**南京航空航天大学**

**数据结构课程设计**

题 目：数据结构

学生姓名：王锡龙

学 号：161640213

所在院(系)：计算机科学与技术学院

专 业：物联网工程

班 级：1616402

指 导 教 师：孙涵

完成日期:2018年1月7日 星期日

## 目录

1. 购物网站信息管理--------------------------------3
2. 迷宫问题---------------------------------------21
3. 二叉树的应用-----------------------------------31
4. Huffman编码与解码-----------------------------45
5. 无线传感器网络---------------------------------56
6. 排序算法比较-----------------------------------67
7. 公交线路提示-----------------------------------96
8. 算术表达式求值--------------------------------112
9. 电子小字典------------------------------------122
10. 连连看-------------------------------------133
11. 结束语（心得体会）-------------------------139

## 购物网站信息管理（链表）

[问题描述]

设计一个程序，对商铺信息管理，商铺信息包括：商铺编号，商铺名，信誉度，（商品名称1，价格1，销量1），（商品名称2，价格2，销量2），（商品名称3，价格3，销量3）…。

假设商品名称包括(毛巾，牙刷，肥皂，牙膏，洗发水，沐浴露等)，每个商铺具有其中事先确定若干商品及价格，由文件输入，销量初始为0。

[基本要求]

（1）建立一个单向链表存储所有商铺信息（至少30个），以编号为序，编号从1开始递增，从文件中读取数据，并能将数据存储在文件。商铺信息结点的数据结构自行设计。

（2）可以增、删商铺。增加商铺，编号自动加一，插入链表尾部；删除商铺以编号为准，并修改后续结点的编号，保持编号连续性。

（3）可以增、删选定商铺中的商品，修改商品价格。

（4）查询某一种商品名称，建立一个双向循环链表，结点信息是包含该商品的所有商铺编号，商铺名，信誉度，商品名称，价格，销量，按销量高至低排序，并可逐一显示。

（5）购买某一商铺的商品，修改单向链表中商品的信息的销量。

（6）任何的商铺信息变化，实现文件存储。

**采用的数据结构：**

本题主要采用链表的结构，这里使用了三个链表结构分别如下:

typedef struct DuLNode{

int num;

char name[30];

int credit;

LNode\_G Goods;

struct DuLNode \*prior;

struct DuLNode \*next;

}DuLNode,\*DuLinkList;//查找商品时构建的循环链表结点

typedef struct LNode\_S{

int num;

char name[30];

int credit;

ListGoods Goods;

struct LNode\_S \*next;

}LNode\_S,\*ListShops;//店铺链表结点

typedef struct LNode\_G{

char name[30];

float price;

int sale\_volume;

struct LNode\_G \*next;

}LNode\_G,\*ListGoods;//商品链表结点

**算法思想：**

每个商店构成一个结点，每个结点中都含有一个商品链表，增减商店结点即链表的插入删除操作。数据实现文件存储即在运行内存中做完处理后全部放到文件中。

**源代码：**

1.Goods.h

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define OVERFLOW -2

#define INFEASIBLE -1

#define ERROR 0

#define OK 1

typedef int Status;

typedef struct LNode\_G{

char name[30];

float price;

int sale\_volume;

struct LNode\_G \*next;

}LNode\_G,\*ListGoods;

Status InitList\_G(ListGoods &L,FILE \*f)

{

L=(LNode\_G\*)malloc(sizeof(LNode\_G));

if(!L)

exit(OVERFLOW);

L->next=NULL;

LNode\_G \*q=L;

while(fgetc(f)!='\n')//文件中没个店铺之间的数据用换行隔开

{

fseek(f,-1,1);//读到的不是结束标志所以退格

LNode\_G \*p=(LNode\_G\*)malloc(sizeof(LNode\_G));

fscanf(f,"%s%f%d",p->name,&(p->price),&(p->sale\_volume));

if(!p)

exit(OVERFLOW);

q->next=p;

q=q->next;

}

q->next=NULL;

return OK;

}

Status InitList\_G\_Put(ListGoods &L,int n)

{

L=(LNode\_G\*)malloc(sizeof(LNode\_G));

if(!L)

exit(OVERFLOW);

L->next=NULL;

LNode\_G \*q=L;

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

{

LNode\_G \*p=(LNode\_G\*)malloc(sizeof(LNode\_G));

if(!p)

exit(OVERFLOW);

printf("please input the Goods' name and price\n");

scanf("%s%f",p->name,&p->price);

p->sale\_volume=0;

q->next=p;

q=q->next;

}

q->next=NULL;

return OK;

}

Status Destroy\_G(ListGoods &L,FILE \*f)

{

LNode\_G \*p=L->next,\*q=p;

while(p->next)

{

fprintf(f,"%s %.2f %d",p->name,p->price,p->sale\_volume);

p=p->next;

free(q);

q=p;

fputc(' ',f);

}

fprintf(f,"%s %.2f %d",p->name,p->price,p->sale\_volume);

free(p);

fputc('\n',f);

free(L);

return OK;

}

Status Destroy\_G\_S(ListGoods &L)

{

LNode\_G \*p=L->next,\*q=L;

while(p)

{

free(q);

q=p;

p=p->next;

}

return OK;

}

Status Insert\_G(ListGoods &L)

{

LNode\_G \*p=L->next,\*q;

q=(LNode\_G\*)malloc(sizeof(LNode\_G));

if(!q)

exit(OVERFLOW);

printf("please enter the information of the goods(name and price):\n");

scanf("%s%f",q->name,&q->price);

q->sale\_volume=0;

while(p->next!=NULL)

{

p=p->next;

}

p->next=q;

q->next=NULL;

return OK;

}

Status Delete\_G(ListGoods &L, char \*n)

{

LNode\_G \*p=L->next,\*q=L;

while(p!=NULL && strcmp(p->name,n)!=0)

{

q=p;

p=p->next;

}

if(!p)

{

printf("Goods is not exist!\n");

return ERROR;

}

q->next=p->next;

free(p);

printf("Delete success!\n");

return OK;

}

Status Price\_revise(ListGoods &L, char \*n)

{

LNode\_G \*p=L->next;

while(p!=NULL && strcmp(p->name,n)!=0)

{

p=p->next;

}

if(!p)

{

printf("Goods is not exist!\n");

return ERROR;

}

printf("enter the price you revise:\n");

scanf("%f",&p->price);

printf("价格修改成功！\n");

return OK;

}

Status Purchase(ListGoods &L, char \*n)

{

LNode\_G \*p=L->next,\*q=L;

while(p!=NULL && strcmp(p->name,n)!=0)

{

q=p;

p=p->next;

}

if(!p)

{

printf("Goods is not exist!\n");

return ERROR;

}

q->next=p->next;

printf("Buy success!\n");

p->sale\_volume++;

q=L;

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

{

q=q->next;

}

p->next=q->next;

q->next=p;

return OK;

}

Status Goods\_Traverse(ListGoods L)

{

LNode\_G \*p=L->next;

while(p)

{

printf("商品名：%s 价格：%.2f 销量：%d\n",p->name,p->price,p->sale\_volume);

p=p->next;

}

return OK;

}

2.Shop.h

#include"Goods.h"

typedef struct LNode\_S{

int num;

char name[30];

int credit;

ListGoods Goods;

struct LNode\_S \*next;

}LNode\_S,\*ListShops;

typedef struct{

ListShops head,rear;

int len;

}LinkList;

typedef struct DuLNode{

int num;

char name[30];

int credit;

LNode\_G Goods;

struct DuLNode \*prior;

struct DuLNode \*next;

}DuLNode,\*DuLinkList;

Status InitList\_S(LinkList &L, FILE \*f)

{

L.head=(LNode\_S\*)malloc(sizeof(LNode\_S));

if(!L.head)

exit(OVERFLOW);

if(!feof(f))

{

fscanf(f,"%d %s %d",&(L.head->num),L.head->name,&(L.head->credit));

InitList\_G(L.head->Goods,f);

L.len=1;

}

LNode\_S \*q=L.head;

while(fgetc(f)!=EOF)

{

fseek(f,-1,1);

LNode\_S \*p=(LNode\_S\*)malloc(sizeof(LNode\_S));

fscanf(f,"%d %s %d",&(p->num),p->name,&(p->credit));

InitList\_G(p->Goods,f);

q->next=p;

q=q->next;

L.len++;

if(feof(f))

break;

}

q->next=NULL;

L.rear=q;

fclose(f);

return OK;

}

Status Destroy\_S(LinkList &L, FILE \*f)

{

LNode\_S \*p=L.head,\*q=p;

while(p)

{

fprintf(f,"%d %s %d",p->num,p->name,p->credit);

fputc(' ',f);

Destroy\_G(p->Goods,f);

q=p;

p=p->next;

free(q);

}

L.rear=L.head=NULL;

fclose(f);

return OK;

}

Status Insert\_S(LinkList &L)

{

LNode\_S \*p;

int n;

p=(LNode\_S\*)malloc(sizeof(LNode\_S));

printf("please input the information of the new Shop:\n");

printf("name creditworthiness\n");

scanf("%s%d",p->name,&(p->credit));

n=L.rear->num;

p->num=n+1;

p->next=L.rear->next;

L.rear->next=p;

L.rear=p;

printf("input the mount of the Goods:\n");

scanf("%d",&n);

InitList\_G\_Put(p->Goods,n);

printf("增加成功！\n");

L.len++;

return OK;

}

Status Goods\_Add(LinkList &L, int n)

{

LNode\_S \*p=L.head;

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

Insert\_G(p->Goods);

printf("添加成功！\n");

return OK;

}

Status Goods\_Del(LinkList &L, int n)

{

LNode\_S \*p=L.head;

char name[30];

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

/\* printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);\*/

printf("请输入你要删除的商品名称:\n");

scanf("%s",name);

Delete\_G(p->Goods,name);

return OK;

}

Status Shop\_Find(LinkList &L,int n)

{

LNode\_S \*p=L.head;

char name[30];

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

return OK;

}

Status Goods\_Revise(LinkList &L, int n)

{

LNode\_S \*p=L.head;

char name[30];

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

/\* printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);\*/

printf("请输入你要修改价格的商品名称:\n");

scanf("%s",name);

Price\_revise(p->Goods,name);

return OK;

}

Status Goods\_Purchase(LinkList &L, int n)

{

LNode\_S \*p=L.head;

char name[30];

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

printf("请输入你要购买的商品名称:\n");

scanf("%s",name);

Purchase(p->Goods,name);

return OK;

}

Status PrintGoods(LinkList &L, int n)

{

LNode\_S \*p=L.head;

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

p=p->next;

}

printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

return OK;

}

Status Delete\_S(LinkList &L, int n)

{

LNode\_S \*p=L.head,\*q=p;

if(n>L.len||n<=0)

{

printf("The Shop is not exist!\n");

return ERROR;

}

while(p->num!=n)

{

q=p;

p=p->next;

}

printf("删除店铺信息：\n编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

q->next=p->next;

q=p;

p=p->next;

while(p!=NULL)

{

p->num--;

p=p->next;

}

Destroy\_G\_S(q->Goods);

free(q);

printf("Delete success!\n");

L.len--;

return OK;

}

Status Shop\_Traverse(LinkList L)

{

LNode\_S \*p=L.head;

while(p!=L.rear)

{

printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

printf("\n");

p=p->next;

}

printf("编号：%d 店名：%s 信誉度：%d\n",p->num,p->name,p->credit);

printf("商品：\n");

Goods\_Traverse(p->Goods);

printf("\n");

return OK;

}

LNode\_G\* Locate(LNode\_S &L, char \*n)

{

LNode\_G \*p=L.Goods->next;

while(p!=NULL)

{

if(strcmp(p->name,n)==0)

{

return p;

}

p=p->next;

}

return NULL;

}

Status Sort\_Du(DuLinkList &L)

{

if(L==NULL)

{

return ERROR;

}

DuLNode \*p=L->next,\*p1=L,\*q=L;

while(p!=L)

{

if(p->Goods.sale\_volume > p1->Goods.sale\_volume)

{

p->prior->next=p->next;

p->next->prior=p->prior;

while(q!=p1&&q->Goods.sale\_volume > p->Goods.sale\_volume)

{

q=q->next;

}

p->prior=q->prior;

q->prior->next=p;

q->prior=p;

p->next=q;

if(p1==L||q==L)

{

L=p;

p1=p;

p=p1->next;

}

q=L;

p=p1->next;

}

else

{

p1=p;

p=p->next;

}

}

return OK;

}

Status Traverse\_Du(DuLinkList &L)

{

if(L==NULL)

{

printf("找不到该商品！\n");

return ERROR;

}

DuLNode \*p=L;

while(p->next!=L)

{

printf("编号：%d 店名：%s 信誉度：%d\n商品名：%s 价格：%.2f 销量：%d\n",p->num,p->name,p->credit,p->Goods.name,p->Goods.price,p->Goods.sale\_volume);

p=p->next;

}

printf("编号：%d 店名：%s 信誉度：%d\n商品名：%s 价格：%.2f 销量：%d\n",p->num,p->name,p->credit,p->Goods.name,p->Goods.price,p->Goods.sale\_volume);

return OK;

}

Status InitList\_Du(DuLinkList &Du\_L, LinkList &L,char \*s)

{

DuLNode \*q1;

LNode\_S \*p=L.head;

LNode\_G \*temp;

int n=0;

Du\_L=NULL;

while(p!=NULL)

{

if(temp=Locate(\*p,s))

{

if(n==0)

{

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

q->credit=p->credit;

strcpy(q->name,p->name);

q->num=p->num;

strcpy((q->Goods.name),temp->name);

q->Goods.price=temp->price;

q->Goods.sale\_volume=temp->sale\_volume;

q->next=q->prior=NULL;

q1=q;

Du\_L=q;

n++;

}

else

{

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

q->credit=p->credit;

strcpy(q->name,p->name);

q->num=p->num;

strcpy((q->Goods.name),temp->name);

q->Goods.price=temp->price;

q->Goods.sale\_volume=temp->sale\_volume;

q1->next=q;

q->prior=q1;

q1=q;

}

}

p=p->next;

}

if(!Du\_L)

return ERROR;

q1->next=Du\_L;

Du\_L->prior=q1;

Sort\_Du(Du\_L);

return OK;

}

Status Destroy\_Du(DuLinkList &L)

{

if(L==NULL)

return ERROR;

DuLNode \*p=L->next,\*q;

while(p!=L)

{

q=p;

p=p->next;

free(q);

}

free(L);

return OK;

}

3.main.cpp

#include"Shop.h"

int main()

{

LinkList Shop;

DuLinkList Du\_L;

FILE \*fp,\*f\_w;

int n,k,t;//用于操作判断

int num;

char filename[30],Goods\_name[30];

printf("please input the filename:\n");

scanf("%s",filename);

if((fp=fopen(filename,"r"))==NULL)

{

printf("cannot open file!\n");

exit(ERROR);

}

if(InitList\_S(Shop,fp))

{

printf("Data Reading Success!\n");

}

printf("请按相应按键进行操作:\n");

printf("1:店铺管理 2:查询商品 3:购买商品 0:退出程序\n");

scanf("%d",&n);

while(n!=0)

{

if(n==1)

{

printf("1:增加商铺 2:删除店铺 3:更改商品 4:查看店铺\n");

scanf("%d",&k);

if(k==1)

{

Insert\_S(Shop);

}

else if(k==2)

{

printf("请输入你要删除的店铺编号：\n");

scanf("%d",&num);

Delete\_S(Shop,num);

}

else if(k==3)

{

printf("请输入店铺编号:\n");

scanf("%d",&num);

Shop\_Find(Shop,num);

printf("1:增加商品 2:删除商品 3:修改商品价格\n");

scanf("%d",&t);

switch(t){

case 1:Goods\_Add(Shop,num);break;

case 2:Goods\_Del(Shop,num);break;

case 3:Goods\_Revise(Shop,num);break;

default:printf("输入错误操作指令.\n");

}

}

else if(k==4)

{

printf("1:查看所有店铺 2:查看单个\n");

scanf("%d",&t);

if(t==1)

{

Shop\_Traverse(Shop);

}

else if(t==2)

{

printf("请输入店铺编号:\n");

scanf("%d",&num);

PrintGoods(Shop,num);

}

else

{

printf("输入错误操作指令.\n");

}

}

else

{

printf("输入错误操作指令.\n");

}

}

else if(n==2)

{

printf("请输入要查询商品名称:\n");

scanf("%s",Goods\_name);

InitList\_Du(Du\_L,Shop,Goods\_name);

Traverse\_Du(Du\_L);

Destroy\_Du(Du\_L);

}

else if(n==3)

{

printf("请输入店铺编号:\n");

scanf("%d",&num);

Goods\_Purchase(Shop,num);

}

else

{

printf("输入错误操作指令.\n");

}

printf("1:店铺管理 2:查询商品 3:购买商品 0:退出程序\n");

scanf("%d",&n);

}

f\_w=fopen(filename,"w");

if(!f\_w)

{

printf("file revise failed!\n");

exit(ERROR);

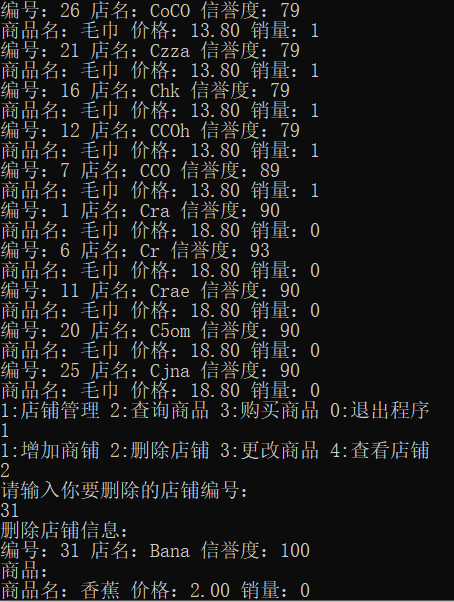
}

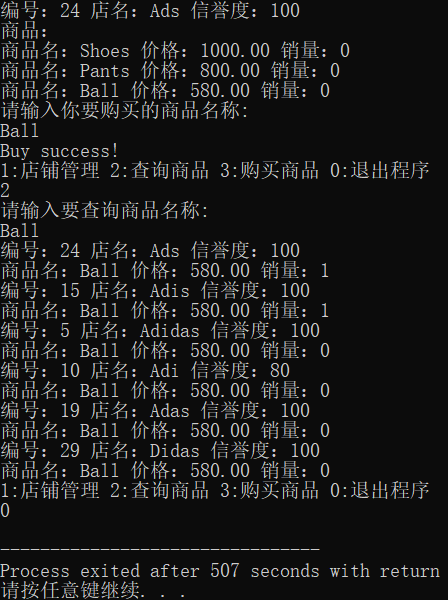
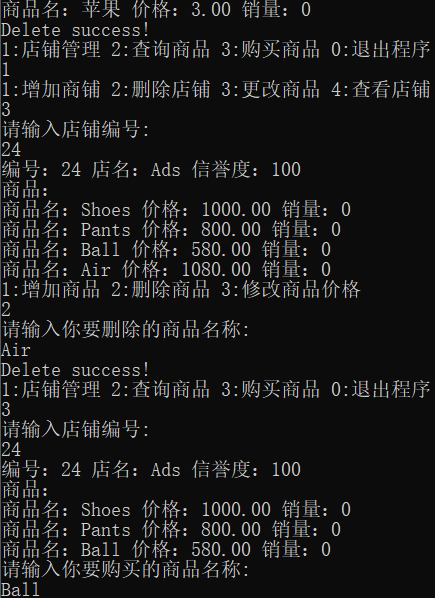
Destroy\_S(Shop,f\_w);

return 0;

}

**测试数据和结果：**





**算法时间复杂度：每次删除店铺O(n)，因为需要调整店铺序号所以无法避免要对整个链表进行一次遍历。增加O(1)，只需在最后插入一个结点即可，商品的修改同理。查询商品的时间复杂度为O(n^2)因为需要摘下结点并形成循环链表同时在插入的时候进行插入排序。**

