第32卷第1期 上海师范大学学报(自然科学版) Vol. 32, No. 1
2003年3月 Journal of Shanghai Teachers University(Natural Sciences) 2003, Mar.

分治算法的两种思路和形式

王海源

(上海师范大学 数理信息学院, 上海 200234)

摘 要: 分治算法是程序设计中常用算法之一,是用划分子问题的方法,由较小尺寸的问题的可解导致原问题的解决. 介绍了分治算法的两种形式,分析和探讨了它们的不同思路、不同特点和适用场合.

关键词: 分治算法; 递归; 分解; 聚合; 原子问题

中图分类号: TP311.52 **文献标识码**: A **文章编号**: 1000-5137(2003)01-0039-05

0 引 言

分治算法是程序设计中常用的算法. 它的基本出发点是把某个需要解决的而一时难以直接解决的问题,化作若干个在尺寸上小于原问题而在形式上和原问题又完全相同的问题,分别加以解决. 虽然一次分解的直接结果一般不能直接解决,但通过继续照此办理,可望最终化为能被直接解决的同类小问题,这些小问题的解决,就意味着原问题的最终解决.

通常,这种分析方法的基本点在于"分解",因此这种方法也被称为"划分(Divide)和解决(Conquer)"方法. 也正因为如此,它和语言工具中的递归结下了不解之缘. 然而,这种由问题的顶部出发进行递归向下分析的思考方法并非分治算法的唯一思路,另一种对于上述方法反方向的思路及非递归的算法,采用对于子问题的"聚合"为基点的分治算法同样有其实用价值,同样实施了"由小尺寸的问题的解决导致大尺寸同类问题解决"的分治法思路,在实用上也有其积极意义.

本文对分治算法的这两种思路和形式加以分析和探讨.

1 基于"分解"的顶向下的分治方法

这种方法是从原问题出发,层层分解. 每一层分解都把原问题分解成尺寸较小的同类子问题,逐次逼近问题的最小化底部,而这个底部的最小问题明显直接可解,我们称之为"原子问题". 原子问题的可解性保证了它的上一层同类问题的可解性,而上一层同类问题的可解性又保证了再上层的可解性……,从而保证了原始问题的解决.

这是分治方法的最常见思路,也由许多实际问题的解决而被证明为正确. 这种方法是由原问题以原始尺寸的分解而开始的,它所分解的第一步骤形成的若干子问题在通常情况下一般不是直接可解的,所以它的层层递归是不可避免的. 例1的快速排序就是一个典型的例子.

收稿日期: 2002-06-23

作者简介: 王海源(1947-),男,上海师范大学数理信息学院副教授.

例 1 快速排序(Quick Sorting)

```
void quick_sort (int e[ ], int low, int high)
{ int i, j, t;
  if (low < high)
    { i = low; j = high; t = e[low];
      while (i < j)
        { while (i < j&&e[j] > t) j - -; if (i < j) e[i + +] = e[j];
          while (i < j&&e[i] < = t) i - -; if (i < j) e[j - -] = e[i];
          }
       e[i] = t; quick_sort (e, low, i - 1); quick_sort (e, i + 1, high);
     }
  }
```

快速排序算法,把一个处在线性形表[low]到[high]范围内的元素序列的排序要求,转化为如下的处理:以一个元素为"界标",进行划分(比"界标"小的元素居左边,比"界标"大的元素居右边,界标居中),然后对左右两半分别进行快速排序.这种划分不断进行,直到区域元素个数为 1 或 0.这种尺寸为 0 或 1 的排序问题就是原子问题.

汉诺塔问题(Hanoi)和斐波那契(Fibonacci)序列的求解,均可采用这种处理方法.这种由顶向下逐层分解的方法,思路清晰,编码简捷.但是,由于它是以递归为基础,不可避免地有其缺憾,主要在如下两个方面:其一是:递归技术本身需耗费较大的机器资源;其二是:某些递归方法不得不引入大量重复计算.

第一点是必要消耗,影响不大,一般可以接受;而第二点则可能造成较大的浪费,例 2 就是这方面的实例.

例 2 计算第 n 个斐波那契数 Fibo(n).

斐波那契数 Fibo(n)定义为: Fibo(1) =0, Fibo (2) =1, 当 $n > 2$ 时有: Fibo(n) = Fibo($n-1$) + Fibo ($n-2$).

