

**本科生实验报告**

**实验课程 数据结构**

**学院名称 地球科学学院**

**专业名称 地理信息科学**

**学生姓名 张梦瑶**

**学生学号 201501110226**

**指导教师 刘刚**

**实验地点 D6814**

**实验成绩**

**二〇一六 年 三 月 二〇一六 年 五 月**

摘 要

针对《数据结构》课程讲授的基本数据结构及算法设计要求，本实验报告分别就线性链表、堆栈、二叉树、图等重要数据结构进行了上机实验，通过实验对这些数据结构的构造原理、基本操作及相关应用有更进一步的认识，并且对算法设计的基本思想、算法分析及算法实现过程也有更深入的理解。通过上机实验，从理论和实践角度加深了对《数据结构》课程的认知程度，学会了如何分析现实问题及设计相应的数据结构和算法来解决问题。

关键词：线性链表；堆栈；二叉树；图；算法

目录

[实验一：线性链表 2](#_Toc450167057)

[1.1 实验目的 2](#_Toc450167058)

[1.2 实验内容 2](#_Toc450167059)

[1.3 删除多余结点 2](#_Toc450167060)

[1.3.1 算法思想及算法流程图 2](#_Toc450167061)

[1.3.2 关键代码 3](#_Toc450167062)

[1.3.3 算法实现 4](#_Toc450167063)

[1.4 合并链表 4](#_Toc450167064)

[1.4.1 算法思想及算法流程图 4](#_Toc450167065)

[1.4.2 关键代码 5](#_Toc450167066)

[1.4.3 算法实现 6](#_Toc450167067)

[1.5 实验小结 6](#_Toc450167068)

[实验二：堆栈 7](#_Toc450167069)

[2.1 实验目的 7](#_Toc450167070)

[2.2 实验内容 7](#_Toc450167071)

[2.3 括号匹配算法 7](#_Toc450167072)

[2.3.1 算法思想及算法流程图 7](#_Toc450167073)

[2.3.2 关键代码 8](#_Toc450167074)

[2.3.3 算法实现 10](#_Toc450167075)

[2.4 实验小结 10](#_Toc450167076)

[实验三：二叉树 12](#_Toc450167077)

[3.1 实验目的 12](#_Toc450167078)

[3.2 实验内容 12](#_Toc450167079)

[3.3 利用广义表建立二叉树算法 12](#_Toc450167080)

[3.3.1 算法思想及算法流程图 12](#_Toc450167081)

[3.3.2 关键代码 14](#_Toc450167082)

[3.3.3 算法实现 14](#_Toc450167083)

[3.4 三种遍历 15](#_Toc450167084)

[3.4.1 算法思想及流程图 15](#_Toc450167085)

[3.4.2 关键代码 17](#_Toc450167086)

[3.4.3 算法实现 19](#_Toc450167087)

[3.5 实验小结 19](#_Toc450167088)

[实验四:图 19](#_Toc450167089)

[4.1 实验目的 19](#_Toc450167090)

[4.2实验内容 20](#_Toc450167091)

[迪杰斯特拉算法 20](#_Toc450167092)

[4.3迪杰斯特拉算法 20](#_Toc450167093)

[4.3.1 算法思想及流程图 20](#_Toc450167094)

[4.3.2 关键代码 22](#_Toc450167095)

[4.3.3 算法实现 23](#_Toc450167096)

[4.4实验小结 23](#_Toc450167097)

[实验五：排序 24](#_Toc450167098)

[5.1 实验目的 24](#_Toc450167099)

[5.2 实验内容 24](#_Toc450167100)

[5.3 冒泡排序 24](#_Toc450167101)

[5.3.1 算法思想及算法流程图 24](#_Toc450167102)

[5.3.2 关键代码 25](#_Toc450167103)

[5.3.3 算法实现 26](#_Toc450167104)

[5.4 实验小结 26](#_Toc450167105)

