Chapter 5

Induction

Techniques based on recursions

For many problems, the use of recursion makes it possible to solve complex problems using algorithms that are concise, easy to comprehend and efficient (from an algorithmic point of view)

In its simplest form, recursion is the process of dividing the problem into one or more subproblems, which are identical in structure to the original problem and then combine the solutions to these subproblems to obtain the solution to the original problem

Three special cases: 1) Induction or tail-recursion; 2) Nonoverlapping subproblems; 3) Overlapping subproblems with redundant invocations to subproblems, allowing trading space for time

递归的概念

- 直接或间接地调用自身的算法称为**递归算法**。用函数自身给出定义的函数称为**递归函数**。
- 由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

下面来看几个实例。

递归的概念

例1　阶乘函数

阶乘函数可递归地定义为：

$$n!=\begin{cases}1 & n=0\\ n(n-1)! & n>0\end{cases}$$

边界条件

递归方程

边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

递归的概念

例2　Fibonacci数列

无穷数列1，1，2，3，5，8，13，21，34，55，…，被称为Fibonacci数列。它可以递归地定义为：

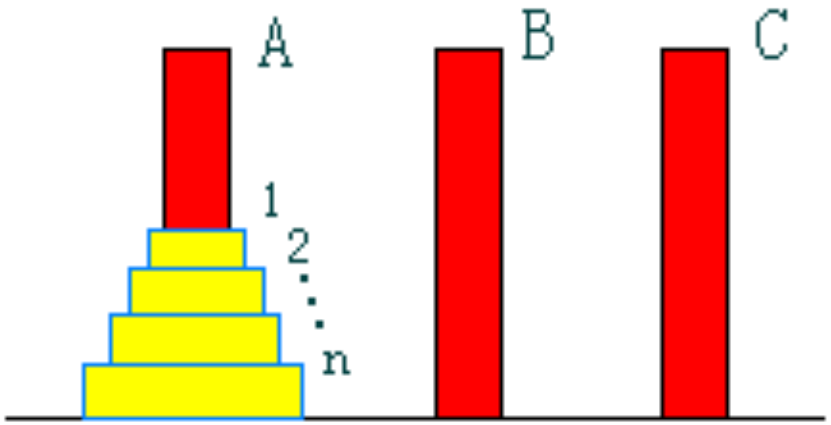
$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

边界条件

递归方程

第n个Fibonacci数可递归地计算如下：

```
public static int fibonacci(int n)
  {
     if (n <= 1) return 1;
     return fibonacci(n-1)+fibonacci(n-2);
  }
```


A
B
C
1
2
n

递归的概念

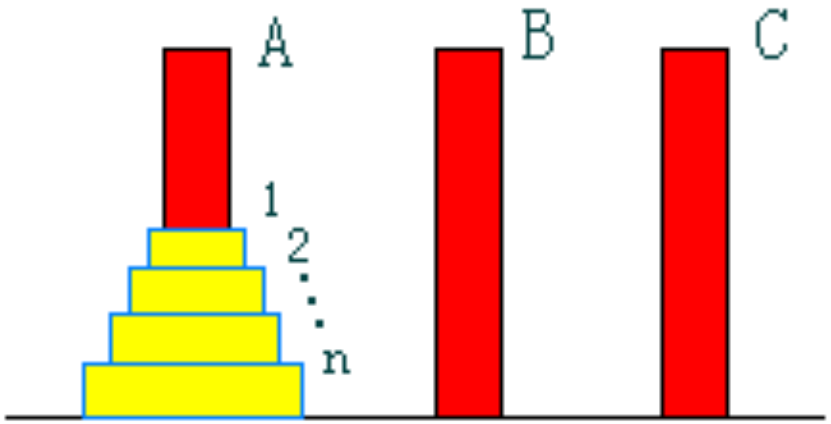
例3　Hanoi塔问题

设a,b,c是3个塔座。开始时，在塔座a上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,n,现要求将塔座a上的这一叠圆盘移到塔座b上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：

