#代码仓库 <https://github.com/TheLostIn/database__2016-2017.git>

#authorized by Jane

第一次上机实验报告

调试成功程序及说明

1.

题目：

算法思想：照着书上一个一个敲

运行结果：编译无错误

附源程序：

有两个程序

1.

#include<iostream>

using namespace std;

#define LIST\_INIT\_SIZE 100

#define LIST\_INCREMENT 10

#define ElemType int

typedef struct {

ElemType \*elem;

int length;

int listsize;

} SqList;

class List{

public:

void InitList(SqList &);

void DestroyList(SqList &);

void ClearList(SqList &);

int ListLength(SqList);

void GetElem(SqList,int,int &);

void LocateElem(SqList,int);

void ListInsert(SqList &,int,int);

void ListDelete(SqList &,int,ElemType &);

void TraverseList(SqList);

void Error(char \*s);

void Increment(SqList &);

};

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LNode;

typedef LNode \*LinkList;

class L\_List

{

public:

void InitList(LinkList &L);

LNode \* LocateElem(LinkList L,ElemType e);

void Error(char \*s);

void DestroyList(LinkList &L);

void ClearList(LinkList &L);

void GetElem(LinkList L,int i,ElemType &e);

void ListInsert(LinkList &L,int i,ElemType e);

void ListDelete(LinkList &L,int i,ElemType &e);

void TraverseList(LinkList L);

};

void List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void List::InitList(SqList &L)

{

L.elem = new ElemType[LIST\_INIT\_SIZE];

if(!L.elem) Error("Overdlow");

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

}

void List::DestroyList(SqList &L)

{

delete []L.elem;

L.length = 0;

L.listsize = 0;

}

void List::ClearList(SqList &L)

{

L.length = 0;

}

int List::ListLength(SqList L)

{

return L.length;

}

void List::GetElem(SqList L,int i,ElemType &e)

{

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

Error("Position Error");

e = L.elem[i-1];

}

int LocateElem(SqList L,ElemType e)

{

int i = 1,\*p;

p = L.elem;

while((i<=L.length)&&(\*p++!=e)) i++;

if(i<=L.length) return i;

else return 0;

}

void List::ListInsert(SqList &L,int i,ElemType e)

{

ElemType \*q,\*p;

if((i<1)||(i>L.length+1))

{

Error(" Position Error");

}

if(L.length>=LIST\_INIT\_SIZE)

{

Increment(L);

}

q = &(L.elem[i-1]);

for(p = &(L.elem[L.length-1]);p>=q;--p)

{

\*(p+1) = \*p;

}

\*q = e;

++L.length;

}

void List::Increment(SqList &L)

{

ElemType \*newlist;

int i;

newlist = new ElemType[L.listsize+LIST\_INCREMENT];

if(! newlist) Error(" Overflow");

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

{

newlist[i] = L.elem[i];

}

delete []L.elem;

L.elem = newlist;//移交空间首地址

L.listsize += LIST\_INCREMENT;

}

void List::ListDelete(SqList &L,int i,ElemType &e)

{

ElemType \*p,\*q;

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

{

Error("Position Error");

}

e = L.elem[i-1];

p = &(L.elem[i-1]);

q = L.elem + L.length -1;

for(++p;p <= q;++p)

{

\*(p-1) = \*p;

}

--L.length;

}

void List::TraverseList(SqList L)

{

int i;

ElemType \*p;

if(L.length != 0)

{

i = 1;

p = L.elem;

while(i <= L.length)

{

cout<<\*p++;

i++;

}

}

}

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

void L\_List::InitList(LinkList &L)

{

L = new LNode;

L -> next = NULL;

}

void L\_List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void L\_List::DestroyList(LinkList &L)

{

LNode \*p;

while(L)

{

p = L;

L = L->next;

delete p;

}

}

void L\_List::ClearList(LinkList &L)

{

LNode \*p,\*q;

p=L->next;

while(p)

{

q = p;

p = p->next;

delete q;

}

L->next = NULL;

}

int ListLength(LinkList L)

{

LNode \*p;

p = L;

int length = 0;

while(p->next)

{

length++;

p = p->next;

}

return length;

}

void L\_List::GetElem(LinkList L,int i,ElemType &e)

{

LNode \*p;

p = L->next;

int j = 1;

while(p&&(j<i))

{

p = p->next;

++j;

}

if(!p||(j>i)) Error("Position Error");

else e = p->data;

}

LNode \* L\_List::LocateElem(LinkList L,ElemType e)

{

LNode \*p;

p = L->next;

while(p&&(p->data!=e))

p = p->next;

return p;

}

void L\_List::ListInsert(LinkList &L,int i,ElemType e)

{

LNode \*p,\*s;

p = L;

int j = 0;

while(p&&(j<i - 1))

{

p = p -> next;

++j;

}

if(!p||(j > i-1))

{

Error("Position Error");

}

else

{

s = new LNode;

s -> data = e;

s -> next = p->next;

p -> next = s;

}

}

void L\_List::ListDelete(LinkList &L,int i,ElemType &e)

{

int j;

LNode \*p,\*q;

p = L;

j = 0;

while((p -> next)&&(j < i-1))

{

p = p->next;

++j;

}

if(!(p -> next)||(j > i-1))

{

Error("Position Error");

}

q = p -> next;

e = q -> data;

p -> next = q -> next;

delete q;

}

void L\_List::TraverseList(LinkList L)

{

LNode \*p;

p = L-> next;

while(p)

{

cout<<p->data;

p = p->next;

}

}

int main()

{

return 0;

}

2.

#include<iostream>

using namespace std;

#define ElemType int

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LNode;

typedef LNode \*LinkList;

class L\_List

{

public:

void InitList(LinkList &L);

LNode \* LocateElem(LinkList L,ElemType e);

void Error(char \*s);

void DestroyList(LinkList &L);

void ClearList(LinkList &L);

void GetElem(LinkList L,int i,ElemType &e);

void ListInsert(LinkList &L,int i,ElemType e);

void ListDelete(LinkList &L,int i,ElemType &e);

void TraverseList(LinkList L);

};

void L\_List::InitList(LinkList &L)

{

L = new LNode;

L -> next = NULL;

}

void L\_List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void L\_List::DestroyList(LinkList &L)

{

LNode \*p;

while(L)

{

p = L;

L = L->next;

delete p;

}

}

void L\_List::ClearList(LinkList &L)

{

LNode \*p,\*q;

p=L->next;

while(p)

{

q = p;

p = p->next;

delete q;

}

L->next = NULL;

}

int ListLength(LinkList L)

{

LNode \*p;

p = L;

int length = 0;

while(p->next)

{

length++;

p = p->next;

}

return length;

}

void L\_List::GetElem(LinkList L,int i,ElemType &e)

{

LNode \*p;

p = L->next;

int j = 1;

while(p&&(j<i))

{

p = p->next;

++j;

}

if(!p||(j>i)) Error("Position Error");

else e = p->data;

}

LNode \* L\_List::LocateElem(LinkList L,ElemType e)

{

LNode \*p;

p = L->next;

while(p&&(p->data!=e))

p = p->next;

return p;

}

void L\_List::ListInsert(LinkList &L,int i,ElemType e)

{

LNode \*p,\*s;

p = L;

int j = 0;

while(p&&(j<i - 1))

{

p = p -> next;

++j;

}

if(!p||(j > i-1))

{

Error("Position Error");

}

else

{

s = new LNode;

s -> data = e;

s -> next = p->next;

p -> next = s;

}

}

void L\_List::ListDelete(LinkList &L,int i,ElemType &e)

{

int j;

LNode \*p,\*q;

p = L;

j = 0;

while((p -> next)&&(j < i-1))

{

p = p->next;

++j;

}

if(!(p -> next)||(j > i-1))

{

Error("Position Error");

}

q = p -> next;

e = q -> data;

p -> next = q -> next;

delete q;

}

void L\_List::TraverseList(LinkList L)

{

LNode \*p;

p = L-> next;

while(p)

{

cout<<p->data;

p = p->next;

}

}

int main()

{

return 0;

}

2.

题目：编程 建立元素值为整型的顺序表 并实现就地逆置

算法思想：

建立数组

将数组中间一位的左右元素交换

a[i]=a[i+(mid-i)\*2-1]

运行结果：

19 18 17 16….1 0

附源程序：

#include<iostream>

using namespace std;

int main()

{

int a[20],i,n,mid,temp;

n=20;

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

{

a[i]=i;

// cout<<a[i];

}

cout<<endl;

mid=n/2;

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

{

temp=a[i];

a[i]=a[i+(mid-i)\*2-1];

a[i+(mid-i)\*2-1]=temp;

cout<<i+(mid-i)\*2-1<<" ";

cout<<a[i]<<endl;

}

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

{

cout<<a[i]<<" ";

}

return 0;

}

3.

题目：编程 建立元素值为整型的单链表 并实现就地逆置

算法思想：

就地逆置 通过循环将第n个元素插入head与第一个元素之间

运行结果：

显示 19 18 17 16 15……1 0

附源程序：

#include<iostream>

using namespace std;

struct List

{

int data;

List \* next;

};

int main()

{

void init(List &L);

void rev(List &L);

void buildL(List &L);

List head;

init(head);

List \* p;

buildL(head);

p=head.next;

rev(head);

return 0;

}

void init(List &L)

{

// L =new List;

L.next = NULL;

}

void buildL(List &L)

{

int i;

List \*p,\*cur;

cur = &L;

for(i=0;i<20;i++)

{

p = new List;

p->next = NULL;

p->data = i;

cur->next = p;

cur = p;

}

}

void rev(List &L)

{

List \* p;

List \* q;

List \* s;

List \* head;

head = &L;

p=head->next;

q=p->next;

p->next = NULL;

while(q!=NULL)

{

s=q->next;

p=head->next;

head->next = q;

q->next = p;

q=s;

}

p=L.next;

while(p!=NULL)

{

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

p=p->next;

}

}

4.

题目：约瑟夫环

算法思想：

新建一个struct 存放data flag

data 默认为信息 flag 为标志 若flag 为1 则 已出局 为 0 存活

计数器 count 计算口号 1 2 3 。。。。

s=8

s=s%人数

从第s个开始计数 先循环s-1次 开始计数

t为循环的数字 该程序中默认为5

若flag=0;

count++;

若count%t==0

则出局flag=1

使用循环链表

运行结果：

显示淘汰的过程

显示存活的号码

显示剩下一个人的时候所有人的flag

附源程序：

#include<iostream>

using namespace std;

struct joseph

{

int data;

int flag;

joseph \* next;

};

int main()

{

void init(joseph &);

void linger(joseph &);

void display(joseph &);

joseph head;

init(head);

display(head);

linger(head);

display(head);

return 0;

}

void init(joseph &head)

{

joseph \* jo;

joseph \* p;

head.data=1;

head.next=NULL;

head.flag=0;

p=&head;

int i;

for(i=2;i<21;i++)

{

jo = new joseph;

jo->data=i;

jo->flag=0;

jo->next=NULL;

p->next = jo;

p=p->next;

}

p->next=&head;

}

void display(joseph &head)

{

int i=1;

joseph \*p;

p=&head;

while(i<21)

{

cout<<"p->data "<<p->data<<" p->flag "<<p->flag<<"\n";

p=p->next;

i++;

}

cout<<'\n';

}

//pretenu circle 4

// strat from s=8

void linger(joseph &head)

{

int total,count,mod,s;

joseph \*p;

p=&head;

count=0;

total=20;

mod=4;

s = 8;

s = s%total;

while(s>1)

{

p=p->next;

s--;

}

while(total>1)

{

if(!p->flag)

{

count++;

if(count%mod==0)

{

p->flag=1;

total--;

cout<<p->data<<" ";

}

}

p=p->next;

}

while(p->flag)

{

p=p->next;

}

cout<<"\nsurvive: "<<p->data<<"\n";

}

第二次上机实验报告

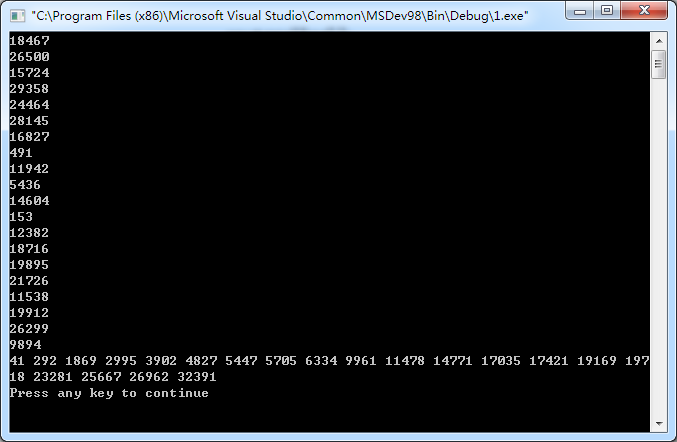
1.

题目：建立元素值为整型的顺序表，编程，用插入排序算法实现顺序表排序。

算法思想：

插入排序 从1号位置开始从前往后查询 若查到a[j]>a[i]&&j<i将a[i]插入到j前面 temp=a[i] 先将a[j]~a[n]依次往后移一位 然后赋值a[j]=temp

运行结果：



附源程序：

#include<iostream>

using namespace std;

void create(int \*a)

{

int i=0;

for(i=0;i<20;i++)

{

a[i]=rand();

cout<<rand()<<endl;

}

}

void sort(int \*a)

{

int i,j,k,temp;

for(i=0;i<20;i++)

{

for(j=0;a[i]>a[j]&&j<i;j++)

{

}

if(j<i)

{

temp=a[i];

for(k=i;k>j;k--)

{

a[k]=a[k-1];

}

a[j]=temp;

}

}

for(i=0;i<20;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

}

int main()

{

int a[20];

create(a);

sort(a);

return 0;

}

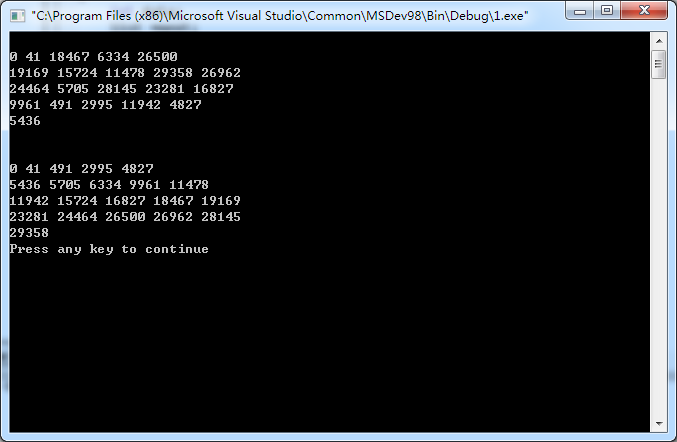
2.

题目：建立元素值为整型的单链表，编程，用插入排序算法实现单链表排序。

算法思想：

插入排序 从head开始从前往后查询 若查到p->data>q->data将p插入到q前面

运行结果：



附源程序：

#include<iostream>

using namespace std;

struct list

{

int data;

list \*next;

};

void create(list &l)

{

list \*p,\*q;

p=&l;

int i=0;

l.data=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new list;

q->data=rand();

q->next=NULL;

p->next=q;

p=q;

// cout<<p->data<<" ";

}

}

void sort(list &l)

{

list \*p,\*q,\*s,\*r;

if(l.next==NULL)

{

exit(1);

}else{

s=l.next;

p=s->next;

if(p->data<s->data)

{

s->next=p->next;

p->next=s;

l.next=p;

}

while(p->next!=NULL)

{

r=&l;

q=r->next;

while(q!=NULL&&p->data>q->data)

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

s=p;

p=p->next;

}

r=&l;

q=r->next;

while(q!=NULL&&p->data>q->data)

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

cout<<endl;

}

}

void print(list &l)

{

list \*p;

p=&l;

int i=0;

while(p!=NULL)

{

if(i%5==0)

cout<<endl;

cout<<p->data<<" ";

p=p->next;

i++;

}

cout<<endl;

}

int main()

{

list L;

create(L);

print(L);

sort(L);

print(L);

return 0;

}

3.