实验一：线性链表

# 1.1 实验目的

（1）学会定义线性表的链式存储类型，利用C语言编程实现该数据结构，并实现线性链表的一些基本操作，例如链表的插入、删除、查找等操作。

（2）加深对算法分析及算法设计过程的理解，学会如何进行算法设计及算法流程图的建立。

（3）进一步熟悉C语言，理解结构体、函数、指针等。

# 1.2 实验内容

算法一（删除多余结点）：已知线性链表第1个链结点的指针为list，试写一个算法，删除数据域值相同的多余结点，即：若链表中有多个结点具有相同的数据域值，只保留其中一个结点，其余结点均从链表中删除，使得到的链表中所有结点的数据域值都不相同。

算法二（合并链表）：设线性表X=(x1,x2,x3,…,xn)与Y=(y1,y2,y3,…,ym)都采用链式存储结构。试写一个算法合并这两个线性链表为一个线性链表，使得：



# 1.3 删除多余结点

1.3.1 算法思想及算法流程图

算法基本思想：设置三个指针，两重循环，一个指针在外层循环中不断后移，同时内层循环中利用两个指针来依次比较链表中的数值与外层指针的值是否相等，如果相等则删去该结点，直到外层循环结束，算法结束。

算法流程图：

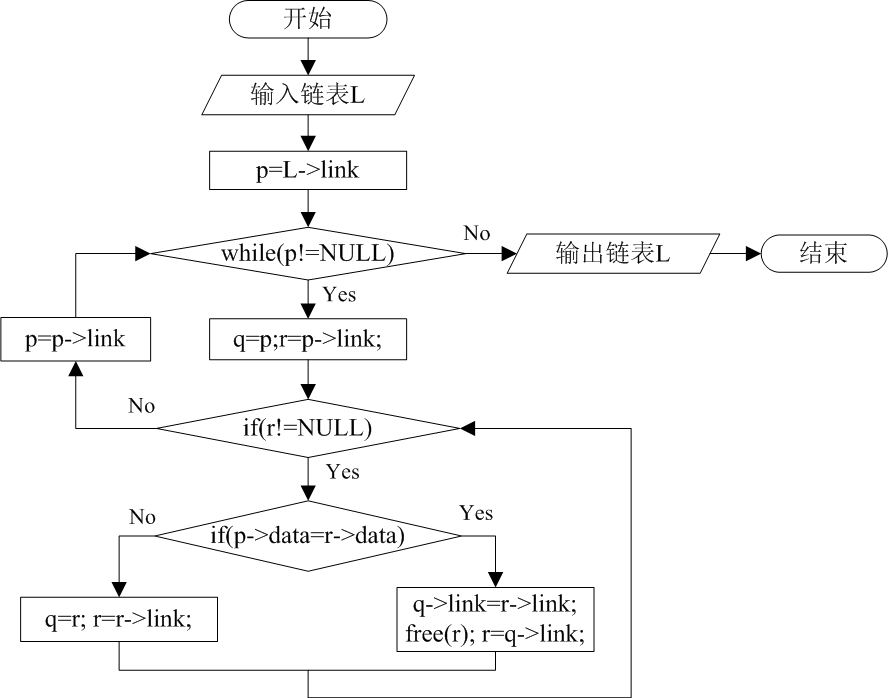


图 1 删除多余结点算法流程图

1.3.2 关键代码

实现该算法的核心代码如下：

//删除值相同的多余节点,list为传入链表

void DeleteSameNode(LinkList list)

{

LinkList p,r,q;

p=list->link;//跳过头结点

while(p!=NULL)

{

q=p; //q指向当前被比较结点（值可能与p所指结点值相同）的前一个结点

r=p->link; //r总是指向当前被比较的结点，值相同则该结点被删除

while(r!=NULL)

{

if(p->data == r->data) //值相同，删除该节点，指针指向下一个节点

{

q->link=r->link;

free(r);

r=q->link;

}

else //值不相同，指针直接指向下一个节点

{

q=r;

r=r->link;

