当它的输入实例为x时，所需的计算时间复杂性记为$t_{A(x)}$。设X_n是算法A的输入规模为n的实例的全体。算法A所需的平均时间复杂性为：

$$\bar{t}_A(n) = \sum_{x \in X_n} t_A(x) / | X_n |$$

这显然不能排除存在x∈Xn使得 $t_A(x) >> \bar{t}_A(n)$ 的可能性。

舍伍德思想：获得一个概率算法B，使得对问题的输入规模为n的每一个实例均有 $t_B(x) = \bar{t}_A(n) + s(n)$。 当s(n)与平均时间复杂性相比可忽略时，可获得很好的平均性能。

7.3舍伍德(Sherwood)算法

回忆学过的舍伍德算法：

（1）线性时间选择算法

（2）快速排序算法

当在数据基本有序情况下，partition（ ）算法的两个分组严重不平衡，导致这两个算法的时间复杂性都是$O(n^2)$。

改造的方法将partition（ ）算法改为randomizedpartition（ ）算法

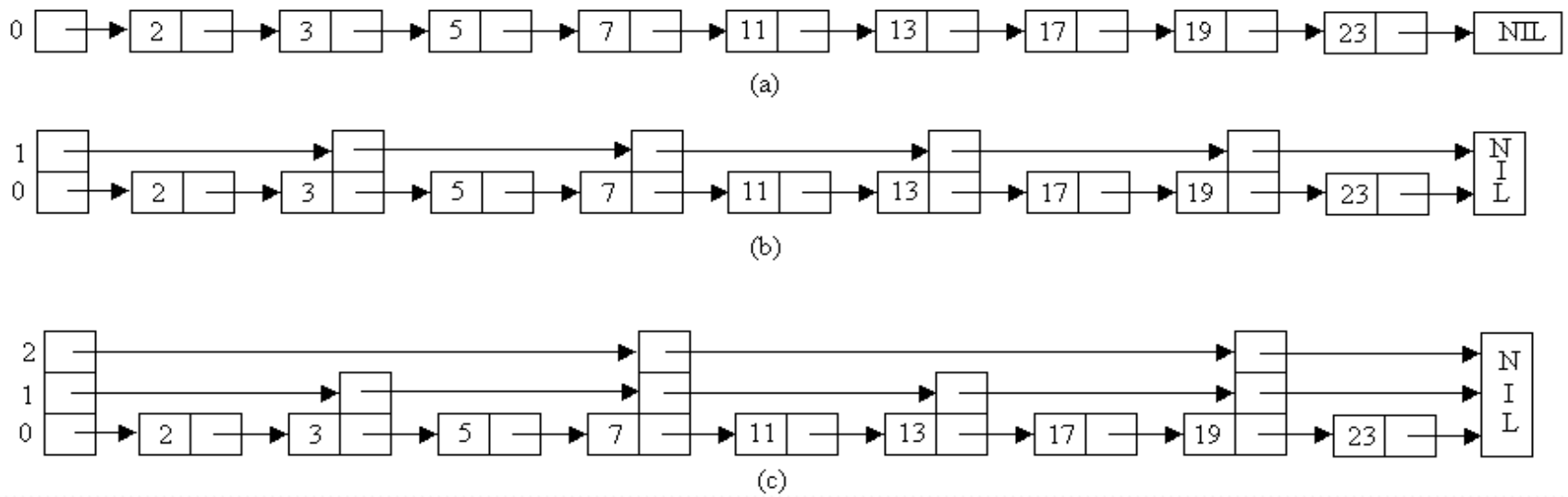
当无法直接改造成舍伍德型算法时，可借助于随机预处理技术（洗牌算法）

```
void Shuffle(Type a[], int n)
{// 随机洗牌算法
  static RandomNumber rnd;
  for (int i=0;i<n;i++) {
      int j=rnd.Random(n-i)+i;
    Swap(a[i], a[j]);   }
}
```

7.3舍伍德(Sherwood)算法

跳跃表

- 如果用有序链表来表示一个含有n个元素的有序集S，则在最坏情况下，搜索S中一个元素需要O(n)计算时间。
- 舍伍德型算法的设计思想还可用于设计**高效的数据结构**。