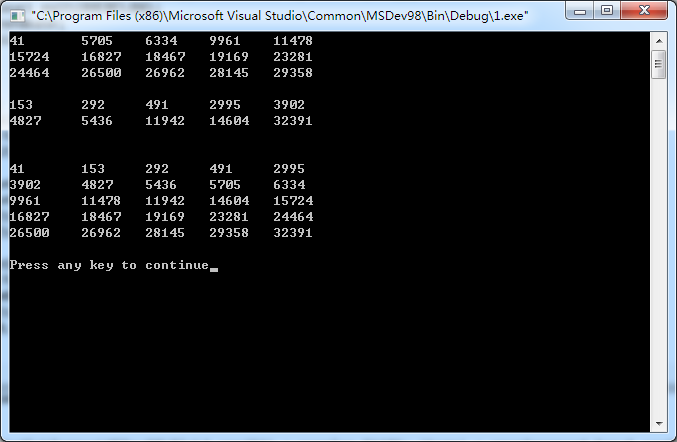
题目：

用顺序结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素也是依值递增有序排列。

算法思想：

归并排序

运行结果：

附上源程序：

#include<iostream>

#include<stdio.h>

#define OVERFLOW -2

using namespace std;

struct list

{

int \*elem;

int length;

int listsize;

};

void create(list &l,int n)

{

int i;

l.elem=(int \*)malloc(n\*sizeof(int));

if(!l.elem) exit(OVERFLOW);

l.length=n;

l.listsize=n;

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

{

l.elem[i]=rand();

}

}

void sort(int \*a,int n)

{

int i,j,k,temp;

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

{

for(j=0;a[i]>a[j]&&j<i;j++)

{

}

if(j<i)

{

temp=a[i];

for(k=i;k>j;k--)

{

a[k]=a[k-1];

}

a[j]=temp;

}

}

// cout<<endl;

}

void print(list &l)

{

int i;

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

{

cout<<l.elem[i]<<"\t ";

if((i+1)%5==0) cout<<endl;

}

cout<<endl;

}

void m\_sort(list &l1,list &l2)

{

int i=0,j=0,k,temp;

int \*a,\*b;

a=l1.elem;

b=l2.elem;

while(i<l1.length&&j<l2.length)

{

if(a[i]>b[j])

{

for(k=l1.length;k>i;k--)

{

a[k]=a[k-1];

}

a[i]=b[j];

j++;

l1.length++;

}else{

i++;

}

}

if(i==l1.length&&i<l1.listsize)

{

for(k=i;k<l1.listsize;k++)

{

a[k]=b[j];

// cout<<k<<" "<<j<<" ";

l1.length++;

j++;

}

}

}

void recreate(list &L1,int n)

{

int \*newbase;

newbase=(int \*)realloc(L1.elem,(L1.listsize+n)\*sizeof(int));

if(!newbase) exit(OVERFLOW);

L1.elem=newbase;

L1.listsize+=n;

}

int main()

{

int n=8;

list L1,L2;

create(L1,15);

create(L2,10);

sort(L1.elem,L1.length);

sort(L2.elem,L2.length);

print(L1);

print(L2);

recreate(L1,L2.length);

m\_sort(L1,L2);

cout<<endl;

print(L1);

return 0;

}

4.

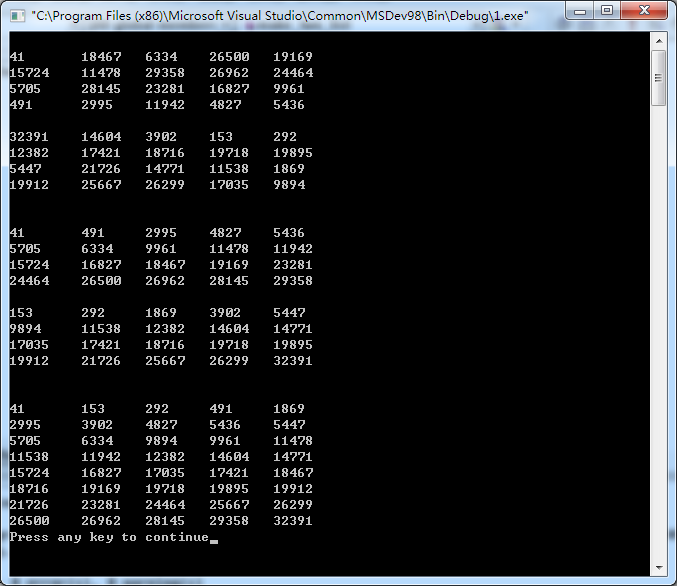
题目：

用单链表结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素也是依值递增有序排列。用单链表结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素也是依值递增有序排列。

算法思想：

归并排序

运行结果：



附上源程序：

#include<iostream>

using namespace std;

struct list

{

int data;

list \*next;

};

void create(list &l)

{

list \*p,\*q;

p=&l;

int i=0;

l.data=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new list;

q->data=rand();

q->next=NULL;

p->next=q;

p=q;

// cout<<p->data<<" ";

}

}

void sort(list &l)

{

list \*p,\*q,\*s,\*r;

if(l.next==NULL)

{

exit(1);

}else{

s=l.next;

p=s->next;

if(p->data<s->data)

{

s->next=p->next;

p->next=s;

l.next=p;

}

while(p->next!=NULL)

{

r=&l;

q=r->next;

while(q!=NULL&&p->data>q->data)

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

s=p;

p=p->next;

}

r=&l;

q=r->next;

while(q!=NULL&&p->data>q->data)

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

cout<<endl;

}

}

void print(list &l)

{

list \*p;

p=l.next;

int i=0;

while(p!=NULL)

{

cout<<p->data<<"\t ";

i++;

if(i%5==0) cout<<endl;

p=p->next;

}

}

void make\_two\_list(list &L1,list &L2)

{

create(L1);

create(L2);

cout<<endl;

print(L1);

cout<<endl;

print(L2);

sort(L1);

sort(L2);

print(L1);

cout<<endl;

print(L2);

cout<<endl;

}

void m\_sort(list &L1,list &L2,list &L3)

{

list \*p,\*q,\*m,\*r;

p=&L1;

q=L2.next;

m=&L3;

while(p->next!=NULL&&q!=NULL)

{

if((p->next->data)>q->data)

{

m=q->next;

q->next=p->next;

p->next=q;

q=m;

}else{

p=p->next;

}

}

if(q!=NULL)

{

p->next=q;

}

}

int main()

{

list L1,L2,L3;

L3.data=0;

L3.next=NULL;

make\_two\_list(L1,L2);

m\_sort(L2,L1,L3);

cout<<endl;

print(L1);

return 0;

}

5.

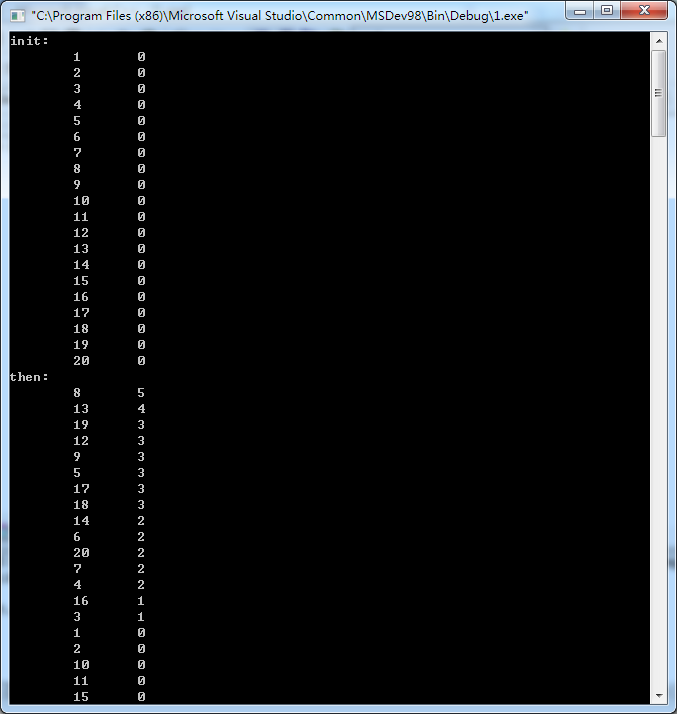
题目：

设有一个双向循环链表，每个结点中除有pre，data和next三个域外，还增设了一个访问频度域freq。在链表被起作用前，频度域freq的值均初始化为零，而当对链表进行一次LOCATE(L,x)的操作之后，被访问的结点（即元素值等于x的节点）中的频度域freq的值便增1，同时调整链表中结点之间的次序，使其按访问频度非递增的顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点。试编程实现之。

算法思想：

插入排序

运行结果：



附上源程序：

#include<iostream>

#include<stdio.h>

#define OVERFLOW -2

using namespace std;

struct both{

int data;

both \* pre;

both \* next;

int freq;

};

void create(both &l)

{

int i;

both \*p,\*q;

p=&l;

l.data=0;

l.freq=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new both;

q->data=i;

q->next=NULL;

q->pre=NULL;

q->freq=0;

p->next=q;

q->pre=p;

p=q;

}

}

void sort(both \*p,both &L)

{

both \*s,\*q,\*r;

s=p->pre;

while(((p->freq)>(s->freq))&&s!=&L)

{

s=s->pre;

}

if(p->next!=NULL)

{

p->pre->next=p->next;

p->next->pre=p->pre;

p->pre=s;

p->next=s->next;

s->next->pre=p;

s->next=p;

}else{

p->pre->next=p->next;

p->pre=s;

p->next=s->next;

s->next->pre=p;

s->next=p;

}

}

void Locate(both &L,int x)

{

int flag=0;

both \*p,\*q;

p=L.next;

while(p!=NULL)

{

if(p->data==x)

{

p->freq++;

sort(p,L);

flag=1;

}

p=p->next;

}

}

void print(both &L)

{

both \*p;

p=L.next;

while(p!=NULL)

{

cout<<"\t"<<p->data<<"\t"<<p->freq<<endl;

p=p->next;

}

}

int main()

{

both head;

int i,t;

create(head);

cout<<"init:"<<endl;

print(head);

int n=rand()%200;

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

{

t=rand()%21;

Locate(head,t);

}

cout<<"then:"<<endl;

print(head);

return 0;

}

第三次上机实验报告

1.

题目：编程实现书P59 ADT Stacke基本操作9个，用顺序存储结构实现；

算法思想：

照着书上一个一个敲

运行结果:

编译无bug

附源代码：

#include<iostream>

using namespace std;

typedef int Status;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int ElemType;

typedef struct{

ElemType \* base;

ElemType top;

int stacksize;

}SqStack;

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

Status InitStack(SqStack &S);

Status DestroyStack(SqStack &S);

Status ClearStack(SqStack &S);

int StackLength(SqStack &S);

Status GetTop(SqStack &S,ElemType &e);

Status Push(SqStack &S,ElemType e);

Status Pop(SqStack &S,ElemType &e);

Status StackTraverse(SqStack &S,Status (\*visit)());

void Increment(SqStack &S);

//从栈底到栈顶依次对栈中的每个元素调用visit()

//--------------------------------------------------

Status InitStack(SqStack &S)

{

S.base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if(!S.base) exit(OVERFLOW);

S.top = -1;

S.stacksize = STACK\_INIT\_SIZE;

return OK;

}

Status DestroyStack(SqStack &S){

delete []S.base;

S.top=-1;

S.stacksize=0;

return OK;

}

Status ClearStack(SqStack &S){

S.top=-1;

return OK;

}

int StackLength(SqStack &S){

return (S.top+1);

}

Status GetTop(SqStack &S,ElemType &e){

if(S.top == -1) return ERROR;

e = S.base[S.top];

return OK;

}

Status Push(SqStack &S,ElemType e){

if(S.top>=S.stacksize){

S.base=(ElemType\*)realloc(S.base,(S.stacksize+STACKINCREMENT)\*sizeof(ElemType));

if(!S.base) exit (OVERFLOW);

// S.top = S.base + S.stacksize;

S.stacksize += STACKINCREMENT;

}

S.base[S.top++] =e;

return OK;

}

Status Pop(SqStack &S,ElemType &e){

if(S.top==-1) return ERROR;

e = S.base[S.top--];

return OK;

}

// Status StackTraverse(SqStack &S,Status (\*visit)()){

// }

void Increment(SqStack &S)

{

int i;

ElemType \*newstack;

newstack = new ElemType[S.stacksize+STACKINCREMENT];

if(!newstack) ERROR("OVERFLOW");

for(i=0;i<S.top;i++)

{

newstack[i]=S.base[i];

}

delete []S.base;

S.base=newstack;

S.stacksize+=STACKINCREMENT;

}

int main()

{

return 0;

}

2.

题目：编程实现书P59 ADT Queue基本操作9个，用链式存储结构实现；

算法思想：

照着书上一个一个敲

运行结果:

编译无bug

附源代码：

#include<iostream>

using namespace std;

#define QUEUE\_MAX\_SIZE 100

typedef int ElemType;

typedef struct QNode{

ElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

} LinkQueue;

void InitQueue\_L(LinkQueue &Q);

void DestroyQueue\_L(LinkQueue &Q);

void ClearQueue\_L(LinkQueue &Q);

int QueueLength\_L(LinkQueue Q);

void GetHead\_Sq\_L(LinkQueue Q,ElemType &e);

void EnQueue\_L(LinkQueue &Q,ElemType e);

void DeQueue\_L(LinkQueue &Q,ElemType &e);

void ERROR(char \*s)

{

cout<<s<<endl;

exit(1);

}

void InitQueue\_L(LinkQueue &Q)

{

Q.front=Q.rear=new QNode;

Q.front->next=NULL;

}

void DestroyQueue\_L(LinkQueue &Q)

{

while(Q.front)

{

Q.rear=Q.front->next;

delete Q.front;

Q.front=Q.rear;

}

}

void ClearQueue\_L(LinkQueue &Q)

{

QNode \*p,\*q;

p=Q.front->next;

while(p)

{

q=p;

p=p->next;

delete q;

}

Q.front->next=NULL;

Q.rear=Q.front;

}

int QueueLength\_L(LinkQueue Q)

{

QNode \*p;

int length=0;

p=Q.front;

while(p->next)

{

length++;

p=p->next;

}

return length;

}

void GetHead\_Sq\_L(LinkQueue Q,ElemType &e)

{

if(Q.front->next==NULL) ERROR("Queue Empty");

e=Q.front->next->data;

}

void EnQueue\_L(LinkQueue &Q,ElemType e)

{

QNode \*p;

p=new QNode;

p->data=e;

p->next=NULL;

Q.rear->next=p;

}

void DeQueue\_L(LinkQueue &Q,ElemType &e)

{

QNode \*p;

if(Q.front->next==NULL)

{

ERROR("Queue Empty");

}

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

delete p;

}

int main()

{

return 0;

}

3.