}

}

p=p->link;//分析下一个结点是否存在值相同的多余结点

}

}

1.3.3 算法实现

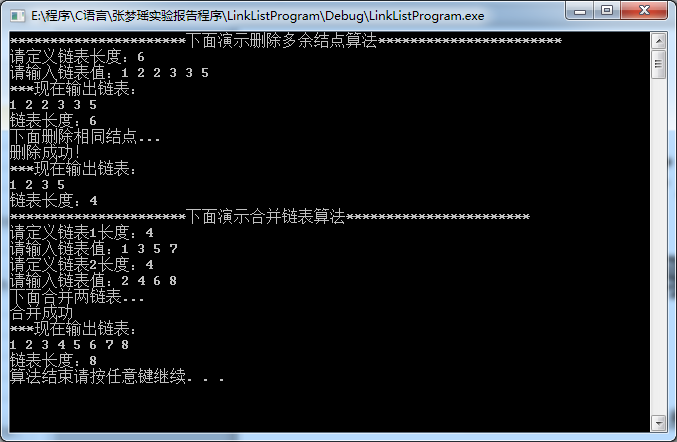


图 2 删除多余结点算法运行结果

# 1.4 合并链表

1.4.1 算法思想及算法流程图

算法基本思想：将X链表复制到一个新链表，并求出X链表的长度。再将Y链表中位置小于等于X链表长度的结点插入新链表的偶数位置，位置比X链表长度长的结点插入新链表尾部。Y链表的结点插入完则返回新链表，算法结束。

算法流程图：



图 3 合并两链表算法流程图

1.4.2 关键代码

实现该算法的核心代码如下：

//合并两个链表，list1和list2为两个传入链表

LinkList UnionLists(LinkList list1,LinkList list2)

{

LinkList p,r; //定义指针p,r分别指向list1,list2的首结点

p=list1->link;

r=list2->link;

LinkList retList=Initial();//初始化合并后的链表

int plist1\_length=0;

while(p!=NULL) //将list1的所有结点添加到retList尾部

{

plist1\_length++;

InsertoTail(retList,p->data);

p=p->link;

}

int plist2\_length=0;

while(r!=NULL)

{

plist2\_length++;

if(plist2\_length<=plist1\_length) //如果list2长度小于list1长度，则将list2的结点依次插入到retList的偶数位

{

InsertoJ(retList,plist2\_length\*2,r->data);

}

else //如果list2长度大于list1长度，则将多余结点插入到retList的尾部

{

InsertoTail(retList,r->data);

}

r=r->link;

}

return retList;

}

1.4.3 算法实现

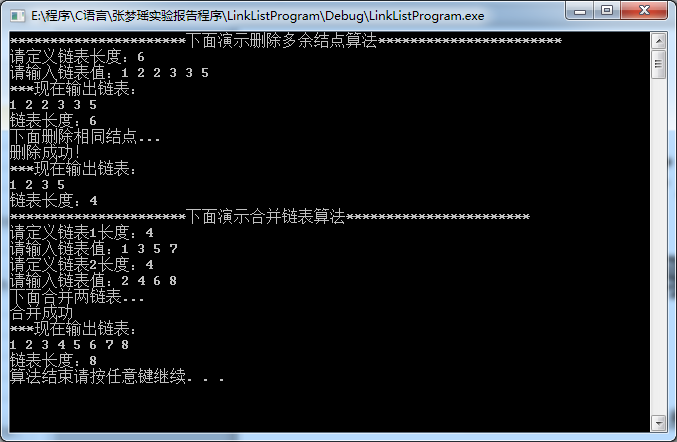


图 4 合并两链表算法运行结果

# 1.5 实验小结

链表的顺序与链式存储结构各有各的优点。

链式存储结构更为灵活，不需要提前定义链表长度，而且链表的各种插入与删除等操作更为灵活，简便。线性链表的各种算法往往都需要借助活动指针完成。

实验二：堆栈

# 2.1 实验目的

（1）学会定义堆栈顺序和链式存储类型，利用C语言编程实现该数据结构，并实现堆栈的一些基本操作，例如堆栈的进栈、退栈等操作。

（2）加深对算法分析及算法设计过程的理解，学会如何进行算法设计及算法流程图的建立。

（3）进一步熟悉C语言。

(4) 理解堆栈先进后出的特性在解决一些算法问题中的作用，如括号符匹配以及递归。

# 2.2 实验内容

括号符匹配：判断一个包含小括号，中括号，大括号的字符串是否匹配。

# 2.3 括号匹配算法

2.3.1 算法思想及算法流程图

算法基本思想：

1. 设置一个堆栈，依次读取字符串里的每一个元素。
2. 如果是”(“,”[“,”{“,则将该元素进栈。
3. 如果是”)”,”]”,”}”,则将栈顶元素退栈并与之匹配，如果不是相应的左括号，则不匹配，返回-1；
4. 所有元素读取完毕，判断堆栈是否为空，如果为空，则匹配成功，如果不为空，则不匹配。
5. 算法结束。

