

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201706**

**学 号： U201714776**

**姓 名： 梁一飞**

**指导教师： 周时阳**

**报告日期： 2018年 11月 13 日**

**计算机科学与技术学院**

目录

[1 基于顺序存储结构的线性表实现 2](#_Toc533971837)

[1.1 问题描述 2](#_Toc533971838)

[1.2 系统设计 3](#_Toc533971839)

[1.3 系统实现 6](#_Toc533971840)

[1.4 实验小结 19](#_Toc533971841)

[2 基于链式存储结构的线性表实现 20](#_Toc533971842)

[2.1 问题描述 20](#_Toc533971843)

[2.2 系统设计 20](#_Toc533971844)

[2.3 系统实现 23](#_Toc533971845)

[2.4 实验小结 37](#_Toc533971846)

[3 基于二叉链表的二叉树实现 37](#_Toc533971847)

[3.1 问题描述 37](#_Toc533971848)

[3.2 系统设计 38](#_Toc533971849)

[3.3 系统实现 43](#_Toc533971850)

[3.4 实验小结 49](#_Toc533971851)

[4基于邻接表的图实现 49](#_Toc533971852)

[4.1 问题描述 49](#_Toc533971853)

[4.2 系统设计 50](#_Toc533971854)

[4.3 系统实现 53](#_Toc533971855)

[4.4 实验小结 60](#_Toc533971856)

[附录1 线性表程序实现清单 61](#_Toc533971857)

[附录2 链表程序实现清单 72](#_Toc533971858)

[附录3 二叉树程序实现清单 85](#_Toc533971859)

[附录4 图程序实现清单 104](#_Toc533971860)

# 1 基于顺序存储结构的线性表实现

### 1.1 问题描述

本次实验要求设计一个顺序表，实现初始化表（构造一个空的线性表）、销毁表（销毁线性表L）、清空表（将L重置为空表）、判定空表（若L为空表则返回TRUE,否则返回FALSE）、求表长（返回L中数据元素的个数）、获得元素（返回L中指定位置的数据元素的值）、查找元素（返回L中第1个与查找值相同的数据元素的位序）、获得前驱（获得指定元素的前驱元素的值）、获得后继（获得指定元素的后继元素的值）、插入元素（在指定位置插入一个元素）、删除元素（删除指定位置的元素）、遍历表（打印表中的所有元素）共12个操作，另外我加入了存入文件（将线性表以文件形式保存）、读取文件（将文件中的数据读取到线性表中）、切换表（操作多个线性表）这三个操作使线性表的功能更全面。

### 1.2 系统设计

**一.数据结构定义**

**线性表的数据结构用C语言定义为：**

typedef int status; //函数类型定义

typedef int ElemType; //数据元素类型定义

typedef struct{

ElemType \* elem;//元素

int length;//表长

int listsize;//表长上限

}SqList;

**各个宏定义：**

#define TRUE 1 （正确类型）

#define FALSE -1 （错误类型）

#define OK 1 （正确类型）

#define ERROR 0 （错误类型）

#define INFEASTABLE -1 （错误类型）

#define OVERFLOW -2 （错误类型）

#define LIST\_INIT\_SIZE 100 （长度类型）

#define LISTINCREMENT 10 （长度类型）

**二.算法设计**

（1）初始化表：

函数定义为status IntiaList(SqList \*L)，基本思想是先给表分配一个空间，然后判断是否溢出，若溢出则返回OVERFLOW，否则将表长初始化为0，将表长上限初始化为LIST\_INIT\_SIZE,返回OK。

输入：无。

输出：是否成功创建线性表。

时间复杂度：O（1）

（2）销毁表：

函数定义为status DestroyList(SqList \*L)，基本思想是先判断表是否初始化，若未初始化返回ERROR，若初始化则释放元素占用的存储空间，并将元素指向NULL，返回OK。

输入：当线性表L存在时执行。

输出：是否成功销毁线性表。

时间复杂度：O（1）

（3）清空表：

函数定义为status ClearList(SqList\* L)，基本思想是先判断表是否初始化，若未初始化返回ERROR，若初始化则将表长设为0，返回OK。

输入：无

输出：是否成功清空

时间复杂度：O（1）

（4）判定空表：

函数定义为status ListEmpty(SqList \*L)，基本思想是判断表长是否为0，是则返回TRUE，不是则返回ERROR

输入：无

输出：是否为空表

时间复杂度：O（1）

（5）求表长：

函数定义为int ListLength(SqList \*L)，基本思想是先判断表是否初始化，若未初始化返回ERROR，否则返回L->length。

输入：无

输出：线性表的长度

时间复杂度：O（1）

（6）获得元素：

函数定义为status GetElem(SqList\* L,int i,ElemType &e)，基本思想是获取用户输入的i，然后e指向L->elem+i-1，返回e

输入：位置i

输出：线性表中第i个位置的元素

时间复杂度：O（1）

（7）查找元素：

函数定义为int LocateElem(SqList \*L,ElemType e)，基本思想是先判断表是否初始化，若未初始化返回ERROR,若初始化则定义一个k=1，然后从表的第一个元素开始与用户要查找的元素e开始比较，若不是则移向下一个元素，k+1，当k大于表长时返回ERROR,在表长范围内找到与e相同的元素则返回k。

输入：元素e

输出：第一个等于元素e的元素的位置

时间复杂度：O（n）

（8）获得前驱：

函数定义为status PriorElem(SqList \*L,ElemType cue,ElemType \*pre)，基本思想是定义一个i=0，在i~表长的范围内进行循环，比较第i个元素与用户要查找的元素cue是否相同，每循环一次i加1,，当i=0或大于表长时循环结束，则返回FALSE，否则指针pre指向L->elem[i-1]，返回OK。

输入：元素cue

输出：cue元素的前一个元素

时间复杂度：O（n）

（9）获得后继：

函数定义为status NextElem(SqList\* L,ElemType cue,ElemType \*next)，大致思想同获得前驱，返回FALSE时i=表长或大于表长，否则指针next指向L->elem[i+1]，返回OK.

输入：元素cue

输出：元素cue的后一个元素

时间复杂度：O（n）

（10）插入元素：

函数定义为status ListInsert(SqList \*L,int i,ElemType e)，基本思想为先判断表是否初始化，若未初始化则返回ERROR，若初始化，则判断要插入的位置是否合法，不合法返回ERROR，合法则先给表分配更大的空间，然后将元素插到指定位置，并将后续元素全部后移一位。

输入：位置i，元素e

输出：操作是否成功

时间复杂度：O（n）

（11）删除元素：

函数定义为status ListDelete(SqList \*L,int i,ElemType\*e)，基本思想是先判断要删除的位置是否合法，若不合法则返回ERROR，若合法则将指定位置的元素删除，并将后续元素全部前移一位。

输入：位置i

输出：删除是否成功

时间复杂度：O（n）

（12）遍历表：

函数定义为status ListTrabverse(SqList \*L)，基本思想是先判断表是否初始化，若未初始化返回ERROR，否则定义一个i，在0~表长范围内循环打印表中所有元素。

输入：无

输出：遍历每个元素的操作

时间复杂度：O(n)

（13）存入文件：

函数定义为int StoreInFile(SqList \*L,char \* filename)，基本思想是定义一个文件指针fp，输入需要被写入的文件名，打开该文件，若无法打开，则返回ERROR，若打开成功，则将L写入文件中，关闭文件，返回存储成功。

输入：需要输入的文件名

输出：存储是否成功

时间复杂度：O(n)

（14）读取文件：

函数定义为int ReadFromFile(SqList \*L,char \* filename)，基本思想是定义一个文件指针fp，先销毁L,再重新初始化L，输入需要载入的文件名，打开该文件，若无法打开，则返回ERROR， 若打开成功，则读入L，关闭文件，返回读取成功。

输入：需要读取的文件名

输出：加载好的线性表L

时间复杂度:O（n）

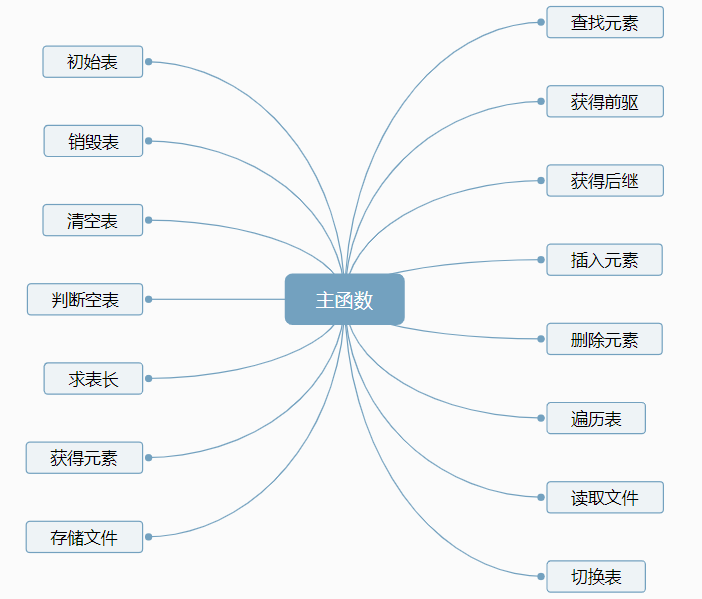
（15）切换表：

基本思想是定义了一个结构指针L，初始创建了三个表L1,L2,L3，当用户输入1时L指向L1，输入2时L指向L2，输入3时，L指向L3，输入其他时，提示输入错误。

### 1.3 系统实现

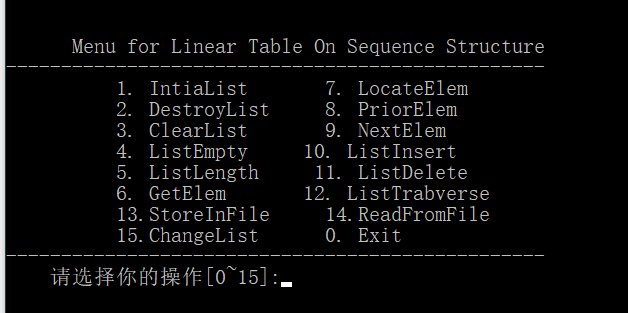
系统设计为

图1.3.1 系统设计图



系统界面为

图1.3.2--系统界面图



一．初始化表

(1)Codeing:

status IntiaList(SqList \* L){

L->elem = (ElemType \*)malloc( LIST\_INIT\_SIZE \* sizeof (ElemType));

if(!L->elem)

exit(OVERFLOW);

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

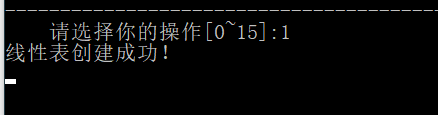
}

(2)Testing:

表1.3.1—线性表初始化测试

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 选择功能1 | 线性表创建成功 |
| 异常输入 | 无 | 无 |

图1.3.3—线性表的创建



二．销毁表

(1)Codeing:

status DestroyList(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

free(L->elem);

L->elem = NULL;

return OK;}

else

return ERROR;}

(2)Testing:

表1.3.2—线性表的销毁

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 选择功能2 | 线性表销毁成功（图1.3.4） |
| 异常输入 | 在未初始化时选择功能2 | 线性表销毁失败（图1.3.5） |

图1.3.4—线性表的销毁

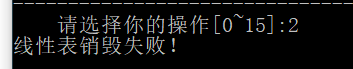
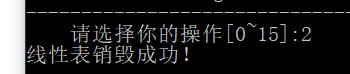


图1.3.5—线性表的销毁

三．清空表

(1)Codeing:

status ClearList(SqList\* L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

L->length = 0;

return OK;}

else

return ERROR;}

(2)Testing:

表1.3.3—线性表的清空

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 选择功能3 | 线性表清空成功（图1.3.6） |
| 异常输入 | 在未初始化的清空下选择功能3 | 线性表清空失败（图1.3.7） |

图1.3.6--线性表的清空

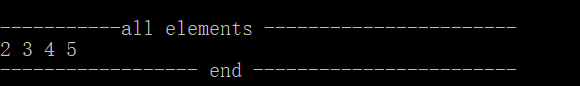
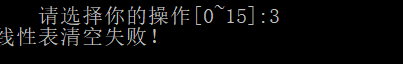
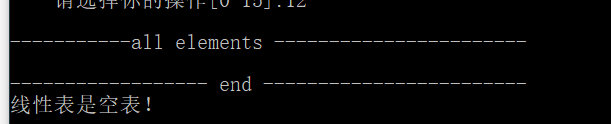


图1.3.7—线性表的清空



四．判定空表

(1)Codeing:

status ListEmpty(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if(L->length == 0)

return TRUE;

else

return FALSE;}

else

return ERROR;}

(2)Testing:

表1.3.4—判定空表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 1. 线性表无元素时选择功能4 2. 线性表有元素时选择功能4 | 1. 线性表已清空（图1.3.8） 2. 线性表未清空（图1.3.9） |
| 异常输入 | 线性表未初始化时选择功能4 | 线性表未初始化（图1.3.10） |

图1.3.8—判定空表



图1.3.9—判定空表

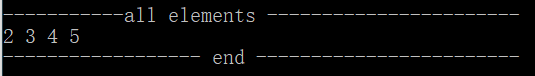
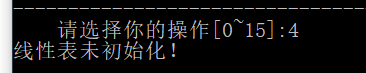


图1.3.10—判定空表



五．求表长

(1)Codeing:

int ListLength(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE))

return L->length;

else

return ERROR;}

(2)Testing:

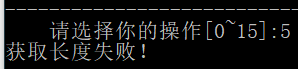
表1.3.5—求表长

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 元素4个时求表长 | 4（图1.3.11） |
| 异常输入 | 线性表未初始化时求表长 | 获取长度失败（图1.3.12） |

图1.3.10—求表长



图1.3.11—求表长



六．获得元素

(1)Codeing:

status GetElem(SqList\* L,int i,ElemType &e){

e=\*(L->elem+i-1);

return e;}

(2)Testing:

表1.3.6—获得元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 1. 获得第3个元素 2. 获得第1个元素 | 1.4(图1.3.13）  2.2(图1.3.14) |
| 异常输入 | 1.获得第8个元素  2.表未初始化时获得第1个元素 | 1. 获取失败(图1.3.15) 2. 获取失败(图1.3.16) |

图1.3.13—获得元素

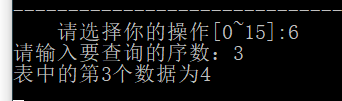


图1.3.14—获得元素

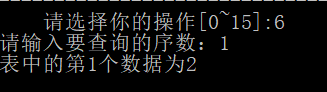


图1.3.15—获得元素

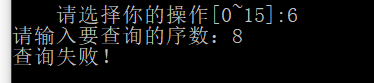
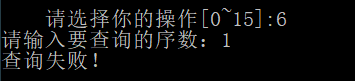


图1.3.16—获得元素



七．查找元素

(1)Codeing:

int LocateElem(SqList \*L,ElemType e){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

int k=1;

while(\*L->elem != e){

L->elem++;

k++;

if(k >L->length){

return FALSE;

}

}

return k;

}

else

return ERROR;}

(2)Testing:

表1.3.7—查找元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 1. 查找元素2 2. 查找元素5 | 1. 位序为1（图1.3.17） 2. 位序为4（图1.3.18） |
| 异常输入 | 1. 查找元素9 2. 在未初始化时查找元素2 | 1. 不存在（图1.3.19） 2. 查询失败（图1.3.20） |

图1.3.17—查找元素

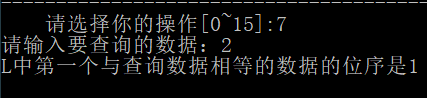


图1.3.18—查找元素

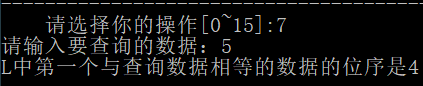


图1.3.19—查找元素

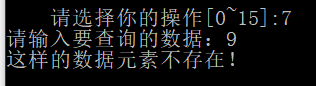
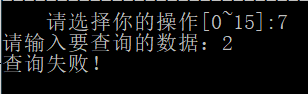


图1.3.20—查找元素



八．获得前驱

(1)Codeing:

status PriorElem(SqList \*L,ElemType cue,ElemType \*pre){

int i;

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

if(L->elem[i] == cue){

if(i==0)

return FALSE;

\*pre=L->elem[i-1];

return OK;

}

}

return FALSE;}

(2)Testing:

表1.3.8—获得前驱

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 1. 查找3的前驱 2. 查找5的前驱 | 1.2（图1.3.21）  2.4（图1.3.22） |
| 异常输入 | 1. 查找2的前驱 2. 查找8的前驱 3. 表未初始化时查找前驱 | 1. 无此前驱（图1.3.23） 2. 无此前驱（图1.3.24） 3. 查找失败（图1.3.25） |

图1.3.21—获得前驱

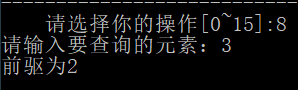


图1.3.22—获得前驱

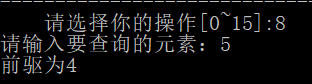


图1.3.23—获得前驱

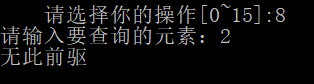


图1.3.24—获得前驱

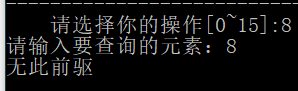
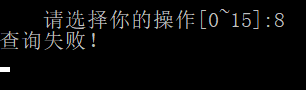


图1.3.25—获得前驱



九．获得后继

(1)Codeing:

status NextElem(SqList\* L,ElemType cue,ElemType \*next){

int i;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

for(i=0;i<L->length-1;i++){

if(L->elem[i] == cue){

if(i==L->length-1)

return FALSE;

\*next = (int)L->elem[i+1];

return OK;

}

}

return FALSE;

}

else

return ERROR;}

(2)Testing:

表1.3.9—获得后继

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中元素为2，3，4，5） | 1. 获得2的后继 2. 获得4的后继 | 1.3（图1.3.26）  2.5（图1.3.27） |
| 异常输入 | 1. 获得5的后继 2. 获得9的后继 3. 表未初始化时获得后继 | 1. 无此后继（图1.3.28） 2. 无此后继（图1.3.29） 3. 查询失败（图1.3.30） |

图1.3.26—获得后继

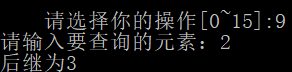


图1.3.27—获得后继

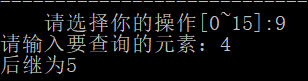


图1.3.28—获得后继

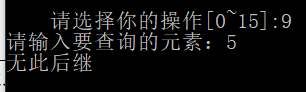


图1.3.29—获得后继

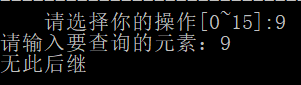
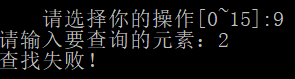


