算法分析与设计 第三讲

递归与分治 基本概念

主要内容

- 递归
- 递归实例
- 递归式
- 分治法
- 分治法实例

递归的概念

- 递归函数
  - 用函数自身给出定义的函数
- 递归算法
  - 一个算法包含对自身的调用
  - 这种调用可以是直接的，也可以是间接的

递归举例-阶乘函数

- 阶乘函数

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

递归出口

递归方程

```
int factorial(int n)
{
    if(n==0) return 1;
    else return n*factorial(n-1);
}
```

$$n!=1\cdot 2\cdot 3\cdot\cdots\cdot(n-1)\cdot n$$

递归举例-Fibonacci数列

- Fibonacci数列
  - 无穷数列1，1，2，3，5，8，13，21，34，55，…

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

递归出口

递归方程

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

递归举例-Fibonacci数列

- Fibonacci数列 非递归定义

$$F(n)=\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right)$$

递归举例-整数划分问题

- 整数划分问题
  - 将正整数n表示成一系列正整数之和：$n=n_1+n_2+\ldots+n_k$，其中$n_1 \geq n_2 \geq \ldots \geq n_k \geq 1$，$k \geq 1$
  - 正整数n的这种表示称为正整数n的划分
  - 正整数n的不同的划分个数称为正整数n的划分数$p(n)$
  - 目标：求正整数n的不同划分个数$p(n)$
  - 例，正整数5的划分,$p(5)=7$

递归举例-整数划分问题

- 整数划分问题
- 引入m,将最大加数n_1不大于m的划分个数记作$q(n,m)$

$$q(n,m)=\begin{cases} 1 & n=1, m=1 \\ q(n,n) & n<m \\ 1+q(n,n-1) & n=m \\ q(n,m-1)+q(n-m,m) & n>m>1 \end{cases}$$

- $p(n)=q(n,n)$

递归举例-整数划分问题

- 整数划分问题

```
int q(int n,int m)
{
    if((n<1)||(m<1)) return 0;
    if((n==1)||(m==1)) return 1;
    if(n<m) return q(n,n);
    if(n==m) return q(n,m-1)+1;
    return q(n,m-1)+q(n-m,m);
}
```

递归举例-汉诺(Hanoi)塔问题

- 汉诺(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*
  - 要求将塔座a上的圆盘移到塔座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：
    - 每次只能移动1个圆盘
    - 任何时刻都不允许将较大的圆盘压在较小的圆盘之上
    - 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

递归举例-汉诺(Hanoi)塔问题

- 汉诺(Hanoi)塔问题分析
  - n=1时，直接a->b即可
  - n>1时，借助c实现移动，可先将n-1个圆盘按照规则a->c，再将大圆盘a->b，最后将n-1个圆盘c->b
  - 可以通过递归实现