算法流程图：



图 5 括号符匹配算法流程图

2.3.2 关键代码

int Brackets3(datatype STACK[],int &top, string E)

{

Initials(top);//初始化堆栈

int i=0;

char item;

while(i<E.length())//依次读取每一个元素

{

if(E[i]=='('||E[i]=='['||E[i]=='{')//如果是左括号，做进栈处理

{

push(STACK,top,E[i]);

i++;

}

//如果是右括号，则退出栈顶元素与之相匹配：

else if(E[i]==')')

{

if(EMPTYS(top))//首先判断栈是否为空

{

printf("不匹配\n");

return -1;

}

else if(pop(STACK,top,item));

{

if(item=='(')

{

i++;

}

else

{

printf("不匹配\n");

return -1;

}

}

}//针对于())这种情况，因为item是引用传递，虽然最后退栈不成功，return 0,但是item之前被修改成为了‘(',所以匹配进去还是成功，所以实现判断堆栈是否为空

else if(E[i]==']')

{

if(EMPTYS(top))

{

printf("不匹配\n");

return -1;

}

else if(pop(STACK,top,item));

{

if(item=='[')

{

i++;

}

else

{

printf("不匹配\n");

return -1;

}

}

}

else if(E[i]=='}')

{

if(EMPTYS(top))

{

printf("不匹配\n");

return -1;

}

else if(pop(STACK,top,item));

{

if(item=='{')

{

i++;

}

else

{

printf("不匹配\n");

return -1;

}

}

}

}

if(EMPTYS(top))//判断栈是否为空

{ printf("匹配成功\n");//栈为空则匹配成功

return 1;

}

else

{

printf("不匹配\n");//否则匹配失败

return -1;

}

}

**2.3.3 算法实现**

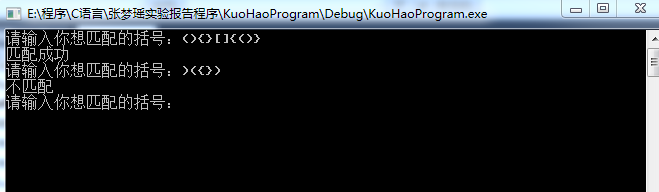


图 6 括号符匹配算法运行结果

# 2.4 实验小结

堆栈的顺序与链式存储结构各有优点。

堆栈先进后出的特性可以解决很多问题，如括号符匹配，进制转换，后缀式运算。

递归算法一般都能通过堆栈改写为非递归算法。

实验三：二叉树

# 3.1 实验目的

（1）学会定义二叉树链式存储类型，利用C语言编程实现该数据结构，并实现二叉树的一些基本操作，例如二叉树的建立和遍历等操作。

（2）加深对算法分析及算法设计过程的理解，学会如何进行算法设计及算法流程图的建立。

（3）进一步熟悉C语言。

(4) 理解二叉树的特性，进一步学会使用结构体，指针，递归以及堆栈的应用。

# 3.2 实验内容

算法一：利用广义表建立二叉树。

算法二：二叉树的前序，中序，后序遍历。

# 3.3 利用广义表建立二叉树算法

3.3.1 算法思想及算法流程图

算法基本思想：

依次从广义表中取得一个元素，则

（1）若当前取得的元素为字母，则如下建立一个新的结点

1. 若该结点为二叉树的根结点，则将该结点的地址送T。

2.若该结点不是二叉树的根节点，则将该结点作为左孩子（若标志flag为1）或者右孩子结点（若标志flag为2）链接到其双亲结点上（此时双亲结点的地址在栈顶位置）。

（2）若当前取到的元素为左括号，则表明一个子表开始，将标志flag置为1，同时将前面那个地址进栈。

(3) 若当前取得的元素为右括号，则表明一个子表结束，做退栈操作。

(4) 若当前取得的元素为逗号，则将flag置为2.