**代码行：600行**

## 迷宫问题（栈的应用）

[问题描述]

利用栈操作实现迷宫问题求解。

[基本要求]

（1）从文件中读取数据，生成模拟迷宫地图，不少于10行10列。

（2）给出任意入口和出口，显示输出迷宫路线。

**数据结构：**

使用栈结构：

typedef struct

{

int row;//row表示"行"号

int line;//line表示"列"号

}PosType;//位置的元素类型

typedef struct

{

int ord;//该通道在路径上的"序号"

PosType seat;//通道块在迷宫中的"坐标位置"

int di;//从此通道走向下以通道块的"方向",用1-4表示四个方向

}ElemType;//栈的元素类型

**算法思想:**

用回溯法的思想，从出发点开始每次前进一步，如果可通就入栈，否则改变栈顶元素的下一方向信息直到四个方向全部走完，不能通则Pop一个直到找到出口，结束或者栈空没有通路。

**源代码：**

**Maze.h**

#include<stdio.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 20

#define OVERFLOW -2

#define OK 1

#define ERROR 0

#define N 22 //迷宫最大行列数

typedef int Status;

typedef struct

{

int row;//row表示"行"号

int line;//line表示"列"号

}PosType;//位置的元素类型

typedef struct

{

int ord;//该通道在路径上的"序号"

PosType seat;//通道块在迷宫中的"坐标位置"

int di;//从此通道走向下以通道块的"方向"

}ElemType;//栈的元素类型

typedef struct

{

ElemType \*base;

ElemType \*top;

int stacksize;

}SqStack;

Status InitStack(SqStack &S)

{

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

if(!S.base)

exit(OVERFLOW);

S.top=S.base;

S.stacksize=STACK\_INIT\_SIZE;

return OK;

}

Status DestroyStack(SqStack &S)

{

if(!S.base)

{

free(S.base);

S.top=S.base=NULL;

S.stacksize=0;

}

return OK;

}

Status Push(SqStack &S, ElemType &e)

{

if(S.top-S.base>=S.stacksize)

{

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

if(!S.base)

exit(OVERFLOW);

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

S.stacksize+=STACKINCREMENT;

}

\*S.top++=e;

return OK;

}

Status Pop(SqStack &S, ElemType &e)

{

if(S.base==S.top)

return ERROR;

e=\*--S.top;

return OK;

}

Status StackEmpty(SqStack S)

{

if(S.top==S.base)

return OK;

else

return ERROR;

}

Status Stack\_Traverse(SqStack &S)//用于输出最终的通路

{

ElemType \*p=S.base;

for(p;p<S.top;p++)

{

printf("%d,%d\n",(\*p).seat.row,(\*p).seat.line);

}

return OK;

}

void InitMaze(int maze[N][N], int &x, int &y)//初始化迷宫

{

FILE \*fp;

char s[N];

int i,j;

printf("请输入初始化迷宫的文件名：\n");

scanf("%s",s);

fp=fopen(s,"r");

if(!fp)

{

printf("Open file failed!\n");

exit(ERROR);

}

fscanf(fp,"%d %d ",&x,&y);//读入迷宫的行数和列数。

while(!feof(fp))

{

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

{

for(j=1; j<=y; j++)

{

fscanf(fp,"%d",&(maze[i][j]));//读入迷宫信息

}

}

}

printf("读入成功！\n\n");

fclose(fp);

}

void PrintMaze(int maze[N][N], int x, int y)//显示读入的迷宫图

{

int i,j;

printf("读入的迷宫行数和列数:");

printf("row:%d,col:%d\n",x,y);

printf("读入的迷宫(#表示墙):\n");

for(i=0;i<y+2;i++)

printf("%c ",'#');

printf("\n");

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

{

printf("%c ",'#');

for(int j=1;j<y+1;j++)

{

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

}

printf("%c",'#');

printf("\n");

}

for(i=0;i<y+2;i++)

printf("%c ",'#');

printf("\n\n");

}

Status Pass(int maze[N][N], PosType CurPos)//判断当前位置是否可通

{

if(maze[CurPos.row][CurPos.line]==0)

return OK;//如果当前位置是可以通过,返回1

else

return ERROR;//其它情况返回0

}

void MarkFoot(int maze[N][N], PosType CurPos)//标记当前位置不可通或者已走过，并用8标记

{

maze[CurPos.row][CurPos.line]=8;

}

PosType NextPos(PosType CurPos, int Di)//进入下一位置

{

PosType ChangePos;

switch(Di)

{

case 1://向右走一步

ChangePos.row=CurPos.row;

ChangePos.line=CurPos.line+1;

break;

case 2://向下走一步

ChangePos.row=CurPos.row+1;

ChangePos.line=CurPos.line;

break;

case 3://向左走一步

ChangePos.row=CurPos.row;

ChangePos.line=CurPos.line-1;

break;

case 4://向上走一步

ChangePos.row=CurPos.row-1;

ChangePos.line=CurPos.line;

break;

}

return ChangePos;

}

Status MazePath(int maze[N][N], SqStack &S, PosType start, PosType end)//找通路

{

PosType curpos;

int curstep;

ElemType e;

curpos=start;//设定"当前位置"为"入口位置

curstep=1;//探索第一步

do{

if(Pass(maze,curpos))//当前位置可通过，即是未曾走到过的通道块

{

MarkFoot(maze,curpos);//留下足迹

e.di=1;

e.ord=curstep;

e.seat=curpos;

Push(S,e);//加入路径

if (curpos.row==end.row && curpos.line==end.line)

return OK;//到达终点（出口）

curpos=NextPos(curpos,1);//下一位置是当前位置的东邻

curstep++;//探索下一步

}

else//当前位置不能通过

{

if(!StackEmpty(S))

{

Pop(S,e);

while (e.di==4 && !StackEmpty(S))

{

MarkFoot(maze,e.seat);//留下不能通过的标记,并退回一步

Pop(S,e);

}

if(e.di<4)

{

e.di++;// 换下一个方向探索

Push(S,e);

curpos=NextPos(e.seat,e.di);//当前位置设为新方向的相邻块

}

}

}

}while(!StackEmpty(S));

return ERROR;//即没有通路

}

void PrintPath(int maze[N][N], SqStack &S, int x, int y, PosType start, PosType end)//显示通路

{

int i,j,m,num;

printf("\n通路路径:\n");

ElemType \*p=S.base;

while(p!=S.top)

{

maze[p->seat.row][p->seat.line]=2;//标记为路径中的点

p++;

}

printf("路径图为：\n");

for(m=0;m<y+2;m++)

printf("%c ",'#');

printf("\n");

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

{

printf("%c ",'#');

for(int j=1;j<y+1;j++)

{

if(maze[i][j]==2)

{

if((i==start.row&&j==start.line)||(i==end.row&&j==end.line))//如果是入口和出口另外标记

{

if(i==start.row&&j==start.line)

printf("S ");

if(i==end.row&&j==end.line)

printf("E ");

}

else

printf("%c ",' ');

}

else

printf("%c ",'#');

}

printf("%c ",'#');

printf("\n");

}

for(i=0;i<y+2;i++)

printf("%c ",'#');

printf("\n");

printf("路径线路为：\n");

SqStack M;

InitStack(M);

ElemType e1;

while(!StackEmpty(S))

{

Pop(S,e1);

Push(M,e1);

}

num=0;

while(!StackEmpty(M))

{

num++;

Pop(M,e1);

printf("(%d,%d)",e1.seat.row,e1.seat.line);

if(num%8==0)

printf("\n");

}

DestroyStack(M);

printf("\n");

printf("路径输出成功！\n");

}

Main.cpp：

#include"Maze.h"

int main()

{

int Maze[N][N],maze[N][N];

SqStack S;

PosType start,end;

start.row=0;

start.line=0;

end.row=0;

end.row=0;

int k,t;//选择器

int x,y;//接收迷宫行数和列数

printf("请按对应按钮执行操作：\n");

printf("1:初始化迷宫并显示迷宫（0表示通路，1表示障碍）\n2:设置迷宫的入口和出口\n3:寻找路线\n4:结束程序\n");

scanf("%d",&k);

while(k!=4)

{

if(k==1)

{

InitMaze(Maze,x,y);

PrintMaze(Maze,x,y);

}

else if(k==2)

{

printf("入口坐标：\nrow:");

scanf("%d",&start.row);

printf("col:");

scanf("%d",&start.line);

printf("出口坐标:\nrow:");

scanf("%d",&end.row);

printf("col:");

scanf("%d",&end.line);

printf("设置成功！\n\n");

}

else if(k==3)

{

if(end.line!=0&&end.row!=0)//判断是否输入口坐标和出口坐标。

{

InitStack(S);

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

{

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

{

maze[i][j]=Maze[i][j];

}

}

if(MazePath(maze,S,start,end))

{

PrintPath(maze,S,x,y,start,end);

}

else

{

printf("该迷宫没有通路！\n");

}

DestroyStack(S);

start.row=0;

start.line=0;

end.row=0;

end.row=0;

printf("请重新初始化迷宫和入口出口坐标。\n\n");

}

else

{

printf("未输入口坐标和出口坐标，请重新选择。\n\n");

}

}

else

{

printf("指令错误，请重新输入\n\n");

}

printf("1:初始化迷宫并显示迷宫（0表示通路，1表示障碍）\n2:设置迷宫的入口和出口\n3:寻找路线\n4:结束程序\n");

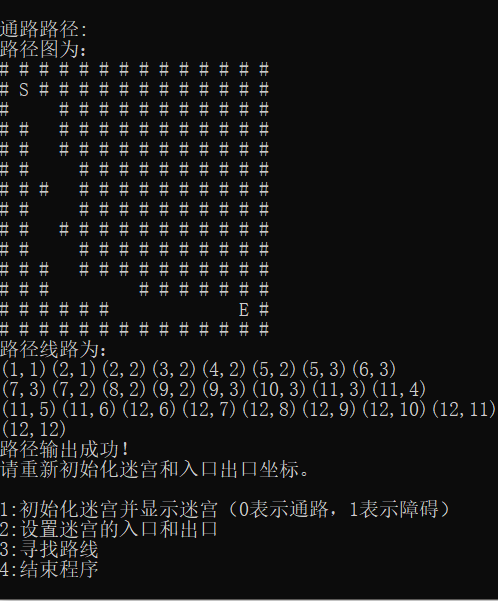
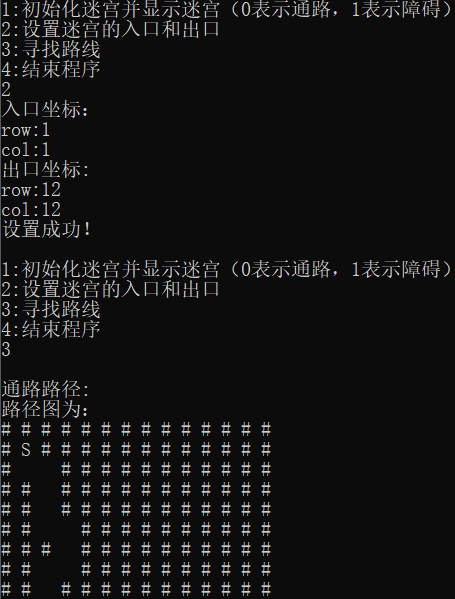
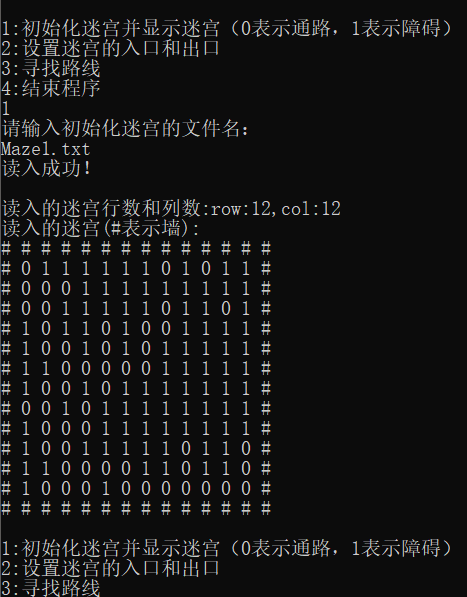
scanf("%d",&k);

}

return 0;

}

**测试数据及结果:**



**算法时间复杂度：**

该算法的时间复杂度为：O(n^4)n为可通点，有四个方向。可通过用深度搜索或者广度搜索的方法将时间复杂度降到O(nlogn)。

**代码行：356行**

## 二叉树的应用(二叉树)

[问题描述]

编程实现二叉树的建立，先序、中序、后序（递归和非递归方法）、层序遍历。求二叉树的高度、宽度，结点数。判断是否为二叉排序树。

[基本要求]

（1） 从文件中读入建树信息，树的节点数目不小于20个，树的高度不小于4。

（2） 采用二叉链表结构。

（3） 至少2组输入数据，分别是二叉排序树和不是二叉排序树，

**数据结构：**

采用二叉链表的数据结构，遍历用到栈跟队列的结构：

typedef char Elem;

typedef struct BiTNode{

Elem data;

struct BiTNode \*lchild;

struct BiTNode \*rchild;

}BiTNode,\*BiTree;//树结点

typedef struct{

char tag;

BiTNode \*ptr;

}StackNode;//后序遍历树的栈结点元素

typedef struct Stack{

StackNode \*Elem;

int top;

int stacksize;

}Stack;//先序中序遍历树的栈节点元素

typedef struct QNode{

BiTree data;

struct QNode \*next;

}QNode,\*QueuePtr;//层次遍历的结点元素

typedef struct LinkQueue{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

**算法思想:**

建树历用递归的方法建树，树的遍历使用递归方法即按顺序输出树的rRL，非递归方法即是用栈和队列实现，在树的实验中已经做到。求树的宽度即使用乒乓算法每次求出一层的节点数，选出其中最大的即为树的宽度。

**源代码实现:**

**BiTree.h：**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define OVERFLOW -2

#define INFEASIBLE -1

#define ERROR 0

#define OK 1

#define INIT\_STACK\_SIZE 100

#define INCREMENT\_SIZE 10

#define N 100

typedef int Status;

typedef char Elem;

typedef struct BiTNode{

Elem data;

struct BiTNode \*lchild;

struct BiTNode \*rchild;

}BiTNode,\*BiTree;

typedef struct{

char tag;

BiTNode \*ptr;

}StackNode;

typedef struct Stack{

StackNode \*Elem;

int top;

int stacksize;

}Stack;

typedef struct QNode{

BiTree data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct LinkQueue{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

Status InitQueue(LinkQueue &Q)

{

Q.front=(QueuePtr)malloc(sizeof(QNode));

if(!Q.front)

exit(OVERFLOW);

Q.rear=Q.front;

Q.front->next=NULL;

}

Status EnQueue(LinkQueue &Q, BiTree &e)

{

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

if(!p)

exit(OVERFLOW);

p->data=e;

Q.rear->next=p;

p->next=NULL;

Q.rear=p;

return OK;

}

Status DeQueue(LinkQueue &Q, BiTree &e)

{

if(Q.rear==Q.front)

{

return ERROR;

}

QNode \*p=Q.front->next;

e=p->data;

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

if(Q.rear==p)

Q.rear=Q.front;

free(p);

return OK;

}

Status EmptyQueue(LinkQueue Q)

{

if(Q.front==Q.rear)

return OK;

else

return ERROR;

}

Status DestroyQueue(LinkQueue &Q)

{

if(Q.front!=NULL)

{

Q.rear=Q.front->next;

free(Q.front);

Q.front=Q.rear;

}

return OK;

}

Status InitStack(Stack &S)

{

S.Elem=(StackNode\*)malloc(sizeof(StackNode)\*INIT\_STACK\_SIZE);

if(!S.Elem)

exit(OVERFLOW);

S.top=0;

S.stacksize=INIT\_STACK\_SIZE;

return OK;

}

Status EmptyStack(Stack S)

{

if(S.top==0&&S.Elem)

return OK;

else

return ERROR;

}

Status Push(Stack &S, StackNode x)

{

if(S.top>=S.stacksize)

{

S.Elem=(StackNode\*)realloc(S.Elem,sizeof(StackNode)\*(S.stacksize+INCREMENT\_SIZE));

if(!S.Elem)

exit(OVERFLOW);

S.stacksize=S.stacksize+INCREMENT\_SIZE;

}

S.Elem[S.top]=x;

S.top++;

return OK;

}

Status Pop(Stack &S, StackNode &x)

{

if(S.top==0)

return ERROR;

x=S.Elem[S.top-1];

S.top--;

return OK;

}

Status DestroyBiTree(BiTree &T)

{

if(T!=NULL)

{

DestroyBiTree(T->lchild);

DestroyBiTree(T->rchild);

free(T);

T=NULL;

}

return OK;

}

Status CreateBiTree(BiTree &T)//按照先序顺序创造二叉树,以空格为空树。

{

char temp;

scanf("%c",&temp);

if(temp!=' ')

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T)

exit(OVERFLOW);

T->data=temp;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

else

T=NULL;

return OK;

}

int Search(char S[],char t)

{

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

{

if(S[i]==t)

return i;

}

return -1;

}

void CrtBT(BiTree &T,char pre[],char ino[],int ps,int is,int n)//通过先序序列跟中序序列建立一颗二叉树

{

if(n==0)

T=NULL;

else

{

int k=Search(ino,pre[ps]);

if(k==-1)

T=NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T)

exit(OVERFLOW);

T->data=pre[ps];

if(k==is)

T->lchild=NULL;

else

{

CrtBT(T->lchild,pre,ino,ps+1,is,k-is);

}

if(k==n-1+is)

T->rchild=NULL;

else

{

CrtBT(T->rchild,pre,ino,ps+1+k-is,k+1,n-k+is-1);

}

}

}

return;

}

Status CreateBiTree\_File(BiTree &T)

{

FILE \*fp;

char filename[81],pre[N],ino[N];

printf("please input the filename of the Tree's infomation:\n");

scanf("%s",filename);

fp=fopen(filename,"r");

if(!fp)

{

printf("Can not find file!\n");

exit(ERROR);

}

fgets(pre,N,fp);

fgets(ino,N,fp);

CrtBT(T,pre,ino,0,0,strlen(pre));

fclose(fp);

return OK;

}

Status PreOrderTraverse(BiTree T,Status (\*Visit)(Elem e))

{

if(T)

{

if(Visit(T->data))

if(PreOrderTraverse(T->lchild,Visit))

if(PreOrderTraverse(T->rchild,Visit))

return OK;

return ERROR;

}

else

return OK;

}

Status InOrderTraverse(BiTree T, Status (\*Visit)(Elem e))

{

if(T)

{

if(InOrderTraverse(T->lchild,Visit))

if(Visit(T->data))

if(InOrderTraverse(T->rchild,Visit))

return OK;

else

return ERROR;

}

return OK;

}

Status PostOrderTraverse(BiTree T, Status (\*Visit)(Elem e))

{

if(T)

{

if(PostOrderTraverse(T->lchild,Visit))

if(PostOrderTraverse(T->rchild,Visit))

if(Visit(T->data))

return OK;

else

return ERROR;

}

return OK;

}

Status Visit(Elem e)

{

printf("%c",e);

return OK;

}

Status LevelOrderTraverse(BiTree T, Status (\*Visit)(Elem e))

{

BiTree p;

LinkQueue Q;

InitQueue(Q);

EnQueue(Q,T);

while(!EmptyQueue(Q))

{

DeQueue(Q,p);

if(p)

{

Visit(p->data);

EnQueue(Q,p->lchild);

EnQueue(Q,p->rchild);

}

}

printf("\n");

return OK;

}

Status PreOrderTraverse\_S(BiTree T,Status (\*Visit)(Elem e))

{

Stack S;

InitStack(S);

BiTree t=T;

