**数据结构预算法**

**作业**

**指导老师：陈安龙**

**姓名：段柏杉**

**学号：2017221101003**

**点名册序号：81**

**第一章**

**13．**

ADT **ASet**

{ 数据对象:

D = {bi | 1<= i <= n, n>1, bi为int类型} //bi是正整数

数据关系:

R = {< bi, b(i+1)>| bi,b(i+1) ∈ D, i=1,…,n-1}

基本运算:

AssignASet(L) : 由整数数组a[0..n-1]创建一个集合，构造集合a。

DispASet(L): 输出数组,当L不为空时顺序显示L中所有的元素。

ASetchr(L): 按元素值查找，若存在这样的值，返回真，否则，返回假。

AndASet(La,Lb,&Lc): 合并La以及Lb中不同于La中元素的元素到数组Lc中。

DifASet(La,Lb,&Lc): 将存在于La中但不存在于Lb中的元素存入数组Lc中。

InterASet(La,Lb,&Lc): 将La，Lb中共有的元素存入数组Lc中。

}

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 1000

typedef int ElemType;

typedef struct

{ ElemType data[Maxsize];

int length;

}SqASet;

void AssignASet(SqASet \*&L,ElemType a[],int n);

void DispASet(SqASet \*L);

bool ASetchr(SqASet \*L, ElemType e);

bool ASetInsert(SqASet \*&L,int i,ElemType e);

void AndASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc);

bool ASetDelete(SqASet \*&L,ElemType e);

void DifASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc);

void InterASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc);

int main(){ //用于测试的main函数

SqASet \*a,\*b,\*c,\*d,\*e;

int m[Maxsize] = {1,2,3,4,5,6};

int n[Maxsize] = {4,5,6,7,8};

int o[Maxsize] = {};

AssignASet(a,m,6);

AssignASet(b,n,5);

AssignASet(c,o,1);

AssignASet(d,o,1);

AssignASet(e,o,1);

printf("输出集合A：");

DispASet(a);

printf("输出集合B：");

DispASet(b);

printf("判断1是否在A中：");

if(ASetchr(a,1))

printf("Yes\n");

printf("A和B的并集：");

AndASet(a,b,c);

DispASet(c);

printf("A和B的差集：");

DifASet(a,b,d);

DispASet(d);

printf("A和B的交集：");

InterASet(a,b,e);

DispASet(e);

}

void AssignASet(SqASet \*&L,ElemType a[],int n) //由a中的n个元素建立集合L

{ int i= 0,k = 0;

L = (SqASet \*)malloc(sizeof(SqASet));

while(i<n)

{ L ->data[k] = a[i];

k++;i++;

}

L -> length = k;

}

void DispASet(SqASet \*L) //按顺序输出集合L

{ if(L->length == 0)

return;

for(int i=0;i<L->length;i++)

printf("%d ",L->data[i]);

printf("\n");

}

bool ASetchr(SqASet \*L, ElemType e) //查找L中是否有e

{ int i= 0;

while(i<L->length && L->data[i]!= e)

i++;

if(i>=L->length)

return false;

else

return true;

}

bool ASetInsert(SqASet \*&L,int i,ElemType e) //在L的第i个位置插入元素e

{ int j = 0;

if(i<1 || i>L->length)

return false;

i--;

for(j = L->length;j>i;j--)

L->data[j] = L->data[j-1];

L->data[i]=e;

L->length++;

return true;

}

bool ASetDelete(SqASet \*&L,ElemType e) //删除L中的元素 e

{ int j= 0,i = 0;

while(i<L->length && L->data[i]!= e)

i++;

for(j = i;j<L->length;j++)

L->data[j] = L->data[j+1];

L->length--;

return true;

}

void AndASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc) //求La和Lb的并集

{ int i= 0;int len = 0;

for(i=0;i<La->length;i++)

ASetInsert(Lc,i+1,La->data[i]);

len = La->length;

for(i=0;i<Lb->length;i++)

{

if(!ASetchr(La,Lb->data[i]))

ASetInsert(Lc,++len,Lb->data[i]);

}

ASetDelete(Lc,0);

}

void DifASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc) //求La和Lb的差集

{ int i;

for(i= 0;i<La->length;i++)

ASetInsert(Lc,i+1,La->data[i]);

for(i= 0;i<Lb->length;i++)

{

if(ASetchr(Lc,Lb->data[i]))

ASetDelete(Lc,Lb->data[i]);

}

ASetDelete(Lc,0);

}

void InterASet(SqASet \* La,SqASet \* Lb,SqASet \*&Lc) //求La和Lb的交集

{ int i;

for(i=0;i<La->length;i++)

ASetInsert(Lc,i+1,La->data[i]);

ASetDelete(Lc,0);

for(i=0;i<La->length;i++){

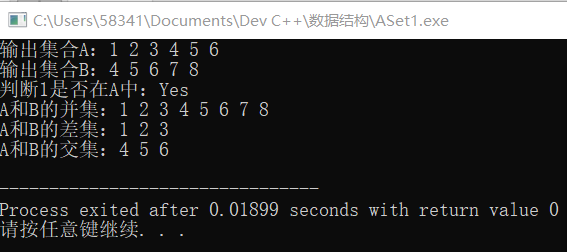
if(!ASetchr(Lb,La->data[i]))

ASetDelete(Lc,La->data[i]);

}

}

测试结果



**第二章**

**P72**

**7-8.**

#include<stdio.h>

#include<stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct

{

ElemType data[MaxSize];

int length;

}SqList;

void CreateList(SqList \*&L, ElemType a[], int n) //创建顺序表

{

int i = 0, k = 0;

L = (SqList\*)malloc(sizeof(SqList));

while (i<n)

{

L->data[k] = a[i];

k++; i++;

}

L->length = k;

}

void InitList(SqList \*&L) //初始化顺序表

{

L = (SqList\*)malloc(sizeof(SqList));

L->length = 0;

}

void DispList(SqList \*&L) //输出顺序表

{

int i;

for (i = 0; i<L->length; i++)

printf("%d ", L->data[i]);

printf("\n");

}

void swap(int &a, int &b) //交换a b的值

{

int c;

c = b;

b = a;

a = c;

}

void Sort(SqList \*&L) //排序

{

int tmp;

for (int i = 1; i<L->length; i++)

{

int j = i - 1;

if (L->data[i]<L->data[j])