题目：试利用循环队列编写求k阶斐波那契序列中前n+1项的算法。

要求满足：

,其中max为某个约定常数。(注意：本题所用循环队列的容量仅为k,则在算法执行结束时，留在循环队 列中的元素应是所求k阶斐波那契序列中的最后k项)

算法思想：

K阶斐波那契数列

定义：第k和k+1项为1，前k - 1项为0，从k项之后每一项都是前k项的和

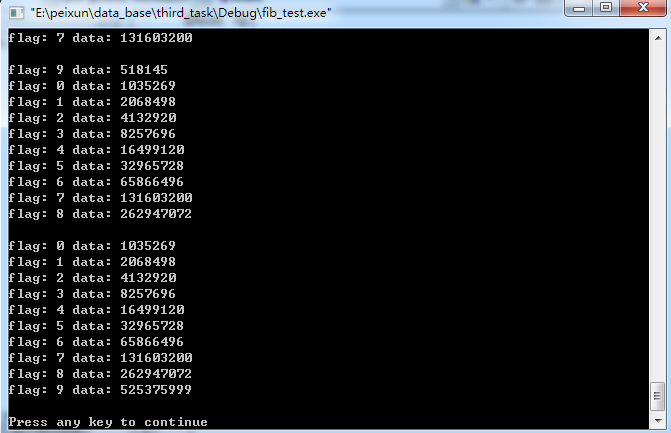
化简一下，得到迭代公式：

①：f(m)=f(m-1)+f(m-2)+…+f(m-k)

②：f(m-1)=f(m-2)+f(m-3)+…+f(m-k-1)

①-②: f(m)-f(m-1)=f(m-1)-f(m-k-1)

f(m)=2f(m-1)-f(m-k-1)

运行结果：

每一次的移动都会显示出来

通过添加flag方便调试

附源程序：

#include<iostream>

using namespace std;

#define QUEUE\_MAX\_SIZE 100

#define MAX\_NUM 400000000

#define k 10

typedef int ElemType;

typedef struct QNode{

ElemType data;

int flag;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

} LinkQueue;

void InitQueue\_L(LinkQueue &Q);

void DestroyQueue\_L(LinkQueue &Q);

void ClearQueue\_L(LinkQueue &Q);

int QueueLength\_L(LinkQueue Q);

void GetHead\_Sq\_L(LinkQueue &Q,ElemType &e);

void EnQueue\_L(LinkQueue &Q,ElemType e);

void DeQueue\_L(LinkQueue &Q,ElemType &e);

void ERROR(char \*s)

{

cout<<s<<endl;

exit(1);

}

void InitQueue\_L(LinkQueue &Q)

{

Q.front=Q.rear=new QNode;

Q.front->next=NULL;

Q.front->flag=0;

Q.front->data=0;

Q.rear->next=Q.front;

}

void DestroyQueue\_L(LinkQueue &Q)

{

while(Q.front)

{

Q.rear=Q.front->next;

delete Q.front;

Q.front=Q.rear;

}

}

void ClearQueue\_L(LinkQueue &Q)

{

QNode \*p,\*q;

p=Q.front->next;

while(p)

{

q=p;

p=p->next;

delete q;

}

Q.front->next=NULL;

Q.rear=Q.front;

}

int QueueLength\_L(LinkQueue Q)

{

QNode \*p;

int length=0;

p=Q.front;

while(p->next)

{

length++;

p=p->next;

}

return length;

}

void GetHead\_Sq\_L(LinkQueue &Q,ElemType &e)

{

if(Q.front->next==NULL) ERROR("Queue Empty");

e=Q.front->data;

Q.front=Q.front->next;

}

void EnQueue\_L(LinkQueue &Q,ElemType e,int i)

{

QNode \*p;

p=new QNode;

p->data=e;

p->flag=i+1;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

}

void DeQueue\_L(LinkQueue &Q,ElemType &e)

{

QNode \*p;

if(Q.front->next==NULL)

{

ERROR("Queue Empty");

}

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

delete p;

}

void update\_Queue(LinkQueue &Q,ElemType e)

{

Q.rear=Q.rear->next;

Q.rear->data=e;

}

void print(LinkQueue &Q)

{

QNode \*p;

p=Q.front;

cout<<"flag: "<<p->flag<<" data: "<<p->data<<endl;

p=p->next;

while(p!=Q.front)

{

cout<<"flag: "<<p->flag<<" data: "<<p->data<<endl;

p=p->next;

}

cout<<endl;

}

int main()

{

LinkQueue Q;

int e;

int i,temp;

InitQueue\_L(Q);

for(i=0;i<k-2;i++)

{

EnQueue\_L(Q,0,i);

cout<<endl<<i;

}

EnQueue\_L(Q,1,k-2);

Q.rear->next=Q.front;

GetHead\_Sq\_L(Q,e);

update\_Queue(Q,1);

temp=1;

while(temp<MAX\_NUM)

{

GetHead\_Sq\_L(Q,e);

update\_Queue(Q,temp\*2-e);

temp=temp\*2-e;

print(Q);

}

return 0;

}

第四次上机实验报告

1.

题目：输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（1、2）；

算法思想：

装置前的矩阵每一列不为零的数的数目==装置后每一行的不为零的书的数目

故新建辅助数组

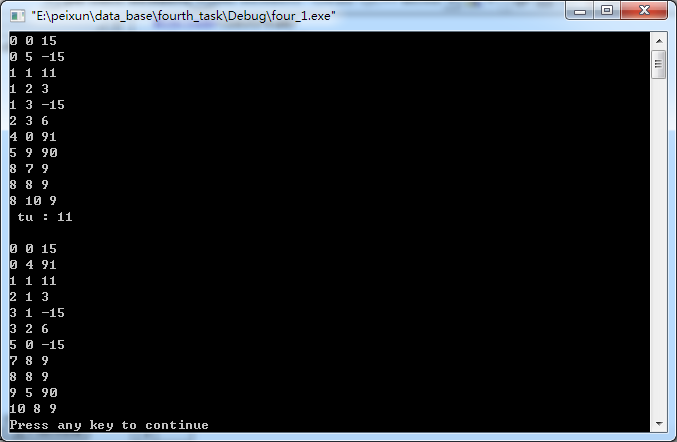
num[n]:存储每一列非零个数 cpot[n]：存储装置后矩阵每一行开始的下标

cpot[0]=1;

cpot[n]=cpot[n-1]+num[n-1]

遍历一遍原矩阵存储 便可实现装置

运行结果：



附源程序：

#include<iostream>

using namespace std;

#include<fstream>

#define TriList\_Size 1000

typedef int Element;

typedef struct {

int row,col;

Element e;

}Triple;

typedef struct {

Triple data[TriList\_Size+1];

int mu,nu,tu;

}TSMatrix;

void InitSMatrix(TSMatrix &M)

{

M.tu=0;

M.nu=0;

M.mu=0;

}

void DestroySMatrix(TSMatrix &M);

void ReadSMatrix(TSMatrix &M)

{

fstream f1;

int i=0;

f1.open("test.txt",ios::in);

f1>>M.mu>>M.nu;

while(!f1.eof())

{

f1>>M.data[i].row>>M.data[i].col>>M.data[i].e;

M.tu++;

i++;

}

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

{

cout<<M.data[i].row<<" "<<M.data[i].col<<" "<<M.data[i].e<<endl;

}

cout<<" tu : "<<M.tu<<endl;

f1.close();

}

void CopySMatrix(TSMatrix &M);

void AddSMatrix(TSMatrix &M);

void MulSMatrix(TSMatrix &M);

void TransposeSMatrix(TSMatrix M,TSMatrix &T)

{

InitSMatrix(T);

int i,j;

int num[100],cpot[100];

T.mu=M.mu;

T.nu=M.nu;

T.tu=M.tu;

if(T.tu){

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

{

num[i]=0;

cpot[i]=0;

// cout<<num[i]<<endl;

}

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

{

++num[M.data[i].col];

}

cpot[0]=0;

for(i=1;i<=M.nu;i++)

{

cpot[i]=cpot[i-1]+num[i-1];

}

/\* for(i=0;i<M.nu;i++)

{

cout<<num[i]<<" "<<cpot[i]<<endl;

}

\*/ for(i=0;i<M.tu;i++)

{

j=M.data[i].col;

T.data[cpot[j]].row=M.data[i].col;

T.data[cpot[j]].col=M.data[i].row;

T.data[cpot[j]].e=M.data[i].e;

cpot[j]++;

}

cout<<endl;

for(i=0;i<T.tu;i++)

{

cout<<T.data[i].row<<" "<<T.data[i].col<<" "<<T.data[i].e<<endl;

}

}

}

int main()

{

TSMatrix M,T;

int \*num,\*cpot;

InitSMatrix(M);

ReadSMatrix(M);

TransposeSMatrix(M,T);

return 0;

}

2.

题目：统计一篇英文文章中各单词出现的频度。

输入： 一篇txt格式英文内容

输出： 单词：频度 例如： the: 10; long: 5

算法思想：

1. 将单词存于链表 构造二叉链表 forest[52] 52个字母开头 便于查找

Forest[0]->aa->aaa->and->... 排序存于链表

2.构造串树 a

count

h

然后构成了单词

s

count

count

m

count

l

l

o

i

e

he hello hi his him

在每一个单词的结尾

存单词频度

顺便把地址也存起来了

count

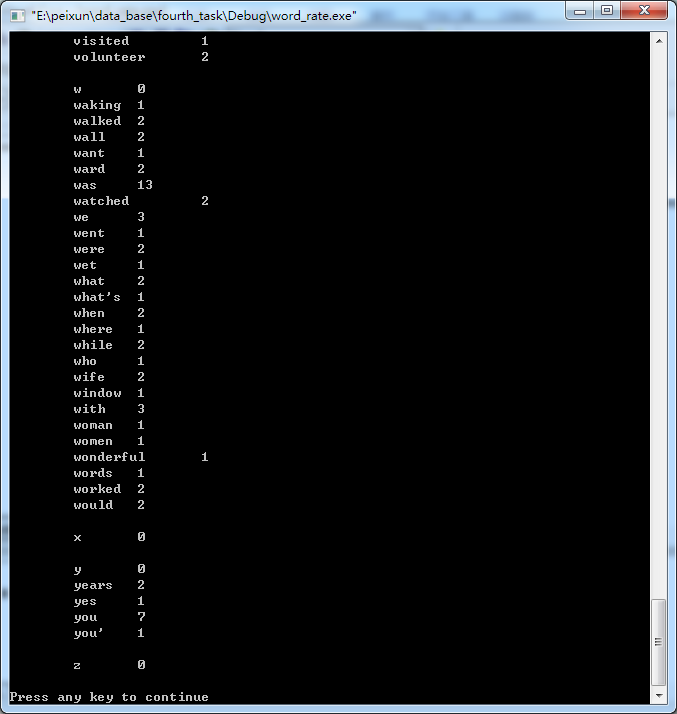
再串树的读取的时候 两种方法

1. 每次讲字母存到string中 每读取

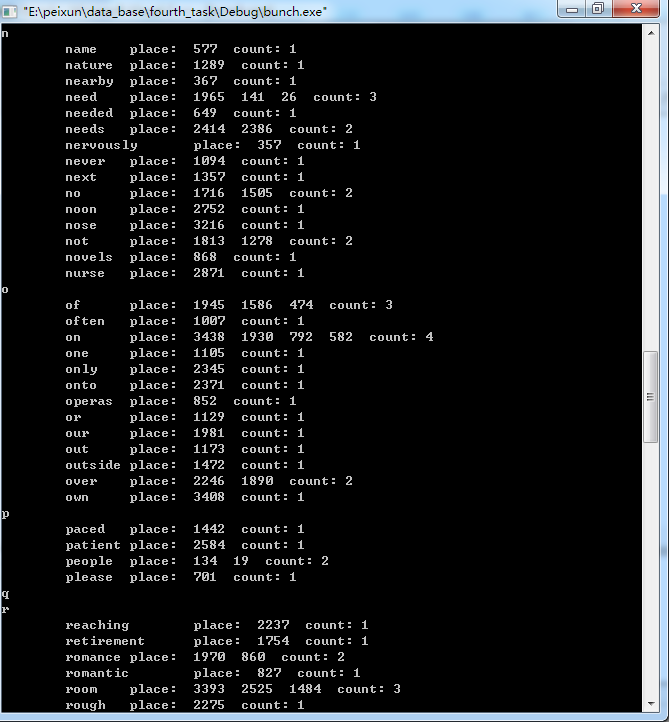
一个节点 string+=alphbet;

2. 通过栈来读取 每次到达叶子结点 通过read\_d\_s(forest) 读取栈元素 从底至上

读取 top元素存储的letter\* 地址 取count 顺便取地址 倒序

1. 运行结果：
2. 链表输出 

会将结果输出到range.txt文件中

1. 串树

附源代码：

1.链表 //完成第一、二、三次上机题

//1．输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（1、2）；

//2. 统计一篇英文文章中各单词出现的频度。

// 输入： 一篇txt格式英文内容

// 输出： 单词：频度 例如： the: 10; long: 5

//3. 利用栈操作实现八皇后问题求解 (选做题) 。

#include<iostream>

#include<string>

using namespace std;

#include<fstream>

struct word\_place{

int place;

word\_place \* w\_next;

};

struct word{

int count;

string words;

word \*next,\*pre;

word\_place \*w\_p\_cur\_next,\*w\_p\_head;

};

void man\_txt()

{

fstream f1,f2;

char chr;

f1.open("eng.txt",ios::in);

f2.open("en2.txt",ios::out);

while(!f1.eof())

{

chr=f1.get() ;

if(chr!=','&&chr!='.'&&chr!='\"'&&chr!='?'&&chr!='!')

{

f2<<chr;

// cout<<chr;

}

}

f1.close();

f2.close();

cout<<"end";

}

void display(word \* p)

{

word \*q;

word\_place \*w\_p;

fstream f1;

f1.open("rand.txt",ios::out);

int i=0;

while(i<52)

{

q=&p[i];

while(q)

{

cout<<"\t"<<q->words<<" \t"<<q->count<<" "<<endl;

f1<<"\t"<<q->words<<" \t"<<q->count<<" "<<endl;

w\_p=q->w\_p\_head;

while(w\_p)

{

f1<<"\t\t"<<w\_p->place<<endl;

w\_p=w\_p->w\_next;

}

q=q->next;

}

cout<<endl;

i++;

}

}

int comp()

{

fstream f1;

f1.open("ee2.txt",ios::in);

int i=0;

word w[52];

word \*cur[52];

word\_place \*w\_p;

word \*p,\*q;

int place;

string tmp,tmp\_first;

for(i=0;i<26;i++)

{

w[i].words=(char)(i+'A');

w[i].next=NULL;

w[i].pre=NULL;

w[i].w\_p\_cur\_next=NULL;

w[i].w\_p\_head=NULL;

w[i].count=0;

w[i+26].words=(char)(i+'a');

w[i+26].next=NULL;

w[i+26].pre=NULL;

w[i+26].w\_p\_cur\_next=NULL;

w[i+26].w\_p\_head=NULL;

w[i+26].count=0;

cout<<w[i].words<<" "<<w[i+26].words<<endl;

}

// cout<<(char)('A'+5)<<endl;

// cout<<(a.substr(0,1).compare("a"))<<endl;

while(!f1.eof())

{

place=f1.tellg();

f1>>tmp;

tmp\_first=tmp.substr(0,1);

cout<<tmp<<endl;

if((tmp\_first.compare("A")>=0)&&(tmp\_first.compare("Z")<=0))

{

i=tmp\_first[0]-'A';

}else if((tmp\_first.compare("a")>=0)&&(tmp\_first.compare("z")<=0)){

i=tmp\_first[0]-'a'+26;

}else{

i=-1;

}

if(i!=-1)

{

p = new word;

p->words=tmp;

p->next=NULL;

cout<<"p->words: "<<p->words<<endl;

if(w[i].next==NULL)

{

p->pre=&w[i];

w[i].next=p;

cur[i]=p;

p->count=1;

cout<<"new: "<<p->words<<endl;

//w\_p

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

cur[i]->w\_p\_head=w\_p;

cur[i]->w\_p\_cur\_next=w\_p;

// cout<<"ppp"<<endl;

}else{

// cout<<cur[i]->words<<" "<<tmp<<" "<<cur[i]->words.compare(tmp)<<" "<<(cur[i]->words.compare(tmp)<0&&cur[i]->next);

if(cur[i]->words.compare(tmp)>0)

{

/\* cout<<"++++++++++++++++++"<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"++++++++"<<endl;

\*/

while(cur[i]->words.compare(tmp)>0&&cur[i]!=&w[i])

{

cur[i]=cur[i]->pre;

// cout<<"---------"<<endl;

}

/\* cout<<"++++++++++++++++++"<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"++++++++"<<endl;

\*/

}else if(cur[i]->words.compare(tmp)<0){

/\* cout<<"................."<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"..........."<<endl;

\*/

while(cur[i]->words.compare(tmp)<0&&cur[i]->next)

{

cur[i]=cur[i]->next;

}

if(cur[i]->words.compare(tmp)>0)

{

cur[i]=cur[i]->pre;

}

/\* cout<<"..........."<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<".........."<<endl;

\*/

}

if(cur[i]->words.compare(tmp)==0)

{

cout<<endl<<endl<<"="<<cur[i]->words<<"===="<<p->words<<endl;

cout<<"pop"<<cur[i]->w\_p\_cur\_next->place<<endl;

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

if(cur[i]->w\_p\_cur\_next)

{

cout<<"pop"<<cur[i]->w\_p\_cur\_next->place<<endl;

}

cur[i]->w\_p\_cur\_next->w\_next=w\_p;

cur[i]->w\_p\_cur\_next=w\_p;

cur[i]->count++;