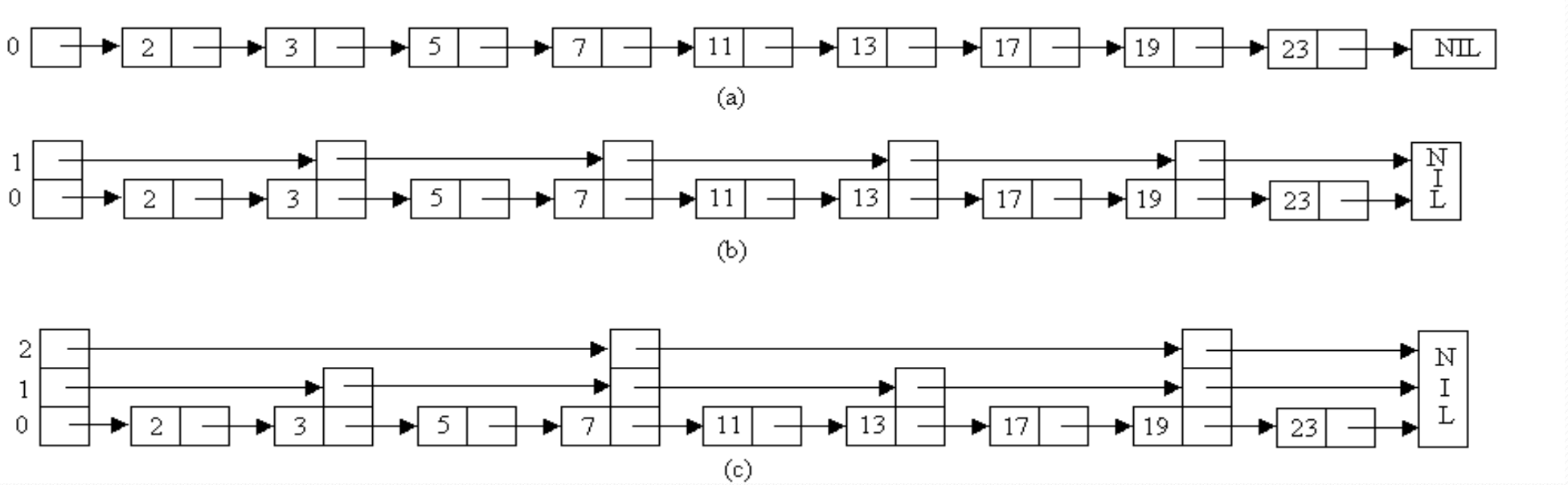


- **跳跃表**：增加了向前附加指针的有序链表。

7.3舍伍德(Sherwood)算法

跳跃表

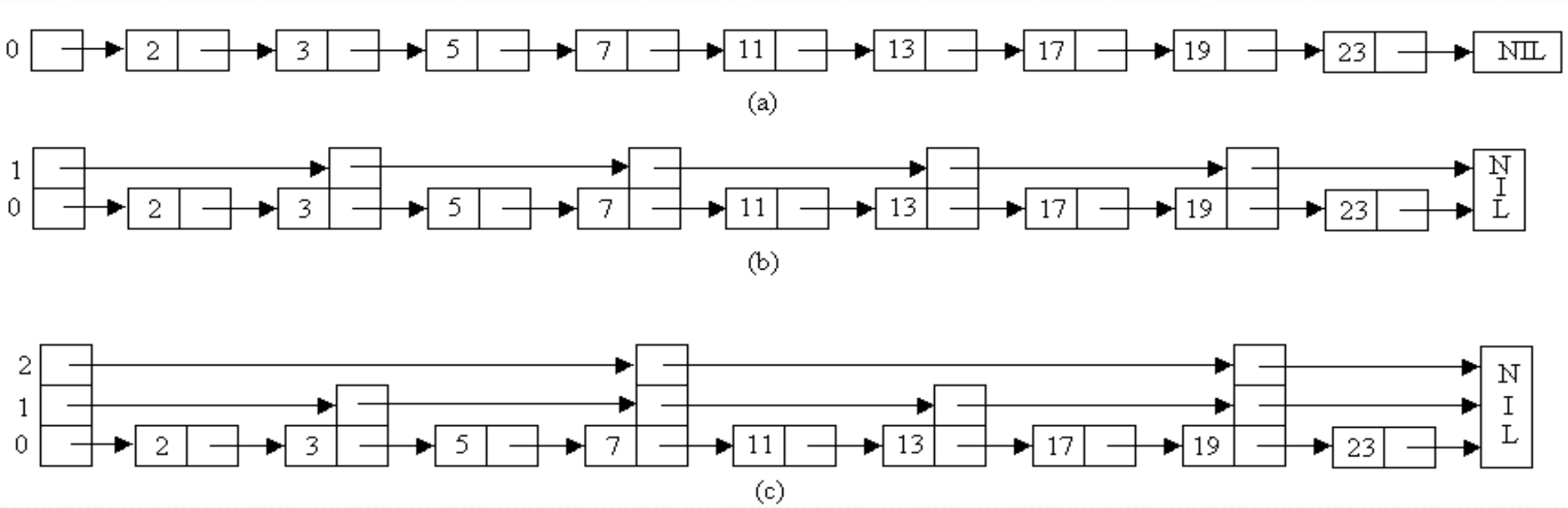
- 跳跃表可在O(logn)平均时间内支持关于有序集的搜索、插入和删除等运算



7.3舍伍德(Sherwood)算法