{

tmp = L->data[i];

L->data[i] = L->data[j];

while (tmp<L->data[j - 1])

{

L->data[j] = L->data[j - 1];

j--;

}

L->data[j] = tmp;

}

}

}

void ListInsert(SqList \*&L, ElemType x) //插入x元素并保持有序 (7)

{

int i = 0;

while (i<L->length && L->data[i] < x)

i++;

for (int j = L->length; j>i; j--)

L->data[j] = L->data[j - 1];

L->data[i] = x;

L->length++;

}

void SqListEx(SqList \*&L) //使小于零的元素放在大于零元素的右边 (8)

{

int i = 0, j = L->length - 1;

while (i<j) {

while (i<j && L->data[i]<0)

i++;

while (i<j && L->data[j] >= 0)

j--;

if (i<j)

swap(L->data[i], L->data[j]);

}

}

int main()

{

int a[MaxSize], n, x;

SqList \*L;

InitList(L);

printf("请输入元素个数:");

scanf("%d", &n);

printf("请输入元素:");

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

scanf("%d", &a[i]);

CreateList(L, a, n);

printf("第8题 正负排序后:");

SqListEx(L);

DispList(L);

Sort(L);

printf("第7题 请输入想插入的元素:");

scanf("%d", &x);

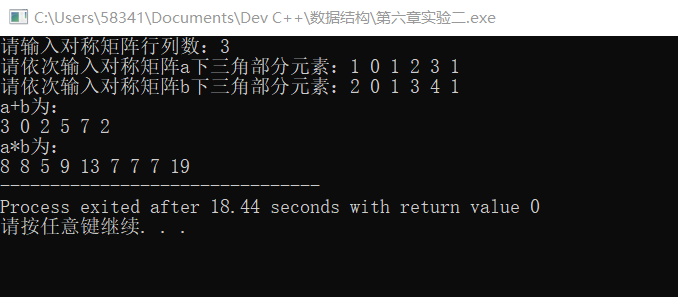
printf("插入后:");

ListInsert(L, x);

DispList(L);

}

测试结果



**第三章**

**P117**

**10（括号匹配）.**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 1000

typedef char ElemType;

typedef struct linknode

{

ElemType data;

struct linknode \*next;

}LinkStNode;

void InitStack(LinkStNode \*&s);

bool StackEmpty(LinkStNode \*s);

void Push(LinkStNode \*&s, ElemType e);

bool Pop(LinkStNode \*&s, ElemType &e);

bool GetTop(LinkStNode \*s, ElemType &e);

bool Match(char k[], int n);

int main()

{

char a[Maxsize];

int count = 0;

while ((a[count] = getchar()) != '\n')

count++;

count++;

if (Match(a, count))

printf("Match");

else

printf("Not Match");

return 0;

}

void InitStack(LinkStNode \*&s) //初始化栈

{

s = (LinkStNode \*)malloc(sizeof(LinkStNode));

s->next = NULL;

}

void DestroyStack(LinkStNode \*&s) //销毁栈

{

LinkStNode \* pre = s, \*p = s->next;

while (p != NULL)

{

free(pre);

pre = p;

p = pre->next;

}

free(pre);

}

bool StackEmpty(LinkStNode \*s) //判断栈是否为空

{

return(s->next == NULL);

}

void Push(LinkStNode \*&s, ElemType e) //进栈

{

LinkStNode \*p;

p = (LinkStNode \*)malloc(sizeof(LinkStNode));

p->data = e;

p->next = s->next;

s->next = p;

}

bool Pop(LinkStNode \*&s, ElemType &e) //出栈

{

LinkStNode \*p;

if (s->next == NULL)

return false;

p = s->next;

e = p->data;

s->next = p->next;

free(p);

return true;

}

bool GetTop(LinkStNode \*s, ElemType &e) //取栈顶元素

{

if (s->next == NULL)

return false;

e = s->next->data;

return true;

}

bool Match(char k[], int n) //第十题括号匹配

{

int i = 0;

ElemType e;

LinkStNode \*s;

InitStack(s);

bool flag = true;

while (i<n && flag)

{

if (k[i] == '(' || k[i] == '[' || k[i] == '{')

Push(s, k[i]);

if (k[i] == ')')

{

if (GetTop(s, e))

{

if (e == '(') Pop(s, e);

else flag = false;

}

}

if (k[i] == ']')

{

if (GetTop(s, e))

{

if (e == '[') Pop(s, e);

else flag = false;

}

}

if (k[i] == '}')

{

if (GetTop(s, e))

{

if (e == '{') Pop(s, e);

else flag = false;

}

}

i++;

}

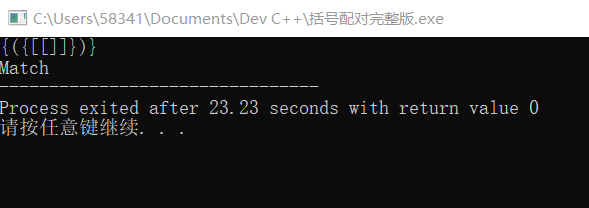
if (!StackEmpty(s)) flag = false;

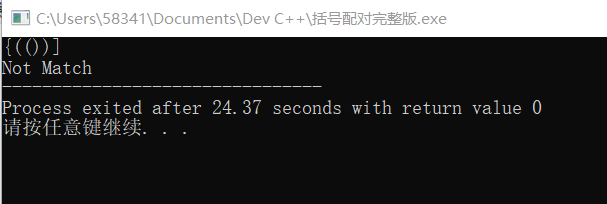
DestroyStack(s);

return flag;

}

测试结果





**11.**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 1000

typedef char ElemType;

typedef struct

{

ElemType data[Maxsize];

int front,rear;

}SqQueue;

void InitQueue(SqQueue \*&q);

bool enQueue(SqQueue \*&q,ElemType e);

bool deQueue(SqQueue \*&q,ElemType &e);

void DestroyQueue(SqQueue \*&q);

void XuLie();

int main(){

XuLie();

return 0;

}

void InitQueue(SqQueue \*&q) //初始化环队

{

q = (SqQueue \* )malloc(sizeof(SqQueue));

q->front = q->rear = 0;

}

void DestroyQueue(SqQueue \*&q) //销毁环队

{

free(q);

}

bool QueueEmpty(SqQueue \*q) // 判断环队是否为空

{

return(q->front == q->rear);