算法流程图：



图 7 利用广义表建立二叉树算法流程图

3.3.2 关键代码

//利用广义表建立二叉树：不加@作为结束标志

void Creat(string E,BTREE &T)

{

BTREE p,STACK[M];//定义一个堆栈来存放父节点的地址

T=NULL;

char ch;

int i=0,top=-1,flag;//flag为标志位

while(i<E.length())

{

ch=E[i];

if(ch=='(')

{

STACK[++top]=p;//一个子表开始，父结点地址进栈

flag=1;

}

else if(ch==')')

{

top--;

}

else if(ch==',')

{

flag=0;

}

else

{

p=(BTREE)malloc(sizeof(BTNode));//开辟一片内存

p->data=ch;

p->lchild=NULL;

p->rchild=NULL;

if(T==NULL)

{

T=p;

}

else if(flag==1)//证明该结点应为左孩子结点

{

STACK[top]->lchild=p;

}

Else//

{

STACK[top]->rchild=p;//作右孩子结点

}

}

i++;

}

}

3.3.3 算法实现

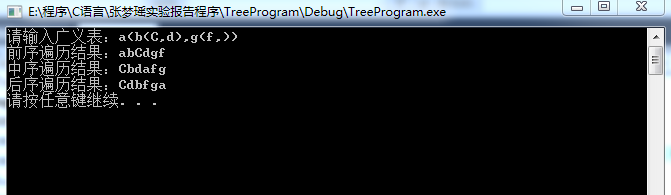


图 8 利用广义表建立二叉树算法运行结果

# 3.4 三种遍历

3.4.1 算法思想及流程图

算法思想：

前序遍历：访问根结点，遍历左子树，遍历右子树。

后序遍历：遍历左子树，访问根结点，遍历右子树。

后序遍历：遍历左子树，遍历右子树，访问根结点。

以下给出非递归算法。

算法流程图：



图 9 前序遍历算法流程图



图 10 中序遍历算法流程图



图 11 后序遍历流程图

3.4.2 关键代码

//前序遍历;

void PREORDER(BTREE T)

{

BTREE p=T,STACK[M];

int top=-1;

if(T!=NULL)

{

while(p!=NULL||top!=-1)

{

while(p!=NULL)

{

visit(p);

STACK[++top]=p;//结点地址进栈

p=p->lchild;//访问左子树

}

p=STACK[top--];//结点地址退栈

p=p->rchild;//访问右子树

}

}

}

//中序遍历：

void INOROER(BTREE T)

{

BTREE p,STACK[M];

int top=-1;

p=T;

if(T!=NULL)

{

while(p!=NULL||top!=-1)

{

while(p!=NULL)

{

STACK[++top]=p;

p=p->lchild;//访问左子树

}

p=STACK[top--];

visit(p);//访问结点

p=p->rchild;//访问右子树

}

}

}

//后序遍历：

void POSTORDER(BTREE T)

{

BTREE STACK1[M],p=T;

int STACK2[M],flag,top=-1;//flag=1表示访问，flag=0表示暂不访问

if(T!=NULL)//递归思路，两次进栈

{

while(p!=NULL||top!=-1)

{

while(p!=NULL)

{

STACK1[++top]=p;//将p所指结点地址进栈

STACK2[top]=0;//标志0进栈

p=p->lchild;//将p指向其左孩子结点

}

p=STACK1[top];

flag=STACK2[top--];

if(flag==0)

{

STACK1[++top]=p;

STACK2[top]=1;

p=p->rchild;

}

else{

visit(p);

p=NULL;

