第一部分　习题及解答

第 1 章　绪论

1.1　基本概念

1.1.1　数据结构

数据结构（Data Structure）：是指相互之间存在一种或多种特定关系的数据元素所组成的集合。具体来说，数据结构包含三个方面的内容，即数据的逻辑结构、数据的存储结构和对数据所施加的运算（也称操作）。

这三个方面的关系为：

（1）数据的逻辑结构独立于计算机，是数据本身所固有的。

（2）存储结构是逻辑结构在计算机存储器中的映像，必须依赖于计算机。

（3）运算是指所施加的一组操作的总称。运算的定义直接依赖于逻辑结构，但运算的实现必须依赖于存储结构。

数据结构从逻辑结构上可以分为：线性结构、非线性结构。

数据结构具体可以表示为：集合、线性表、栈、队列、串、多维数组、广义表、树、二叉树、图形结构。

1.1.2　存储方式

数据结构从存储结构上可以分为如下几种：

（1）顺序存储（向量存储）

所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻（只需存放元素的值，而不需存放元素之间的关系）。

（2）链式存储

所有元素存放在可以是不连续的存储单元中，但元素之间的关系可以通过地址确定，逻辑上相邻的元素存放到计算机内存后不一定是相邻的（也可能是相邻的）。

（3）索引存储

使用该方法存放元素的同时，还应建立附加的索引表，索引表中的每一项都称为索引项，索引项的一般形式是：（关键字，地址），其中的关键字能唯一标识一个结点的数据项。而地址是存储关键字的具体位置。