}

bool enQueue(SqQueue \*&q,ElemType e) //进队

{

if((q->rear + 1)%Maxsize == q->front)

return false;

q->rear = (q->rear+1)%Maxsize;

q->data[q->rear] = e;

return true;

}

bool deQueue(SqQueue \*&q,ElemType &e) //出队

{

if(q->front== q->rear)

return false;

q->front = (q->front+1)%Maxsize;

e = q->data[q->front];

return true;

}

void XuLie() //第11题

{

int i = 0;

char s,e;

SqQueue \*q;

InitQueue(q);

while(true)

{

s = getchar();

if(s >= '0' && s <= '9')

{

if(!enQueue(q,s))

printf("队满，不能入队\n");

}

else if(s >='a' && s<= 'z')

{

if(!deQueue(q,e)) printf("队空，无法出队\n");

else printf("第%d个数字为：%c\n",++i,e);

}

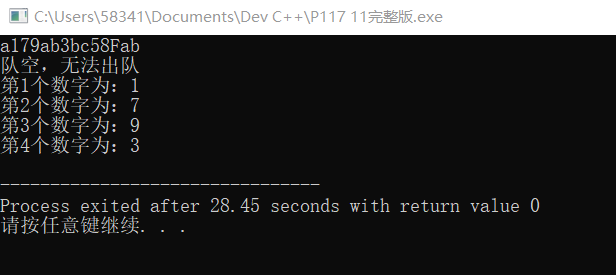
else break;

}

DestroyQueue(q);

}

测试结果



**12（倒置环队）.**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 1000

typedef int ElemType;

typedef struct

{

ElemType data[Maxsize];

int front,rear;

}SqQueue;

typedef struct linknode

{

ElemType data;

struct linknode \*next;

}LinkStNode;

void InitStack(LinkStNode \*&s) //初始化栈

{

s = (LinkStNode \* )malloc(sizeof(LinkStNode));

s->next = NULL;

}

bool StackEmpty(LinkStNode \*s) //判断栈是否为空

{

return(s->next == NULL);

}

void Push(LinkStNode \*&s,ElemType e) //进栈

{

LinkStNode \*p;

p = (LinkStNode \* )malloc(sizeof(LinkStNode));

p->data = e;

p->next = s->next;

s->next = p;

}

bool Pop(LinkStNode \*&s,ElemType &e) //出栈

{

LinkStNode \*p;

if(s->next == NULL)

return false;

p = s->next;

e = p->data;

s->next = p->next;

free(p);

return true;

}

void InitQueue(SqQueue \*&q) //初始化环队

{

q = (SqQueue \* )malloc(sizeof(SqQueue));

q->front = q->rear = 0;

}

void DestroyQueue(SqQueue \*&q) //销毁环队

{

free(q);

}

bool QueueEmpty(SqQueue \*q) // 判断环队是否为空

{

return(q->front == q->rear);

}

bool enQueue(SqQueue \*&q,ElemType e) //进队

{

if((q->rear + 1)%Maxsize == q->front)

return false;

q->rear = (q->rear+1)%Maxsize;

q->data[q->rear] = e;

return true;

}

bool deQueue(SqQueue \*&q,ElemType &e) //出队

{

if(q->front== q->rear)

return false;

q->front = (q->front+1)%Maxsize;

e = q->data[q->front];

return true;

}

void ExChange(SqQueue \*&q) //倒置环队 (第12题)

{

int e;

LinkStNode \*p;

InitStack(p);

while(!QueueEmpty(q))

{

deQueue(q,e);

Push(p,e);

}

while(!StackEmpty(p))

{

Pop(p,e);

enQueue(q,e);

}

}

int main(){

int a[Maxsize];

int e;

SqQueue \*q;

InitQueue(q);

printf("入队元素：");

for(int i =0;i<10;i++)

{

scanf("%d",&a[i]);

enQueue(q,a[i]);

}

ExChange(q);

printf("倒置后：");

while(!QueueEmpty(q))

{

deQueue(q,e);

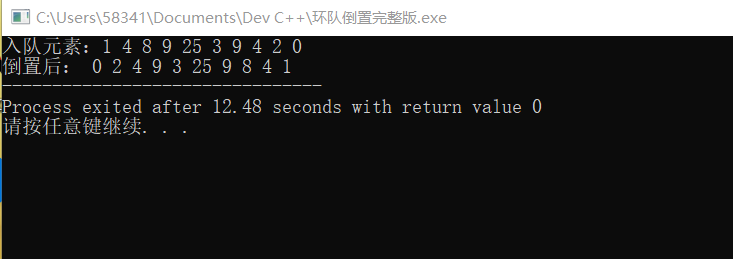
printf(" %d",e);

}

return 0;

}

测试结果



**13.**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 10

typedef struct node

{

int data;

struct node \*next;

}DataNode;

void EnDataNode(DataNode \*head[], DataNode \*tail[], int e) //将元素e插入第 e个队列中

{

DataNode \*s;

s = (DataNode \*)malloc(sizeof(DataNode));

s->data = e; s->next = NULL;

if (head[e] == NULL)

head[e] = tail[e] = s;

else

{

tail[e]->next = s;

tail[e] = s;

}

}

void CreDataNode(DataNode \*head[], DataNode \*tail[]) //创建所需队列

{

int n, e;

printf("n: ");

scanf("%d", &n);

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

{

printf("第%d个数为： ", i + 1);

scanf("%d", &e);

if (e >= 0 && e <= 9)

EnDataNode(head, tail, e);

}

}

void DestroyDataNode(DataNode \*&L) //销毁链表

{

DataNode \*pre = L, \*p = pre->next;

if (p != NULL)

{

free(pre);

pre = p;

p = pre->next;

}

free(pre);

}

void PutDataNode(DataNode \*L) //输出列链元素

{

printf("\n输出链表： ");

while (L != NULL)

{

printf("%d", L->data);

L = L->next;

}

}

DataNode\* CobDataNode(DataNode \*head[], DataNode \*tail[]) //合并队列

{

DataNode \*L = NULL, \*p;

for (int i = 0; i<Maxsize; i++)

{

if (head[i] != NULL)

{

if (L == NULL)

{

L = head[i];

p = tail[i];

}

else

{

p->next = head[i];

p = tail[i];

}

}

}

p->next = NULL;

return L;

}

int main()

{

DataNode \*L;

