计算机算法设计与分析
The Design and Analysis of Computer Algorithms

2024年秋季

关于计算机算法

- 计算机科学 是一种创造性思维活动，其教育必须面向设计；
- 计算机算法 是任何定义好了的计算程式（或叫步骤），它取某些值或值的集合作为输入，并产生某些值或值的集合作为输出。
- 计算机算法的特征
  - 数据输入与输出；
  - 确定性： 算法的每一步计算（包括判断）必须要有确切的定义，即每一步计算动作必须是清楚的，无二义性。
  - 可实现性：每种计算至少在原理上能由人用纸和笔在有限的时间内完成。
  - 有穷性： 一个算法总是在执行了有穷步之后终止。

关于算法设计与分析

- 设计算法：求解问题的数学模型、经典算法、分析和改进。
- 表示算法：输入数据结构、描述算法的语言、算法流程、逻辑结构。**C++** 或 **ALGEN**（伪代码）。
- 分析算法：发现算法特性、估计对各种应用的适合程度、相同应用中与其它算法的比较。确定输入数据模型、确定每项基本操作所用时间及空间占用情况，统计数据、分析推求。
- 测试算法：正确性、有穷性；
  白盒法、黑盒法。

教 学 要 求

- 教学内容
- 复杂性分析初步
- 图与遍历算法
- 分治算法
- 贪心算法
- 动态规划算法
- 回溯算法
- 分枝定界算法
- 算法复杂性理论
- 近似算法设计与分析
- 随机算法
- 考核
- **60学时/3学分，**
- **平时20%，期末80%**

- 参考文献
- **Horowwitz E. 等著，冯博琴等译《计算机算法》(C++版)，机械工业出版社，2006。**
- **Alfred V. Aho et al，《The Design and Analysis of Computer Algorithms》影印版，机械工业出版社，2006。**
- **Cormen T.H. et al，《Introduction to Algorithms》，影印版，高等教育出版社，2001。**
- 计算机算法基础，余祥宣，华中理工大学出版社，2005。
- 算法设计与分析，曲婉霞，清华大学出版社，2011。
- 现代优化计算方法，邢文训，清华大学出版社，2000。
- 联系： yfchen@ucas.ac.cn

第一章
复杂性分析初步

- 计算复杂性的本质是研究完善的模型，在这样的模型范围内可以评估重要问题的固有困难，并得以进一步研究“有效的”算法。对于许多重要问题，已知算法的复杂度在最坏情形下的渐近下界和上界之间存在显著的差距。计算复杂性研究能够在探索这些问题的新算法方面提供指导。
- 计算复杂性体现在算法占用机器空间资源和时间资源的情况，是关于选定模型下输入数据规模的函数。起决定作用的是这些函数当输入数据规模趋于无穷大时的渐进性态。
- 一种输入数据模型的选用可能使复杂性分析更简单，而另一种输入模型可能更好地反映程序将被使用的情况。

空间复杂性

- 考虑空间复杂性的理由

  在多用户系统中运行时，需指明分配给该程序的内存大小；
  想预先知道计算机系统是否有足够的内存来运行该程序；
  用空间复杂性来估计一个程序可能解决的问题的最大规模；
  一个问题可能有若干个不同的内存需求解决方案，从中择优。

指令空间: 存储编译后的程序指令

- 指令空间的大小取决于如下因素：
- 把程序编译成机器代码的编译器，编译器不同，则产生的机器代码的长度就会有差异；
- 编译时实际采用的编译器选项，如优化模式、覆盖模式，选用优化模式可缩短代码长度，但会增加运行时间；
- 目标计算机的配置，如带有浮点处理硬件的，每个浮点操作转换为一条机器指令，否则，必须生成仿真的浮点计算代码，使整个机器代码加长。

