1.3 算法分析基础

分析算法占用的资源

- CPU时间 → 时间性能分析
- 内存空间 → 空间性能分析

算法分析目的： 分析算法的时空效率以便改进算法性能。

1.3.1 算法时间复杂度分析

一个算法是由控制结构（顺序、分支和循环三种）和原操作（指固有数据类型的操作，如+、-、*、/、++和--等）构成的。算法执行时间取决于两者的综合效果。

一个算法的基本构成：

控制语句1 原操作	控制语句2 原操作	…	控制语句n 原操作

例如：

```c
void fun(int a[],int n)
{   int i;
    for (i=0;i<n;i++)
        a[i]=2*i;
    for (i=0;i<n;i++)
        printf("%d", a[i]);
    printf("\n");
}
```

原操作

算法分析方式：

❶ 事后分析统计方法：编写算法对应程序，统计其执行时间。

- 编写程序的语言不同
- 执行程序的环境不同
- 其他因素

所以不能用绝对执行时间进行比较。

❷ 事前估算分析方法：撇开上述因素，认为算法的执行时间是问题规模n的函数。✓

❶ 分析算法的执行时间

- 求出算法所有原操作的执行次数（也称为**频度**），它是**问题规模n**的函数，用$T(n)$表示。
- 算法执行时间大致＝原操作所需的时间×$T(n)$。所以**$T(n)$与算法的执行时间成正比**。为此用$T(n)$表示算法的执行时间。
- 比较不同算法的$T(n)$大小得出算法执行时间的好坏。

用于表示求解问题大小的正整数，如n个记录排序

【例1-4】求两个n阶方阵的相加$C=A+B$的算法如下，分析其时间复杂度。

```
#define MAX   20    //定义最大的方阶
void matrixadd(int n,int A[MAX][MAX],int B[MAX][MAX],
        int C[MAX][MAX])
 {          int i,j;
            for (i=0;i<n;i++)                      //①
                  for (j=0;j<n;j++)                //②
                        C[i][j]=A[i][j]+B[i][j];   //③
 }
```

```
#define MAX   20   //定义最大的方阶
void matrixadd(int n,int A[MAX][MAX],
     int B[MAX][MAX], int C[MAX][MAX])
 {        int i,j;
          for (i=0;i<n;i++)                 //①
              for (j=0;j<n;j++)             //②
                  C[i][j]=A[i][j]+B[i][j];  //③
 }
```

解：除变量定义语句外，该算法包括3个可执行语句①、②和③。

- ① 频度为$n+1$，循环体执行n次
- ② 频度为$n(n+1)$
- ③ 频度为n^2

所有语句频度之和为：

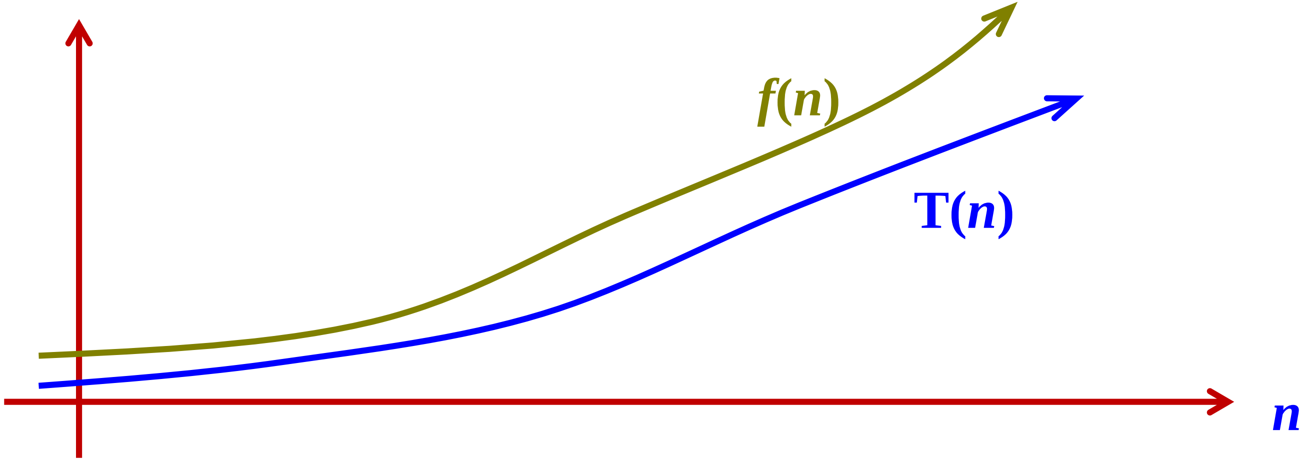
$$T(n) = n+1+n(n+1)+n^2$$
$$= 2n^2+2n+1$$