StackNode x;

while(t||!EmptyStack(S))

{

while(t)

{

Visit(t->data);

x.ptr=t;

Push(S,x);

t=t->lchild;

}

if(!EmptyStack(S))

{

Pop(S,x);

t=x.ptr->rchild;

}

}

printf("\n");

return OK;

}

Status InOrderTraverse\_S(BiTree T, Status (\*Visit)(Elem e))

{

Stack S;

InitStack(S);

BiTree t=T;

StackNode x;

while(t||!EmptyStack(S))

{

while(t)

{

x.ptr=t;

Push(S,x);

t=t->lchild;

}

if(!EmptyStack(S))

{

Pop(S,x);

t=x.ptr;

Visit(t->data);

t=t->rchild;

}

}

printf("\n");

return OK;

}

Status PostOrderTraverse\_S(BiTree T)

{

Stack S;

InitStack(S);

StackNode x;

BiTree t=T;

do

{

while(t)

{

x.tag='L';

x.ptr=t;

Push(S,x);

t=t->lchild;

}

while(!EmptyStack(S)&&S.Elem[S.top-1].tag=='R')

{

Pop(S,x);

t=x.ptr;

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

}

if(!EmptyStack(S))

{

S.Elem[S.top-1].tag='R';

t=S.Elem[S.top-1].ptr->rchild;

}

}while(!EmptyStack(S));

printf("\n");

return OK;

}

int Height(BiTree T)

{

int Hei\_l,Hei\_r,Hei;

if(T)

{

Hei\_l=Height(T->lchild);

Hei\_r=Height(T->rchild);

Hei=((Hei\_l>Hei\_r)?Hei\_l:Hei\_r)+1;

return Hei;

}

return 0;

}

void CountNodes(BiTree T,int &num)

{

if(T)

{

num++;

CountNodes(T->lchild,num);

CountNodes(T->rchild,num);

}

}

int BiTreeWidth(BiTree T)//求数的宽度即求每一层结点的多少，使用乒乓算法将结点最多的数保留下来即宽度

{

LinkQueue Q1,Q2;

InitQueue(Q1);

InitQueue(Q2);

int num,max=0,d;

if(!T)

{

return 0;

}

EnQueue(Q1,T);

d=Height(T);

BiTree t;

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

{

num=0;

if(!EmptyQueue(Q1)&&EmptyQueue(Q2))

while(!EmptyQueue(Q1))

{

DeQueue(Q1,t);

num++;

if(t->lchild)

EnQueue(Q2,t->lchild);

if(t->rchild)

EnQueue(Q2,t->rchild);

}

else

{

if(!EmptyQueue(Q2)&&EmptyQueue(Q1))

while(!EmptyQueue(Q2))

{

DeQueue(Q2,t);

num++;

if(t->lchild)

EnQueue(Q1,t->lchild);

if(t->rchild)

EnQueue(Q1,t->rchild);

}

}

if(num>max)

max=num;

}

return max;

}

Status InOrder(BiTree T, LinkQueue &Q)//得到中序序列

{

if(T)

{

if(InOrder(T->lchild,Q))

if(EnQueue(Q,T))

if(InOrder(T->rchild,Q))

return OK;

else

return ERROR;

}

return OK;

}

Status IsBiSortTree(BiTree &T)//通过判断树的中序序列来判断是否为二叉排序树

{

if(!T)

return OK;

LinkQueue Q;

InitQueue(Q);

InOrder(T,Q);

BiTree t;

Elem temp;

DeQueue(Q,t);

temp=t->data;

while(!EmptyQueue(Q))

{

DeQueue(Q,t);

if(temp<t->data)

temp=t->data;

else

return ERROR;

}

return OK;

}

Main.cpp：

#include"BiTree.h"

int main()

{

BiTree T=NULL;

int select;

int Hei,num\_nodes,wid;

printf("please enter the number to operate:\n");

printf("1:构建二叉树2:遍历二叉树(递归)3:遍历二叉树(非递归)\n4:求二叉树的高度、宽度、结点数5:判断是否为二叉排序树6:结束程序\n");

scanf("%d",&select);

while(select!=6)

{

if(select==1)

{

if(T)

DestroyBiTree(T);

CreateBiTree\_File(T);

}

else if(select==2)

{

if(!T)

printf("请先构建二叉树!\n");

else

{

printf("PreOrderTraverse:\n");

PreOrderTraverse(T,Visit);

printf("\n");

printf("InOrderTraverse:\n");

InOrderTraverse(T,Visit);

printf("\n");

printf("PostOrderTraverse:\n");

PostOrderTraverse(T,Visit);

printf("\n");

printf("LevelOrderTraverse:\n");

LevelOrderTraverse(T,Visit);

}

}

else if(select==3)

{

if(!T)

printf("请先构建二叉树!\n");

else

{

printf("PreOrderTraverse:\n");

PreOrderTraverse\_S(T,Visit);

printf("InOrderTraverse:\n");

InOrderTraverse\_S(T,Visit);

printf("PostOrderTraverse:\n");

PostOrderTraverse\_S(T);

}

}

else if(select==4)

{

if(!T)

printf("请先构建二叉树!\n");

else

{

num\_nodes=0;

Hei=Height(T);

printf("该二叉树的高度为:%d\n",Hei);

CountNodes(T,num\_nodes);

printf("该二叉树的结点数为:%d\n",num\_nodes);

wid=BiTreeWidth(T);

printf("该二叉树的宽度为:%d\n",wid);

}

}

else if(select==5)

{

if(!T)

printf("请先构建二叉树!\n");

else if(IsBiSortTree(T))

printf("This BiTree is a SearchTree!\n");

else

printf("This BiTree isn't a SearchTree!\n");

}

printf("请选择接下来的操作:\n");

printf("1:构建二叉树2:遍历二叉树(递归)3:遍历二叉树(非递归)\n4:求二叉树的高度、宽度、结点数5:判断是否为二叉排序树6:结束程序\n");

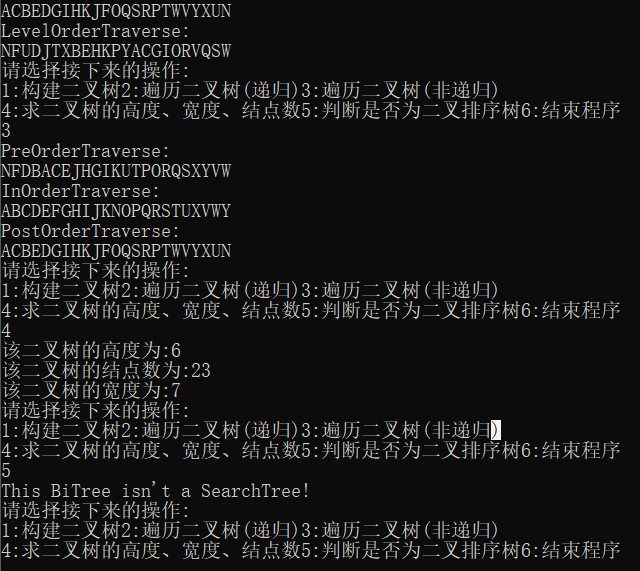
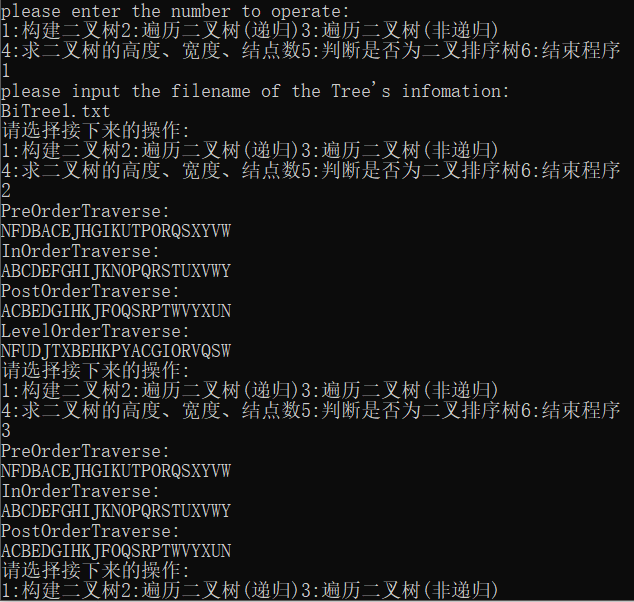
scanf("%d",&select);

}

return 0;

}

**测试数据及结果：**



**算法时间复杂度：**

树的遍历递归时间复杂度为O(n),非递归算法时间复杂度为O(n),关于二叉树的遍历都是只需访问一次结点不过空间占用比较多。

**代码行数：528行**

## Huffman编码与解码（Huffman树）

[问题描述]

对一篇不少于2000字符的英文文章，统计各字符出现的次数，实现Huffman编码，以及对编码结果的解码。

[基本要求]

（1） 输出每个字符出现的次数和编码。

（2） 在Huffman编码后，要将编码表和英文文章编码结果保存到文件中，编码结果必须是二进制形式，即0 1的信息用比特位表示，不能用字符’0’和’1’表示。

（3） 提供读编码文件生成原文件的功能。

**数据结构：**

本题采用静态链表来构建一颗哈夫曼树

typedef struct{

int weight;

int parent;

int lchild;

int rchild;

}HTNode,\*HuffmanTree;//哈夫曼树节点

typedef char\* \*HuffmanCode;

**算法设计思想：**

先将英文文章遍历一遍，统计出字符的种类已经个数及权重，依此构建一颗哈夫曼树。编码则是利用已构建好的哈夫曼树每次从它的叶子节点往上走，左为0，右为1得到所有字符的哈夫曼码，再遍历一遍文章得到编码的01字符串，再使用位运算操作进行压缩，得到一个dat文件即为编码文件。文件的解码即每次读入一个0or1根据这个查找哈夫曼树，原来文章有多少个字就遍历多少次。

**源代码：**

**Huffuman.h:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define OVERFLOW -2

#define INFEASIBLE -1

#define ERROR 0

#define OK 1

#define MAX\_CHARNUM 256

typedef int Status;

typedef struct{

int weight;

int parent;

int lchild;

int rchild;

}HTNode,\*HuffmanTree;

typedef char\* \*HuffmanCode;

void Select(HuffmanTree HT,int n,int &l,int &r)

{

int i,min;

min=INT\_MAX;

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

{

if(HT[i].parent==0&&HT[i].weight<=min)

{

min=HT[i].weight;

l=i;

}

}

min=INT\_MAX;

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

{

if(i==l)

continue;

if(HT[i].parent==0&&HT[i].weight<=min)

{

min=HT[i].weight;

r=i;

}

}

}

void CreateHuffmanTree(HuffmanTree &HT, int \*w, int n)//\*w为权重，n为有个权重

{

int p,i,j,m,lc,rc;

if(n<=1)

return;//n<=1时无法生成一颗huffmantree。

m=2\*n-1;

if(HT)

free(HT);

HT=(HuffmanTree)malloc(sizeof(HTNode)\*(m+1));//HT[0]不放节点用来当头节点

for(p=1,i=1,j=1;i<=n;++i,++p,++j)//第一步生成所有权重的叶子节点

{

HT[p].weight=w[j];

HT[p].parent=0;

HT[p].lchild=0;

HT[p].rchild=0;

}

for(;i<=m;i++)//按照Huffman的算法依次挑选出最小权重的结点组成树

{

Select(HT,i-1,lc,rc);//从HT[1..i-1]中找出当前权数最小且parent=0的两个结点位置分别为lc,rc。

HT[lc].parent=i;

HT[rc].parent=i;

HT[i].parent=0;

HT[i].lchild=lc;

HT[i].rchild=rc;

HT[i].weight=HT[lc].weight+HT[rc].weight;

}

HT[0].lchild=i-1;

HT[0].weight=-1;

}

Status Get\_HuffmanCode(HuffmanTree &HT, HuffmanCode &HC, int n)

{

if(HC)

free(HC);

HC=(HuffmanCode)malloc(sizeof(char \*)\*(n+1));//0号位置不用

if(!HC)

exit(OVERFLOW);

char \*cd=(char\*)malloc(sizeof(char)\*n);

if(!cd)

exit(OVERFLOW);

cd[n-1]='\0';

int start,c,f;//存放位置、孩子结点位置、父节点位置

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

{

start=n-1;

for(c=i,f=HT[i].parent;f!=0; c=f,f=HT[f].parent)

{

if(HT[f].lchild==c)

cd[--start]='0';

else

cd[--start]='1';

}

HC[i] = (char\*)malloc(sizeof(n-start)\*sizeof(char));

strcpy(HC[i],&cd[start]);

}

free(cd);

return OK;

}

int Search(char \*a,char c)

{

for(int i=1;a[i]!='\0';i++)

{

if(a[i]==c)

return i;

}

return 0;

}

int \*Getweight(FILE \*fp,char \*(\*s\_type))

{

int \*w,num=0,i=1,j;

w=(int\*)malloc(sizeof(int)\*MAX\_CHARNUM);

char c;

for(j=1;j<MAX\_CHARNUM;j++)

{

(\*s\_type)[j]='\0';

w[j]=0;

}

w[0]=0;

while(!feof(fp))

{

c=fgetc(fp);

if(c==EOF)

break;

num++;

if(j=Search(\*s\_type,c))

{

w[j]++;

}

else

{

(\*s\_type)[i]=c;

w[i]++;

i++;

w[0]++;

}

}

return w;

}

void writeBit(char \*ch,FILE \*fp)

{

int i;

int j;

int count = 0;

int num; //存储字符需要的多少个字节

int left; //字符串剩余不足8位的个数

char \*p = NULL;

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

{

count++; //统计字符个数

}

num = count / 8;

left = count % 8;

if (left == 0) //如果left为0，那么num个字节刚好能够存放,否则需要num + 1字节

{

p = (char \*)malloc(sizeof(char) \* num);

memset(p, 0, num);

}

else

{

p = (char \*)malloc(sizeof(char) \* (num + 1));

memset(p, 0, num + 1);

}

j = -1;

for (i = 0; i < count; i++)//位运算，每8个字符以2进制的形式储存在一个字符中

{

if (i % 8 == 0)

{

j++;

}

p[j] <<= 1;

ch[i] -= '0';

p[j] |= ch[i];

}

if (left != 0) //如果left不为0，需要把剩余的几个位向左边靠拢

{

p[j] <<= 8 - left;

fwrite(&count, sizeof(count), 1, fp);

fwrite(p,1, num + 1,fp);

}

else

{

fwrite(&count, sizeof(count), 1, fp);

fwrite(p, 1, num , fp);

}

}

Status CreateCodeFile(HuffmanCode HC,char \*s,char \*filename,int n)

{

FILE \*fp,\*ftxt;

char c,\*temp=(char\*)malloc(sizeof(char)\*INT\_MAX);

strcpy(temp,"");

int p;

fp=fopen("CodeList.txt","w");

if(!fp)

{

printf("open failed!\n");

exit(ERROR);

}

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

{

fprintf(fp,"%c%s\n",s[i],HC[i]);

}

fclose(fp);

fp=fopen("HCCode.data","wb");

ftxt=fopen(filename,"r");

if(!ftxt||!fp)

{

printf("open failed!\n");

exit(ERROR);

}

while(!feof(ftxt))

{

c=fgetc(ftxt);

if(c==EOF)

break;

p=Search(s,c);

strcat(temp,HC[p]);

}

writeBit(temp,fp);

fclose(fp);

fclose(ftxt);

return OK;

}

char \*readBit(FILE \*fp)

{

int count;

int num;

int left;

int i,j,k=0;

unsigned char flag=128;//即10000000

char \*p = NULL,\*temp;

fread(&count, sizeof(count), 1, fp);

temp=(char\*)malloc(sizeof(char)\*(count+1));

num=count/8;

left=count%8;

if (left == 0)

{

p = (char \*)malloc(sizeof(char) \* num);

fread(p,1,num,fp);

}

else

{

p = (char \*)malloc(sizeof(char) \* (num + 1));

fread(p, 1, num + 1, fp);

}

j = -1;

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

{

if(i % 8 == 0)

{

j++;

flag = 128;

}

if((p[j] & flag))//通过改变flag字符2进制1的位置判读，一个字节哪个位是1，哪个位是0

{

temp[k++]='1';

flag /= 2;

}

else

{

temp[k++]='0';

flag /= 2;

}

}

temp[k]='\0';

return temp;

}

Status Decoding(FILE \*fp,HuffmanTree HT,HuffmanCode HC,char \*s,int n)

{

int j;

char \*temp,c,filename[81],back[20];

FILE \*out;

printf("please input the filename of the decoding file:\n");

scanf("%s",filename);

out=fopen(filename,"w");

if(!out)

{

printf("open failed!\n");

exit(ERROR);

}

temp=readBit(fp);

int i=0,k,m;

c=temp[i++];

while(c!='\0')

{

m=0;

k=2\*n-1;

while(HT[k].lchild!=0&&HT[k].rchild!=0)

{

back[m++]=c;

if(c=='0')

k=HT[k].lchild;

else if(c=='1')

k=HT[k].rchild;

c=temp[i++];

}

back[m]='\0';

for(j=1;j<=n;j++)

{

if(strcmp(back,HC[j])==0)

{

printf("%c",s[j]);

fputc(s[j],out);

break;

}

}

}

fclose(out);

}

Main.cpp:

#include"Huffman.h"

int main()

{

HuffmanTree HT=NULL;

HuffmanCode HC=NULL;

FILE \*f;

int \*w,n,k;

char filename[81],\*s\_type=(char\*)malloc(sizeof(char)\*MAX\_CHARNUM);

printf("请按相应按键进行操作：\n");

printf("1:读入文件构建编码并输出字符及其对应编码\n2:将编码表及文章编码保存于文件\n3:解码并输出文章\n0:结束程序\n");

scanf("%d",&k);

while(k!=0)

{

if(k==1)

{

printf("please intput the filename of the essay:\n");

scanf("%s",filename);

f=fopen(filename,"r");

if(!f)

{

printf("open failed!\n");

exit(ERROR);

}

w=Getweight(f,&s\_type);

fclose(f);

n=w[0];

printf("%d\n",n);

CreateHuffmanTree(HT,w,n);

Get\_HuffmanCode(HT,HC,n);

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

{

putchar(s\_type[i]);

printf("出现次数:%d,编码:%s\n",w[i],HC[i]);

}

printf("\n");

}

else if(k==2)

{

if(HC!=NULL)

{

CreateCodeFile(HC,s\_type,filename,n);

printf("编码表已保存于CodeList.txt,文章编码保存于HCCode.data。\n");

printf("\n");

}

else

printf("请先读入文件并编码。\n\n");

}

else if(k==3)

{

if(HC!=NULL)

{

f=fopen("HCCode.data","rb");

if(!f)

{

printf("open failed!\n");

exit(ERROR);

}

Decoding(f,HT,HC,s\_type,n);

printf("\n");

fclose(f);

}

else

printf("请先读入文件并编码。\n\n");

}

printf("1:读入文件构建编码并输出字符及其对应编码\n2:将编码表及文章编码保存于文件\n3:解码并输出文章\n0:结束程序\n");