}else{

p->pre=cur[i];

p->next=cur[i]->next;

if(cur[i]->next)

{

cur[i]->next->pre=p;

}

cur[i]->next=p;//插入word节点

//插入w\_p节点

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

p->w\_p\_head=w\_p;

p->w\_p\_cur\_next=w\_p;

p->count=1;

cur[i]=p;

cout<<cur[i]->words<<" place: "<<cur[i]->w\_p\_cur\_next->place<<endl;

}

}

}

cout<<"old"<<endl;

}

display(w);

f1.close();

return 0;

}

int main()

{

comp();

return 0;

}

1. 串树：

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

typedef struct stack \* link;

#define maxn 100

struct addr

{

int place;

addr \*next;

};

struct letter

{

char alphabet;

int count;

addr \*addr\_count;

letter \*sibling;

letter \*next;

};

struct play\_stack

{

letter \*p;

char al;

};

play\_stack p\_s[maxn];

struct stack

{

string cur\_str;

letter \*p;

link next;

};

struct show

{

int length;

letter \*p;

};

void init\_letter(letter \*p)

{

p->addr\_count=NULL;

p->alphabet='A';

p->count=0;

p->sibling=0;

p->next=NULL;

}

void init(letter \*p)

{

int i=0;

for(i=0;i<26;i++)

{

p[i].alphabet='A'+i;

p[i].sibling=NULL;

p[i].addr\_count=NULL;

p[i].next=NULL;

p[i].count=0;

p[i+26].alphabet='a'+i;

p[i+26].sibling=NULL;

p[i+26].next=NULL;

p[i+26].count=0;

p[i+26].addr\_count=NULL;

}

}

letter\* set\_p(char chr,letter \*p,letter \*forest)

{

if(chr>='A'&&chr<='Z')

{

p=&forest[((int)chr-'A')];

}else if(chr>='a'&&chr<='z')

{

p=&forest[((int)chr-'a'+26)];

}

return p;

}

letter\* insert(char chr,letter \*p)

{

letter \*cur,\*origin,\*q\_l;

q\_l = new letter;

q\_l->addr\_count=NULL;

q\_l->alphabet=chr;

q\_l->sibling=NULL;

q\_l->next=NULL;

q\_l->count=0;

if(p->next==NULL)

{

p->next=q\_l;

p=q\_l;

}else{

origin=p;

p=p->next;

cur=p;

if(p->alphabet>chr)

{

q\_l->sibling=p;

origin->next=q\_l;

p=q\_l;

}else{

while(p!=NULL)

{

if(p->alphabet<chr)

{

cur=p;

p=p->sibling;

}else{

break;

}

}

if(p==NULL)

{

cur->sibling=q\_l;

p=q\_l;

}else if(p->alphabet>chr)

{

q\_l->sibling=p;

cur->sibling=q\_l;

p=q\_l;

}

}

}

return p;

}

void set\_record(char chr,letter \*p,int place)

{

addr \*q;

q = new addr;

q->next=NULL;

q->place=place;

p->count++;

if(p->addr\_count)

q->next=p->addr\_count->next;

p->addr\_count=q;

}

link push(link Head,string a,letter \*l)

{

link p;

p= new stack;

if(p==NULL)

{

cout<<"\nMemory Error\n";

return Head;

}

p->p=l;

p->next=Head;

p->cur\_str=a;

return p;

}

link pop(link Head)

{

link p;

p=Head;

if(p==NULL)

{

cout<<"\nStack is Empty\n";

}

else{

p=p->next;

}

return p;

}

void travel(letter \*forest)

{

int i=0;

string a;

link Head;

letter \*q\_l;

cout<<"oooo"<<endl;

Head=NULL;

for(i=0;i<52;i++)

{

cout<<forest[i].alphabet<<endl;

// a+=forest[i].alphabet;

if(forest[i].next!=NULL)

{

q\_l=&forest[i];

while(q\_l!=NULL)

{

if(q\_l->sibling!=NULL)

{

letter \*qp;

qp=q\_l->sibling;

Head=push(Head,a,q\_l->sibling);

while(qp)

{

qp=qp->sibling;

}

}

a+=q\_l->alphabet;

if(q\_l->count>0)

{

addr \*p\_arr;

p\_arr=q\_l->addr\_count;

cout<<a<<"\t place: \t";

while(p\_arr)

{

// cout<<"true "<<p\_arr->place;

p\_arr=p\_arr->next;

}

cout<<"\tcount: \t"<<q\_l->count<<endl;

}

if(q\_l->next)

{

q\_l=q\_l->next;

}else{

break;

}

}

while(Head)

{

if(Head->p)

{

}

q\_l=Head->p;

a=Head->cur\_str;

Head=Head->next;

while(q\_l!=NULL)

{

if(q\_l->sibling!=NULL)

{

Head=push(Head,a,q\_l->sibling);

// cout<<"Head"<<Head->p->alphabet<<endl;

}

a+=q\_l->alphabet;

if(q\_l->count>0)

{

cout<<a<<" ";

addr \*p\_arr;

p\_arr=q\_l->addr\_count;

while(p\_arr)

{

cout<<" "<<p\_arr->place<<" ";

p\_arr=p\_arr->next;

}

cout<<q\_l->count<<"\t";

}

q\_l=q\_l->next;

}

}

}

cout<<" "<<endl;

a='.';

}

}

void push\_d\_s(letter \*p,int & top)

{

top++;

p\_s[top].al=p->alphabet;

p\_s[top].p=p;

}

void read\_ps(int top)

{

cout<<"\t";

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

{

cout<<p\_s[i].p->alphabet;

}

addr \*p;

p=p\_s[top].p->addr\_count;

if(p)

cout<<"\tplace: ";

while(p)

{

cout<<" "<<p->place<<" ";

p=p->next;

}

cout<<" count: "<<p\_s[top].p->count<<endl;

}

letter\* pop\_d\_s(int &top)

{

int tmp;

tmp=top;

top--;

return p\_s[tmp].p;

}

void display\_stack(letter \*forest)

{

int top=0;

int cur=0;

letter \*q\_l;

for(int i=0;i<52;i++)

{

top=0;

cout<<forest[i].alphabet<<" "<<endl;

push\_d\_s(&forest[i],top);

q\_l=&forest[i];

if(q\_l->count>0)

{

read\_ps(top);

}

if(forest[i].next)

{

q\_l=forest[i].next;

while(top!=0)

{

while(q\_l!=NULL)

{

push\_d\_s(q\_l,top);

if(q\_l->count>0)

{

read\_ps(top);

}

q\_l=q\_l->next;

}

q\_l=pop\_d\_s(top);

while(q\_l->sibling==NULL&&top!=0)

{

q\_l=pop\_d\_s(top);

}

if(q\_l->sibling!=NULL)

{

q\_l=q\_l->sibling;

}

}

}

}

}

void display(letter \*forest)

{

int i=0;

string a;

letter \*q\_l;

// cout<<"oooo"<<endl;

for(i=0;i<52;i++)

{

cout<<forest[i].alphabet<<" ";

a+=forest[i].alphabet;

if(forest[i].next!=NULL)

{

q\_l=forest[i].next;

while(q\_l->next!=NULL)

{

a+=q\_l->alphabet;

if(q\_l->count>0)

{

cout<<a<<" "<<q\_l->count<<"\t";

}

q\_l=q\_l->next;

}

a+=q\_l->alphabet;

cout<<a<<"\t";

cout<<q\_l->count;

}

cout<<" "<<endl;

a='.';

}

}

void man\_txt()

{

fstream f1;

letter \*p;

addr \*q;

letter \*q\_l;

letter forest[52];

letter \*origin,\*cur;

int place=0;

char chr;

init(forest);

f1.open("1.txt",ios::in);

place=f1.tellg();

chr=f1.get();

while(!f1.eof())

{

while(!((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z'))&&!f1.eof())

{

place=f1.tellg();

chr=f1.get();

}

if(!f1.eof()&&((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z')))

p=set\_p(chr,p,forest);

while(!f1.eof()&&((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z')))

{

chr=f1.get();

if(f1.eof()) break;

if((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z'))

{

p=insert(chr,p);

}

}

q = new addr;

q->next=NULL;

q->place=place;

p->count++;

addr \*m;

m=p->addr\_count;

if(p->addr\_count!=NULL)

m=m->next;

if(p->addr\_count)

{

q->next=p->addr\_count;

p->addr\_count=q;

// cout<<"1: "<<1<<endl;

}

else

{

p->addr\_count=q;

// cout<<"0: "<<0<<endl;

}

// cout<<place<<endl;

place=f1.tellg();

chr=f1.get();

}

f1.close();

travel(forest);//string输出

display\_stack(forest);//栈输出

cout<<"end";

}

int main()

{

man\_txt();

return 0;

}

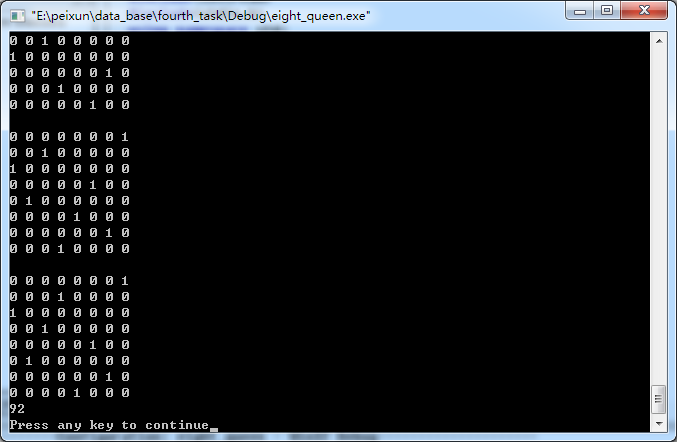
3.

题目：八皇后

算法思想：

栈的基本操作

运行结果：



附源代码：

#include<iostream>

#include<stdio.h>

using namespace std;

#define N 8

struct stack{

int row,col;

};

void pop(stack \*p,int &top,int &row,int &col)

{

if(top>=0&&top<=N)

{

row=p[top].row;

col=p[top].col;

p[top].col=-1;

top--;

}else{

cout<<"wrong top pop"<<endl;

exit(1);

}

}

void push(stack \*p,int &top,int row,int col)

{

if(top<=N&&top>=0)

{

top++;

p[top].row=row;

p[top].col=col;

}else{

cout<<"wrong top"<<endl;

exit(1);

}

}

int back\_search(stack \*p,int top,int row,int col)

{

int i;

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

{

if(p[i].col==col||(row-p[i].row)==(col-p[i].col)||(p[i].row-row)==(col-p[i].col))

{

return 0;

}

}

return 1;

}

void display(stack \*p)

{

int i=0,j;

cout<<endl;

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

{

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

{

if(j!=p[i].col)

{

cout<<0<<" ";

}else{

cout<<1<<" ";

}

}

cout<<endl;

}

}

void queen()

{

stack s[N+1];

int i=0,j=0,top=0;

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

{

s[i].row=0;

s[i].col=-1;

}

i=0;

j=0;

push(s,top,i,j);

i++;

int count=0;

while(!(i==0&&j==N))

{

if(j<N){

if(back\_search(s,top,i,j))

{

push(s,top,i,j);

i++;

j=0;

if(i==N)

{ count++;

display(s);

}

}else{

j++;

}

}else{

if(j>=N){

pop(s,top,i,j);

j++;

}

}

}

cout<<count<<endl;

}

int main()

{

queen();

return 0;

}

第五次上机实验报告

1. 题目：编程实现书P121 ADT BinaryTree 基本操作20个，用二叉链表结构实现；

算法思想：照着书上一个一个敲

运行结果：编译无错误

附源程序：

#include<iostream>

#include<stack>

#include<queue>

using namespace std;

//Sequence

#define BiTree\_Size 100

typedef int ElemType;

typedef ElemType SqBiTree[BiTree\_Size + 1];

//struct

typedef struct BTNode{

ElemType data;

BTNode \*lchild,\*rchild;

} BiTNode;

typedef BiTNode \*BiTree;

class BinaryTree{

public:

void InitBiTree(SqBiTree &T);

void DestroyBiTree(SqBiTree &T);

void CreateBiTree(SqBiTree &T);

void ClearBiTree(SqBiTree &T);

void BiTreeDepth(SqBiTree &T);

BiTree Root(BiTNode &T);

ElemType Value(BiTNode T);

BiTree Parent(BiTree T);

BiTree LeftChild(BiTree T,BiTree p);

BiTree RightChild(BiTree T,BiTree p);

void InsertChild(SqBiTree &T,SqBiTree p,SqBiTree LR,ElemType c);

void DeleteChild(SqBiTree &T,SqBiTree LR);

void PreOrderTraverse(SqBiTree T);

void InOrderTraverse(BiTree T);

void PostOrderTraverse(SqBiTree T);

};

BiTree LeftChild(BiTree T)

{

if(T->lchild)

return T->lchild;

else return NULL;

}

BiTree RightChild(BiTree T)

{

if(T->rchild)

return T->rchild;

else return NULL;

}

BiTree Parent(BiTree T,BiTree e)

{

queue<BiTree> q;

BiTree p;

if (!T) {

return NULL;

}else{

q.push(T);

}

while(!q.empty())

{

p=q.front();

if(p->lchild)

{

if(p->lchild==e)

return p;

q.push(p->lchild);

}

if(p->rchild)

{

if(p->rchild==e)

return p;

q.push(p->rchild);

}

q.pop();

}

return NULL;

}

ElemType Value(BiTree T)

{

return T->data;

}

BiTree Root(BiTree T)

{

if(T)

{

return T;

}

}

void PreOrderTraverse(SqBiTree BT)

{

int n = BT[0];

for(int i=1,int j;i<n;i++)

{

if(i==1) j=1;

else if(2\*j<=n) j=2\*j;

else if((j%2==0)&&(j<n)) j=j+1;

else if(j>1){

while((j/2)%2!=0) j=j/2;

j=j/2+1;

}

cout<<BT[j];

}

}

void InOrderTraverse(BiTree BT)

{

queue<BiTree> S;

BiTree p,q;

p=BT;

while(p)

{

if(p){

S.push(p);

p=p->lchild;

}

else{

q=S.front();

S.pop();

cout<<p->data;

p=p->lchild;

}

}

}

void PostOrderTraverse(BiTNode\* root)

{

if(root)

{

PostOrderTraverse(root->lchild);

PostOrderTraverse(root->rchild);

cout<<root->data;

}

}

void CreateBiTree(BiTNode \*BT)

{

int ch;

cin>>ch;

if(ch==0) BT=NULL;

else{

BT=new BiTNode;

BT->data=ch;

CreateBiTree(BT->lchild);

CreateBiTree(BT->rchild);

}

}

int main()

{

return 0;

}

2.