}

}

}

}

3.4.3 算法实现

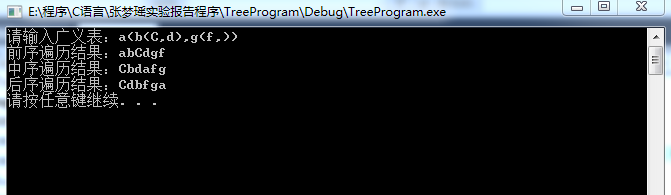


图 12 二叉树遍历运行结果

# 3.5 实验小结

写算法时二叉链表更为灵活，实用。

二叉树的构建和遍历都可以通过递归来完成，也可以通过堆栈来写出他们的非递归算法

实验四:图

# 4.1 实验目的

（1）学会定义图的两种存储类型，利用C语言编程实现该数据结构，并实现图的一些基本操作，例如图的建立和遍历，最小生成树，迪杰斯特拉等算法及操作。

（2）加深对算法分析及算法设计过程的理解，学会如何进行算法设计及算法流程图的建立。

（3）进一步熟悉C语言。

(4) 理解图的特性，进一步学会使用结构体，指针，递归以及堆栈的应用。

# 4.2实验内容

迪杰斯特拉算法

# 4.3迪杰斯特拉算法

4.3.1 算法思想及流程图

算法思想：1.确定dist、s、path 三个数组的初值。

2.利用s数组与dist 数组在那些尚未找到最短路径的顶点中确定一个与源点v最近的顶点u,并置s[u]为1,同时将顶点u加入path[u]。

3. 根据顶点u修改源点到所有尚未找到最短路径的顶点的路径长度。即

1）将源点v到顶点u的(最短)路径长度分别加到源点v通过顶点u可以直接到达、且尚未找到最短路径那些的顶点的路径长度上.若加后的长度小于原来v 到某顶点r的路径长度，则用加后的长度替换原来的长度，否则不替换。

2）若替换，将源点v 到顶点u 的路径(最短路径)上经过的所有顶点替换path[r]。

4. 重复上述过程的第2至第3步n–1次。

算法流程图：



图 13 迪杰斯特拉算法流程图

4.3.2 关键代码

//cost是邻接矩阵,v是起始点，n是顶点数量，path是最短路径，dist是最短权值,pos是最短路径经过的顶点个数

void SHORTEST\_PATH(int \*\*cost,int v,int n,int \*\*path,int\*dist,int \*pos)

{

int k;

int i=0,w=1,u,count;

int \*s=new int[n]; //标志数组

for(i=0;i<n;i++) //初始化

{

s[i]=0;

dist[i]=cost[v][i];

path[i][0]=v;

pos[i]=0;

}

s[v]=1;

count=1;

while(count<n)

{

int temp=10000;

for(i=0;i<n;i++)//寻找没有找到最短路径的当前最近的顶点

{

if(s[i]==0&&dist[i]<temp)

{

u=i;

temp=dist[i];

}

}

s[u]=1;

path[u][++pos[u]]=u;

//path[w]=path[u]

for(i=0;i<n;i++)//寻找与u直接连通但没有确定最短路径的顶点,修改权值和顶点

{

if(s[i]==0&&dist[u]+cost[u][i]<dist[i])

{

dist[i]=dist[u]+cost[u][i];

for(k=0;k<=pos[u];k++)

path[i][k]=path[u][k];

pos[i]=pos[u];

}

}

count++;

}

}

4.3.3 算法实现

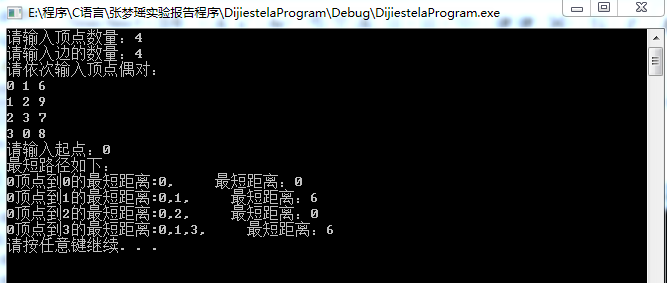


图 14迪杰斯特拉算法运行结果

# 4.4实验小结

图的特性：多对多。所以图的算法比前面的几种数据结构更为复杂，图的很多算法运用递归更为方便，不过迪杰斯特拉算法是非递归算法

最小生成树(普里姆算法和克鲁斯卡尔算法)和最短路径(迪杰斯特拉算法)应该深刻掌握.