图1.3.30—获得后继



十．插入元素

(1)Codeing:

status ListInsert(SqList \*L,int i,ElemType e){

ElemType \*f,\*t,\*p;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if(!L->elem)

return ERROR;

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

return ERROR;

if(L->length >= L->length+1)

return ERROR;

if(L->length >= L->listsize){

f = (ElemType \*)realloc(L->elem,(L->listsize +LISTINCREMENT)\*sizeof(ElemType));

if(!f)

return ERROR;

L->elem = f;

L->listsize += LISTINCREMENT;

}

t =&(L->elem[i-1]);

for(p=&(L->elem[L->length-1]);p>=t;p--){

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

}

\*t=e;

++L->length;

return OK;

}

else

return ERROR;}

(2)Testing:

表1.3.10—插入元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 分别插入1，2，3 | 1，2，3（图1.3.31） |
| 异常输入 | 1. 在位置8插入5 2. 未初始化时插入2 | 1. 插入失败（图1.3.32） 2. 插入失败（图1.3.33） |

图1.3.31—插入元素

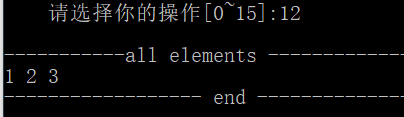
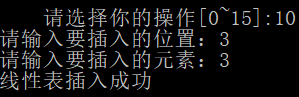
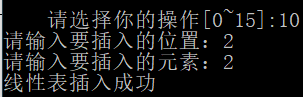
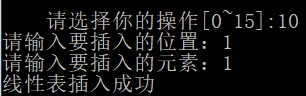


图1.3.32—插入元素

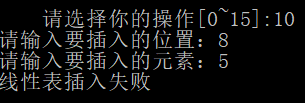
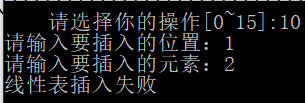


图1.3.33—插入元素



十一.删除元素

1. Codeing:

status ListDelete(SqList \*L,int i,ElemType\*e){

ElemType \*t,\*p;

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

return ERROR;

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

e=p;

t=&(L->elem[L->length-1]);

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

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

--L->length;

return OK;}

(2)Testing:

表1.3.11—删除元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据2，3，4，5） | 先删除第一个元素，再删除第二个元素 | 3，5（图1.3.34） |
| 异常输入 | 1. 删除第7个元素 2. 未初始化时删除第一个元素 | 1. 删除失败（图1.3.35） 2. 删除失败（图1.3.36） |

图1.3.34—删除元素

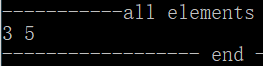
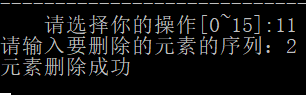
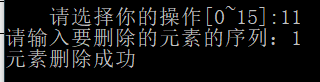


图1.3.35—删除元素

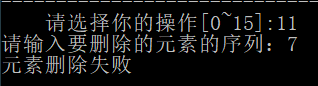
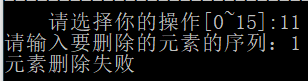


图1.3.36—删除元素



十二.遍历表

(1)Codeing:

status ListTrabverse(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

int i;

printf("\n-----------all elements -----------------------\n");

for(i=0;i<L->length;i++) printf("%d ",L->elem[i]);

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

return L->length;

}

else

return ERROR;}

(2)Testing:

表1.3.12—遍历表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中元素2，3，4，5） | 选择功能12 | 2，3，4，5（图1.3.37） |
| 异常输入 | 未初始化时选择功能12 | 线性表是空表（图1.3.38） |

图1.3.37—遍历表

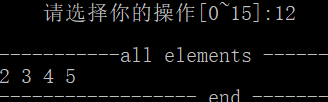
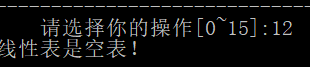


图1.3.38—遍历表



十三.存入文件

1. Codeing：

int StoreInFile(SqList \*L,char \* filename){

FILE \*fp;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if ((fp=fopen(filename,"w"))==NULL){

printf("File open error\n ");}

fwrite(L->elem,sizeof(ElemType),L->length,fp);

fclose(fp);

return OK;}

else

return ERROR;}

1. Testing:

表1.3.13—存入文件

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 将数据2，3，4，5存入文件list1 | 生成文件list1（图1.3.39） |
| 异常输入 | 在未初始化时存入文件 | 未初始化（图1.3.40） |

图1.3.39—存入文件

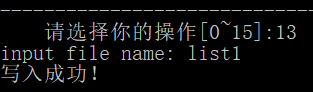
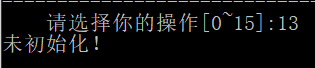


图1.3.40—存入文件



十四.读取文件

Codeing:

int ReadFromFile(SqList \*L,char \* filename){

L->length=0;

FILE \*fp;

if (L->length<=LIST\_INIT\_SIZE){

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

printf("File open error\n ");

return 1;}

while(fread(&(L->elem[L->length]),sizeof(ElemType),1,fp)){

L->length++;}

fclose(fp);

return OK;}}

(2)Testing:

表1.3.14—读取文件

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 读取文件list1 | 显示2，3，4，5（图1.3.41） |
| 异常输入 | 1. 在未初始化时读取文件 2. 读取文件list99 | 1.未初始化（图1.3.42）  2.文件打开失败（图1.3.43） |

图1.3.41—读取文件

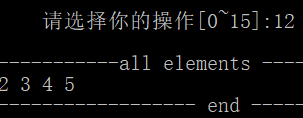
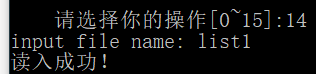


图1.3.42—读取文件

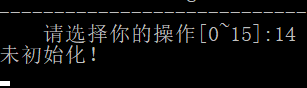
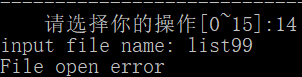


图1.3.43—读取文件



十五.切换表

Codeing:

SqList L1,L2,L3,\*L=&L1;

int m;

printf("请输入你要操作的表[1-3]:");

scanf("%d",&m);

if(m==1){

L=&L1;

printf("切换成功\n");

getchar();getchar();}

else if(m==2){

L=&L2;

printf("切换成功\n");

getchar();getchar();}

else if(m==3){

L=&L3;

printf("切换成功\n");

getchar();getchar();}

else {

printf("切换失败！\n");

getchar();getchar();}

(2)Testing:

表1.3.15—切换表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 从表1切换到表2 | 切换成功（图1.3.44） |
| 异常输入 | 切换到表4 | 切换失败（图1.3.45） |

图1.3.44—切换表

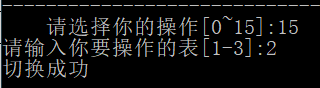
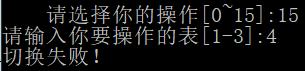


图1.3.45—切换表



### 1.4 实验小结

通过本次实验，我掌握了很多指针操作的细节，第一做完时没有操作多个表的功能，于是我加了一个指针来操作多个表，但是由于参数不一致，导致其他函数都用不了了，修改了好长时间才改好，这次实验也让我明白了做好注释和规范coding的重要性，因为一开始没做好注释而且coding格式不规范，导致在debug时遇到了很多的麻烦，耽误了很多时间。做这个线性表也是一个循序渐进的过程，一开始只有基本的12个功能，而且各个功能也没有异常情况判断，完善这些判断的过程让我进步了很多。

2 基于链式存储结构的线性表实现

### 2.1 问题描述

本次实验需要实现线链表的12种功能，分别为：初始化表（构造一个空的链表L）、销毁表（销毁链表L）、清空表（将L重置为空表）、判定空表（判断L是否为空表、求表长（返回L中数据元素的个数）、获得元素（返回L中指定位置的元素的值）、查找元素（返回L中指定元素的位置）、获得前驱（返回指定元素的前一个元素）、获得后继（返回指定元素的后一个元素）、插入元素（在指定位置插入元素）、删除元素（删除指定位置的元素）、遍历表（打印表中所有元素）。为使功能齐全，我另外加入了3个功能，分别是存入文件（将表中数据存储到文件中），读取文件（将文件中的数据读取到表中），切换表（同时操作多个表）。

### 系统设计

**一.数据结构定义**

**链表的数据结构用C语言定义为：**

typedef struct Node{ //单链表的定义

ElemType elem;//元素

struct Node \*next;//指向下一个结点的指针

}Node//链表节点,\*pNode//链表节点指针;

**各个宏定义：**

#define TRUE 1 （正确类型）

#define FALSE -1 （错误类型）

#define OK 1 （正确类型）

#define ERROR 0 （错误类型）

#define INFEASTABLE -1 （错误类型）

#define OVERFLOW -2 （错误类型）

typedef int status;//函数类型定义

typedef int ElemType; //数据元素类型定义

**二.算法设计**

（1）初始化表：

函数定义为status IntiaList(pNode &L)；基本思想是：函数传入头结点指针的引用，释放其储存空间，并给它分配新的存储空间/指针的指针域指向NULL，数据域设置为0。头结点的数据域用于存储链表中元素的个数。

输入：无

输出：是否成功创建链表

时间复杂度：O(1)

（2）销毁表：

函数定义为status DestroyList(pNode & L);基本思想是：函数传入头结点的引用。如果链表内元素为0，则直接释放头结点储存空间，如果链表元素不为0，则依次释放链表内的所有元素。

输入：无

输出：是否成功销毁链表

时间复杂度：O(1)

（3）清空表

函数定义为status ClearList(pNode &L);基本思想是：与销毁列表类似，不同的是，清空链表操作保留了头结点,转而从第二个结点开始对结点进行释放存储空间的处理。在清空完成后，把头结点L->elem置为0，表示链表内的元素为空。

时间复杂度：O（1）

（4）判断表空

函数定义为status ListEmpty(pNode L);基本思想是：运用L->elem的值进行判断。若数值为0，则表示表内不存在元素，即链表为空。若数值不为0，则不为空。

（5）求表长

函数定义为int ListLength(pNode L);基本思想是：头结点中储存了链表的长度信息，直接返回L->elem的数值即可。

时间复杂度：O（1）

（6）获得元素

函数定义为status GetElem(pNode L,int seq,ElemType & e);基本思想是：首先判断序号seq的范围是否合理，若不合理，返回ERROR；如果合理，则继续运算。设置一个计数器k以及一个指向头结点的指针，顺指针向后查找，直到指向第seq个元素为止，取第seq个元素。

时间复杂度：O（n）

（7）查找元素

函数定义为status LocateElem(pNode L,ElemType e);基本思想是：设置一个指向第一个元素的指针和一个计数器k，并顺指针向后查找，并对每个结点与e进行匹配判断，若匹配成功，则返回所计的数k。若未找到，则返回0。

时间复杂度：O（n）

（8）获取前驱

函数定义为status PriorElem(pNode L,ElemType cue,ElemType &pre);基本思想是：首先对L->elem的值进行判定。若L->elem==0，链表为空，返回FALSE。设置一个计数器k,并设置两个指针Lp，Lq。其中Lp初始指向L。Lp顺指针向后查找，Lq一直指向Lp的前驱，若Lp指向的结点与cue相等，且Lp不为第一个元素，则给pre赋上当前所指指针的值，并返回OK，若为第一个结点，则返回ERROR。若未能找到，则返回FALSE。

时间复杂度：O（n）

（9）获取后继

函数定义为status NextElem(pNode L,ElemType cue,ElemType & next);基本思想是：与获取前驱类似，不同的是找到相等的元素后，判断Lp是否为最后一个结点，若不为最后一个结点，则给next附上该结点后继结点的值，并返回OK。若为最后一个结点，则返回ERROR。若未能找到，则返回FALSE。

时间复杂度O（n）

（10）插入元素

函数定义为status ListInsert(pNode & L,int seq,ElemType e);基本思想是：先判断seq是否符合条件；若seq<1或大于表长加1，则返回ERROR。设置一个新节点newNode,并为它分配空间。与此同时，设置一个指向头指针的结点p和一个指向p的前驱的指针q，并设置一个计数器k。顺指针进行移动，知道p指向第seq个元素。此时q指向p的前驱。将q的后继指向新元素，并将新元素指向p即可完成插入操作。最后，让头结点的元素记录数加上1，表示链表中增加了一个元素。

时间复杂度：O（n）

（11）删除元素

函数定义为status ListDelete(pNode & L,int seq,ElemType &e);基本思想同插入元素，不同的是用q指向p的后继，并让p的后继指向q的后继。接着，释放q所指向结点的存储空间，最后，让头结点的元素记录数减去1，表示链表中删除了一个元素。

时间复杂度：O（n）

（12）遍历链表

函数定义为status ListTrabverse(pNode L); 基本思想是：首先对链表进行判定，看是否为空。若为空，提示链表为空。若不为空，设置一个指向头结点的指针P。让p顺指针逐个移动，并逐个打印每个节点中的元素，知道p为空。

时间复杂度：O（n）

（13）文件存储

函数定义为status StoreInFile(pNode L,char \* filename);基本思想是：首先设置一个指向头结点的指针，打开文件，让p顺指针逐个移动，并向文件中逐一存入结点数据，直至p为空。最后，关闭文件。

时间复杂度：O（n）

（14）文件读取

函数定义为status ReadFromFile(pNode & L,char \* filename);基本思想是：首先新建链表L,并设置一个指向头结点的指针p。打开文件，给p的后继分配存储空间，并将文件中的一个结点数据写入p的后继。让p顺指针逐个移动，并将数据一一对应地写入p的后继。每写入一个数据，头结点中记录表长的变量就自增1。待文件读到末尾后，关闭文件，即载入链表数据成功。

时间复杂度：O（n）

（15）切换链表

基本思想是：声明了一个指针的指针L，让用户输入1~3选择要操作的表L1~L3，L指向结构指针L1~L3，从而达到操作多个表的操作。

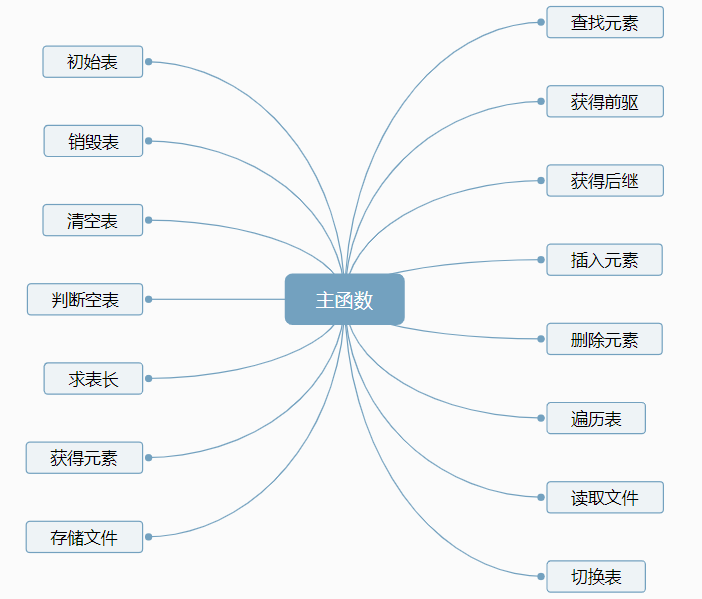
（16）补充说明

每个函数操作我都加入了判断表是否初始化的操作，不知是何原因导致L!=NULL的判断语句陷入死循环，于是我定义了一个变量Initial=0表示未初始化，在初始化函数里将Initial改为1表示初始化，在销毁链表函数里将Initial改为0表示未初始化，每个函数前都加入判断if(!Initial)。

### 系统实现

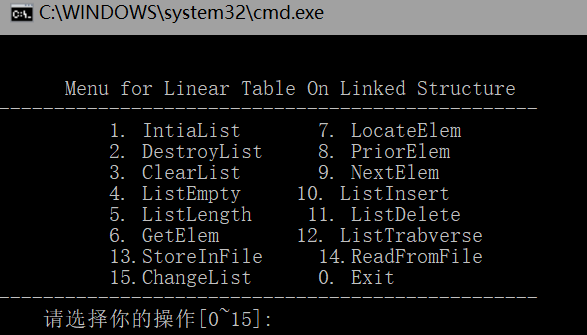
系统设计为

图2.3.1—系统设计图



系统界面为

图2.3.2—系统界面图



一．初始化表

Coding:

status IntiaList(pNode& L){

L=(pNode)malloc(2\*sizeof(pNode));

L->elem=0;

L->next=NULL;

return OK;

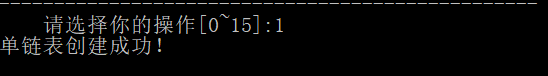
}//初始化表

Testing:

表1.3.1—初始化表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 选择功能1 | 线性表创建成功（图2.3.3） |
| 异常输入 | 无 | 无 |

图2.3.3—初始化表



二．销毁表

Coding:

status DestroyList(pNode &L){

pNode Lp;

if(L->elem==0){

free(L);

return OK;

}

while(L->next!=NULL){

Lp=L;

L=L->next;

free(Lp);

}

free(L);

return OK;

}//销毁表

Testing:

表1.3.2—销毁表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 选择功能2 | 线性表销毁成功（图2.3.4） |
| 异常输入 | 在未初始化时选择功能2 | 请先初始化（图2.3.5） |

图2.3.4—销毁表

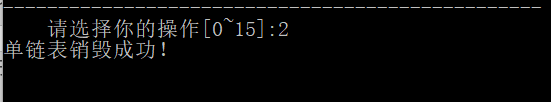
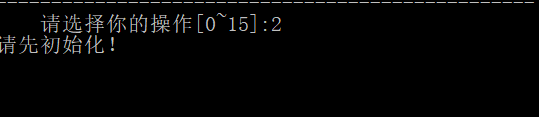


图2.3.5—销毁表



三．清空表

Coding:

status ClearList(pNode & L){

pNode Lp,Lq;

if(L->elem==0)

return ERROR;

Lp=L->next;

while(Lp->next!=NULL){

Lq=Lp;

Lp=Lp->next;

free(Lq);

}

free(Lp);

L->elem=0;

return OK;

}//清空表

Testing:

表1.3.3—清空表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 1. 在空表的情况下选择功能3 2. 在有元素的情况下选择功能3 | 线性表清空成功（图2.3.6）  线性表清空失败（图2.3.7） |
| 异常输入 | 在未初始化的清空下选择功能3 | 请先初始化（图2.3.8） |