规则1：每次只能移动1个圆盘；

规则2：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；

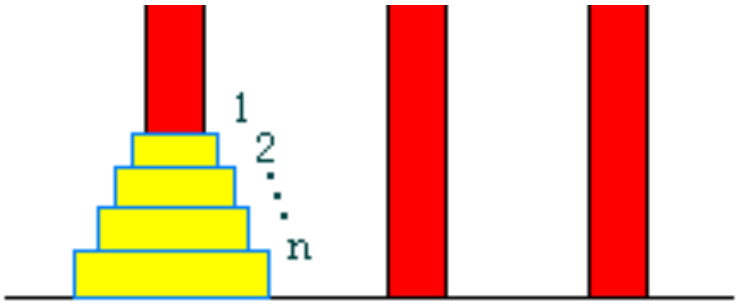
规则3：在满足移动规则1和2的前提下，可将圆盘移至a,b,c中任一塔座上。



递归的概念

例3　Hanoi 塔问题

```
public static void hanoi(int n, int a, int b, int c)
  {
     if (n > 0)
     {
        hanoi(n-1, a, c, b);
        move(a,b);
        hanoi(n-1, c, b, a);
     }
  }
```



递归小结

优点：结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。

缺点：递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。

递归小结

解决方法：在递归算法中消除递归调用，使其转化为非递归算法。

1.采用一个用户定义的栈来模拟系统的递归调用工作栈。该方法通用性强，但本质上还是递归，只不过人工做了本来由编译器做的事情，优化效果不明显。

2.用递推来实现递归函数。

3.通过Cooper变换、反演变换能将一些递归转化为尾递归，从而迭代求出结果。

后两种方法在时空复杂度上均有较大改善，但其适用范围有限。

递归小结

解决方法：在递归算法中消除递归调用，使其转化为非递归算法。

1.采用一个用户定义的栈来模拟系统的递归调用工作栈。该方法通用性强，但本质上还是递归，只不过人工做了本来由编译器做的事情，优化效果不明显。

2.用递推来实现递归函数。

3.通过Cooper变换、反演变换能将一些递归转化为尾递归，从而迭代求出结果。

后两种方法在时空复杂度上均有较大改善，但其适用范围有限。

Mathematical induction

First we prove that the property holds for n0. This is called the basis step. Then we prove that whenever the property is true for n0, n0+1, ... , n-1, then it must follow that the property is true for n. This is called the induction step. We then conclude that the property holds for all values of $n \geq n0$.

Induction

Given a problem with parameter n, designing an algorithm by induction is based on the fact that if we know how to solve the problem when presented with a parameter less than n, called the induction hypothesis, then our task reduces to extending that solution to include those instances with parameter n

The advantage of this technique is that the proof of correctness of the designed algorithm is naturally embedded in its description

原理

(1) 解决问题的一个小规模事例是可能的(基础事例)

(2) 每一个问题的解答都可以由更小规模问题的解答构造出来(归纳步骤)。

关键：如何简化问题。

Selection sort

Algorithm 5.1 SELECTIONSORTSEC

Input: An array A[1..n] of n elements

Output: A[1..n] sorted in nondecreasing order

1. sort(1)

Procedure sort(i) {Sort A[i..n]}

1. if i<n then

2. k←i

3. for j←i+1 to n

4. if A[j]<A[k] then k←j

5. endfor

6. if k≠i then interchange A[i] and A[k]

7. sort(i+1)

8. end if

$$C(n) = \begin{cases} 0 & if\ n = 1 \\ \mathrm{C(n-1)} + \mathrm{(n-1)} & if\ n \geq 2 \end{cases}$$

$$C(n) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \Theta(n^2)$$

Insertion sort

Algorithm 5.2 INSERTIONSORTREC

Input: An array A[1..n] of n elements

Output: A[1..n] sorted in nondecreasing order

```
1. sort(n)

Procedure sort(i)    {Sort A[1..i]}
1.  if i>1 then
2.      x←A[i]
3.      sort(i-1)
4.      j←i-1
5.      while j>0 and A[j]>x
6.          A[j+1]←A[j]
7.          j←j-1
8.      end while
9.      A[j+1]←x
10. end if
```

Radix sort

7467	6792	9134	9134	1239
1247	9134	1239	9187	1247
3275	3275	1247	1239	3275
6792	4675	7467	1247	4675
9187	7467	3275	3275	6792
9134	1247	4675	7467	7467
4675	9187	9187	4675	9134
1239	1239	6792	6792	9187