跳跃表

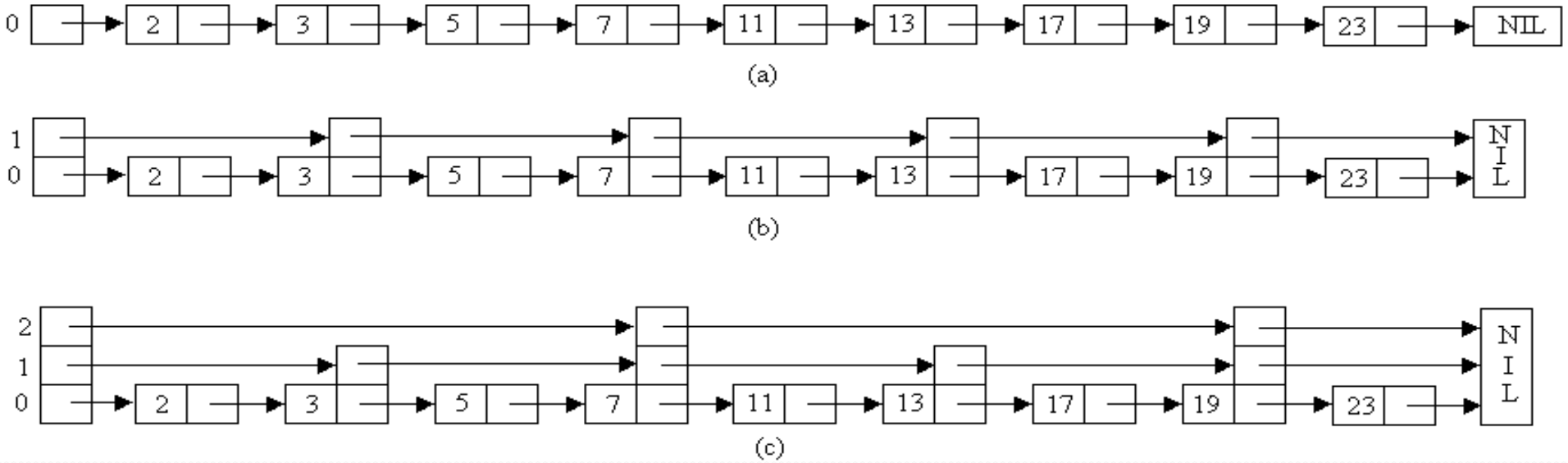
含有n个元素的有序链表，改造成一个完全跳跃表，使得每一个k级结点含有k+1个指针，分别跳过2^k-1，$2^{k-1}-1$，…，2^0-1个中间结点。第i个k级结点安排在跳跃表的位置$i*2^k$处，$i\geq 0$。最高级的结点是$\lceil \log n \rceil$级结点。