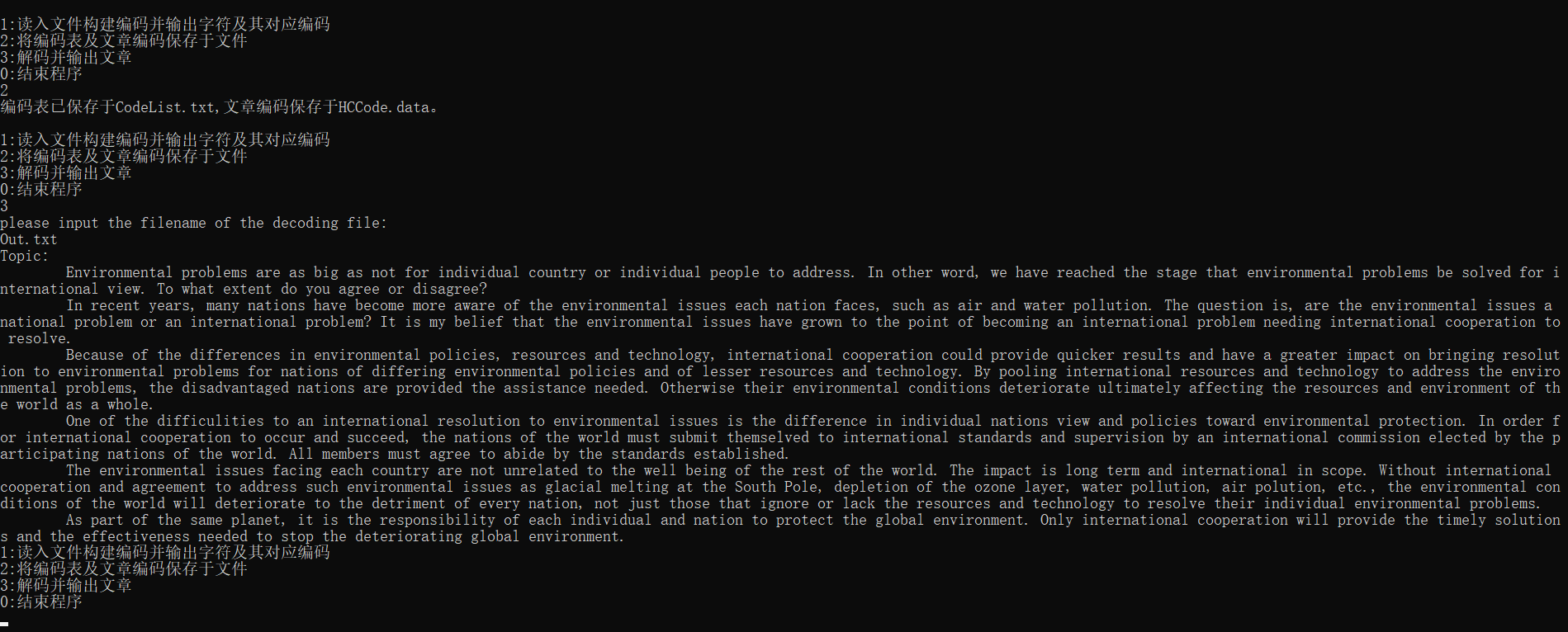
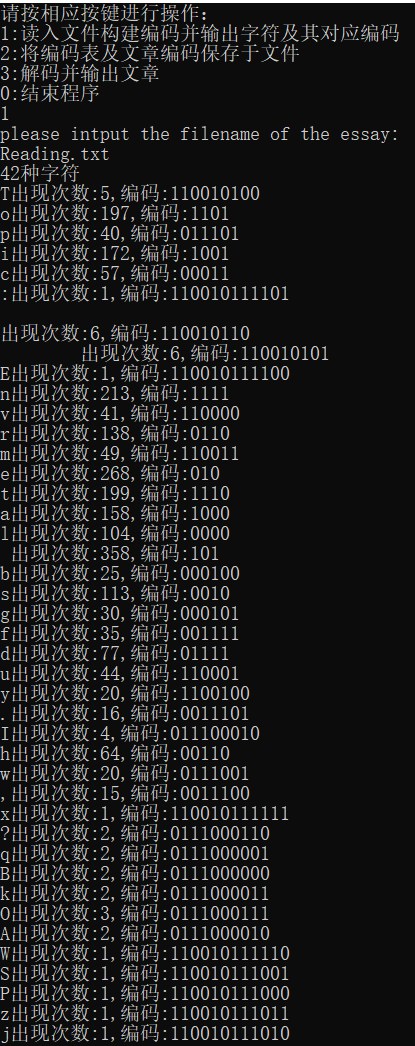
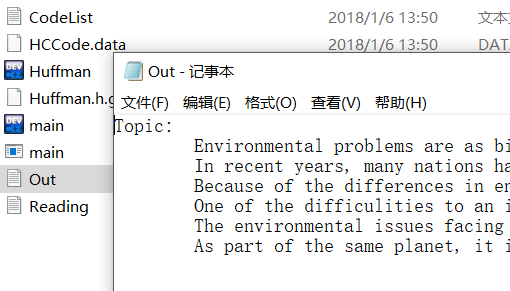
scanf("%d",&k);

}

return 0;

}

**测试数据及结果：**



**算法时间复杂度：**

建树过程是O(logn),编码过程：O(n),解码过程：O(nlogn)

**代码行数：378行**

## 无线传感器网络(图)

[问题描述]

在一个直角坐标（设为100\*100）平面中，随机分布n无线传感器结点，汇聚点为原点（0，0），要求将每个无线传感器结点的信息传输到汇聚点，传输可以通过多跳方式实现，设无线传感器结点最大无线通信距离为10，传输能耗与距离平方成正比，传输时间与距离成正比。

问题一：建立每个结点传输到汇聚点的最短时间通道，并给无法实现传输的结点，将其排除。

问题二：建立整个网络传输到汇聚点的平均能耗最小网络结构。

[基本要求]

输入格式： 输入的第一行包含一个正整数n，表示无线传感器结点数量。结点使用1, 2, 3, ……n依次标号。 接下来n行，每行包含三个整数ni, xi, yi，其中xi, yi表示第ni个结点的坐标，要求从文本文件中输入。

输出格式：

问题一：输出每个结点到汇聚点的最短时间通道，包括最短时间和经过结点。

问题二：输出整个网络到汇聚点的平均能耗最小网络结构，包括平均最小能耗和连接方式。

**数据结构：**

本题采用的是图的邻接矩阵结构：

typedef float VRType;/\*边上的权值类型 \*/

typedef int VertexType;/\*顶点类型\*/

typedef int Status;

typedef struct AreCell{

VRType w;

}ArcCell,AdjMatrix[MAXVEX][MAXVEX];

typedef struct{

VertexType vexs[MAXVEX];

AdjMatrix arcs;

int vexnum,arcnum;

}MGraph;

typedef struct {

VertexType adjvex;

VRType lowcost;

}money,closedge[MAXVEX];

typedef struct{

VertexType i;

VertexType j;

}Plan;//用于求最低能耗路径

**算法思想：**

先用一个二维数组将所有结点的位置存储。然后根据位置信息建立图的结构，有多少个节点即有多少个顶点，距离即是它们之间弧的权值，将到源点即(0,0)距离小于10的节点全部跟源点相连，第二次遍历所有节点将节点之间距离小于10的节点全部相连，排除无法连入节点网络的节点选出来排除。问题一即利用FLOYD算法求出每个点到汇聚点的最短路径即为最短时间。问题二即用PRIM算法求最小生成树即为最少耗能路径。

**源代码：**

WSN.h:

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

using namespace std;

#define OK 1

#define ERROR 0

#define MAXVEX 100/\*最大顶点数\*/

typedef float VRType;/\*边上的权值类型 \*/

typedef int VertexType;/\*顶点类型\*/

typedef int Status;

typedef struct AreCell{

VRType w;

}ArcCell,AdjMatrix[MAXVEX][MAXVEX];

typedef struct{

VertexType vexs[MAXVEX];

AdjMatrix arcs;

int vexnum,arcnum;

}MGraph;

typedef struct {

VertexType adjvex;

VRType lowcost;

}money,closedge[MAXVEX];

typedef struct{

VertexType i;

VertexType j;

}Plan;//用于求最低能耗路径

Status CreateGraph(MGraph \*\*G)

{

FILE \*fp;

char filename[81];

cout<<"请输入包含传感器结点信息的文件名："<<endl;

cin>>filename;

fp=fopen(filename,"r");

if(!fp)

{

cout<<"文件打开失败！"<<endl;

exit(ERROR);

}

int n,ni,xi,yi;//记录结点信息

int i,j,k\_1,l\_1,k\_2,l\_2;

int Position[101][101];//模拟结点所在的平面位置

for(i=0;i<101;i++)

{

for(j=0;j<101;j++)

{

Position[i][j]=0;

}

}

fscanf(fp,"%d",&n);

if(feof(fp))

return ERROR;

while(!feof(fp))

{

fscanf(fp,"%d%d%d",&ni,&xi,&yi);

Position[xi][yi]=ni;

}

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

(\*G)->vexnum=n+1;

for(i=0;i<(\*G)->vexnum;i++)//构造图

{

(\*G)->vexs[i]=i;

}

(\*G)->arcnum=0;

for(i=0;i<(\*G)->vexnum;i++)

{

for(j=0;j<(\*G)->vexnum;j++)

{

(\*G)->arcs[i][j].w=INFINITY;

}

}

for(i=0;i<11;i++)//将到汇聚点距离小于等于10的结点与汇聚点连通，其权值即为他们之间的距离。

{

for(j=0;j<11;j++)

{

if(Position[i][j]!=0&&(i\*i)+(j\*j)<=100)

{

ni=Position[i][j];

(\*G)->arcs[0][ni].w=sqrt((i\*i)+(j\*j));

(\*G)->arcs[ni][0].w=sqrt((i\*i)+(j\*j));

(\*G)->arcnum++;

}

}

}

for(i=0;i<(\*G)->vexnum;i++)//将距离小于等于10的结点与结点连通，其权值即为他们之间的距离。从而构成一个无向网

{

for(k\_1=0;k\_1<101;k\_1++)

{

for(l\_1=0;l\_1<101;l\_1++)

{

if(Position[k\_1][l\_1]==i)

{

break;

}

}

if(Position[k\_1][l\_1]==i)

{

break;

}

}

for(j=i+1;j<(\*G)->vexnum;j++)

{

for(k\_2=0;k\_2<101;k\_2++)

{

for(l\_2=0;l\_2<101;l\_2++)

{

if(Position[k\_2][l\_2]==j)

{

break;

}

}

if(Position[k\_2][l\_2]==j)

{

break;

}

}

if((k\_1-k\_2)\*(k\_1-k\_2)+(l\_1-l\_2)\*(l\_1-l\_2)<=100)

{

(\*G)->arcs[i][j].w=sqrt((k\_1-k\_2)\*(k\_1-k\_2)+(l\_1-l\_2)\*(l\_1-l\_2));

(\*G)->arcs[j][i].w=sqrt((k\_1-k\_2)\*(k\_1-k\_2)+(l\_1-l\_2)\*(l\_1-l\_2));

(\*G)->arcnum++;

}

}

}

int num=0;//记录无法实现网络传输的结点个数

for(i=0;i<(\*G)->vexnum;i++)

{

for(j=0;j<(\*G)->vexnum;j++)

{

if((\*G)->arcs[i][j].w!=INFINITY)

break;

}

if(j>=(\*G)->vexnum)

{

cout<<(\*G)->vexs[i]<<"号结点无法实现网络传输."<<endl;

num++;

}

}

if(num==0)

cout<<"所有结点均能实现网络传输。"<<endl;

/\* cout<<"当前结点分布如下："<<endl;

for(i=0;i<101;i++)

{

for(j=0;j<101;j++)

{

cout<<Position[i][j];

}

cout<<endl;

}\*/

fclose(fp);

return OK;

}

Status ShortestTimePath(MGraph G)//利用FLOYD算法求出每个点到汇聚点的最短路径即为最短时间。

{

int i,j,k;

float D[MAXVEX][MAXVEX];

bool Path[MAXVEX][MAXVEX][MAXVEX];

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

{

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

{

if(i!=j)

D[i][j]=G.arcs[i][j].w;

else

D[i][j]=0;

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

{

Path[i][j][k]=false;

}

if(D[i][j]<INFINITY)

{

Path[i][j][i]=true;

Path[i][j][j]=true;

}

}

}

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

{

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

{

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

{

if(D[i][k]+D[k][j]<D[i][j]&&D[i][k]!=INFINITY&&D[k][j]!=INFINITY)

{

D[i][j]=D[i][k]+D[k][j];

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

{

Path[i][j][u]=Path[i][k][u]||Path[k][j][u];

}

}

}

}

}

float t;

cout<<"请输入单位距离的传输时间t:"<<endl;

cin>>t;

for(i=1;i<G.vexnum;i++)

{

if(D[i][0]!=INFINITY)

{

cout<<"从"<<i<<"号结点到汇聚点最短时间为:"<<t\*D[i][0]<<"s."<<endl;

cout<<"路径结点为:"<<endl;

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

{

if(Path[i][0][j])

cout<<j<<" ";

}

cout<<endl<<endl;

}

}

return OK;

}

int LocateVex(MGraph &G, VertexType u)

{

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

{

if(G.vexs[i]==u)

return i;

}

return -1;

}

int Mini(int n,closedge &m)

{

int p=0;

float min=INFINITY;

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

{

if(m[i].lowcost<min&&m[i].lowcost!=0)

{

min=m[i].lowcost;

p=i;

}

}

if(p!=0)

return p;

else

return -1;

}

void MinEnergyComsuption(MGraph &MG,VertexType u)//用PRIM算法求最小生成树即为最少耗能路径

{

MGraph G;

G.arcnum=MG.arcnum;

G.vexnum=MG.vexnum;

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

{

G.vexs[i]=MG.vexs[i];

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

{

G.arcs[i][h].w=MG.arcs[i][h].w\*MG.arcs[i][h].w;

}

}

closedge cost;

Plan \*s=(Plan\*)malloc(sizeof(Plan)\*G.vexnum);

int k=LocateVex(G,u);

int min\_energy=0,position=0;

float en\_comsuption;

cout<<"请输入单位距离平方的传输能耗:"<<endl;

cin>>en\_comsuption;

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

{

if(j!=k)

{

cost[j].adjvex=u;

cost[j].lowcost=G.arcs[k][j].w;

}

}

cost[k].lowcost=0;

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

{

k=Mini(G.vexnum,cost);

if(k==-1)

break;

min\_energy+=cost[k].lowcost\*en\_comsuption;

cost[k].lowcost=0;

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

{

if(G.arcs[k][j].w<cost[j].lowcost)

{

cost[j].adjvex=G.vexs[k];

cost[j].lowcost=G.arcs[k][j].w;

}

}

s[position].i=cost[k].adjvex;

s[position].j=G.vexs[k];

position++;

}

cout<<"平均最小能耗为:"<<min\_energy<<endl;

cout<<"建立以下"<<position-1<<"条连接方式："<<endl;

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

{

if(i!=position-1)

cout<<"结点"<<s[i].i<<"与结点"<<s[i].j<<"、"<<endl;

else

cout<<"结点"<<s[i].i<<"与结点"<<s[i].j<<"。"<<endl;

}

free(s);

}

Status PrintMtrix(MGraph &G)

{

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

{

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

{

if(G.arcs[i][j].w!=INFINITY)

cout<<G.arcs[i][j].w<<" ";

else

cout<<"∞ ";

}

cout<<endl;

}

return OK;

}

Main.cpp:

#include"WSN.h"

int main()

{

MGraph \*G=NULL;

int select;

cout<<"请输入相应按键进行操作：\n";

cout<<"1:创建结点信息，排除无法实现传输的结点。\n2:输出每个结点到汇聚点的最短时间通道。"<<endl;

cout<<"3:输出整个网络到汇聚点的平均能耗最小网络结构。\n0:结束程序"<<endl;

cin>>select;

while(select!=0)

{

if(select==1)

{

if(G)

free(G);

CreateGraph(&G);

cout<<endl;

}

else if(select==2)

{

if(G)

ShortestTimePath(\*G);

else

cout<<"请先创建结点信息。"<<endl;

cout<<endl;

}

else if(select==3)

{

if(G)

MinEnergyComsuption(\*G,0);

else

cout<<"请先创建结点信息。"<<endl;

cout<<endl;

}

cout<<"1:创建结点信息，排除无法实现传输的结点。\n2:输出每个结点到汇聚点的最短时间通道。"<<endl;

cout<<"3:输出整个网络到汇聚点的平均能耗最小网络结构。\n0:结束程序"<<endl;

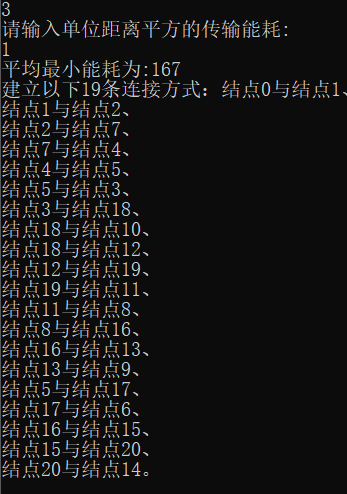
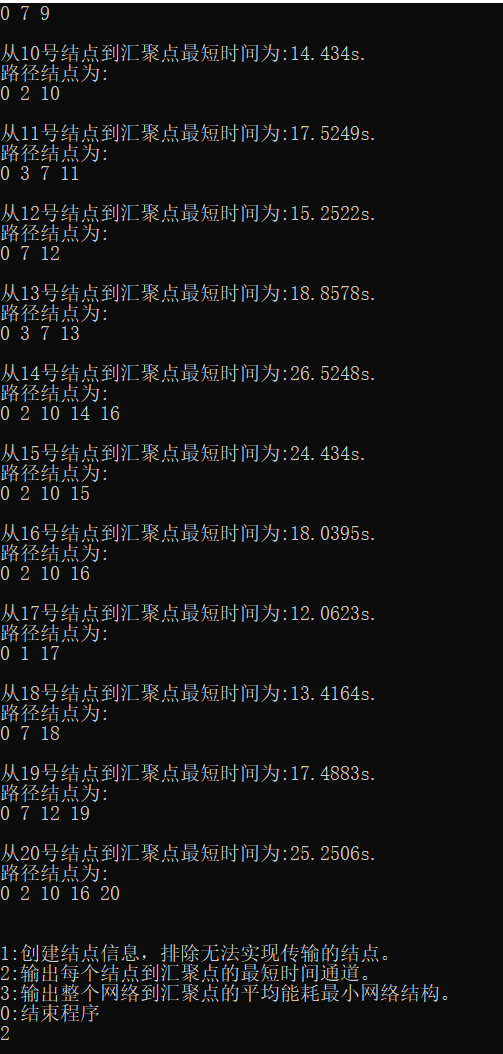
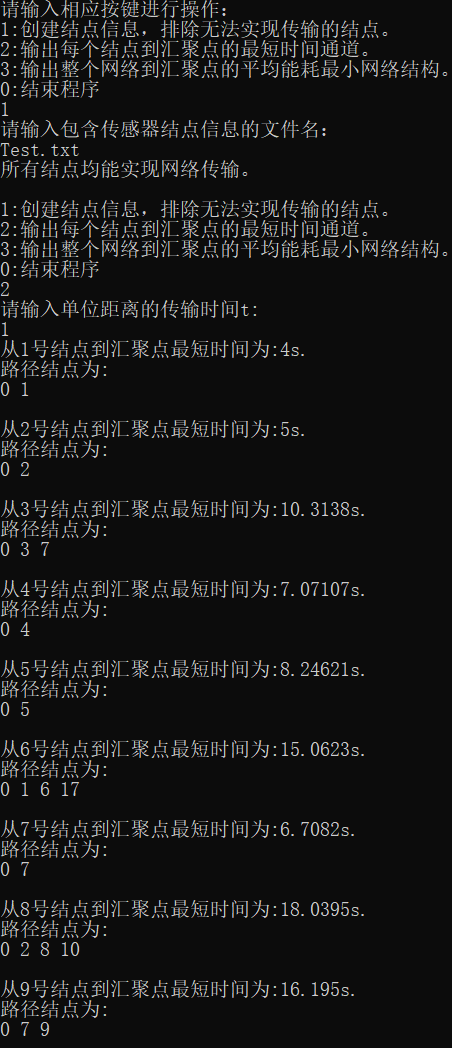
cin>>select;

}

return 0;