伪码：

```
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

- 递归的概念非常重要
- 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，为设计算法、调试程序带来很大便利
- 缺点
  - 递归算法的运行效率较低

递归式（Recurrence）

- 当一个算法包含对自身的递归调用时，其运行时间通常可以用递归式来表示
- 例，对于合并排序，其最坏情况时间复杂度

$$T(n)=\begin{cases}\Theta(1) & if\ \ n=1\\ 2T(n/2)+\Theta(n) & if\ \ n>1\end{cases}$$

- 其解为 $T(n)=\Theta(n\lg n)$

递归式的解法

- 代换法（substitution method）
- 递归树方法（recursion-tree method）
- 主方法（master method）

代换法（substitution method）

- 步骤
  - 猜测解的形式
  - 用数学归纳法证明之
- 只适用于解的形式很容易猜的情形
- 如何猜测则需要经验
- 例 $T(n)=2T(\lfloor n/2 \rfloor)+n$ $\quad T(n)=O(n\lg n)$

$$T(n)=2T(\lfloor n/2 \rfloor+17)+n \qquad T(n)=O(n\lg n)$$

$$T(n)=T(\lceil n/2 \rceil)+1 \qquad T(n)=O(\lg n)$$

递归树方法（recursion-tree method）

- 每一个节点代表递归函数调用集合中一个子问题的代价，将所有层的代价相加得到总代价
- 当用递归式表示算法的时间复杂度时，可用递归树的方法
- 递归树方法模拟了算法的递归执行，可以由递归树方法产生对算法时间复杂度的较好猜测

递归树方法示例

求解 $T(n) = T(n/4) + T(n/2) + n^2$

主方法（master method）

- $T(n) = aT(n/b) + f(n)$
- $a>=1$, $b>1$, a和b均为常数
- $f(n)$是渐近正函数

主定理（master theorem）

- 对于递归式，比较$f(n)$和$n^{\log_b a}$
- 若对于某常数$\varepsilon > 0$, $f(n) = O(n^{\log_b a-\varepsilon})$,则$T(n) = O(n^{\log_b a})$
- 若 $f(n) = \Theta(n^{\log_b a})$,则$T(n) = \Theta(n^{\log_b a} \lg n)$
- 若对于某常数$\varepsilon > 0$,有 $f(n) = \Omega(n^{\log_b a+\varepsilon})$

  且对常数$c<1$与所有足够大的n,有

  $af(n/b) \le cf(n)$,则$T(n) = \Theta(f(n))$

主方法的应用

- 请注意，上述三种情况没有覆盖所有的$f(n)$
- 在应用时需要注意是否符合这三种情况
- $T(n) = 4T(n/2) + n$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^3$
- $T(n) = 4T(n/2) + n^2/\lg n$
- $T(n) = 2T(n/2) + n\lg n$

分治法

- 分治法的基本策略
  - 分解（Divide）：将原问题分解为子问题
  - 解决（Conquer）：求解子问题
  - 合并（Combine）：组合子问题的解得到原问题的解

分治法的适用条件

- 适合分治法求解的问题一般具有以下特征
  - 问题的规模缩小到一定程度就可以容易地解决
  - 问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质
  - 基于子问题的解可以合并为原问题的解
  - 问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题

平衡

- 使子问题规模尽量接近的做法，就是平衡
- 在使用分治法和递归时，要尽量把问题分成规模相等，或至少是规模相近的子问题以提高算法的效率

分治法实例

- 二分搜索（Binary search）
  - 问题：在已排好序的数组中寻找特定元素
  - 分解：检查中间元素
  - 解决：递归搜索子数组
  - 合并：……
- 例，在数组中寻找9
- 3　5　7　8　9　12　15

二分搜索分析

$$T(n) = 1T(n/2) + \Theta(1)$$

子问题数目

子问题规模

分解和合并代价

- 应用主方法，对应第二种情况
- 即 $a = 1, b = 2, n^{\log_b a} = n^0 = 1$

$$f(n) = \Theta(1) = \Theta(n^{\log_b a}), \text{则} T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$$

分治法实例

- Powering a number，求a^n
  - 直接求解：Θ(n)
  - 分治法求解

$$a^n = \begin{cases} a, & if\ \ n = 1; \\ a^{n/2} * a^{n/2} & if\ \ n > 1\, and\ \ a\ is\ even; \\ a^{(n-1)/2} * a^{(n-1)/2} * a & if\ \ n > 1\, and\ \ a\ is\ odd. \end{cases}$$

$$T(n) = T(n/2) + \Theta(1)$$

$$a = 1, b = 2, n^{\log_b a} = n^0 = 1$$

$$f(n) = \Theta(1) = \Theta(n^{\log_b a}), \text{则} T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$$

分治法实例

- 快速排序算法，对数组A[p..r]进行排序
  - 分解：数组A[p..r]被划分为子数组A[p..q-1]和A[q+1..r]， A[p..q-1]中的每个元素都小于等于A[q]，A[q+1..r]中的每个元素都大于等于A[q]，q在划分时确定
  - 解决：通过递归调用快速排序算法，对子数组A[q+1..r]和A[p..q-1]进行排序
  - 合并：由于子数组的排序为原地排序，解的合并不需要操作，整个数组已经排好序

快速排序算法