（4）散列存储

通过构造散列函数，用函数的值来确定元素存放的地址（理论上不需用到比较）。

1.1.3 算法及评价

通俗地讲，算法就是一种解题的方法。更严格地说，算法是由若干条指令组成的有穷序列。

评价一种算法的好坏，主要考虑以下三个方面：

（1）执行算法所耗费的时间（时间复杂度）。

（2）执行算法所占用的内存开销（主要考虑占用的辅助存储空间，称为空间复杂度）。

（3）算法应该易于理解、易于调试、易于编码。

本书中，主要是讨论算法的时间频度和时间复杂度。

1.2 习题及解答

1.2.1 配套教材中的习题

1-1．对下列用二元组表示的数据结构，画出它们的逻辑结构图，并指出它们属于何种结构。

（1）A=(K,R)，其中：

$K=\{a_1,a_2,a_3,\cdots,a_n\}$

$R=\{\langle a_i,a_{i+1}\rangle,i=1,2,\cdots,n-1\}$

（2）B=(K,R)，其中：

K={a,b,c,d,e,f}

R={<a,b>,<b,c>,<c,a>,<a,d>,<d,e>,<e,f>,<c,f>}

（3）C=(K,R)，其中：

K={1,2,3,4,5,6}

R={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}

（4）D=(K,R)，其中：

K={48,25,64,57,82,36,75,43}

R={r1,r2,r3}

r1={<48,25>,<25,64>,<64,57>,<57,82>,<82,36>,<36,75>,<75,43>}

r2={<48,25>,<48,64>,<64,57>,<64,82>,<25,36>,<82,75>,<36,43>}

r3={<25,36>,<36,43>,<43,48>,<48,57>,<57,64>,<64,75>,<75,82>}

解：（1）的逻辑结构如图 1-1 所示。该结构为线性结构。

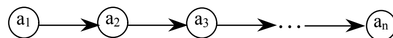

图 1-1 （1）的逻辑结构

（2）的逻辑结构如图 1-2 所示。该结构为图形结构。

（3）的逻辑结构如图 1-3 所示。该结构为图形结构。

（4）的逻辑结构如图 1-4 所示。该结构为图形结构。

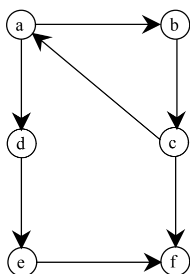

图 1-2　（2）的逻辑结构

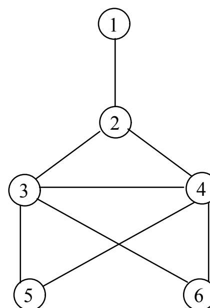

图 1-3　（3）的逻辑结构

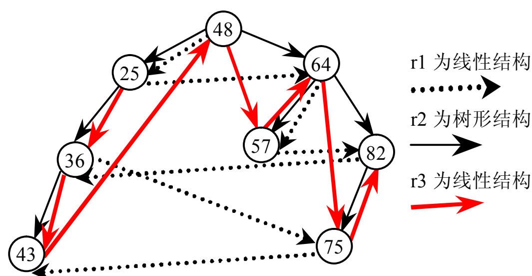

图 1-4　（4）的逻辑结构

1-2．简述概念：数据、数据元素、数据结构、逻辑结构、存储结构、线性结构、非线性结构。

解：

数据：是指能够输入到计算机中，并被计算机识别和处理的符号的集合。

数据元素：是组成数据的基本单位，是一个数据整体中相对独立的单位，但它还可以分割成若干个具有不同属性的项（字段），故其不是组成数据的最小单位。

数据结构：是指相互之间存在一种或多种特定关系的数据元素所组成的集合。具体来说，数据结构包含三个方面的内容，即数据的逻辑结构、数据的存储结构和对数据所施加的运算。

逻辑结构：是数据之间的内在关系，是数据本身所固有的，数据的逻辑结构独立于计算机。

存储结构：是逻辑结构在计算机存储器中的映像，必须依赖于计算机。

线性结构：元素之间存在一种一对一的线性关系的数据结构。

非线性结构：元素之间存在一对多或多对多的非线性关系的数据结构。

1-3．试举日常生活中的一个例子来说明数据结构三个方面的内容。

解：可以用日常生活中的一种物质——水，来说明数据结构的三个方面。

水由许多水分子组成，一个水分子由氢和氧两种元素组成，这与外界的环境无关，相当于数据的逻辑结构。

水在日常生活中有液态、气态、固态三种不同形态，相当于数据有三种存储结构。

对水进行升温、降温等操作相当于对数据所施加的操作。

1-4．什么是算法？它的五个特性是什么？

解：通俗地讲，算法就是一种解题的方法。更严格地说，算法是由若干条指令组成的有穷序列，它必须满足下述条件（也称为算法的五大特性）：

（1）输入：具有 0 个或多个输入的外界量（算法开始前的初始量）。

（2）输出：至少产生一个输出，它们是算法执行完后的结果。

（3）有穷性：每条指令的执行次数必须是有限的。

（4）确定性：每条指令的含义都必须明确，无二义性。

（5）可行性：每条指令的执行时间都是有限的。

1-5．设 n 为整数，分析下列程序中用#标明的语句的语句频度及时间复杂度。

```
（1）for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {   c[i][j]=0;
            for (k=1; k<=n; k++)
            # c[i][j]=c[i][j]+a[i][k]*b[k][j];
        }
（2）int i=1, j=1;
     while (i<=n&&j<=n)   {# i=i+1; j=j+i; };
（3）int i=1;
     do {for (j=1; j<=n; j++)   # i=i+j
     } while (i<100+n);
（4）for (i=1; i<=n; i++)
        for (j=i; j>=1; j--)
            for (k=1; k<=j; k++)   # x=x+1;
```

解：（1）语句频度 $T(n)=n^3$，时间复杂度为：$O(n^3)$。

（2）语句频度 $T(n)=\lfloor \frac{\sqrt{8n+1}-1}{2} \rfloor$，时间复杂度为：$O(\lfloor \frac{\sqrt{8n+1}-1}{2} \rfloor)$。

（3）语句频度 $T(n)=\lceil \frac{198+2n}{n(n+1)} \rceil n$，时间复杂度为：$O(\lceil \frac{198+2n}{n(n+1)} \rceil n)$。

（4）语句频度 $T(n)=\frac{1}{2}[\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}]$，时间复杂度为：$O(n^3)$。

1-6．设计出一个算法，将 n 个元素按升序排列，并分析出它的时间频度及时间复杂度。

解：

```
void sort(int a[ ],int n)
{
  int i,j,k,t;
  for(i=0;i<n-1;i++)
  {
    k=i;
    for(j=i+1;j<=n-1;j++)
    if(a[k]>a[j])
    k=j;
    if(k!=i)
    {
```

```
            t=a[k];
            a[k]=a[i];
            a[i]=t;
        }
    }
}
```

该算法的时间频度为 $T(n)=\frac{(n-1)(n+8)}{2}$，时间复杂度为 $O(n^2)$。

1-7．计算 $\sum_{i=0}^{n}\frac{x^i}{i+1}$ 的值，用 C 语言函数写出算法，并求出它的时间复杂度。

解：

```
float sum(float x,int n)
{
   float s=1,p=1;
   int i;
   for( i=1;i<=n;i++)
   {
         p=p*x;
         s+=p/(i+1);
   }
   return s; }
```