}

**测试数据及结果：**



**时间复杂度：**

创建图的时间复杂度为O(n^2)需要从一个顶点出发判断到每个顶点的距离，符合要求的建立连接。问题一求最短路径的时间复杂度为O(n),最小生成树也是O(n)。

**代码行数：353**

## 排序算法比较

[问题描述]

利用随机函数产生10个样本的20000个随机整数（其中之一已经是正序，之一是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。

[基本要求]

（1） 原始数据存在文件中，用相同样本对不同算法进行测试；

（2） 屏幕显示每种排序算法对不同样本所花的时间；

**数据结构：**

本题比较各个排序算法的效率除了基数排序均采用顺序表的结构，基数排序使用静态链表。

typedef int KeyType;

typedef int KeysType;

typedef int Status;

typedef struct{

KeyType key;

}RcdType,RcdTypes[MAXSIZE+1];

typedef struct{

RcdType r[MAXSIZE+1];//r[0]闲置

int length;

}SqList;

typedef struct{

KeysType keys[MAX\_NUM\_OF\_KEY];//关键字

int next;

}SLCell,SLCells[MAX\_SPACE+1];//静态链表的结点类型

typedef struct{

SLCell r[MAX\_SPACE+1];//静态链表的可利用空间，r[0]为头结点

int keynum;//记录当前关键字个数

int recnum;//静态链表的当前长度

}SLList;//静态链表类型

typedef int ArrType[RADIX];//用于基数排序

typedef SqList HeapType;//堆排序

**算法思想：**

冒泡排序，每一次将最大（小）的值排到最后一位，直到整个序列有序。简单选择排序每次从无序序列中选取最大或最小值，将其与无需序列最后一位交换位置。归并排序利用分而治之的思想;每个递归过程涉及三个步骤：第一,分解:把待排序的n个元素的序列分解成两个子序列,每个子序列包括n/2个元素。第二,治理:对每个子序列分别调用归并排序MergeSort,进行递归操作。第三，合并两个排好序的子序列,生成排序结果。通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。堆分为最大堆和最小堆，其实就是完全二叉树。最大堆要求节点的元素都要大于其孩子，最小堆要求节点元素都小于其左右孩子，两者对左右孩子的大小关系不做任何要求，其实很好理解。有了上面的定义，我们可以得知，处于最大堆的根节点的元素一定是这个堆中的最大值。其实我们的堆排序算法就是抓住了堆的这一特点，每次都取堆顶的元素，将其放在序列最后面，然后将剩余的元素重新调整为最大堆，依次类推，最终得到排序的序列。n个记录的关键字进行排序，每个关键字看成是一个d元组:ki=(ki1, ki2,..., kid)其中c0 <=kij <=cr-1 ( 1 <=i <=n, 1 <=j <=d )排序时先按kid 的值，从小到大将记录分到r(称为基数)个盒子中，再依次收集;然后按kid-1的值再这样作。直至按ki1分配和收集序列为止，排序结束。希尔排序实际上是插入排序的一种改良通过多次插入排序实现每次插入排序有一个间隔，间隔取的合适可以达到最快。

**源程序：**

Sort2.h:

#include<iostream>

#include<fstream>

using namespace std;

#include<time.h>

#include<string.h>

#include<stdlib.h>

#include<stdio.h>

#define OK 1

#define ERROR 0

#define MAXSIZE 20000

#define MAX\_SPACE 20000

#define MAX\_NUM\_OF\_KEY 8 //关键字项数的最大值，基数排序

#define RADIX 10

int d[]={16383,8191,4095,2047,1023,511,255,127,63,31,15,7,3,1};

typedef int KeyType;

typedef int KeysType;

typedef int Status;

typedef struct{

KeyType key;

}RcdType,RcdTypes[MAXSIZE+1];

typedef struct{

RcdType r[MAXSIZE+1];//r[0]闲置

int length;

}SqList;

typedef struct{

KeysType keys[MAX\_NUM\_OF\_KEY];//关键字

int next;

}SLCell,SLCells[MAX\_SPACE+1];//静态链表的结点类型

typedef struct{

SLCell r[MAX\_SPACE+1];//静态链表的可利用空间，r[0]为头结点

int keynum;//记录当前关键字个数

int recnum;//静态链表的当前长度

}SLList;//静态链表类型

typedef int ArrType[RADIX];

typedef SqList HeapType;

Status InitList(SqList &L,string a)

{

fstream in;

char filename[81];

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

{

filename[i]=a[i];

}

filename[a.length()]='\0';

in.open(filename,ios::in);

if(!in)

{

cout<<"文件打开失败."<<endl;

exit(0);

}

L.r[0].key=0;

for(int i=1;i<=20000;i++)

{

in>>L.r[i].key;

}

L.length=20000;

in.close();

return OK;

}

void ShellInsert(SqList &L, int dk)

{

int i,j;

for(i=dk+1;i<=L.length;++i)

{

if(L.r[i].key<L.r[i-dk].key)

{

L.r[0]=L.r[i];

for(j=i-dk;j>0&&(L.r[0].key<L.r[j].key);j=j-dk)

{

L.r[j+dk]=L.r[j];

}

L.r[j+dk]=L.r[0];

}

}

}

void ShellSort(SqList &L, int dlta[], int t)

{

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

{

ShellInsert(L,dlta[k]);

}

}

void InsertSort(SqList &L)

{

int n=L.length;

int i,j;

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

{

if(L.r[i].key<L.r[i-1].key)

L.r[0]=L.r[i];

for(j=i-1;(L.r[j].key>L.r[0].key);--j)

{

L.r[j+1]=L.r[j];

}

L.r[j+1]=L.r[0];

}

}

int Patition(SqList &L,int low, int high)

{

KeyType pivotkey;

L.r[0].key=L.r[low].key;//用表的第一个空间记录作枢轴记录

pivotkey=L.r[low].key;

while(low<high)

{

while(low<high&&L.r[high].key>=pivotkey) --high;

L.r[low].key=L.r[high].key;

while(low<high&&L.r[low].key<=pivotkey) ++low;

L.r[high].key=L.r[low].key;

}

L.r[low].key=L.r[0].key;

return low;

}

void QSort(SqList &L,int low, int high)

{

int pivotloc;

if(low<high)

{

pivotloc=Patition(L,low,high);

QSort(L,low,pivotloc-1);

QSort(L,pivotloc+1,high);

}

}

Status QuickSort(SqList &L)

{

QSort(L,1,L.length);

return OK;

}

Status BubbleSort(SqList &L)

{

int LastPos,temp,j;

LastPos=L.length;

j=LastPos;

while(j>1)

{

LastPos=1;

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

{

if(L.r[i].key>L.r[i+1].key)

{

temp=L.r[i].key;

L.r[i].key=L.r[i+1].key;

L.r[i+1].key=temp;

LastPos=i;

}

}

j=LastPos;

}

return OK;

}

Status SelectSort(SqList &L)

{

int min;

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

{

min=i;

for(int j=i+1;j<=L.length;++j)

{

if(L.r[j].key<L.r[min].key)

{

min=j;

}

}

if(i!=min)

{

L.r[0].key=L.r[i].key;

L.r[i].key=L.r[min].key;

L.r[min].key=L.r[0].key;

}

}

return OK;

}

void Merge(RcdType A[],int p,int q,int r)

{

int i,j,k;

int n1=q-p+1;

int n2=r-q;

RcdType \*L=new RcdType[n1+1]; //开辟临时存储空间

RcdType \*R=new RcdType[n2+1];

for(i=0;i<n1;i++)

L[i]=A[i+p]; //数组下标从0开始时，这里为i+p

for(j=0;j<n2;j++)

R[j]=A[j+q+1]; //数组下标从0开始时，这里为就j+q+1

L[n1].key=INT\_MAX; //"哨兵"设置为整数的最大值，INT\_MAX包含在limits.h头文件中

R[n2].key=INT\_MAX;

i=0;

j=0;

for(k=p;k<=r;k++) //开始合并

{

if(L[i].key<=R[j].key)

A[k].key=L[i++].key;

else

A[k].key=R[j++].key;

}

}

void MSort(RcdType A[],int p,int r)

{

if(p<r)

{

int q=(p+r)/2;

MSort(A,p,q);

MSort(A,q+1,r);

Merge(A,p,q,r);

}

}

Status MergeSort(SqList &L)

{

MSort(L.r,1,L.length);

}

void HeapAdjust(HeapType &H, int s, int m)

{

RcdType rc=H.r[s];

int i;

for(i=2\*s; i<=m; i\*=2)

{

if(i<m&&H.r[i].key<H.r[i+1].key)

++i;

if(!(rc.key<H.r[i].key))

break;

H.r[s]=H.r[i];

s=i;

}

H.r[s]=rc;

}

Status HeapSort(SqList &L)

{

RcdType temp;

int i;

for(i=L.length/2;i>0;--i)

{

HeapAdjust(L,i,L.length);

}

for(i=L.length;i>1;--i)

{

temp=L.r[1];

L.r[1]=L.r[i];

L.r[i]=temp;

HeapAdjust(L,1,i-1);

}

return OK;

}

Status ListTraverse(SqList L)

{

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

{

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

}

cout<<endl;

return OK;

}

Status InitSLList(SLList &L,string a)

{

int key;

fstream in;

char filename[81];

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

{

filename[i]=a[i];

}

filename[a.length()]='\0';

in.open(filename,ios::in);

if(!in)

{

cout<<"文件打开失败."<<endl;

exit(0);

}

L.keynum=3;

for(int i=1;i<=20000;i++)

{

in>>key;

for(int j=L.keynum-1;j>=0;--j)

{

L.r[i].keys[j]=key%10;

key=key/10;

}

}

L.recnum=20000;

return OK;

}

void Distribute(SLCell \*r, int i, ArrType &f, ArrType &e)

{

int p,j;

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

{

f[j]=0;

e[j]=0;

}

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

{

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

if(!f[j])

{

f[j]=p;

e[j]=p;

}

else

{

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

e[j]=p;

}

}

}

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

{

int j,t,p;

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

;

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

t=e[j];

while(j<RADIX)

{

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

;

if(f[j]&&j<RADIX)

{

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

t=e[j];

}

}

r[t].next=0;

}

void RadixSort(SLList &L)//基数排序

{

int i,p;

ArrType f;

ArrType e;

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

{

L.r[i].next=i+1;

}

L.r[L.recnum].next=0;

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

{

Distribute(L.r,i,f,e);

Collect(L.r,i,f,e);

}

}

void PrintRadixSort(SLList &L)

{

int p;

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

{

for(int j=L.keynum-1;j>=0;j--)

{

cout<<L.r[p].keys[j];

}

cout<<" ";

}

cout<<endl;

}

Main.cpp:

#include"Sort2.h"

int main()

{

int k,select,select1;

SqList L;

SLList SLL;

clock\_t start,finish;//用于计数排序时间

double t0,t1,t2,t3,t4,t5,t6,t7,t8,t9;

cout<<"请选择按提示选择操作："<<endl;

cout<<"1:显示每种排序算法对不同样本所花的时间\n2:用相同样本对不同算法进行测试\n0:结束程序"<<endl;

cin>>select1 ;

while(select1!=0)

{

if(select1==1)

{

cout<<"1:直接插入排序"<<endl<<"2:希尔排序"<<endl<<"3:冒泡排序"<<endl

<<"4:快速排序"<<endl<<"5:选择排序"<<endl<<"6:堆排序"<<endl

<<"7:归并排序"<<endl<<"8:基数排序"<<endl<<"0:结束"<<endl;

cin>>k;

while(k!=0)

{

if(k==1)

{

InitList(L,"Data0.txt");

start=clock();

InsertSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

InsertSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

InsertSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

InsertSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

InsertSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

InsertSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

InsertSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

InsertSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

InsertSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

InsertSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==2)

{

InitList(L,"Data0.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

ShellSort(L,d,14);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==3)

{

InitList(L,"Data0.txt");

start=clock();

BubbleSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

BubbleSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

BubbleSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

BubbleSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

BubbleSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

BubbleSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

BubbleSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

BubbleSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

BubbleSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

BubbleSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==4)

{

InitList(L,"Data0.txt");

start=clock();

QuickSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

QuickSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

QuickSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

QuickSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

QuickSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

QuickSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

QuickSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

QuickSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

QuickSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

QuickSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==5)

{

InitList(L,"Data0.txt");

start=clock();

SelectSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

SelectSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

SelectSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

SelectSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

SelectSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

SelectSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

SelectSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

SelectSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

SelectSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

SelectSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==6)

{

InitList(L,"Data0.txt");

start=clock();

HeapSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

HeapSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

HeapSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

HeapSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

HeapSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

HeapSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

HeapSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

HeapSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

HeapSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

HeapSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==7)

{

InitList(L,"Data0.txt");

start=clock();

MergeSort(L);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data1.txt");

start=clock();

MergeSort(L);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data2.txt");

start=clock();

MergeSort(L);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data3.txt");

start=clock();

MergeSort(L);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data4.txt");

start=clock();

MergeSort(L);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data5.txt");

start=clock();

MergeSort(L);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data6.txt");

start=clock();

MergeSort(L);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data7.txt");

start=clock();

MergeSort(L);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data8.txt");

start=clock();

MergeSort(L);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitList(L,"Data9.txt");

start=clock();

MergeSort(L);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

else if(k==8)

{

InitSLList(SLL,"Data0.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t0=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data1.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t1=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data2.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t2=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data3.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t3=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data4.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t4=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data5.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t5=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data6.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t6=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data7.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t7=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data8.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t8=(double)(finish-start)/CLOCKS\_PER\_SEC;

InitSLList(SLL,"Data9.txt");

start=clock();

RadixSort(SLL);

finish=clock();

t9=(double)(finish-start)/CLOCKS\_PER\_SEC;

printf("样本0排序所用时间为：%lfs\n",t0);

printf("样本1排序所用时间为：%lfs\n",t1);

printf("样本2排序所用时间为：%lfs\n",t2);

printf("样本3排序所用时间为：%lfs\n",t3);

printf("样本4排序所用时间为：%lfs\n",t4);

printf("样本5排序所用时间为：%lfs\n",t5);

printf("样本6排序所用时间为：%lfs\n",t6);

printf("样本7排序所用时间为：%lfs\n",t7);

printf("样本8排序所用时间为：%lfs\n",t8);

printf("样本9排序所用时间为：%lfs\n",t9);

}

cout<<"1:直接插入排序"<<endl<<"2:希尔排序"<<endl<<"3:冒泡排序"<<endl

<<"4:快速排序"<<endl<<"5:选择排序"<<endl<<"6:堆排序"<<endl

<<"7:归并排序"<<endl<<"8:基数排序"<<endl<<"0:结束"<<endl;

cin>>k;

}

}

else if(select1==2)

{

string a;

cout<<"请输入样本文件名:\n";

cin>>a;

InitList(L,a);

start=clock();

InsertSort(L);

finish=clock();

t1=(double)(finish-start);

InitList(L,a);

start=clock();

ShellSort(L,d,14);

finish=clock();

t2=(double)(finish-start);

InitList(L,a);

start=clock();

BubbleSort(L);

finish=clock();

t3=(double)(finish-start);

InitList(L,a);

start=clock();

QuickSort(L);

finish=clock();

t4=(double)(finish-start);

InitList(L,a);

start=clock();

SelectSort(L);

finish=clock();

t5=(double)(finish-start);

InitList(L,a);

start=clock();

HeapSort(L);

finish=clock();

t6=(double)(finish-start);

InitList(L,a);

start=clock();

MergeSort(L);

finish=clock();

t7=(double)(finish-start);

InitSLList(SLL,a);

start=clock();

RadixSort(SLL);

finish=clock();

t8=(double)(finish-start);

cout<<"该样本不同排序所花时间为："<<endl;

printf("直接插入排序所用时间为：%lfms\n",t1);

printf("希尔排序所用时间为：%lfms\n",t2);

printf("冒泡排序所用时间为：%lfms\n",t3);

printf("快速排序所用时间为：%lfms\n",t4);

printf("选择排序所用时间为：%lfms\n",t5);

printf("堆排序所用时间为：%lfms\n",t6);

printf("归并排序所用时间为：%lfms\n",t7);

printf("基数排序所用时间为：%lfms\n",t8);

}

cout<<"请选择按提示选择操作："<<endl;

cout<<"1:显示每种排序算法对不同样本所花的时间\n2:用相同样本对不同算法进行测试\n0:结束程序"<<endl;

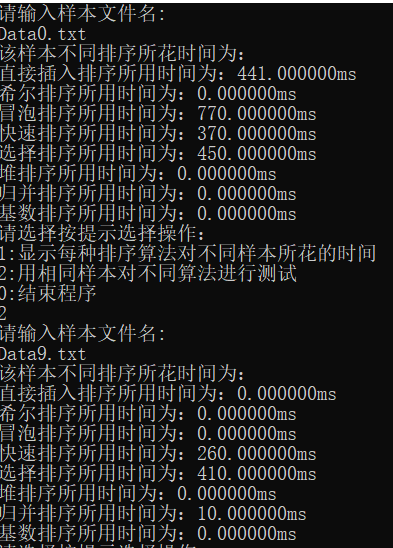
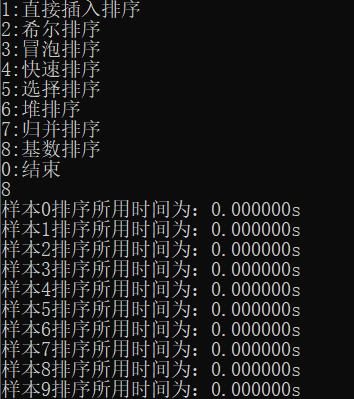
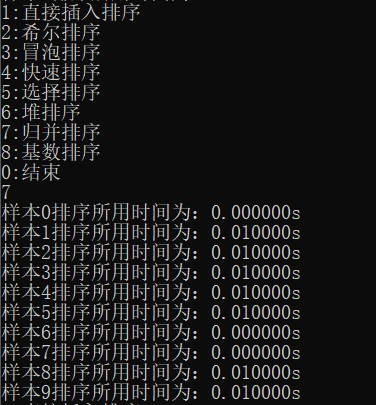
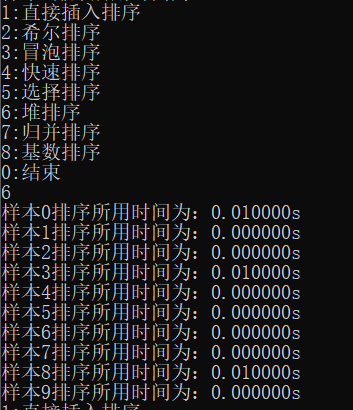
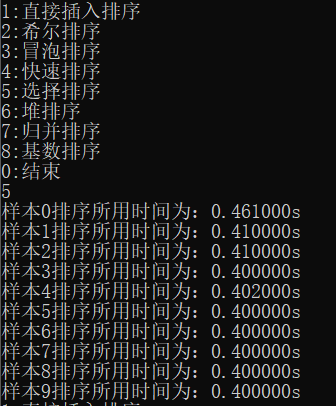
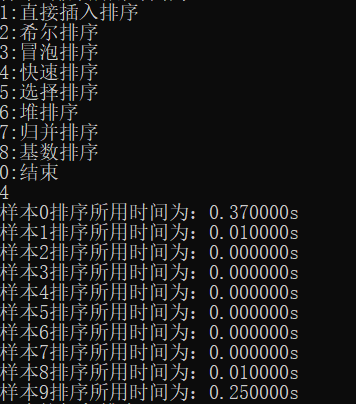
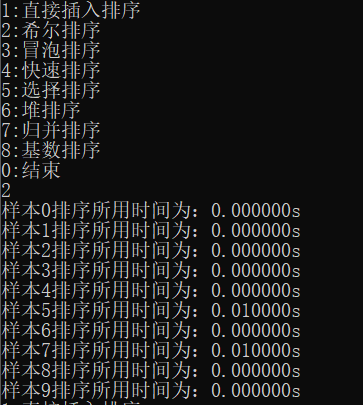
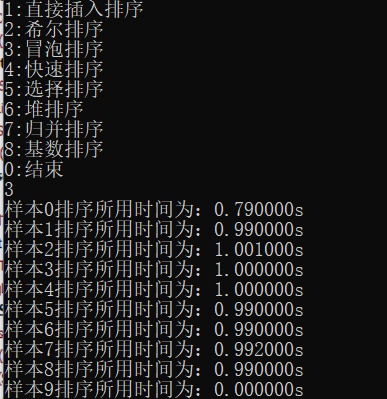
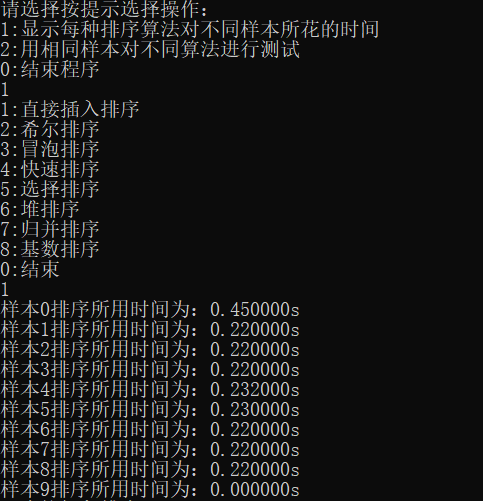
cin>>select1 ;