这一划分方式导致了如下递归子程序的自然产生:

```
int fibo(int n)
  { if (n = =1) return 0;
    else if (n = =2) return 1;
         else return fibo(n - 1) + fibo(n - 2);
  }
```

这个函数若用 fibo(6)调用,可以用图 1 树状图描绘其调用结构:

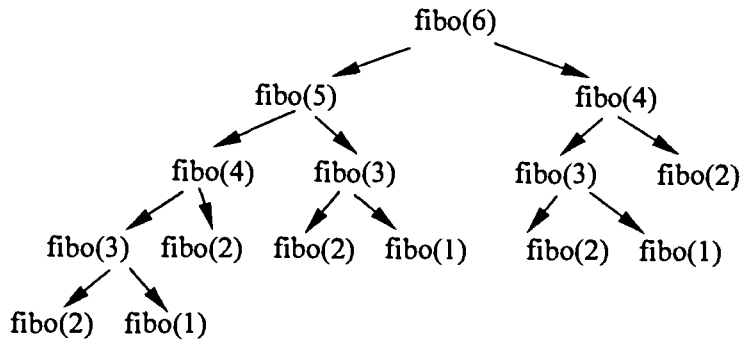

图 1 调用结构树状图

在图 1 调用结构图中, 每棵子树所表示的处理都是独立进行的, 这就意味着无法按处理的先后序借用已有的同类运算的结果. 比如, fibo(4) 要计算(调用) 2 次, fibo(3) 要计算(调用) 3 次, 等. 事实上, 当要计算的 fibo(n) 的尺寸 n 很大时, 若以每次子程序调用作为计数单位, 在图 1 的子程序调用结构图中可以看出, 在参数大于 2 那些递归调用中, 每个调用都包含着两个调用, 呈现一种"一而二, 二而四"的调用格局. 在 n 相当大时, 这个树状图接近于丰满, 而高度为 $n-2$. 所以, 调用次数(也就是树的结点总数) 近似于 $2n-2$, 这是一种不能等闲视之的低效率.

2　基于"聚合"的底向上的分治方法

如果知道了最终直接可解的原子问题的全部情况, 那么可采用由原子问题出发, 向上递推的方法, 由已有结论的小尺寸问题的解决, 导致比它高一层次的同类问题的解决, 最终使原问题彻底解决. 这就是和递归向下方法对应的由底向上的分治方法的基本思路.

例 3　对于 n 个球队($n=2k$, k 为自然数), 要排定它们进行单循环比赛的全部赛程, 使每队每天无重复地赛一场, 使全部赛事在 $n-1$ 天完成.

问题的最小尺寸即原子问题, 是只有 2 个队的赛事安排(这时 $k=1$), 可解性是明显的: 赛一场, 即: 在这一天, 第 1 队的对手是第 2 队, 第 2 队的对手是第 1 队. 如图 2 表示.

	第 1 天
第 1 队	2
第 2 队	1

图 2　最小尺寸的赛事安排

若已产生了 n ($n=2k$) 个球队的赛事表, 如何由此推出 $2n$ 个球队的赛事表呢?其实, 只要在已有的 n 个球队的 $n-1$ 天的赛事表上补上第 $n+1$, 第 $n+2$, ..., 第 $n+n$(即第 $2n$) 这 n 个球队的这前 $n-1$ 天的赛事; 再补上全部这 $2n$ 个球队的后 n 天的赛事就行了. 以 $n=4$ 为例, 在 $n1=22=4$ 已解决的情况下, $n2=2n1=8$ 的赛事表可以看作四部分的组合. 图3 表示了由 2 个队的赛事表求 4 个队的赛事表, 及由 4 个队的赛事表求 8 个队的赛事表的演变.

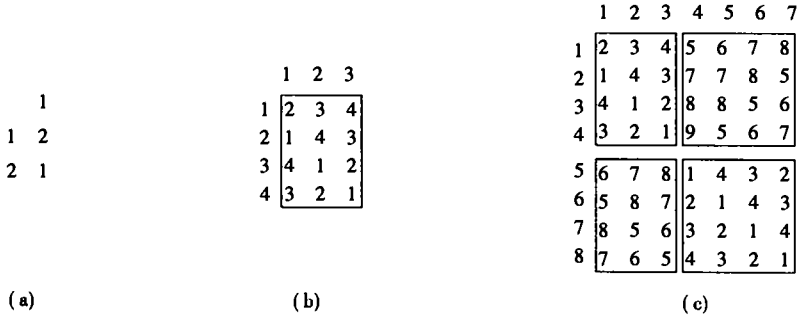

图 3　赛事表的演变

图 3 中, 每一个子图的第一列表示球队号, 每一个子图的第一行表示第几天, 我们观察(b) 到(c) 的演变, 可以看出: (c) 图按虚线划分为 4 个区域, 左上角区域和(b) 完全一致, 而其他 3 个区域按如下方式由左上角区域直接或间接生成, 即可由(b) 导出. (c) 图中 4 个区域的内容生成方式如下: 左上角: 直接由(b) 复制而来; 左下角: 由(c) 图左上角对应位置的每一元素加 4, 其实在前 3 天第 5, 6, 7, 8 队间进

行相互比赛;右上角:第 4 天,让 1 至 4 号队分别对应 5 至 8 号队比赛,以后各天,依次由前 1 天的赛事安排轮转;右下角:根据比赛的相互性(例如:第 4 天,1 号队对 5 号队,同日,5 号队对 1 号队),参照右上角完成.