题目：二叉树的先序、中序、后序遍历递归（非递归选做）方法，层次遍历。

算法：先构造二叉树 小的放左大的放右 预期的输出结果为 中序时为已排序的数据

初始化栈 #include<stack> stack<BiTree> q;

先序：通过while遍历左子树 cout<<p->data; 入栈q.push() 当为NULL时 p=q.top() ;p=p->right; q.pop();

中序：通过while遍历左子树 入栈q.push() 当为NULL时 p=q.top() ; cout<<p->data;p=-p>right; q.pop();

后序：初始化 last 指向上次遍历过的节点 遍历一个节点push一个节点

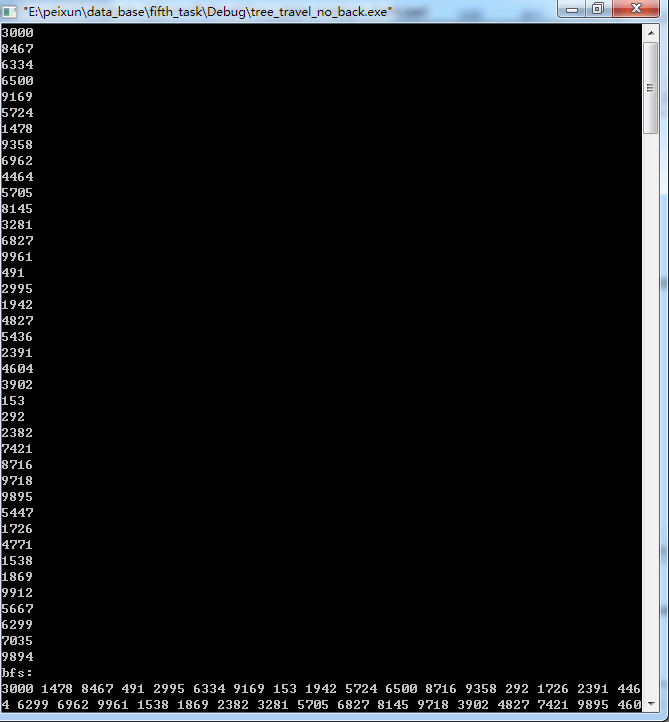
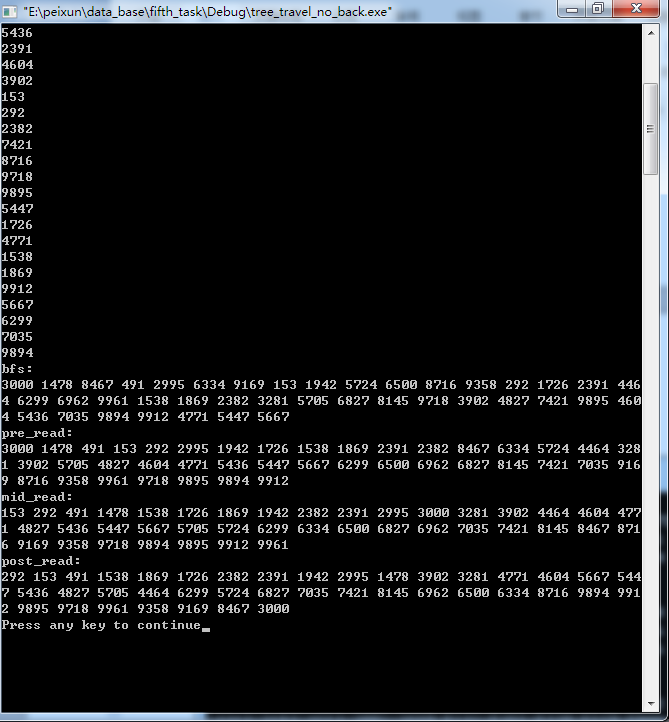
if (p->right==NULL||p->right==last) cout<<p->data;else p=q.top();p=p->right; 继续循 环

层次 ：初始化一个队列 queue<BiTree> q;

遍历sibling 进行入队出队操作

运行结果：

40个数据为：

 遍历结果为：

附源代码：

#include<iostream>

#include<stack>

#include<queue>

using namespace std;

#define N 50

struct binary\_tree{

int data;

binary\_tree \* left;

binary\_tree \* right;

binary\_tree(int a = 0):data(a),left(NULL),right(NULL) {}

};

void insert(binary\_tree \*cur,binary\_tree \* p)

{

if(cur->data>p->data)

{

if(cur->left==NULL)

{

cur->left=p;

}

else

{

insert(cur->left,p);

}

}else

{

if(cur->right==NULL)

{

cur->right=p;

}

else

{

insert(cur->right,p);

}

}

}

void bfs(binary\_tree \*root)

{

queue<binary\_tree \*> q;

q.push(root);

while(!q.empty())

{

binary\_tree \* p;

p=q.front();

q.pop();

cout<<p->data<<" ";

if(p->left!=NULL) q.push(p->left);

if(p->right!=NULL) q.push(p->right);

}

cout<<endl;

}

binary\_tree\* read\_num(binary\_tree \* root)

{

int tmp;

binary\_tree \*p,\*q;

root=NULL;

p=root;

int n=N;

while(n--)

{

tmp=rand()%10000;

if(n==N-1)

tmp=3000;

cout<<tmp<<endl;

q = new binary\_tree;

q->data=tmp;

if(p!=NULL){

insert(root,q);

}

else{

root=q;

p=root;

}

}

return root;

}

void mid\_read\_while(binary\_tree \*root)

{

binary\_tree \*p;

p=root;

stack<binary\_tree\*>q;

while(!q.empty()||p)

{

while(p)

{

q.push(p);

p=p->left;

}

p=q.top();

q.pop();

cout<<p->data<<" ";

p=p->right;

}

cout<<endl;

}

void pre\_read\_while(binary\_tree \*root)

{

binary\_tree \*p;

p=root;

stack<binary\_tree\*>q;

while(!q.empty()||p)

{

while(p)

{

cout<<p->data<<" ";

q.push(p);

p=p->left;

}

p=q.top();

q.pop();

p=p->right;

}

cout<<endl;

}

void post\_read\_while(binary\_tree \*root)

{

binary\_tree \*p,\*last;

p=root;

stack<binary\_tree\*>q;

last=NULL;

while(p)

{

q.push(p);

p=p->left;

}

while(!q.empty())

{

p=q.top();

q.pop();

if(p->right==NULL||p->right==last)

{

cout<<p->data<<" ";

last=p;

}

else

{

q.push(p);

p=p->right;

while(p)

{

q.push(p);

p=p->left;

}

}

}

cout<<endl;

}

int main()

{

binary\_tree \*root;

root=read\_num(root);

cout<<"bfs: "<<endl;

bfs(root);

cout<<"pre\_read: "<<endl;

pre\_read\_while(root);

cout<<"mid\_read: "<<endl;

mid\_read\_while(root);

cout<<"post\_read: "<<endl;

post\_read\_while(root);

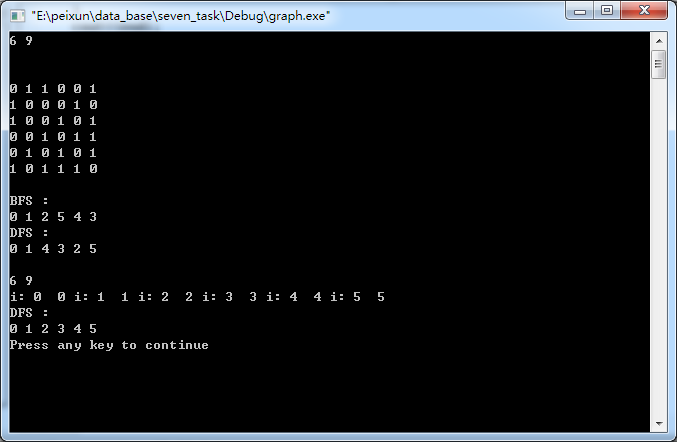
return 0;

}

第七次上机实验报告

1.题目：图的深度优先和广度优先遍历；

算法思想：深度优先 stack 广度优先 queue

运行结果：

附源程序：

#include<iostream>

#include<stack>

#include<queue>

#include<algorithm>

#include<fstream>

using namespace std;

#define Infinity MAX

#define Vertex\_Max 20

typedef int VRType;

typedef int ElemType;

typedef enum{DG,UDG,UDN}GraphKind;

typedef struct {

ElemType vexs[Vertex\_Max];

VRType edges[Vertex\_Max][Vertex\_Max];

int vexnum,edgenum;

GraphKind kind;

}MGraph;

typedef struct EdgeNode\_L

{

int adjvex;

EdgeNode\_L \* nextedge;

int weight;

}EdgeNode\_L;

typedef struct VexNode\_L

{

ElemType vex;

EdgeNode\_L \*firstedge;

VexNode\_L(){firstedge=NULL;}

}VexNode\_L;

typedef struct

{

VexNode\_L vexs[Vertex\_Max];

int vexnum,edgenum;

int kind;

}ALGraph;

int LocateVex\_L(ALGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k].vex!=x);k++);

if(k<G.vexnum) return k;

else return -1;

}

//if input is file

/\*

6 9

0 1 2 3 4 5

1 4

0 1

0 2

0 5

2 5

2 3

3 5

3 4

4 5

\*/

//save as graph.txt

void CreateDG\_ALG(ALGraph &G,int n,int e)

{

fstream f1;

f1.open("graph.txt",ios::in);

int i,j;

ElemType v1,v2;

EdgeNode\_L \*p;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

// cin>>G.vexnum;

// cin>>G.edgenum;

cout<<G.vexnum<<" "<<G.edgenum<<endl;

G.edgenum=e;

for(i=0;i<G.vexnum;i++)

{

f1>>G.vexs[i].vex;

// cin>>G.vexs[i].vex;

cout<<"i: "<<i<<"　"<<G.vexs[i].vex<<" ";

}

cout<<endl;

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2;

// cin>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

// cout<<"iy: "<<i<<"j: "<<j<<endl;

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

f1>>v1>>v2;

// cin>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

}

p = new EdgeNode\_L;

p->adjvex=j;

p->nextedge=G.vexs[i].firstedge;

G.vexs[i].firstedge=p;

}

f1.close();

}

int LocateVex(MGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k]!=x);k++);

// cout<<"k: "<<k<<" "<<x<<" "<<G.vexnum<<endl;

if(k<G.vexnum) return k;

else return -1;

}

void CreateUDG\_MG(MGraph &G,int n,int e)

{

fstream f1;

f1.open("graph.txt",ios::in);

int i,j;

ElemType v1,v2;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

// cin>>G.vexnum;

// cin>>G.edgenum;

cout<<G.vexnum<<" "<<G.edgenum<<endl;

// G.edgenum=e;

for(i=0;i<G.vexnum;i++)

{

// cin>>G.vexs[i];

f1>>G.vexs[i];

}

cout<<endl;

for(i=0;i<G.vexnum;i++)

for(j=0;j<G.vexnum;j++)

G.edges[i][j]=0;

cout<<endl;

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2;

// cin>>v1>>v2;

i=LocateVex(G,v1);

j=LocateVex(G,v2);

// cout<<"i: "<<i<<"j: "<<j<<endl;

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

f1>>v1>>v2;

i=LocateVex(G,v1);

j=LocateVex(G,v2);

}

G.edges[i][j]=G.edges[j][i]=1;

}

f1.close();

}

void display\_G(MGraph G)

{

for(int i=0;i<G.vexnum;i++)

{

for(int j=0;j<G.vexnum;j++)

cout<<G.edges[i][j]<<" ";

cout<<endl;

}

}

int visited[Vertex\_Max];

void BFSTraverse(MGraph G)

{

int v;

for(v=0;v<G.vexnum;v++) visited[v]=0;

queue<int> q;

for(v=0;v<G.vexnum;v++)

{

if(!visited[v])

{

cout<<G.vexs[v]<<" ";

visited[v]=1;

q.push(v);

while(!q.empty())

{

int u;

u=q.front();

for(int w=0;w<G.vexnum;w++)

{

if(G.edges[u][w]!=0&&(!visited[w]))

{

visited[w]=1;

cout<<G.vexs[w]<<" ";

q.push(w);

}

}

q.pop();

}

}

}

}

void DFS\_ALG(ALGraph G,int v)

{

EdgeNode\_L \*p;

cout<<G.vexs[v].vex<<" ";

visited[v]=1;

int j;

for(p=G.vexs[v].firstedge;p;p=p->nextedge)

{

j=p->adjvex;

if(!visited[j]) DFS\_ALG(G,j);

}

}

void DFSTraverse\_L(ALGraph G)

{

int visited[Vertex\_Max];

int v;

for(v=0;v<G.vexnum;v++)

{

visited[v]=0;

}

for(v=0;v<G.vexnum;v++)

{

if(!visited[v])

DFS\_ALG(G,v);

}

}

void DFSTraverse(MGraph G)

{

int visited[Vertex\_Max];

int v;

for(v=0;v<G.vexnum;v++) visited[v]=0;

stack<int> q;

for(v=0;v<G.vexnum;v++)

{

if(!visited[v])

{

cout<<G.vexs[v]<<" ";

visited[v]=1;

q.push(v);

int u;

u=v;

while(!q.empty())

{

//cout<<"get front:"<<u<<endl;

for(int w=0;w<G.vexnum;w++)

{

if(G.edges[u][w]!=0&&(!visited[w]))

{

visited[w]=1;

cout<<G.vexs[w]<<" ";

q.push(w);

u=w;

break;

}

}

if(w==G.vexnum)

{

u=q.top();

q.pop();

}

}

}

}

}

int main()

{

MGraph G;

ALGraph A\_G;

CreateUDG\_MG(G,10,10);

display\_G(G);

cout<<endl;

cout<<"BFS :"<<endl;

BFSTraverse(G);

cout<<endl;

cout<<"DFS :"<<endl;

DFSTraverse(G);

cout<<endl<<endl;

CreateDG\_ALG(A\_G,1,1);

cout<<"DFS :"<<endl;

DFSTraverse\_L(A\_G);

cout<<endl;

return 0;

}

附txt文件

//graph.txt

6 9

0 1 2 3 4 5

1 4

0 1

0 2

0 5

2 5

2 3

3 5

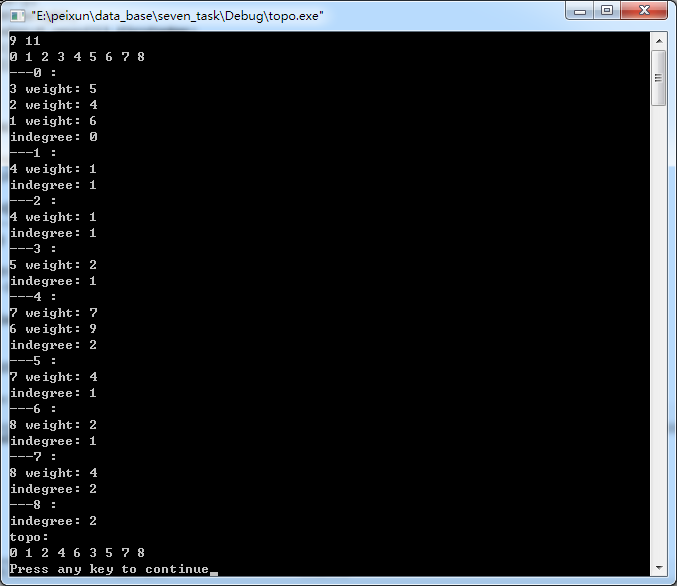
3 4

4 5

//

2.题目：拓扑排序；

算法思想：入度为0先入栈 依次出栈 删去与之相关的边 若入度为0 入栈

运行结果：

附源程序：

/\*

9 11

0 1 2 3 4 5 6 7 8

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

7 8 4

6 8 2

\*/

//graph\_w.txt

#include<iostream>

#include<stack>

#include<math.h>

#include<queue>

#include<algorithm>

#include<fstream>

using namespace std;

#define Infinity MAX

#define Vertex\_Max 20

typedef int VRType;

typedef int ElemType;

typedef enum{DG,UDG,UDN}GraphKind;

typedef struct EdgeNode\_L

{

int adjvex;

EdgeNode\_L \* nextedge;

int weight;

}EdgeNode\_L;

typedef struct VexNode\_L

{

ElemType vex;

int indegree;

EdgeNode\_L \*firstedge;

VexNode\_L(){firstedge=NULL;indegree=0;}

}VexNode\_L;

typedef struct

{

VexNode\_L vexs[Vertex\_Max];

int vexnum,edgenum;

int kind;

} ALGraph;

int LocateVex\_L(ALGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k].vex!=x);k++);

if(k<G.vexnum) return k;

else return -1;

}

void CreateDG\_ALG(ALGraph &G,int n,int e)

{

fstream f1;

f1.open("graph\_w.txt",ios::in);

int i,j;

ElemType v1,v2;

int weight;

EdgeNode\_L \*p;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

cout<<G.vexnum<<" "<<G.edgenum<<endl;

// G.edgenum=e;

for(i=0;i<G.vexnum;i++)

{

f1>>G.vexs[i].vex;

cout<<G.vexs[i].vex<<" ";

}

cout<<endl;

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2>>weight;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

// cout<<"i: "<<i<<"j: "<<j<<endl;

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

system("pause");

f1>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

}

p = new EdgeNode\_L;

p->adjvex=j;

p->weight=weight;

p->nextedge=G.vexs[i].firstedge;

G.vexs[i].firstedge=p;

G.vexs[j].indegree++;

}

f1.close();

}

void display\_G(ALGraph G)

{

EdgeNode\_L \*p;

for(int i=0;i<G.vexnum;i++)

