第 1 章 绪 论

课后习题讲解

4. 分析以下各程序段，并用大 O 记号表示其执行时间。

(1)
```
i=1; k=0;
while (i<n-1)
{
  k=k+10*i;
  i++;
}
```

(2)
```
i=1; k=0;
do
{
  k=k+10*i;
  i++;
} while (i<=n)
```

(3)
```
i=1; j=0;
while (i+j<=n)
  if (i>j) j++;
    else i++;
```

(4)
```
y=0;
while ((y+1)*(y+1)<=n)
    y=y+1;
```

(5)
```
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    for (k=1; k<=j; k++)
        x++;
```

【解答】⑴ 基本语句是 k=k+10*i，共执行了 n-2 次，所以 T(n)=O(n)。
⑵ 基本语句是 k=k+10*i，共执行了 n 次，所以 T(n)=O(n)。
⑶ 分析条件语句，每循环一次，i+j 整体加 1，共循环 n 次，所以 T(n)=O(n)。
⑷ 设循环体共执行 T(n)次，每循环一次，循环变量 y 加 1，最终 T(n)=y，即：
(T(n)+1)2≤n，所以 T(n)=O(n1/2)。
⑸ x++是基本语句，所以 $T(n)=\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=O(n^3)$

5．设有数据结构（D，R），其中 D={1, 2, 3, 4, 5, 6}，R={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}。试画出其逻辑结构图并指出属于何种结构。
【解答】其逻辑结构图如图 1-3 所示，它是一种图结构。

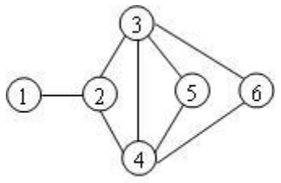


图 1-3 逻辑结构图

6. 为整数定义一个抽象数据类型，包含整数的常见运算，每个运算对应一个基本操作，每个基本操作的接口需定义前置条件、输入、功能、输出和后置条件。
【解答】整数的抽象数据类型定义如下：
ADT integer
Data
整数 a：可以是正整数(1, 2, 3, …)、负整数(-1, -2, -3, …)和零
Operation
Constructor
前置条件：整数 a 不存在
输入：一个整数 b
功能：构造一个与输入值相同的整数
输出：无
后置条件：整数 a 具有输入的值
Set
前置条件：存在一个整数 a
输入：一个整数 b
功能：修改整数 a 的值，使之与输入的整数值相同
输出：无
后置条件：整数 a 的值发生改变
Add
前置条件：存在一个整数 a
输入：一个整数 b
功能：将整数 a 与输入的整数 b 相加
输出：相加后的结果
后置条件：整数 a 的值发生改变
Sub
前置条件：存在一个整数 a
输入：一个整数 b
功能：将整数 a 与输入的整数 b 相减
输出：相减的结果
后置条件：整数 a 的值发生改变
Multi
前置条件：存在一个整数 a
输入：一个整数 b
功能：将整数 a 与输入的整数 b 相乘
输出：相乘的结果
后置条件：整数 a 的值发生改变
Div
前置条件：存在一个整数 a
输入：一个整数 b
功能：将整数 a 与输入的整数 b 相除
输出：若整数 b 为零，则抛出除零异常，否则输出相除的结果
后置条件：整数 a 的值发生改变
Mod
前置条件：存在一个整数 a
输入：一个整数 b
功能：求当前整数与输入整数的模，即正的余数
输出：若整数 b 为零，则抛出除零异常，否则输出取模的结果
后置条件：整数 a 的值发生改变
Equal
前置条件：存在一个整数 a
输入：一个整数 b
功能：判断整数 a 与输入的整数 b 是否相等
输出：若相等返回 1，否则返回 0
后置条件：整数 a 的值不发生改变
endADT
7. 求多项式 A(x)的算法可根据下列两个公式之一来设计：
⑴ A(x)=anxn+an-1xn-1+…+a1x+a0
⑵ A(x)=(…(anx+an-1)x+…+a1)x)+a0

根据算法的时间复杂度分析比较这两种算法的优劣。

【解答】第二种算法的时间性能要好些。第一种算法需执行大量的乘法运算，而第二种算法进行了优化，减少了不必要的乘法运算。

8. 算法设计（要求：算法用伪代码和 C++描述，并分析最坏情况下的时间复杂度）

⑴ 对一个整型数组 A[n]设计一个排序算法。

【解答】下面是简单选择排序算法的伪代码描述。

伪代码

1. 对 n 个记录进行 n-1 趟简单选择排序：
   1.1 在无序区[i，n-1]中选取最小记录，设其下标为 index；
   1.2 将最小记录与第 i 个记录交换；

下面是简单选择排序算法的 C++描述。

简单选择排序算法 SelectSort

```
void SelectSort(int r[ ], int n)
{
   for (i=0; i<n-1; i++)          //对 n 个记录进行 n-1 趟简单选择排序
   {
       index=i;
       for (j=i+1; j<n; j++)      //在无序区中选取最小记录
           if (r[j]<r[index]) index=j;
       if (index!=i) r[i]←→r[index];   //交换元素
   }
}
```

分析算法，有两层嵌套的 for 循环，所以，$T(n)=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=O(n^2)$。

⑵ 找出整型数组 A[n]中元素的最大值和次最大值。

【解答】算法的伪代码描述如下：

伪代码

1. 将前两个元素进行比较，较大者放到 max 中，较小者放到 nmax 中；
2. 从第 3 个元素开始直到最后一个元素依次取元素 A[i]，执行下列操作：
   2.1 如果 A[i]>max，则 A[i]为最大值，原来的最大值为次最大值；
   2.2 否则，如果 A[i]>nmax，则最大值不变，A[i]为次最大值；
3. 输出最大值 max，次最大值 nmax；

算法的 C++描述如下：

最大值和次最大值算法 Max_NextMax

```
void Max_NextMax(int A[ ], int n, int & max, int & nmax)
{
   if (A[0]>= A[1]) {
                    max=A[0];
                    nmax=A[1];
   }
   else {
        max=A[1];
        nmax=A[0];
   }
   for (i=2; i<n; i++)
        if (A[i]>=max) {
                       nmax=max;
                       max=A[i];
        }
        else if (A[i]>nmax) nmax=A[i];
   cout<<"最大值为："<<max<<"\n 次最大值为："<<nmax<<endl;
}
```

分析算法，只有一层循环，共执行 n-2 次，所以，T(n)=O(n)。

学习自测及答案

4．将下列函数按它们在 n 时的无穷大阶数，从小到大排列。

n, n-n3+7n5, nlogn, 2n/2, n3, log2n, n1/2+log2n, (3/2)n, n!, n2+log2n

【解答】log2n, n1/2+log2n, n, nlog2n, n2+log2n, n3, n-n3+7n5, 2n/2, (3/2)n, n!

5．试描述数据结构和抽象数据类型的概念与程序设计语言中数据类型概念的区别。

【解答】数据结构是指相互之间存在一定关系的数据元素的集合。而抽象数据类型是指一个数据结构以及定义在该结构上的一组操作。程序设计语言中的数据类型是一个值的集合和定义在这个值集上一组操作的总称。抽象数据类型可以看成是对数据类型的一种抽象。

6. 对下列用二元组表示的数据结构,试分别画出对应的逻辑结构图，并指出属于何种结构。

⑴ A=(D，R)， 其中 D={a1, a2, a3, a4}，R={ }

⑵ B=(D，R)， 其中 D={a, b, c, d, e, f}，R={,,,,}

⑶ C=(D，R)，其中 D={a，b，c，d，e，f}，R={,,,,,}

⑷ D=(D，R)， 其中 D={1, 2, 3, 4, 5, 6}，

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

【解答】⑴ 属于集合，其逻辑结构图如图 1-4(a)所示；⑵ 属于线性结构，其逻辑结构图如图 1-4(b)所示；⑶ 属于树结构，其逻辑结构图如图 1-4(c)所示；⑷ 属于图结构，其逻辑结构图如图 1-4(d)所示。

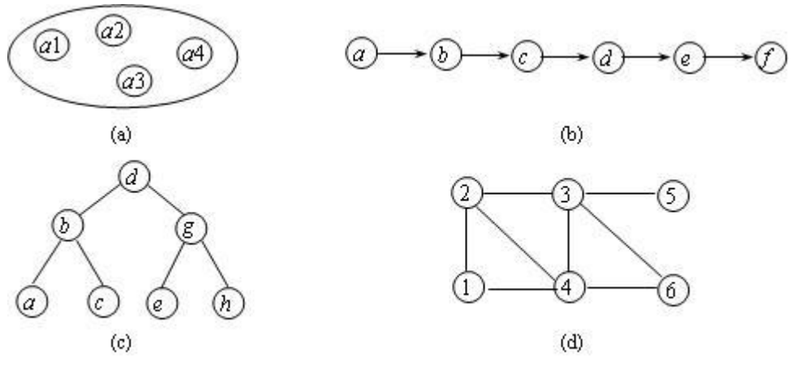


图 1-4 第 7 题对应的逻辑结构图

7. 求下列算法的时间复杂度。

```
count=0; x=1;
while (x {
x*=2;
count++;
}
return count;
```

【解答】O(log2n)