7.3舍伍德(Sherwood)算法

跳跃表

同完全二叉搜索树类似。它虽然可以有效地支持成员搜索运算，但不适应动态变化的情况。元素的插入和删除运算会破坏跳跃表原有的平衡状态，影响之后的搜索的效率。

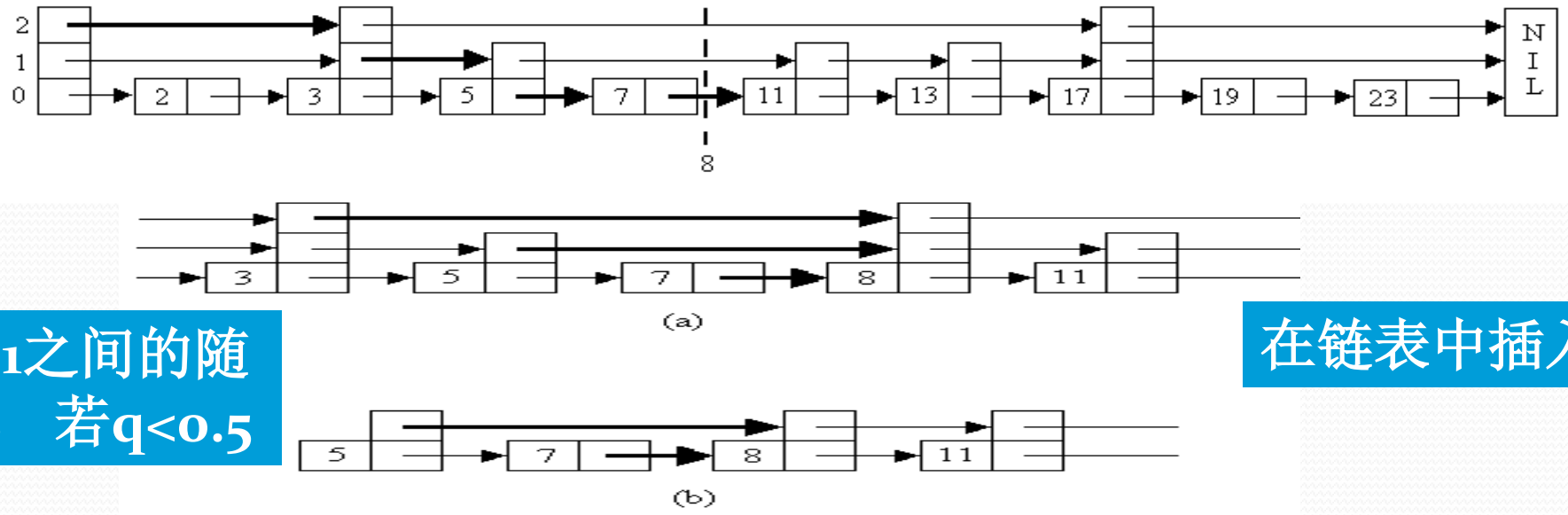


7.3舍伍德(Sherwood)算法

在动态变化中维持跳跃表中附加指针的平衡性：随机化策略

在一个完全跳跃表中，**高一级指针结点**个数/**低一级指针结点**个数=$\frac{1}{2}$

因此，在插入一个元素时应以概率$1/2^k$引入一个k级指针结点。



产生0-1之间的随机数q， 若q<0.5

在链表中插入8

7.3舍伍德(Sherwood)算法

跳跃表

为维持跳跃表的平衡性，确定一实数0<p<1(通常取p=0.5)，

目的：含有**i+1级指针**的结点数/含有**i级指针**的结点数≈p。

在插入一个新结点时

（1）先将其结点级别k初始化为0，

（2）然后用随机数生成器反复地产生一个[0，1]间的随机实数q。

（3）如果q<p，则使新结点级别k增加1，直至q≥p或k=$\log_{1/p} n$

用 $\log_{1/p} n$ 作为新结点级别的上界。其中n是当前跳跃表中结点个数。

7. 4拉斯维加斯（ Las Vegas ）算法

Las Vegas 算法

是一个一定能找到正确解的随机算法，

其运行次数随机但有界。

显著特征：

其随机性决策有可能导致算法某次执行找不到所需的解。

7. 4拉斯维加斯（ Las Vegas ）算法

```
void obstinate(Object x, Object y)
  {// 反复调用拉斯维加斯算法LV(x,y)，直到找到问题x的一个解y
    bool success= false;
    while (!success) success=lv(x,y);
  }
```

x是一个具体实例.p(x)>0是获得问题的一个解的概率。设t(x)是找到一个解所需平均时间， s(x)和e(x)是求解成功或失败所需的平均时间，则有：

$$t(x) = p(x)s(x) + (1 - p(x))(e(x) + t(x))$$

解此方程可得：

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

7. 4拉斯维加斯（ Las Vegas ）算法

7.4.1 n后问题(拉斯维加斯算法)

