算法分析举例

乔海燕
中山大学数据科学与计算机学院

2017 年 9 月 22 日

摘 要

本文给出几个算法复杂度估算的例子。

§1 时间复杂度定义

一个算法的时间复杂度指算法运行所化的时间。例如，求一个数组中的最大元素算法1

算法 1 ArrayMax(A, n)

输入: A是长度为$n > 0$的整数数组
输出: 返回A中的最大元素

```
1: currentMax ← A[0]
2: for i = 1 to n − 1 do
3:    if currentMax < A[i] then
4:       currentMax ← A[i]
5:    end if
6: end for
7: return currentMax
```

显然，算法ArrayMax(A,n)运行的时间长度与输入的规模n有关。常用$T(n)$表示一个算法的运行时间，它通常是n的函数。

事后估算

我们可以使用函数clock()，对于不同大小的数组A测量算法的运行时间(见程序清单1)。

程序清单 1: 事后统计

```
int ArrayMax(int A[], int n){
    int currentMax;
    currentMax = A[0];
    for (int i=1; i< n-1; i++)
        if (currentMax < A[i])
```

```
            currentMax = A[i];
    return currentMax;
}

int main(int argc, char* argv[]){
    if (argc < 2){
      cout <<"useage: ArrayMax n"<<endl;
      exit(1);
    }
    int n = atoi(argv[1]);
    int *A = new int[n];
    for (int i=0; i<n; i++)
       A[i] = rand();
    unsigned start, end;

    start = clock();
    int m = ArrayMax(A,n);
    end = clock();

    cout <<"Timing for ArrayMax(A,n) for n = " << n << ": "
         << (double)(end-start)/CLOCKS_PER_SEC <<endl;
    delete [] A;
    return 0;
}
```

由此得到运行时间表1。从表中看出，运行时间大致随规模线性增长。但是，这样的统计结果显然依赖于运行的机器和使用的编译器等多种因素。

n	10000000	20000000	40000000	50000000	80000000	100000000
$T(n)$(s)	0.016	0.031	0.047	0.062	0.094	0.11

表 1: 算法ArrayMax(A,n)运行时间的实验统计

事前估算

一个算法的时间复杂度可以在某种理想计算模型下运行的步数来表示。设想一个计算机模型，一步可以运行一个基本指令，如赋值、比较和返回结果等。那么一个算法的运行时间定义为算法在该模型下从开始到结束的步数。仍然用$T(n)$表示。例如，对于算法1，第一条指令（第1行）运行一次，计一步，接下来的循环运行$n-1$次，每次循环体（3-5行）运行计一步，最后的返回指令（第7行）计一步，因此，总的步数为

$$T(n) = 1 + n - 1 + 1 = n + 1$$

算法的时间复杂度最用用大O记号表示，在这里为$T(n) = O(n)$。

如果将循环体（3-5）行的计算细化得到什么结果呢？假如条件语句中比

较算一步，赋值也算一步，则条件语句在条件成立时计两步，条件不成立时计一步，因此有

$$T(n) \leq 1 + 2(n-1) + 1 = 2n$$

因此，$T(n) = O(n)$。也说算法ArrayMax的最坏情况复杂度为$O(n)$。

另外，

$$T(n) \geq 1 + (n-1) + 1 = n + 1$$

因此，$T(n) = \Omega(n)$。也称算法的最好情况复杂度为线性的。

因为$T(n)$上下界的增长率一样，此时可以记ArrayMax的时间复杂度为$T(n) = \Theta(n)$，称其时间复杂度是线性的。

§2 复杂度估算举例

顺序查找的时间复杂度

算法 2 SequentialSearch(A,n, v)

输入: A是长度为$n > 0$的整数数组

输出: 如果A中存在一个元素等于v，则返回其下标，否则返回-1。

```
1: for i = 0 to n − 1 do
2:   if A[i] = v then
3:     return i
4:   end if
5: end for
6: return -1
```

对于顺序查找算法SequentialSearch，复杂度可以用比较次数表示。查找分成功和失败两种情况。显然，查找失败总是比较n次，所以查找失败情况的复杂度为$O(n)$。