第 2 章 线性表

课后习题讲解

4．请说明顺序表和单链表各有何优缺点，并分析下列情况下，采用何种存储结构更好些。

⑴ 若线性表的总长度基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素。

⑵ 如果 n 个线性表同时并存，并且在处理过程中各表的长度会动态发生变化。

⑶ 描述一个城市的设计和规划。

【解答】顺序表的优点：① 无需为表示表中元素之间的逻辑关系而增加额外的存储空间；② 可以快速地存取表中任一位置的元素（即随机存取）。顺序表的缺点：① 插入和删除操作需移动大量元素；② 表的容量难以确定；③ 造成存储空间的“碎片”。

单链表的优点：① 不必事先知道线性表的长度；② 插入和删除元素时只需修改指针，不用移动元素。单链表的缺点：① 指针的结构性开销；② 存取表中任意元素不方便，只能进行顺序存取。

⑴ 应选用顺序存储结构。因为顺序表是随机存取结构，单链表是顺序存取结构。本题很少进行插入和删除操作，所以空间变化不大，且需要快速存取，所以应选用顺序存储结构。

⑵ 应选用链接存储结构。链表容易实现表容量的扩充，适合表的长度动态发生变化。

⑶ 应选用链接存储结构。因为一个城市的设计和规划涉及活动很多，需要经常修改、扩充和删除各种信息，才能适应不断发展的需要。而顺序表的插入、删除的效率低，故不合适。

5．算法设计

⑴ 设计一个时间复杂度为 O(n)的算法，实现将数组 A[n]中所有元素循环右移 k 个位置。

【解答】算法思想请参见主教材第一章思想火花。下面给出具体算法。

循环右移算法 Converse

```
void Converse(int A[ ], int n, int k)
{
    Reverse(A, 0, k-1);
    Reverse(A, k, n-1);
    Reverse(A, 0, n-1);
}
void Reverse(int A[ ], int from, int to)        //将数组 A 中元素从 from 到 to 逆置
{
    for (i=0; i<(to-from+1)/2; i++)
        A[from+i]←→A[to-i];        //交换元素
}
```

分析算法，第一次调用 Reverse 函数的时间复杂度为 O(k)，第二次调用 Reverse 函数的时间复杂度为 O(n-k)，第三次调用 Reverse 函数的时间复杂度为 O(n)，所以，总的时间复杂度为 O(n)。

⑵ 已知数组 A[n]中的元素为整型，设计算法将其调整为左右两部分，左边所有元素为奇数，右边所有元素为偶数，并要求算法的时间复杂度为 O(n)。

【解答】从数组的两端向中间比较，设置两个变量 i 和 j，初始时 i=0，j=n-1，若 A[i]为偶数并且 A[j]为奇数，则将 A[i]与 A[j]交换。具体算法如下：

数组奇偶调整算法 Adjust

```
void Adjust(int A[ ], n)
{
  i=0; j=n-1;
  while (i<j)
  {
    while (A[i] % 2!=0) i++;
    while (A[j] % 2==0) j--
    if (i<j) A[i]←→A[j];
  }
}
```

分析算法，两层循环将数组扫描一遍，所以，时间复杂度为 O(n)。

⑶ 试编写在无头结点的单链表上实现线性表的插入操作的算法，并和带头结点的单链表上的插入操作的实现进行比较。

【解答】参见 2.2.3。

⑷ 试分别以顺序表和单链表作存储结构，各写一实现线性表就地逆置的算法。

【解答】顺序表的逆置，即是将对称元素交换，设顺序表的长度为 length，则将表中第 i 个元素与第 length-i-1 个元素相交换。具体算法如下：

顺序表逆置算法 Reverse

```
template <class T>
void Reverse(T data[ ], int length)
{
    for (i=0; i<=length/2; i++)
    {
        temp=data[i];
        data[i]=data[length-i-1];
        data[length-i-1]=temp;
    }
}
```

单链表的逆置请参见 2.2.4 算法 2-4 和算法 2-6。

⑸ 假设在长度大于 1 的循环链表中，即无头结点也无头指针，s 为指向链表中某个结点的指针，试编写算法删除结点 s 的前趋结点。

【解答】利用单循环链表的特点，通过指针 s 可找到其前驱结点 r 以及 r 的前驱结点 p，然后将结点 r 删除，如图 2-11 所示，具体算法如下：

循环链表删除算法 Del

```
template <class T>
void Del(Node<T> *s)
{
  p=s;              //工作指针 p 初始化，查找 s 的前驱结点的前驱结点，用 p 指示
  while (p->next->next!=s)
     p=p->next;
  r=p->next;        //r 为 p 的前驱结点，q 为 r 的前驱结点
  p->next=s;        //删除 r 所指结点
  delete r;
}
```

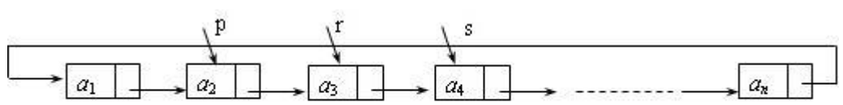


图 2-11　删除结点 s 的前驱结点操作示意图

⑹ 已知一单链表中的数据元素含有三类字符：字母、数字和其他字符。试编写算法，构造三个循环链表，使每个循环链表中只含同一类字符。

【解答】在单链表 A 中依次取元素，若取出的元素是字母，把它插入到字母链表 B 中，若取出的元素是数字，则把它插

入到数字链表 D 中，直到链表的尾部，这样表 B，D，A 中分别存放字母、数字和其他字符。具体算法如下：

单链表拆分算法 Adjust

```
template <class T>
void Adjust(Node<T> *A, Node<int> *D, Node<char> *B)
{
  D=new Node<int>; D->next=D;      //创建空循环链表 D，存放数字
  B=new Node<char>; B->next=B;     //创建空循环链表 B，存放字符
  p=A; q=p->next;                  //工作指针 q 初始化
  while (q)
  {
    if (('A'<=q->data)&&(q->data>='Z')||('a' <=q->data)&&(q->data>='z')) {
                    p->next=q->next;
                    q->next=B->next;
                    B->next=q;      //采用头插法插在循环链表 B 的头结点的后面
    }
    else if (('0'<=q->data)&&(q->data>='9')) {
                    p->next=q->next;
                    q->next=D->next;
                    D->next=q;    //采用头插法插在循环链表 D 的头结点的后面
        }
        else p=q;
    q=p->next;
  }
  p->next=A; R=A;   //将链表 A 构造为循环链表，为除字母和数字的其他字符
}
```

⑺ 设单链表以非递减有序排列，设计算法实现在单链表中删去值相同的多余结点。

【解答】从头到尾扫描单链表，若当前结点的元素值与后继结点的元素值不相等，则指针后移；否则删除该后继结点。具体算法如下：

单链表删除相同值算法 Purge

```
void Purge(Node<T> *first)
{
    p=first->next;
    while (p->next)
      if (p->data= =p->next->data) {
                                  q=p->next;
                                  p->next=q->next;
                                  delete q;
        }
        else p=p->next;
}
```

⑻ 判断带头结点的双循环链表是否对称。

【解答】设工作指针 p 和 q 分别指向循环双链表的开始结点和终端结点，若结点 p 和结点 q 的数据域相等，则工作指针 p 后移，工作指针 q 前移，直到指针 p 和指针 q 指向同一结点（循环双链表中结点个数为奇数），或结点 q 成为结点 p 的前驱（循环双链表中结点个数为偶数）。如图 2-12 所示。

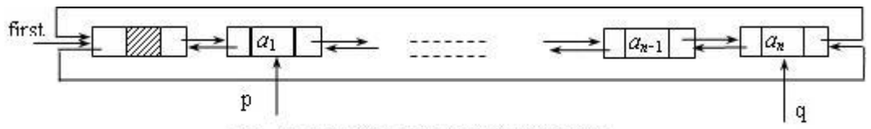


图 2-12　判断循环双链表对称的操作示意图

判断双链表对称算法 Equal

```
template <class T>
struct DulNode
{
  T data;
  DulNode<T> *prior, *next;
};
template <class T>
bool Equal (DulNode<T> *first)
{
  p=first->next; q=first->prior;
  while (p!=q && p->prior!=q)
      if (p->data= =q->data) {
                                  p=p->next;    //工作指针 p 后移
                                  q=q->prior;   //工作指针 q 前移
      }
      else return 0;
  return 1;
}
```