{

p=G.vexs[i].firstedge;

cout<<"---"<<G.vexs[i].vex<<" : "<<endl;

while(p)

{

cout<<G.vexs[p->adjvex].vex<<" weight: "<<p->weight<<endl;

p=p->nextedge;

}

cout<<"indegree: "<<G.vexs[i].indegree<<endl;

}

}

void topologicalSort()

{

stack<int> s;

ALGraph G;

int i=0;

int m;

EdgeNode\_L \*p;

CreateDG\_ALG(G,2,2);

display\_G(G);

for(i=0;i<G.vexnum;i++)

{

if(!G.vexs[i].indegree)

s.push(i);

}

cout<<"topo: "<<endl;

while(!s.empty())

{

m=s.top();

s.pop();

cout<<G.vexs[m].vex<<" ";

p=G.vexs[m].firstedge;

for(;p;p=p->nextedge)

{

G.vexs[p->adjvex].indegree--;

if(G.vexs[p->adjvex].indegree==0)

{

s.push(p->adjvex);

}

}

}

}

int main()

{

topologicalSort();

cout<<endl;

return 0;

}

//附txt文件

//graph\_w.txt

/\*

9 11

0 1 2 3 4 5 6 7 8

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

7 8 4

6 8 2

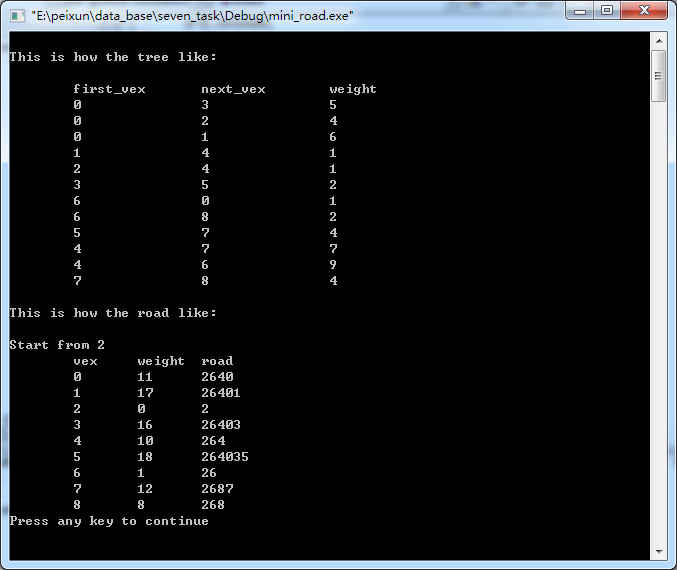
\*/

//graph\_w.txt

3.

题目：最小路径练习(选做题)；

算法思想：标记法

运行结果：

附源程序：

#include<iostream>

#include<stack>

#include<math.h>

#include<queue>

#include<string.h>

#include<string>

#include<algorithm>

#include<fstream>

using namespace std;

#define MAX 99999

#define Infinity MAX

#define Vertex\_Max 20

int visit[100];

typedef int VRType;

int mini[100];

string road[100];

typedef int ElemType;

typedef enum{DG,UDG,UDN}GraphKind;

typedef struct EdgeNode\_L

{

int adjvex;

EdgeNode\_L \* nextedge;

int weight;

}EdgeNode\_L;

typedef struct VexNode\_L

{

ElemType vex;

int indegree;

EdgeNode\_L \*firstedge;

VexNode\_L(){firstedge=NULL;indegree=0;}

}VexNode\_L;

typedef struct

{

VexNode\_L vexs[Vertex\_Max];

int vexnum,edgenum;

int kind;

} ALGraph;

int LocateVex\_L(ALGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k].vex!=x);k++);

if(k<G.vexnum) return k;

else return -1;

}

void display\_tree(ALGraph G)

{

int n=G.vexnum;

int i=0;

cout<<"\tfirst\_vex\t"<<"next\_vex\t"<<"weight"<<endl;

while(n--)

{

EdgeNode\_L \*p;

p=G.vexs[i].firstedge;

while(p)

{

cout<<"\t"<<G.vexs[i].vex<<"\t\t"<<G.vexs[p->adjvex].vex<<"\t\t"<<p->weight<<endl;

p=p->nextedge;

}

i++;

}

}

void CreateDG\_ALG(ALGraph &G,int n,int e)

{

fstream f1;

f1.open("graph\_w.txt",ios::in);

int i,j;

ElemType v1,v2;

int weight;

EdgeNode\_L \*p;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

for(i=0;i<G.vexnum;i++)

{

f1>>G.vexs[i].vex;

}

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2>>weight;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

system("pause");

f1>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

}

p = new EdgeNode\_L;

p->adjvex=j;

p->weight=weight;

p->nextedge=G.vexs[i].firstedge;

G.vexs[i].firstedge=p;

G.vexs[j].indegree++;

}

f1.close();

}

int get\_mini(ALGraph G,int n)

{

int i=0;

int min=MAX;

int re=-1;

while(i<G.vexnum)

{

if((mini[i]!=-1)&&visit[i]==0)

if(mini[i]<min)

{

min=mini[i];

re=i;

}

i++;

}

return re;

}

int main()

{

ALGraph G;

int n;

int e;

int min;

int cur;

int v0=2;

CreateDG\_ALG(G,1,1);

cout<<endl<<"This is how the tree like:"<<endl<<endl;

display\_tree(G);

n=G.vexnum;

int i=0;

cur=v0;

string a;

memset(visit,0,sizeof(visit));

memset(mini,-1,sizeof(mini));

visit[v0]=1;

mini[cur]=0;

while(i<n)

{

road[i]=G.vexs[i].vex+'0';

i++;

}

while(n)

{

EdgeNode\_L \*p;

p=G.vexs[cur].firstedge;

int t;

while(p)

{

t=p->adjvex;

if(mini[t]==-1)

{

a='0'+t;

mini[t]=mini[cur]+p->weight;

road[t]=road[cur]+a;

}

else if(mini[t]>(mini[cur]+p->weight))

{

mini[t]=mini[cur]+p->weight;

a='0'+road[cur];

cout<<road[cur]+a<<endl;

road[t]=road[cur]+a;

}

p=p->nextedge;

}

cur=get\_mini(G,v0);

if(cur==-1) break;

visit[cur]=1;

n--;

}

n=G.vexnum;

i=0;

cout<<endl<<"This is how the road like:"<<endl<<endl;

cout<<"Start from "<<v0<<endl;

cout<<"\tvex\tweight\troad"<<endl;

while(i<n)

{

cout<<"\t"<<i<<"\t"<<mini[i]<<"\t"<<road[i++]<<endl;

}

return 0;

}

//附txt文件

/\*

9 11

0 1 2 3 4 5 6 7 8

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

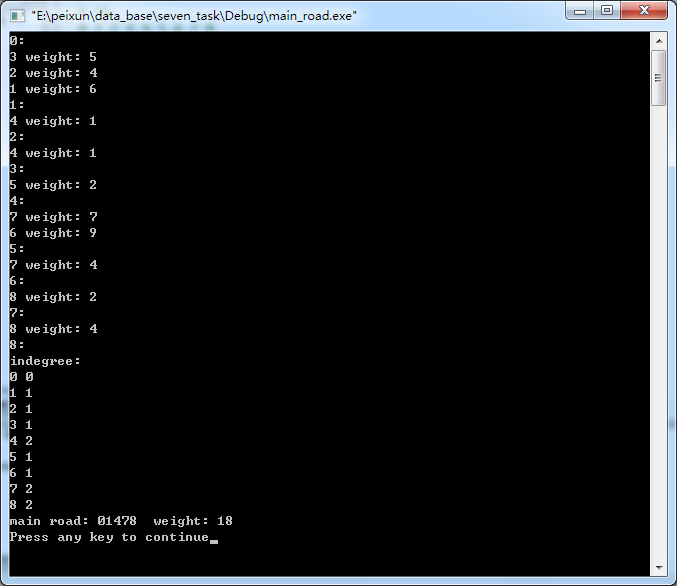
7 8 4

6 8 2

\*/

//graph\_w.txt

1. 题目：关键路径练习(选做题)；
2. 算法思想：动态规划算法

运行结果：

附源程序：

#include<iostream>

#include<stack>

#include<math.h>

#include<queue>

#include<string.h>

#include<string>

#include<algorithm>

#include<fstream>

using namespace std;

#define Infinity MAX

#define Vertex\_Max 20

typedef int VRType;

typedef int ElemType;

typedef enum{DG,UDG,UDN}GraphKind;

typedef struct EdgeNode\_L

{

int adjvex;

EdgeNode\_L \* nextedge;

int weight;

}EdgeNode\_L;

typedef struct VexNode\_L

{

ElemType vex;

int indegree;

EdgeNode\_L \*firstedge;

VexNode\_L(){firstedge=NULL;indegree=0;}

}VexNode\_L;

typedef struct

{

VexNode\_L vexs[Vertex\_Max];

int vexnum,edgenum;

int kind;

} ALGraph;

int visit[100];

int LocateVex\_L(ALGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k].vex!=x);k++);

if(k<G.vexnum) return k;

else return -1;

}

void display\_tree(ALGraph G)

{

int n=G.vexnum;

int i=0;

while(n--)

{

EdgeNode\_L \*p;

p=G.vexs[i].firstedge;

cout<<G.vexs[i].vex<<": "<<endl;

while(p)

{

cout<<G.vexs[p->adjvex].vex<<" weight: "<<p->weight<<endl;

p=p->nextedge;

}

i++;

}

}

void CreateDG\_ALG(ALGraph &G,int n,int e)

{

fstream f1;

f1.open("main\_road.txt",ios::in);

int i,j;

ElemType v1,v2;

int weight;

EdgeNode\_L \*p;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

for(i=0;i<G.vexnum;i++)

{

f1>>G.vexs[i].vex;

}

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2>>weight;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

// cout<<"i: "<<i<<"j: "<<j<<endl;

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

system("pause");

f1>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

}

p = new EdgeNode\_L;

p->adjvex=j;

p->weight=weight;

p->nextedge=G.vexs[i].firstedge;

G.vexs[i].firstedge=p;

G.vexs[j].indegree++;

}

f1.close();

}

int get\_mini(ALGraph G,int n)

{

int mini=9999;

EdgeNode\_L \*p;

p=G.vexs[n].firstedge;

int t;

int re=-1;

while(p)

{

t=p->adjvex;

if(visit[t]==0)

{

if(p->weight<mini)

{

mini=p->weight;

re=p->adjvex;

}

}

p=p->nextedge;

}

return re;

}

int main()

{

ALGraph G;

int n;

int e;

int min;

int cur;

int v0=2;

int final;

CreateDG\_ALG(G,1,1);

display\_tree(G);

n=G.vexnum;

int i=0;

cout<<"indegree: "<<endl;

while(n--)

{

cout<<i<<" "<<G.vexs[i].indegree<<endl;

i++;

}

int state[100];

memset(state,0,sizeof(state));

string road[100];

queue<int> q;

n=G.vexnum;

i=0;

while(n--)

{

if(G.vexs[i].indegree==0)

{

q.push(i);

}

road[i]=i+'0';

i++;

}

while(!q.empty())

{ cur=q.front();

q.pop();

EdgeNode\_L \*p;

p=G.vexs[cur].firstedge;

int t;

int re=-1;

string a;

while(p)

{

if((state[cur]+p->weight)>state[p->adjvex])

{

a='0'+p->adjvex;

state[p->adjvex]=state[cur]+p->weight;

final=p->adjvex;

road[p->adjvex]=road[cur]+a;

}

G.vexs[p->adjvex].indegree--;

if(G.vexs[p->adjvex].indegree==0)

{

q.push(p->adjvex);

}

p=p->nextedge;

}

}

n=G.vexnum;

i=0;

cout<<"main road: "<<road[n-1]<<"　weight: "<<state[final]<<endl;

return 0;

}

//附txt文件

/\*

9 11

0 1 2 3 4 5 6 7 8

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

7 8 4

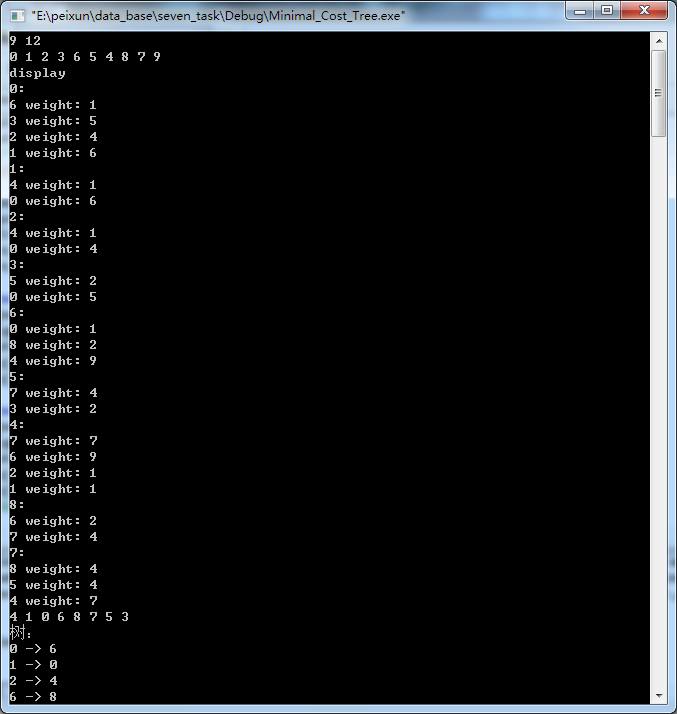
6 8 2

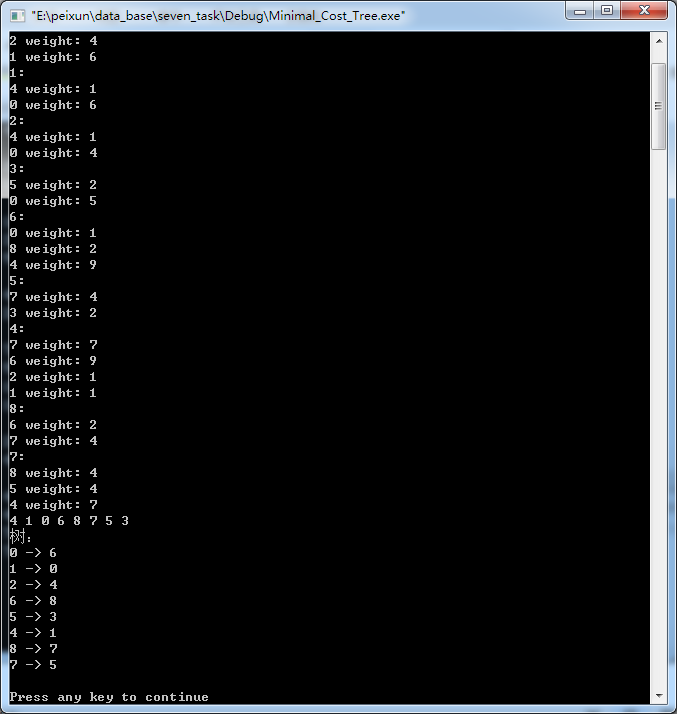
\*/

//graph\_w.txt

5.题目：最小生成树

算法思想：每次查找最小的

运行结果：



附源程序：

/\*

9 11

0 1 2 3 4 5 6 7 8

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

7 8 4

6 8 2

\*/

//graph\_w.txt

#include<iostream>

#include<stack>

#include<map>

#include<iterator>

#include<math.h>

#include<queue>

#include<string>

#include<algorithm>

#include<fstream>

using namespace std;

#define Infinity MAX

#define Vertex\_Max 20

typedef int VRType;

typedef int ElemType;

typedef enum{DG,UDG,UDN}GraphKind;

typedef struct EdgeNode\_L

{

int adjvex;

EdgeNode\_L \* nextedge;

int weight;

}EdgeNode\_L;

typedef struct VexNode\_L

{

ElemType vex;

int indegree;

EdgeNode\_L \*firstedge;

VexNode\_L(){firstedge=NULL;indegree=0;}

}VexNode\_L;

typedef struct

{

VexNode\_L vexs[Vertex\_Max];

int vexnum,edgenum;

int kind;

} ALGraph;

int visit[100];

int LocateVex\_L(ALGraph G,ElemType x)