数据空间：存储所有常量和变量的值

存储常量和简单变量：所需的空间取决于所使用的计算机和编译器, 以及变量与常量的数目；

存储复合变量：包括数据结构所需的空间及动态分配的空间；

计算方法：复合变量所占空间等于各个成员所占空间的累加。

例如：数组变量所占空间等于数组大小乘以单个数组元素所占的空间。

double a[100]; 所需空间为**100×8＝800**

int matrix[r][c]; 所需空间为 **4×r×c**

Borland C++基本数据类型（32位字长机器）

类型名	说明	字节数	使用范围
char	字符型	**1**	**-128~127**
unsigned char	无符号字符型	**1**	**0~255**
pointer	指针型	**2**	
short [int]	短整型	**2**	**-32768~32767**
signed short [int]	有符号短整型	**2**	**-32768~32767**
unsigned short [int]	无符号短整型	**2**	**0~65535**
int	整型	**4**	**-2147483648~ 2147483647**
signed [int]	有符号整型	**4**	**-2147483648~ 2147483647**
unsigned [int]	无符号整型	**4**	**0~4294967295**
long [int]	长整型	**4**	**-2147483648~ 2147483647**
signed long [int]	有符号长整型	**4**	**-2147483648~ 2147483647**
unsigned long [int]	无符号长整型	**4**	**0~4294967295**
float	单精度浮点型	**4**	**±3.4E ±38**
double	双精度浮点型	**8**	**±1.7E ±308**
long double	长双精度浮点型	**16**	**-3.4E-4932~ 1.1E+4932**

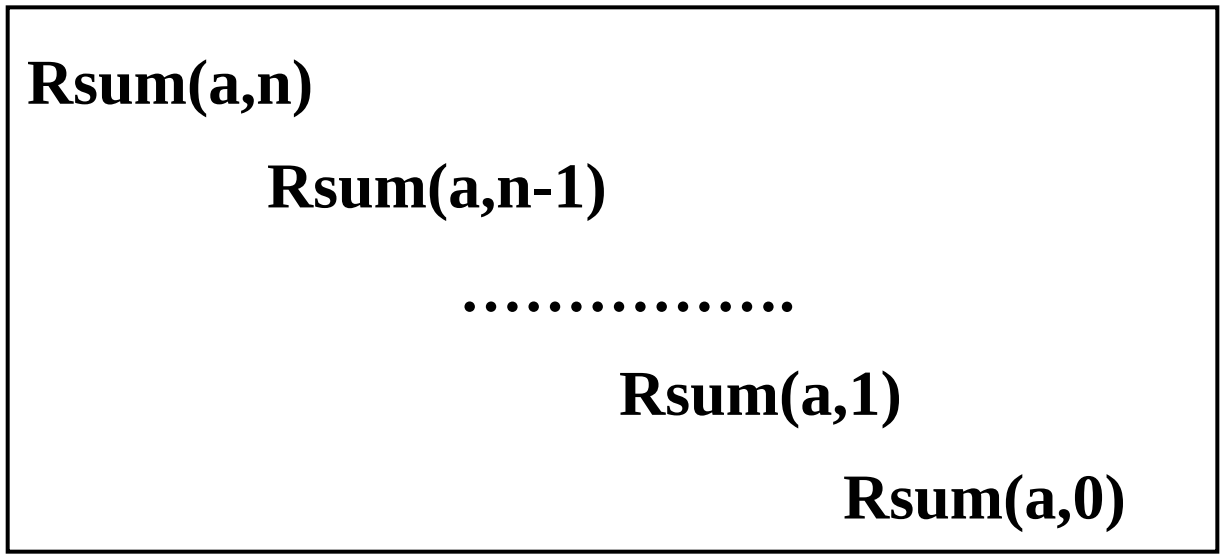
环境栈空间：保存函数调用返回时恢复运行所需要的信息

返回地址；

所有局部变量的值；

递归函数的传值形式参数的值；

所有引用参数及常量引用参数的定义。



数值求和的例子

```
template<class T>
  T Sum( T,a[ ], int n )
  {//计算 a[0:n-1]的和
     T tsum=0;
      for(int i=0, i<n; i++)
          tsum+=a[i];
      return tsum;
   }
```