学习自测及答案

7．设 A 是一个线性表（a1, a2, …, an），采用顺序存储结构，则在等概率的前提下，平均每插入一个元素需要移动的元素个数为多少？若元素插在 ai 与 ai+1 之间（1≤i≤n）的概率为 $\frac{n-i}{n(n+1)/2}$，则平均每插入一个元素所要移动的元素个数又是多少？

【解答】

$\sum_{i=1}^{n+1}(n-i+1)=\frac{n}{2}$，

$\sum_{i=1}^{n}\frac{(n-i)^2}{n(n+1)/2}=(2n+1)/3$。

8．线性表存放在整型数组 A[arrsize]的前 elenum 个单元中，且递增有序。编写算法，将元素 x 插入到线性表的适当位置上，以保持线性表的有序性，并且分析算法的时间复杂度。

【解答】本题是在一个递增有序表中插入元素 x，基本思路是从有序表的尾部开始依次取元素与 x 比较，若大于 x，此元素后移一位，再取它前面一个元素重复上述步骤；否则，找到插入位置，将 x 插入。具体算法如下：

有序顺序表插入算法 Insert

```
const int arrsize =100;
class SeqList
{
private:
   int data[arrsize];
   int elenum;
public:
   Insert(int x);
   …
};
void SeqList::Insert(int x)
{
    if (elenum= =arrsize) throw "overflow";
    else {
         i=elenum-1;
         while (i>=1 && x<data[i])
         {
             data[i+1]=data[i];
             i++;
         }
         data[i+1]=x;
    }
}
```

9. 已知单链表中各结点的元素值为整型且递增有序，设计算法删除链表中所有大于 mink 且小于 maxk 的所有元素，并释放被删结点的存储空间。

【解答】因为是在有序单链表上的操作，所以，要充分利用其有序性。在单链表中查找第一个大于 mink 的结点和第一个小于 maxk 的结点，再将二者间的所有结点删除。

有序链表删除算法 DeleteBetween

```
template <class T>
void DeleteBetween(Node<T> *first, int mink, int maxk)
{
    p=first;
    while (p->next && p->next->data<=mink)
        p=p->next;
    if (p->next) {
                 q=p->next;
                 while (q->data<maxk)
                 {
                     u=q->next;
                     p->next=q->next;
                     delete q;
                     q=u;
                 }
    }
}
```

10. 设单循环链表 L1，对其遍历的结果是：x1, x2, x3,…, xn-1, xn。请将该循环链表拆成两个单循环链表 L1 和 L2，使得 L1 中含有原 L1 表中序号为奇数的结点且遍历结果为：x1, x3,… ；L2 中含有原 L1 表中序号为偶数的结点且遍历结果为：… , x4, x2。

【解答】算法如下：

循环链表拆分算法 DePatch

```
template <class T>
Node<T> *DePatch(Node<T> *L1)
{
    L2=new Node<T>; L2->next=L2;
    q=L1->next; L1->next=L1;
    p=L1; i=1;
    while (q!=L1)
    {
        if (i % 2==1) {    //应用尾插法
                    p->next=q; p=q; p->next=L1;
                    q=q->next; i++;

        }
        else {
            L2->next=q; u=q->next; q->next=L2->next;
            q=u; i++;
        }
    }
}
```

第 3 章 特殊线性表——栈、队列和串

4. 设有一个栈，元素进栈的次序为 A，B，C，D，E，能否得到如下出栈序列，若能，请写出操作序列，若不能，请说明原因。

⑴ C，E，A，B，D

⑵ C，B，A，D，E

【解答】⑴不能，因为在 C、E 出栈的情况下，A 一定在栈中，而且在 B 的下面，不可能先于 B 出栈。⑵可以，设 I 为进栈操作， O 为入栈操作，则其操作序列为 IIIOOOIOIO。

5. 举例说明顺序队列的“假溢出”现象。

【解答】假设有一个顺序队列，如图 3-6 所示，队尾指针 rear=4，队头指针 front=1，如果再有元素入队，就会产生“上溢”，此时的“上溢”又称为“假溢出”，因为队列并不是真的溢出了，存储队列的数组中还有 2 个存储单元空闲，其下标分别为 0 和 1。

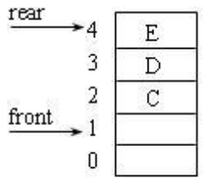


图 3-6 顺序队列的假溢出

6. 在操作序列 push(1)、push(2)、pop、push(5)、push(7)、pop、push(6)之后，栈顶元素和栈底元素分别是什么？（push(k)表示整数 k 入栈，pop 表示栈顶元素出栈。）

【解答】栈顶元素为 6，栈底元素为 1。其执行过程如图 3-7 所示。

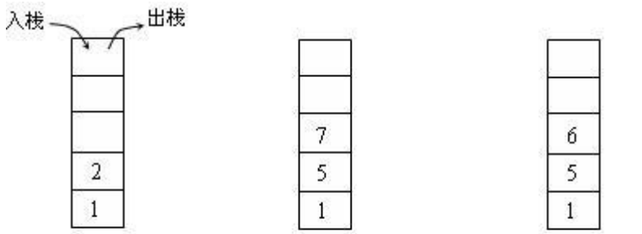


图 3-7　栈的执行过程示意图

7．在操作序列 EnQueue(1)、 EnQueue(3)、 DeQueue、EnQueue(5)、EnQueue(7)、DeQueue、EnQueue(9)之后，队头元素和队尾元素分别是什么？（EnQueue(k)表示整数 k 入队，DeQueue 表示队头元素出队）。
【解答】队头元素为 5，队尾元素为 9。其执行过程如图 3-8 所示。

8．空串和空格串有何区别？串中的空格符有何意义？空串在串处理中有何作用？
【解答】不含任何字符的串称为空串，其长度为零。仅含空格的串称为空格串，它的长度为串中空格符的个数。串中的空格符可用来分隔一般的字符，便于人们识别和阅读，但计算串长时应包括这些空格符。空串在串处理中可作为任意串的子串。
9．算法设计
⑴ 假设以不带头结点的循环链表表示队列，并且只设一个指针指向队尾结点，但不设头指针。试设计相应的入队和出队的算法。
【解答】出队操作是在循环链表的头部进行，相当于删除开始结点，而入队操作是在循环链表的尾部进行，相当于在终端结点之后插入一个结点。由于循环链表不带头结点，需要处理空表的特殊情况。
入队算法如下：

循环队列入队算法 Enqueue
```
template <class T>
void Enqueue(Node<T> *rear, T x)
{
    s=new Node<T>;
    s->data=x;
    if (rear==NULL) {                    //处理空表的特殊情况
                    rear=s;
                    rear->next=s;
    }
    else {                               //处理除空表以外的一般情况
         s->next=rear->next;
         rear->next=s;
         rear=s;
    }
}
```

出队算法如下：

循环队列出队算法 Dequeue
```
template <class T>
T Dequeue(Node<T> *rear)
{
    if (rear==NULL) throw "underflow";   //判断表空
       else {
            s=rear->next;
            if (s==rear) rear=NULL;      //链表中只有一个结点
                 else rear->next=s->next;
            delete s;
       }
}
```

⑵ 设顺序栈 S 中有 2n 个元素，从栈顶到栈底的元素依次为 a2n，a2n-1，…，a1，要求通过一个循环队列重新排列栈中元素，使得从栈顶到栈底的元素依次 为 a2n，a2n-2，…，a2，a2n-1，a2n-3，…，a1，请设计算法实现该操作，要求空间复杂度和时间复杂度均为 O(n)。
【解答】操作步骤为：
① 将所有元素出栈并入队；
② 依次将队列元素出队，如果是偶数结点，则再入队，如果是奇数结点，则入栈；
③ 将奇数结点出栈并入队；
④ 将偶数结点出队并入栈；
⑤ 将所有元素出栈并入队；
⑥ 将所有元素出队并入栈即为所求。
⑶ 用顺序存储结构存储串 S，编写算法删除 S 中第 i 个字符开始的连续 j 个字符。
【解答】先判断串 S 中要删除的内容是否存在，若存在，则将第 i+j-1 之后的字符前移 j 个位置。算法如下：

串删除算法 Del
```
void Del(char S[ ], int i, int j)     //数组 0 号单元存放串的长度
{
  if (i+j<S[0]) {
                for (k=i; k<S[0]; k++)
                    S[k]=S[k+j];
                S[0]=S[0]-j;
  }
  else cout<<"参数非法";
}
```

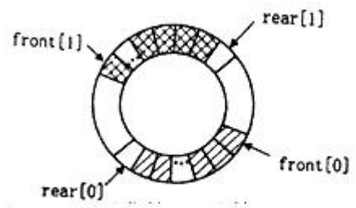
⑷ 对于采用顺序存储结构的串 S，编写一个函数删除其值等于 ch 的所有字符。
【解答】从后向前删除值为 ch 的所有元素，这样所有移动的元素中没有值为 ch 的元素，能减少移动元素的次数，提高算法的效率。算法如下：

