第 3 章 模 拟 法

模拟法(simulation method)通常基于问题描述,或完成简单的建模,或模拟过程的实现,是最简单的算法设计技术。有些问题很难通过数学推导找到规律,一般采用模拟法直接模拟问题中事物的变化过程,从而完成相应的任务。

3.1 概 述


课件 3-1

3.1.1 模拟法的设计思想

用模拟法求解问题的基本思想是对问题进行抽象,将现实世界的问题映射成计算机能够识别的符号表示,将事物之间的关系映射成运算或逻辑控制。模拟法求解的问题通常是对某一类事件进行描述,然后经过简单计算给出符合要求的结果。

模拟法是算法设计的基本功,没有复杂的公式和技巧,只需读懂问题、明确要求,照着逻辑整理步骤,基本都可以完成。需要注意的是,有些问题的背景错综复杂,如果没有理顺逻辑就可能步入歧途。

3.1.2 一个简单的例子:鸡兔同笼问题

【问题】 笼子里有若干只鸡和兔子,鸡有两只脚,兔子有四只脚,没有例外情况。已知笼子里脚的数量,问笼子里至多有多少只动物?至少有多少只动物?

【想法】 对于同样数目的动物,鸡脚的总数肯定比兔子脚的总数要少,因此在计算笼子里至多有多少只动物时,应该把脚都算作鸡脚,在计算笼子里至少有多少只动物时,应该尽可能把脚都算作兔子脚。

【算法】 设函数 Feets 实现鸡兔同笼问题,算法如下。

算法:鸡兔同笼问题 Feets
输入:脚的数量 n

输出：至多的动物数 maxNum，至少的动物数 minNum
1. 如果 n 是奇数，则没有满足要求的解，maxNum=0，minNum=0；
2. 如果 n 是偶数且能被 4 整除，则 maxNum=$n/2$，minNum=$n/4$；
3. 如果 n 是偶数但不能被 4 整除，则 maxNum=$n/2$，minNum=$(n-2)/4+1$；
4. 输出 maxNum 和 minNum；

【算法分析】 算法 Feets 只是进行了简单的判断和赋值，时间复杂度是 $O(1)$。

【算法实现】 设形参 maxNum 和 minNum 以传引用方式接收求得的结果，程序如下。

源代码 3-1

```
void Feets(int n, int &maxNum, int &minNum)
{
   if (n % 2 != 0) {maxNum = 0; minNum = 0;}
   else if (n % 4 == 0) {maxNum = n/2; minNum = n/4;}
   else {maxNum = n/2; minNum = (n-2)/4 + 1;}
}
```

课件 3-2

3.2 数学问题中的模拟法

3.2.1 约瑟夫环问题

【问题】 约瑟夫环问题(Josephus circle problem)由古罗马史学家约瑟夫提出，他参加并记录了公元 66—70 年犹太人反抗罗马的起义。约瑟夫作为一个将军，守住了裘达伯特城达 47 天之久。在城市沦陷后，他和 40 名视死如归的将士在一个洞穴中避难，这些反抗者表决说“要投降毋宁死”。于是，约瑟夫建议每个人轮流杀死他旁边的人，而这个顺序是由抽签决定的。约瑟夫有预谋地抓到了最后一签，并且，作为洞穴中的两个幸存者之一，他说服了同伴一起投降了罗马。

【想法】 将参与抽签的人从 1 至 n 进行编号并构成一个环，从而将约瑟夫环问题抽象为如图 3-1 所示数据模型。假设密码是 m，从第 1 个人开始报数，报到 m 时停止报数，报 m 的人出环；再从他的下一个人起重新报数，报到 m 时停止报数，报 m 的人出环。如此下去，直至所有人全部出环。当任意给定 n 和 m 后，求 n 个人出环的次序。

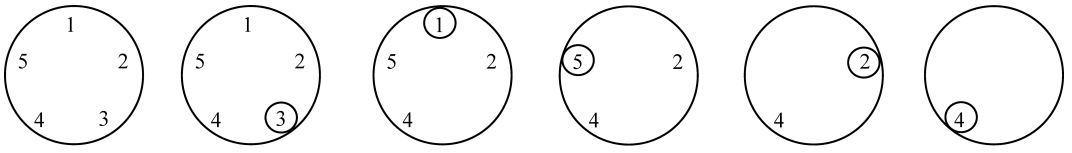

图 3-1 约瑟夫环问题的数据模型($n=5$，$m=3$ 时的出圈次序：3,1,5,2,4)