存数组的地址a,

变量n, i, tsum:

S_c=2+4+4+sizeof(T)

```
template<class T>
 T Rsum( T a[ ], int n)
{//计算a[0:n-1]的和
       if (n>0)
        return Rsum(a,n-1)+a[n-1];
       return 0;
     }
```

保留a的地址，函数返回地址，

存储变量n , 递归深度为 n+1

S_v=(2+2+4)(n+1)

几点说明

- 指令空间的大小对于所解决的问题不够敏感；
- 常量和简单变量所需的数据空间也常常独立于所解决的问题，除非被处理的相关数的大小对于所选的数据类型来说实在太大；
- 定长复合变量和未使用递归函数所需要的环境栈空间都独立于问题的规模；
- 递归函数所需要的环境栈空间主要依赖于局部变量及形式参数所需要的空间，还依赖于递归的深度。

时间复杂性

- 考虑时间复杂性的理由

- 某些计算机用户需要提供程序运行时间的上限（用户可接受的）；
- 所开发的程序需要提供一个满意的实时反应；
- 把握问题求解的难易程度；
- 比较算法的优劣，改进算法。

时间复杂度的构成

程序所占时间 T_p=编辑时间+运行时间 (编译时间与实例特征无关，主要关注运行时间)；

基本操作（运算）：加、减、乘、除、比较（有时还包括读、写），

$$T_p=T_c+ c_a\cdot ADD(n) + c_s\cdot SUB(n) + c_m\cdot MUL(n)+ c_d\cdot DIV(n) + c_c\cdot CMP(n) + \cdots$$

估算运行时间的方法：关键操作计数、总的执行步统计。

约定：每种基本操作所用时间都是一个单位。

复杂性函数：f (n)，当它以某一个多项式函数 p (n) 为上界时，称为好算法（或叫有效算法）。

关键操作计数：选择关键操作，统计次数

- **寻找数组中最大元素**

```
template<class T>
int Max(T a[], int n)
{//寻找a[0:n-1]中的最大元素
    int pos=0;
    for (int i=1; i<n; i++)
      if (a[pos]<a[i])
        pos=i;
    return pos;
}
```

- **这里的关键操作是比较。for循环中共进行了n-1次比较。**

- n次多项式求值程序

```
template<class T>
 T PolyEval(T coeff[], int n, const T& x)
 {//计算n次多项式的值，coeff[0:n]为多项式的系数
    T  y=1, value=coeff[0];
    for (int i=1; i<=n; i++)    //n循环
   {//累加下一项
      y*=x;                    //一次乘法
      value+=y*coeff[i];     //一次加法和一次乘法
    }
    return value;
}                              //3n次基本运算
```

- 这里的关键操作是数的加法与乘法。

- 利用**Horner**法则求多项式的值
- **template<class T>**
- **T Horner(T coeff[], int n, const T& x)**
- **{//**计算**n**次多项式的值，**coeff[0:n]**为多项式的系数
- **T value=coeff[n];**
- **for(i=1; i<=n; i++) //n**循环
- **T value＝value*x+coeff[n-i]; //**一次加法和一次乘法
- **return value;**
- **} //2n**次基本运算

- **Horner**法则：
- $P(x)=(\cdots(c_nx+c_{n-1})x+c_{n-2})x+c_{n-3})x+\cdots)x+c_0$

计算名次

```
template<class T>
void Rank(T a[ ], int n, int r[ ])
{//计算a[0:n-1]中元素的排名
   for(int i=0; i<n; i++)
      r[i]=0;  //初始化
     //逐对比较所有的元素
    for(int i=1; i<n; i++)
    for(int j=0; j<i; j++)
     if(a[j]<=a[i]) r[i]++;
     else r[j]++;
}
```