串删除算法 Del
```
void Del (char S[ ], char ch )   //数组的 0 号单元存放串的长度
{
    for (i=S[0]; i>0; i--)
      if (S[i]==ch) {
                    for (j=i+1; j<=S[0]; j++)
                       S[j-1]=S[j];
                    S[0]--;
      }
}
```

⑸ 对串的模式匹配 KMP 算法设计求模式滑动位置的 next 函数。
【解答】参见 3.2.5

习题补充

20. void AE(Stack& S)
{
InitStack(S);
Push(S,30);
Push(S,40);
Push(S,50);
int x=Pop(S)+2*Pop(S);
Push(S,x);
int i,a[4]={5,8,12,15};
for(i=0;i<4;i++) Push(S,a[i]);
while(!StackEmpty(S)) cout<<Pop(S)<<' ';
}
该算法被调用后得到的输出结果为：



21.假设两个队列共享一个循环向量空间（参见下图），其类型 Queue2 定义如下：
typedef struct{
DataType data[MAXSIZE];
int front[2],rear[2];
}Queue2;
对于 i=0 或 1，front[i],rear[i]分别为第 i 个队列的头指针和尾指针。请对以下算法填空，实现第 i 个对列的入队操作。
int EnQueue(Queue2 *Q, int i, DataType x)
{//若第 i 个队列不满，则元素 x 入队列，并返回 1；否则返回 0。
if(i<0||i>1) return 0;
if(Qrear[i]==Qfront[_____(1)_____] return 0;
Q—>data[_____(2)_____]=x;
Qrear[i]=_____(3)_____;
return 1;
}
22. 阅读下面程序，指出其算法的功能
#include "stack.h"
int BaseTrans(int N,int B) {
int i,result=0;Stack<int>S;
while(N!=0){i=N%B;N=N/B;S.Push(i);}
while(!S.IsEmpty()){i=S.GetTop();S.Pop();
result=result*10+i;}
return result;
}

学习自测及答案

7．如果进栈序列为 A、B、C、D，则可能的出栈序列是什么？
答：共 14 种，分别是：ABCD，ABDC，ACBD，ACDB，ADCB，BACD，BADC，BCAD，BCDA，BDCA，CBAD，CBDA，CDBA，DCBA
8．简述队列和栈这两种数据结构的相同点和不同点。
【解答】相同点：它们都是插入和删除操作的位置受限制的线性表。
不同点：栈是限定仅在表尾进行插入和删除的线性表，是后进先出的线性表，而队列是限定在表的一端进行插入，在另一端进行删除的线性表，是先进先出的线性表。
9. 利用两个栈 S1 和 S2 模拟一个队列，如何利用栈的运算实现队列的插入和删除操作，请简述算法思想。
【解答】利用两个栈 S1 和 S2 模拟一个队列，当需要向队列中插入一个元素时，用 S1 来存放已输入的元素，即通过向栈 S1 执行入栈操作来实现；当需要从队列 中删除元素时，则将 S1 中元素全部送入到 S2 中，再从 S2 中删除栈顶元素，最后再将 S2 中元素全部送入到 S1 中；判断队空的条件是：栈 S1 和 S2 同时为 空。
10. 设计算法把一个十进制整数转换为二至九进制之间的任一进制数输出。
【解答】算法基于原理：N=(N div d)×d + N mod d （div 为整除运算，mod 为求余运算）。

进制转换算法 Decimaltor

```
void Decimaltor (int num, int r)
{
    top=-1;    //假设采用顺序栈
    while (num!=0)
    {
        k=num % r;
        S[++top]=k;
        num=num/r;
    }
    while (top!=-1)
        cout<<S[top--];
}
```

11．假设一个算术表达式中可以包含三种括号：圆括号“（”和“）”，方括号“[”和“]”以及花括号“{”和“}”，且这三种括号可按任意的次序嵌套使用。编写算法判断给定表达式中所含括号是否配对出现。
【解答】假设表达式已存入字符数组 A[n]中，具体算法如下：

括号匹配算法 Prool

```
void Prool(char A[ ], int n)
{
  top=-1; i=0; flag =1;
  while (i<n && flag)
  {
      if  (A[i]='('||A[i]='['||A[i]='{') S[++top]=A[i++];
      else {
            switch A[i]
            {
              case ')': if (top==-1||S[top--]!='(') flag =0; break;
              case ']': if (top==-1||S[top--]!='[') flag=0; break;
              case '}': if (top==-1||S[top--]!='{') flag=0; break;
            }
      }
      i++;
  }
}
```

第 4 章 广义线性表——多维数组和广义表

课后习题讲解

4．一个稀疏矩阵如图 4-4 所示，写出对应的三元组顺序表和十字链表存储表示。

$$\begin{pmatrix} 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

图 4-4　稀疏矩阵

【解答】对应的三元组顺序表如图 4-5 所示，十字链表如图 4-6 所示。

下标	行号	列号	非零元素
0	1	3	2
1	2	1	3
2	3	3	-1
3	3	4	5
	4（行数）		
	4（列数）		
	4（非零元个数）		

图 4-5　稀疏矩阵的三元组顺序表

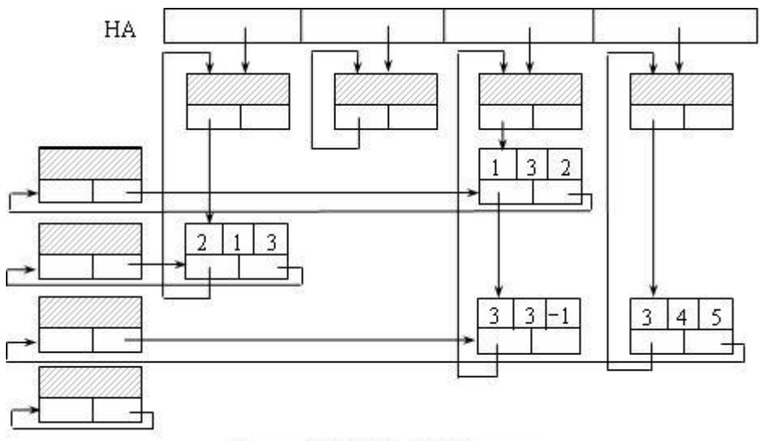


图 4-6　稀疏矩阵的十字链表

5．已知 A 为稀疏矩阵，试从空间和时间角度比较采用二维数组和三元组顺序表两种不同的存储结构完成求 运算的优缺点。

【解答】设稀疏矩阵为 m 行 n 列，如果采用二维数组存储，其空间复杂度为 O (m×n)；因为要将所有的矩阵元素累加起来，所以，需要用一个两层的嵌套循环， 其时间复杂度亦为 O (m×n)。如果采用三元组顺序表进行压缩存储，假设矩阵中有 t 个非零元素，其空间复杂度为 O (t)，将所有的矩阵元素累加起来只需将 三元组顺序表扫描一遍，其时间复杂度亦为 O (t)。当 t << m×n 时，采用三元组顺序表存储可获得较好的时、空性能。

6．设某单位职工工资表 ST 由“工资”、“扣除”和“实发金额”三项组成，其中工资项包括“基本工资”、“津贴”和“奖金”，扣除项包括“水”、“电”和“煤气” 。

⑴ 请用广义表形式表示所描述的工资表 ST，并用表头和表尾求表中的“奖金”项；

⑵ 画出该工资表 ST 的存储结构。

【解答】⑴ ST=((基本工资，津贴，奖金)，(水，电，煤气)，实发金额)

Head(Tail(Tail(Head(ST))))=奖金

⑵ 工资表 ST 的头尾表示法如图 4-7 所示。

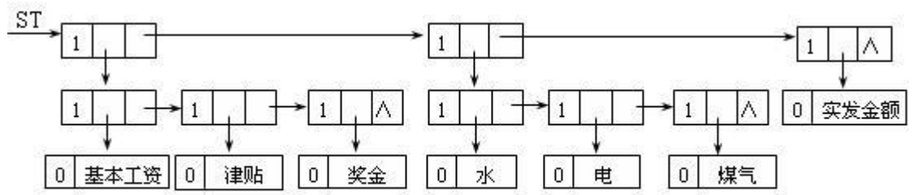


图 4-7　工资表的存储示意图

7．若在矩阵 A 中存在一个元素 ai,j（0≤i≤n-1，0≤j≤m-1），该元素是第 i 行元素中最小值且又是第 j 列元素中最大值，则称此元素为该矩阵的一个马鞍点。假设以二维数组存储矩阵 A，试设计一个求该矩阵所有马鞍点的算法，并分析最坏情况下的时间复杂度。