```
QUICKSORT(A,p,r)
        if p<r
           then   q=PARTITIION(A,p,r)
                  QUICKSORT(A,p,q-1)
                  QUICKSORT(A,q+1,r)
```

QUICKSORT(A,1,length[A])

快速排序算法

PARTITION(A, p, r)

$x \leftarrow A[r]$

$i \leftarrow p$-1

for $j \leftarrow p$ to r-1

do if $A[j] \leq x$

then $i \leftarrow i+1$

exchange $A[i] \leftrightarrow A[j]$

exchange $A[i+1] \leftrightarrow A[r]$

return i+1

PARTITION过程

- 以 2 8 7 1 3 5 6 4 为例
- 以 6 10 13 5 8 3 2 11为例

PARTITION(A, p, r)

$x \leftarrow A[r]$

$i \leftarrow p$-1

for $j \leftarrow p$ to r-1

do if $A[j] \leq x$

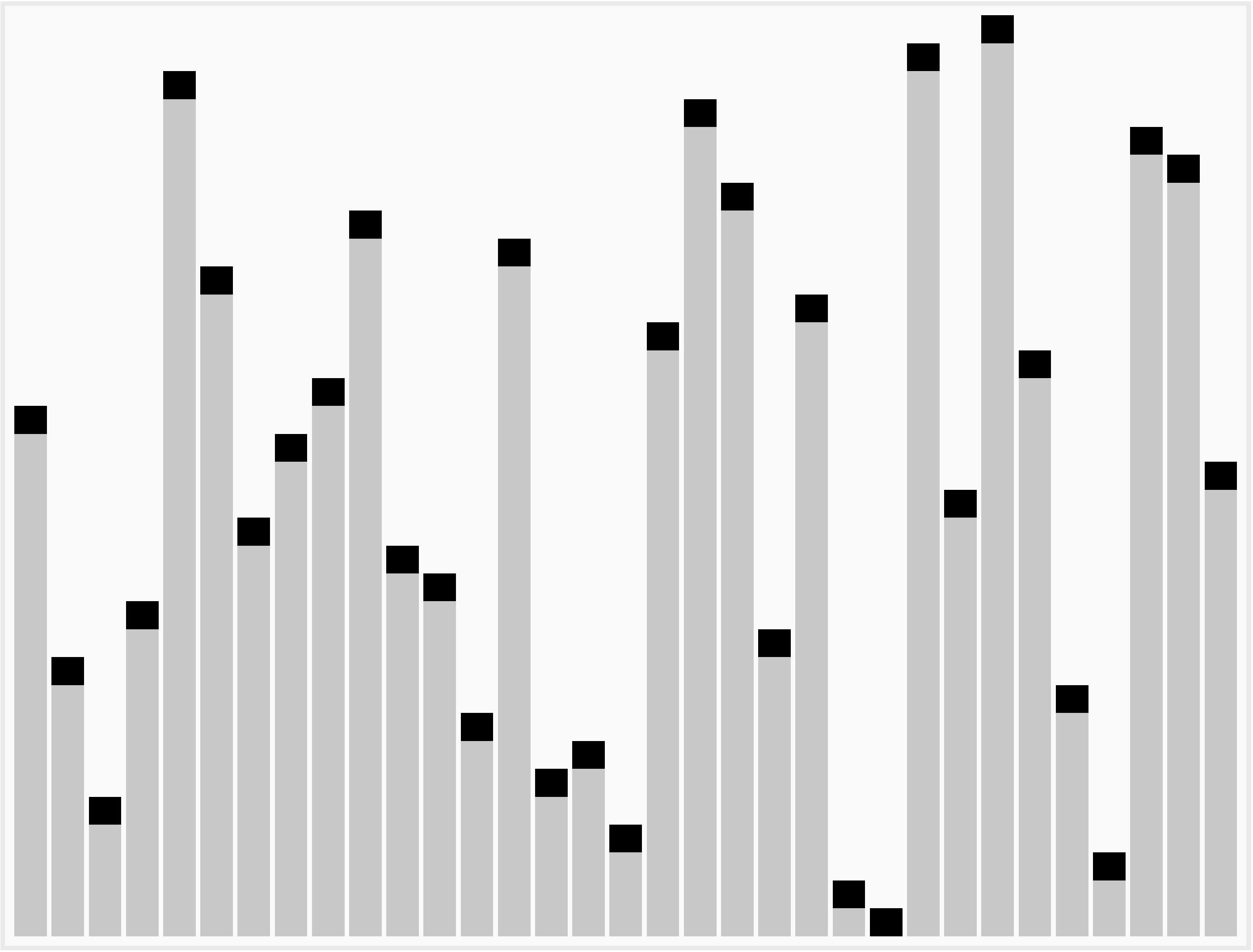
then $i \leftarrow i+1$

exchange $A[i] \leftrightarrow A[j]$

exchange $A[i+1] \leftrightarrow A[r]$

return $i+1$

快速排序算法



快速排序算法的分析

- 最坏情况时间复杂度$\Theta(n^2)$
- 平均情况时间复杂度$\Theta(n\lg n)$