南京大学
NANJING UNIVERSITY

算法设计与分析基础《Introduction to the Design and Analysis of Algorithms》

算法设计思想

南京大学软件学院
李传艺
lcy@nju.edu.cn
费彝民楼917

目录

蛮力法

- 说明
  - 是一种简单直接地解决问题的方法，直接基于问题的描述和涉及的概念定义
    - 常用于手工处理小规模例子以寻求对问题的认识和理解
  - 常用“直接做吧”来描述蛮力算法

- 最简单的设计策略就是没有策略
  - 例如，计算a^n、求数组最大元素
  - 其依赖的关键技术是“扫描”
    - 依次处理每个元素——遍历

- 不容忽视的算法设计策略
  - 唯一一个能够应用于所有问题的策略
  - 肯定会给出一种对应的解，不受数据规模的限制
  - 可能比设计一个好的算法需要更少的时间
  - 给出一个问题解的时间复杂度上界，衡量其它算法的时间效率

选择排序

- 每次遍历整个列表，选择一个最小的与第一个没有被交换的进行交换，下一次从被交换的下一个元素开始遍历。
- 算法伪代码

Algorithm SelectionSort($A[0\ldots n-1]$)
//用选择排序堆给定的数组进行排序
//输入：需要排序的数组
//输出：升序排序的数组
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$
            $min \leftarrow j$
    swap $A[i]$ **and** $A[min]$

- 比较执行的次数

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

- 交换次数
- 最好最坏情况？

冒泡排序

- 从第一个元素开始遍历列表直到不确定位置的最后一个元素，如果相邻元素是逆序则交换位置；每一次遍历后，有一个元素被放到对应的位置上。
- 算法伪代码

Algorithm BubbleSort($A[0\ldots n-1]$)
//该算法用冒泡排序算法对给定的数组进行排序
//输入：一个可排序的数组
//输出：升序排列的数组
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow 0$ **to** $n-2-i$ **do**
    if $A[j+1] < A[j]$
      swap $A[j]$ **and** $A[j+1]$

- 比较次数

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

- 交换次数
  - 最坏情况：和比较次数相同
  - 最好情况？

例题

- 设计一个蛮力算法对给定的x计算下列多项式值

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

- 蛮力算法的效率应该属于$\Theta(n^2)$
- 如何改进为$\Theta(n)$

分治法（1）

- 生活中
  - 军事、政治
  - 大到国家管理，小到班级管理

- 操作过程
  - 将问题实例划分为同一个问题的几个较小的实例（最好拥有相同的规模）
  - 对这些较小的实例求解
  - 最后合并这些较小问题实例的解得到原问题的解

- 可能是最著名的通用算法设计策略：Divide and Conquer
- 能够使用分治策略的问题一般具有如下特质
  - 问题规模缩小到一定程度就很容易解决
    - 时间上、空间上
  - 问题能够划分为多个相互独立的子问题
    - 无公共子问题、最终合并

- 对顺序执行的算法没有非常理想的优化效果；但是对可并行执行的算法优化效果明显

分治法（2）

- 顺序算法

  $T_s(n) = a * T\left(\frac{n}{b}\right) + f(n)$ 分解为b份，其中a份需要求解，f(n)表示合并需要的时间----------（1）

- 并行算法

  $T_p(n) = T\left(\frac{n}{b}\right) + f(n)$ 完全是并行执行

通用分治递推式

- 主定理
  - 如果在式（1）中$f(n) \in \Theta(n^d)$，其中d≥ 0，则
  - $$T(n) \in \begin{cases} \Theta(n^d) & \text{当} a < b^d \\ \Theta(n^d \log(n)) & \text{当} a = b^d \\ \Theta(n^{\log_b a}) & \text{当} a > b^d \end{cases}$$

- 例如. 计算$a_0 + a_1 + \cdots + a_{n-1} = \left(a_0 + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1}\right) + \left(a_{\lfloor \frac{n}{2} \rfloor} + \cdots + a_{n-1}\right)$需要的加法次数
  - A(n) = 2A(n/2) + 1，a=2, b=2, d=0 => $a>b^d$ => A(n) $\in \Theta(n^{\log_b a})=\Theta(n)$

合并排序（1）

- 将一个需要排序的数组A[0..n-1]一分为两个子数组A[0..⌊n/2⌋-1]和A[⌊n/2⌋..n-1]，分别对两个子数组排序，最后将两个子数组合并
- 伪代码