【解答】在矩阵中逐行寻找该行中的最小值，然后对其所在的列寻找最大值，如果该列上的最大值与该行上的最小值相等，则说明该元素是鞍点，将它所在行号和列号输出。

具体算法如下：

马鞍点算法 Andian

```
void Andian(int a[ ][ ], int m, int n)
{
  for (i=0; i<n; i++)
  {
     d=a[i][0]; k=0;     //d 为第 i 行中的最小值
     for (j=1; j<m; j++)
       if (a[i][j]<d) {
                        d=a[i][j];
                        k=j;
       }     // a[i][k]为第 i 行最小值
     for (j=0; j<n; j++)
       if (a[j][k]>d) break;
     if (j= =n) cout<<"输出鞍点"<<i<<k<< a[i][k];
  }
}
```

分析算法，外层 for 循环共执行 n 次，内层第一个 for 循环执行 m 次，第二个 for 循环最坏情况下执行 n 次，所以，最坏情况下的时间复杂度为 O (mn+n2)。

学习自测及答案

6．设有三对角矩阵 An×n（行、列下标均从 0 开始），将其三条对角线上的元素逐行存于数组 B[3n-2]中，使得 B[k]=aij 求：

⑴ 用 i, j 表示 k 的下标变换公式；

⑵ 用 k 表示 i, j 的下标变换公式。

【解答】⑴ 要求 i, j 表示 k 的下标变换公式，就是要求在 k 之前已经存储了多少个非零元素，这些非零元素的个数就是 k 的值。元素 aij 求所在的行为 i，列为 j，则在其前面的非零元素的个数是；k=2 + 3(i－1)+(j－i + 1)= 2i+ j。

⑵ 因为 k 和 i, j 之间是一一对应的关系，k+1 是当前非零元素的个数，整除即为其所在行号，取余表示当前行中第几个非零元素，加上前面零元素所在列数就是当前列号，即：

$$\begin{cases} i=(k+1)/3 \\ j=(k+1)\%3+(k+1)/3-1 \end{cases}$$

7．已知两个 n×n 的对称矩阵按压缩存储方法存储在已维数组 A 和 B 中，编写算法计算对称矩阵的乘积。

【解答】对称矩阵采用压缩存储，乘积矩阵也采用压缩存储。注意矩阵元素的表示方法。

矩阵乘积算法 Mul

```
void Mul(int A[ ], int B[ ], int C[ ], int n)
{
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
    {
      mi=max(i, j); mj=min(i, j);
      x=mi*(mi-1)/2+mj-1;    //计算矩阵元素 C[i][j]压缩后的存储地址
      C[x]=0;
      for (k=0; k<n; k++)
      {
        u1=max(i, k); v1=min(i, k);
        u2=max(k, j); v2=min(k, j);
        w1=u1*(u1-1)/2+v1-1;     //计算 A[i][k]的存储地址
        w2=u2*(u2-1)/2+v2-1;     //计算 B[k][j]的存储地址
        c[x]=A[w1]*B[w2];
      }
    }
}
```

第 5 章 树和二叉树

课后习题讲解

4．证明：对任一满二叉树，其分枝数 B＝2(n0-1) 。（其中，n0 为终端结点数）

【解答】因为在满二叉树中没有度为 1 的结点，所以有：

n=n0+n2

设 B 为树中分枝数，则

n=B+1

所以

B=n0 +n2-1

再由二叉树性质：

n0=n2+1

代入上式有：

B=n0+n0-1-1=2(n0-1)

5．证明：已知一棵二叉树的前序序列和中序序列，则可唯一确定该二叉树。

【解答】证明采用归纳法。

设二叉树的前序遍历序列为 a1a2a3… an，中序遍历序列为 b1b2b3… bn。

当 n=1 时，前序遍历序列为 a1，中序遍历序列为 b1，二叉树只有一个根结点，所以，a1= b1，可以唯一确定该二叉树；

假设当 n<=k 时，前序遍历序列 a1a2a3… ak 和中序遍历序列 b1b2b3… bk 可唯一确定该二叉树，下面证明当 n=k+1 时，前序遍历序列 a1a2a3… akak+1 和中序遍历序列 b1b2b3… bk bk+1 可唯一确定一棵二叉树。

在前序遍历序列中第一个访问的一定是根结点，即二叉树的根结点是 a1，在中序遍历序列中查找值为 a1 的结点，假设为 bi，则 a1=bi 且 b1b2… bi-1 是对根结点 a1 的左子树进行中序遍历的结果，前序遍历序列 a2a3… ai 是对根结点 a1 的左子树进行前序遍历的结果，由归纳假设，前序遍历序列 a2a3… ai 和中序遍历序列 b1b2… bi-1 唯一确定了根结点的左子树，同样可证前序遍历序列 ai+1ai+2… ak+1 和中序遍历序列 bi+1bi+2… bk+1 唯一确定了根结点的右子树。

6．已知一棵度为 m 的树中有：n1 个度为 1 的结点，n2 个度为 2 的结点，……，nm 个度为 m 的结点，问该树中共有多少个叶子结点？

【解答】设该树的总结点数为 n，则

n=n0+n1+n2+……+nm

又：

n=分枝数+1=0×n0+1×n1+2×n2+……+m×nm+1

由上述两式可得：

n0= n2+2n3+……+(m-1)nm+1

7．已知二叉树的中序和后序序列分别为 CBEDAFIGH 和 CEDBIFHGA，试构造该二叉树。

【解答】二叉树的构造过程如图 5-12 所示。

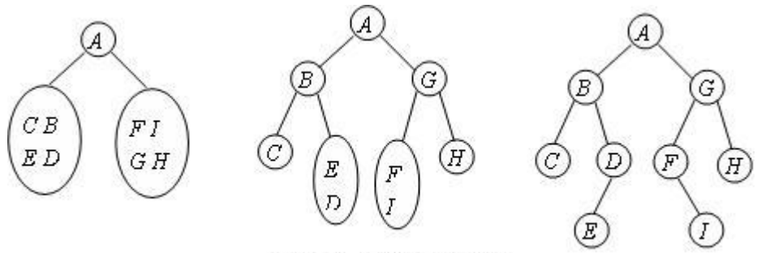


图 5-12　构造二叉树的过程

8．对给定的一组权值 W＝（5，2，9，11，8，3，7），试构造相应的哈夫曼树，并计算它的带权路径长度。

【解答】构造的哈夫曼树如图 5-13 所示。

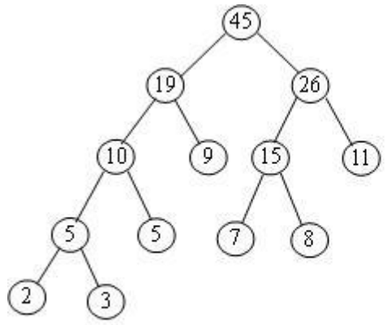


图 5-13　构造的哈夫曼树及带权路径长度

树的带权路径长度为：

WPL=2×4+3×4+5×3+7×3+8×3+9×2+11×2

=120

9．已知某字符串 S 中共有 8 种字符，各种字符分别出现 2 次、1 次、4 次、5 次、7 次、3 次、4 次和 9 次，对该字符串用[0，1]进行前缀编码，问该字符串的编码至少有多少位。

【解答】以各字符出现的次数作为叶子结点的权值构造的哈夫曼编码树如图 5-14 所示。其带权路径长度=2×5+1×5+3×4+5×3+9×2+4×3+4×3+7×2=98，所以，该字符串的编码长度至少为 98 位。

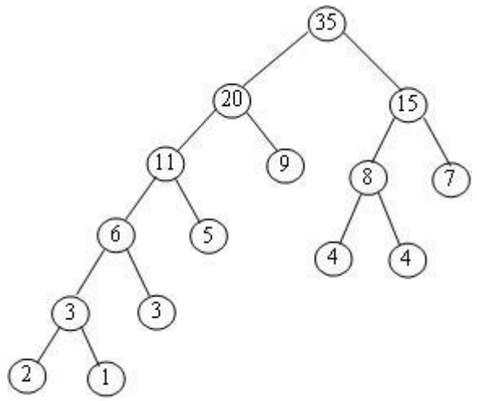


图 5-14　哈夫曼编码树

10．算法设计

⑴ 设计算法求二叉树的结点个数。

【解答】本算法不是要打印每个结点的值，而是求出结点的个数。所以可将遍历算法中的“访问”操作改为“计数操作”，将结点的数目累加到一个全局变量中，每个结点累加一次即完成了结点个数的求解。

具体算法如下：

求二叉树结点个数算法 Count

```
void Count(BiNode *root)    //n 为全局量并已初始化为 0
{
  if (root) {
          Count(root->lchild);
          n++;
          Count(root->rchild);
  }
}
```

⑵ 设计算法按前序次序打印二叉树中的叶子结点。

【解答】本算法的要求与前序遍历算法既有相同之处，又有不同之处。相同之处是打印次序均为前序，不同之处是此处不是打印每个结点的值，而是打印出其中的叶子结点，即为有条件打印。为此，将前序遍历算法中的访问操作改为条件打印即可。算法如下：