❷ 算法的执行时间用时间复杂度来表示

算法中执行时间T(n)是问题规模n的某个函数$f(n)$，记作：

$$T(n) = O(f(n))$$

记号“O”读作“大O”，它表示随问题规模n的增大算法执行时间的增长率和$f(n)$的增长率相同。⇨ 趋势分析



“O”的形式定义为：$T(n) = O(f(n))$表示存在一个正的常数M，使得当$n \geqslant n_0$时都满足：

$$|T(n)| \leqslant M|f(n)|$$

$f(n)$是$T(n)$的上界

这种上界可能很多，通常取最接近的上界，即紧凑上界

大致情况：

$$\lim_{n \to \infty} \frac{T(n)}{f(n)} = M$$

本质上讲，是一种T(n)最高数量级的比较

也就是只求出T(n)的最高阶，忽略其低阶项和常系数，这样既可简化T(n)的计算，又能比较客观地反映出当n很大时算法的时间性能。

例如： $T(n) = 2n^2+2n+1 = O(n^2)$

一般地：

- 一个没有循环的算法的执行时间与问题规模n无关，记作$O(1)$，也称作常数阶。
- 一个只有一重循环的算法的执行时间与问题规模n的增长呈线性增大关系，记作$O(n)$，也称线性阶。
- 其余常用的算法时间复杂度还有平方阶$O(n^2)$、立方阶$O(n^3)$、对数阶$O(\log_2 n)$、指数阶$O(2^n)$等。

各种不同算法时间复杂度的比较关系如下：

$$\underbrace{O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)}_{\text{多项式阶：P问题}}<\underbrace{O(2^n)<O(n!)}_{\text{指数阶：NP问题}}$$

多项式阶：P问题

指数阶：NP问题

NP = P？是目前计算机科学的难题之一

算法时间性能比较：假如求同一问题有两个算法：*A*和*B*，如果算法*A*的平均时间复杂度为O(n)，而算法*B*的平均时间复杂度为O(n^2)。

一般情况下，认为算法*A*的时间性能好比算法*B*。

❸ 简化的算法时间复杂度分析

算法中的**基本操作**一般是最深层循环内的**原操作**。

算法执行时间大致 = 基本操作所需的时间 × 其运算次数。

↓ 转化

在算法分析时，计算$T(n)$时仅仅考虑基本操作的运算次数。

【例1-4】求两个n阶方阵的相加$C=A+B$的算法如下，分析其时间复杂度。

```
#define MAX   20    //定义最大的方阶
void matrixadd(int n,int A[MAX][MAX],int B[MAX][MAX],
    int C[MAX][MAX])
{           int i,j;
            for (i=0;i<n;i++)
                        for (j=0;j<n;j++)
                                    C[i][j]=A[i][j]+B[i][j];
}
```

基本操作

解：该算法中的基本操作是两重循环中最深层的语句C[i][j]=A[i][j]+B[i][j]，分析它的频度，即：

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n = n\sum_{i=0}^{n-1} 1 = n*n = n^2$$

$$= O(n^2)$$

这种简化的时间复杂度分析方法得到的结果相同，但分析过程更简单。

思考题

下列程序段的时间复杂度是______。

```
count=0;
for(k=1;k<=n;k*=2)
    for(j=1;j<=n;j++)
        count++;
```

基本操作

A.O($\log_2 n$)　　B.O(n)　　C.O($n\log_2 n$)　　D.O(n^2)

说明：本题为2014年全国考研题

【例1−5】分析以下算法的时间复杂度。

```
void func(int n)
{    int i=0,s=0;
     while (s<n)
     {    i++;
          s=s+i;
     }
}
```

基本操作

解： 对于while循环语句，设执行的次数为m，变量i从0开始递增1，直到m为止，有：

循环结束：$s=m(m+1)/2\geqslant n$，或者$m(m+1)/2+k=n$ 。

用于修正的常量

则：

$$m=\frac{-1+\sqrt{8n+1-8k}}{2}$$

$$T(n)=m=O(\sqrt{n})$$

所以，该算法的时间复杂度为$O(\sqrt{n})$。

1.3.2 算法空间复杂度分析

空间复杂度：用于量度一个算法在运行过程中临时占用的存储空间大小。

一般也作为问题规模n的函数，采用数量级形式描述，记作：

$$S(n) = O(g(n))$$

若一个算法的空间复杂度为O(1)，则称此算法为原地工作或就地工作算法。

为什么空间复杂度分析只考虑临时占用的存储空间？

```
int max(int  a[],int n)
{       int i, maxi=0;
        for (i=1;i<=n;i++)
                if (a[i]>a[maxi])
                        maxi=i;
        return a[maxi];
}
```

如果max函数中再考虑形参*a*的空间，就重复累计了执行整个算法所需的空间。

max算法的空间复杂度为O(1)

max()

max(b,n)

maxfun()

```
void maxfun()
{       int b[]={1,2,3,4,5},n=5;
        printf("Max=%d\n",max(b,n));
}
```

maxfun算法中为*b*数组分配了相应的内存空间，其空间复杂度为O(*n*)

【例1–6】 分析如下算法的空间复杂度。

```
int fun(int n)
{     int i, j, k, s;
      s=0;
      for (i=0;i<=n;i++)
          for (j=0;j<=i;j++)
              for (k=0;k<=j;k++)
                  s++;
   return(s);
}
```

临时占用的存储空间：函数体内分配的空间

解：算法中临时分配的变量个数与问题规模n无关，所以空间复杂度均为O(1)。

思考题

为什么算法的时、空分析都采用复杂度的形式表示?

——本讲完——