【算法】 设函数 Joseph 求解约瑟夫环问题，用数组 r[n]存储 n 个人是否出列，下标

表示人的编号，从 1 开始数到密码 m 则将其出列。如果编号 i 的人出列则将数组 r[i]置为 1，用求模运算%实现下标在数组内循环增 1。算法如下。

算法：约瑟夫环问题 Joseph
输入：参与游戏的人数 n，密码 m
输出：最后一个出列的编号
1. 初始化数组 r[n]={0}；
2. 计数器 count=0；下标 i=−1；出列人数 num=0；
3. 重复下述操作直到数组 r[n]仅剩一个人：
　3.1 当 count < m 时重复下述操作：
　　3.1.1 i=(i+1) % n；
　　3.1.2 如果 r[i]未出列，则计数器 count++；
　3.2 令 r[i]=1；num++；
4. 查找并返回仅剩的编号；

【算法分析】　步骤 3 在数组 r[n]中反复查找待出列编号，外循环执行 $n-1$ 次，内循环执行 m 次，时间复杂度为 $O(n\times m)$。

【算法实现】　设函数 Joseph 求解约瑟夫环问题，程序如下。


源代码 3-2

```
int Joseph(int r[ ], int n, int m)
{
  int count, i = -1, num = 0;
  while (num < n - 1)
  {
    count = 0;
    while (count < m)                          //查找报到 m 的人
    {
      i = (i+1) % n;
      if (r[i] != 1) count++;
    }
    r[i] = 1; num++;                           //标记出列的人
  }
  for (i = 0; i < n; i++)
    if (r[i] == 0) return i+1;                 //返回编号
}
```

3.2.2　埃拉托色尼筛法

【问题】　埃拉托色尼筛法(the sieve of Eratosthenes)简称埃氏筛法，是古希腊数学家埃拉托色尼提出的算法，用于求一定区间内的所有素数。算法的基本思想是，从区间[1，n]内的所有数中去掉所有合数，剩下的就是所有素数。判断合数的方法是从 2 开始依次过筛，如果是 2 的倍数则该数不是素数，进行标记处理，直至将 $n/2$ 过筛，将所有合数

打上标记。

【想法】 假设有一个筛子存放整数1～n,依次将2,3,5,…的倍数筛去(标记),最后没有打上标记的数都是素数。埃拉托色尼筛法的计算过程如图3-2所示。

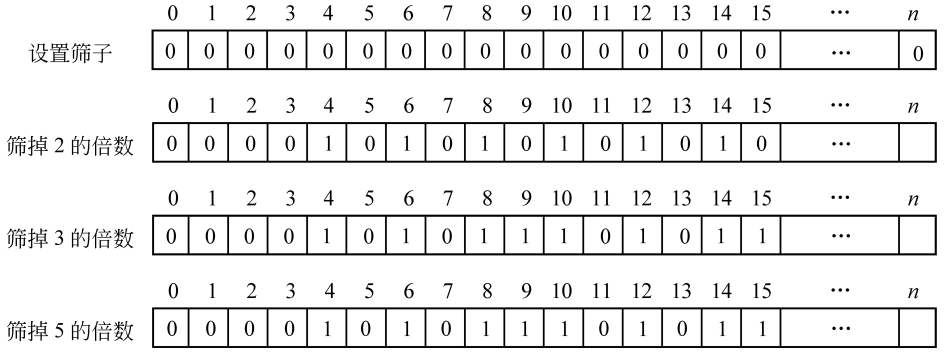

图3-2 埃拉托色尼筛法的计算过程

【算法】 设数组A[n]表示筛子,元素值全部初始化为0,依次将下标是2,3,5,…倍数的元素值置1进行标记处理,最后所有元素值为0对应的下标都是素数,算法如下。

```
算法:埃拉托色尼筛 EratoSieve
输入:待确定素数的范围 n,数组 A[n]
输出:区间[1, n]的所有素数
1. 循环变量 i 从 2 至 n/2 重复执行下述操作:
   1.1 如果 A[i]不等于 0,说明整数 i 不是素数,转步骤 1.3 取下一个素数;
   1.2 将所有下标是 i 的倍数的元素值置为 1;
   1.3 i++;
2. 输出数组 A[n]中所有元素值为 0 对应的下标;
```