随机放置策略：

在棋盘上各行中随机地放置皇后. 判断是否满足约束条件，如满足，找到解。

随机放置策略与回溯法相结合，可能会获得更好的效果。随着前面放置的皇后越来越多，后面选择皇后位置冲突的可能性也越大

7.4拉斯维加斯（ Las Vegas ）算法

与回溯法相结合

先在棋盘的前若干行中随机地放置皇后，判定是否满足约束条件，如满足则在后继行中用回溯法继续放置，直至找到一个解或宣告失败。

随机放置皇后越多，后继回溯搜索所需时间就越少，但失败概率就越大。

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

N=12皇后的问题

7. 4拉斯维加斯（ Las Vegas ）算法

7.4.2整数因子分解

设n>1是一个整数。关于整数n的**因子分解**问题是: $n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}$

$p_1<p_2<\ldots<p_k$是k个素数，$m_1,m_2,\ldots,m_k$ 是k个正整数。

```
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;
}
```

确定性算法求解：

时间复杂性O($\sqrt{n}$)

7.4拉斯维加斯（Las Vegas）算法

Pollard算法思想

（1）选取0～n-1范围内的随机数x1，

（2）由 $x_i = (x_{i-1}^2 - 1) \bmod n$ 产生无穷序列 $x_1, x_2, \cdots, x_k, \cdots$

（3）对于$i=2^k$，以及$2^k<j\leq 2^{k+1}$， 算法计算出x_j-x_i与n的最大公因子

$d=gcd(x_j-x_i，n)$。

（4）如果d是n的非平凡因子，则实现对n的一次分割，算法因子d。

Pollard算法

```
void Pollard(int n)
{  RandomNumber rnd;
   int i=1，  y;
   int  x=rnd.Random(n);
   int  y=x;
   int k=2;
   while (true) {
          i++;
          x=(x*x-1)%n;
          int d=gcd(x-y,n);
          if ((d>1) && (d<n))
              {cout<<d<<endl; break;}
          if (i==k) { y=x;  k*=2;}
    }
}
```

2^0	2^1		2^2				2^3		
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
y	x/y	x	x/y	x	x	x	x/y	x	x

while循环执行约 $\sqrt{p}$ 次后，算法会输出n的一个因子p。

由于n的最小素因子p≤ $\sqrt{n}$ 。故可在$O(n^{1/4})$时间内找到n的一个素因子。

7. 5蒙特卡罗(Monte Carlo)算法

Monte Carlo 算法

一些问题，不论采用确定性算法或概率算法都无法保证每次都能得到正确的解答。

蒙特卡罗算法则在一般情况下可以保证对问题的所有实例都以高概率给出正确解，但是通常无法判定一个具体解是否正确。

7.5蒙特卡罗(Monte Carlo)算法

- 设p是一个实数，且1/2<p<1。如果一个蒙特卡罗算法对于问题的任一实例得到正确解的概率不小于 p，则称该蒙特卡罗算法是p正确的，且称p-1/2是该算法的优势。
- 如果对于同一实例，蒙特卡罗算法不会给出2个不同的正确解答，则称该蒙特卡罗算法是一致的。
- 对于一个解所给问题的蒙特卡罗算法MC(x)，如果存在问题实例的子集X使得：