}

return 0;

}

**测试数据及结果：（样本0为逆序，样本9为顺序）**



**算法时间复杂度：**

基数排序：时间复杂度为O(rn)；堆排序的平均时间复杂度为O(nlogn),接近于最坏的时间复杂度。在最好情况下，时间复杂度为O(1). 快速排序：时间复杂度

O(n^2)数据序列对其无影响。归并：时间复杂度为O(logn)，选择：O(n！)，冒泡：时间复杂度最小为O(n),最大为O(n^2)，插入：时间复杂度最小为O(n),最大为O(n^2)，希尔排序：间隔取的合理可达到O(logn)。

**代码行数：941行.**

## 公交线路提示(图)

[问题描述]

上网下载南京公交线路图，建立南京主要公交线路图的存储结构。

[基本要求]

（1）输入任意两站点，给出转车次数最少的乘车路线。

（2）输入任意两站点，给经过站点最少的乘车路线。

**数据结构：**

本题采用的数据结构为图的邻接表,开始考虑使用邻接矩阵的存储结构发现空间不够。

还使用了栈的结构用来查找路线。

typedef char\* VexType;//图结构

typedef struct InforType{

char Route[MAXROU][5];

int rounum;

}InforType;

typedef struct ArcNode{

int adjvex;

InforType route;

struct ArcNode \*nextarc;

}ArcNode;

typedef struct VNode{

VexType Sitename;

ArcNode \*firstarc;

}VNode,AdjList[MAXVEX];

typedef struct MGraph{

AdjList vertices;

int vexnum;

int arcnum;

}ALGraph;

typedef struct Record{

int vexpostion;

ArcNode \*p;

}Record;//记录经过站点，深度优先遍历

typedef Record ElemType;//栈定义

typedef struct SqStack{

ElemType \*base;

ElemType \*top;

int stacksize;

}SqStack;

**算法思想：**

通过深度优先遍历查找从起点到终点的所有线路，将其中经过站点最少及Stacklength最小的保存起来，通过一个查并集的思想判断出这条线路共需要多次换乘，将换乘次数最少的线路用另外一个数组保存起来。最后输出。

**源代码：**

BusRoute.h:

#include<iostream>

#include<fstream>

#include<stdlib.h>

using namespace std;

#include<string.h>

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define STACK\_INIT\_SIZE 300

#define STACK\_INCREMENT 100

#define MAXVEX 1000

#define MAXROU 40

typedef int Status;

typedef char\* VexType;//图结构

typedef struct InforType{

char Route[MAXROU][5];

int rounum;

}InforType;

typedef struct ArcNode{

int adjvex;

InforType route;

struct ArcNode \*nextarc;

}ArcNode;

typedef struct VNode{

VexType Sitename;

ArcNode \*firstarc;

}VNode,AdjList[MAXVEX];

typedef struct MGraph{

AdjList vertices;

int vexnum;

int arcnum;

}ALGraph;

typedef struct Record{

int vexpostion;

ArcNode \*p;

}Record;

typedef Record ElemType;//栈定义

typedef struct SqStack{

ElemType \*base;

ElemType \*top;

int stacksize;

}SqStack;

Status InitStack(SqStack &S)

{

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

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

S.top=S.base;

S.stacksize=STACK\_INIT\_SIZE;

return OK;

}

Status DestroyStack(SqStack &S)

{

if(S.base!=NULL||S.top!=NULL)

{

free(S.base);

S.base=NULL;

S.top=NULL;

}

return OK;

}

int StackLength(SqStack &S)

{

int len;

if(S.base==NULL)

return ERROR;

len=S.top-S.base;

return len;

}

Status StackEmpty(SqStack &S)

{

if(S.base==S.top && S.base!=NULL && S.top!=NULL)

return OK;

else

return ERROR;

}

Status GetTop(SqStack &S,ElemType &e)

{

if(S.base==S.top)

return ERROR;

e=\*(S.top-1);

return OK;

}

Status Push(SqStack &S,ElemType e)

{

if((S.top-S.base)>=S.stacksize)

{

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

if(!S.base)

exit(OVERFLOW);

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

S.stacksize+=STACK\_INCREMENT;

}

\*S.top++=e;

return OK;

}

Status Pop(SqStack &S,ElemType &e)

{

if(S.base==S.top)

return ERROR;

e=\*--S.top;

return OK;

}

bool IsInStack(SqStack S,ElemType x)/\*判断节点是否在栈中\*/

{

if(S.base==S.top)

return false;

ElemType \*p;

p=S.base;

while(p<S.top)

{

if((\*p).vexpostion==x.vexpostion)

return true;

p++;

}

return false;

}

SqStack S;

int Shortestpath\_site[MAXVEX];

int Shortestpath\_route[MAXVEX];

int Locate(ALGraph &G,string n)

{

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

{

if(strcmp(G.vertices[i].Sitename,n.c\_str())==0)

return i;

}

return -1;

}

int Locate\_route(ArcNode cell,char \*n)//在边上查找线路是否在上面

{

for(int i=0;i<cell.route.rounum;i++)

{

if(strcmp(cell.route.Route[i],n)==0)

return i;

}

return -1;

}

Status CreateALGraph(ALGraph &G)

{

G.vexnum=0;

G.arcnum=0;

char filename[20];

cout<<"请输入公交车站点信息的文件名:"<<endl;

cin>>filename;

ifstream in;

in.open(filename);

if(!in)

{

cout<<"文件打开失败！"<<endl;

return ERROR;

}

string sitename,sitename1,rouname;

int sitenum,temp,location,location1;

while(!in.eof())

{

in>>rouname;

if(in.eof())

{

break;

}

in>>sitenum;

in>>sitename;

location=Locate(G,sitename);

if(location<0)

{

G.vertices[G.vexnum].Sitename=new char[sitename.length()+1];

strcpy(G.vertices[G.vexnum].Sitename,sitename.c\_str());

G.vertices[G.vexnum].firstarc=NULL;

location=G.vexnum;

G.vexnum++;

}

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

{

in>>sitename1;

location1=Locate(G,sitename1);

if(location1<0)

{

G.vertices[G.vexnum].Sitename=new char[sitename1.length()+1];

strcpy(G.vertices[G.vexnum].Sitename,sitename1.c\_str());

G.vertices[G.vexnum].firstarc=NULL;

location1=G.vexnum;

G.vexnum++;

}

ArcNode \*p,\*p1;

if(!G.vertices[location].firstarc)//第一个站名的情况

{

G.vertices[location].firstarc=new ArcNode;

G.vertices[location].firstarc->adjvex=location1;

G.vertices[location].firstarc->nextarc=NULL;

strcpy((G.vertices[location].firstarc)->route.Route[0],rouname.c\_str());

G.vertices[location].firstarc->route.rounum=1;

G.arcnum++;

}

else

{

p=G.vertices[location].firstarc;

if(p->adjvex==location1)

{

temp=G.vertices[location].firstarc->route.rounum;

strcpy(G.vertices[location].firstarc->route.Route[temp],rouname.c\_str());

G.vertices[location].firstarc->route.rounum++;

G.arcnum++;

}

else

{

while(p->nextarc!=NULL&&p->nextarc->adjvex!=location1)

{

p=p->nextarc;

}

if(!p->nextarc)

{

p->nextarc=new ArcNode;

p->nextarc->nextarc=NULL;

p->nextarc->adjvex=location1;

strcpy(p->nextarc->route.Route[0],rouname.c\_str());

p->nextarc->route.rounum=1;

G.arcnum++;

}

else

{

temp=p->nextarc->route.rounum;

strcpy(p->nextarc->route.Route[temp],rouname.c\_str());

p->nextarc->route.rounum++;

G.arcnum++;

}

}

}

if(!G.vertices[location1].firstarc)//第二个站名的情况

{

G.vertices[location1].firstarc=new ArcNode;

G.vertices[location1].firstarc->adjvex=location;

G.vertices[location1].firstarc->nextarc=NULL;

strcpy(G.vertices[location1].firstarc->route.Route[0],rouname.c\_str());

G.vertices[location1].firstarc->route.rounum=1;

}

else

{

p1=G.vertices[location1].firstarc;

if(p1->adjvex==location)

{

temp=G.vertices[location1].firstarc->route.rounum;

strcpy(G.vertices[location1].firstarc->route.Route[temp],rouname.c\_str());

G.vertices[location1].firstarc->route.rounum++;

}

else

{

while(p1->nextarc!=NULL&&p1->nextarc->adjvex!=location)

{

p1=p1->nextarc;

}

if(!p1->nextarc)

{

p1->nextarc=new ArcNode;

p1->nextarc->nextarc=NULL;

p1->nextarc->adjvex=location;

strcpy(p1->nextarc->route.Route[0],rouname.c\_str());

p1->nextarc->route.rounum=1;

}

else

{

temp=p1->nextarc->route.rounum;

strcpy(p1->nextarc->route.Route[temp],rouname.c\_str());

p1->nextarc->route.rounum++;

}

}

}

sitename=sitename1;

location=location1;

}//endfor

}//endwhile

in.close();

return OK;

}

char \*GetRoute(ALGraph G,int i,int j)

{

int k;

ArcNode \*p=G.vertices[i].firstarc;

while(p->adjvex!=j)

{

p=p->nextarc;

}

for(k=0;k<p->route.rounum-1;k++)

{

cout<<p->route.Route[k]<<"、";

}

cout<<p->route.Route[k];

}

void PrintPath\_rou(ALGraph G)

{

int i=1;

while(i<Shortestpath\_route[0])

{

cout<<G.vertices[Shortestpath\_route[i]].Sitename<<"-";

GetRoute(G,Shortestpath\_route[i],Shortestpath\_route[i+1]);

cout<<"->";

i++;

}

cout<<G.vertices[Shortestpath\_route[i]].Sitename;

cout<<endl;

}

void SavePath\_rou(SqStack S,int e)

{

ElemType \*p=S.base;

Shortestpath\_route[0]=StackLength(S)+1;

for(int i=1;p<S.top;i++,p++)

{

Shortestpath\_route[i]=p->vexpostion;

}

Shortestpath\_route[Shortestpath\_route[0]]=e;

}

void PrintPath\_site(ALGraph G)

{

int i=1;

while(i<Shortestpath\_site[0])

{

cout<<G.vertices[Shortestpath\_site[i]].Sitename<<"-";

GetRoute(G,Shortestpath\_site[i],Shortestpath\_site[i+1]);

cout<<"->";

i++;

}

cout<<G.vertices[Shortestpath\_site[i]].Sitename;

cout<<endl;

}

void SavePath\_site(SqStack S,int e)

{

ElemType \*p=S.base;

Shortestpath\_site[0]=StackLength(S)+1;

for(int i=1;p<S.top;i++,p++)

{

Shortestpath\_site[i]=p->vexpostion;

}

Shortestpath\_site[Shortestpath\_site[0]]=e;

}

bool IsExit(char r[MAXVEX][4],char \*s,int num)

{

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

{

if(strcmp(r[i],s)==0)

return true;

}

return false;

}

void Countnum(int \*a,char r[MAXVEX][4],char \*s,int num)

{

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

{

if(strcmp(r[i],s)==0)

{

a[i]++;

}

}

}

bool Adjust(bool \*m,char \*s)

{

int i;

ElemType \*q;

bool tag=false;

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

{

if(m[i]==false)

for(int j=0;j<q->p->route.rounum;j++)

{

if(strcmp(q->p->route.Route[j],s)==0)

{

m[i]=true;

tag=true;

break;

}

}

}

return tag;

}

bool Selectway(int \*a,bool \*t,bool \*m,int num,char r[MAXVEX][4])//选择合适的线路

{

int max=0;

int j=0;

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

{

if(t[i]==true)//先选出必经路线

{

t[i]=false;

a[i]=0;

if(Adjust(m,r[i]))//将这条路线经过的站点之间的弧进行标记

return true;

}

}

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

{

if(a[i]>max)//次优先选出所有出现次数最多并未被选出的线路

{

j=i;

max=a[i];

}

}

a[j]=0;

if(Adjust(m,r[j]))//如果对线路状态发生改变则有效否则这条线路多余

return true;

return false;

}

bool Check(bool \*m)

{

for(int i=0;i<StackLength(S);i++)

{

if(m[i]==false)

{

return false;

}

}

return true;

}

int GetRoutenum(ALGraph G)

{

char routename[MAXVEX][4];

int num=0,routenum=0,n;

n=StackLength(S);

ElemType \*q=S.base;

while(q<S.top)

{

for(int i=0;i<q->p->route.rounum;i++)

{

if(!IsExit(routename,q->p->route.Route[i],num))//记录该条路径出现的所有公交线路和数量

{

strcpy(routename[num],q->p->route.Route[i]);

num++;

}

}

q++;

}

int \*Count=new int[num]; //用于统计每条线路出现的次数

bool \*tag=new bool[num],\*Must=new bool[n];//tag用于判断是否该公交线路是必须经过的即唯一线路可达

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

{

Count[j]=0;

tag[j]=false;

}

for(int j=0;j<n;j++)//Must记录每两个站点之间是否已经存在线路可达

{

Must[j]=false;

}

q=S.base;

while(q<S.top)//统计并选出必须线路

{

for(int i=0;i<q->p->route.rounum;i++)

{

if(q->p->route.rounum==1)

{

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

{

if(strcmp(routename[i],q->p->route.Route[i])==0)

{

tag[i]=true;

}

}

}

Countnum(Count,routename,q->p->route.Route[i],num);

}

q++;

}

while(!Check(Must))//查看是否每条弧都已经存在线路可达

{

if(Selectway(Count,tag,Must,num,routename))//选择必进的线路或者出现次数最多的线路

routenum++;

}

free(Count);

free(tag);

free(Must);

return routenum;

}

Status AllPath\_DFSTraverse(ALGraph G,int s,int e)//深度优先遍历整个图找出所有通路选出其中需要的路径

{

int sitenum=2000;

int rounum=2000;

Record temp,top;

temp.vexpostion=s;

temp.p=G.vertices[s].firstarc;

if(!IsInStack(S,temp))

{

Push(S,temp);

}

while(!StackEmpty(S))

{

GetTop(S,top);

if(top.p==NULL)

{

Pop(S,top);

if(StackEmpty(S))

break;

(\*(S.top-1)).p=(\*(S.top-1)).p->nextarc;

continue;

}

temp.vexpostion=top.p->adjvex;

temp.p=G.vertices[temp.vexpostion].firstarc;

if(!IsInStack(S,temp))

{

if(temp.vexpostion==e)

{

int k=GetRoutenum(G);

if(k<rounum)//选换乘最少的路径

{

rounum=k;

SavePath\_rou(S,e);

}

if(StackLength(S)<sitenum)//选站点最少的路径

{

sitenum=StackLength(S);

SavePath\_site(S,e);

}

(\*(S.top-1)).p=(\*(S.top-1)).p->nextarc;

continue;

}

Push(S,temp);

}

else

{

(\*(S.top-1)).p=(\*(S.top-1)).p->nextarc;

}

}

cout<<"经过站点最少的乘车路线:\n"<<"共经过"<<sitenum+1<<"站"<<endl;

PrintPath\_site(G);

cout<<endl;

cout<<"转车次数最少的乘车路线:\n"<<"共转车"<<rounum-1<<"次,共经过"<<Shortestpath\_route[0]<<"个站点"<<endl;

PrintPath\_rou(G);

cout<<endl;

return OK;

}

Main.cpp:

#include"BusRoute.h"

int main()

{

ALGraph G;

if(!CreateALGraph(G))

return 0;

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

int select;

cout<<"请按相应按键进行操作：\n";

cout<<"1:查询公交车路线\n0:结束程序"<<endl;

cin>>select;

while(select!=0)

{

if(select==1)

{

cout<<"请输入起始车站名和终点车站名："<<endl;

string s,t;

InitStack(S);

cin>>s>>t;

int s1,t1;

s1=Locate(G,s);

t1=Locate(G,t);

while(s1<0||t1<0)

{

cout<<"车站名输入错误，请重新输入\n";

cin>>s>>t;

s1=Locate(G,s);

t1=Locate(G,t);

}

AllPath\_DFSTraverse(G,s1,t1);

DestroyStack(S);

}

else

{

cout<<"操作指令输入错误请重新输入\n";

}

cout<<"是否继续查询？(1:继续查找0:结束程序)"<<endl;

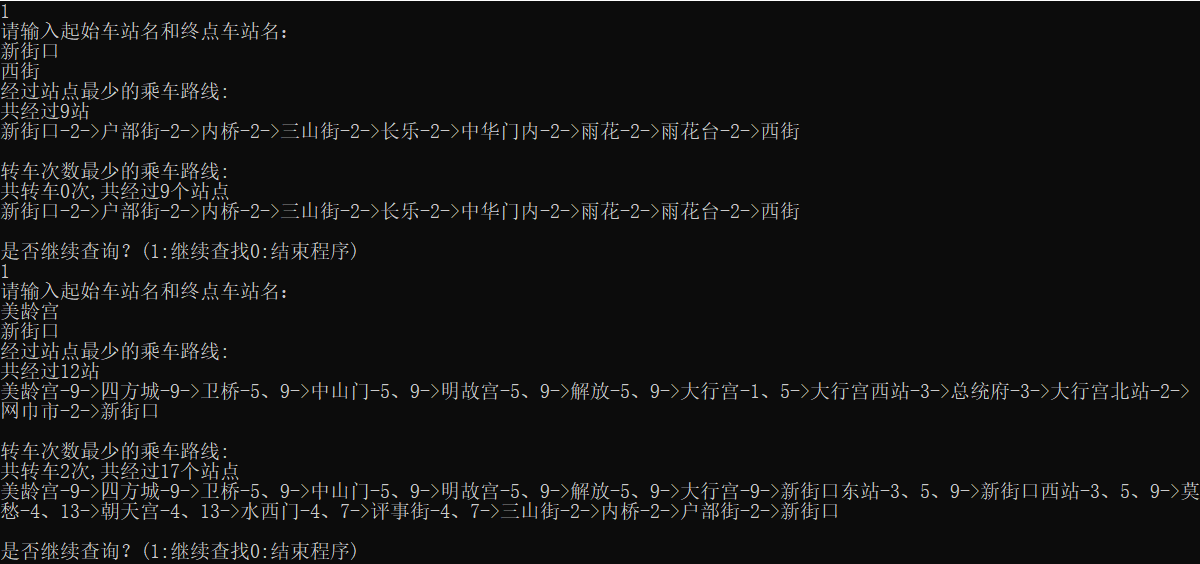
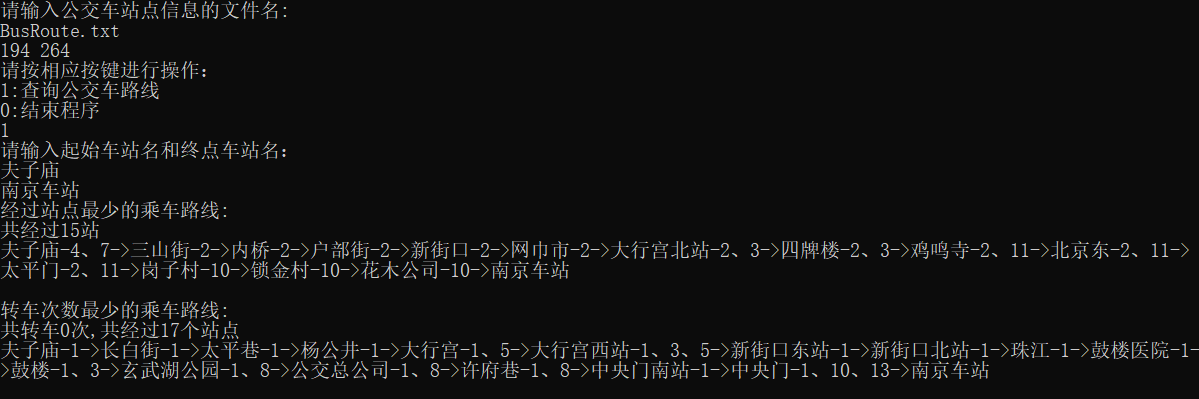
cin>>select;