DataNode \*head[Maxsize], \*tail[Maxsize];

for (int i = 0; i<Maxsize; i++)

head[i] = tail[i] = NULL;

CreDataNode(head, tail);

L = CobDataNode(head, tail);

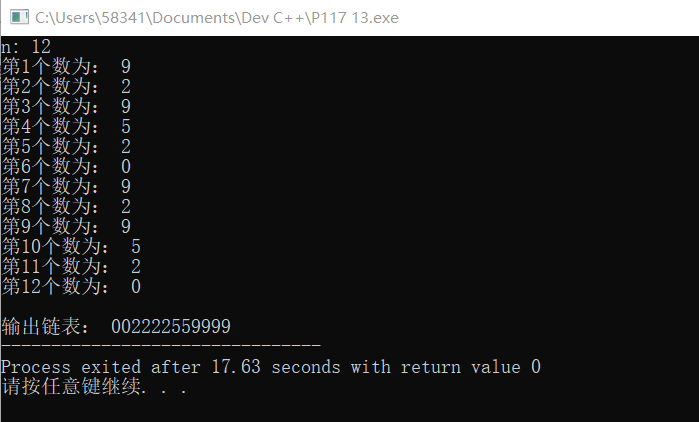
PutDataNode(L);

DestroyDataNode(L);

return 0;

}

测试结果



**第四章**

**12(最长重复子串)**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 100

typedef struct

{

char data[Maxsize];

int length;

} SqString;

void StrAssign(SqString &s,char str[]) //生成串

{

int i;

for(i=0;str[i]!='\0';i++)

s.data[i] = str[i];

s.length = i;

}

void CompSqString(SqString &s) //求最长重复子串

{

int i = 0,j = s.length-1,max = 0,maxlength = 0,length;

while(i<s.length)

{

j = i+1;

while(j<s.length)

{

if(s.data[i] == s.data[j])

{

length = 1;

for(int k=1;s.data[i+k] == s.data[j+k];k++)

length++;

if(length>maxlength)

{

maxlength = length;

max = i;

}

j += length;

}

else j++;

}

i++;

}

printf("%d %d",max,maxlength);

}

int main()

{

char s[Maxsize];

SqString ss;

int i=0;

while((s[i] = getchar())!='\n')

i++;

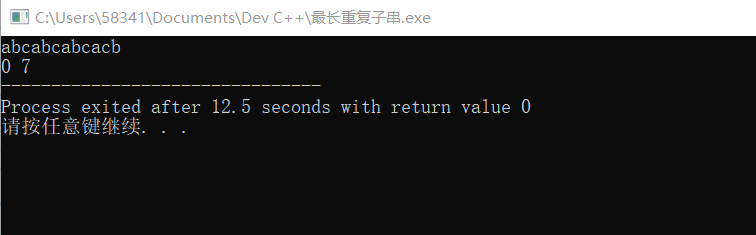
StrAssign(ss,s);

CompSqString(ss);

return 0;

}

测试结果



**12.**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 100

typedef struct snode

{

char data;

struct snode \*next;

} LinkStrNode;

void StrAssign(LinkStrNode \*&s,char cstr[]) //生成链串

{

int i;

LinkStrNode \*r,\*p;

s = (LinkStrNode \*)malloc(sizeof(LinkStrNode));

r = s;

for(i=0;cstr[i]!='\0';i++)

{

p = (LinkStrNode \*)malloc(sizeof(LinkStrNode));

p->data = cstr[i];

r->next = p;

r = p;

}

r->next=NULL;

}

void DispStr(LinkStrNode \*s) //输出串

{

LinkStrNode \*p = s->next;

while(p!=NULL)

{

printf("%c ",p->data);

p = p->next;

}

printf("\n");

}

void ChooseStr(LinkStrNode \*&s) //删除x元素

{

LinkStrNode \*p,\*q;

p = s;

while(p->next!=NULL)

{

q = p->next;

if(q->data == 'x')

{

p->next = q->next;

free(q);

}

else

p = p->next;

}

}

int main(void)

{

int i=0;

char s[Maxsize];

LinkStrNode \*L;

while((s[i] = getchar())!='\n')

i++;

StrAssign(L,s);

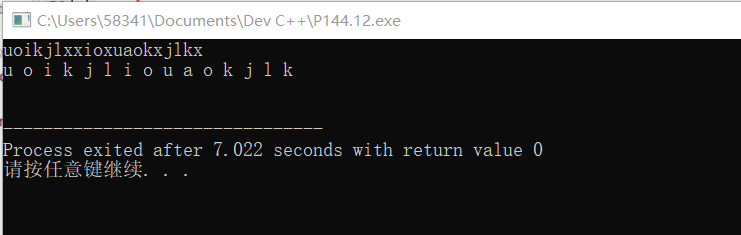
ChooseStr(L);

DispStr(L);

return 0;

}

测试结果



**第五章**

**P162**

**4.**

#include<stdio.h>

double Average(double a[],int n)

{

if(n==2)

return (a[0]+a[1])/2;

else

return (Average(a,n-1)\*(n-1)+a[n-1])/n;

}

int main()

{

double f;

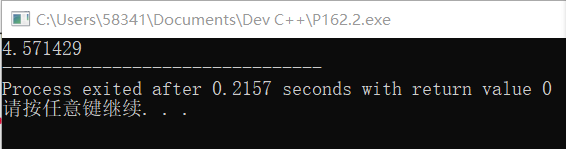
double a[7] = {1,2,3,4,5,9,8};

f = Average(a,7);

printf("%f",f);

}

测试结果



**3.**

#include<stdio.h>

int Count(int n)

{

if(n/10 == 0)

return 1;

else

return Count(n/10)+1;

}

int main()

{

int n;

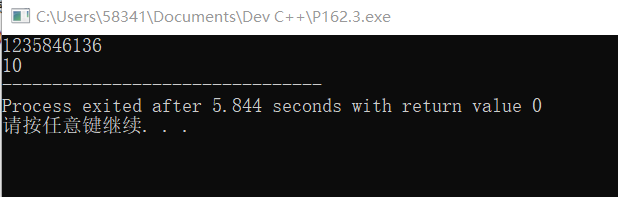
scanf("%d",&n);

n = Count(n);

printf("%d",n);

}

测试结果



**5.求逆串**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 10001

typedef struct

{

char data[Maxsize];

int length;

} SqString;

void StrAssign(SqString &s,char ch[]) //生成串

{

int i;

for(i=0;ch[i] != '\0';i++)

s.data[i] = ch[i];

s.length = i;