Algorithm MergeSort(*A*[0…*n*-1])
//递归调用mergesort函数对数组合并排序
//输入：一个可排序的数组
//输出：升序排列的数组
if *n* > 1
  copy *A*[0..⌊*n*/2⌋-1] to *B*[0..⌊*n*/2⌋-1]
  copy *A*[⌊*n*/2⌋..*n*-1] to *C*[0..⌈*n*/2⌉-1]
  MergeSort(*B*[0..⌊*n*/2⌋-1])
  MergeSort(*C*[0..⌈*n*/2⌉-1])
  Merge(*B*, *C*, *A*)

Algorithm Merge(*B*[0.. *p*-1] , *C*[0.. *q*-1],A[0..*p*+*q*-1])
//将两个有序数组合并为一个有序数组
i ← 0,; *j* ← 0; *k*← 0
while *i* < *p* **and** *j* < *q* **do**
  if B[i] ≤ C[j] **then** A[k] ← B[i]; i ← i+1
  else A[k] ← C[j]; j ← j+1
  k ← k+1
if i = p **then** **copy** C[j..q-1] **to** A[k..p+q-1]
else **copy** B[i..p-1] **to** A[k..p+q-1]

合并排序（2）

- 时间复杂度
  - 难以直接计算
  - 使用通用分治递推式
  - 设$n = 2^k$
  - $C(n) = 2C\left(\frac{n}{2}\right) + C_{merge}(n), \quad C(1) = 0$
  - $C_{merge}(n)$比较次数最坏的情况是：
    - 除了最后一个元素其它每一个元素都是通过比较后存放到结果数组中的，即比较n-1次

- 最坏情况的递推关系
  - $C(n) = 2C\left(\frac{n}{2}\right) + n - 1, \quad C(1) = 0$
  - $a = 2, b = 2, d = 1 \Rightarrow \ a = b^d \Rightarrow C(n) \in \Theta(nlogn)$

- 检验是否正确？
  - $\frac{n}{2} * (2-1); \frac{n}{4} * (4-1); \ldots; \frac{n}{2^k} * (2^k - 1)$
  - $k * n - \left(2^{k-1} + \cdots + 2^{k-k}\right) = k * n - \left(2^k - 1\right) = n * log_2 n - n + 1$

快速排序（1）

- 合并排序是通过元素的位置对它们进行分组
- 快速排序则通过元素的值对它们进行分组

- 对所有元素进行分区，使得在下标s前的元素的值都小于等于下标为s的元素值，之后的则大于s位置的元素值
- 建立这样的分区后，下标为s的元素所在的位置与目标升序数组中的位置相同；接下来就是对前后两个分区的数据使用同样的方式进行分区；直到每一个分区都只有一个元素

- 使用递归的方式

Algorithm QuickSort($A[l\ldots r]$)
//用快速排序方法对子数组进行排序
//输入：一个可排序的子数组
//输出：非降序排列的子数组
if $l < r$ **then** $s \leftarrow$ Partition($A[l\ldots r]$) //s是分裂的位置
QuickSort($A[l\ldots s\text{-}1]$)
QuickSort($A[s\text{+}1\ldots r]$)

快速排序（2）

- Partition

Algorithm Partition($A[l \ldots r]$)

//以第一个元素为中轴，将子数组分区

//输入：原数组的一个子数组，通过下标定义区间

//输出：原数组中的分裂点

$p \leftarrow A[l]$; $i \leftarrow l$; $j \leftarrow r + 1$

repeat

  repeat $i \leftarrow i + 1$ **until** $A[i] \geq p$ //找到大于或等于中轴的准备交换

  repeat $j \leftarrow$ j -1 **until** $A[j] \leq p$ //找到小于或等于中轴的准备交换

  swa**p**($A[i], A[j]$)

until $i \geq j$

swap($A[i], A[j]$) //为什么要这一步？

swap($A[l], A[j]$)

return j

- 时间复杂度
  - 最好情况和最坏情况

快速排序（3）

- 最好情况
  - 选择的中轴恰好是每一个数组的中值点，则满足通用分治递推公式
  - $C_{best}(n) = 2C_{best}\left(\frac{n}{2}\right) + C_{paritition}, \quad C(1) = 0$
  - 最好的情况下partition时出现i=j，比较的次数是n（为什么？）即$C_{paritition} = n$
  - $C_{best}(n) = 2C_{best}\left(\frac{n}{2}\right) + n \Rightarrow C_{best}(n) \in \Theta(nlogn)$

- 最坏的情况
  - 分区并不能带来效率上的提升，即不满足递推公式，最不满足的情况是原来就是升序的
  - 每次只会从第一个位置将数组Partition
  - k次后剩下的元素为n-k，需要比较的次数是(n+1)-k+1；直到第n-1次比较3次后结束
  - $C_{worst}(n) = (n+1) + n + (n-1) + \ldots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2)$