{

int k;

for(k=0;(k<G.vexnum)&&(G.vexs[k].vex!=x);k++);

if(k<G.vexnum) return k;

else return -1;

}

void display\_tree(ALGraph G)

{

int n=G.vexnum;

int i=0;

cout<<n<<endl;

cout<<"display"<<endl;

while(n--)

{

EdgeNode\_L \*p;

p=G.vexs[i].firstedge;

cout<<G.vexs[i].vex<<": "<<endl;

while(p)

{

cout<<G.vexs[p->adjvex].vex<<" weight: "<<p->weight<<endl;

p=p->nextedge;

}

i++;

}

}

void CreateDG\_ALG(ALGraph &G,int n,int e)

{

fstream f1;

f1.open("graph\_w.txt",ios::in);

int i,j;

ElemType v1,v2;

int weight;

EdgeNode\_L \*p;

int k;

f1>>G.vexnum;

f1>>G.edgenum;

cout<<G.vexnum<<" "<<G.edgenum<<endl;

// G.edgenum=e;

// cout<<e<<endl;

for(i=0;i<G.vexnum;i++)

{

f1>>G.vexs[i].vex;

cout<<G.vexs[i].vex<<" ";

}

for(k=0;k<G.edgenum;++k)

{

f1>>v1>>v2>>weight;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

// cout<<"i: "<<i<<"j: "<<j<<endl;

while((i<0)||(i>(G.vexnum-1))||(j<0)||(j>(G.vexnum-1)))

{

cout<<"The edge doesn't exist,please input again"<<endl;

system("pause");

f1>>v1>>v2;

i=LocateVex\_L(G,v1);

j=LocateVex\_L(G,v2);

}

p = new EdgeNode\_L;

p->adjvex=j;

p->weight=weight;

p->nextedge=G.vexs[i].firstedge;

G.vexs[i].firstedge=p;

G.vexs[j].indegree++;

p = new EdgeNode\_L;

p->adjvex=i;

p->weight=weight;

p->nextedge=G.vexs[j].firstedge;

G.vexs[j].firstedge=p;

G.vexs[i].indegree++;

// cout<<G.vexs[i].vex<<endl;

}

f1.close();

// cout<<"end create"<<endl;

}

int get\_mini(ALGraph G,int n)

{

int mini=9999;

EdgeNode\_L \*p;

p=G.vexs[n].firstedge;

int t;

int re=-1;

while(p)

{

t=p->adjvex;

if(visit[t]==0)

{

if(p->weight<mini)

{

mini=p->weight;

re=p->adjvex;

}

}

p=p->nextedge;

}

return re;

}

int main()

{

ALGraph G;

int n;

int min;

int i=3;

int cur;

int v0=2;

map<int,int> road;

memset(visit,0,sizeof(visit));

CreateDG\_ALG(G,1,1);

display\_tree(G);

n = G.vexnum;

stack<int> s;

//start from v0

v0=LocateVex\_L(G,v0);

visit[v0]=1;

s.push(v0);

n--;

while(n)

{

cur=s.top();

min=get\_mini(G,cur);

if(min!=-1)

{

road[cur]=min;

cout<<G.vexs[min].vex<<" ";

visit[min]=1;

s.push(min);

n--;

}

}

cout<<endl<<"树："<<endl;

for (map<int,int>::iterator it=road.begin(); it!=road.end(); it++)

{

cout<<G.vexs[it->first].vex<<" -> "<<G.vexs[it->second].vex<<" "<<endl;

}

cout<<endl;

return 0;

}

//附txt文件

/\*

9 12

0 1 2 3 6 5 4 8 7

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

4 6 9

4 7 7

5 7 4

7 8 4

6 8 2

6 0 1

\*/

//graph\_w.txt

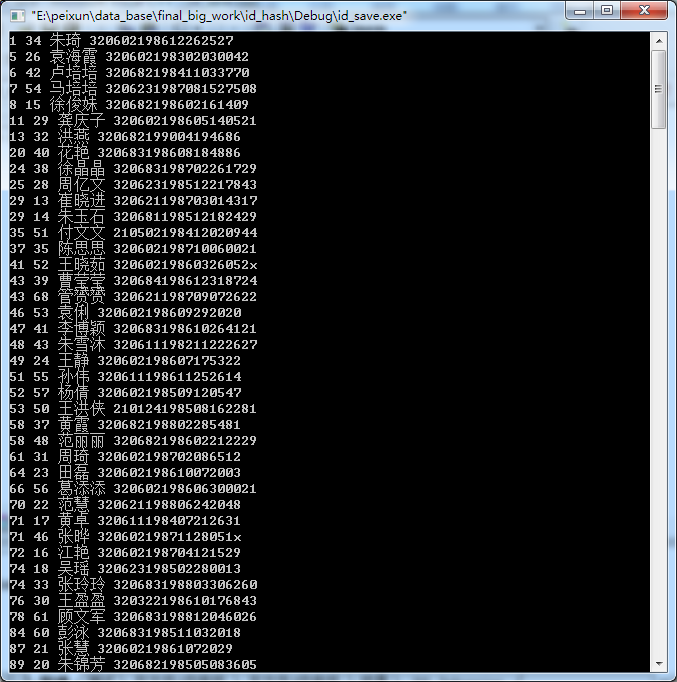
第八次上机实验报告

1.题目：哈希表设计 ( 习题集 P166. 6.2 )；

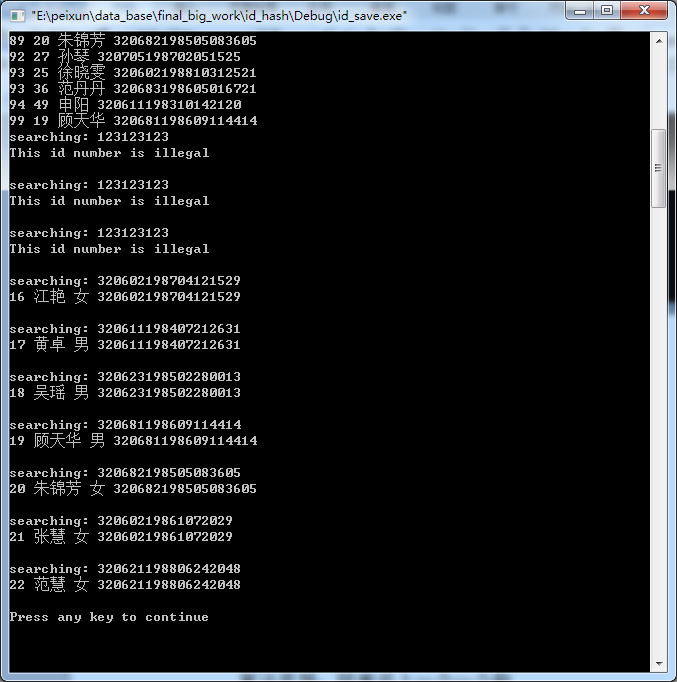
算法思想：折叠法 hash

运行结果：

显示信息



显示查询结果：

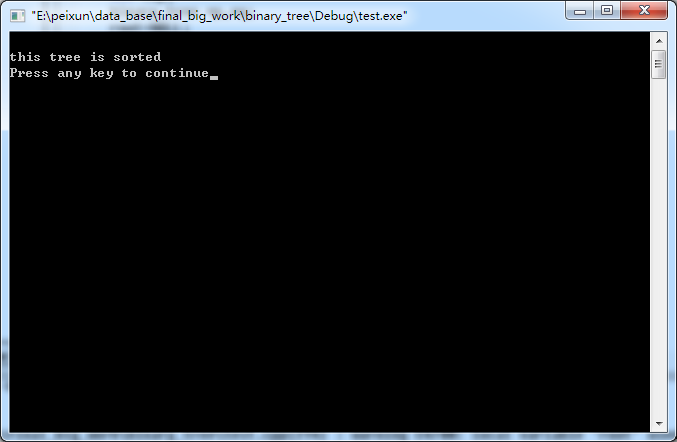


附源程序：

2.题目：  
编写程序，判断一棵二叉树是否为二叉排序树（习题集 P58 9.31 ) (选做题)；

算法思想：中序遍历

运行结果：



附源程序：

#include<iostream>

#include<stack>

#include<fstream>

#include<queue>

using namespace std;

#define N 40

struct binary\_tree{

int data;

binary\_tree \* left;

binary\_tree \* right;

binary\_tree(int a = 0):data(a),left(NULL),right(NULL) {}

};

fstream f1;

void CreateBiTree(binary\_tree\* &BT)

{

int ch;

if(!f1.eof())

{

f1>>ch;

cout<<ch<<endl;

if(ch==0) BT=NULL;

else{

BT=new binary\_tree;

BT->data=ch;

cout<<"left "<<ch<<endl;

CreateBiTree(BT->left);

cout<<"right "<<ch<<endl;

CreateBiTree(BT->right);

}

}

}

void insert(binary\_tree \*cur,binary\_tree \* p)

{

if(cur->data>p->data)

{

if(cur->left==NULL)

{

cur->left=p;

}

else

{

insert(cur->left,p);

}

}else

{

if(cur->right==NULL)

{

cur->right=p;

}

else

{

insert(cur->right,p);

}

}

}

void bfs(binary\_tree \*root)

{

queue<binary\_tree \*> q;

q.push(root);

while(!q.empty())

{

binary\_tree \* p;

p=q.front();

q.pop();

cout<<p->data<<" ";

if(p->left!=NULL) q.push(p->left);

if(p->right!=NULL) q.push(p->right);

}

cout<<endl;

}

binary\_tree\* read\_num(binary\_tree \* root)

{

int tmp;

binary\_tree \*p,\*q;

root=NULL;

p=root;

int n=N;

while(n--)

{

tmp=rand()%10000;

if(n==N-1)

tmp=3000;

// cout<<tmp<<endl;

q = new binary\_tree;

q->data=tmp;

if(p!=NULL){

insert(root,q);

}

else{

root=q;

p=root;

}

}

return root;

}

void mid\_read\_while(binary\_tree \*root)

{

binary\_tree \*p;

p=root;

stack<binary\_tree\*>q;

while(!q.empty()||p)

{

while(p)

{

q.push(p);

p=p->left;

}

p=q.top();

q.pop();

cout<<p->data<<" ";

p=p->right;

}

cout<<endl;

}

void pre\_read\_while(binary\_tree \*root)

{

binary\_tree \*p;

p=root;

stack<binary\_tree\*>q;

while(!q.empty()||p)

{

while(p)

{

cout<<p->data<<" ";

q.push(p);

p=p->left;

}

p=q.top();

q.pop();

p=p->right;

}

cout<<endl;

}

void post\_read\_while(binary\_tree \*root)

{

binary\_tree \*p,\*last;

p=root;

stack<binary\_tree\*>q;

last=NULL;

while(p)

{

q.push(p);

p=p->left;

}

while(!q.empty())

{

p=q.top();

q.pop();

if(p->right==NULL||p->right==last)

{

cout<<p->data<<" ";

last=p;

}

else

{

q.push(p);

p=p->right;

while(p)

{

q.push(p);

p=p->left;

}

}

}

cout<<endl;

}

int check\_is\_sorted(binary\_tree \*root)

{

binary\_tree \*p;

int last;

p=root;

int flag=0;

stack<binary\_tree\*>q;

while(!q.empty()||p)

{

while(p)

{

q.push(p);

p=p->left;

}

p=q.top();

q.pop();

if(!flag)

{

last=p->data;

flag=1;

}

// cout<<p->data<<" ";

if(p->data>=last)

{

last=p->data;

}

else

{

return 0;

}

p=p->right;

}

cout<<endl;

return 1;

}

int main()

{

binary\_tree \*root;

f1.open("bin.txt",ios::in);

root=read\_num(root);

// CreateBiTree(root);//两种模式

/\* cout<<"bfs: "<<endl;

bfs(root);

cout<<"pre\_read: "<<endl;

pre\_read\_while(root);

cout<<"mid\_read: "<<endl;

mid\_read\_while(root);

cout<<"post\_read: "<<endl;

post\_read\_while(root);

\*/

if(!check\_is\_sorted(root))

cout<<"this tree is not sorted"<<endl;

else

cout<<"this tree is sorted"<<endl;

return 0;

}

//附txt文件

/\*

1 2 3 0 0 6 0 0 9 10 0 0 13 0 0

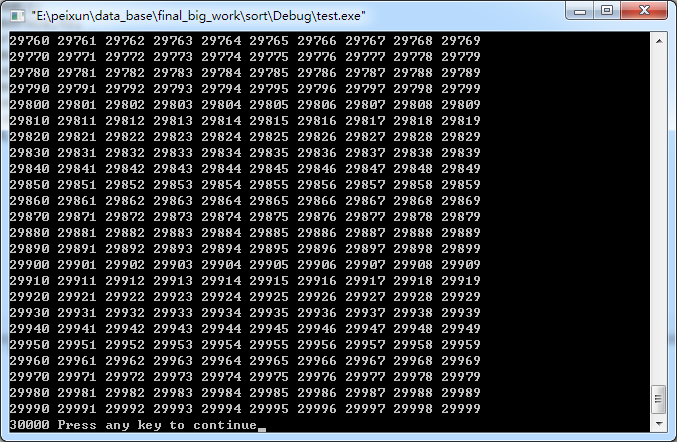
\*/

//bin.txt

3.题目：  
实现交换、选择、归并等简单排序算法；

算法思想：照着书上一个一个敲

运行结果：



附源程序：

//invert\_number.txt 30000 numbers

#include<iostream>

#include<fstream>

using namespace std;

int maxn=30000 ;

void read\_numbers(int\* b,int type)

{

fstream f1;

char a[20];

char \*p;

p=a;

switch(type)

{

case 0:

p="order\_number.txt";

break;

case 1:

p="invert\_number.txt";

break;

case 2:

p="random\_number.txt";

break;

default:

p="random\_number.txt";

}

f1.open(p,ios::in);

int i=1;

while(i<=maxn)

{

f1>>b[i++];

}

f1.close();

}

void display\_number(int a[])

{

int i=1;

while(i<=maxn)

{

cout<<a[i++]<<" ";

if(i%10==0)

cout<<endl;

}

}

void Msort(int \*a,int low,int mid,int high)

{

int \*t;

int i,j,p;

p=0;

i=low;

j=mid+1;

t = new int[high-low+1];

while(i<=mid&&j<=high)

{

if(a[i]<a[j])

{

t[p++]=a[i++];

}

else

{

t[p++]=a[j++];

}

}

while(i<=mid)

{

t[p++]=a[i++];

}

while(j<=high)

{

t[p++]=a[j++];

}

for(p=0,i=low;i<=high;i++,p++)

{

a[i]=t[p];

}

}

void Merge\_sort(int \*a,int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

Merge\_sort(a,low,mid);

Merge\_sort(a,mid+1,high);

Msort(a,low,mid,high);

}

}

void bubble\_sort(int \*a)

{

int tag=0;

int count=0;

for(int bound=1;tag==0;bound++)

{

tag=1;

// cout<<"bound:"<<bound<<endl;

// system("pause");

for(int i=maxn;i>bound;--i)

{

count++;

if(a[i]<a[i-1])

{

// cout<<a[i]<<" "<<a[i-1]<<endl;

// system("pause");

a[0]=a[i];

a[i]=a[i-1];

a[i-1]=a[0];

tag=0;

}

}

// display\_number(a);

}

cout<<"bubble: "<<count<<endl;

}

void select\_sort(int \*a)

{

int count=0;

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

{

int k=i;

for(int j=i+1;j<=maxn;j++)

{

count++;

if(a[j]<a[k])

k=j;

}

if(i!=k)

{

a[0]=a[i];

a[i]=a[k];

a[k]=a[0];

}

}

cout<<"select: "<<count<<endl;

}

int main()

{

int a[30005];

read\_numbers(a,1);

Merge\_sort(a,1,maxn);

read\_numbers(a,1);

bubble\_sort(a);

read\_numbers(a,1);

select\_sort(a);

display\_number(a);