该算法的时间复杂度为 O(n)。

1-8．用 C 函数描述如何将一维数组中的元素逆置，并分析出时间频度及时间复杂度。

解：

```
void exchange(int a[ ],int n)
{int t;
   for(int i=0;i<=n/2-1;i++)
   {
      t=a[i];
      a[i]=a[n-i-1];
      a[n-i-1]=t;
   }
}
```

该算法的时间频度为 $T(n)=\frac{3n}{2}$，时间复杂度为 O(n)。

1-9．将三个元素 X、Y、Z 按从小到大排列，用 C 语言函数描述算法，要求所用的比较和移动元素次数最少。

解：

```
void abc(int &x,int &y,int &z)
{
   if(x>y)
   {int t=x;x=y;y=t;}
   if(y>z)
```

```
    { int t=z;z=y;
      if(x<=t) y=t;
      else {y=x; x=t;}
    }
}
```

该算法最坏时仅需 3 次比较和 7 次移动。

1-10．用 C 语言中冒泡排序和选择排序两种方法，分别描述出 n 个元素 $a_1,a_2,a_3,\cdots,a_n$ 的升序排列算法，并分析两种方法的平均比较和移动元素次数。

解：

冒泡排序：

```
void bubblesort(int a[ ],int n)
{
  int i,j,t;
  for( i=0;i<n-1;i++)
    for( j=n-1;j>=i+1;j--)
      if(a[j]<a[j-1])
       {int t=a[j];a[j]=a[j-1];a[j-1]=t;}
}
```

冒泡排序的最小比较次数为 n-1，最小移动次数为 0，最大比较次数为 $\frac{n(n-1)}{2}$，最大移动次数为 $\frac{3n(n-1)}{2}$，平均比较次数为 $\frac{(n-1)(n+2)}{4}$，平均移动次数为 $\frac{3n(n-1)}{4}$。

选择排序：

```
void bubblesort(int a[ ],int n)
{
  int i,j,t;
   for(i=0;i<n-1;i++)
     for(j=n-1;j>=i+1;j--)
      if(a[j]<a[j-1])
      {int t=a[j];a[j]=a[j-1];a[j-1]=t;}
}
```

选择排序的最小比较次数为 $\frac{n(n-1)}{2}$，最小移动次数为 0，最大比较次数为 $\frac{n(n-1)}{2}$，最大移动次数为 3(n-1)，平均比较次数为 $\frac{n(n-1)}{2}$，平均移动次数为 $\frac{3n(n-1)}{2}$。

1-11．设计一个二次多项式 ax^2+bx+c 的抽象数据类型，假定起名为 AX2BXC，该类型的数据部分分为 3 个系数 a、b 和 c，操作部分为：

（1）初始化成员 a、b 和 c（假定用结构体 D 来定义 a、b、c）。

```
AX2BXC initAX2BXC(x,y,z);
```

（2）做两个多项式的加法运算，即将它们的系数相加，并返回相加的结果。

```
AX2BXC add(AX2BXC f1, AX2BXC f2);
```

（3）根据给定的 x 值，求多项式的值。

```
float value(AX2BXC f,float x);
```

（4）计算方程 $ax^2+bx+c=0$ 的两个实根，要求分有实根、无实根和不是二次方程这三种情况讨论，并返回不同的值，以便调用时做不同的处理。

```
int root(AX2BXC f, float &r1,float &r2);
```

（5）按照 ax**2+bx+c 的格式（x^2 用 x**2 表示）输出二次多项式，在输出时必须注意去掉系数为 0 的项，并且当 b 和 c 的值为负时，其前面不能出现加号。

```
void print(AX2BXC f);
```

请写出上面每一个操作的具体实现。

解：

抽象数据类型描述为：

```
ADT Ax2BxC is
Data: a,b,c 为 3 个实型数
Operation:
Ax2BxC initAx2BxC(float x,float y,float z);
Ax2BxC add(Ax2BxC f1, Ax2BxC f2);
float value(Ax2BxC f, float x);
int root(Ax2BxC f,float &r1,float &r2);
void print(Ax2BxC f);
End Ax2BxC
```