选择排序

```
template<class T>
void Selectionsort(T a[], int n)
{//对数组a[0:n-1]中元素排序
    for(int k=n; k>1; k--)
     { int j=Max(a,k);
        Swap(a[j],a[k-1]);
        }
  }
```

```
 template<class T>
  void Swap(T& a. T& b)
 {T temp=a; a=b; b=temp; }
```

这里的关键操作是比较，次数为：**1+2+…+n-1=n(n-1)/2**

- 冒泡排序
- template<class T>
- void BubbleSort (T a[ ], int n)
- {
- for(int i=n; i>0; i--)
- Bubble (a, i);
- }
- 最坏情形下， 关键操作数：∑ 4(i-1)=2n(n-1)
- 一次冒泡
- template<class T>
- void Bubble (T a[ ], int n)
- {
- for (int i=0; i<n-1; i++)
- if (a[i]>a[i+1]) Swap(a[i],a[i+1]); 一次比较，3次赋值
- }

平均时间复杂度

数据分布不同可能会影响实际操作数。程序 **P**

- **P**在最好情况下的操作计数为

$$O_P^{BC}(n_1, n_2, \cdots, n_k) = \min\{operation_P(I) \mid I \in S(n_1, n_2, \cdots, n_k)\}$$

- **P**在最坏情况下的操作计数为

$$O_P^{WC}(n_1, n_2, \cdots, n_k) = \max\{operation_P(I) \mid I \in S(n_1, n_2, \cdots, n_k)\}$$

- **P**的平均的操作计数为

$$O_P^{AVG}(n_1, n_2, \cdots, n_k) = \frac{1}{|S(n_1, n_2, \cdots, n_k)|} \sum_{I \in S(n_1, n_2, \cdots, n_k)} operation_P(I)$$

- **P**的期望的操作计数为

$$O_P^{EVG}(n_1, n_2, \cdots, n_k) = \sum_{I \in S(n_1, n_2, \cdots, n_k)} p(I) \cdot operation_P(I)$$

- 其中，$p(I)$是实例 I 出现的概率。

- 顺序查找
- template < class T >
- int SeqSearch (T a[ ], const T& x, int n)
- { //在a[0:n-1]中搜索**x,**若找到则返回所在的位置，
- 否则返回**-1**
- int i;
- for (i=0; i<n && a[i]!= x; i++);
- if (i= =n) return -1;
- return i;
- }
- **最好情况：比较 1 次；**
- **最坏情况：比较 n 次；**
- **平均比较次数：** $\frac{1}{n+1}\left(n+\sum_{i=1}^{n} i\right)=\frac{n}{2}+\frac{n}{n+1}$

向递增数组插入元素

```
template<class T>
void Insert(T a[], int& n, const T& x)
    n)
{//向数组a[0:n-1]中插入元素x
  //假定a的大小超过n
  int i;
  for(i=n-1; i>=0 && x<a[i]; i--)
    a[i+1]=a[i];
    a[i+1]=x;
 }
```

x被插入数组的可能位置有

n+1个：(i-1, i), i=0,1,…,n

平均比较次数为：

插入排序

```
template<class T>
void InsertionSort(T a[], int

{//对a[0:n-1]进行排序
    for(int i=1; i<n; i++) {
       T t=a[i];
   Insert(a, i, t);
    }
 }
```

如何计算插入排序的平均时间复杂度？

$$\frac{1}{n+1}\left(n+\sum_{i=0}^{n-1}(n-i)\right)=\frac{1}{n+1}\left(n+\sum_{j=1}^{n}j\right)=\frac{n}{2}+\frac{n}{n+1}$$

统计执行步数：按程序步、执行语句统计

- 程序步：独立程序片段，执行时间与实例特征无关；
- 执行语句：一个表意完整的程序语句；
- **s/e**：每个语句中包含的执行程序步数；
- 频率：语句出现的次数；
- 总执行步数＝$\Sigma\omega_i k_i$