打印叶子结点算法 PreOrder

```
void PreOrder(BiNode *root)
{
  if (root) {
          if (!root->lchild && !root->rchild) cout<<root->data;
           PreOrder(root->lchild);
           PreOrder(root->rchild);
  }
}
```

⑶ 设计算法求二叉树的深度。

【解答】当二叉树为空时，深度为 0；若二叉树不为空，深度应是其左右子树深度的最大值加 1，而其左右子树深度的求解又可通过递归调用本算法来完成。具体算法如下：

求二叉树深度算法 Depth

```
int Depth(BiNode *root)
{
    if (!root) return 0;
      else {
          hl= Depth(root->lchild);
          hr= Depth(root ->rchild);
          return max(hl, hr)+1;
    }
}
```

⑷ 编写算法，要求输出二叉树后序遍历序列的逆序。

【解答】要想得到后序的逆序，只要按照后序遍历相反的顺序即可，即先访问根结点，再遍历根结点的右子树，最后遍历根结点的左子树。注意和前序遍历的区别，具体算法如下：

后序的逆序遍历算法 PostOrder

```
void PostOrder(BiNode *root)
{
  if (root) {
          cout<<root->data;
          PostOrder(root->rchild);
          PostOrder(root->lchild);
  }
}
```

⑸ 以二叉链表为存储结构，编写算法求二叉树中结点 x 的双亲。

【解答】对二叉链表进行遍历，在遍历的过程中查找结点 x 并记载其双亲。具体算法如下：

查找某结点的双亲算法 Parent

```
BiNode *Parent(BiNode *root, T x)   //p 是全局量，初值为空
{
  if (root) {
        if (root->data==x) return p;
          else {
              p=root;
              Parent(root->lchild, x);
              Parent(root->rchild, x);
          }
  }
}
```

⑹ 以二叉链表为存储结构，在二叉树中删除以值 x 为根结点的子树。

【解答】对二叉链表进行遍历，在遍历的过程中查找结点 x 并记载其双亲，然后将结点 x 的双亲结点中指向结点 x 的指针置空。具体算法如下：

删除结点 x 算法 Delete

```
void Delete(BiNode *root, T x)   //p 是全局量，初值为空
{
  if (root) {
        if (root->data==x)
          if (!p) root=NULL;
            else if (p->lchild==root)  p->lchild=NULL;
                  else p->rchild=NULL;
        else {
              p=root;
              Delete(root->lchild, x);
              Delete(root->rchild, x);
        }
  }
}
```

⑺ 一棵具有 n 个结点的二叉树采用顺序存储结构，编写算法对该二叉树进行前序遍历。

【解答】按照题目要求，设置一个工作栈以完成对该树的非递归算法，思路如下：

① 每访问一个结点，将此结点压栈，查看此结点是否有左子树，若有，访问左子树，重复执行该过程直到左子树为空。

② 从栈弹出一个结点，如果此结点有右子树，访问右子树执行步骤①，否则重复执行步骤②。

具体算法如下：

顺序存储的前序遍历算法 Exchange

```
template <class T>
void PreOrder(T A[ ], int n)
{
    top=-1;   //栈初始化，采用顺序栈并假定不会发生溢出
    i=1; cout<<A[i]; S[++top]=i;
    j=2*i;
    while (top!=-1)
    {
        while (j<=n)
        {
            cout<<A[j];
            S[++top]=j;
            i=j; j=2*i;
        }
        i=S[top--];
        i=2*i+1;
    }
}
```

(8) 编写算法交换二叉树中所有结点的左右子树。

【解答】对二叉树进行后序遍历，在遍历过程中访问某结点时交换该结点的左右子树。

具体算法如下：

交换左右子树算法 Exchange

```
void Exchange(BiNode *root)
{
  if (root) {
            Exchange(root->lchild);
            Exchange(root->rchild);
            root->lchild←→root->rchild;   //交换左右子树
    }
}
```

(9) 以孩子兄弟表示法做存储结构，求树中结点 x 的第 i 个孩子。

【解答】先在链表中进行遍历，在遍历过程中查找值等于 x 的结点，然后由此结点的最左孩子域 firstchild 找到值为 x 结点的第一个孩子，再沿右兄弟域 rightsib 找到值为 x 结点的第 i 个孩子并返回指向这个孩子的指针。

树的孩子兄弟表示法中的结点结构定义如下：

```
template
struct TNode
{
T data;
TNode *firstchild, *rightsib;
};
```

具体算法如下：

查找第 i 个孩子算法 Search

```
template <class T>
TNode *Search(TNode *root, T x, int i)
{
    if (root->data==x) {
                        j=1;
                        p=root->firstchild;
                        while (p!=NULL && j<i)
                        {
                            j++;
                            p=p->rightsib;
                        }
                        if (p) return p;
                          else return NULL;
    }
    Search(root->firstchild, x, i);
    Search(root->rightsib, x, i);
}
```

学习自测及答案

11．现有按前序遍历二叉树的结果 ABC，问有哪几种不同的二叉树可以得到这一结果？

【解答】共有 5 种二叉树可以得到这一结果，如图 5-15 所示。

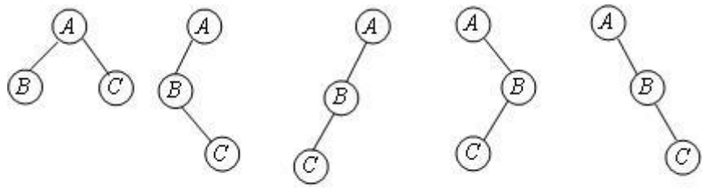


图 5-15　前序序列为 ABC 的二叉树

12．试找出分别满足下列条件的所有二叉树：

(1) 前序序列和中序序列相同。

(2) 中序序列和后序序列相同。

(3) 前序序列和后序序列相同。

【解答】 (1) 空二叉树、只有一个根结点的二叉树和右斜树。

(2) 空二叉树、只有一个根结点的二叉树和左斜树。

(3) 空二叉树、只有一个根结点的二叉树

13．将下面图 5-16 所示的树转换为二叉树，图 5-17 所示的二叉树转换为树或森林。

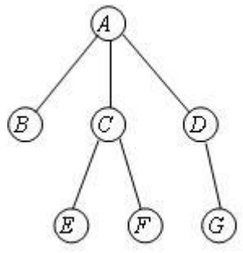


图 5-16　一棵树

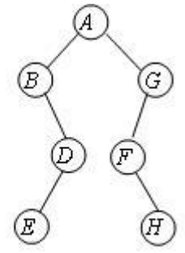


图 5-17　一棵二叉树

【解答】图 5-16 所示树转换的二叉树如图 5-18 所示，图 5-17 所示二叉树转换的森林如图 5-19 所示。

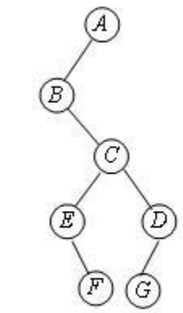


图 5-18　转换后的二叉树

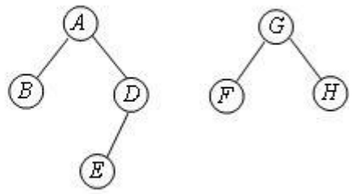


图 5-19　转换后的森林

14．以孩子兄弟表示法作为存储结构，编写算法求树的深度。

【解答】采用递归算法实现。若树为空树，则其深度为 0，否则其深度等于第一棵子树的深度+1 和兄弟子树的深度中的较大者。具体算法如下：

求树的深度算法 Depth

```
int Depth(TNode *root)
{
  if (!root) return 0;
     else {
          h1=Depth(root->firstchild);
          h2=Depth(root->rightsib);
          return max(h1+1, h2);
     }
}
```

15．设计算法，判断一棵二叉树是否为完全二叉树。

【解答】根据完全二叉树的定义可知，对完全二叉树按照从上到下、从左到右的次序（即层序）遍历应该满足：

⑴ 若某结点没有左孩子，则其一定没有右孩子；

⑵ 若某结点无右孩子，则其所有后继结点一定无孩子。

若有一结点不满足其中任意一条，则该二叉树就一定不是完全二叉树。因此可采用按层次遍历二叉树的方法依次对每个结点进行判断是否满足上述两个条件。为此，需设两个标志变量 BJ 和 CM，其中 BJ 表示已扫描过的结点是否均有左右孩子，CM 存放局部判断结果及最后的结果。