【算法分析】 埃拉托色尼筛法实际上是一种空间换时间的算法优化,对于判断单个数的素数性质来说,相对于朴素的算法没有优化,但是对于求解某一区间的素数问题,埃拉托色尼筛法可以很快打印一份素数表,时间复杂度只有$O(n\log\log n)$。

【算法实现】 设函数EratoSieve实现埃拉托色尼筛法,程序如下。

源代码3-3

```
void EratoSieve(int A[ ], int n)
{
  int i, j;
  for (i = 2; i <= n/2; i++)
  {
    if (A[i] != 0) continue;
    else
    {
      for (j = 2; i * j <= n; j++)
```

```
            A[i * j] = 1;
        }
    }
}
```

3.3　排序问题中的模拟法


课件 3-3

3.3.1　计数排序

【问题】　假设待排序记录均为整数且取自区间[0，k]，计数排序[①](count sort)的基本思想是对每一个记录 x，确定小于 x 的记录个数，然后直接将 x 放在应该的位置。例如，小于 x 的记录个数是 10，则 x 就位于第 11 个位置。

【想法】　对于待排序序列 A[n]={2，1，5，2，4，3，0，5，3，2}，k=5，首先统计值为 i($0\leqslant i\leqslant k$)的记录个数存储在 num[i]中，则 num[k]={1，1，3，2，1，2}，再统计小于等于 i($1\leqslant i\leqslant k$)的记录个数存储在 num[i]中，则 num[k]={1，2，5，7，8，10}，最后反向读取数组 A[n]填到数组 B 中，例如读取 A[9]的值是 2，则将 num[2]减 1，然后将 2 填到 B[4]中，如图 3-3 所示。注意，统计小于 i($1\leqslant i\leqslant k$)的记录个数不能就地进行(利用数组 num)，需要再设一个数组存放小于 i 的记录个数，就可以正向读取数组 A[n]。

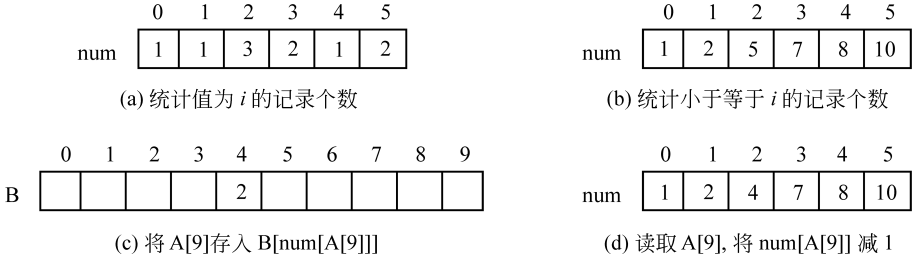

图 3-3　计数排序过程

【算法】　设函数 CountSort 实现计数排序，数组 num[k+1]存储每个记录出现的次数以及小于等于值为 i 的记录个数，算法如下。

算法：计数排序 CountSort
输入：待排序记录序列 A[n]，记录的取值区间 k
输出：排序数组 B[n]
1. 统计值为 i 的记录个数存入 num[i]；
2. 统计小于等于 i 的记录个数存入 num[i]；
3. 反向填充目标数组，将 A[i]放在 B[--num[A[i]]]中；
4. 输出数组 B[n]；

① 计数排序算法在 1954 年由 Harold H. Seward 提出，可以在线性时间对取值范围为某一区间的记录序列进行排序。

【算法分析】 计数排序是一种以空间换时间的排序算法,并且只适用于对一定范围内的整数进行排序,时间复杂度为 $O(n+k)$,其中 k 为序列中整数的范围。

【算法实现】 计数排序的关键在于确定待排序记录 A[i]在目标数组 B[n]中的位置,由于数组元素 num[i]存储的是 A[n]中小于等于 i 的记录个数,所以填充数组 B[n]时要反向读取 A[n]。程序如下。