图2.3.6—清空表

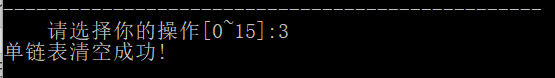


图2.3.7—清空表

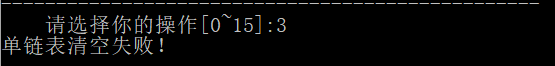
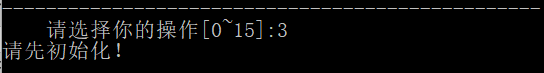


图2.3.8—清空表



四．判断表空

Coding:

status ListEmpty(pNode L){

if(L->elem == 0)

return TRUE;

else

return FALSE;

}//判定空表

Testing:

表2.3.4—判断表空

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 1.线性表无元素时选择功能4  2.线性表有元素时选择功能4 | 1. 线性表已清空（图2.3.9） 2. 线性表未清空（图2.3.10） |
| 异常输入 | 线性表未初始化时选择功能4 | 请先初始化（图2.3.11） |

图2.3.9—判断表空

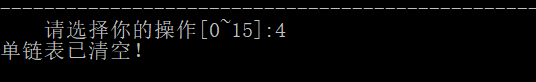


图2.3.10—判断表空

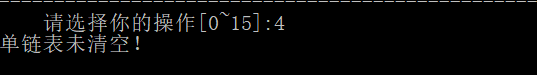
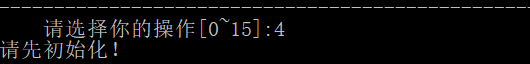


图2.3.11—判断表空



五．求表长

Coding:

int ListLength(pNode L){

return L->elem;

} //求表长

Testing:

表2.3.5—求表长

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 元素4个时求表长 | 4（图2.3.12） |
| 异常输入 | 线性表未初始化时求表长 | 请先初始化（图2.3.13） |

图2.3.12—求表长

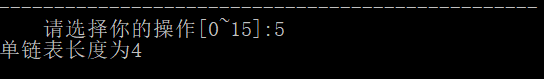
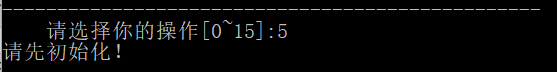


图2.3.13—求表长



六．获得元素

Coding:

status GetElem(pNode L,int i,ElemType &e){

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

return ERROR;

pNode Lp=L;

int k;

for(k=0;k<i;k++){

Lp=Lp->next;

}

e=Lp->elem;

return OK;

}//获得元素

Testing:

表2.3.6—获得元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 获得第3个元素 | 1. 4(图2.3.14） |
| 异常输入 | 1. 获得第8个元素  2. 表未初始化时获得第1个元素 | 1. 获取失败(图2.3.15) 2. 请先初始化(图2.3.16) |

图2.3.14—获得元素

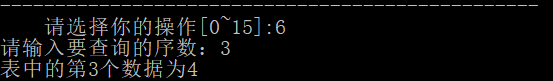


图2.3.15—获得元素

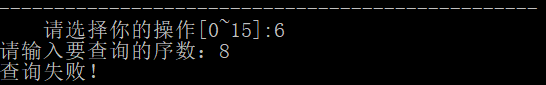
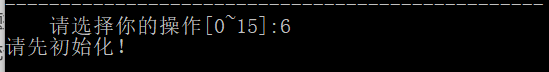


图2.3.16—获得元素



七．查找元素

Coding:

int LocateElem(pNode L,ElemType e){

int k=0;

pNode Lp=L->next;

while(Lp!=NULL){

k++;

if(Lp->elem==e){

return k;

}

Lp=Lp->next;

}

return ERROR;

}//查找元素

Testing:

表2.3.7—查找元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 1.查找元素2  2.查找元素5 | 1. 位序为1（图2.3.17）  2. 位序为4（图2.3.18） |
| 异常输入 | 1. 查找元素9 2. 在未初始化时查找元素2 | 1. 不存在（图2.3.19）  2. 请先初始化（图2.3.20） |

图2.3.17—查找元素

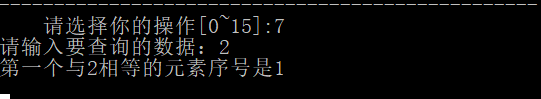


图2.3.18—查找元素

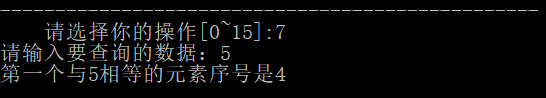


图2.3.19—查找元素

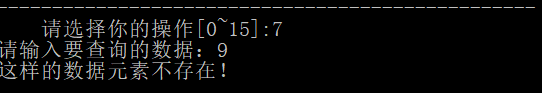
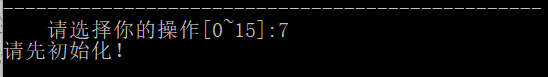


图2.3.20—查找元素



八．获得前驱

Coding:

status PriorElem(pNode L,ElemType cue,ElemType &pre){

if(!L->elem){

return FALSE;

}

int k=0;

pNode Lp,Lq;

Lp=L;

while(Lp!=NULL){

k++;

Lq=Lp;

Lp=Lp->next;

if(Lp==NULL)

return FALSE;

if(cue==Lp->elem&&k!=1){

pre=Lq->elem;

return OK;

}

}

return FALSE;

}//获得前驱

Testing:

表2.3.8—获得前驱

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据为2，3，4，5） | 1. 查找3的前驱  2. 查找5的前驱 | 1. 2（图2.3.21）  2. 4（图2.3.22） |
| 异常输入 | 1. 查找2的前驱 2. 查找8的前驱 3. 表未初始化时查找前驱 | 1. 无此前驱（图2.3.23） 2. 无此前驱（图2.3.24） 3. 请先初始化（图2.3.25） |

图2.3.21—获得前驱

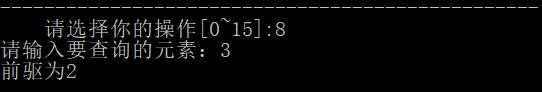


图2.3.22—获得前驱

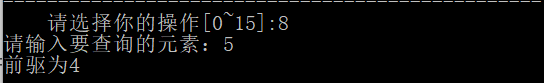


图2.3.23—获得前驱

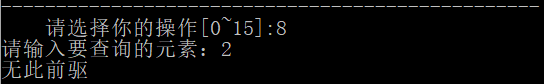


图2.3.24—获得前驱

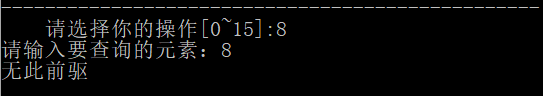
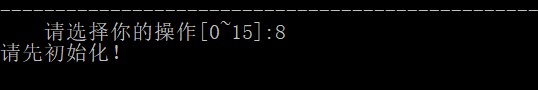


图2.3.25—获得前驱



九．获得后继

Coding:

status NextElem(pNode L,ElemType cue,ElemType &next){

if(!L->elem)

return FALSE;

pNode Lp=L;

while(Lp->next!=NULL){

Lp=Lp->next;

if(cue==Lp->elem){

if(Lp->next==NULL)

return ERROR;

next=Lp->next->elem;

return OK;}

}

return FALSE;

}//获得后继

Testing:

表2.3.9—获得后继

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中元素为2，3，4，5） | 1. 获得2的后继 2. 获得4的后继 | 1.3（图2.3.26）  2.5（图2.3.27） |
| 异常输入 | 1. 获得5的后继 2. 获得9的后继 3. 表未初始化时获得后继 | 1. 查找失败（图2.3.28） 2. 无此后继（图2.3.29） 3. 请先初始化（图2.3.30） |

图2.3.26—获得后继

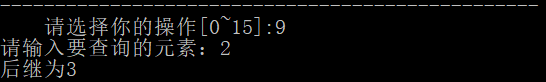


图2.3.27—获得后继

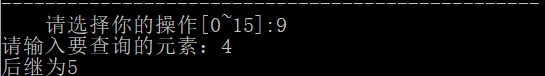


图2.3.28—获得后继

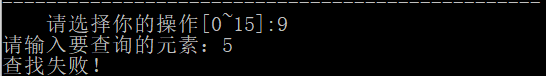


图2.3.29—获得后继

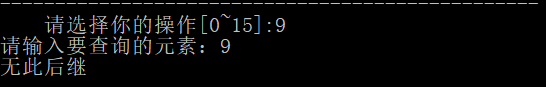
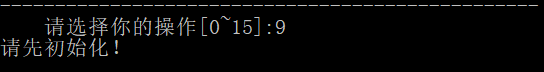


图2.3.30—获得后继



十．插入元素

Coding:

status ListInsert(pNode &L,int i,ElemType e){

if(i<1||i>L->elem+1)

return ERROR;

pNode p=L,q,newNode;

newNode=(pNode)malloc(2\*sizeof(pNode));

newNode->elem=e;

int k=0;

while(p!=NULL&&k<i){

q=p;

p=p->next;

k++;}

q->next=newNode;

newNode->next=p;

L->elem=L->elem+1;

return OK;

}//插入元素

Testing:

表2.3.10—插入元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 插入1 | 插入成功（图2.3.31） |
| 异常输入 | 1. 在位置8插入5 2. 未初始化时插入 | 1. 插入失败（图2.3.32） 2. 请先初始化（图2.3.33） |

图2.3.31—插入元素

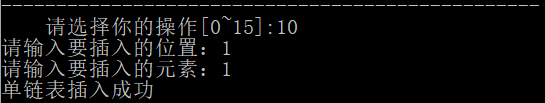


图2.3.32—插入元素

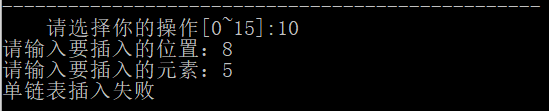
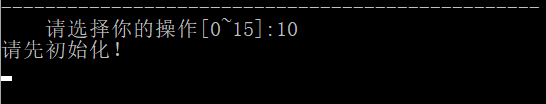


图2.3.33—插入元素



十一. 删除元素

Coding:

status ListDelete(pNode & L,int i,ElemType &e){

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

return ERROR;

pNode p=L,q;

int k=0;

while(p!=NULL&&k<i-1){

p=p->next;

k++;

}

q=p->next;

p->next=q->next;

e=q->elem;

free(q);

L->elem=L->elem-1;

return OK;

}//删除元素

Testing:

表2.3.11—删除元素

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中数据2，3，4，5） | 删除第一个元素 | 删除成功（图1.3.34） |
| 异常输入 | 1. 删除第7个元素 2. 未初始化时删除元素 | 1. 删除失败（图1.3.35） 2. 请先初始化（图1.3.36） |

图2.3.34—删除元素

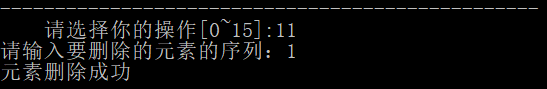


图2.3.35—删除元素

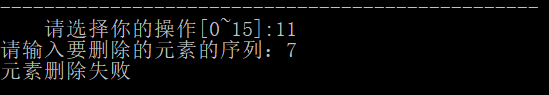
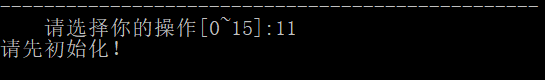


图2.3.36—删除元素



十二. 遍历表

Coding:

status ListTrabverse(pNode L){

if(L->elem==0)

return ERROR;

pNode p=L->next;

printf("\n-----------all elements -----------------------\n");

while(p!=NULL){

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

p=p->next;

}

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

return OK;

}//遍历表

Testing:

表2.3.12—遍历表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入（表中元素2，3，4，5） | 选择功能12 | 2，3，4，5（图2.3.37） |
| 异常输入 | 未初始化时选择功能12 | 请先初始化（图2.3.38） |

图2.3.37—遍历表

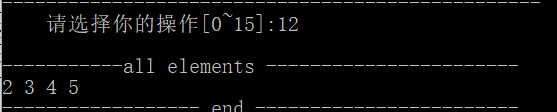
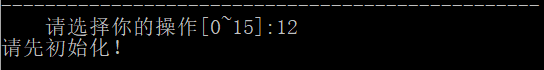


图2.3.38—遍历表



十三. 存储文件

Coding:

status StoreInFile(pNode L,char \* filename){

if(L->elem==0)

return ERROR;

pNode p=L->next;

FILE \*fp;

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return ERROR;

}

while(p!=NULL){

fwrite(p,2\*sizeof(pNode),1,fp);//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素并写入到文件中

p=p->next;

}

fclose(fp);

return OK;

}//存储文件

Testing:

表2.3.13—存储文件

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 将数据2，3，4，5存入文件test1 | 生成文件list1（图2.3.39） |
| 异常输入 | 在未初始化时存入文件 | 请先初始化（图2.3.40） |

图2.3.39—存入文件

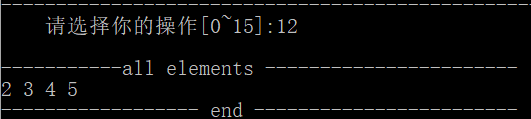
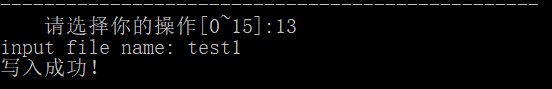
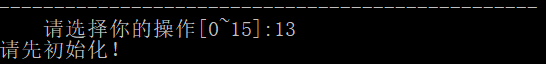


图2.3.40—存入文件



十四. 读取文件

Coding:

status ReadFromFile(pNode &L,char \* filename){

ClearList(L);

L->elem=0;

pNode p=L;

FILE \*fp;

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return ERROR;

}

Node temp;

while(fread(&temp,2\*sizeof(pNode),1,fp)){

p->next=(pNode)malloc(2\*sizeof(pNode));

p=p->next;

\*p=temp;

L->elem++;

}

p->next=NULL;

fclose(fp);

return OK;

}//读取文件

Testing:

表2.3.14—读取文件

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 读取文件test1 | 显示2，3，4，5（图2.3.41） |
| 异常输入 | 1. 在未初始化时读取文件  2. 读取文件test99 | 1.未初始化（图2.3.42）  2.请先初始化（图2.3.43） |

图2.3.41—读取文件

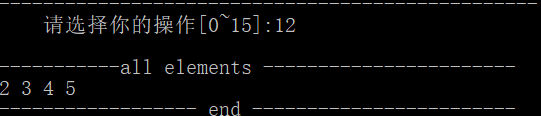
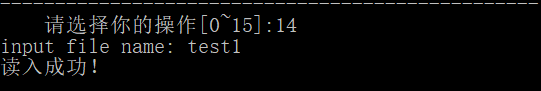


图2.3.42—读取文件

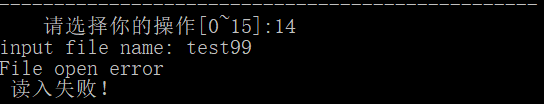
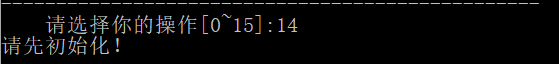


图2.3.43—读取文件



十五. 切换表

Coding:

int m;

printf("请输入要操作的链表[1~3]：");

scanf("%d",&m);

if(m==1){

L=&L1;

printf("切换成功！\n");

if(!L1)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m==2){

L=&L2;

printf("切换成功！\n");

if(!L2)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m==3){

L=&L3;

printf("切换成功！\n");

if(!L3)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m<1||m>3){

printf("切换失败！\n");

getchar();getchar();

}

Testing:

表2.3.15—切换表

|  |  |  |
| --- | --- | --- |
|  | 输入 | 理论结果 |
| 正常输入 | 从表1切换到表2 | 切换成功（图1.3.44） |
| 异常输入 | 切换到表4 | 切换失败（图1.3.45） |

图2.3.44—切换表

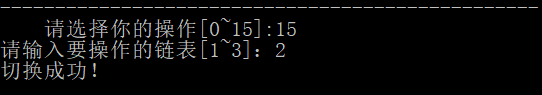
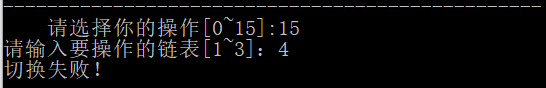


图2.3.45—切换表



### 2.4 实验小结

本次实验让我熟悉了链表的基本操作，在写判断链表初始化时的条件时，一开始我写的if(!L)，但不知是何原因导致所有函数都会进入死循环，于是我设置了一个变量Initial=0，初始化后将Initial改为1，销毁后将Initial改回0，然后将所有的判断改为if(!Initial)，于是所有功能得以照常使用，然而在切换表后没有修改Initial的值，所以即使表2表3未初始化也会导致函数判定已初始化而出错，于是在切换表的时候，我又加了三个判断if(!L1) Initial=1 ;else Initial=0;于是改好了这个错误，虽然不明白为何if(!L)会进入死循环，但是在修改过程中能感觉的此错误出现的原因是因为L是一个指针的指针，而L1，L2，L3都是指针，if(!L1)不会出现问题。通过这次实验，我掌握了链表的基本功能。

3 基于二叉链表的二叉树实现

### 问题描述

本次实验需要实现二叉树的二十种功能，分别为：初始化二叉树（构造空二叉树T）、销毁二叉树（销毁二叉树T）、创建二叉树（按definition构造二叉树T）、清空二叉树（将二叉树T清空）、判定空二叉树（若T为空二叉树则返回TRUE，否则返回FALSE）、求二叉树深度（返回T的深度）、获得根结点（返回T的根）、获得结点（返回结点的值）、结点赋值（给某个结点赋值）、获得双亲结点（返回结点的双亲结点指针）、获得左孩子结点（返回结点的左孩子结点指针）、获得右孩子结点（返回结点的右孩子结点指针）、获得左兄弟结点（返回结点的左兄弟结点指针）、获得右兄弟结点（返回结点的右兄弟结点指针）、插入子树（给结点插入一个左子树或者右子树）、 删除子树（删除某个结点的左子树或右子树）、

前序遍历（按前序遍历树的所有结点）、中序遍历（按中序遍历树的所有结点）、

后序遍历（按后序遍历树的所有结点）、按层遍历（按层遍历树的所有结点）。另外根据实际需要，我加入了：存入文件（将二叉树存入文件）、读取文件（从文件中读取文件），切换树（操作多个二叉树）这三个功能。

### 3.2 系统设计

**一．数据结构定义**

**二叉树数据结构用C语言定义为：**

typedef struct node{

Elemtype data;

struct node \*lchild ,\* rchild;

}Tnode,\*Bitree;

**其中每个结点的数据结构用C语言定义为：**

typedef struct {

int key ;

int value ;