Radix sort

Algorithm 5.3 RADIXSORT

Input: A linked list of numbers L={a1, a2, ..., an} and k, the number of digits

Output: L sorted in nondecreasing order

```
1.  for j←1 to k
2.      Prepare 10 empty lists L0, L1, ..., L9
3.      while L is not empty
4.          a←next element in L. Delete a from L
5.          i← jth digit in a. Append a to list Li
6.      end while
7.      L←L0
8.      for i←1 to 9
9.          L←L,Li     {append list Li to L}
10.     end for
11. end for
12. return L
```

Time: $\Theta(kn)$

Space: $\Theta(n)$

Integer exponentiation

Q: How to raise a real number x to the nth power?

A: x^n=(((x*x)*x)*...*x)*x

But it is inefficient, as it requires Θ(n) multiplication

An efficient method:

Let m=⌊n/2⌋ and suppose we know how to compute x^m, then

If n is even, x^n=(x^m)^2

If n is odd, x^n=x(x^m)^2

Integer exponentiation

Algorithm 5.4 EXPREC

Input: A real number x and a nonnegative integer n

Output: x^n

1. power(x,n)

Procedure power(x,m) {Compute x^m}

1. if m=0 then y←1

2. else

3. y←power(x,⌊m/2⌋)

4. y←y^2

5. if m is odd then y←xy

6. end if

7. return y

Only Θ(logn)

Integer exponentiation

Algorithm 5.5 EXP

Input: A real number x and a nonnegative integer n

Output: x^n

1. y←1
2. Let n be $d_k d_{k-1} \ldots d_0$ in binary notation
3. for j←k down to 0
4. y←y^2
5. if d_j=1 then y←xy
6. end for
7. return y

This is the iterative version

多项式求值

问题：给定一串实数a_n，a_{n-1}，…，a_1，a_0，和一个实数x，计算多项式$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$的值。

归纳假设：已知如何在给定a_{n-1}，…，a_1，a_0和点x的情况下求解多项式(即已知如何求解$P_{n-1}(x)$)。

$$P_n(x)= P_{n-1}(x)+a_nx^n$$

需要$n(n+1)/2$次乘法和n次加法运算。

观察：有许多冗余的计算，即x的幂被到处计算。

更强的归纳假设：已知如何计算多项式$P_{n-1}(x)$的值，也知道如何计算x^{n-1}。

计算x^n仅需要一次乘法，然后再用一次乘法得到a_nx^n，最后用一次加法完成计算，总共需要2n次乘法和n次加法。

多项式求值

归纳假设(翻转了顺序的)：已知如何计算$P'_{n-1}(x)=a_nx^{n-1}+a_{n-1}x^{n-2}+\ldots+a_1$。

$P_n(x)=xP'_{n-1}(x)+a_0$。所以，从$P'_{n-1}(x)$计算$P_n(x)$仅需要一次乘法和一次加法。

该算法仅需要n次乘法和n次加法，以及一个额外的存储空间。

窍门是很少见的从左到右地考虑问题的输入，而不是直觉上的从右到左。另一个常见的可能是对比自上而下与自下而上(当包含一个树结构时)。

Evaluating polynomials

Suppose we want to evaluate the polynomial

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

The straightforward approach would be to evaluate each term separately. It is very inefficient since it requires n+(n-1)+...+1=n(n+1)/2 multiplications

Horner's rule:

$$P_n(x) = ((\ldots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})\ldots)x + a_1)x + a_0$$

Suppose we know how to evaluate

$$P_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_2 x + a_1$$

we have

$$P_n(x) = xP_{n-1}(x) + a_0$$

Evaluating polynomials

Algorithm 5.6 HORNER

Input: A sequence of n+2 real numbers a_0, a_1, ..., a_n and x

Output: $Pn(x)=a_nx^n+a_{n-1}x^{n-1}+...+a_1x+a0$

1. $p \leftarrow a_n$
2. for $j \leftarrow 1$ to n
3. $p \leftarrow xp+a_{n-j}$
4. end for
5. return p

Only n multiplications and n additions

Generating permutations

Algorithm 5.7 PERMUTATIONS1

Input: A positive integer n

Output: All permutations of the number 1,2,...,n

```
1. for j←1 to n
2.     P[j]←j
3. end for
4. perm1(1)
```