上面每一个操作的具体实现如下：

```
struct Ax2BxC
{   float a,b,c;
}D;
Ax2BxC initAx2BxC(float x,float y,float z)
{
    D.a=x;D.b=y;D.c=z;
    return D;
}

Ax2BxC add(Ax2BxC f1, Ax2BxC f2)
{
    f1.a=f1.a+f2.a;
    f1.b=f1.b+f2.b;
    f1.c=f1.c+f2.c;
    return f1;
}

float value(Ax2BxC f, float x)
{
    return f.a*x*x+f.b*x+f.c;
}

int root(Ax2BxC f,float &r1,float &r2)
{
    float r1,r2, d=f.b*f.b-4*f.a*f.c;
    int e;
```

```
        if(f.a!=0)
        {   if(d>=0)
            {   r1=(-f.b+sqrt(d))/(2*f.a);
                r2=(-f.b-sqrt(d))/(2*f.a);
                e=1;                              //有实根
            }
            else e=0;                             //无实根
        }
        else e=-1;                                //不是二次方程
        return e;
    }

    void print(Ax2BxC f)
    {   if(f.a!=0)
        {   printf("%f x**2",f.a);
            if(f.b!=0)
            {   if(f.b>0)   printf("+");
                printf("%fx",f.b);
            }
            if(f.c!=0)
            {   if(f.c>0)   printf("+");
                printf("%f\n",f.c);
            }
        }
        else { if(f.b!=0)   printf("%fx",f.b);
                if(f.c!=0)
                {if(f.c>0) printf(" +");
                 printf("%f\n",f.c);
                }
        }
    }
```

1-12．指出下列各算法的功能并求出其时间复杂度。

（1）int prime(int n)

```
    {
      int i=2;
      int x=(int)sqrt(n);
      while(i<=x)
      {   if(n%i==0) break;
          i++;
      }
      if(i>x) return 1;
      else return 0;
    }
```

（2）int sum1(int n)

```
    { int p=1,s=0;
      for(int i=1;i<=n;i++)
```

```
      { p*=i;
        s+=p;}
      return s;
    }
```

（3）int sum2(int n)

```
    { int s=0;
      for(int i=1;i<=n;i++)
      { int p=1;
        for(int j=1;j<=i;j++)
        p*=j;
        s+=p;}
      return s;
    }
```

（4）void print(int n)

```
    { int i;
       for(i=1;i<=n;i++)
       {
          if(i%2) printf("*");
          else continue;
          printf("#");
       }
    printf("$\n");}
```

（5）void print1(int n)

```
    { int i,k;
      long j;
      for(i=1;i<=n;i++)
      { j=i*i;
        if(j<10)
        {if(j==i)printf("%d %d\n",i,j);}
        else if(j<100)
        {if(j%10==i) printf("%d %d\n",i,j);}
        else if(j<1000)
        {if(j%100==i) printf("%d %d\n",i,j);}
        else if(j<10000)
        {if(j%1000==i) printf("%d %d\n",i,j);}
        else if(j<100000)
        {if(j%10000==i) printf("%d %d\n",i,j);}
        else
        {if(j%100000==i) printf("%d %d\n",i,j);}
      }
    }
```

（6）void matrix(int a[m][n],int b[n][l],int c[m][l])

```
    { int i,j,k;
      for(i=0;i<m;i++)
      for(j=0;j<l;j++)
      { c[i][j]=0;
```

```
            for(k=0;k<n;k++)
            c[i][j]+=a[i][k]*b[k][j];
        }
    }
（7）void xyx(int n)        //n<1000
    { int i,j,k,l;
      for (l=100;l<=n;l++)
      {
          i=l/100;
          j=l/10%10;
          k=l%10;
          if(k*100+j*10+i==l)
          printf("%d\n",l);
      }
    }
（8）int sum3(int n)
    { int i,s=0;
      for(i=1;i<=n;i++)
      if(i%2==1)
          s+=i;
      else s+=i*i;
      return s;
    }
```

解：

（1）功能：判断 n 是否为素数，时间复杂度为 $O(\sqrt{n})$。

（2）功能：求 $\sum_{i=1}^{n} i!$，时间复杂度为 O(n)。

（3）功能：求 $\sum_{i=1}^{n} i!$，时间复杂度为 $O(n^2)$。

（4）功能：输出 $\frac{n+1}{2}$ 组“*#”后，最后输出一个“$”，时间复杂度为 O(n)。

（5）功能：求 1000 以内的所有同构数（该数出现在它的平方的右边），时间复杂度为 O(n)。

（6）功能：求矩阵的乘积，时间复杂度为 O(m*n*l)。

（7）功能：求 1000 以内的回文数，时间复杂度为 O(n)。

（8）功能：求 1～n 中奇数的和与偶数的平方和，时间复杂度为 O(n)。

1.2.2 综合题

1-1．将时间复杂度 O(1)、O(n)、$O(n^2)$、$O(n^3)$、$O(n\log_2 n)$、$O(\log_2 n)$、$O(2^n)$按递增顺序排列。