实验五：排序

# 5.1 实验目的

（1）学会基本的冒泡排序，选择排序，插入排序，快速排序

（2）加深对算法分析及算法设计过程的理解，学会如何进行算法设计及算法流程图的建立。

（3）进一步熟悉C语言。

(4) 比较各个排序算法的优劣

# 5.2 实验内容

冒泡排序

# 5.3 冒泡排序

5.3.1 算法思想及算法流程图

算法思想：

冒泡排序:依次比较相邻的两个记录的关键字，若两个记录是反序的(即前一个记录的关键字大于后前一个记录的关键字)，则进行交换，直到没有反序的记录为止。

插入排序：将待排序的记录Ri，插入到已排好序的记录表R1, R2 ,….,Ri-1中，得到一个新的、记录数增加1的有序表。 直到所有的记录都插入完为止。

设待排序的记录顺序存放在数组R[1…n]中，在排序的某一时刻，将记录序列分成两部分：

◆ R[1…i-1]：已排好序的有序部分；

◆ R[i…n]：未排好序的无序部分。

显然，在刚开始排序时，R[1]是已经排好序的。

选择排序：第1趟，从第1个到第n个记录中选择关键码最小的记录与第1个记录交换；

第2趟，从第2个到第n个记录中选择关键码最小的记录与第2个记录交换；

…

第i趟， 从第i个到第n个记录中选择关键码最小的记录与第i个记录交换;

…

直到第n-1趟，从最后两个记录中选择较小的记录放置在第n-1 位置。排序结束



图 15 冒泡排序算法流程图

5.3.2 关键代码

//冒泡排序（从右开始）

void Bubble\_Sort2(int \*a,int n)

{

int i,j,temp,swap;

printf("现在是冒泡排序：\n");

for(i=0;i<n-1;i++)

{

swap=0;

for(j=0;j<n-i-1;j++)//先排右，则从左排过去

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

swap=1;

}

}

printf("现在是第%d步排序结果：",i+1);

for(int k=0;k<n;k++)

{

printf("%d ",a[k]);

}

printf("\n");

if(swap==0) break;

}

}

5.3.3 算法实现

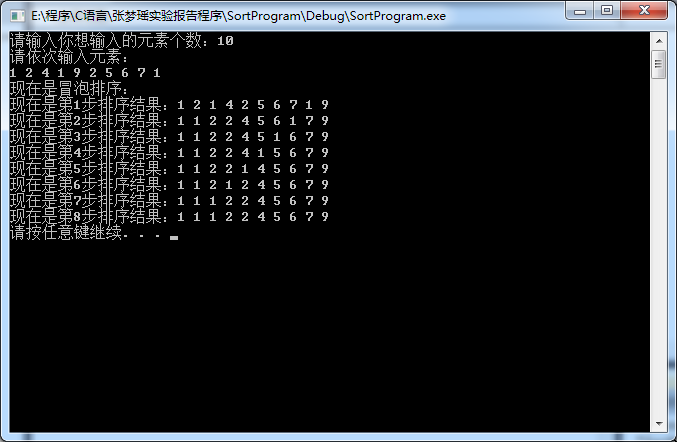


图 16 排序算法运行结果

# 5.4 实验小结

排序算法各有各自的优点和时间复杂度。

排序算法比较简单，不过应该重点掌握。

|  |  |
| --- | --- |
| **学生实验 心得** | 这学期学习的这门数据结构,感觉是在熟练掌握C语言的基础上,更深入一层的学习,自我感觉来说它比C语言更加难以理解,但是确实学习了许多更实用的算法,比如最短路径,堆栈,排序,二叉树等等,以前不明白学习这方面的知识,或者说去写一个几行甚至几百行的代码有什么具体的作用,通过自己去认真地理解代码,运行代码,以及书本上的所给的知识,我意识到了,计算机方面许多程序的运行,应用等这些都是离不开我们所学习的知识,更离不开我们所写的代码.开始的学习是困难的枯燥的,甚至我在没有深入的学习之前就下意识的觉得自己搞不懂学不懂这门课程,但是通过自己静下心来的摸索,我开始对这门课程有了新的认识,并且有点对它感兴趣.我甚至期待着自己能有一天通过自己学习的知识去写出一个能够起到具体作用的程序.老师说过,我们现在学的数据结构上的知识只是这门课程的一点皮毛,更深层次的东西还要我们以后自己去学习去挖掘.我想我应该自己继续学习关于它的知识.并且希望自己能在这门课上取得好的成绩!最后,感谢老师所讲授的知识.希望以后还能听到老师讲课!  学生（签名）：  年 月 日 |
| **指导**  **教师**  **评语** | 成绩评定：  指导教师（签名）：  年 月 日 |