具体算法如下：

判断完全二叉树算法 ComBiTree

```
bool ComBiTree(BiNode *root)
{
  front=rear=-1;   //队列初始化，采用顺序队列并假定不会发生溢出
  BJ=1; CM=1;
  if (root) {
            Q[++rear]=root;
            while (front!=rear)
            {
                p=Q[++front];
                if (!p->lchild) {
                                BJ=0;
                                if (p->rchild) CM=0;
                }
                else {
                      CM=BJ;
                      Q[++rear]=p->lchild;
                      if (!p->rchild) BJ=0;
                          else Q[++front]=p->rchild;
                }
            }
   }
   return CM;
}
```

习题补充

18. 假定一棵二叉树广义表表示为 a(b(c),d(e,f))，分别写出对它进行先序、中序、后序、按层遍历的结果。

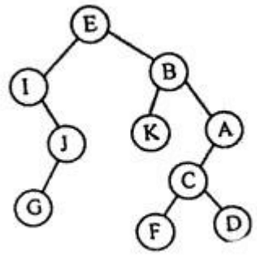
先序：

中序：

后序：

按层：

19. 请画出与下列二叉树对应的森林。



20.已知二叉树的存储结构是二叉链表，阅读下面的算法：

```
typedef struct node{
DataType data;
struct node *next;
}ListNode;
typedef ListNode * Linklist;
LinkList Leafhead=NULL;
void Inorder(BiTree T)
{ LinkList s;
if(T)
```

```
{ Inorder(Tlchild);
if(!Tlchild) &&(!Trchild))
{ s=(ListNode *)malloc(sizeof(ListNode));
sdata=T—>data;
snext=Leafhead;
Leafhead=s;
}
Inorder(Trchild);
}
}
```

（2） 画出执行上述算法后所建立的结构；

（3） 说明该算法的功能。

21. 给定序列｛56,78,5,21,89,23,12,2｝，请建立一个最小堆，若删除一个元素后，应如何调整为最小堆，请画图说明。

22. 已知一棵树的先根和后根遍历次序如下：

ABEFGCDHI

EFGBCHIDA

试画出此树，并画出转化后的二叉树。

23. 设二叉树 bt 的二叉链表静态存储结构如下：

leftchild	0	0	2	3	7	5	8	0	10	1
data	j	h	f	d	b	a	c	e	g	i
rightchild	0	0	0	9	4	0	0	0	0	0

要求：（1）请画出该二叉树 bt 的图形表示；

（2）请分别写出前序、中序、后序遍历二叉树 bt 的序列。

24. 将下面的森林变换成二叉树。

25. 已知二叉树中的结点类型用 BinTreeNode 表示，被定义为:

```
struct BinTreeNode { char data;
BinTreeNode *lChild, *rChild;
};
```

其中 data 为结点的数据域，lChild 和 rChild 分别为指向左、右子女结点的指针域。

根据下面的函数声明，编写统计并返回一棵二叉树中所有叶子结点个数的递归算法。（假定参数 BT 为指向这棵二叉树的根结点的指针）

```
int Leaf( BinTreeNode* BT );
```

第 6 章 图

课后习题讲解

4．n 个顶点的无向图，采用邻接表存储，回答下列问题

⑴ 图中有多少条边？

⑵ 任意两个顶点 i 和 j 是否有边相连？

⑶ 任意一个顶点的度是多少?

【解答】

⑴ 边表中的结点个数之和除以 2。

⑵ 第 i 个边表中是否含有结点 j。

⑶ 该顶点所对应的边表中所含结点个数。

5．n 个顶点的无向图，采用邻接矩阵存储，回答下列问题：

⑴ 图中有多少条边？

⑵ 任意两个顶点 i 和 j 是否有边相连？

⑶ 任意一个顶点的度是多少？

【解答】

⑴ 邻接矩阵中非零元素个数的总和除以 2。

⑵ 当邻接矩阵 A 中 A[i][j]=1（或 A[j][i]=1）时，表示两顶点之间有边相连。

⑶ 计算邻接矩阵上该顶点对应的行上非零元素的个数。

6．证明：生成树中最长路径的起点和终点的度均为 1 。

【解答】用反证法证明。

设 v1, v2, …, vk 是生成树的一条最长路径，其中，v1 为起点，vk 为终点。若 vk 的度为 2，取 vk 的另一个邻接点 v，由于生成树中无回路，所以，v 在最长路径上，显然 v1, v2, …, vk , v 的路径最长，与假设矛盾。所以生成树中最长路径的终点的度为 1。

同理可证起点 v1 的度不能大于 1，只能为 1。

7．已知一个连通图如图 6-6 所示，试给出图的邻接矩阵和邻接表存储示意图，若从顶点 v1 出发对该图进行遍历，分别给出一个按深度优先遍历和广度优先遍历的顶点序列。

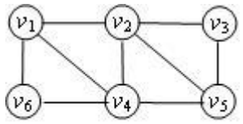


图 6-6 第 7 题图

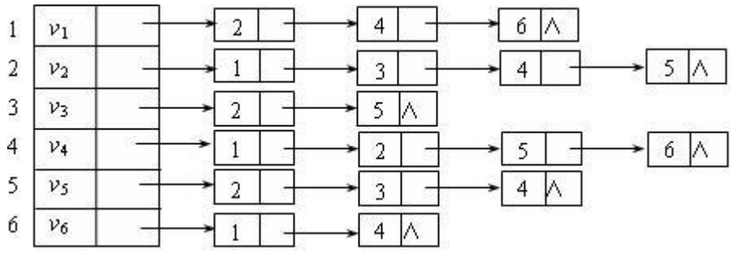
【解答】邻接矩阵表示如下：

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

深度优先遍历序列为：v1 v2 v3 v5 v4 v6

广度优先遍历序列为：v1 v2 v4 v6 v3 v5

邻接表表示如下：



8．图 6-7 所示是一个无向带权图，请分别按 Prim 算法和 Kruskal 算法求最小生成树。

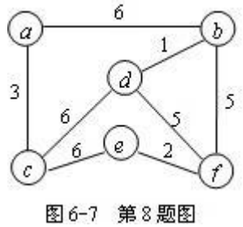
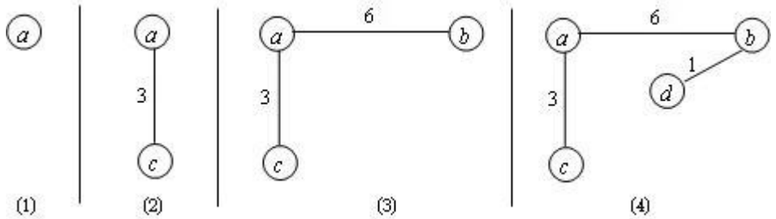
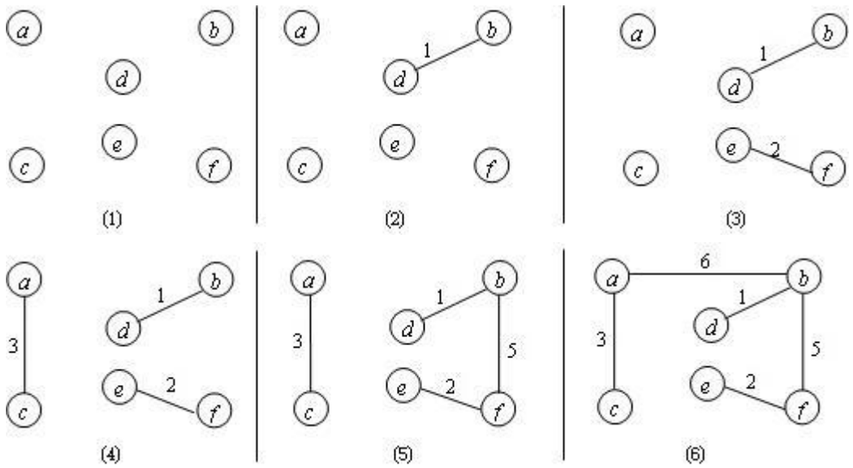


