第2章 算法初步

2020年9月22日　　10:35

2.1 插入排序

1. idea

从左到右有序化，步骤如下:

初始时：第1个位置仅1个字符，当然有序，从第2位置开始（j=2）

每次 j（迭代步）：将A[j]插入到有序的A[1...j-1]中适当位置（如何做？）

直到 j=n，结束

2 .伪代码

```
InsertionSort(A)
//输入A[1...n]，输出有序A
{ for j = 2 to n do //不变性：A[1...j-1]已有序
     { key = A[j];
       i = j -1;     //i从j-1起向左遍历
       while i > 0 and A[i] > key do
       { A[i+1] = A[i];
         i = i - 1;
       }
       A[i+1] = key; //边界正确
     }
}
```

3.示例：Page 10 Fig 2.2

注：a.只使用一个额外的存储单元key

b.伪代码写法：类C

2.2 算法分析

1.算法分析可以

预测所需资源：计算时间，存储空间，有时也包括通讯带宽等

若有n个算法，可从中择优或择合适的

分析时，应考虑计算机体系结构和计算模型

2.计算模型

硬件->计算模型（Alg）->编程模型（实现）

（1）作用：简化分析成本

独立于硬件和程设语言

一个公正平等的评价标准

（2）RAM(Random Access Machine)模型：单处理器模型

指令：算术运算，数据移动，控制指令等，并假设每条指令均耗费1个单位时间

数据类型：整型，浮点型等

计算2^k耗费单位时间，内存存取为单位时间，且无限容量

不考虑Cache和disk之间差异

3.插入排序时间分析

（1）影响计算时间的因素

输入规模，输入数据分布

与数据结构相关

计算时间通常使用关于输入规模的函数表示，如：T(n)，n为规模

（2）运行时间的2种表示：基本操作数，执行语句（步）数

（3）运行时间的3种度量：

设Dn是问题规模为n的输入集，$I \in D_n$，P(I)为I出现概率，t(I)是算法在输入I时执行的基本操作数，则

Best Case Running Time: B(n)

$$B(n) = \min_{I \in D_n}\{t(I)\}$$

Worst Case Running Time: W(n)

$$W(n) = \max_{I \in Dn}\{t(I)\}$$

Average case Running Time: A(n)

$$A(n) = \sum_{I \in Dn} p(I)t(I)$$

（4）时间分析

see Page 14:

line 1 执行n次

...

$$T(n) = c_1\mathrm{n} + \mathrm{c}_2(\mathrm{n}-1) + \mathrm{c}_4(\mathrm{n}-1) + \mathrm{c}_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

这里t_j是while循环对于j值进行的循环次数

$B(n)$：对$\mathrm{j} = 2 \sim \mathrm{n}, \mathrm{t_j} = 1$

$$B(n) = T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_8) = an + b \ \ //线性的$$

$W(n)$：对$j = 2 \sim n, t_j = j$

$$W(n) = T(n) = an^2 + bn + c \ //n^2级$$

$A(n)$：对$\mathrm{j} = 2 \sim \mathrm{n}, \mathrm{t_j} = \frac{\mathrm{j}}{2}$

$$A(n) = T(n) = a'n^2 + b'n + c' \ //n^2级，但\mathrm{a}' < \mathrm{a}$$

注：平均时间较难分析，通常使用最坏时间，作为算法运行时间的上界

运行时间函数的增长率（阶）：

忽略c_j(视$\mathrm{c_j} = 1$)：$\mathrm{W(n) = an^2 + bn + c}$

忽略低阶项：$W(n) = an^2$

忽略高阶项的系数：$W(n) = O(n^2)$

2.3 算法设计

1.算法设计技术

（1）Divide and Conquer （ch2,7,9,12,28,30）

（2）Greedy Strategy （ch16,23）

（3）Dymanic Programming （ch15,25）

（4）Linear Programming （ch29）

（5）Backtracking (Additional)

（6）Branch and Bound (Additional)

（7）Others （ch28,31,32）

2.分治法及归并排序示例

（1）idea

原问题划分成若干个更小的子问题

要求：子问题求解的方法同原问题（可以递归）

（2）求解步骤

Divide：将当前问题划分成若干个字问题

Conquer：递归解子问题

Combine：将子问题的解合成原问题的解

(3) 示例：MergeSort

主过程：

```
MergeSort(A, p, r) {                    //A[1…p…r…n]
    if p < r then {
        q = (p + r) / 2; //向下取整，划分点
        MergeSort(A, p, q); //第1个子问题递归排序，使A[p…q]有序
        MergeSort(A, q+1, r); //第2个子问题递归排序，使A[q+1…r]有序
        Merge(A, p, q, r); //将有序A[p..q], A[q+1…r]合并为有序A[p…r]
    }
}
```

子过程：

```
Merge(A, p, q, r) {
    n1 = q - p + 1;
    n2 = r - q;
    for i = 1 to n1 do
        B[i] = A[p + i - 1]; //将A数组前部分数放入B数组中
    for i = 1 to n2 do
        C[i] = A[q + i]; //C[1…n2]←A[q+1…r]
    B[n1 + 1] = +∞; C[n2 + 1] = +∞; //哨兵
    i = 1; j = 1; //i, j分别指向B、C第一个元素
    for k = p to r do {
        if B[i] <= C[j] then
            A[k] = B[i++];
        else
            A[k] = C[j++];
    }
}
```

注：a. $Merge$时间是$O(n)$

b. 计算过程：Fig 2.4

c. 整个算法时间：$\mathrm{T}(n)=\begin{cases} O(1),\ n=1 \\ 2T\left(\frac{n}{2}\right)+O(n),\ n>1 \end{cases}$

求解：$T(n)=O(nlog\,n)$

d. 快速排序优于归并排序的原因是，快速排序能更好的利用程序的局部性原理，Cache命中率更高