}

void StrCopy(SqString &s,SqString t) //串的复制

{

for(int i=0;i<t.length;i++)

s.data[i] = t.data[i];

s.length = t.length;

}

SqString Concat(SqString s,SqString t) //串的链接

{

SqString str;

int i;

str.length = s.length+t.length;

for(i=0;i<s.length;i++)

str.data[i] = s.data[i];

for(i=0;i<t.length;i++)

str.data[s.length+i] = t.data[i];

return str;

}

SqString SubStr(SqString s,int i,int j) //求子串

{

SqString str;

str.length = 0;

if(i<=0||i>s.length||i+j-1>=s.length)

return str;

for(int k=i-1;k<i+j-1;k++)

{

str.data[k-i+1] = s.data[k];

}

str.length = j;

return str;

}

int StrLength(SqString s) //求串的长度

{

return s.length;

}

SqString StrExchange(SqString s) //串倒置

{

SqString p,q;

if(StrLength(s)>0)

{

p=StrExchange(SubStr(s,2,StrLength(s)-1));

q=Concat(p,SubStr(s,1,1));

}

else

q=s;

return q;

}

int main() //用于测试的主函数

{

char c[Maxsize];

printf("请输入需要求逆的串:");

scanf("%s",c);

SqString s,t;

StrAssign(s,c);

t=StrExchange(s);

printf("求逆后:");

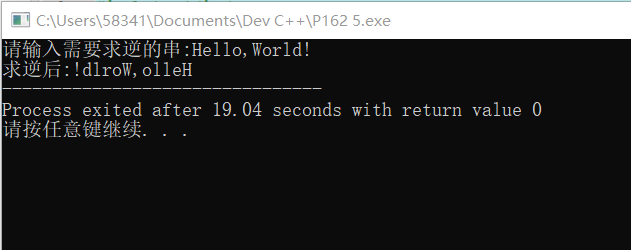
for(int i=0;i<t.length;i++)

printf("%c",t.data[i]);

return 0;

}

测试结果



**第六章**

**P186**

**实验一**

#include<stdio.h>

#include<stdlib.h>

#define M 4 //稀疏矩阵的行数

#define N 4 //稀疏矩阵的列数

#define MaxSize 16 //稀疏矩阵最多非零元素的个数

typedef int ElemType;

typedef struct

{

int r;

int c;

ElemType d;

} TupNode; //三元组类型

typedef struct

{

int rows;

int cols;

int nums;

TupNode data[MaxSize];

}TSMatrix; //三元组顺序表类型

void CreatMat(TSMatrix &t,ElemType A[M][N]) //创建三元组顺序表

{

int i,j;

t.rows = M;t.cols = N;t.nums = 0;

for(i=0;i<M;i++)

{

for(j=0;j<N;j++)

{

if(A[i][j] != 0)

{

t.data[t.nums].r = i;t.data[t.nums].c = j;t.data[t.nums].d = A[i][j];

t.nums++;

}

}

}

}

void DisMat(TSMatrix t) //输出三元组

{

int k;

if(t.nums<0)

return;

printf("\t%d\t%d\t%d\n",t.rows,t.cols,t.nums);

printf("\t-------------------------------\n");

for(k=0;k<t.nums;k++)

printf("\t%d\t%d\t%d\n",t.data[k].r,t.data[k].c,t.data[k].d);

}

void TranTat(TSMatrix t,TSMatrix &tb) //求矩阵转置的三元组

{

int k,kl = 0,v;

tb.rows = t.cols;tb.cols = t.rows;tb.nums = t.nums;

if(t.nums != 0)

{

for(v=0;v<t.cols;v++)

{

for(k=0;k<t.nums;k++)

if(t.data[k].c == v)

{

tb.data[k].r = t.data[k].c;

tb.data[k].c = t.data[k].r;

tb.data[k].d = t.data[k].d;

kl++;

}

}

}

}

void SumMat(ElemType A[M][N],ElemType B[M][N],TSMatrix &t) //求A+B的三元组

{

ElemType C[M][N] = {0};

for(int i=0;i<M;i++)

{

for(int j=0;j<N;j++)

C[i][j] = A[i][j] + B[i][j];

}

CreatMat(t,C);

}

void MultMat(ElemType A[M][N],ElemType B[M][N],TSMatrix &t) //求AXB的三元组

{

ElemType C[M][N] = {0};

for(int i=0;i<M;i++)

{

for(int j=0;j<N;j++)

{

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

C[i][j] = C[i][j] + A[i][k] \* B[k][j];

}

}

CreatMat(t,C);

}

int main()

{

ElemType A[M][N]={0},B[M][N]={0};

TSMatrix a,b,c,d,e;

int i,j;

printf("输入a矩阵的元素：\n");

for(i=0;i<M;i++) //输入矩阵A

{

for(j=0;j<N;j++)

scanf("%d",&A[i][j]);

}

printf("输入b矩阵的元素：\n");

for(i=0;i<M;i++) //输入矩阵B

{

for(j=0;j<N;j++)

scanf("%d",&B[i][j]);

}

CreatMat(a,A);

CreatMat(b,B);

TranTat(a,c);

SumMat(A,B,d);

MultMat(A,B,e);

printf("a矩阵转置的三元组：\n");

DisMat(c);

printf("a+b的三元组：\n");

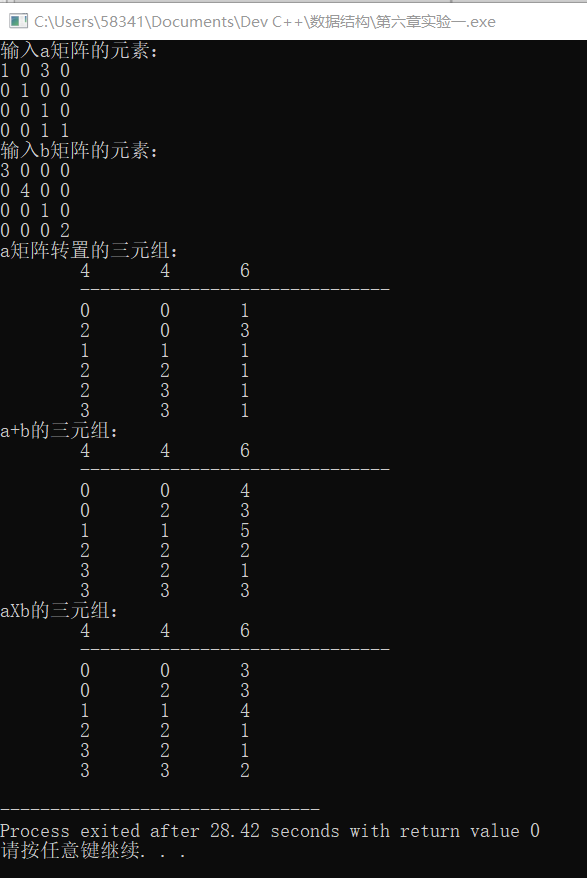
DisMat(d);

printf("aXb的三元组：\n");