解：上述时间复杂度按递增顺序排列为 O(1)、$O(\log_2 n)$、O(n)、$O(n\log_2 n)$、$O(n^2)$、$O(n^3)$、$O(2^n)$。

1-2．编程，从 n（最大为 100）个数据中求出最大值和最小值，并分析时间复杂度。

解：

```
#define N 100
#include<stdio.h>
void search(int a[ ], int n)
{ int j,max,min;
  max=min=a[0];
  for(j=1;j<n;j++)
  {
    if(a[j]>max) max=a[j];
    if(a[j]<min) min=a[j];
  }
  printf("最大值为：%d\n",max);
  printf("最小值为：%d\n",min);
}
void main()
{ int n,j,a[N];
  printf("输入数组最多的元素个数\n");
  scanf("%d",&n);
  printf("输入数组元素的值:\n ");
  for(j=0;j<n;j++)
  scanf("%d",&a[j]);
  search(a,n);
}
```

该算法的时间复杂度为 O(n)。

1-3．设计算法实现矩阵的相加，并分析该算法的时间复杂度。

解：

```
void matrixadd( int a[m][n],int b[m][n],c[m][n])
{
    int j,k;
        for(j=0;j<m;j++)
          for(k=0;k<n;k++)
            c[j][k]=a[j][k]+b[j][k];
}
```

该算法的时间复杂度为 O(m×n)。

1-4．设计算法实现矩阵的相乘，并分析该算法的时间复杂度。

解：

```
void matrixmul(int a[m][n],b[n][l],c[m][l])
{
    int i,j,k;
    for(i=0;i<m;i++)
    for(j=0;j<l;j++)
    {
          c[i][j]=0;
          for(k=0;k<n;k++)
          c[i][j]=c[i][j]+a[i][k]*b[k][j];
```

```
        }
    }
```

该算法的时间复杂度为 O(m*n*l)。

1-5．求出下面程序的时间复杂度。

```
#include<stdio.h>
int a[ ]={2,5,1,7,9,3,6,8};
void order(int j,int m)
{
   int i,temp;
   if(j<m)
   {
      for(i=j;i<=m;i++)
      if(a[i]<a[j])
      {
         temp=a[i];
         a[i]=a[j];
         a[j]=temp;
      }
      j++;
      order(j,m);
   }
}
void main()
{
   int i;
   order(0,7);
   for(i=0;i<=7;i++)
   printf("%d     ",a[i]);
   printf("\n ");
}
```

解：order 函数是一个递归的排序函数，该算法花费的时间主要是在递归函数上面。设排序花费的时间为 T(n)，则有 T(n)=T(n-1)+n-1，其中 T(1)=0。

于是有：T(n)=T(n-1)+n-1

=T(n-2)+n-2+n-1

=T(1)+1+2+…+n-1

=n(n-1)/2

因此，该算法的时间复杂度为 $O(n^2)$。

1-6．试编写算法求一元多项式 $P_n(x)=\sum_{i=0}^{n} a_i x^i$ 的值 $P_n(x_0)$，并分析整个算法的时间复杂度。

解：

```
float PnX( float a[ ], int n, float x)
{
   float p, t;
   int i;
```

```
    p=0; t=1;
    for(i=0; i<n; i++)
    {
        p=p+a[i]*t;
        t=t*x;
    }
    return p;
}
```

该算法的时间复杂度为 O(n)。

1-7．设计时间复杂度为 $O(n^2)$和 O(n)的两种算法，求 $s=\sum_{i=1}^{n} i!$。

解：

算法 1：

```
float mul( int n )
{
    int j;
    float s, p;
    s=0;
    for( j=1; j<=n; j++)
    { p=1;
      for( k=1; k<=j; k++ )
      p=p*k;
      s=s+p;
    }
    return s;
}
```

该算法的时间复杂度为 $O(n^2)$。

算法 2：

```
float mul1( int n)
{
    int j;
    float s, p;
    s=0;
    p=1;
    for( j=1;j<=n;j++)
    {
      p=p*j;
      s=s+p;
    }
    return s;
}
```

该算法的时间复杂度为 O(n)。