从以上算法不难看出:只要原子问题的解答是明确的和已知的,分解的子问题和高一级同类问题的依赖关系是明确的,就可以按由底向上的非递归方法,由分治法得到解决.例 3 程序如下:

```
# include <stdio.h>
# define MAXN 64
int a[MAXN+1][MAXN]
void main( )
 {int twom1, twom, i, j, m, k;
  printf("请指定 n(=2 的 k 次幂)个球队,请输入 k: \n");
  scanf("%d", &k);
  a[1][1]=2; a[2][1]=1;              /*原子问题:两个球队的赛程*/

  m=1; twom1=1;
  while(m<k)
   {m++; twom1+=twom1; twom=2*twom1;
        /*由 2m-1 个球队的赛程求 2m 个球队的赛程*/
    for (i=twom1+1; i<=twom; i++)   /*这个二重循环填 2m 个队赛程表的左下角*/
      for (j=1; j<=twom1-1; j++) a[i][j] = a[i-twom1][j]+twom1;
    a[1][twom1] = twom1+1;          /*如下填 2m 个队赛程表的右上角*/
    for (i=2; i<=twom1; i++) a[i][twom1] = a[i-1][twom1]+1
    for (j=twom1+1; j<twom; j++)
      {for (i=1; i<twom1; i++) a[i][j] = a[i+1][j-1];
       a[twom1][j] =a[1][j-1];
      }
    for (j=twom1; j<twom; j++)      /*这个二重循环填 2m 个队赛程表的右下角*/
      for (i=1; i<=twom1; i++) a[a[i][j]][j] = i;
    for (i=1; i<=twom; i++)         /*输出 2m 个队赛程表*/
      { for (j=1; j<twom; j++) printf("%4d", a[i][j]);
        printf("\n");
      }
    printf("\n");
    }
  }
```

除了上述例 3 以外,也易于用非递归方法求解斐波那契序列第 n 项.简要写出其程序段为:

```
a=1; b=2;
for ( i=3; i<=n; i++) {c=a+b; a=b; b=c; }
printf("第%d 个斐波那契数为:%d\n", n, c);
```

3 两种分治方法的分析对比

通过以上分析,可以看到:自顶向下的递归分析方法具有较强的适应性;无论对问题的底部原子问题的出现次序、个数及具体情况是否可以完全预测,只要原子问题可解,原问题总可解出;另外,其算法设计过程也较为明确和直观.正因为如此,这种方法的应用面较广.例如:树、图的遍历及众多的处理,多种排序算法(例快速排序,合并排序)都是这种方法的应用实例.

另一方面,只要原子问题的解答是明确的和已知的,则这类问题除了可以用顶向下的递归分析法实施分治以外,还可以按由底向上分治法用非递归方法得到解决.它可以避开对语言工具的递归功能要求,避免了种种由递归所带来的效率问题.比如例2斐波那契序列第 n 项的求解,若采用这种思路,那么可以根据 fibo(1) 和 fibo(2) 的已知求出 fibo(3),然后根据 fibo(2), fibo(3) 确定 fibo(4),……,于是,它不需要递归,只需要这种递推,只要对 fibo(3), fibo(4) 和 fibo(5) 各计算一次,就可以最终计算出 fibo(6).这样,它避免了前例中由递归带来的大量重复计算,较为明显地加速了算法过程.

很多问题的分治解法,既可以用递归向下分析来解决,也可以用非递归向上分析来解决.例如合并排序(Merge_sorting)和前述的斐波那契序列的解法.合并排序是以"成对合并较小有序序列,形成较大有序序列"为基本算法,无论用分解,或用合并,其原子问题都是相邻的长度为1的有序序列的合并;斐波那契序列问题,原子问题则为序列第一第二元素为0和1.合并排序无论用顶向下或是底向上,时间复杂度是一样的;斐波那契序列的解法则如前文所分析,两种分治方法的时间复杂度有明显的差异.

一般情况下,用底向上非递归分治方法解决的问题,大多可以用递归向下分析来解决.但是,并不是所有可以用递归向下分治方法解决的问题都是适用底向上非递归方法解决的.比如在例1中的快速排序,就因为划分点是动态确定的,导致无法预测原子问题(尺寸为0或1的问题)的分布的确切情况,无法用底向上分治方法解决.另外,非递归方法在具体算法设计上,一般也会比非递归方法复杂.

参考文献:

[1] 夏宽理.C程序设计实例详解[M].上海:复旦大学出版社,1996.
[2] 吴伟民,严蔚敏.数据结构[M].北京:清华大学出版社,1997.
[3] 王春森.高级程序员教程[M].北京:清华大学出版社,2001.

Two Idears and Forms of the Algorithm "Divide and Conquer"

WANG Hai-yuan
(Mathematics and Sciences College, Shanghai Teachers University, Shanghai 200234, China)

Abstract: The algorithm "divide and conquer" is one of the frequently used method. On this algorithm the divide process is used, a problem is solved by solving the smaller size cognate problems. Two forms of the algorithm "divide and conquer" are introduced; their ideal differences, features and application situations are analyzed and discussed.

Key words: divide and conquer; recurrence; aggregation; decomposition; atomic problem