DisMat(e);

}

运行结果



**实验二**

#include<stdio.h>

void Sum(int a[],int b[],int c[],int m) //求可逆矩阵a+b的和

{

int n;

n = m\*(m+1)/2;

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

c[i] = a[i] + b[i];

}

void Multiply(int a[],int b[],int c[],int m) //求可逆矩阵a\*b的乘积

{

int n=0,p=0,q=0;

for(int i=0;i<m;i++)

{

for(int j=0;j<m;j++)

{

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

{

if(i<k) p = k\*(k+1)/2 + i;

else p = i\*(i+1)/2 + k;

if(j<k) q = k\*(k+1)/2 + j;

else q = j\*(j+1)/2 + k;

c[n] = c[n] + (a[p] \* b[q]);

}

n++;

}

}

}

int main()

{

int i,n,m;

printf("请输入对称矩阵行列数：");

scanf("%d",&n);

m = n\*(n+1)/2;

int a[m],b[m],c[m]={0},d[n\*n]={0};

printf("请依次输入对称矩阵a下三角部分元素：");

for(int i=0;i<m;i++)

scanf("%d",&a[i]);

printf("请依次输入对称矩阵b下三角部分元素：");

for(int i=0;i<m;i++)

scanf("%d",&b[i]);

Sum(a,b,c,n);

printf("a+b为：\n");

for(int i=0;i<m;i++)

printf("%d ",c[i]);

printf("\n");

Multiply(a,b,d,n);

printf("a\*b为：\n");

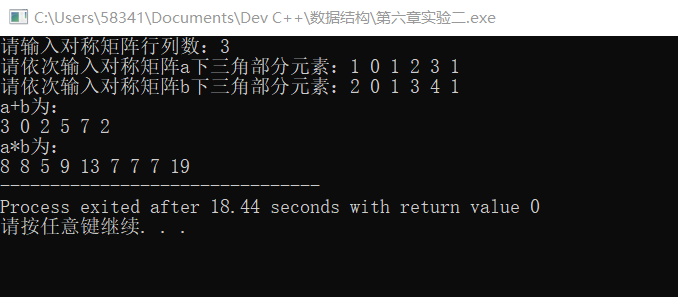
for(i=0;i<(n\*n);i++)

printf("%d ",d[i]);

if((i+1)%n == 0) printf("\n");

}

测试结果

****

**第七章**

**10-20题**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MaxSize 30

typedef char ElemType;

typedef ElemType SqBinTree[MaxSize]; //二叉树的舒徐储存结构

typedef struct node //二叉树的链式储存结构

{

ElemType data;

struct node\* lchild;

struct node\* rchild;

}BTNode;

void CreateBTree(BTNode \*&b, char \*str) //创建二叉树

{

BTNode \* St[MaxSize], \*p;

int top = -1, k, j = 0;

char ch;

b = NULL;

ch = str[j];

while (ch != '\0')

{

switch (ch)

{

case '(': top++; St[top] = p; k = 1; break;

case ')': top--; break;

case ',': k = 2; break;

default:

p = (BTNode\*)malloc(sizeof(BTNode));

p->data = ch;

p->lchild = p->rchild = NULL;

if (b == NULL)

b = p;

else

{

switch (k)

{

case 1:St[top]->lchild = p; break;

case 2:St[top]->rchild = p; break;

}

}

}

j++;

ch = str[j];

}

}

void DispBTree(BTNode \*b) //输出二叉树

{

if (b != NULL)

{

printf("%c", b->data);

if (b->lchild != NULL || b->rchild != NULL)

{

printf("(");

DispBTree(b->lchild);

if (b->rchild != NULL)

printf(",");

DispBTree(b->rchild);

printf(")");

}

}

}

void PreOrder(BTNode \*b) //先序遍历

{

if (b != NULL)

{

printf("%c", b->data);

PreOrder(b->lchild);

PreOrder(b->rchild);

}

}

void TransformBTree(BTNode \*b, SqBinTree a, int n) //将链式储存结构转化成顺序储存结构(10)

{

if (b != NULL)

{

BTNode \*p = b->lchild;

BTNode \*q = b->rchild;

a[n] = b->data;

TransformBTree(p, a, 2 \* n);

TransformBTree(q, a, 2 \* n + 1);

}

else

a[n] = '#';

}

int NumleafNode(SqBinTree a) //求顺序储存二叉树中的叶子结点个数(11)

{

int i,count=0;

for (i = 1; i < MaxSize; i++)

{

if (a[i] != '#' && a[2 \* i] == '#' && a[2 \* i + 1] == '#')

count++;

}

return count;

}

void NumsingleNode(BTNode \*b,int &n) //求链式储存二叉树中的单分支结点个数(12)

{ //n初值为0

if (b != NULL)

{

if ((b->lchild == NULL && b->rchild != NULL) || (b->lchild != NULL && b->rchild == NULL))

n++;

NumsingleNode(b->lchild,n);

NumsingleNode(b->rchild,n);

}

else

return;

}

void MinNode(BTNode \*b, ElemType &e) //求链式储存二叉树中的最小结点值(13)

{

if (b != NULL)

{

if (b->lchild == NULL && b->rchild == NULL);

else if (b->lchild == NULL)

{

if (b->rchild->data < e)

e = b->rchild->data;

}

else if (b->rchild == NULL)

{

if (b->lchild->data < e)

e = b->lchild->data;

}

else

{

e = e < b->lchild->data ? e:b->lchild->data;

e = e < b->rchild->data ? e:b->rchild->data;

}

MinNode(b->lchild, e);

MinNode(b->rchild, e);

}

else

return;

}

void PutMinNode(BTNode \*b) //输出最小值结点(13)