} Elemtype;

**存储二叉树的线性表用C语言定义为：**

typedef struct {

int \* elem;

int length;;

}SqList;

**各个宏定义：**

#define OK 1

#define ERROR 0

//数据元素类型定义

typedef int status;

**二．算法设计**

（1）初始化二叉树

函数定义为status InitBiTree(Bitree &T)，基本思想是先为树T分配空间，然后让左右子树指针指向NULL。

输入：无

输出：是否成功初始化二叉树

时间复杂度：O（1）

（2）销毁二叉树

函数定义为status DestroyBiTree(Bitree &T)，基本思想是首先判断二叉树是否存在，若存在则递归调用函数销毁左右子树，销毁过程是先释放结点空间，然后令结点=NULL。

输入：无

输出：是否成功销毁二叉树

时间复杂度：O（n）

（3）创建二叉树

函数定义为status CreateBiTree(Bitree & T,int \* p)，基本思想是先为结点分配空间，指针p记录要创建结点的标记和值，分别读入结点，递归创建各结点，当p值为-1时，该节点为空。

输入：各结点的标记和值

输出：是否创建二叉树

时间复杂度：O(n)

（4）清空二叉树

函数定义为status ClearBiTree(Bitree T)，基本思想同销毁二叉树，在销毁二叉树后重新初始化了二叉树。

输入：无

输出：是否清空二叉树

时间复杂度：O(n)

(5)判断空树

函数定义为status BiTreeEmpty(Bitree T)，基本思想是判断头结点的左右子树是否为空，若都为空则为空树，否则不为空树。

输入：无

输出：是否为空树

时间复杂度：O（1）

（6）求二叉树深度

函数定义为status BiTreeDepth(Bitree T,int j,int &x)，基本思想是设立两个计数器，j和x，若结点不为空则j+1，若j>x，则x=j。递归调用求树的深度，返回x。

输入：无

输出：二叉树的深度

时间复杂度：O（n）

（7）求根结点

函数定义为status Root(Bitree T)，基本思想是直接输出根节点的值。

输入：无

输出：根结点的值

时间复杂度：O（1）

（8）求结点

函数定义为Tnode \* Value(Bitree T,int e)，首先对根节点进行判断，如果根节点不是要求的结点，则递归调用求左右子树。

输入：要求结点标记

输出：结点的值

时间复杂度：O(n)

(9)给结点赋值

函数定义为status Assign(Bitree T,int e,int d)，基本思想是调用Value（T，e）找到该结点，然给改变其值。

输入：要求结点标记以及要赋的值

输出：是否成功赋值

时间复杂度:O(n)

(10)求双亲结点

函数定义为Tnode \* Parent(Bitree T,int e)，基本思想是先判断该结点是否存在，否则从根节点递归调用找到孩子满足改结点的结点，返回找到的双亲结点。

输入：要求双亲结点的结点

输出：该节点的双亲结点

时间复杂度：O(n)

(11)求左孩子结点

函数定义为Tnode \* LeftChild(Bitree T,int e)，基本思想是先调用Value（T，e）找到该结点，然后返回该节点的左孩子结点

输入：要求左孩子结点的结点

输出：该结点的左孩子结点

时间复杂度:O（n）

（12）求右孩子结点

函数定义为Tnode \* RightChild(Bitree T,int e)，基本思想同求左孩子结点

输入：要求右孩子结点的结点

输出：该结点的右孩子结点

时间复杂度：O(n)

(13)求左兄弟结点

函数定义为Tnode \* LeftSibling(Bitree T,int e)，基本思想是先调用Parent（T，e）找到该结点的双亲结点，然后返回双亲结点的左孩子结点。

输入：要求左兄弟结点的结点

输出：该结点的左兄弟结点

时间复杂度：O(n)

(14)求右兄弟结点

函数定义为Tnode \* RightSibling(Bitree T,int e)，基本思想同求左兄弟结点。

输入：要求右兄弟结点的结点

输出：该结点的右兄弟结点

时间复杂度：O（n）

（15）插入子树

函数定义为status InsertChild(Bitree T,int e,int LR,Bitree c)，基本思想是先调用Value（T，e）找到该结点，然后根据LR的值选择将子树插入到左孩子还是右孩子。

输入：要插入子树的结点，要插入的子树，要插入左子树还是右子树

输入：是否成功插入

时间复杂度：O（n）

（16）删除子树

函数定义为status DeleteChild(Bitree T,int e ,int LR),基本思想是先调用Value（T，e）找到该结点，然后根据LR的值调用DestroyBiTree删除左子树或右子树。

输入：要删除子树的结点，要删除左子树还是右子树

输出：是否删除子树

时间复杂度：O(n)

(17)前序遍历

函数定义为status PreOrderTraverse(Bitree T)，基本思想是前序递归遍历结点，代码如下：

if(T)

{  printf("%d.%d ",T->data.key,T->data.value);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

return OK;

}

return ERROR;

输入：无

输出:前序输出所有结点

时间复杂度：O（n）

（18）中序遍历

函数定义为status InOrderTraverse(Bitree T)，基本思想是中序递归遍历所有结点，代码如下：

if(T)

{ InOrderTraverse(T->lchild);

printf("%d.%d ",T->data.key,T->data.value);

InOrderTraverse(T->rchild);

return OK;

}

return ERROR;

输入：无

输出：中序输出所有结点

时间复杂度：O（n）

（19）后序遍历

函数定义为status PostOrderTraverse(Bitree T)，基本思想是后序递归遍历所有结点，代码如下：

if(T)

{ PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%d.%d ",T->data.key,T->data.value);

return OK;

}

return ERROR;

输入：无

输出：后序遍历所有结点

时间复杂度：O（n）

(20)层序遍历

函数定义为status LevelOrderTraverse(Bitree T)，基本思想是定义一个结构数组按层保存二叉树，然后输出各层结点的值。

输入：无

输出：层序遍历所有结点

时间复杂度：O(n)

（21）文件输入

函数定义为status LoadToFile(Bitree T, char \* filename,FILE \*fp)，基本思想是先存入头结点的值和标记，然后递归存入各结点的值和标记。

输入：要存入文件的名称

输出：是否存入文件

（22）文件输出

函数定义为int LoadFromFile(Bitree & T, char \* filename)，基本思想是，先将文件中结点的标记和值读入线性表L，然后调用函数CreatBiTree(T,L->elem)。

输入：要读取文件的名称

输出：是否读取文件

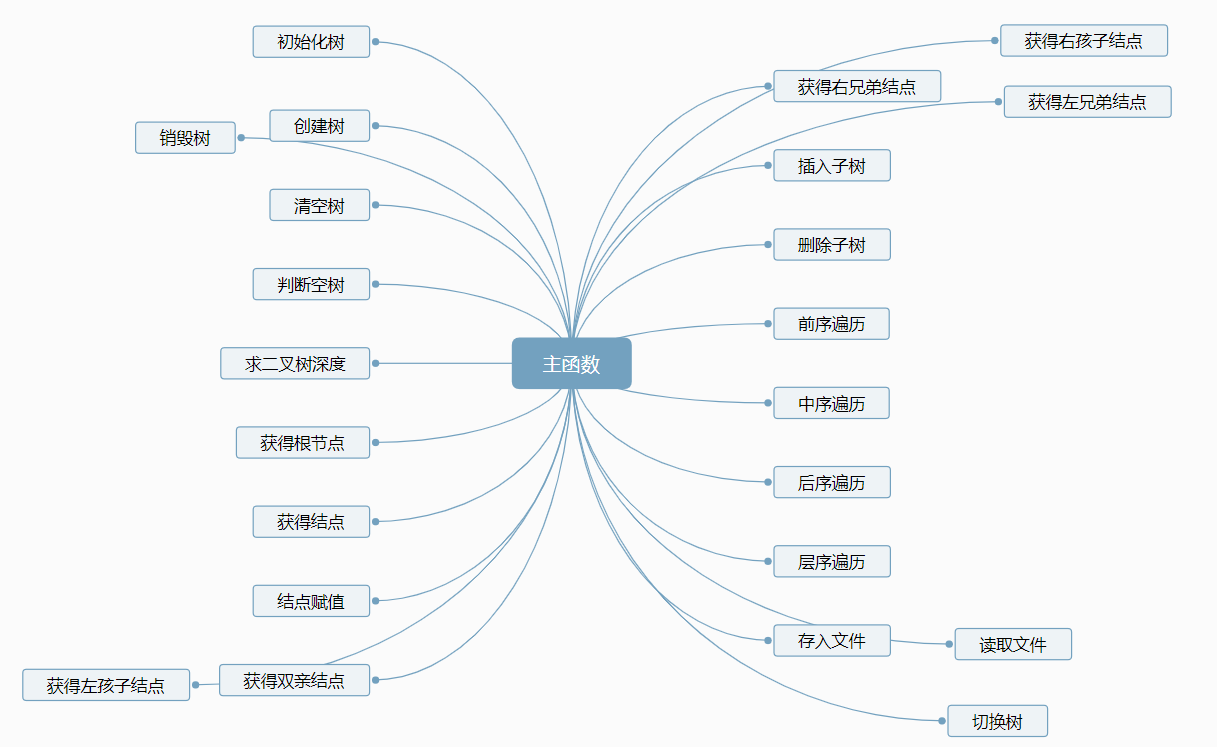
（23）切换表

基本思想是创建了三个线性表L1,L2,L3保存二叉树，然后创建了一个指针L，L初始指向L1,当用户要切换二叉树时，根据用户输入将L指向L1或L2。

### 3.3 系统实现

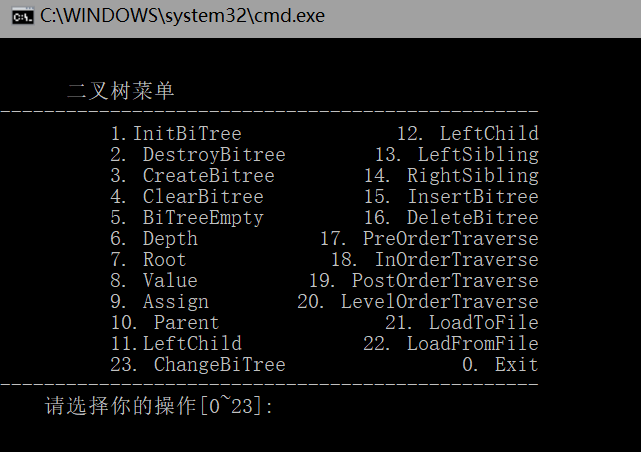
系统设计为：

图3.3.1 系统设计图



系统界面为：

图3.3.2 系统界面图



由于函数过多，不一一进行测试，只进行具有代表性的测试。

1. 初始容错测试

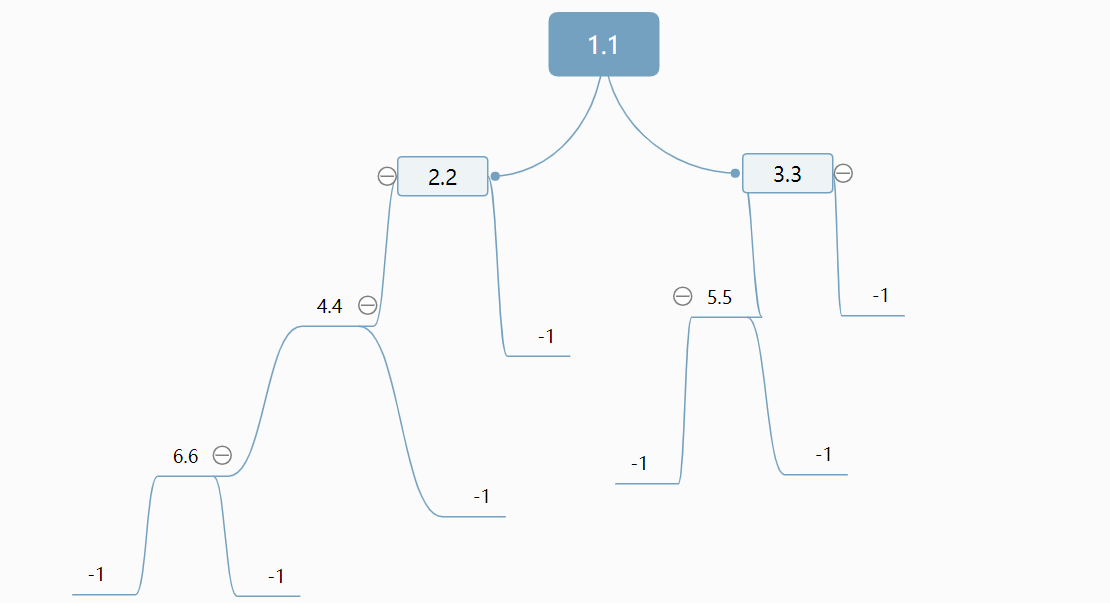
表3.3.1—初始容错性测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 测试结果 |
| 用例1 | 输入2 |  |
| 用例2 | 输入4 |  |
| 用例3 | 输入8 |  |
| 用例4 | 输入12 |  |
| 用例5 | 输入20 |  |
| 用例6 | 输入22 |  |

1. 创建二叉树以及遍历二叉树

所创建的二叉树如图3.3.3所示

图3.3.3 所创建的二叉树



（-1表示该节点不存在）

按照上图创建二叉树，结果如图3.3.4

图3.3.4—成功创建二叉树

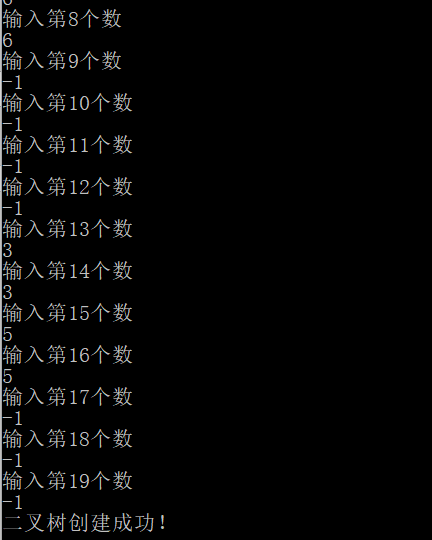


表3.3.2—二叉树的遍历

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 相关遍历 | 理论结果 | 测试结果 |
| 用例1 | 前序遍历 | 输出1、2、4、6、3、5 |  |
| 用例2 | 中序遍历 | 输出6，4，2，1，5，3 |  |
| 用例3 | 后序遍历 | 输出6，4，2，5，3，1 |  |
| 用例4 | 层序遍历 | 输出1，2，3，4，5，6 |  |

三．测试二叉树的查询功能

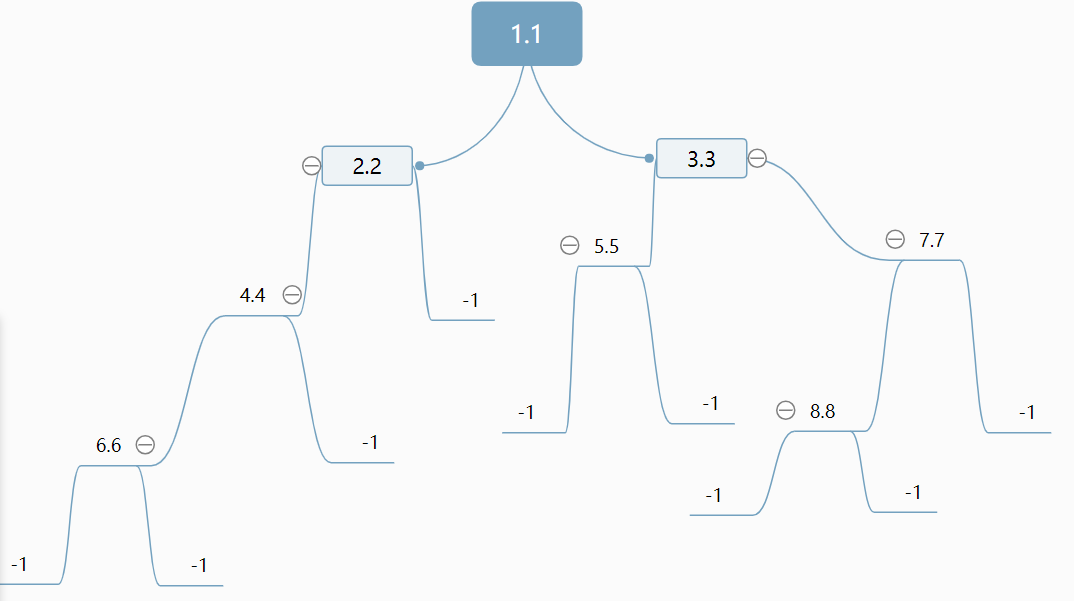
表3.3.3二叉树的查询

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 功能 | 理论结果 | 测试结果 |
| 用例1 | 判断空树 | 非空 |  |
| 用例2 | 求深度 | 4 |  |
| 用例3 | 获取根节点 | 1 |  |
| 用例4 | 求6的双亲结点 | 4 |  |
| 用例5 | 求2的左孩子结点 | 4 |  |
| 用例6 | 求2的右孩子结点 | 无 |  |
| 用例7 | 求3的左兄弟结点 | 2 |  |
| 用例8 | 求2的右兄弟结点 | 3 |  |
| 用例9 | 求1的右孩子结点 | 3 |  |