源代码 3-4

```
void CountSort(int A[ ], int n, int k, int B[ ])
{
  int i, num[k+1] = {0};
  for(i = 0; i < n; i++)
    num[A[i]]++;
  for (i = 1; i <= k; i++)
    num[i] = num[i] + num[i-1];
  for (i = n-1; i >= 0; i--)
    B[--num[A[i]]] = A[i];
}
```

3.3.2 颜色排序

【问题】 现有 Red、Green 和 Blue 三种不同颜色(色彩中不能再分解的三种颜色,称为三原色)的小球,乱序排列在一起,请按照 Red、Green 和 Blue 顺序重新排列这些小球,使得相同颜色的小球排在一起。

【想法】 设数组 a[n]存储 Red、Green 和 Blue 三种元素,设置三个参数 i、j、k,其中 i 之前的元素(不包括 a[i])全部为 Red;k 之后的元素(不包括 a[k])全部为 Blue;i 和 j 之间的元素(不包括 a[j])全部为 Green;j 指向当前正在处理的元素。首先将 i 初始化为 0,k 初始化为 $n-1$,j 初始化为 0。然后 j 从前向后扫描,在扫描过程中根据 a[j]的颜色,将其交换到序列的前面或后面,当 j 等于 k 时,算法结束。颜色排序的求解思想如图 3-4 所示。

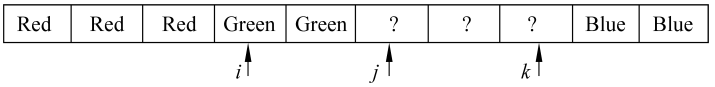

图 3-4 颜色排序的求解思想

注意,当 j 扫描到 Red 时,将 a[i]和 a[j]交换,只有当前面全部是 Red 时,交换到位置 j 的元素是 Red,否则交换到位置 j 的元素一定是 Green,因此交换后 j 应该加 1;当 j 扫描到 Blue 时,将 a[k]和 a[j]交换,Red、Green 和 Blue 均有可能交换到位置 j,则 a[j]需要再次判断,因此交换后不能改变 j 的值。

【算法】 设数组 a[n]有 Red、Green 和 Blue 三种元素,函数 ColorSort 实现颜色重排问题,算法如下。

算法：颜色排序 ColorSort
输入：待排序记录序列 a[n]
输出：排好序的数组 a[n]
1. 初始化 i=0；k=n－1；j=0；
2. 当 j≤k 时，依次考查元素 a[j]，有以下三种情况：
（1）如果 a[j]是 Red，则交换 a[i]和 a[j]；i＋＋；j＋＋；
（2）如果 a[j]是 Green，则 j＋＋；
（3）如果 a[j]是 Blue，则交换 a[k]和 a[j]；k－－；

【算法分析】　由于下标 j 和 k 整体将数组扫描一遍，因此时间复杂度为 $O(n)$。

【算法实现】　由于数组 a[n]只有三种元素，假设 Red、Green 和 Blue 三种颜色分别用 1、2、3 来代替，程序如下。


源代码 3-5

```
void ColorSort(int a[ ], int n)
{
  int i = 0, k = n - 1, j = 0, temp;
  while (j <= k)
    switch (a[j])                                    //考查当前元素
    {
      case 1: temp = a[i]; a[i] = a[j]; a[j] = temp; i++; j++; break;
      case 2: j++; break;
      case 3: temp = a[j]; a[j] = a[k]; a[k] = temp; k--; break;
    }
}
```

3.4　拓展与演练


课件 3-4

3.4.1　装箱问题

【问题】　有一个工厂制造的产品形状都是长方体，一共有 6 种型号，每种型号长方体的长和宽分别是 1×1、2×2、3×3、4×4、5×5、6×6，高都是 h。这些产品使用统一规格的箱子进行包装，箱子的长、宽和高分别是 6、6 和 h。对于每个订单工厂希望用最少的箱子进行包装。每个订单包括用空格分开的 6 个整数，分别代表这 6 种型号的产品数量。输出是包装需要箱子的个数。