- 平均情况

$$C_{avg}(n) = \frac{1}{n} * \sum_{s=0}^{n-1} (C_{avg}(s) + C_{avg}(n-1-s) + (n+1))$$

减治法

- 两个士兵给两匹马拔毛的故事
- 利用一个问题给定实例的解和同样问题较小实例的解之间的某种关系，将一个大规模的问题逐步化简为一个小规模的问题
- 关键是：建立与小规模问题之间的联系=>本质上是一种递推关系
- 有3种主要的缩小问题规模的方式
  - 减去一个常量，通常是1
  - 减去一个常量因子，通常为2
  - 减去一个可变的规模
- 和分治法之间的区别和联系？
  - 分治：是多个小问题，小问题之间的联系
  - 减治：还是一个小问题，小问题与原问题之间的联系

插入排序

- 减一法
  - 假设前n-1个元素已经排序好了，则如何将最后一个元素**插入**到其它元素中去→递推关系
  - 三种方式插入最后一个元素
    - 从左到右扫描，遇到第一个大于或等于的元素，将其插入前面
    - 从右向左扫描，遇到第一个小于或等于的元素，将其插入后面
    - 折半查找？

- 递归伪代码
- 迭代伪代码

Algorithm InsertionSort($A[0\ldots n-1]$)
//输入：长度为n的可排序数组
//输出：非降序排列的数组
for $i \leftarrow 0$ **to** $n-1$ **do**
    $v \leftarrow A[i]; j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$

- 最坏情况：原来是倒序的，每一个元素需要比较i-1次

例题

- N个士兵过又宽又深的河，没有桥，不用船就会牺牲；发现有2个小朋友在划船，但是这个船只能搭两个小朋友或1个士兵，如何过河？需要来回多少次？

- Shell排序
  - 不停地分组，按照设计好的间隔分组，递减的间隔（等价于组数在减少）
  - 组内插入排序

变治法

- 生活的秘密在于——用一个烦恼代替另一个烦恼 （史努比之父）

- 通过转换问题使得原问题更容易求解
  - 实例化简
    - 还是原来的问题，只是进行了一些中间操作，使得问题求解变得容易
  - 改变表现
    - 主要是改变使用的数据结构
  - 问题化简
    - 将给定的问题变换为一个新的问题，对新的问题求解

- 预排序
  - 一种变治的思想，现对输入列表进行排序，再求解原问题

- 检验数组中元素的唯一性
- 模式计算：找到出现次数最多的元素
- 查找问题

例题

- 有n个数字构成的一个数组和一个整数s，如何最快确定数组中是否存在两个数的和恰好等于s?
- 设计一个找英文单词变位词的算法。

时空权衡

- 空间换时间
- 计算机或非计算机领域都有十分普遍的应用，包括生活中
  - 在数学中经常会有数学公式的表
  - 图书馆查找图书所在位置的系统
  - 文字记录

- 算法设计中的表述
  - 对问题的部分或者全部的输入作预处理，然后对获得的额外信息进行存储，以加速后面问题的求解
  - “输入增强”
    - 计数排序
    - Boyer-Moore字符串匹配

- 其它技术
  - 简单的使用额外的空间实现更快、更方便的数据存储——“预构造”
    - 只涉及存储结构
    - 散列法Hash——HashMap等
    - B树索引
  - 动态规划
    - 减治法找到递推关系；记录上一次求解结果

计数排序

- 多次扫描列表，利用额外空间记录比每一个元素小的元素的个数，最后把原列表的元素复制到新数组对应下标地方
  - 扫描n次，第i次扫描n-i个元素
  - 时间复杂度是$O(n^2)$
  - 上额外的同样长度的计数空间
  - 非常不实用
- 使用额外空间存储额外获得的信息
- 特定场景：要排序的数组中元素来自一个已知的小区间，且有重复
  - 使用额外的空间记录每一个元素出现的次数；然后写入一个新的数组
  - 如果是一个键值对需要排序？需要记录元素的键
    - 分布计数

例题

- 设计一个只有一行语句的算法，对任意规模为n，且元素值是从1到n的不同整数的数组排序

- 写一个算法计算两个稀疏矩阵的乘积，一个是p*q，一个是q*r；
- 思考其与时空权衡思想的关系？

课后习题思考

- 使用多种排序方法对2^31个IPv4地址进行排序；只有1G内存空间可以使用
- 2^31*4byte=8G
- 共有IPv4 2^32个

总结

- 蛮力法
- 分治法
- 减治法
- 变治法
- 时空权衡

- 选择排序、冒泡排序、合并排序、快速排序、插入排序、折半排序、预排序、计数排序、分布排序

- 下节课
  - 蛮力之顺序查找、字符串匹配、最近对、凸包问题、穷举查找

谢 谢！