三.对插入子树和删除子树进行功能测试

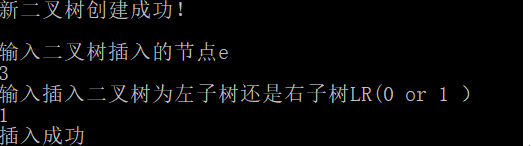
插入子树后的树如图3.3.5所示

图3.3.5—插入子树后的树



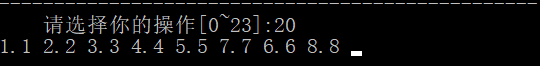
成功插入子树，如图3.3.6所示

图3.3.6—成功插入子树



对插入后的新树进行层序遍历，结果如图3.3.7

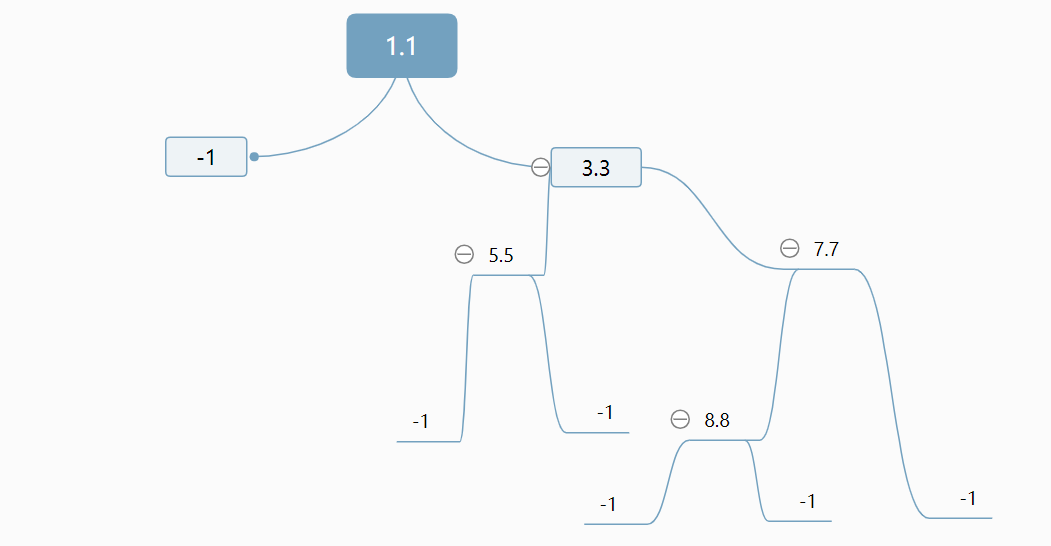
图3.3.7—新树的层序遍历



结果符合理论值，可见，插入是成功的。

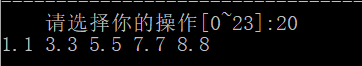
下面进行删除子树的测试，删除后的树如图3.3.8所示

图3.3.8—删除后的树



对删除后的树进行层序遍历，结果如图3.3.9所示

图3.3.9—删除子树后的树的层序遍历

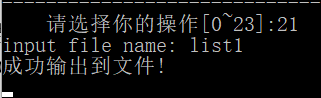


结果符合预期，可见插入是成功的

四. 对切换树和保存树进行测试

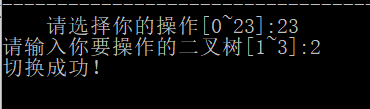
首先将二叉树保存在list1中，保存结果如图3.3.10所示

图3.3.10—保存文件



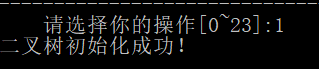
然后切换到表2，结果如图3.3.11所示

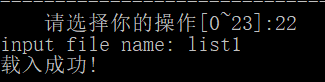
图3.3.11—切换到表2

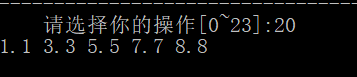


先对表2进行初始化，然后将list1输入到表2中，进行层序遍历，结果如图3.3.12所示

图3.3.12—对表2进行测试



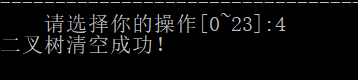


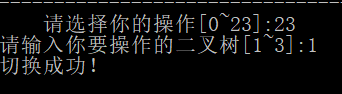


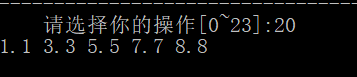
可见，结果符合理论值。

然后将表2清空，切换回表1，进行层序遍历，结果如图3.3.13所示

图3.3.13—测试结果







可见，表1还是原来的表1，测试通过

### 3.4 实验小结

通过这次实验，我对二叉树的各种操作有了一个整体的认识，尤其是对递归这个功能有了更深刻的认识，实验过程中，许多函数都用到了递归这一功能，解决了很多问题，在做二叉树的创建这一函数时，一开始我没有用递归的思想，导致函数特别长，出错的地方很多，改也不好改，当我把递归的思想加进去后，函数变短了，也没有出现问题了，在这个过程中，感受到了递归功能的强大之处，另外这次实验虽然函数重大，但许多函数之间都有相通之处，往往解决了一个就解决了很多个，这次实验让我收获了很多。

# 4基于邻接表的图实现

### 4.1 问题描述

本次实验需要实现图的十三种功能，分别是创建图（按V和VR的定义构造图G）、销毁图（销毁图G）、查找顶点（若u在图G中存在，返回顶点u的位置信息，否则返回其它信息）、获得顶点值（返回v的值）、顶点赋值（对v赋值value）、获得第一邻接点（返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”、获得下一邻接点）、获取下一个邻接点（返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回“空”）、插入顶点（在图G中增加新顶点v）、删除顶点（在图G中删除顶点v和与v相关的弧）、插入弧（在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>）、删除弧（在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>）、深度优先搜索遍历（图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次）、广深度优先搜索遍历（图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次）。另外根据实际需要，我加入了存入文件（将线性表以文件形式保存）、读取文件（将文件中的数据读取到线性表中）、切换表（操作多个线性表）这三个操作使功能更全面。

### 4.2 系统设计

**一．数据结构定义**

typedef struct VertexType{

char key[MAX\_CHAR]; //图顶点信息

int value;//顶点的值

int isInit;//是否为头结点

};

typedef struct ArcNode{

int adjvex;//该弧指向的顶点的位置

struct ArcNode \*nextarc;//指向下一条弧的指针

InfoType \*info;//该弧相关信息的指针

};

typedef struct VNode{

VertexType data;//顶点信息

ArcNode \*firstarc;//指向第一条依附该顶点的弧的指针

}List[MAX\_VERTEX\_NUM];

typedef struct Graph{

List vertices;

int vexnum,arcnum;//图的当前顶点数和弧数

int type;//图的种类标志

};

**有关常量和类型的定义**

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_CHAR 3

#define MAX\_VERTEX\_NUM 20

typedef int status;

typedef int InfoType;//弧相关的信息

typedef enum{DG,DN}GraphKind;//{有向图，有向网}

**二.算法设计**

（1）构造图

函数定义为status CreatGraph(Graph&G)，基本思想是：先让用户输入图的基本信息，包括图的类型（有向图或有向网）、图的顶点数、图的边数。然后输入每个图的信息，在图的结构体中构造顶点的线性表，同时对每个顶点的邻接边指针进行置空，接下来构造表的结点链表，将新输入的结点插入头部。

输入：图的基本信息

输出：是否成功创建图

时间复杂度：O（n+e）

(2)销毁图

函数定义为status DestroyGraph(Graph&G),基本思想是：首先将图的结点数和边数都置0，再对线性表中的每个顶点所连接的邻边进行遍历，并一一释放各条弧的存储空间。如果是网，还需要释放掉权值的存储空间。

输入：无

输出：是否成功销毁图

时间复杂度：O（n+e）

(3)查找顶点

函数定义为status LocateVex(Graph G,VertexType elem)，基本思想是：遍历图结构体存放顶点的线性表，找到顶点的位置，并返回位置序数。

输入：要查的顶点值

输出：所查顶点的位置

时间复杂度：O（n）

(4)获得结点

函数定义为status GetVex(Graph G,VertexType elem,int &value)，基本思想同查找顶点，不同的是返回结点的值。

输入：要查的顶点位置

输出：所查顶点的值

时间复杂度：O(n)

(5)给结点赋值

函数定义为status PutVex(Graph&G,VertexType elem,int value)，基本思想同上，找到结点后，改变结点中data.value的值即可。

输入：要赋值的顶点位置以及要赋的值

输出：是否赋值成功

时间复杂度：O（n）

(6) 获得第一个邻接结点

函数定义为int FirstAdjVex(Graph G,VertexType elem)，基本思想是同上，返回结点中fristarc所指向结点所对应的adjvex值。

输入：要求顶点

输出：要求顶点的第一个邻接结点

时间复杂度：O（n）

(7)获得下一个邻接结点

函数定义为int NextAdjVex(Graph G,VertexType elem,VertexType elem2)，基本思想是遍历顶点所在的邻接链表，找到elem2,返回elem2后继所对应的adjvex值。

输入：要求顶点及其一个邻接结点

输出：要求顶点的下一个邻接结点

时间复杂度：O（n+e）

(8)插入顶点

函数定义为status InsertVex(Graph&G,VertexType elem)，基本思想是：在G所在的结构的vertices数组的末尾插入新的结点，对图结构的vexnum加1。

输入：要插入的顶点值

输出：是否成功插入

时间复杂度：O(1)

(9)删除顶点

函数定义为status DeleteVex(Graph &G,VertexType elem)，基本思想是：先遍历以该顶点为出度的邻边，全部释放存储空间，之后再遍历存储顶点的线性表，找到所有以该顶点为出度的邻边释放空间，同时将该邻边的前驱和后继相连，并对图结构的vexnum减1。

输入：要删除的顶点值

输出：是否成功删除

时间复杂度：O(n+e)

(10)插入邻边

函数定义为status InsertArc(Graph&G,VertexType elem,VertexType elem2)。基本思想是：先获取所输入的两个顶点的位置，并判断两点间是否已经存在邻边。若存在，则返回不可指向。否则，在出度点的fristarc上插入含入读点的相关信息的弧。同时，对图结构的arcnum加1.

输入：要插入邻边的两个点

输出：是否成功插入

（11）删除邻边

函数定义为status DeleteArc(Graph&G,VertexType elem,VertexType elem2)，基本思想是：首先查找邻边是否存在，不存在返回INFEASIBLE。存在则遍历弧的出度顶点所连接的邻边，找到弧后，连接弧的前驱和后继，并释放这条弧的存储空间。同时，对图结构的arcnum减1。

输入：要删除邻边的两个点

输出：是否成功删除

时间复杂度：O(n+e)

（12）深度优先遍历

函数定义为status DFSTraverse(Graph G)，基本思想是：1.访问初始结点v，并对v进行已访问的标记2.查找v的第一个邻接结点w 3.若w存在，则执行下一步，否则结束遍历 4.若w未被访问，则对w进行深度优先遍历 5.查找w的下一个邻接结点，从3开始

输入：无

输出：输出深度遍历的结果

时间复杂度:O(n+e)

(13)广度优先遍历

函数定义为status BFSTraverse(Graph G,LinkQueue Q)，基本思想是：1.访问初始结点v,并对v进行已访问的标记 2.将结点v加入队列 3.若队列已空，则结束遍历，否则继续 4.出队列，取得头结点u 5.查询u的第一个邻接结点w 6.若w不存在，则转到第3步，若存在，则:（1）若w未被访问，则访问w并标记已访问(2)让结点w入队（3）查找结点u在w之后的下一个邻接结点w1，并转到第6步。

输入：无

输出：输出广度遍历的结果

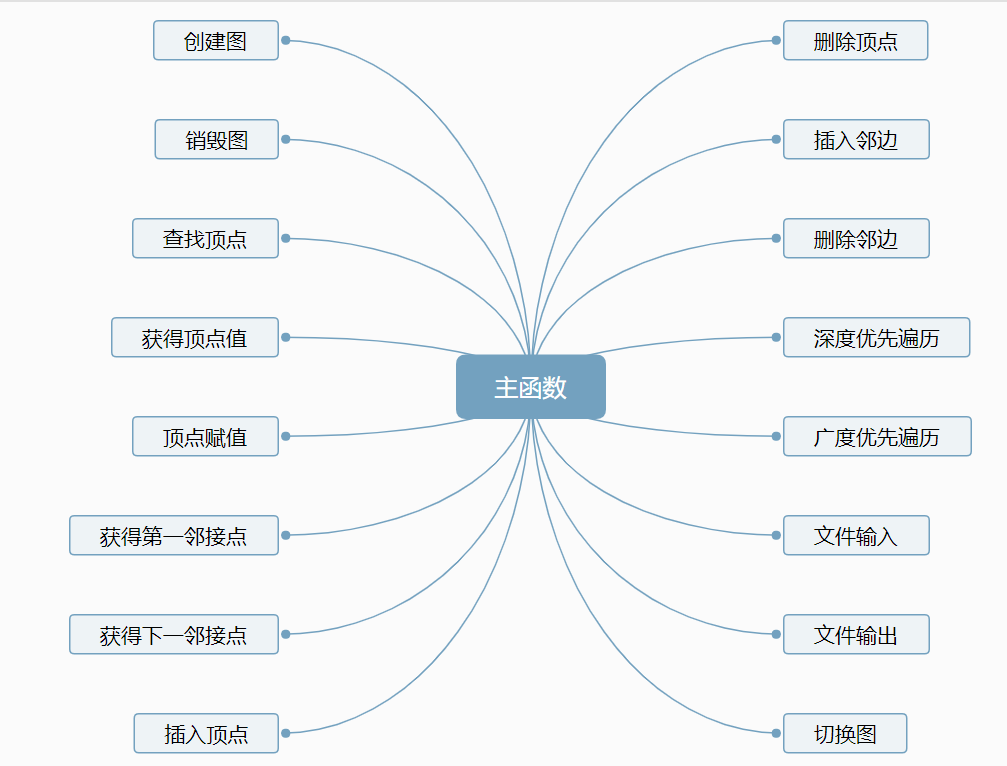
时间复杂度：O(n+e)

文件输入输出以及切换图与前三次实验大致相同，只是根据图结构做了一些小调整，不再介绍。

### 4.3 系统实现

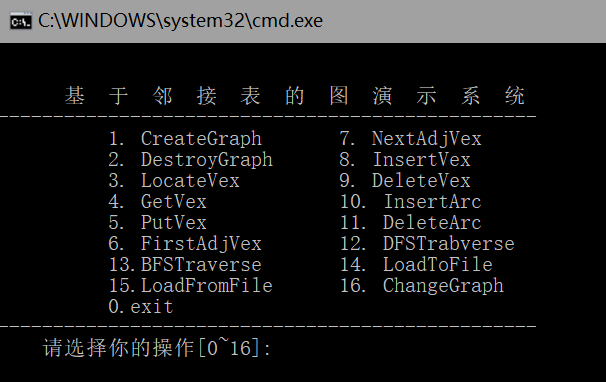
系统设计为：

图4.3.1—系统设计图



系统界面为：

图4.3.2—系统界面图



下面进行初始容错性测试

1. 初始容错测试

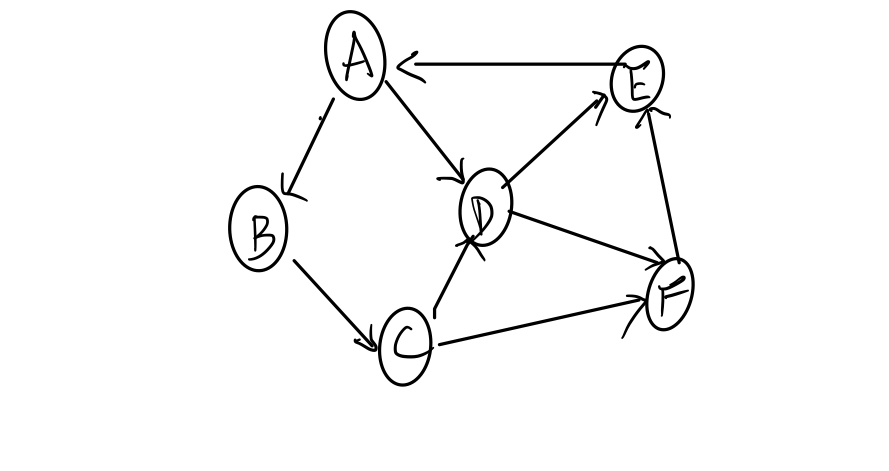
表4.3.1—初始容错性测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 测试结果 |
| 用例1 | 输入2 |  |
| 用例2 | 输入4 |  |
| 用例3 | 输入8 |  |
| 用例4 | 输入12 |  |
| 用例5 | 输入15 |  |

二.创建图以及遍历图

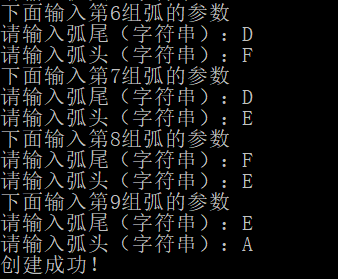
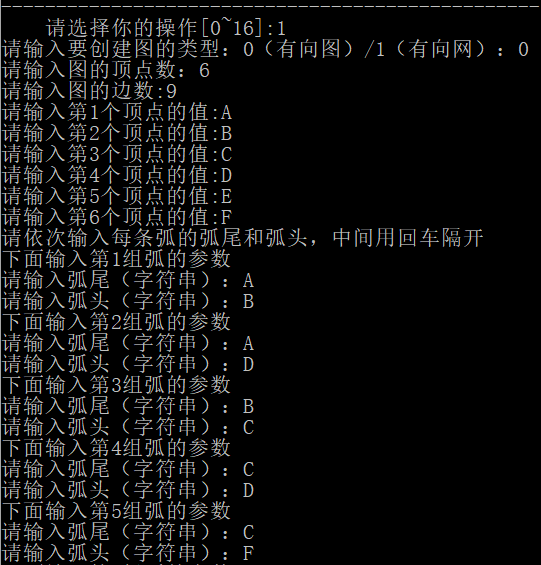
所创建的图如下图所示

图4.3.3--所创建的图



按照上图创建图，创建结果如下图所示

图4.3.4—创建图



下面进行遍历测试

表4.3.2—图的遍历

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 相关遍历 | 理论结果 | 测试结果 |
| 用例1 | 深度优先遍历 | 输出A、D、E、F、B、C |  |
| 用例2 | 广度优先遍历 | 输出A、D、B、E、F、C |  |

遍历准确无误，说明创建成功

三.查询图

下面对图的查询功能进行测试

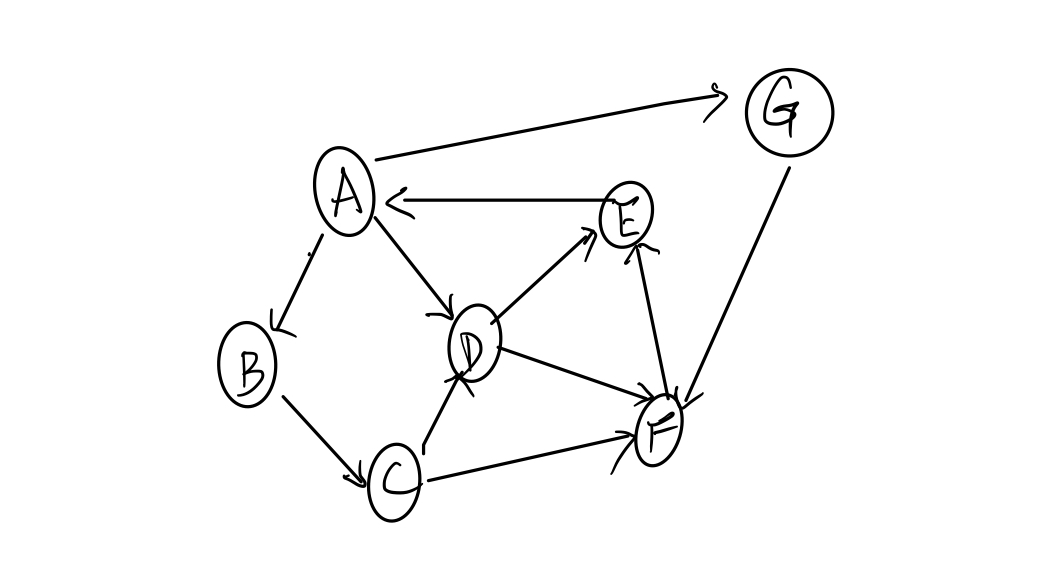
表4.3.3图的查询

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 功能 | 理论结果 | 测试结果 |
| 用例1 | 查找结点C | 标记为2 |  |
| 用例2 | 给结点B赋值100 | 赋值成功 |  |
| 用例3 | 获得结点B | 100 |  |
| 用例4 | 获得A的第一个邻接结点 | D |  |
| 用例5 | 获得A相对于D的下一个邻接结点 | B |  |