Procedure perm1(m)

```
1. if m=n then output P[1..n]
2. else
3.     for j←m to n
4.         interchange P[j] and P[m]
5.         perm1(m+1)
6.         interchange P[j] and P[m]
7.         at this point P[m...n]=m, m+1, ..., n
8.     end for
9. end if
```

Count the number iterations of the for loop in the Procedure perm1

Generating permutations

Algorithm 5.8 PERMUTATIONS2

Input: A positive integer n

Output: All permutations of the number 1,2,...,n

```
1. for j←1 to n
2.     P[j]←0
3. end for
4. perm2(n)
```

Procedure perm2(m)

```
1.  if m=0 then output P[1...n]
2.  else
3.      for j←1 to n
4.          if P[j]=0 then
5.              P[j]←m
6.              perm2(m-1)
7.              P[j]←0
8.          end if
9.      end for
10. end if
```

Count the number iterations of the for loop in the Procedure perm1

社会名流问题

在n个人中，一个被所有人知道但却不知道别人的人，被定义为社会名流。

最坏情况下可能需要问n(n-1)个问题。

问题：给定一个n×n邻接矩阵，确定是否存在一个i，其满足在第i列所有项(除了第ii项)都为1，并且第i行所有项(除了第ii项)都为0。

社会名流问题

考察n-1个人和n个人问题的不同。由归纳法，我们假定能够在n-1个人中找到社会名流。由于至多只有一个社会名流，所以有三种可能：(1) 社会名流在最初的n-1人中，(2) 社会名流是第n个人，(3) 没有社会名流。但仍有可能需要n(n-1)次提问

“倒推”考虑问题。确定一个社会名流可能很难，但是确定某人不是社会名流可能会容易些。如果我们把某人排除在考虑之外，则问题规模从n减小到n-1。

算法如下：问A是否知道B，并根据答案删除A或者B。假定删除的是A。则由归纳法在剩下的n-1个人中找到一个社会名流。如果没有社会名流，算法就终止；否则，我们检测A是否知道此社会名流，而此社会名流是否不知道A。

```
Algorithm Celebrity(Know)
Input: Know (an n*n Boolean matrix)
Output: celebrity
begin
   i=1;
   j=2;
   next=3;
   {in the first phase we eliminate all but one candidate}
   while next<=n+1 do
      if Know[i,j] then i=next
      else j=next;
      next=next+1;
   if i=n+1 then candidate=j
   else candidate=i
   {now we check that the candidate is indeed the celebrity}
   wrong:=false;
   k=1;
   Know[candidate,candidate]=false;
   while not wrong and k<=n do
      if Know[candidate,k] then wrong=true;
      if not Know[k,candidate] then
         if candidate!=k then wrong=true;
      k=k+1;
   if not wrong then celebrity=candidate
   else celebrity=0
end
```

算法被分为两个阶段：
1）通过消除只留下一个候选者，
2）检查这个候选者是否就是社会名流。
至多要询问3(n-1)个问题：
第一阶段的n-1个问题用于消除n-1个人，
而为了验证侯选者就是社会名流至多要
2(n-1)个问题。

Finding the majority element

Let A[1..n] be a sequence of integers. An integer a in A is called the majority if it appears more than $\lfloor n/2 \rfloor$ times in A

Brute-force method? $\Theta(n^2)$

Sort and count? $\Theta(n \log n)$ in the worst case

Find the median? $\Theta(n)$

Observation: If two different elements in the original sequence are removed, then the majority in the original sequence remains the majority in the new sequence

Finding the majority element

Algorithm 5.9 MAJORITY

Input: An array A[1..n] of n elements

Output: The majority element if it exists, otherwise none

1. c←candidate(1)
2. count←0
3. for j←1 to n
4. if A[j]=c then count←count+1
5. end for
6. if count>⌊n/2⌋ then return c
7. else return none

Procedure candidate(m)

1. j←m, c←A[m], count←1
2. while j<n and count>0
3. j←j+1
4. if A[j]=c then count←count+1
5. else count←count-1
6. end while
7. if j=n then return c
8. else return candidate(j+1)

作业

- 5.8 5.17(a) 5.19(a) 5.20 5.21 5.28 5.33