}

return 0;

}

**测试数据及结果:**



**时间复杂度：**

建立图的过程中时间复杂度为：O(n)；查找过程取决于各顶点之间的线路x，时间复杂度为O(n^x)。当线路交叉过多的时候，时间复杂度就会变的十分高，并不实用，应该有办法改进，目前还未想到。

**代码行数：600行**

## 算术表达式求值(栈)

[问题描述]

　　一个算术表达式是由操作数(operand)、运算符(operator)和界限符(delimiter)组成的。假设操作数是正实数，运算符只含加减乘除等四种运算符，界限符有左右括号和表达式起始、结束符“#”，如：#6+15\*（21-8/4）#。引入表达式起始、结束符是为了方便。编程利用“运算符优先法”求算术表达式的值。

[基本要求]

（1） 从键盘或文件读入一个合法的算术表达式，输出正确的结果。

（2） 显示输入序列和栈的变化过程。

（3） 考虑算法的健壮性，当表达式错误时，要给出错误原因的提示。

（4） 实现非整数的处理。

**数据结构：**

本题采用栈的数据结构，利用两个栈，一个存放数值，一个存放操作符.

typedef struct Stack\_OPND{ //操作数栈

float \*base;

float \*top;

int stacksize;

}Stack\_OPND;

typedef struct Stack\_OPTR{ //操作符栈

char \*base;

char \*top;

int stacksize;

}Stack\_OPTR;

**算法思想：**

主要运用算符优先算法，使用两个工作栈。一个OPTR用于寄存运算符，另一个OPND寄存操作数和运算结果。（1）首先置OPND为空栈，表达式起始符“#”为OPTR栈的栈底元素；（2）依次读入表达式中的每个字符，若是操作数则读下一个字符直到读到操作符，再将得到的操作数值字符串转换成实数入栈，若是运算符则和OPTR栈的栈顶运算符比较优先权后做相应操作，直至整个表达式求值完毕（即OPTE栈的栈顶元素和当前读入的字符均为“#”）。

**源代码：**

Expression.h:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#define OK 1

#define ERROR 0

#define INIT\_STACK\_SIZE 20

#define INCREMENT\_SIZE 10

typedef int Status;

typedef struct Stack\_OPND{ //操作数栈

float \*base;

float \*top;

int stacksize;

}Stack\_OPND;

typedef struct Stack\_OPTR{ //操作符栈

char \*base;

char \*top;

int stacksize;

}Stack\_OPTR;

Status InitStack\_OPND(Stack\_OPND &S)

{

S.base=(float\*)malloc(sizeof(float)\*INIT\_STACK\_SIZE);

if(!S.base)

{

exit(OVERFLOW);

}

S.top=S.base;

S.stacksize=INIT\_STACK\_SIZE;

return OK;

}

float GetTop\_OPND(Stack\_OPND &S)

{

float f;

if(S.base==S.top)

return ERROR;

f=\*(S.top-1);

return f;

}

Status Push\_OPND(Stack\_OPND &S,float f)

{

if(S.top-S.base>=S.stacksize)

{

S.base=(float\*)realloc(S.base,sizeof(float)\*(S.stacksize+INCREMENT\_SIZE));

if(!S.base)

{

exit(OVERFLOW);

}

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

S.stacksize+=INCREMENT\_SIZE;

}

\*(S.top)=f;

S.top++;

return OK;

}

Status Pop\_OPND(Stack\_OPND &S,float &f)

{

if(S.top==S.base)

{

return ERROR;

}

f=\*(--S.top);

return OK;

}

Status EmptyStack\_OPND(Stack\_OPND S)

{

if(S.base==S.top)

return OK;

else

return ERROR;

}

Status InitStack\_OPTR(Stack\_OPTR &S)

{

S.base=(char\*)malloc(sizeof(char)\*INIT\_STACK\_SIZE);

if(!S.base)

{

exit(OVERFLOW);

}

S.top=S.base;

S.stacksize=INIT\_STACK\_SIZE;

return OK;

}

char GetTop\_OPTR(Stack\_OPTR &S)

{

char c;

if(S.base==S.top)

return ERROR;

c=\*(S.top-1);

return c;

}

Status EmptyStack\_OPTR(Stack\_OPTR S)

{

if(S.base==S.top)

return OK;

else

return ERROR;

}

Status Push\_OPTR(Stack\_OPTR &S,char c)

{

if(S.top-S.base>=S.stacksize)

{

S.base=(char\*)realloc(S.base,sizeof(char)\*(S.stacksize+INCREMENT\_SIZE));

if(!S.base)

{

exit(OVERFLOW);

}

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

S.stacksize+=INCREMENT\_SIZE;

}

\*(S.top)=c;

S.top++;

return OK;

}

Status Pop\_OPTR(Stack\_OPTR &S,char &c)

{

if(S.top==S.base)

{

return ERROR;

}

c=\*(--S.top);

return OK;

}

void PrintStack\_OPND(Stack\_OPND S)

{

printf("OPND栈:\n");

for(float \*p=S.base;p<S.top;p++)

{

printf("%g ",\*p);

}

printf("\n");

}

void PrintStack\_OPTR(Stack\_OPTR S)

{

printf("OPTR栈:\n");

for(char \*p=S.base;p<S.top;p++)

{

printf("%c ",\*p);

}

printf("\n");

}

float StrToFloat(char \*s)

{

int n,i;

float temp=0;

for(i=0;i<strlen(s);i++)

{

if(s[i]=='.')

break;

}

for(int j=0;j<strlen(s);j++)

{

if(j<i)

temp+=(float)(s[j]-'0')\*pow(10,i-j-1);

else if(j==i)

continue;

else

temp+=(s[j]-'0')\*pow(10,i-j);

}

return temp;

}

char OPSET[7]={'+','-','\*','/','(',')','#'};

unsigned char Prior[7][7] =

{ // 运算符优先级表

// '+' '-' '\*' '/' '(' ')' '#'

/\*'+'\*/'>','>','<','<','<','>','>',

/\*'-'\*/'>','>','<','<','<','>','>',

/\*'\*'\*/'>','>','>','>','<','>','>',

/\*'/'\*/'>','>','>','>','<','>','>',

/\*'('\*/'<','<','<','<','<','=','>',

/\*')'\*/'>','>','>','>','=','>',' ',

/\*'#'\*/'<','<','<','<','<',' ','='

};

Status In(char Test)//检测该字符是否为操作符号

{

int Find=false;

for (int i=0; i< 7; i++)

{

if(Test == OPSET[i])

Find= true;

}

return Find;

}

Status In\_num(char c)

{

if((c<='9'&&c>='0')||c=='.')

return OK;

return ERROR;

}

float Operate(float a,unsigned char optr, float b)//计算

{

switch(optr)

{

case '+': return a+b;

case '-': return a-b;

case '\*': return a\*b;

case '/': return a/b;

default : return 0;

}

}

Status ReturnOpOrd(char op)//找到操作符号的位置以比较优先级

{

for(int i=0; i< 7; i++)

{

if (op == OPSET[i])

return i;

}

}

Status Locate(char \*s,char c)

{

for(int i=0;i<strlen(s);i++)

{

if(s[i]==c)

return OK;

}

return ERROR;

}

char precede(char A, char B)//比较两个操作符的优先级

{

return Prior[ReturnOpOrd(A)][ReturnOpOrd(B)];

}

float EvaluateExpression(char\* Expression,float &result)

{

// 算术表达式求值的算符优先算法

// 设OPTR和OPND分别为运算符栈和运算数栈

Stack\_OPTR OPTR; // 运算符栈，字符元素

Stack\_OPND OPND; // 运算数栈，实数元素

InitStack\_OPTR(OPTR);

InitStack\_OPND(OPND);

char TempData[20],temp;//用来表示实数

float Data,a,b;

char optr,\*c,Dr[]={'#','\0'};

Push\_OPTR(OPTR,'#');

printf("输入:#\n");

PrintStack\_OPND(OPND);

PrintStack\_OPTR(OPTR);

printf("\n");

c=strcat(Expression,Dr);//不改变原表达式

strcpy(TempData,"\0");

while (\*c!= '#'||GetTop\_OPTR(OPTR)!='#')

{

printf("输入:%c\n",\*c);

if(EmptyStack\_OPND(OPND)&&(In(\*c)&&\*c!='('))//如果操作数栈是空的读入的第一个字符除了'('以外其他操作符均非法.

{

printf("表达式错误,非法运算符\n\n");

return ERROR;

}

if (In\_num(\*c))

{

Dr[0]=\*c;

if(Locate(TempData,\*c))

{

printf("表达式错误,非法数值\n\n");

return ERROR;

}

strcat(TempData,Dr);

c++;

if (In(\*c))

{

Data=StrToFloat(TempData);

Push\_OPND(OPND, Data);

strcpy(TempData,"\0");

}

}

else if(In(\*c))//运算符

{

switch (precede(GetTop\_OPTR(OPTR),\*c))

{

case '<': // 栈顶元素优先级低

if(GetTop\_OPTR(OPTR)=='#'&&!EmptyStack\_OPND(OPND)&&\*c=='(')//判断“23(45)...”情况的发生

{

printf("表达式错误,非法运算符\n\n");

return ERROR;

}

Push\_OPTR(OPTR, \*c);

c++;

break;

case '=': // 脱括号并接收下一字符

Pop\_OPTR(OPTR,temp);

c++;

break;

case '>': // 退栈并将运算结果入栈

Pop\_OPTR(OPTR,optr);

if(!Pop\_OPND(OPND,b)||!Pop\_OPND(OPND,a)||optr=='#')//输入多个运算符不符合运算规则

{

printf("表达式错误,非法运算符\n\n");

return ERROR;

}

Push\_OPND(OPND,Operate(a,optr,b));

break;

default://‘)’没有(与之配对

printf("表达式错误,非法运算符\n\n");

return ERROR;

}

}

else

{

printf("表达式错误,非法字符\n\n");

return ERROR;

}

PrintStack\_OPND(OPND);

PrintStack\_OPTR(OPTR);

printf("\n");

}

Pop\_OPND(OPND,result);

if(!EmptyStack\_OPND(OPND))

{

printf("表达式错误,非法运算符\n\n");

return ERROR;

}

return OK;

}

Main.cpp:

#include"Expression.h"

int main()

{

int select;

float Out;

char s[128];

printf("请选择读取表达式方式：\n");

printf("1:键盘输入\n2:文件读取\n0:结束程序\n");

scanf("%d",&select);

while(select!=0)

{

if(select==1)

{

puts("请输入表达式:");

scanf("%s",s);

if(EvaluateExpression(s,Out))

{

puts("该表达式的值为:");

printf("%s\b=%g\n\n",s,Out);

}

}

else if(select==2)

{

puts("请输入文件名:");

char filename[81],temp;

scanf("%s",filename);

FILE \*fp;

fp=fopen(filename,"r");

if(!fp)

{

puts("文件打开失败！\n");

exit(0);

}

int i=0;

while(!feof(fp))

{

temp=fgetc(fp);

if(temp!=EOF)

s[i++]=temp;

}

s[i]='\0';

fclose(fp);

printf("该表达式为:%s\n",s);

if(EvaluateExpression(s,Out))

{

puts("该表达式的值为:");

printf("%s\b=%g\n\n",s,Out);

}

}

else

printf("错误输入！\n\n");

printf("请选择读取表达式方式：\n");

printf("1:键盘输入\n2:文件读取\n0:结束程序\n");

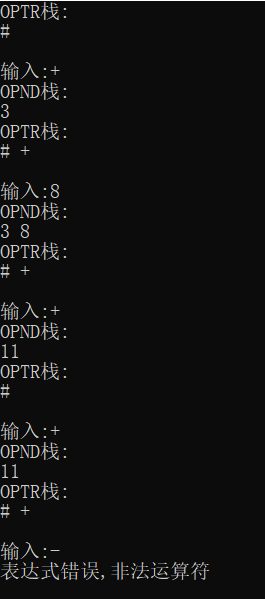
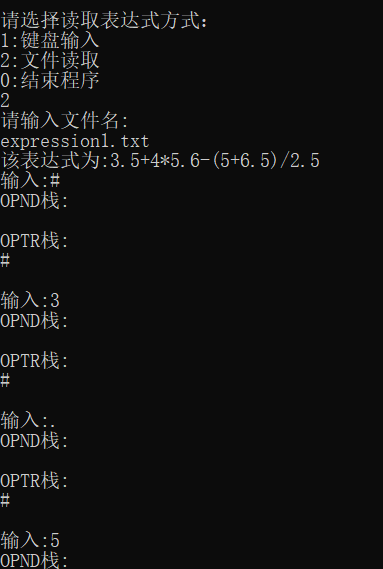
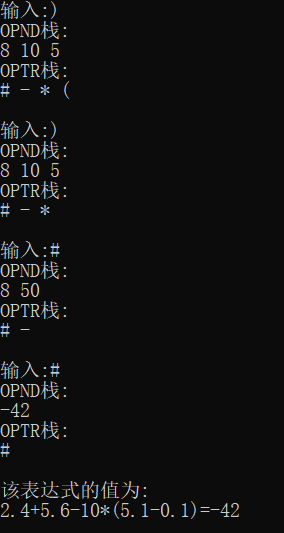
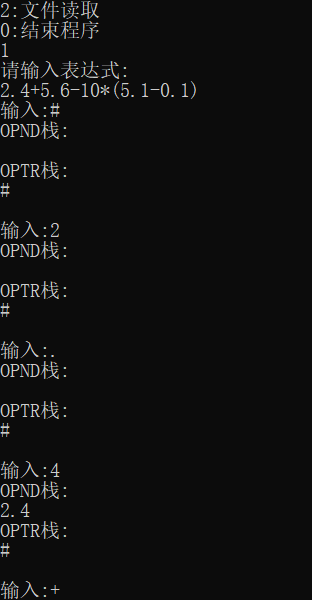
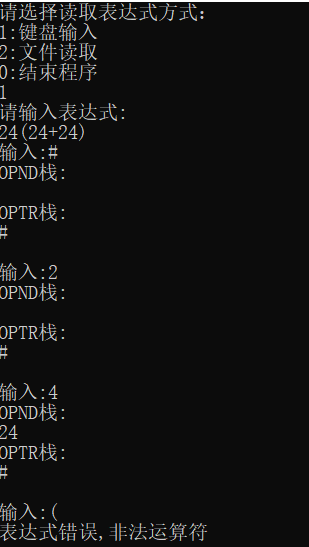
scanf("%d",&select);

}

return 0;

}

**测试数据及结果:**



**算法时间复杂度：**

该算法的时间复杂度为O(n),n及表达式字符长度。**代码行数：367行**

## 电子小字典（查找）

[问题描述]

利用键树结构，建立一个微型电子字典。

[基本要求]

实现生词的加入，单词的查找、删除，修改等操作。

**数据结构:**

本题采用键书及字典树的结构：

typedef struct TrieNode{

int count;//统计单词前缀出现次数。

struct TrieNode \*next[26];//26个英文字母

bool IsWord;//结束标志，表示到这个结点能够构成一个单词

char trans[30];//单词意思，翻译

}TrieNode,\*Trie;

**算法思想：**

本题利用键树的结构，每输入一个单词从每个字符开始减去’a’得到0-25，在next[i]上生成一个节点，直到整个单词结束将结点的IsWord标志改为true，再将翻译放在上面,查找的过程跟建立一样直到单词结束访问到的结点看其标志是否为一个单词或者单词未结束就已经是空指针即查找失败。修改即先找到再进行单词修改。删除使用两个指针查找，将要删除的结点的父节点返回直接delete。

**源程序：**

Trie.h:

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<string.h>

using namespace std;

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef int Status;

typedef struct TrieNode{

int count;//统计单词前缀出现次数。

struct TrieNode \*next[26];//26个英文字母

bool IsWord;//结束标志，表示到这个结点能够构成一个单词

char trans[30];//单词意思，翻译

}TrieNode,\*Trie;

TrieNode \*CreateTrieNode()

{

TrieNode\* node=(TrieNode \*)malloc(sizeof(TrieNode));

if(!node)

{

cout<<"OVERFLOW!"<<endl;

return NULL;

}

node->count=0;

node->IsWord=false;

memset(node->next,0,sizeof(node->next));//初始化为空指针

return node;

}

Trie Trie\_Search\_D(Trie &T,char \*word,int &i)

{

Trie node=T,parent=NULL;

char \*p=word;

int id;

while(\*p)

{

id=\*p - 'a';

parent=node;

i=id;

node= node->next[id];

++p;

if(node == NULL)

return NULL;

}

if(node->IsWord)// 查找成功

return parent;

else // 查找失败

return NULL;

}

char\* Trie\_Search(Trie root, char\* word)

{

Trie node = root;

char \*p = word;

int id;

while( \*p )

{

id = \*p - 'a';

node = node->next[id];

++p;

if(node == NULL)

return NULL;

}

if(node->IsWord) // 查找成功

return node->trans;

else // 查找失败

return NULL;

}

void FileDeal\_Delete(char \*word)

{

fstream in,out;

char w[200][26],t[200][30];

int i=0;

in.open("DataBase.txt",ios::in);

while(!in.eof())

{

in>>w[i];

if(in.eof())

{

break;

}

in>>t[i];

i++;

}

in.close();

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

{

if(strcmp(word,w[j])==0)

{

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

{

strcpy(w[k],w[k+1]);

strcpy(t[k],t[k+1]);

}

i--;

break;

}

}

out.open("DataBase.txt",ios::out);

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

{

out<<w[j]<<" "<<t[j]<<endl;

}

out.close();

}

void FileDeal\_Revisetran(char \*word,char \*tran)

{

fstream in,out;

char w[200][26],t[200][30];

int i=0;

in.open("DataBase.txt",ios::in);

while(!in.eof())

{

in>>w[i];

if(in.eof())

{

break;

}

in>>t[i];

i++;

}

in.close();

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

{

if(strcmp(word,w[j])==0)

{

strcpy(w[j],word);

strcpy(t[j],tran);

break;

}

}

out.open("DataBase.txt",ios::out);

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

{

out<<w[j]<<" "<<t[j]<<endl;

}

out.close();

}

void FileDeal\_Reviseword(char \*word,char \*new\_word)

{

fstream in,out;

char w[200][26],t[200][30];

int i=0;

in.open("DataBase.txt",ios::in);

while(!in.eof())

{

in>>w[i];

if(in.eof())

{

break;

}

in>>t[i];

i++;