四.插入和删除

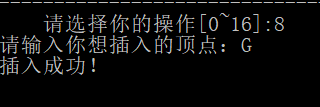
下面对图的插入删除功能进行测试

插入一个结点G，插入后的图如图所示

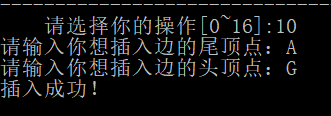
图4.3.5—插入后的图

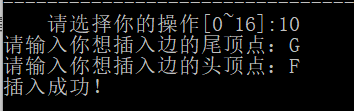
插入步骤如下：

1.插入结点G

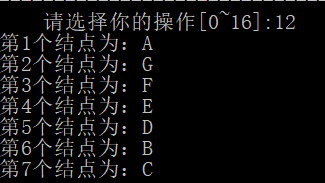


2.插入邻边AG,GF





对插入后的图进行深度优先遍历

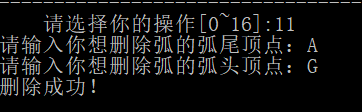


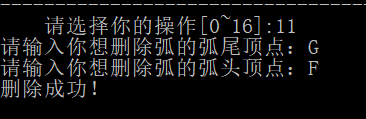
结果符合预期，说明插入成功。

接着，删除G及其邻边，使图恢复之前的状态

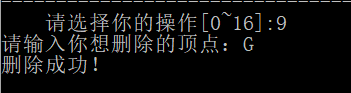
删除步骤如下：

1.删除邻边AG，GF

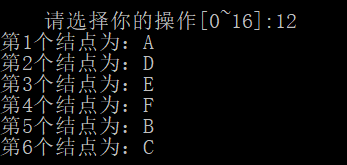




2.删除顶点G



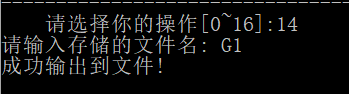
对删除后的图进行深度优先遍历



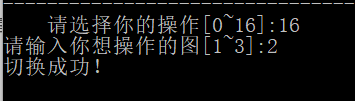
结果符合预期，说明删除成功。

五.销毁图以及文件输入输出及切换图

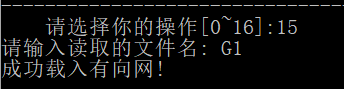
首先将图1存入文件G1



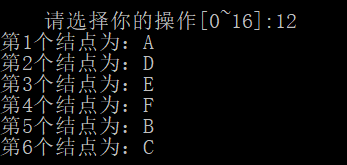
然后切换到G2



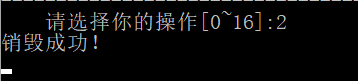
将G1读取到G2

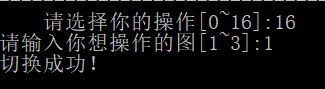


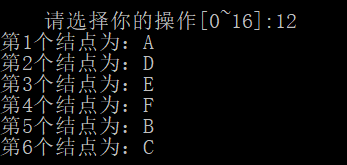
进行深度优先遍历



说明读取成功，接下来将表2销毁，切换回表1进行遍历







至此有向图的测试完毕。

此外这个程序还可以创建有向网，具体是为每条边加了权值，由于没有对应于有向网的测试功能，所以不进行测试

### 4.4 实验小结

这次实验作为数据结构的最后一个实验，圆满完成后也算是给数据结构实验课画上了一个圆满的句号，在实验过程中，我理解了以下问题：在邻接表中，顶点是作为弧的头结点存放在线性表中的，与顶点想连的弧则以链表的形式串在顶点所在的头结点上，利用这种存储形式，很容易求得所要求的各种信息。然后就是递归思想的运用，在深度优先遍历中得到了很好的提现，广度优先遍历借助了队列这一数据结构。先入先出式的遍历，从一点为中心，层层铺设开来，这一点很像树的层序遍历。

附录1 线性表程序实现清单

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*---------page 10 on textbook ---------\*/

#define TRUE 1

#define FALSE -1

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

/\*-----page 19 on textbook ---------\*/

status IntiaList(SqList \*L);

status DestroyList(SqList \* L);

status ClearList(SqList \*L);

status ListEmpty(SqList\* L);

int ListLength(SqList \*L);

status GetElem(SqList \*L,int i,ElemType & e);

status LocateElem(SqList \*L,ElemType e); //简化过

status PriorElem(SqList \*L,ElemType cue,ElemType \*pre);

status NextElem(SqList \*L,ElemType cue,ElemType \* next);

status ListInsert(SqList \* L,int i,ElemType e);

status ListDelete(SqList \* L,int i,ElemType \* e);

status ListTrabverse(SqList \*L); //简化过

int ReadFromFile(SqList \* L,char \* filename);

int StoreInFile(SqList \* L,char \* filename);

/\*--------------------------------------------\*/

int main(int argc, char \*argv[]){

SqList L1,L2,L3,\*L=&L1; int op=1;

L1.elem=NULL;

L2.elem=NULL;

L3.elem=NULL;

ElemType e;

ElemType temp,\*pre=&temp;

ElemType temp1,\*next=&temp1;

ElemType cue;

char filename[30];

int i;

while(op){

system("cls"); printf("\n\n");

printf(" Menu for Linear Table On Sequence Structure \n");

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

printf(" 1. IntiaList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTrabverse\n");

printf(" 13.StoreInFile 14.ReadFromFile\n");

printf(" 15.ChangeList 0. Exit\n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

//printf("\n----IntiaList功能待实现！\n");

if(IntiaList(L) == OK) {

printf("线性表创建成功！\n");

getchar();getchar();

}

else{

printf("线性表创建失败！\n");

getchar();getchar();

}

break;

case 2:

// printf("\n----DestroyList功能待实现！\n");

if(DestroyList(L) == OK){

printf("线性表销毁成功！\n");

getchar();getchar();

}

else {

printf("线性表销毁失败！\n");

getchar();getchar();

}

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if(ClearList(L) == OK){

printf("线性表清空成功!\n");

getchar();getchar();

}

else {

printf("线性表清空失败！\n");

getchar();getchar();

}

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if(ListEmpty(L) == OK){

printf("线性表已清空！\n");

getchar();getchar();

}

else if(ListEmpty(L) ==FALSE){

printf("线性表未清空！\n");

getchar();getchar();

}

else if(ListEmpty(L)==ERROR){

printf("线性表未初始化！\n");

getchar();getchar();

}

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

if( ListLength(L)!=ERROR){

printf("线性表长度为%d\n",ListLength(L));

getchar();getchar();

}

else{

printf("获取长度失败！\n");

getchar();getchar();

}

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

printf("请输入要查询的序数：");

scanf("%d",&i);

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)&&i<=L->length){

GetElem(L,i,e) ;

printf("表中的第%d个数据为%d\n",i,e);

getchar();getchar();

}

else{

printf("查询失败！\n");

getchar();getchar();

}

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

printf("请输入要查询的数据：");

scanf("%d",&e);

i=LocateElem(L,e);

if(i==FALSE) {

printf("这样的数据元素不存在！\n");

getchar();getchar();

}

else if(i==ERROR){

printf("查询失败！\n");

getchar();getchar();

}

else {

printf("L中第一个与查询数据相等的数据的位序是%d\n",i);

getchar();getchar();

}

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

printf("请输入要查询的元素：");

scanf("%d",&cue);

if(PriorElem(L,cue,pre) == OK){

printf("前驱为%d\n",\*pre);

getchar();getchar();

}

else {

printf("无此前驱\n");

getchar();getchar();

}

}

else{

printf("查询失败！\n");

getchar();getchar();

}

break;

case 9:

// printf("\n----NextElem功能待实现！\n");

printf("请输入要查询的元素：");

scanf("%d",&cue);

if(NextElem(L,cue,next) == OK){

printf("后继为%d\n",\*next);

getchar();getchar();

}

else if(NextElem(L,cue,next) == FALSE){

printf("无此后继\n");

getchar();getchar();

}

else{

printf("查找失败！\n");

getchar();getchar();

}

break;

case 10:

// printf("\n----ListInsert功能待实现！\n");

printf("请输入要插入的位置：");

scanf("%d",&i);

printf("请输入要插入的元素：");

scanf("%d",&e);

if(ListInsert(L,i,e) == OK){

printf("线性表插入成功\n");

getchar();getchar();

}

else{

printf("线性表插入失败\n");

getchar();getchar();

}

break;

case 11:

// printf("\n----ListDelete功能待实现！\n");

printf("请输入要删除的元素的序列：");

scanf("%d",&i);

if(ListDelete(L,i,&e) == OK){

printf("元素删除成功\n");

getchar();getchar();

}

else{

printf("元素删除失败\n");

getchar();getchar();

}

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if(!ListTrabverse(L))

printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

printf("input file name: ");

scanf("%s",filename);//写文件的方法

if(StoreInFile(L,filename)==OK){

printf("写入成功！\n");

getchar();getchar();

}

else{

printf("写入失败！\n");

getchar();getchar();

}

}

else{

printf("未初始化！\n");

getchar();getchar();

}

break;

case 14:

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

printf("input file name: ");

scanf("%s",filename);//写文件的方法

if(ReadFromFile(L,filename)==OK){

printf("读入成功！\n");

getchar();getchar();

}

else{

printf("读入失败！\n");

getchar();getchar();

}

}

else{

printf("未初始化！\n");

getchar();getchar();

}

break;

case 15:

int m;

printf("请输入你要操作的表[1-3]:");

scanf("%d",&m);

if(m==1){

L=&L1;

printf("切换成功\n");

getchar();getchar();

}

else if(m==2){

L=&L2;

printf("切换成功\n");

getchar();getchar();

}

else if(m==3){

L=&L3;

printf("切换成功\n");

getchar();getchar();

}

else {

printf("切换失败！\n");

getchar();getchar();

}

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

/\*--------page 23 on textbook --------------------\*/

status IntiaList(SqList \* L){

L->elem = (ElemType \*)malloc( LIST\_INIT\_SIZE \* sizeof (ElemType));

if(!L->elem)

exit(OVERFLOW);

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

}//初始化表

status DestroyList(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

free(L->elem);

L->elem = NULL;

return OK;

}

else

return ERROR;

}//销毁表

status ClearList(SqList\* L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

L->length = 0;

return OK;

}

else

return ERROR;

}//清空表

status ListEmpty(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if(L->length == 0)

return TRUE;

else

return FALSE;

}

else

return ERROR;

}//判定空表

int ListLength(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE))

return L->length;

else

return ERROR;

}//求表长

status GetElem(SqList\* L,int i,ElemType &e){

e=\*(L->elem+i-1);

return e;

}//获得元素

int LocateElem(SqList \*L,ElemType e){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

int k=1;

while(\*L->elem != e){

L->elem++;

k++;

if(k >L->length){

return FALSE;

}

}

return k;

}

else

return ERROR;

}//查找元素

status PriorElem(SqList \*L,ElemType cue,ElemType \*pre){

int i;

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

if(L->elem[i] == cue){

if(i==0)

return FALSE;

\*pre=L->elem[i-1];

return OK;

}

}

return FALSE;

}//获得前驱

status NextElem(SqList\* L,ElemType cue,ElemType \*next){

int i;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

for(i=0;i<L->length-1;i++){

if(L->elem[i] == cue){

if(i==L->length-1)

return FALSE;

\*next = (int)L->elem[i+1];

return OK;

}

}

return FALSE;

}

else

return ERROR;

}//获得后继

status ListInsert(SqList \*L,int i,ElemType e){

ElemType \*f,\*t,\*p;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if(!L->elem)

return ERROR;

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

return ERROR;

if(L->length >= L->length+1)

return ERROR;

if(L->length >= L->listsize){

f = (ElemType \*)realloc(L->elem,(L->listsize +LISTINCREMENT)\*sizeof(ElemType));

if(!f)

return ERROR;

L->elem = f;

L->listsize += LISTINCREMENT;

}

t =&(L->elem[i-1]);

for(p=&(L->elem[L->length-1]);p>=t;p--){

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

}

\*t=e;

++L->length;

return OK;

}

else

return ERROR;

}//插入元素

status ListDelete(SqList \*L,int i,ElemType\*e){

ElemType \*t,\*p;

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

return ERROR;

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

e=p;

t=&(L->elem[L->length-1]);

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

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

--L->length;

return OK;

}//删除元素

status ListTrabverse(SqList \*L){

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

int i;

printf("\n-----------all elements -----------------------\n");

for(i=0;i<L->length;i++) printf("%d ",L->elem[i]);

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

return L->length;

}

else

return ERROR;

}//遍历表

int StoreInFile(SqList \*L,char \* filename){

FILE \*fp;

if (( L->elem!=NULL)&&( L->listsize>=LIST\_INIT\_SIZE)){

if ((fp=fopen(filename,"w"))==NULL){

printf("File open error\n ");

}

fwrite(L->elem,sizeof(ElemType),L->length,fp);//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素并写入到文件中

fclose(fp);

return OK;

}

else

return ERROR;

}

int ReadFromFile(SqList \*L,char \* filename){

L->length=0;

FILE \*fp;

if (L->length<=LIST\_INIT\_SIZE){

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

printf("File open error\n ");

return 1;

}

while(fread(&(L->elem[L->length]),sizeof(ElemType),1,fp)){

L->length++;//这里从文件中逐个读取数据元素恢复顺序表，对于不同的物理结构，可通过读取的数据元素恢复内存中的物理结构。

}

fclose(fp);

return OK;

}

}

附录2 链表程序实现清单

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*---------page 10 on textbook ---------\*/

#define TRUE 1

#define FALSE -1

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct Node{ //单链表的定义

ElemType elem;

struct Node \*next;

}Node,\*pNode;

/\*-----page 19 on textbook ---------\*/

status IntiaList(pNode &L);

status DestroyList(pNode & L);

status ClearList(pNode &L);

status ListEmpty(pNode L);

int ListLength(pNode L);

status GetElem(pNode L,int seq,ElemType & e);

status LocateElem(pNode L,ElemType e); //简化过

status PriorElem(pNode L,ElemType cue,ElemType &pre);

status NextElem(pNode L,ElemType cue,ElemType & next);

status ListInsert(pNode & L,int seq,ElemType e);

status ListDelete(pNode & L,int seq,ElemType &e);

status ListTrabverse(pNode L); //简化过

status StoreInFile(pNode L,char \* filename);

status ReadFromFile(pNode & L,char \* filename);

/\*--------------------------------------------\*/

int main(void){

int op=1;

pNode\* L;

pNode L1,L2,L3;

L1=NULL;L2=NULL;L3=NULL;

L=&L1;

int seq,result;//seq表示序号数,result返回结果；e用来代入函数获取结果

ElemType e,cue,pre,next;

int Initial=0;//判断链表是否存在

while(op){

system("cls"); printf("\n\n");

printf(" Menu for Linear Table On Linked Structure \n");

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

printf(" 1. IntiaList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTrabverse\n");

printf(" 13.StoreInFile 14.ReadFromFile\n");

printf(" 15.ChangeList 0. Exit\n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

//printf("\n----IntiaList功能待实现！\n");

if(IntiaList(\*L) == OK) {

printf("单链表创建成功！\n");

Initial=1;

getchar();getchar();

}

else{

printf("单链表创建失败！\n");

getchar();getchar();

}

break;

case 2:

// printf("\n----DestroyList功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

if(DestroyList(\*L) == OK){

printf("单链表销毁成功！\n");

Initial=0;

getchar();getchar();

}

else {

printf("单链表销毁失败！\n");

getchar();getchar();

}

}

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

if(ClearList(\*L) == OK){

printf("单链表清空成功!\n");

getchar();getchar();

}

else {

printf("单链表清空失败！\n");

getchar();getchar();

}

}

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

if(ListEmpty(\*L) == TRUE){

printf("单链表已清空！\n");

getchar();getchar();

}

else {

printf("单链表未清空！\n");

getchar();getchar();

}

}

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("单链表长度为%d\n",ListLength(\*L));

getchar();getchar();

}

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要查询的序数：");

scanf("%d",&seq);

if( GetElem(\*L,seq,e)==OK) {

printf("表中的第%d个数据为%d\n",seq,e);

getchar();getchar();}

else{

printf("查询失败！\n");

getchar();getchar();

}

}

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要查询的数据：");

scanf("%d",&e);

result=LocateElem(\*L,e);

if(result!=ERROR){

printf("第一个与%d相等的元素序号是%d\n",e,result);

getchar();getchar();}

else {

printf("这样的数据元素不存在！\n");

getchar();getchar();

}

}

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要查询的元素：");

scanf("%d",&cue);

if(PriorElem(\*L,cue,pre) == OK){

printf("前驱为%d\n",pre);

getchar();getchar();

}

else {

printf("无此前驱\n");

getchar();getchar();

}

}

break;

case 9:

// printf("\n----NextElem功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要查询的元素：");

scanf("%d",&cue);

if(NextElem(\*L,cue,next) == OK){

printf("后继为%d\n",next);

getchar();getchar();

}

else if(NextElem(\*L,cue,next) == FALSE){

printf("无此后继\n");

getchar();getchar();

}

else{

printf("查找失败！\n");

getchar();getchar();

}

}

break;

case 10:

// printf("\n----ListInsert功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要插入的位置：");

scanf("%d",&seq);

printf("请输入要插入的元素：");

scanf("%d",&e);

if(ListInsert(\*L,seq,e) == OK){

printf("单链表插入成功\n");

getchar();getchar();

}

else{

printf("单链表插入失败\n");

getchar();getchar();

}

}

break;

case 11:

// printf("\n----ListDelete功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("请输入要删除的元素的序列：");

scanf("%d",&seq);

if(ListDelete(\*L,seq,e) == OK){

printf("元素删除成功\n");

getchar();getchar();