对于查找成功，最好情况是比较一次查找成功。最差情况是比较n次，所以，最坏情况时间复杂度为$T(n) = O(n)$。

查找成功有n种情况，即分别比较i次成功，$i = 1, 2, \cdots, n$。如果用p_i表示比较i次成功的概率，$C_i = i$表示这种情况下的比较次数，则查找成功的平均比较次数为

$$T(n) = \Sigma_{i=1}^{n} p_i C_i$$

假设每种情况出现的概率是相同的，即$p_i = 1/n$, 则

$$T(n) = \frac{\Sigma_{i=1}^{n} C_i}{n} = \frac{1 + 2 + \cdots + n}{n} = (n+1)/2$$

因此，在这样的假设下，平均时间复杂度为$T(n) = \Theta(n)$。

插入排序的时间复杂度

首先编写一趟插入的算法3。

插入排序算法4通过调用Insert(A, i)($i = 1, 2, \cdots, n-1$)$n-1$次完成排序。

算法 3 Insert(A, k)

输入： A是一个可比较大小的数组，长度为$n \geq 0$。$1 < k < n$, $A[0..k-1]$递增有序。

输出： 将$A[k]$插入$A[0..k-1]$适当位置，使得$A[0..k]$从小到大有序。

```
if A[k] < A[k-1] then
    current ← A[k]
    j ← k-1
    while j ≥ 0 and current < A[j] do
        A[j+1] ← A[j] //将A[j]后移
        j ← j-1
    end while
    A[j+1] ← current //A[k]的最终位置
end if
```

算法 4 Sort(A,n)

输入： A是一个可比较大小的数组，长度为$n \geq 0$。

输出： A中元素按从小到大有序

```
for k = 1 to n-1 do
    Insert(A,k) //将A[k]插入A[0..k-1]使得A[0..k]有序
end for
```

我们用算法中的元素比较和元素赋值（或称移动元素）次数表示排序算法的复杂度。

首先分析插入算法3，Insert(A,k)的复杂度。

最好情况下，插入进行一次比较，没有赋值。最坏情况需要进行k次比较和$k+2$次赋值（while循环前后各一次），即比较和赋值总次数

$$T_{Insert}(n,k) = 2k+2$$

由此得到插入排序算法4，Sort(A, n)的最好情况$T_{Sort}(n) = \Omega(n)$。最坏情况下

$$T_{Sort}(n) = \Sigma_{k=1}^{n-1}(2k+2) = O(n^2)$$

对于平均情况，仍然先看插入的平均情况。

将$A[k]$插入$A[0..k-1]$的可能位置有$0, 1, \cdots, k$，共$k+1$种情况，比较次数分别为$k, k, k-1, \cdots, 2, 1$, 赋值次数分别为$k+2, k+1, \cdots, 2, 0$。假定每种情况出现的概率相同，即均为$p_i = 1/(k+1)$, 则平均比较和赋值次数为

$$T_{Insert}(n,k) = \frac{1}{k+1}((k+k+(k-1)+\cdots+2+1)+((k+2)+(k+1)+\cdots+2+0)) = \frac{k^2+4k+2}{k+1}$$

因此，插入算法的平均时间复杂度为

$$T_{Sort}(n) = \Sigma_{k=1}^{n-1}T_{Insert}(n,k) \leq \Sigma_{k=1}^{n-1}(2(k+1)) = n^2+n-2$$

所以，插入排序的平均时间复杂度为$T_{Sort}(n) = O(n^2)$。

鸣谢： 感谢16级计算机专业同学们给予老师的激励，并指出文中的错漏问题!

参考文献

[1] 严蔚敏、吴伟民，数据结构，清华大学出版社，1997。

[2] Robert L. Kruse, Alexander J. Ryba. *Data Structures and Program Design in C++*, Higher Education Press, 2001.

[3] Michael T. Goodrich, Roberto Tamassia. *Algorithm Design and Applications*, Wiley, 2014.