}

in.close();

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

{

if(strcmp(word,w[j])==0)

{

strcpy(w[j],new\_word);

break;

}

}

out.open("DataBase.txt",ios::out);

for(int j=0;j<i-1;j++)

{

out<<w[j]<<" "<<t[j]<<endl;

}

out.close();

}

Status Trie\_Insert(Trie &root, char\* word, char\* trans)

{

fstream out;

out.open("DataBase.txt",ios::out|ios::app);

if(!out)

{

cout<<"数据文件打开失败！"<<endl;

return ERROR;

}

Trie node=root;

int i;

if(Trie\_Search\_D(root,word,i)!=NULL)

{

cout<<"单词已存在！"<<endl;

return ERROR;

}

char \*p=word;

int id;

while(\*p)

{

id = \*p-'a';

if(node->next[id] == NULL)

{

node->next[id]=CreateTrieNode();

}

node = node->next[id]; // 每插入一步，相当于有一个新串经过，指针向下移动

++p;

node->count += 1; // 这行代码用于统计每个单词前缀出现的次数（也包括统计每个单词出现的次数

}

node->IsWord=true; //单词结束的地方标记此处可以构成一个单词

strcpy(node->trans,trans);

out<<word<<" "<<trans<<endl;

out.close();

return OK;

}

Status InitTrie(Trie &T)

{

T=CreateTrieNode();

if(!T)

{

return ERROR;

}

fstream in;

in.open("DataBase.txt",ios::in);

if(!in)

{

cout<<"数据文件打开失败。\n";

}

char word[26],tran[20];

while(!in.eof())

{

in>>word;

if(in.eof())

{

break;

}

in>>tran;

Trie node=T;

char \*p=word;

int id;

while(\*p)

{

id = \*p-'a';

if(node->next[id] == NULL)

{

node->next[id]=CreateTrieNode();

}

node = node->next[id];

++p;

node->count += 1;

}

node->IsWord=true;

strcpy(node->trans,tran);

}

in.close();

return OK;

}

Status DeleteWord(Trie &T,char \*word)

{

Trie p;

int location;//删除的位置

p=Trie\_Search\_D(T,word,location);

if(p)

{

p->count--;

delete p->next[location];

p->next[location]=NULL;

FileDeal\_Delete(word);

return OK;

}

else

return ERROR;

}

Status Revise\_Word(Trie &T,char \*s,char \*t)

{

Trie p;

int location;

char tran[30];

p=Trie\_Search\_D(T,s,location);

if(p)

{

p->count--;

strcpy(tran,p->trans);

delete p->next[location];

p->next[location]=NULL;

Trie\_Insert(T,t,tran);

FileDeal\_Reviseword(s,t);

return OK;

}

else

{

return ERROR;

}

}

Status Revise\_Tran(Trie &T,char \*word,char \*new\_tran)

{

Trie p;

int location;

char \*tran;

p=Trie\_Search\_D(T,word,location);

if(p)

{

strcpy(p->next[location]->trans,new\_tran);

FileDeal\_Revisetran(word,new\_tran);

return OK;

}

else

return ERROR;

}

void PrintAllword()

{

fstream in;

char word[26];

char tran[30];

in.open("DataBase.txt",ios::in);

while(!in.eof())

{

in>>word;

if(in.eof())

{

break;

}

in>>tran;

cout<<word<<endl<<"译："<<tran<<endl;

}

in.close();

}

Main.cpp:

#include"Trie.h"

int main()

{

Trie Root;

int select,k;

char word[26],tran[30],revise[30];

char \*find;

InitTrie(Root);

cout<<"请按根据提示进行操作："<<endl;

cout<<"1:查找生词\n2:添加单词\n3:删除单词\n4:修改单词\n5:显示所有生词\n0:结束程序"<<endl;

cin>>select;

while(select!=0)

{

if(select==1)

{

cout<<"请输入要查找的单词:"<<endl;

cin>>word;

find=Trie\_Search(Root,word);

if(find)

{

cout<<word<<endl<<"译:"<<find<<endl;

}

else

{

cout<<"字典中不存在该单词，是否选择插入(1:Y,0:N):"<<endl;

cin>>k;

if(k==1)

{

cout<<"请输入"<<word<<"的翻译:"<<endl;

cin>>tran;

Trie\_Insert(Root,word,tran);

cout<<"生词添加成功!"<<endl;

}

else if(k==0)

{

}

else

{

cout<<"操作指令错误！"<<endl;

}

}

}

else if(select==2)

{

cout<<"请输入你要添加的单词跟它的翻译:"<<endl;

cin>>word;

cin>>tran;

if(Trie\_Insert(Root,word,tran))

{

cout<<"生词添加成功！"<<endl;

}

}

else if(select==3)

{

cout<<"请输入要删除的单词:"<<endl;

cin>>word;

if(DeleteWord(Root,word))

cout<<"单词删除成功!"<<endl;

else

cout<<"字典中无此单词!"<<endl;

}

else if(select==4)

{

cout<<"请选择修改内容:\n1:修改单词\n2:修改翻译"<<endl;

cin>>k;

if(k==1)

{

cout<<"请输入要修改的单词和修改后的单词:"<<endl;

cin>>word;

cin>>revise;

if(Revise\_Word(Root,word,revise))

{

cout<<"修改成功！"<<endl;

}

else

{

cout<<"字典中无您要修改的单词!"<<endl;

}

}

else if(k==2)

{

cout<<"请输入要修改的单词和修改后的翻译:"<<endl;

cin>>word;

cin>>revise;

if(Revise\_Tran(Root,word,revise))

{

cout<<"修改成功！"<<endl;

}

else

{

cout<<"字典中无您要修改的单词!"<<endl;

}

}

else

{

cout<<"操作指令错误!"<<endl;

}

}

else if(select==5)

{

PrintAllword();

}

else

{

cout<<"操作指令错误!"<<endl;

}

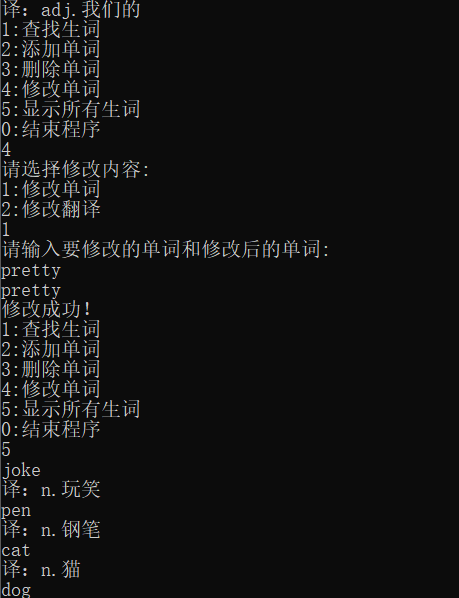
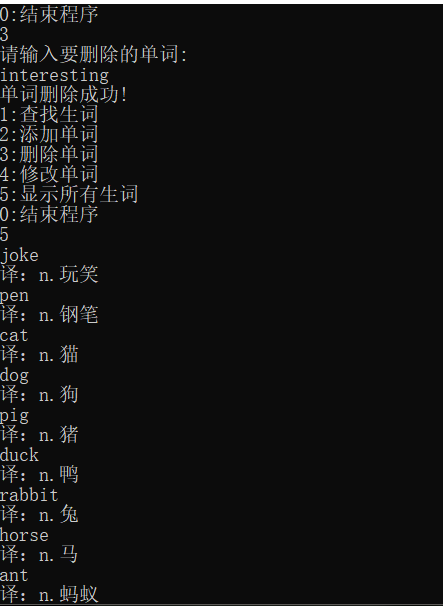
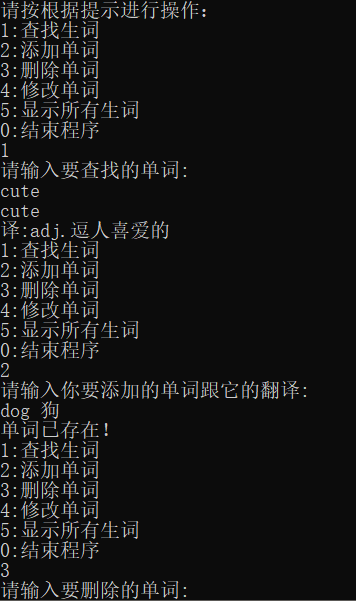
cout<<"1:查找生词\n2:添加单词\n3:删除单词\n4:修改单词\n5:显示所有生词\n0:结束程序"<<endl;

cin>>select;

}

}

**测试数据及结果：**



**算法时间复杂度：**

本算法的特点就是以空间换时间，对于单个字符查找都是O(1),单词长度为n,则查找时间为O(n)。删除和修改同样。

**代码行数：425行**

## 连连看

[问题描述]

建立一个10\*20的矩形方格图，其中有10种不同的图案，每种图案偶数个，填满矩形方格图。

[基本要求]

输入两个位置，如果两者图案相同，并且用少于等于3的直线相连，可消除该两个图案。

**数据结构：**

本题采用的数据结构为栈结构和数组，结点信息存放图案的位置信息：

struct data{

int x;

int y;//x,y记录当前位置信息

int dir;//记录移动的方向

int turn;//记录转弯次数即直线数量

bool operator < (const data &c)const//重载运算符用于比较两个基础元素

{

return c.turn<turn;

}

};

**算法思想：**

本题我采用的算法跟迷宫类似，原始地图用数组存储，每次获得两个相同图案的位置坐标以后，用回溯法去找是不是存在通路，即使用迷宫的方法，不同的是没有一次路径上的改变都需要记录，只有满足路劲改变次数小于等于2的通路才能消除两个图案，消除了图案用’ ‘来表示同时也是可通位置。直到整个数组全为空或者消除的数量等于总图案数，游戏结束。

**源代码：**

LLK.h:

#include<iostream>

#include<stdio.h>

#include<queue>

#include<string.h>

#include<stdlib.h>

#include<time.h>

using namespace std;

struct data{

int x;

int y;//x,y记录当前位置信息

int dir;//记录移动的方向

int turn;//记录转弯次数即直线数量

bool operator < (const data &c)const//重载运算符用于比较两个基础元素

{

return c.turn<turn;

}

};

priority\_queue<data> q;//调用队列

data temp,temp2;//用于在连线过程中的查找

int tempx,tempy,tempdir,tempturn;

int m,n,sum,x1,x2,yy1,y2;//m,n为地图尺寸，sum为图像总数量

int map[11][21];

int direct[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//四个方向

bool already[11][21];//用于标记已走过的路劲

bool Judge()//判断是否当前位置存在图像

{

if(map[tempx][tempy]!=0)

return false;

if(already[tempx][tempy])

return false;

return true;

}

void Push()//标记已经消除的图像

{

temp2.x=tempx;

temp2.y=tempy;

temp2.dir=tempdir;

temp2.turn=tempturn;

q.push(temp2);

already[tempx][tempy]=true;

}

bool Linking()//判断是否能够消除图像

{

if(x1<=0 || x2<=0 || yy1<=0 || y2<=0 || x1>=m+1 || yy1>=n+1 || x2>=m+1 || y2>=n+1)//输入的位置不在范围内

return false;

if(map[x1][yy1]==0 || map[x2][y2]==0)//输入的位置图像已被清除

return false;

if(map[x1][yy1]!=map[x2][y2])//所要连接的两个图像不同

return false;

temp.x=x1;

temp.y=yy1;

temp.dir=9999;//初始方向

temp.turn=-1;//初始表示未发生转弯

while(!q.empty())//清空队列

q.pop();

memset(already,false,sizeof(already));//清空路径情况

q.push(temp);

already[x1][yy1]=true;

while (!q.empty())

{

temp=q.top();

for(int i=0;i<=3;i++)//四个方向一一移动看能否在三条直线内连接

{

tempx=temp.x+direct[i][0];

tempy=temp.y+direct[i][1];

tempdir=i;

if(tempx<0 || tempy<0 || tempx>m+1 || tempy>n+1)

continue;

if(temp.dir==9999 || (temp.dir+i)%2==1)//第一次移动产生第一条直线或者每转一次方向就转弯加一

tempturn=temp.turn+1;

else

tempturn=temp.turn;

if(tempturn>=3)//大于三条直线，无法消除

continue;

if(tempx==x2 && tempy==y2)//可成功消除

return true;

if(!Judge())//路途中间有图像无法通过

continue;

Push();

}

q.pop();

}

return false;

}

double Random(double start, double end)//随机函数，产生范围内的随机数产生字符

{

return start+(end-start)\*rand()/(RAND\_MAX + 1.0);

}

bool Isexist(int a[],int x)

{

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

{

if(x==a[i])

return true;

}

return false;

}

void Creat()//初始化地图

{

int t=0,a,i,j;

int k,l=0;

int Is[10]={0};//用于生成10个不同图案

srand(time(0));

memset(map,0,sizeof(map));

while(t!=sum&&l<10)

{

a=(int)Random(0,26)+1;

if(Isexist(Is,a))//保证10个不相同的图案，如果生成的图案已有则再次随机

continue;

Is[l]=a;

l++;//用于记录已经产生的图案种类

while(1)

{

if(l==10)//最后一次生成数量由总图案减去已生成的所有图案数量

{

k=sum-t;

break;

}

k=(int)Random(2,30);

if(k%2==0&&(k+2\*(10-l)+t)<=sum)//保证每个图案个数为偶数并且数量都大于2

break;

}

for(int l=1;l<=k;l++)

{

while(1)

{

i=(int)Random(0,m)+1;

j=(int)Random(0,n)+1;

if(map[i][j]==0)

{

map[i][j]=a;

break;

}

}

}

t+=k;

}

}

void Print()//打印版面

{

char c;

cout<<" ";

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

cout<<i%10;

cout<<endl<<" ";

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

cout << "-";

cout<<endl;

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

{

cout<<i%10<< "|";

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

{

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

{

c=map[i][j]-1+'a';

cout<<c;

}

else

cout<<" ";

}

cout<<"|"<<endl;

}

cout<<" ";

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

cout<<"-";

cout << endl;

}

void Dealing()//处理每次的连接函数

{

system("cls");

if(Linking())

{

map[x1][yy1]=map[x2][y2]=0;

sum-=2;

if(sum!=0)

{

cout<<"真棒!请继续加油！"<<endl<<endl;

Print();

}

}

else

{

cout<<"这不可以哦，请再找找!"<<endl<<endl;

Print();

}

}

Main.cpp:

#include"LLK.h"

int main()

{

cout << "-----------------------连连看-------------------------" << endl;

cout << "然后每次输入4个数，代表要消的两个格子的所在行数和列数." << endl;

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

m=10,n=20;

cout << endl;

sum=m\*n;

Creat();//初始化游戏地图

cout<<"游戏即将开始！"<<endl;

system("pause");

cout<<"Map has created!"<<endl<<endl;

Print();

while(1)

{

if(sum==0)

{

cout<<"你真厉害，恭喜通关啦!" << endl;

break; //直到整个地图全部清除完成游戏结束

}

cout<<"xi yi:";

cin>> x1;

cin>> yy1;

cout<<"xi yi:";

cin>> x2;

cin>> y2;

cout<<endl;

Dealing();

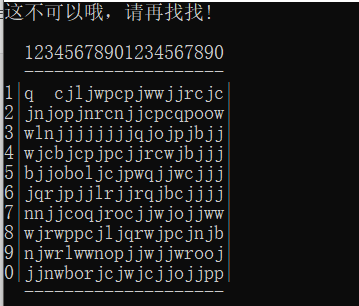
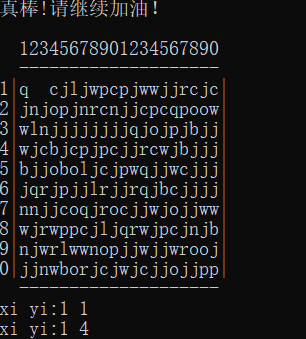
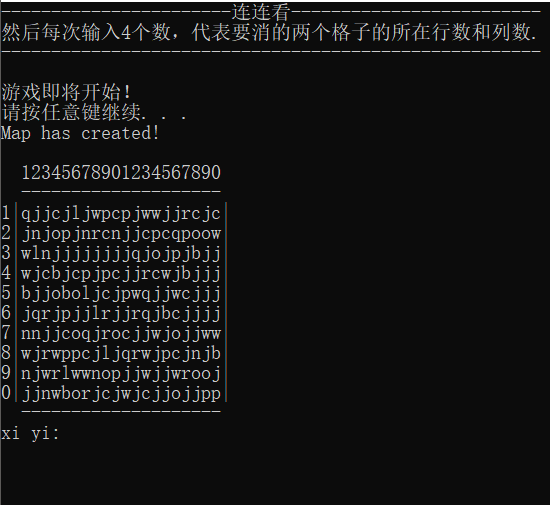
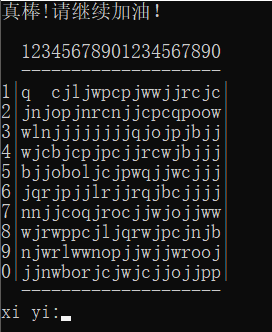
}

system("pause"); //游戏结束后暂停

return 0;

}

**测试数据及结果:**



**算法时间复杂度：**

由于使用的回溯法每次查找两个相同图案的时间都不一样，最坏情况为O(n^4)n表述图案个数，即每次都遍历整个数组并且四个方向都走一遍。

**代码行数：220行**

## 结束语（心得体会）

本次数据结构课程设计只算作业时长总共花了15多天，在这其中程序出现了各种问题，每次找到之后一一调试花了很多时间也发现了代码注释的重要性。我一共写了11道题，但是其中有一道题B-树的应用没有调试成功，准备再课后再花时间调试出来。

代码量情况：

1：600行

2：356行

3：528行

4：378行

5：353行

6：941行

7：600行

8：367行

9：425行

10：220行

总代码行数：4768行

感想：

通过这次的学习和实践，解决实际问题，让我对数据的各种结构有了更深的了解，对数据结构产生了浓厚的兴趣，同时也让我提高了解决实际问题的能力。我们要不断的通过上机来提高自己的学习水平，在上机的同时改正了自己对某些算法的错误使用，使自己在通过程序解决问题时抓住关键算法，有了算法设计思想和流程图，并用C语言描绘出关键算法。以前我对数据结构（C语言描述）的一些标准库函数不太了解，还有对函数调用的正确使用不够熟悉，还有对C语言中经常出现的错误也不了解，通过实践，使我在这几个方面的认识有所提高。让自己有一定的能力去改正一些常见的错误语法，很高兴这一周的学习让我对数据结构（C语言描述）有了新的认识，所以后在学习过程中，我会更加注视实践操作，使自己便好地学好计算机。在这次课程设计的实验中，我收获了许多知识，通过查找大量资料，请教同学、学姐，以及不懈的努力，也培养了独立思考、动手操作的能力。我也学会了许多学习和解决实际问题的方法，让我受益匪浅。时间的紧缺成为一个很大的问题。也希望老师可以为我们指导一下以后的发展方向。如果可以让每个人都有动手焊接以及参与其他的各个流程，有专门的知道就更好了。课程设计对我来说，趣味性强，不仅锻炼能力，而且可以学到很多东西，在与老师和同学的交流过程中，互动学习，将知识融会贯通，也增强了我和同学之间的团队合作的能力。让我们知道只要努力，集中精力解决问题，一定会有收获的，过程也是很重要的。在这次课程设计中我们要学会利用时间，在规定的时间内完成我们的任务，要逐渐养成用C/C++语言编写程序的良好习惯。这些对我来说都是一种锻炼，一个知识积累的过程，一种能力的提高。要打好基础，才能用更好的办法，更简洁明了的程序解决实际问题，只有这样才能进一步的取得更好的成绩。我们会更加努力，努力的去弥补自己的缺点，发展自己的优点，去充实自己，只有在了解了自己的长短之后，我们会更加珍惜拥有的，更加努力的去完善它，增进它。