语句	s/e	频率	执行步数
Void Addm(T **a, …)	0	0	0
{	0	0	0
for(int i=0; i<rows; i++)	1	rows+1	rows+1
for(int j=0; j<cols; j++)	1	rows×(cols+1)	rows×cols+rows
c[i][j]=a[i][j]+b[i][j];	1	rows×cols	rows×cols
}	0	0	0

- 执行步数总计 2×rows×cols+2×rows+1

- 使用计数器矩阵加法与执行步数统计
- **template<class T>**
- **void Addm(T **a, T **b, T **c, int rows, int cols)**
- **{//矩阵a和b相加得到矩阵c**
- **for(int i=0; i<rows; i++){**
- **count++; //对应于上一条for语句 (共执行了rows步)**
- **for(int j=0; j<cols; j++){**
- **count++; //对应于上一条for语句 (共执行了rows×cols步)**
- **c[i][j]=a[i][j]+b[i][j];**
- **count++; //对应于赋值语句 (共执行了rows×cols步)**
- **}**
- **count++; //对应j的最后一次for循环 (共执行了rows步)**
- **}**
- **count++；//对应i的最后一次for循环 (共执行了1步)**
- **}**

- 总的执行步数为： **2×rows×cols＋2×rows＋1**

渐 近 函 数

- 计算复杂性反映程序运行时间和占用空间随实例特征（如输入规模）变化的情况；
- 复杂性函数 T(n) 或 S(n) 非负、递增；
- 函数 f (n)的渐进函数 g(n): 当n→∞时

  (f(n) – g(n))/f(n) → 0
- 非负函数 f(n) 与 g(n)具有相同的阶（作为无穷大量来看）：
- 当n→∞时，f (n)/g(n) → c (正实数）
- 非负函数 f(n) 的主项 p(n)：p_i (n)/p(n) → 0

  其中 f (n) = p(n) + p_1(n) + ⋯ + p_k(n)

  比如，多项式函数： $f(n)=a_k n^k+a_{k-1}n^{k-1}+\cdots+a_1 n+a_0$

  主项为 $a_k n^k$
- 一般取主项作为渐进函数

各阶复杂性函数对应的机器运行时间比较

T(n)	n=10	n=20	n=30	n=40	n=50	n=60
n	0.00001’s	0.00002’s	0.00003’s	0.00004’s	0.00005’s	0.00006’s
n^2	0.0001’s	0.0004’s	0.0009’s	0.0016’s	0.0025’s	0.0036’s
n^3	0.001’s	0.008’s	0.027’s	0.064’s	0.125’s	0.216’s
n^5	0.1’s	3.2’s	0.405’c	1.707’c	5.208’c	0.216’h
2^n	0.01024’s	1.0486’s	17.896’c	12.726’d	35.7’y	365.6’Dc
3^n	0.059’s	58.12’c	6.53’y	3855.2’Dc	227.644 亿	…….

渐近上界：f(n) = O(g(n))

- 定义：存在正实数 c 和正整数 N, 使得当n>N时，

  $$f(n) \le c \cdot g(n)$$

- 例子：
- const = O(1); $3n+1 = O(n)$; $10n^2+4n+3 = O(n^2)$;
- $6 \cdot 2^n + n^5 = O(2^n)$; $n \cdot \log n + n^2 = O(n^2)$;
- $3n + 2 = O(n^2)$
- 注：1. 记号f(n)=O(g(n))不能写成g(n)=O(f(n));
- 2. 最后一个是松散的上界，最好给出最小渐进上界。
- 大欧比率定理：如果极限 lim(f(n)/g(n))存在，则
- f(n) = O(g(n)) 的充要条件是：存在正实数 c , 使得
- $\lim (f(n)/g(n)) \le c$.

