1.4 其他情况的算法分析

1.4.1 最好、最坏和平均时间复杂度分析

定义： 设一个算法的输入规模为 n，D_n是所有输入的集合，任一输入$I \in D_n$，$P(I)$是I出现的概率，有 $\sum_{I \in D_n} P(I) = 1$，$T(I)$是算法在输入$I$下的执行时间，则算法的平均时间复杂度为：

$$A(n) = \sum_{I \in D_n} P(I) \times T(I)$$

例如，10个1～10的整数序列递增排序：

$$\left.\begin{array}{l} n=10 \\ I_1=\{1,2,3,4,5,6,7,8,9,10\} \\ I_2=\{2,1,3,4,5,6,7,8,9,10\} \\ \quad\cdots \\ I_m=\{10,9,8,7,6,5,4,3,2,1\} \end{array}\right\}\text{构成}D_n\text{，}P(I)=1/m$$

所有可能的初始序列有m个，$m=10!$

算法的最坏时间复杂度为：$W(n)=\max_{I\in D_n}\{T(I)\}$

算法的最好时间复杂度为：$B(n)=\min_{I\in D_n}\{T(I)\}$

一种或几种特殊情况

【例1-6】设计一个算法，求含n个整数元素的序列中前i（$1 \leqslant i \leqslant n$）个元素的最大值。并分析算法的平均时间复杂度。

解：整数序列用数组a表示，前i（$1 \leqslant i \leqslant n$）个元素为$a[0..i-1]$。

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

解：i的取值范围为1～n（共n种情况），对于求前i个元素的最大值时，需要元素比较$(i-1)-1+1=i-1$次。在等概率情况（每种情况的概率为$1/n$）：

$$T(n)=\sum_{i}^{n}\frac{1}{n}\times(i-1)=\frac{1}{n}\sum_{i}^{n}(i-1)=\frac{n-1}{2}$$

$$=O(n)\longleftarrow\text{平均时间复杂度}$$

该算法的最坏复杂度：$W(n)=O(n)$（当$i=n$时）

该算法的最好复杂度：$B(n)=O(1)$（当$i=1$时）

1.4.2 递归算法的时空复杂度分析

递归算法是指算法中出现调用自己的成分。

- 递归算法分析也称为变长时空分析。
- 非递归算法分析也称为定长时空分析。

1、递归算法的时间复杂度分析

【例1−7】 有如下递归算法：

```
void fun(int a[],int n,int k)   //数组a共有n个元素
{     int i;
      if (k==n-1)
          for (i=0;i<n;i++)         //n次
               printf("%d\n",a[i]);
      else
      {        for (i=k;i<n;i++)    //n-k次
                    a[i]=a[i]+i*i;
               fun(a,n,k+1);
       }
}
```

调用上述算法的语句为fun(*a*,*n*,0)，求其时间复杂度。

递归算法：

```
void fun(int a[],int n,int k)   //数组a共有n个元素
 {     int i;
       if (k==n-1)
           for (i=0;i<n;i++)        //n次
                printf("%d\n",a[i]);
       else
       {        for (i=k;i<n;i++)    //n-k次
                     a[i]=a[i]+i*i;
                fun(a,n,k+1);
        }
 }
```

含一重循环

fun(a,n,0)的时间复杂度为O(n)。 ⟵ 错误

解： 设fun(a,n,0)的执行时间为T(n)，fun(a,n,k)的执行时间为$T_1(n,k)$ ⇨ $T(n) = T_1(n,0)$。

由fun()递归算法可知：

$T_1(n,k) = n$	当$k=n-1$时
$T_1(n,k) = (n-k)+T_1(n,k+1)$	其他情况

则

$$
\begin{aligned}
T(n) &= T_1(n,0) = n+T_1(n,1) = n+(n-1)+T_1(n,2) \\
&= \cdots = n+(n-1)+\cdots+2+T_1(n,n-1) \\
&= n+(n-1)+\cdots+2+n \\
&= O(n^2)
\end{aligned}
$$

所以调用fun(a,n,0)的时间复杂度为$O(n^2)$。

2、递归算法的空间复杂度分析

【例1-8】有如下递归算法，分析调用fun(*a*,*n*,0)的空间复杂度。

```
void fun(int a[],int n,int k)   //数组a共有n个元素
 {     int i;
       if (k==n-1)
           for (i=0;i<n;i++)          //n次
                printf("%d\n",a[i]);
       else
       {       for (i=k;i<n;i++)    //n-k次
                    a[i]=a[i]+i*i;
               fun(a,n,k+1);
       }
 }
```

递归算法：

```
void fun(int a[],int n,int k)   //数组a共有n个元素
 {    int i;
      if (k==n-1)
           for (i=0;i<n;i++)        //n次
                 printf("%d\n",a[i]);
      else
      {        for (i=k;i<n;i++)    //n-k次
                      a[i]=a[i]+i*i;
               fun(a,n,k+1);
       }
 }
```

仅仅定义了一个临时变量*i*

fun(*a*,*n*,0)的空间复杂度为O(1)。 ⟵ 错误

解： 设fun($a,n,0$)的空间为$S(n)$，fun(a,n,k)的空间为$S_1(n,k)$ ⇨ $S(n) = S_1(n,0)$。

由fun()递归算法可知：

$S_1(n,k) = 1$	当$k=n-1$时
$S_1(k) = 1+S_1(n,k+1)$	其他情况

则：

$$S(n) = S_1(n,0) = 1+S_1(n,1) = 1+1+S_1(n,2)$$
$$= \cdots = \underbrace{1 + 1 + \cdots + 1}_{n\text{个}1} = O(n)$$

所以调用fun($a,n,0$)的空间复杂度为$O(n)$。

思考题

递归算法和非递归算法在分析时间复杂度和空间复杂度上为什么不同？

——本章完——