【想法】　这个问题很难建立一个数学模型，只能模拟包装过程，分析装入 6 种产品后箱子的剩余空间。装箱情况分析如表 3-1 所示。

表 3-1 6种产品占用箱子与剩余空间的装箱情况分析

1个箱子容纳产品		剩余空间的装载情况		
产品	个数	2×2	1×1	解 释
6×6	1	0	0	无剩余空间
5×5	1	0	11	11个长宽为1的产品
4×4	1	5	0	5个长宽为2的产品
3×3	1	5	7	5个长宽为2的产品和7个长宽为1的产品
3×3	2	3	6	3个长宽为2的产品和6个长宽为1的产品
3×3	3	1	5	1个长宽为2的产品和5个长宽为1的产品
3×3	4	0	0	无剩余空间
2×2	9	0	0	无剩余空间
1×1	36	0	0	无剩余空间

【算法】 设 k_1、k_2、k_3、k_4、k_5 和 k_6 分别表示6种型号的产品数量，x 和 y 分别表示长宽为2和1的空位数量，n 表示需要的箱子个数，算法如下。

算法：装箱问题 Packing
输入：6种型号的产品数量 k_1、k_2、k_3、k_4、k_5、k_6
输出：箱子个数 n
1. n=装入长宽为3×3、4×4、5×5、6×6所需箱子数；
2. $x=n$ 个箱子剩余2×2的空位数；
3. 如果 $k_2>x$，则 $n=n+(k_2-x$ 个产品需要的箱子数)；
4. $y=n$ 个箱子剩余1×1的空位数；
5. 如果 $k_1>y$，则 $n=n+(k_1-y$ 个产品需要的箱子数)；
6. 输出箱子个数 n；

【算法分析】 算法 Packing 所有操作步骤都是简单的计算，时间复杂度为 $O(1)$。

【算法实现】 设变量 k1、k2、k3、k4、k5 和 k6 分别表示6种型号的产品数量，变量 x 和 y 分别表示长宽为2和1的空位数量，变量 n 表示需要的箱子个数。设数组 p2[4]存储装入3×3的产品个数分别是4、1、2、3时箱子剩余2×2的空位数。注意，程序中所有的整除都应该保证向上取整。程序如下。


源代码 3-6

```
int Packing(int k1, int k2, int k3, int k4, int k5, int k6)
{
  int n, x, y;
  int p2[4] = {0, 5, 3, 1};
  n = (k3 + 3)/4 + k4 + k5 + k6;
  x = 5 * k4 + p2[k3 % 4];
  if (k2 > x) n += (k2 - x + 8) / 9;
```

```
    y = 36 * n - 36 * k6 - 25 * k5 - 16 * k4 - 9 * k3 - 4 * k2;
    if (k1 > y) n += (k1 - y + 35) / 36;
    return n;
}
```

3.4.2　数字回转方阵

【问题】　n 阶数字回转方阵是将数字 1 置于方阵的左上角，然后从 1 开始递增，将 n^2 个整数填写到 n 阶方阵中，偶数层从第 1 行开始，先向下再折转向左，奇数层从第 1 列开始先向右再折转向上，呈首尾相接，图 3-5 所示为一个 5 阶数字回转方阵。

1	2	9	10	25
4	3	8	11	24
5	6	7	12	23
16	15	14	13	22
17	18	19	20	21

图 3-5　5 阶数字回转方阵

【想法】　根据问题描述的填数规则，找到下标变化规律，用模拟法直接求解。对于方阵的偶数行和列，填数的起始位置是$(1, i)$，然后列号不变行号加 1，至位置(i, i)时折转，行号不变列号减 1，直至位置$(i, 1)$；对于方阵的奇数行和列，填数的起始位置是$(j, 1)$，然后行号不变列号加 1，至位置(j, j)时折转，列号不变行号减 1，直至位置$(1, j)$。

【算法】　设函数 Full 实现填写数字回转方阵，注意数组下标从 0 开始，算法如下。

```
算法：数字回转方阵 Full
输入：方阵的阶数 n
输出：数字回转方阵 z[n][n]
1. z[0][0]=1,number=2;
2. for (i=0, j=1; i < n && j < n; );
   2.1 填写偶数层：
      2.1.1 填数直至 i 等于 j：
            z[i++][j]=number++;
      2.1.2 填数直至 j 等于 0：
            z[i][j--]=number++;
   2.2 填写奇数层：
      2.2.1 填数直至 i 等于 j：
            z[i][++j]=number++;
      2.2.2 填数直至 i 等于 0：
            z[i--][j]=number++;
```