图 6-7 第 8 题图

【解答】按 Prim 算法求最小生成树的过程如下：



按 Kruskal 算法求最小生成树的过程如下：



9．对于图 6-8 所示的带权有向图，求从源点 v1 到其他各顶点的最短路径。

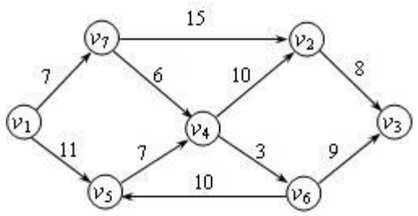


图 6-8　第 9 题图

【解答】从源点 v1 到其他各顶点的最短路径如下表所示。

源点 终点 最短路径 最短路径长度

v1 v7 v1 v7 7

v1 v5 v1 v5 11

v1 v4 v1 v7 v4 13

v1 v6 v1 v7 v4 v6 16

v1 v2 v1 v7 v2 22

v1 v3 v1 v7 v4 v6 v3 25

10．如图 6-9 所示的有向网图，利用 Dijkstra 算法求从顶点 v1 到其他各顶点的最短路径。

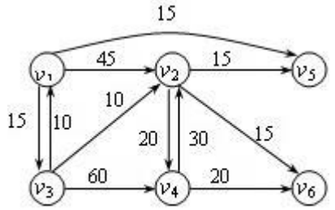


图 6-9　第 10 题图

【解答】从源点 v1 到其他各顶点的最短路径如下表所示。

源点 终点 最短路径 最短路径长度

v1 v3 v1 v3 15

v1 v5 v1 v5 15

v1 v2 v1 v3 v2 25

v1 v6 v1 v3 v2 v6 40

v1 v4 v1 v3 v2 v4 45

11．证明：只要适当地排列顶点的次序，就能使有向无环图的邻接矩阵中主对角线以下的元素全部为 0。

【解答】任意 n 个结点的有向无环图都可以得到一个拓扑序列。设拓扑序列为 v0v1v2…vn-1，我们来证明此时的邻接矩阵 A 为上三角矩阵。证明采用反证法。

假设此时的邻接矩阵不是上三角矩阵，那么，存在下标 i 和 j（i>j），使得 A[i][j]不等于零，即图中存在从 vi 到 vj 的一条有向边。由拓扑序 列的定义可知，在任意拓扑序列中，vi 的位置一定在 vj 之前，而在上述拓扑序列 v0v1v2…vn-1 中，由于 i>j，即 vi 的位置在 vj 之后，导 致矛盾。因此命题正确。

12. 算法设计

⑴ 设计算法，将一个无向图的邻接矩阵转换为邻接表。

【解答】先设置一个空的邻接表，然后在邻接矩阵上查找值不为零的元素，找到后在邻接表的对应单链表中插入相应的边表结点。

邻接矩阵存储结构定义如下：

```
const int MaxSize=10;
template
struct AdjMatrix
{
T vertex[MaxSize]; //存放图中顶点的数组
int arc[MaxSize][MaxSize]; //存放图中边的数组
int vertexNum, arcNum; //图的顶点数和边数
};
```

邻接表存储结构定义如下：

```
const int MaxSize=10;
struct ArcNode //定义边表结点
{
int adjvex; //邻接点域
ArcNode *next;
};
template
struct VertexNode //定义顶点表结点
{
T vertex;
ArcNode *firstedge;
};
struct AdjList
{
```

VertexNode adjlist[MaxSize];
int vertexNum, arcNum; //图的顶点数和边数
};
具体算法如下：

邻接矩阵转为邻接表算法 MatToList

```
void MatToList(AdjMatrix &A, AdjList &B)
{
    B.vertexNum= A.vertexNum;
    B.arcNum= A.arcNum;
    for (i=0; i<A.vertexNum; i++)
      B.adjlist[i].firstedge=NULL;
    for   (i=0; i<A.vertexNum; i++ )
      for   (j=0; j<i; j++)
        if   (A.arc[i][j]!=0) {
                        p=new ArcNode;
                        p->adjvex=j;
                        p->next=B.adjlist[i].firstedge;
                        B.adjlist[i].firstedge=p;
         }
}
```

⑵ 设计算法，将一个无向图的邻接表转换成邻接矩阵。
【解答】在邻接表上顺序地取每个边表中的结点，将邻接矩阵中对应单元的值置为 1。邻接矩阵和邻接表的存储结构定义与上题相同。具体算法如下：

邻接表转为邻接矩阵算法 ListToMat

```
void ListToMat(AdjMatrix &A, AdjList &B)
{
    A.vertexNum=B.vertexNum;
    A.arcNum=B.arcNum;
    for (i=0; i<A.vertexNum; i++)
        for  (j=0; j<A.vertexNum; j++)
            A.arc[i][j]=0;
     for  (i=0; i<A.vertexNum; i++)
     {
         p=B.adjlist[i].firstedge;
         while (p)
         {
             j= p->adjvex;
             a[i][j]=1;
             p=p->next;
         }
     }
}
```

⑶ 设计算法，计算图中出度为零的顶点个数。
【解答】在有向图的邻接矩阵中，一行对应一个顶点，每行的非零元素的个数等于对应顶点的出度。因此，当某行非零元素的个数为零时，则对应顶点的出度为零。据此，从第一行开始，查找每行的非零元素个数是否为零，若是则计数器加 1。具体算法如下：

统计出度为 0 的算法 SumZero

```
int SumZero (AdjMatrix A)
{
    count=0;
    for (i=0; i<A.vertexNum; i++)
    {
        tag=0;
        for (j=0; j<A.vertexNum; j++)
            if (arcs[i][j] !=0) {
                                tag=1;
                                break;
            }
        if (tag==0) count++;
    }
    return count;
}
```

⑷ 以邻接表作存储结构，设计按深度优先遍历图的非递归算法。
【解答】参见 6.2.1。
⑸ 已知一个有向图的邻接表，编写算法建立其逆邻接表。
【解答】在有向图中，若邻接表中顶点 vi 有邻接点 vj，在逆邻接表中 vj 一定有邻接点 vi，由此得到本题算法思路：首先将逆邻接表的表头结点 firstedge 域置空，然后逐行将表头结点的邻接点进行转化。

建立逆邻接表算法 List

```
void List (AdjList A, AdjList &B)
{
    B.vertexNum= A.vertexNum;
    B.arcNum= A.arcNum;
    for (i=0; i<A.vertexNum; i++)
        B.adjlist[i].firstedge=NULL;
    for (i=0; i<A.vertexNum; i++)
    {
        p1=A.adjlist[i].firstedge;
        while (p1)
        {
            j=p1->adjvex;
            p2=new ArcNode;
            p2->adjvex=i;
            p2->next=B.adjlist[j].firstedge;
            B.adjlist[j].firstedge=p2;
            p1=p1->next;
        }
    }
}
```

⑹ 分别基于深度优先搜索和广度优先搜索编写算法，判断以邻接表存储的有向图中是否存在由顶点 vi 到顶点 vj 的路径（i≠j）。
【解答】⑴ 基于深度优先遍历：

判断路径算法 DFS

```
int DFS (int i, int j)    //visited[]数组已初始化为 0
{
   top=-1;
   visited[i]=1; stack[++top]=i; yes=0;
   while (top!=-1 || yes==0)
   {
        i=stack[top];
        p=adjlist[i].firstedge;
        while (p && yes==0)
        {
            t=p->adjvex;
            if (t==j) yes=1;
               else if (visited[t]==0) {
                                          visited[t]=1;
                                          stack[++top]=t;
                    }
                    else p=p->next;
        }
        if   (!p)   top--;
    }
    return yes;
}
```

(2) 基于广度优先遍历：

判断路径算法 BFS

```
int   BFS(int i, int j)        //visited[]数组已初始化为 0
{
    front=-1; rear=-1;       //队列首尾指针初始化
    visited[i]=1; queue[++rear]=i; yes=0;
    while (front!=rear || yes==0)
    {
          i=queue[++front];
          p=adjlist[i].firstedge;
          while (p && yes==0)
          {
               t=p->adjvex;
               if (t==j) yes=1;
                  else if (visited[t]==0) {
                                             visited[t]=1;
                                             queue[++rear]=t;
                       }
                       else p=p->next;
          }
    }
    return yes;
}
```

学习自测及答案

11. 已知无向图 G 的邻接表如图 6-10 所示，分别写出从顶点 1 出发的深度遍历和广度遍历序列，并画出相应的生成树。

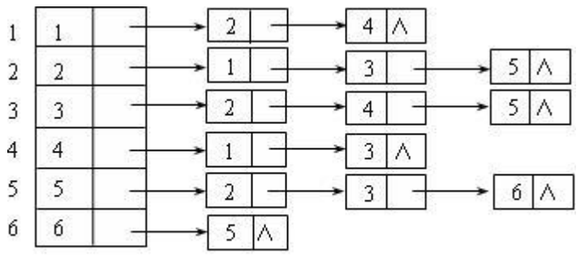
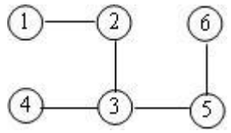


图 6-10　无向图的邻接表

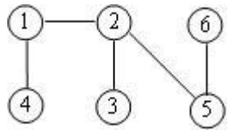
【解答】深度优先遍历序列为：1，2，3，4，5，6

对应的生成树为：



广度优先遍历序列为：1，2，4，3，5，6

对应的生成树为：



12. 已知已个 AOV 网如图 6-11 所示，写出所有拓扑序列。

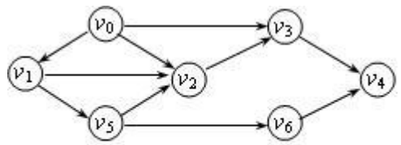


图 6-11　第 12 题图

【解答】拓扑序列为：v0 v1 v5 v2 v3 v6 v4、 v0 v1 v5 v2 v6 v3 v4、 v0 v1 v5 v6 v2 v3 v4。