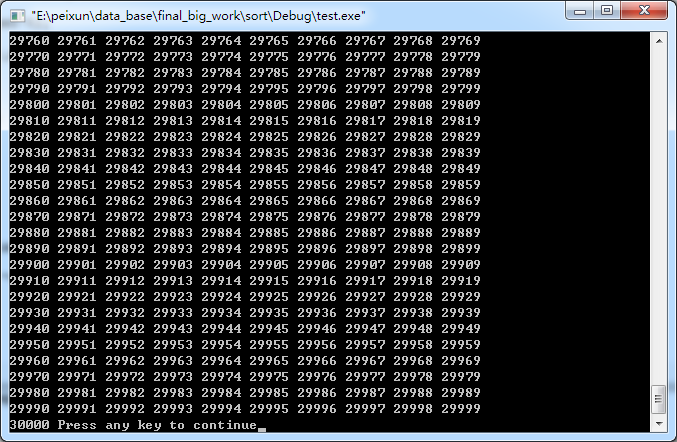
return 0;

}

4.题目：实现快速排序算法；

算法思想：照着书上一个一个敲

运行结果：



附源程序：

#include<iostream>

#include<fstream>

using namespace std;

int maxn=9000;

/\*

30000 29999 29998 29997 29996 29995 29994 29993 29992 29991 29990 29989 29988 29987 29986 29985 29984 29983 29982 29981 29980 29979 29978 29977 29976 29975 29974 29973 29972 29971 29970 29969 29968 29967 29966 29965 29964 29963 29962 29961 29960 29959 29958 29957 29956 29955 29954 29953 29952 29951 29950 29949 29948 29947 29946 29945 29944 29943 29942 29941 29940 29939 29938 29937 29936 29935 29934 29933 29932 29931 29930 29929 29928 29927 29926 29925 29924 29923 29922 29921 29920 29919 29918 29917 29916 29915 29914 29913 29912 29911 29910 29909 29908 29907 29906 29905 29904 29903 29902 29901 29900 29899 29898 29897 29896 29895 29894 29893 29892 29891 29890 29889 29888 29887 29886 29885 29884 29883 29882 29881 29880 29879 29878 29877 29876 29875 29874 29873 29872 29871 29870 29869 29868 29867 29866 29865 29864 29863 29862 29861 29860 29859 29858 29857

\*/

//invert\_number.txt

/\*

9043 2265 19587 3331 12417 16569 25318 31507 4321 25898 21799 29912 25659 4703 7164 26685 6301 5825 17017 4375 9051 22501 1844 9653 19285 955 6311 12728 26744 25097 28885 4901 15496 23064 5169 19897 18716 174 14267 10365 18154 16955 4716 18842 8772 32087 24020 17517 8419 8556 18946 19934 5348 9648 26788 2798 16656 27735 21359 11664 21088 26118 4545 21434 4800 2442 389 23739 14612 29642 18264 10447 31353 17402 2537 30449 16589 2066 31910 21744 10865 19839 32149 21957 14157 10776 19071 548 18197 20819 4436 14161 19659 2489 24523 26677 23430 5783 13897 29873

\*/

//random\_number.txt

/\*

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

\*/

//order\_number.txt

void read\_numbers(int\* b,int type)

{

fstream f1;

char a[20];

char \*p;

p=a;

switch(type)

{

case 0:

p="order\_number.txt";

break;

case 1:

p="invert\_number.txt";

break;

case 2:

p="random\_number.txt";

break;

default:

p="random\_number.txt";

}

f1.open(p,ios::in);

int i=1;

while(i<=maxn)

{

f1>>b[i++];

}

f1.close();

}

void display\_number(int a[])

{

int i=1;

while(i<=maxn)

{

cout<<a[i++]<<" ";

if(i%10==0)

cout<<endl;

}

}

int partition(int \*a,int low,int high)

{

int pivotkey;

a[0]=a[low];

pivotkey=a[low];

while(low<high)

{

while(low<high&&a[high]>pivotkey)

{

high--;

}

a[low]=a[high];

while(low<high&&a[low]<pivotkey)

{

low++;

}

a[high]=a[low];

// cout<<endl<<a[low]<<endl;

// system("pause");

}

a[low]=a[0];

// cout<<"low: "<<low<<endl;

// system("pause");

return low;

}

void q\_sort(int \*a,int low,int high)

{

if(low<high){

int pivotpos;

pivotpos=partition(a,low,high);

// if(low!=pivotpos)

q\_sort(a,low,pivotpos-1);

// if(high!=pivotpos)

q\_sort(a,pivotpos+1,high);

}

}

void quick\_sort(int \*a)

{

q\_sort(a,1,maxn);

}

int main()

{

int a[30005];

read\_numbers(a,1);

display\_number(a);

quick\_sort(a);

cout<<endl<<"then: "<<endl;

display\_number(a);

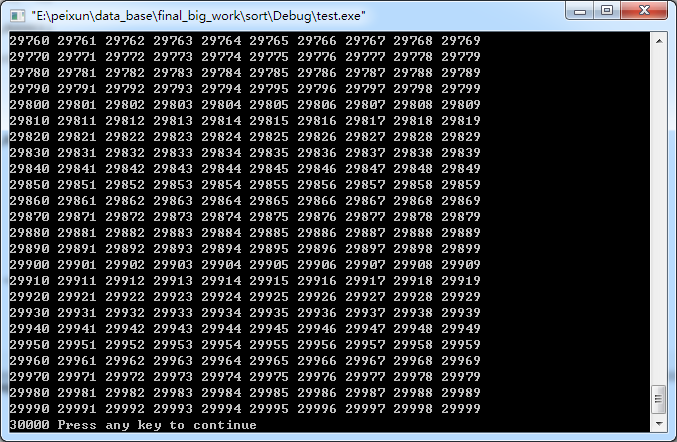
return 0;

}

5.题目：  
实现堆排序算法；

算法思想：照着书上一个一个敲

运行结果：



附源程序：

/\*

30000 29999 29998 29997 29996 29995 29994 29993 29992 29991 29990 29989 29988 29987 29986 29985 29984 29983 29982 29981 29980 29979 29978 29977 29976 29975 29974 29973 29972 29971 29970 29969 29968 29967 29966 29965 29964 29963 29962 29961 29960 29959 29958 29957 29956 29955 29954 29953 29952 29951 29950 29949 29948 29947 29946 29945 29944 29943 29942 29941 29940 29939 29938 29937 29936 29935 29934 29933 29932 29931 29930 29929 29928 29927 29926 29925 29924 29923 29922 29921 29920 29919 29918 29917 29916 29915 29914 29913 29912 29911 29910 29909 29908 29907 29906 29905 29904 29903 29902 29901 29900 29899 29898 29897 29896 29895 29894 29893 29892 29891 29890 29889 29888 29887 29886 29885 29884 29883 29882 29881 29880 29879 29878 29877 29876 29875 29874 29873 29872 29871 29870 29869 29868 29867 29866 29865 29864 29863 29862 29861 29860 29859 29858 29857

\*/

//invert\_number.txt

/\*

9043 2265 19587 3331 12417 16569 25318 31507 4321 25898 21799 29912 25659 4703 7164 26685 6301 5825 17017 4375 9051 22501 1844 9653 19285 955 6311 12728 26744 25097 28885 4901 15496 23064 5169 19897 18716 174 14267 10365 18154 16955 4716 18842 8772 32087 24020 17517 8419 8556 18946 19934 5348 9648 26788 2798 16656 27735 21359 11664 21088 26118 4545 21434 4800 2442 389 23739 14612 29642 18264 10447 31353 17402 2537 30449 16589 2066 31910 21744 10865 19839 32149 21957 14157 10776 19071 548 18197 20819 4436 14161 19659 2489 24523 26677 23430 5783 13897 29873

\*/

//random\_number.txt

/\*

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

\*/

//order\_number.txt

#include<iostream>

#include<fstream>

using namespace std;

int maxn=30000;

void read\_numbers(int\* b,int type)

{

fstream f1;

char a[20];

char \*p;

p=a;

switch(type)

{

case 0:

p="order\_number.txt";

break;

case 1:

p="invert\_number.txt";

break;

case 2:

p="random\_number.txt";

break;

default:

p="random\_number.txt";

}

f1.open(p,ios::in);

int i=1;

while(i<=maxn)

{

f1>>b[i++];

}

f1.close();

}

void display\_number(int a[])

{

int i=1;

while(i<=maxn)

{

cout<<a[i++]<<" ";

if(i%10==0)

cout<<endl;

}

}

void HeapAdjust(int \*a,int low,int high)

{

a[0]=a[low];

for(int j=2\*low;j<=high;j\*=2)

{

if((j<high)&&(a[j]<a[j+1]))

j++;

if(a[0]>=a[j])

break;

a[low]=a[j];

low=j;

}

a[low]=a[0];

}

void HeapSort(int \*a)

{

int high;

for(int i=maxn/2;i>0;--i)

{

HeapAdjust(a,i,maxn);

}

for(high=maxn;high>1;high--)

{

a[0]=a[1];

a[1]=a[high];

a[high]=a[0];

HeapAdjust(a,1,high-1);

}

}

int main()

{

int a[30005];

read\_numbers(a,1);

HeapSort(a);

cout<<endl<<"then: "<<endl;

display\_number(a);

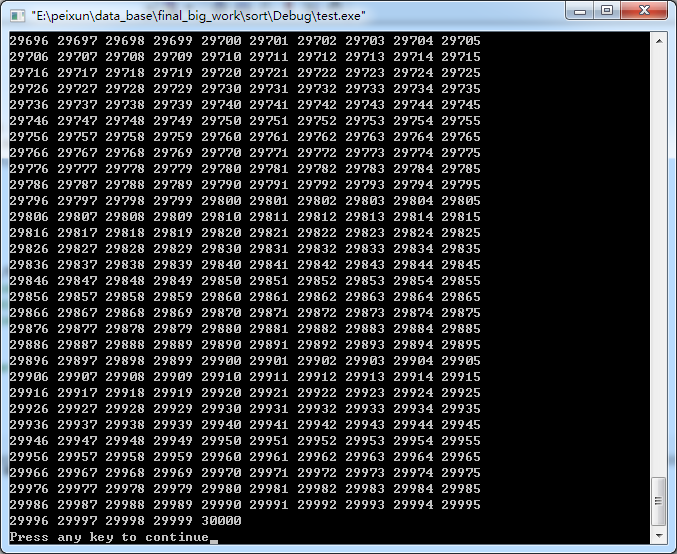
return 0;

}

6.题目：实现基数排序

算法思想：照着书上一个一个敲

运行结果：



附源程序：

/\*

30000 29999 29998 29997 29996 29995 29994 29993 29992 29991 29990 29989 29988 29987 29986 29985 29984 29983 29982 29981 29980 29979 29978 29977 29976 29975 29974 29973 29972 29971 29970 29969 29968 29967 29966 29965 29964 29963 29962 29961 29960 29959 29958 29957 29956 29955 29954 29953 29952 29951 29950 29949 29948 29947 29946 29945 29944 29943 29942 29941 29940 29939 29938 29937 29936 29935 29934 29933 29932 29931 29930 29929 29928 29927 29926 29925 29924 29923 29922 29921 29920 29919 29918 29917 29916 29915 29914 29913 29912 29911 29910 29909 29908 29907 29906 29905 29904 29903 29902 29901 29900 29899 29898 29897 29896 29895 29894 29893 29892 29891 29890 29889 29888 29887 29886 29885 29884 29883 29882 29881 29880 29879 29878 29877 29876 29875 29874 29873 29872 29871 29870 29869 29868 29867 29866 29865 29864 29863 29862 29861 29860 29859 29858 29857

\*/

//invert\_number.txt

/\*

9043 2265 19587 3331 12417 16569 25318 31507 4321 25898 21799 29912 25659 4703 7164 26685 6301 5825 17017 4375 9051 22501 1844 9653 19285 955 6311 12728 26744 25097 28885 4901 15496 23064 5169 19897 18716 174 14267 10365 18154 16955 4716 18842 8772 32087 24020 17517 8419 8556 18946 19934 5348 9648 26788 2798 16656 27735 21359 11664 21088 26118 4545 21434 4800 2442 389 23739 14612 29642 18264 10447 31353 17402 2537 30449 16589 2066 31910 21744 10865 19839 32149 21957 14157 10776 19071 548 18197 20819 4436 14161 19659 2489 24523 26677 23430 5783 13897 29873

\*/

//random\_number.txt

/\*

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

\*/

//order\_number.txt

#include<iostream>

#include<fstream>

using namespace std;

#define Space\_Size 30000

#define RD 10

#define Key\_Size 8

typedef int InfoType;

int maxn=Space\_Size-5;

int f[10];

int e[10];

typedef struct SLNode

{

int keys[Key\_Size];

InfoType otherInfo;

int next;

int key;

SLNode()

{

key=0;

//memset(keys,0,sizeof(keys));

for(int i=0;i<Key\_Size;i++)

{

keys[i]=0;

}

next=0;

}

}LNode;

typedef struct

{

SLNode r[Space\_Size+1];

int keynum;

int length;

}SLList;

typedef int ArrType[RD];

void init\_L(SLList \*b,int i,int tmp)

{

int j=0;

b->r[i].key=tmp;

b->r[i-1].next=i;

while(tmp>0)

{

b->r[i].keys[j]=tmp%10;

tmp/=10;

j++;

}

}

void read\_numbers(SLList \*b,int type)

{

fstream f1;

char a[20];

char \*p;

p=a;

switch(type)

{

case 0:

p="order\_number.txt";

break;

case 1:

p="invert\_number.txt";

break;

case 2:

p="random\_number.txt";

break;

default:

p="random\_number.txt";

}

f1.open(p,ios::in);

int i=1;

int tmp;

while(i<maxn)

{

f1>>tmp;

init\_L(b,i++,tmp);

}

f1.close();

}

void display\_number(SLList \*b)

{

int i=0;

while(i<maxn)

{

cout<<b->r[i++].next<<" ";

if(i%10==0)

cout<<endl;

}

}

void Distribute(SLNode \*r,int i,int \*f,int \*e)

{

int p;

int j;

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

{

f[j]=0;

}

for(p=r[0].next;p;p=r[p].next)

{

j=r[p].keys[i];

if(!f[j]) f[j]=p;

else r[e[j]].next=p;

e[j]=p;

}

}

void Collect(SLNode \*r,int i,int \*f,int \*e)

{

int j;

int t;

for(j=0;!f[j];j++);

r[0].next=f[j];

t=e[j];

//cout<<"0 :"<<endl;

//cout<<endl<<j<<" f "<<f[j]<<" "<<e[j]<<" "<<" "<<r[f[j]].next<<" "<<r[e[j]].next<<endl;

/\* for(j=0;j<RD;j++)

{

cout<<endl<<f[j]<<" "<<e[j]<<endl;

i=j;

while(i!=0)

{

cout<<r[i].key<<" ";

i=r[i].next;

}

}

\*/

while(j<(RD-1))

{

// cout<<"find:"<<endl;

for(j++;(j<(RD-1))&&(!f[j]);j++);

// cout<<"j: "<<j<<" f[j]:"<<f[j]<<" e[j]"<<e[j]<<" "<<r[e[j]].next<<endl;

// cout<<j<<endl;

// system("pause");

// cout<<f[j]<<" ";

// cout<<endl<<"j: "<<j<<" f[j]:"<<f[j]<<" "<<e[j]<<" "<<r[t].next<<endl;

if(f[j])

{

r[t].next=f[j];

t=e[j];

}

// cout<<"j: "<<j<<" f[j]:"<<f[j]<<" "<<e[j]<<" "<<r[t].next<<endl;

//cout<<endl<<j<<" f "<<f[j]<<" "<<e[j]<<" "<<" "<<r[f[j]].next<<" "<<r[e[j]].next<<endl;

}

r[t].next=0;

}

void RadixSort(SLList \*L)

{

int i;

int j=0;

for(i=0;i<5;i++)

{

Distribute(L->r,i,f,e);

// display\_number(a);

Collect(L->r,i,f,e);

}

SLNode \*r;

r=L->r;

i=r[0].next;

while(i!=0)

{

j++;

if(j%10==0) cout<<endl;

cout<<r[i].key<<" ";

i=r[i].next;

}

}

int main()

{

SLList \*a;

a=new SLList;

read\_numbers(a,1);

// display\_number(a);

// Straight\_sort(a);

// display\_number(a);

RadixSort(a);

cout<<endl;

//display\_number(a);

return 0;

}