【算法分析】　算法 Full 需要依次填写矩阵的每一个元素，时间复杂度是 $O(n^2)$。

【算法实现】　注意每一层填数后都要调整下标，程序如下。

```
void Full(int z[100][100], int n)
{
  int number, i, j;
```

源代码 3-7

```
    z[0][0] = 1; number = 2;
    for (i = 0, j = 1; i < n && j < n; )                    //依次填写每一层
    {
      while (i < j)
        z[i++][j] = number++;
      while (j >= 0)
        z[i][j--] = number++;
      i++; j = 0;
      while (i > j)
        z[i][j++] = number++;
      while (i >= 0)
        z[i--][j] = number++;
      j++; i = 0;
    }
  }
```

实验3　埃氏筛法的优化

【实验题目】　对于埃拉托色尼筛法，观察图3-2，很多合数被标记了不止1次，例如合数6，在将2过筛时标记了1次，将3过筛时又标记一次，造成了不必要的重复标记。请改进埃拉托色尼筛法，使得在线性时间内完成所有合数的标记。

【实验要求】　对于优化的埃拉托色尼筛法，要保证两点：①合数一定被标记；②每个合数都没有被重复标记。

【实验提示】　注意到任何合数都能表示成一系列素数的乘积，如果保证每个合数只被其最小质因数筛去，就能够达到不重复标记。对于整数 i，对所有不超过 i 的素数 p，将 $i * p$ 标记为合数，如果 p 是 i 的因子则将整数 i 进行标记并停止过筛，直至 p 等于 i，说明整数 i 是素数。

习　题　3

1. 如果一个十进制的正整数能够被7整除，或者某个位置的数字是7，则称该正整数为与7相关的数。求小于 $n(n<100)$ 的所有与7无关的正整数的平方和。

2. 对于一元二次方程 $ax^2+bx+c=0$，给定系数 a、b 和 c，求方程的根。要求所有实数精确到小数点后5位。

3. 恺撒加密由古罗马恺撒大帝在其政府的秘密通信中使用而得名，其基本思想是：将待加密信息(称为明文)的每个字母在字母表中向后移动常量key(称为密钥)，得到加密信息(称为密文)。假设字母表为英文字母表，对于给定的明文和密钥，请给出恺撒加密后的密文。

4. 校门外的树。校门外长度为 L 的马路上有一排树，每两棵相邻的树之间的距离都

是 1m。可以把马路看成一个数轴，校门口在数轴 0 的位置，另一端在 L 的位置，数轴上每个整数点都有一棵树。马路上有一些区域要用来修建地铁，这些区域用数轴上的半开区间$[s, t)$来表示，已知 s 和 t 都是整数，且区域之间可能有部分重叠。由于修建地铁要把区域中的树全部移走，问马路上还剩下多少棵树？

5. 对于约瑟夫环问题，假设每个人持有的密码不同，请修改 3.2.1 节给出的算法。

6. 桥牌共 52 张，没有大小王，按 E、S、W、N 的顺序把 52 张牌随机发给四个玩家，请列出每个玩家的发牌情况。

7. 定义 n 阶间断折叠方阵是把 n^2 个整数折叠填写到 n 阶方阵中，起始数 1 置于方阵的左上角，然后从起始数开始递增，每一层从第 1 行开始，先向下再折转向左，层层折叠地排列为间断折叠方阵。图 3-6 所示为 5 阶间断折叠方阵。请构造并输出任意 n 阶间断折叠方阵。

1	2	5	10	17
4	3	6	11	18
9	8	7	12	19
16	15	14	13	20
25	24	23	22	21

图 3-6　5 阶间断折叠方阵

8. 泊松分酒。法国数学家泊松(Poisson)提出的分酒趣题：有一瓶 12 品脱(容量单位)的酒，同时有容积为 5 品脱和 8 品脱的空杯各一个，借助这两个空杯，如何将这瓶 12 品脱的酒平分？