}

else{

printf("元素删除失败\n");

getchar();getchar();

}

}

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

if(ListTrabverse(\*L)==ERROR){

printf("链表为空！\n");

}

getchar();getchar();

}

break;

case 13:

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("input file name: ");

char filename[30];

scanf("%s",filename);//写文件的方法

if(StoreInFile(\*L,filename)==ERROR){

printf("写入出错！\n");

getchar();getchar();

}

else{

printf("写入成功！\n");

getchar();getchar();

}

}

break;

case 14:

if(!Initial){

printf("请先初始化！\n");

getchar();getchar();

}

else{

printf("input file name: ");

char filename[30];

scanf("%s",filename);//写文件的方法

if(ReadFromFile(\*L,filename)==ERROR){

printf("读入失败！\n");

getchar();getchar();

}

else{

printf("读入成功！\n");

getchar();getchar();

}

}

break;

case 15:

int m;

printf("请输入要操作的链表[1~3]：");

scanf("%d",&m);

if(m==1){

L=&L1;

printf("切换成功！\n");

if(!L1)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m==2){

L=&L2;

printf("切换成功！\n");

if(!L2)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m==3){

L=&L3;

printf("切换成功！\n");

if(!L3)

Initial=0;

else

Initial=1;

getchar();getchar();

}

if(m<1||m>3){

printf("切换失败！\n");

getchar();getchar();

}

break;

case 0:

break;

}//end of switch

} //end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

/\*--------page 23 on textbook --------------------\*/

status IntiaList(pNode& L){

L=(pNode)malloc(2\*sizeof(pNode));

L->elem=0;

L->next=NULL;

return OK;

}//初始化表

status DestroyList(pNode &L){

pNode Lp;

if(L->elem==0){

free(L);

return OK;

}

while(L->next!=NULL){

Lp=L;

L=L->next;

free(Lp);

}

free(L);

return OK;

}//销毁表

status ClearList(pNode & L){

pNode Lp,Lq;

if(L->elem==0)

return ERROR;

Lp=L->next;

while(Lp->next!=NULL){

Lq=Lp;

Lp=Lp->next;

free(Lq);

}

free(Lp);

L->elem=0;

return OK;

}//清空表

status ListEmpty(pNode L){

if(L->elem == 0)

return TRUE;

else

return FALSE;

}//判定空表

int ListLength(pNode L){

return L->elem;

} //求表长

status GetElem(pNode L,int i,ElemType &e){

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

return ERROR;

pNode Lp=L;

int k;

for(k=0;k<i;k++){

Lp=Lp->next;

}

e=Lp->elem;

return OK;

}//获得元素

int LocateElem(pNode L,ElemType e){

int k=0;

pNode Lp=L->next;

while(Lp!=NULL){

k++;

if(Lp->elem==e){

return k;

}

Lp=Lp->next;

}

return ERROR;

}//查找元素

status PriorElem(pNode L,ElemType cue,ElemType &pre){

if(!L->elem){

return FALSE;

}

int k=0;

pNode Lp,Lq;

Lp=L;

while(Lp!=NULL){

k++;

Lq=Lp;

Lp=Lp->next;

if(Lp==NULL)

return FALSE;

if(cue==Lp->elem&&k!=1){

pre=Lq->elem;

return OK;

}

}

return FALSE;

}//获得前驱

status NextElem(pNode L,ElemType cue,ElemType &next){

if(!L->elem)

return FALSE;

pNode Lp=L;

while(Lp->next!=NULL){

Lp=Lp->next;

if(cue==Lp->elem){

if(Lp->next==NULL)

return ERROR;

next=Lp->next->elem;

return OK;}

}

return FALSE;

}//获得后继

status ListInsert(pNode &L,int i,ElemType e){

if(i<1||i>L->elem+1)

return ERROR;

pNode p=L,q,newNode;

newNode=(pNode)malloc(2\*sizeof(pNode));

newNode->elem=e;

int k=0;

while(p!=NULL&&k<i){

q=p;

p=p->next;

k++;}

q->next=newNode;

newNode->next=p;

L->elem=L->elem+1;

return OK;

}//插入元素

status ListDelete(pNode & L,int i,ElemType &e){

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

return ERROR;

pNode p=L,q;

int k=0;

while(p!=NULL&&k<i-1){

p=p->next;

k++;

}

q=p->next;

p->next=q->next;

e=q->elem;

free(q);

L->elem=L->elem-1;

return OK;

}//删除元素

status ListTrabverse(pNode L){

if(L->elem==0)

return ERROR;

pNode p=L->next;

printf("\n-----------all elements -----------------------\n");

while(p!=NULL){

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

p=p->next;

}

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

return OK;

}//遍历表

status StoreInFile(pNode L,char \* filename){

if(L->elem==0)

return ERROR;

pNode p=L->next;

FILE \*fp;

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return ERROR;

}

while(p!=NULL){

fwrite(p,2\*sizeof(pNode),1,fp);//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素并写入到文件中

p=p->next;

}

fclose(fp);

return OK;

}

status ReadFromFile(pNode &L,char \* filename){

ClearList(L);

L->elem=0;

pNode p=L;

FILE \*fp;

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return ERROR;

}

Node temp;

while(fread(&temp,2\*sizeof(pNode),1,fp)){

p->next=(pNode)malloc(2\*sizeof(pNode));

p=p->next;

\*p=temp;

L->elem++;

}

p->next=NULL;

fclose(fp);

return OK;

}

附录3 二叉树程序实现清单

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define OK 1

#define ERROR 0

//数据元素类型定义

typedef int status;

typedef struct {

int key ;

int value ;

} Elemtype;

typedef struct node{

Elemtype data;

struct node \*lchild ,\* rchild;

}Tnode,\*Bitree;

typedef struct {

int \* elem;

int length;;

}SqList;

Tnode \*\*T;

Tnode \*q,\*c,\*L1,\*L2,\*L3;

int j=0,i=0,x=0,xx=0,res=0;

int a[6];

//case 1 初始化二叉树

status InitBiTree(Bitree &T)

{

T=(Bitree)malloc(sizeof(Tnode)) ;

T->lchild=T->rchild=NULL;

return OK;

}

//case2 销毁二叉树

status DestroyBiTree(Bitree &T)

{

if(T)

{

if((T)->lchild)

DestroyBiTree(T->lchild);

if((T)->rchild)

DestroyBiTree(T->rchild);

free(T);

T=NULL;

}

return OK;

}

//case3 创建二叉树

status CreateBiTree(Bitree & T,int \* p){

if(\*(p+xx)==-1)

{xx++;

return OK;

}

T=(Bitree)malloc(sizeof(Tnode)) ;

T->lchild=T->rchild=NULL;

T->data.key=\*(p+xx);

xx++;

T->data.value=\*(p+xx);

xx++;

CreateBiTree(T->lchild,(p));

CreateBiTree(T->rchild,(p));

return OK;

}

//case4 清空二叉树

status ClearBiTree(Bitree T)

{

if(T)

{

if((T)->lchild)

DestroyBiTree(T->lchild);

if((T)->rchild)

DestroyBiTree(T->rchild);

free(T);

T=NULL;

}

T=(Bitree)malloc(sizeof(Tnode)) ;

T->lchild=T->rchild=NULL;

return OK;

}

//case 5 判断二叉树是否为空

status BiTreeEmpty(Bitree T)

{

if((T)->lchild||T->rchild)

return ERROR;

else return OK;

}

//case 6 求二叉树深度

status BiTreeDepth(Bitree T,int j,int &x){

if(T)

{ j=j+1;

if(j>x)x=j;

BiTreeDepth(T->lchild,j,x);

BiTreeDepth(T->rchild,j,x);

}

return x;

}

//case7 求根节点

status Root(Bitree T)

{ printf("该二叉树的根结点值为:%d\n\n",T->data.value);

return OK;

}

//case8 求节点

Tnode \* Value(Bitree T,int e)

{ Tnode \* t1;

if(T==NULL)

return NULL;

if(T->data.key==e)

return T;

t1=Value(T->lchild,e);

if(t1!=NULL)

return t1;

return Value (T->rchild,e);

}

//case9 改变节点的值

status Assign(Bitree T,int e,int d)

{ if( Value(T,e))

Value(T,e)->data.value=d; return OK;

return ERROR;

}

//case10 求双亲节点

Tnode \* Parent(Bitree T,int e)

{ Tnode \* t1;

if(T)

{

if(T->data.key==e)return NULL;

else

{

if(T->lchild!=NULL&&T->lchild->data.key==e||T->rchild!=NULL&&T->rchild->data.key==e)

return T;

else if (t1=Parent( T->lchild, e))return t1;

else if (t1=Parent( T->rchild, e))return t1;

}

}

return NULL;

}

//case11 求左孩子节点

Tnode \* LeftChild(Bitree T,int e)

{

if(Value(T,e)==NULL) return NULL;

else return Value(T,e)->lchild;

}

//case12 求右孩子节点

Tnode \* RightChild(Bitree T,int e)

{

if(Value(T,e)==NULL) return NULL;

else return Value(T,e)->rchild;

}

//case13 求左兄弟

Tnode \* LeftSibling(Bitree T,int e)

{

if(Parent( T, e)&&Parent( T, e)->lchild)

{

if(Parent(T, e)->lchild->data.key == e)

return NULL;

return Parent( T, e)->lchild;

}

else return NULL;

}

//case14 求右兄弟

Tnode \* RightSibling(Bitree T,int e)

{ if(Parent( T, e)&&Parent( T, e)->rchild)

{

if(Parent(T, e)->rchild->data.key == e)

return NULL;

return Parent( T, e)->rchild;

}

else return NULL;

}

//case15 插入子树

status InsertChild(Bitree T,int e,int LR,Bitree c)

{ if(Value(T,e))

{

if(LR==0)

{

c->rchild=Value(T,e)->lchild;

Value(T,e)->lchild=c;

}

if(LR==1)

{

c->rchild=Value(T,e)->rchild;

Value(T,e)->rchild=c;

}

return OK;

}

else return ERROR;

}

//case16 删除子树

status DeleteChild(Bitree T,int e ,int LR)

{ if(Value(T,e))

{

if(LR==0&&Value(T,e)->lchild)DestroyBiTree(T->lchild);

if(LR==1&&Value(T,e)->rchild)DestroyBiTree(T->rchild);

return OK;

}

else return ERROR;

}

//case17 前序遍历

status PreOrderTraverse(Bitree T)

{ if(T)

{ printf("%d.%d ",T->data.key,T->data.value);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

return OK;

}

return ERROR;

}

//case18 中序遍历

status InOrderTraverse(Bitree T)

{ if(T)

{ InOrderTraverse(T->lchild);

printf("%d.%d ",T->data.key,T->data.value);

InOrderTraverse(T->rchild);

return OK;

}

return ERROR;

}

//case19 后序遍历

status PostOrderTraverse(Bitree T)

{ if(T)

{ PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%d.%d ",T->data.key,T->data.value);

return OK;

}

return ERROR;

}

//case20 层序遍历

status LevelOrderTraverse(Bitree T)

{ int i=0,n=0;

Bitree temp[100],p;

temp[n]=T;

while(i<=n)

{p=temp[i];

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

if(p->lchild)temp[++n]=p->lchild;

if(p->rchild)temp[++n]=p->rchild;

i++;

}

return OK;

}

//case 21 文件输入

status LoadToFile(Bitree T, char \* filename,FILE \*fp)

{

int a=-1;

if(!T)

{

fwrite(&a, sizeof(int),1, fp);

return 0;

}

fwrite(&(T->data.key), sizeof(int),1, fp);

fwrite(&(T->data.value), sizeof(int), 1, fp);

LoadToFile( T->lchild, filename,fp);

LoadToFile( T->rchild, filename,fp);

return 0;

}

//case 22 文件输出

int LoadFromFile(Bitree & T, char \* filename)

{

FILE \* fp;

SqList \*L;

L=NULL;

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

L->elem = (int \*)malloc( 100 \* sizeof (int));

L->length=0;

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

{

printf("File open error!\n");

return 1;

}

if(T->lchild||T->rchild)

return 1;

while (fread(&L->elem[L->length], sizeof(int), 1, fp))

{

L->length++;

}

fclose(fp);

CreateBiTree(T,L->elem);

return 0;

}

void main(void){

int op=1,d=0;

char filename[30];

SqList \*L;

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

L->elem = (int \*)malloc( 100 \* sizeof (int));

L->length=0;

while(op){

system("cls"); printf("\n\n");

printf(" 二叉树菜单 \n");

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

printf(" 1.InitBiTree 12. LeftChild\n");

printf(" 2. DestroyBitree 13. LeftSibling \n");

printf(" 3. CreateBitree 14. RightSibling\n");

printf(" 4. ClearBitree 15. InsertBitree\n");

printf(" 5. BiTreeEmpty 16. DeleteBitree\n");

printf(" 6. Depth 17. PreOrderTraverse\n");

printf(" 7. Root 18. InOrderTraverse\n");

printf(" 8. Value 19. PostOrderTraverse\n");

printf(" 9. Assign 20. LevelOrderTraverse\n");

printf(" 10. Parent 21. LoadToFile\n");

printf(" 11.LeftChild 22. LoadFromFile\n");

printf(" 23. ChangeBiTree 0. Exit\n");

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

T=&L1;

printf(" 请选择你的操作[0~23]:");

while (scanf("%d", &op) != 1 || (op < 0 || op > 23))

{

printf("输入错误，请重新选择你的操作[0~23]: ");

while (getchar() != '\n');

}

switch(op){

//初始化二叉树

case 1:

if(InitBiTree(\*T)==OK)

{

printf("二叉树初始化成功！\n"); a[d]=1;

}

else printf("二叉树初始化失败！\n");

getchar();getchar();

break;

//二叉树销毁

case 2:

if(a[d]==0)

printf("二叉树不存在！\n");

else {

if(DestroyBiTree(\*T)==OK)

{

printf("二叉树销毁成功！\n"); a[d]=0;

}

else printf("二叉树销毁失败！\n");

}

getchar();getchar();

break;

//创建二叉树

case 3:

if(a[d]==0)

printf("二叉树不存在！\n");

else {xx=0;

int \*p,fl=0,n=0;

p=L->elem+1;

printf("输入要创建的二叉树总标记数\n");

scanf("%d",&j);

printf("输入要创建的二叉树\n");

while (n<j )

{ printf("输入第%d个数\n",n+1);

scanf("%d",&fl);

\*p=fl;

p++;

n++;

}

if (CreateBiTree(\*T,(L->elem+1))==OK)

{

printf("二叉树创建成功！\n");

a[d]=1;

}

else printf("二叉树创建失败！\n"); }

getchar();getchar();

break;

//二叉树清空

case 4:

if(a[d]==0)

printf("二叉树不存在！\n");

else

{

if(ClearBiTree(\*T)==OK)

{

printf("二叉树清空成功！\n"); a[d]=2;

}

else printf("二叉树清空失败！\n");

}

getchar();getchar();

break;

//判定二叉树是否为空

case 5:

if(a[d]==0)

printf("二叉树不存在！\n");

else

{

if( BiTreeEmpty(\*T)==OK) printf("二叉树为空！\n");

else printf("二叉树不为空！\n");

}

getchar();getchar();

break;

//求二叉树的深度

case 6:

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

printf("%d\n",BiTreeDepth(\*T,0,x));

getchar();getchar();

}

break;

//求二叉树的根节点

case 7:

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else {

Root( \*T) ;

getchar();getchar();

}

break;

//获得节点

case 8:

int e;

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else {

printf("输入一个值\n");

scanf("%d",&e);

if(Value(\*T,e)) {

printf("该节点的值为%d ",Value(\*T,e)->data.value);

getchar();getchar();

}

else{

printf("无该节点");

getchar();getchar();

}

}

break;

//节点赋值

case 9:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else {

int e,d;

printf("要改变值的节点\n");

scanf("%d",&e);

printf("要改变的值\n");

scanf("%d",&d);

if (Assign(\*T,e,d)==OK) {

printf("赋值完成");

getchar();getchar();

}

else{

printf("没有对应的节点！\n");

getchar();getchar();

}

}

break;

//求双亲节点

case 10:

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else

{

printf("要查询的值\n");

scanf("%d",&e);

if(e==1||Parent(\*T,e)==NULL){

printf ("该值没有双亲节点");

getchar();getchar();

}

else {

printf("要求节点的值为:%d",Parent(\*T,e)->data.value);

getchar();getchar();

}

}

break;

//求左孩子节点

case 11:

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

printf("要查询的值\n");

scanf("%d",&e);

if(LeftChild(\*T,e)==NULL){

printf ("该值没有左孩子节点");

getchar();getchar();

}

else {

printf("要求节点的值为:%d",LeftChild(\*T,e)->data.value);

getchar();getchar();

}

}

break;

//求右孩子节点

case 12:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

printf("要查询的值\n");

scanf("%d",&e);

if(RightChild(\*T,e)==NULL){

printf ("该值没有右孩子节点");

getchar();getchar();

}

else {

printf("要求节点的值为:%d",RightChild(\*T,e)->data.value);

getchar();getchar();

}

}

break;

//求左兄弟节点

case 13:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else

{

printf("要查询的值\n");

scanf("%d",&e);

if(LeftSibling(\*T,e)==NULL){

printf ("该值没有左兄弟节点");

getchar();getchar();

}

else {

printf("要求节点的值为:%d",LeftSibling(\*T,e)->data.value);

getchar();getchar();

}

}

break;

//求右兄弟节点

case 14:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

printf("要查询的值\n");

scanf("%d",&e);

if(RightSibling(\*T,e)==NULL){

printf ("该值没有右兄弟节点");

getchar();getchar();

}

else {

printf("要求节点的值为:%d",RightSibling(\*T,e)->data.value);

getchar();getchar();

}

}

break;

// 插入子树

case 15:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else

{ int \*p1,fl1=0,n1=0,LR;

Bitree T1;

T1=NULL;

p1=L->elem+1;

printf("输入要创建的新二叉树数\n");

scanf("%d",&j);

printf("输入要创建的新二叉树\n");

while (n1<j )

{ printf("输入第%d个数\n",n1+1);

scanf("%d",&fl1);

\*p1=fl1;

p1++;

n1++;

}

xx=0;

if (CreateBiTree(T1,(L->elem+1))==OK) {

printf("新二叉树创建成功！\n");

getchar();getchar();

}

else {printf("新二叉树创建失败！\n");

getchar();getchar();

}

printf("输入二叉树插入的节点e\n");

scanf("%d",&e);

printf("输入插入二叉树为左子树还是右子树LR(0 or 1 ）\n");

scanf("%d",&LR);

if(InsertChild(\*T,e,LR,T1)==OK){

printf("插入成功");

getchar();getchar();