常用估计渐近上界的几个不等式

定理：对于任意正实数 χ 、ε 和常数 c ，下面的不等式成立：

1. 存在正整数 N, 使得对于任何n>N, 有 $(c + \log n)^{\chi} < (\log n)^{\chi+\varepsilon}$;
2. 存在正整数 N, 使得对于任何n>N, 有 $(\log n)^{\chi} < n^{\varepsilon}$;
3. 存在正整数 N, 使得对于任何n>N, 有 $(c + n)^{\chi} < n^{\chi+\varepsilon}$;
4. 存在正整数 N, 使得对于任何n>N, 有 $n^{\chi} < (1+ \varepsilon)^{n}$;
5. 组合：对于常数 γ ，当 n 充分大时，有 $n^{\chi}(\log n)^{\gamma} < n^{\chi+\varepsilon}$ 。

例子 $n^3 + n^2 + \log^{10} n = O(n^3)$; $n^4 + n^{2.5} \log^{20} n = O(n^4)$;

$2^n n^4 \log^3 n + 2^n n^5 / \log^3 n = O(2^n n^5)$;

因为 $(10n^2 + 4n + 2)/n^2 \to 10$ ，所以 $(10n^2 + 4n + 2)/n^2 = O(n^2)$

因为 $(6 \cdot 2^n + 4 n^2)/2^n \to 6$ ，所以 $6 \cdot 2^n + 4 n^2 = O(2^n)$

因为 $(n^{16} + 3 n^2)/2^n \to 0$ ，所以 $n^{16} + 3 n^2 = O(2^n)$

渐近下界: $f(n) = \Omega(g(n))$

- 定义：存在正实数 c 和正整数 N, 使得当n>N时，
  $$f(n) \geq c \cdot g(n)$$
- 大欧米茄比率定理：如果极限 lim(f(n)/g(n))存在，则
- $f(n) = \Omega(g(n))$ 的充要条件是：存在正实数 c , 使得
- $\lim(f(n)/g(n)) \geq c$.
- 注：容易发现一个渐近下界，但人们更关注的是找到
- 最大的渐进下界。
- 渐近同阶： $f(n) = \Theta(g(n))$ ，指函数g(n)既是f (n)的渐近上界，又是f (n)的渐近下界。此时，如果极限 lim(f(n)/g(n))存在的话，应该是一个正实数。
- 例子 $3n+2= \Theta(n)$; $1.5n^2-4n+100 = \Theta(n^2)$;
  $5 \cdot 2^n + 3n \log n = \Theta(2^n)$;
- 一般地，函数主项即是该函数的渐进同阶。

折半搜索

```
template<class T>
 int BinarySearch(T a[], const T& x, int n)
 {//在a[0]<=a[1]<=…<=a[n-1]中搜索x
   //如果找到，则返回所在位置，否则返回 –1
   int left = 0; int right = n-1;
   while(left <= right){
     int middle = (left+right)/2;
     if(x = = a[middle]) return middle;
     if(x > a[middle]) left = middle+1;
     else right = middle – 1;
   }
   return –1; //未找到x
}
```

每次while循环 执行的操作数是固定的常数，而循环的次数为 log n , 所以，总的操作数以log n 为渐近上界。

1，2 阶递归方程求解

- 1 阶递归方程

  $T(n)=aT(n-1)+b,\ \ T(1)=c$:

  $T(n)=a(aT(n-2)+b)+b=a^2T(n-2)+ab+b$

  $=a^{n-1}T(1)+(a^{n-1}+\cdots+a+1)b$

  $=ca^{n-1}+b(a^n-1)/(a-1)$ 或 $c+nb$

- 2 阶递归方程

  $T(n)=aT(n-1)+bT(n-2)+c$，$T(0)=p$, $T(1)=q$:

  设α,β 是特征方程$\lambda^2-a\lambda-b=0$的两个复根，则

  $\alpha+\beta=a,\ \alpha\beta=-b$

于是，$T(n)-\alpha T(n-1)=\beta(T(n-1)-\alpha T(n-2))+c$，令

$D(n)=T(n)-\alpha T(n-1)$,

则 $D(n)=\beta D(n-1)+c$，转化成1阶的情形