{

char e = b->data;

MinNode(b, e);

printf("第13题 最小结点值为:%c\n", e);

}

void CopyBTree(BTNode \*b1, BTNode \*&b2) //二叉链复制(14)

{

if (b1 == NULL)

b2 = NULL;

else

{

b2 = (BTNode\*)malloc(sizeof(BTNode));

b2->data = b1->data;

CopyBTree(b1->lchild, b2->lchild);

CopyBTree(b1->rchild, b2->rchild);

}

}

void KleafNode(BTNode \*b, int k,int &n) //第k层叶子结点个数(15)

{ //n初值为0

if (b != NULL)

{

if (k == 1)

{

if (b->lchild == NULL && b->rchild == NULL)

n++;

return;

}

else

{

k--;

KleafNode(b->lchild, k, n);

KleafNode(b->rchild, k, n);

}

}

else

return;

}

bool BrotherNode(BTNode \*b, ElemType e1, ElemType e2) //判断两个结点是否为兄弟结点(16)

{

if (b != NULL)

{

if (b->lchild != NULL && b->rchild != NULL)

{

if ((b->lchild->data == e1 && b->rchild->data == e2) || (b->rchild->data == e1 && b->lchild->data == e2))

return true;

else

{

BrotherNode(b->lchild, e1, e2);

BrotherNode(b->rchild, e1, e2);

}

}

else

return false;

}

else

return false;

}

void XSonNode(BTNode \*b, ElemType e) //求值为e的结点的先序遍历)(17)

{

if (b != NULL)

{

if (b->data == e)

{

PreOrder(b);

}

else

{

XSonNode(b->lchild, e);

XSonNode(b->rchild, e);

}

}

}

void SwapSonTree(BTNode \*b1,BTNode \*&b2) //交换子树(18)

{

if (b1 == NULL)

b2 = NULL;

else

{

b2 = (BTNode\*)malloc(sizeof(BTNode));

b2->data = b1->data;

SwapSonTree(b1->lchild, b2->rchild);

SwapSonTree(b1->rchild, b2->lchild);

}

}

bool SameBTree(BTNode \*b1,BTNode \*b2) //判断两棵树是否同构(19)

{

if (b1 == NULL && b2 == NULL)

return true;

if (b1 == NULL || b2 == NULL)

return false;

else

return SameBTree(b1->lchild, b2->lchild) && SameBTree(b1->rchild, b2->rchild);

}

bool SamesonTree(BTNode \*b) //判断一棵树的左右子树是否同构(19)

{

if (b == NULL)

return false;

else

return SameBTree(b->lchild, b->rchild);

}

bool CompBTree(BTNode \*b) //判断一棵树是否为完全二叉树(20)

{

bool lf = true,sp = true, rear = -1, front = -1;

BTNode \*p,\*Q[MaxSize];

if (b == NULL)

return true;

rear++;

Q[rear] = b;

while (front != rear)

{

front++;

p = Q[front];

if (p->lchild == NULL)

{

lf = false;

if (p->rchild != NULL)

sp = false;

}

else

{

if (!lf)

sp = false;

else

{

rear++;

Q[rear] = p->lchild;

if (p->rchild == NULL)

lf = false;

else

{

rear++;

Q[rear] = p->rchild;

}

}

}

}

return sp;

}

int main()

{

char s[MaxSize],e;

int m,n=0,o=0;

BTNode \*b,\*c,\*d;

SqBinTree a;

memset(a, '#', sizeof(a) / sizeof(a[0]));

printf("请输入二叉树:");

scanf("%s", s);

CreateBTree(b, s);

TransformBTree(b, a, 1); //第10题

printf("第10题 转化成的顺序储存结构:\n");

for (int i = 1; i<MaxSize; i++)

{

printf("%d %c\n", i, a[i]);

}

printf("第11题 叶子节点个数:%d\n", NumleafNode(a)); //第11题

NumsingleNode(b, n); //第12题

printf("第12题 单分支结点个数:%d\n", n);

PutMinNode(b); //第13题

CopyBTree(b, c); //第14题

printf("第14题 求已知二叉链的复制二叉链:");

DispBTree(c);

KleafNode(b, 3, o);

printf("\n第15题 求第3层叶子结点个数:%d\n",o); //第15题

if (BrotherNode(b, 'G', 'E')) //第16题

printf("第16题 GE为兄弟结点.\n");

printf("第17题 B结点子孙的先序遍历为:"); //第17题

XSonNode(b, 'B');

printf("\n第18题 交换前:"); //第18题

DispBTree(b);

printf("\n 交换后:");

SwapSonTree(b, d);

DispBTree(d);

if (SamesonTree(b)) //第19题

printf("\n第19题 左右子树同构.");

else

printf("\n第19题 左右子树不同构.");

if (CompBTree(b)) //第20题

printf("\n第20题 不为完全二叉树.");