  (1)当$x \notin X$时，MC(x)返回的解总是正确的；

  (2)当$x \in X$时，正确解是y_0，但MC(x)返回解未必是y_0。称MC(x)是偏y_0的算法。

7. 5蒙特卡罗(Monte Carlo)算法

重复调用一个一致的、p正确的、偏y0蒙特卡罗算法k次，可得到一个1-(1-p)^{k}正确的蒙特卡罗算法，且所得算法仍是一个一致的偏y0蒙特卡罗算法。

换句话说：如果蒙特卡洛对同一个实例，连续运行k次， 答案都是一样的， 则这个答案正确的可能性是 1-(1-p)^k。

如k足够大，则该答案的正确性就足够高

7. 5蒙特卡罗（Monte Carlo）算法

7.5.1主元素问题

设T[1:n]是一个含有n个元素的数组。当|{i|T[i]=x}|>n/2时，称元素x是数组T的主元素。

1	2	3	4	3	4	1	2

1	3	4	3	5	3	3	3

确定性的算法：

- 排序、一遍扫描
- 分治法也可以

7. 5蒙特卡罗(Monte Carlo)算法

```
template<class Type>
bool Majority(Type *T, int n)
{// 判定主元素的蒙特卡罗算法
    int i=rnd.Random(n)+1;
    Type x=T[i];    // 随机选择数组元素
    int k=0;
    for (int j=1;j<=n;j++)
      if (T[j]==x) k++;
    return (k>n/2);  // k>n/2 时T有主元素
}
```

```
template<class Type>
bool MajorityMC(Type *T, int n,
double e)
{// e是出错率，k由e控制。
    int k=ceil(log(1/e)/log(2));
    for (int i=1;i<=k;i++)
      if (Majority(T,n)) return true;
    return false;
}
```

7. 5蒙特卡罗(Monte Carlo)算法

对于任何给定的$\varepsilon>0$，算法**majorityMC**重复调用$k=\lceil \log(1/\varepsilon) \rceil$次算法**majority**。

它是一个偏真蒙特卡罗算法，且其错误概率小于ε。

算法**majorityMC**所需的计算时间显然是$O(n\log(1/\varepsilon))$。

7. 5蒙特卡罗(Monte Carlo)算法

7.5.2素数测试

Wilson定理：对于给定的正整数n，判定n是一个素数的充要条件是$(n-1)!\equiv -1(\mathrm{mod}\ n)$。

费尔马小定理：如果n是一个素数，且$0<a<n$，则$a^{n-1}=1\ (\mathrm{mod}\ n)$。

二次探测定理：如果n是一个素数，且$0<x<n$，则方程$x^2\equiv 1(\mathrm{mod}\ n)$的解为x=1，n-1。

因为费尔马小定理和二次探测定理都是充分条件，因此在不是素数的数可能被判定为素数。一致的偏假的蒙特卡洛算法。

7. 5蒙特卡罗(Monte Carlo)算法

0<a<n, p=n-1, n, a^p , false

```
void power( unsigned int a, unsigned int p, unsigned int n,  unsigned int &result,  bool &composite)
{// 计算mod n，并实施对n的二次探测
   unsigned int x;
   if (p==0) result=1;       //result is a^p
   else {
     power(a,p/2,n,x,composite);  // 递归计算
     result=(x*x)%n;          // 二次探测
     if ((result==1)&&(x!=1)&&(x!=n-1))
       composite=true;
     if ((p%2)==1)      // p是奇数
       result=(result*a)%n;
     }
}
```

```
bool Prime(unsigned int n)
{// 素数测试的蒙特卡罗算法
   RandomNumber rnd;
   unsigned int a, result;
   bool composite=false;
   a=rnd.Random(n-3)+2;
   power(a,n-1,n,result,composite);
   if (composite||(result!=1))
          return false;
   else return true;
}
```

算法**prime**是一个偏假3/4正确的蒙特卡罗算法。通过k次重复调用**prime**，错误概率不超过$(1/4)^k$

小结

- **本章介绍利用随机数的随机性来解决问题**。
- **这些算法大多是非确定性算法**。在某一时刻，下一步动作有多种选择。
- **这些算法没有具体框架，分类是基于其算法的思想**。

数值随机化算法：利用随机数求问题答案

舍伍德算法：将算法加入概率算法，使算法不再有：对某些实例，算法的复杂性必然很高的规律。

拉斯维加斯算法：引入随机化策略，运行一次算法可能找不到答案， 但运行多次一定能找到答案。

蒙特卡洛算法：引入随机化策略，运行一次算法总能给出答案， 但未必正确，可以肯定的是其正确性超过**50%,** 若运行多次，答案不变， 则该答案正确的可能性非常高。 （蒙特卡洛算法可进行故障检测）