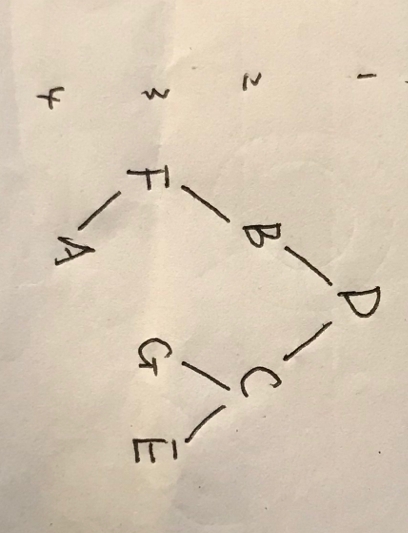
else

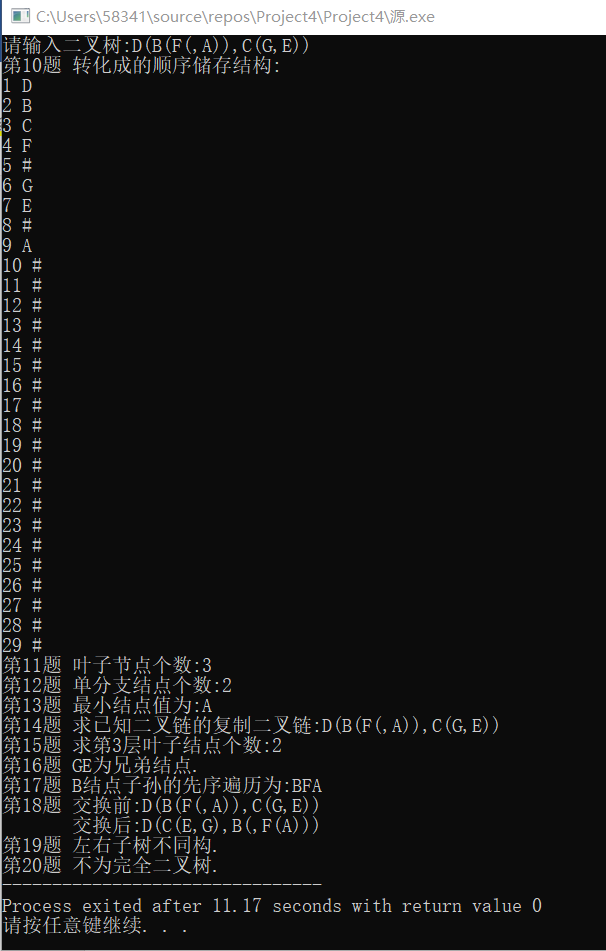
printf("\n第20题 为完全二叉树.");

return 0;

}

用于测试的树





**第八章**

**实验一 实现图的邻接矩阵和邻接表储存**

**实验二 实现图的遍历算法**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAXV 20

#define MaxSize 200

typedef struct

{

int no;

char data[MAXV];

}VertexType;

typedef struct

{

int edges[MAXV][MAXV];

int n, e; /\*顶点数，边数\*/

VertexType vexs[MAXV];

}MatGraph;

typedef struct ANode

{

int adjvex;

struct ANode\* nextarc;

int weight;

}ArcNode;

typedef struct Vnode

{

char data[MAXV];

ArcNode \* firstarc;

}VNode;

typedef struct

{

VNode adjlist[MAXV];

int n,e;

}AdjGraph;

void CreateMat(MatGraph \*g) //创建临接矩阵并输出

{

int u, v, w,i,j;

memset(g->edges, -1, sizeof(g->edges));

printf("请输入顶点数和边数:");

scanf("%d %d", &g->n, &g->e);

printf("请输入邻接边信息:");

printf("起始u 终止v 权值w\n");

for (i = 0; i < g->e; i++)

{

scanf("%d %d %d", &u, &v, &w);

g->edges[u][v] = w;

}

for (i = 0; i < g->n; i++)

{

g->edges[i][i] = 0;

}

printf("邻接矩阵为:\n");

for (i = 0; i < g->n; i++)

{

for (j = 0; j < g->n; j++)

{

if (j == g->n - 1)

printf("%3d\n", g->edges[i][j]);

else

printf("%3d ", g->edges[i][j]);

}

}

}

void CreateAdj(AdjGraph \*&G,MatGraph g) //创建图的邻接表

{

int i,j,n,e;

n = g.n;e = g.e;

ArcNode \*p;

G = (AdjGraph\*)malloc(sizeof(AdjGraph));

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

G->adjlist[i].firstarc = NULL;

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

for(j=n-1;j>=0;j--)

{

if(g.edges[i][j] != 0 && g.edges[i][j] != -1)

{

p = (ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex = j;

p->weight = g.edges[i][j];

p->nextarc = G->adjlist[i].firstarc;

G->adjlist[i].firstarc = p;

}

}

G->n = n;G->e = e;

}

void DispAdj(AdjGraph \*G) //输出邻接表

{

int i;

ArcNode \*p;

for(i=0;i<G->n;i++)

{

p = G->adjlist[i].firstarc;

printf("%3d",i);

while(p!=NULL)

{

printf("%3d[%d]->",p->adjvex,p->weight);

p = p->nextarc;

}

printf("/\\\n");

}

}

void DestroyAdj(AdjGraph \*&G) //销毁邻接表

{

ArcNode \*pre,\*p;

for(int i=0;i<G->n;i++)

{

pre = G->adjlist[i].firstarc;

if(pre!=NULL)

{

p = pre->nextarc;

while(p!=NULL)

{

free(pre);

pre = p;p = p->nextarc;

}

free(pre);

}

}

free(G);

printf("邻接表销毁成功!");

}

int visited[MAXV] = {0};

void DFS(AdjGraph \*G,int v) //深度优先搜索（递归）

{

ArcNode \*p;

visited[v] = 1;

printf("%d ",v);

p = G->adjlist[v].firstarc;

while(p!=NULL)

{

if(visited[p->adjvex] == 0)

DFS(G,p->adjvex);

p = p->nextarc;

}

}

void DFS2(AdjGraph \*G,int v) //深度优先搜索（非递归）

{

ArcNode \*p;

int V[MAXV];

int top = -1,x,y,i;

for(i=0;i<G->n;i++)

visited[i] = 0;

printf("%d ",v);

visited[v] = 1;

top++;V[top] = v;

while(top>-1)

{

x = V[top];

p = G->adjlist[x].firstarc;

while(p!=NULL)

{

y = p->adjvex;

if(visited[y]==0)

{

printf("%d ",y);

visited[y] = 1;

top++;

V[top] = y;

break;

}

p = p->nextarc;

}

if(p==NULL)

top--;

}

}

void BFS(AdjGraph \*G,int v) //广度优先搜索

{

ArcNode \*p;

int Que[MAXV],front=0,rear=0;

int visited[MAXV];

int x,i;

for(i=0;i<G->n;i++) visited[i] = 0;

printf("%d ",v);

visited[v] = 1;

rear = (rear+1)%MAXV;

Que[rear] = v;

while(front!=rear)

{

front = (front+1)%MAXV;

x = Que[front];

p = G->adjlist[x].firstarc;

while(p!=NULL)

{

if(visited[p->adjvex]==0)

{

printf("%d ",p->adjvex);

visited[p->adjvex] = 1;

rear = (rear+1)%MAXV;

Que[rear] = p->adjvex;

}

p = p->nextarc;

}

}

}

int main()

{

MatGraph g;

AdjGraph \*G;

printf("实验一:\n");

CreateMat(&g);

CreateAdj(G,g);

printf("\n邻接表为:\n");

DispAdj(G);

printf("实验二:\n");

printf("DFS(递归):");

DFS(G,0);

printf("\nDFS(非递归):");

DFS2(G,0);

printf("\nBFS:");

BFS(G,0);

printf("\n");

DestroyAdj(G);

return 0;

}

测试结果