}

else{ printf("插入失败");

getchar();getchar();

}

}

break;

// 删除子树

case 16:

if(a[d]==0){

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else {

int LR;

printf("输入二叉树删除的节点e\n");

scanf("%d",&e);

printf("输入删除二叉树为左子树还是右子树LR(0 or 1 ）\n");

scanf("%d",&LR);

DeleteChild(\*T,e,LR);

if(DeleteChild(\*T,e,LR)==OK){

printf("删除成功");

getchar();getchar();

}

else{

printf("删除失败");

getchar();getchar();

}

}

break;

//前序遍历

case 17:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

PreOrderTraverse(\*T);

getchar();getchar();

}

break;

//中序遍历

case 18:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

InOrderTraverse(\*T);

getchar();getchar();

}

break;

//后序遍历

case 19:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

PostOrderTraverse(\*T);

getchar();getchar();

}

break;

//层序遍历

case 20:

if(a[d]==0) {

printf("二叉树不存在！\n");

getchar();getchar();

}

else if(a[d]==2){

printf("二叉树为空！\n");

getchar();getchar();

}

else{

LevelOrderTraverse(\*T);

getchar();getchar();

}

break;

//文件输入

case 21:

if(a[d]==0)

printf("二叉树不存在！\n");

else if(a[d]==2)

printf("二叉树为空！\n");

else{ FILE \* fp;

printf("input file name: ");

scanf("%s", filename);

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

{

printf("File open error!\n");

}

res = LoadToFile(\*T, filename,fp);

if (res == 0)

printf("成功输出到文件!\n");

else

printf("文件打开失败!\n");

fclose(fp);

}

getchar();getchar();

break;

//读入文件

case 22:

if(a[d]==0)

printf("二叉树不存在！\n");

else if(a[d]==2)

printf("二叉树为空！\n");

else {

printf("input file name: ");

scanf("%s", filename);

xx=0;

res = LoadFromFile(\*T, filename);

if (res == 1)

printf("二叉树中已存在元素，请清空后再从文件载入!\n");

else

printf("载入成功!\n");

}

getchar();getchar();

break;

case 23:

printf("请输入你要操作的二叉树[1~3]:");

int m;

scanf("%d",&m);

if(m==1){

T=&L1;

d=1;

printf("切换成功！");

getchar();getchar();

}

else if(m==2){

T=&L2;

d=2;

printf("切换成功！");

getchar();getchar();

}

else if(m==3){

T=&L3;

d=3;

printf("切换成功！");

getchar();getchar();

}

else{

printf("切换失败!\n");

getchar();getchar();

}

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

附录4 图程序实现清单

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_CHAR 3

#define MAX\_VERTEX\_NUM 20

typedef int status;

typedef int InfoType;//弧相关的信息

typedef struct VertexType{

char key[MAX\_CHAR];

int value;

int isInit;

};

typedef enum{DG,DN}GraphKind;//{有向图，有向网}

typedef struct ArcNode{

int adjvex;//该弧指向的顶点的位置

struct ArcNode \*nextarc;//指向下一条弧的指针

InfoType \*info;//该弧相关信息的指针

};

typedef struct VNode{

VertexType data;//顶点信息

ArcNode \*firstarc;//指向第一条依附该顶点的弧的指针

}List[MAX\_VERTEX\_NUM];

typedef struct Graph{

List vertices;

int vexnum,arcnum;//图的当前顶点数和弧数

int type;//图的种类标志

};

typedef int QElemType;

typedef struct QNode{

QElemType data;

struct QNode \*next;

}\*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

int visited[MAX\_VERTEX\_NUM];

int count=1;

status LocateVex(Graph G,VertexType elem);

status CreatGraph(Graph&G){//创建图

int i,j;

int weigh;//边的权

VertexType vtail,vhead;

int headloc,tailloc;

ArcNode \*cur,\*last;

printf("请输入要创建图的类型：0（有向图）/1（有向网）：");

scanf("%d",&G.type);

printf("请输入图的顶点数：");

scanf("%d",&G.vexnum);

printf("请输入图的边数:");

scanf("%d",&G.arcnum);

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

printf("请输入第%d个顶点的值:",i+1);

scanf("%s",G.vertices[i].data.key);

G.vertices[i].firstarc=NULL;

G.vertices[i].data.isInit=0;

}

if(G.type==0){

printf("请依次输入每条弧的弧尾和弧头，中间用回车隔开\n");

}

else{

printf("请依次输入每条弧的权值、弧尾和弧头，中间用回车隔开\n");

}

for(i=0;i<G.arcnum;i++){

printf("下面输入第%d组弧的参数\n",i+1);

if(G.type==0){

printf("请输入弧尾（字符串）：");

scanf("%s",vtail.key);

printf("请输入弧头（字符串）：");

scanf("%s",vhead.key);

}

else{

printf("请输入弧尾（字符串）：");

scanf("%s",vtail.key);

printf("请输入弧头（字符串）：");

scanf("%s",vhead.key);

printf("请输入权值（整数）：");

scanf("%d",&weigh);

}

if((tailloc=LocateVex(G,vtail))==-1)

return ERROR;

if((headloc=LocateVex(G,vhead))==-1)

return ERROR;

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

cur->adjvex=headloc;

if(G.type==0){

cur->info=NULL;

}

else{

cur->info=(int\*)malloc(sizeof(int));

\*(cur->info)=weigh;

}

cur->nextarc=G.vertices[tailloc].firstarc;

G.vertices[tailloc].firstarc=cur;

}

return OK;

}

status DestroyGraph(Graph&G){//销毁图

G.vexnum=0;

G.arcnum=0;

ArcNode \*cur,\*temp;

int i;

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

cur=G.vertices[i].firstarc;

G.vertices[i].firstarc=NULL;

while(cur!=NULL){

temp=cur->nextarc;

if(G.type==1){

free(cur->info);

}

free(cur);

cur=temp;

}

}

return OK;

}

status LocateVex(Graph G,VertexType elem){//查找顶点

int i;

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

if(strcmp(elem.key,G.vertices[i].data.key)==0){

return i;

}

}

return -1;

}

status GetVex(Graph G,VertexType elem,int &value){//获得结点值

int i;

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

if(strcmp(elem.key,G.vertices[i].data.key)==0){

if(G.vertices[i].data.isInit==0){

return INFEASIBLE;

}

else{

value=G.vertices[i].data.value;

return OK;

}

}

}

return FALSE;

}

status PutVex(Graph&G,VertexType elem,int value){//给结点赋值

int i;

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

if(strcmp(elem.key,G.vertices[i].data.key)==0){

G.vertices[i].data.isInit=1;

G.vertices[i].data.value=value;

return OK;

}

}

return FALSE;

}

int FirstAdjVex(Graph G,VertexType elem){//获得第一个邻接结点

int n;

n=LocateVex(G,elem);

if(n==-1)

return -2;

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

if(p!=NULL){

return p->adjvex;

}

else{

return -1;

}

}

int NextAdjVex(Graph G,VertexType elem,VertexType elem2){//获得下一个邻接结点

int n,n2;

n=LocateVex(G,elem);

n2=LocateVex(G,elem2);

if(n==-1||n2==-1)

return -1;

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

if(p!=NULL){

while(p){

if(p->adjvex==n2){

if(p->nextarc!=NULL){

return p->nextarc->adjvex;

}

else{

return -4;//该结点不存在下一个结点

}

}

p=p->nextarc;

}

return -3;//顶点elem不与elem2相连

}

else{

return -2;//顶点elem不存在邻接结点

}

}

status InsertVex(Graph&G,VertexType elem){//添加一个新顶点

strcpy(G.vertices[G.vexnum].data.key,elem.key);

G.vertices[G.vexnum].data.isInit=0;

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

G.vexnum++;

return OK;

}

status DeleteVex(Graph &G,VertexType elem){//删除一个结点以及和它相关

int n,i;

ArcNode \*temp,\*q,\*pretemp;

n=LocateVex(G,elem);

if(n==-1)

return FALSE;

temp=G.vertices[n].firstarc;

while(temp!=NULL){

q=temp;

if(G.type==-1){

free(temp->info);

}

temp=temp->nextarc;

free(q);

G.arcnum--;

}

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

G.vertices[i]=G.vertices[i+1];

}

G.vexnum--;

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

temp=G.vertices[i].firstarc;

if(temp!=NULL&&temp->adjvex==n){

G.vertices[i].firstarc->nextarc=G.vertices[i].firstarc->nextarc->nextarc;

if(G.type==1){

free(temp->info);

}

free(temp);

G.arcnum--;

}

else if(temp==NULL){

G.arcnum=G.arcnum;

}

else{

if(temp->adjvex>n){

temp->adjvex--;

}

pretemp=temp;

temp=temp->nextarc;

while(temp!=NULL){

q=temp;

if(temp->adjvex>n){

temp->adjvex--;

}

if(temp->adjvex==n){

pretemp->nextarc=pretemp->nextarc->nextarc;

temp=temp->nextarc;

if(G.type==1){

free(q->info);

}

free(q);

G.arcnum--;

}

pretemp=pretemp->nextarc;

temp=temp->nextarc;

}

}

}

return OK;

}

status InsertArc(Graph&G,VertexType elem,VertexType elem2){//c插入一条邻边

int n,n2;

n=LocateVex(G,elem);

n2=LocateVex(G,elem2);

if(n==-1||n2==-1)

return ERROR;

ArcNode \*p;

p=G.vertices[n].firstarc;

while(p!=NULL){

if(p->adjvex==n2){

return -2;

}

p=p->nextarc;

}

ArcNode \*temp;

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

temp->adjvex=n2;

if(G.type==1){

temp->info=(int\*)malloc(sizeof(int));

int weigh;

printf("\n请输入插入边的权值:");

scanf("%d",&weigh);

\*(temp->info)=weigh;

}

else if(G.type==0){

temp->info=NULL;

}

temp->nextarc=G.vertices[n].firstarc;

G.vertices[n].firstarc=temp;

G.arcnum++;

return OK;

}

status DeleteArc(Graph&G,VertexType elem,VertexType elem2){//删除一条邻边

int n,n2;

n=LocateVex(G,elem);

n2=LocateVex(G,elem2);

if(n==-1||n2==-1)

return ERROR;

ArcNode \*p,\*q,\*temp;

p=G.vertices[n].firstarc;

int found=0;

if(p==NULL)

return ERROR;

if(p->adjvex==n2){

q=p;

G.vertices[n].firstarc=p->nextarc;

if(G.type==1){

free(q->info);

}

found=1;

free(q);

G.arcnum--;

}

else{

while(p!=NULL){

q=p;

p=p->nextarc;

if(p->adjvex==n2){

found=1;

q->nextarc=p->nextarc;

if(G.type==1){

free(p->info);

}

free(p);

break;

}

}

}

if(!found){

return INFEASIBLE;

}

return OK;

}

void DFS(Graph G,int num){//进行深度遍历

ArcNode \*temp;

int t;

printf("第%d个结点为：%s\n",count,G.vertices[num].data.key);

count++;

visited[num]=TRUE;

for(temp=G.vertices[num].firstarc;temp!=NULL;temp=temp->nextarc){

t=temp->adjvex;

if(visited[t]!=1)

DFS(G,t);

}

}

status DFSTraverse(Graph G){

int i;

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

visited[i]=FALSE;

}

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

if(visited[i]!=1)

DFS(G,i);

}

return OK;

}

status InitQueue(LinkQueue &Q){

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

if(!Q.front)

exit(OVERFLOW);

Q.front->next=NULL;

return OK;

}

status QueueEmpty(LinkQueue Q){

if(Q.front==Q.rear)

return TRUE;

else

return FALSE;

}

status EnQueue(LinkQueue &Q,QElemType e){

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if(!p)

exit(OVERFLOW);

p->data=e;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

return OK;

}

status DeQueue(LinkQueue &Q,QElemType &e){

QueuePtr p;

if(Q.front==Q.rear)

return ERROR;

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 BFSTraverse(Graph G,LinkQueue Q){

int v,u;

ArcNode \*temp;

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

visited[v]=FALSE;

}

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

if(!visited[v]){

visited[v]=TRUE;

printf("第%d个结点为: %s\n",v+1,G.vertices[v].data.key);

EnQueue(Q,v);

while(!QueueEmpty(Q)){

DeQueue(Q,u);

for(temp=G.vertices[u].firstarc;temp!=NULL;temp=temp->nextarc){

if(visited[temp->adjvex]!=1){

visited[temp->adjvex]=TRUE;

printf("第%d个结点为：%s\n",++count,G.vertices[temp->adjvex].data.key);

EnQueue(Q,temp->adjvex);

}

}

}

}

}

return OK;

}

status LoadToFile(Graph \*G, char \* filename)

{

int i;

int number = -1;

ArcNode \* node;

FILE \* fp = fopen(filename, "w");

if (fp == NULL)

return ERROR; //file open error

fwrite(&(G->vexnum), sizeof(int), 1, fp);

fwrite(&(G->arcnum), sizeof(int), 1, fp);

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

fwrite(&(G->vertices[i].data), sizeof(VertexType), 1, fp);

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

{

node = G->vertices[i].firstarc;

while (node != NULL)

{

fwrite(&(node->adjvex), sizeof(int), 1, fp);

fwrite(&(node->info), sizeof(int), 1, fp);

node = node->nextarc;

}

fwrite(&number, sizeof(int), 1, fp);

fwrite(&number, sizeof(int), 1, fp);

}

fclose(fp);

return OK;

}

status LoadFromFile(Graph \*G, char \* filename)

{

int i;

ArcNode \* node;

ArcNode \* tempnode;

FILE \* fp = fopen(filename, "r");

if (fp == NULL)

return ERROR; //file open error

fread(&(G->vexnum), sizeof(int), 1, fp);

fread(&(G->arcnum), sizeof(int), 1, fp);

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

fread(&(G->vertices[i].data), sizeof(VertexType), 1, fp);

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

{

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

fread(&(tempnode->adjvex), sizeof(int), 1, fp);

fread(&(tempnode->info), sizeof(int), 1, fp);

if (tempnode->adjvex == -1)

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

else

{

G->vertices[i].firstarc = tempnode;

node = tempnode;

while (1)

{

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

fread(&(tempnode->adjvex), sizeof(int), 1, fp);

fread(&(tempnode->info), sizeof(int), 1, fp);

if (tempnode->adjvex == -1)

{

node->nextarc = NULL;

break;

}

node->nextarc = tempnode;

node = node->nextarc;

}

}

}

fclose(fp);

return OK;

}

int main(void){

int reCreate=2;

Graph \*G;

Graph G1,G2,G3;

G1.vexnum=0;

G2.vexnum=0;

G3.vexnum=0;

G=&G1;

VertexType elem,elem2;

VertexType insertElem;

int value=0;

int Initial=0;

int i,j;

QElemType e;

LinkQueue Q;

char filename[30];

int op=1;

while(op){

system("cls"); printf("\n\n");

printf(" 基 于 邻 接 表 的 图 演 示 系 统 \n");

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

printf(" 1. CreateGraph 7. NextAdjVex\n");

printf(" 2. DestroyGraph 8. InsertVex\n");

printf(" 3. LocateVex 9. DeleteVex \n");

printf(" 4. GetVex 10. InsertArc\n");

printf(" 5. PutVex 11. DeleteArc\n");

printf(" 6. FirstAdjVex 12. DFSTrabverse\n");

printf(" 13.BFSTraverse 14. LoadToFile\n");

printf(" 15.LoadFromFile 16. ChangeGraph\n");

printf(" 0.exit\n");

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

printf(" 请选择你的操作[0~13]:");

scanf("%d",&op);

switch(op){

case 1:

if(Initial){

printf("图已创建，是否覆盖？（1/0--YES/NO）\n");

scanf("%d",&reCreate);

}

if(reCreate==0)

break;

if(CreatGraph(\*G)==OK){

printf("创建成功！\n");

getchar();getchar();

Initial=1;

}

else{

printf("创建失败!\n");

getchar();getchar();

}

break;

case 2:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

if(DestroyGraph(\*G)==OK){

printf("销毁成功！\n");

Initial=0;

getchar();getchar();

}

else{

printf("销毁失败！\n");

getchar();getchar();

}

break;

case 3:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你要查找的顶点：");

scanf("%s",elem.key);

if((i=LocateVex(\*G,elem))>-1){

printf("该结点位于邻接表的第%d个，其标记为%d\n",i+1,i);

getchar();getchar();

}

else{

printf("未找到该顶点！\n");

getchar();getchar();

}

break;

case 4:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你要查找的顶点：");

scanf("%s",elem.key);

if((i=GetVex(\*G,elem,value))==OK){

printf("顶点%s的值为%d\n",elem.key,value);

getchar();getchar();

}

else if(i=INFEASIBLE){

printf("该顶点还未赋值！\n");

getchar();getchar();

}

else if(i==FALSE){

printf("未找到该顶点！\n");

getchar();getchar();

}

break;

case 5:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想赋值的顶点：");

scanf("%s",elem.key);

printf("请输入你要赋的值：");

scanf("%d",&value);

if((i=PutVex(\*G,elem,value))==OK){

printf("赋值成功！顶点%s的值为%d\n",elem.key,value);

getchar();getchar();

}

else if(i==FALSE){

printf("未找到该顶点！\n");

getchar();getchar();

}

break;

case 6:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想查找的顶点：");

scanf("%s",elem.key);

if((i=FirstAdjVex(\*G,elem))>=0){

printf("顶点%s的邻接结点为%s\n",elem.key,G->vertices[i].data.key);

getchar();getchar();

}

else if(i==-2){

printf("未找到该结点！\n");

getchar();getchar();

}

else if(i==-1){

printf("该结点无邻接结点\n");

getchar();getchar();

}

else{

printf("查找失败！\n");

getchar();getchar();

}

break;

case 7:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想查找的顶点：");

scanf("%s",elem.key);

printf("请输入与该顶点相连的一个顶点：");

scanf("%s",elem2.key);

if((i=NextAdjVex(\*G,elem,elem2))>=0){

printf("所查询的结点为%s\n",G->vertices[i].data.key);

getchar();getchar();

}

else if(i==-1){

printf("未找到该结点！\n");

getchar();getchar();

}

else if(i==-2){

printf("该结点不存在邻接结点！\n");

getchar();getchar();

}

else if(i==-3){

printf("所输入的顶点%s不与%s相连\n",elem2.key,elem.key);

getchar();getchar();

}

else if(i==-4){

printf("顶点%s不存在下一个结点\n",elem2.key);

getchar();getchar();

}

break;

case 8:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想插入的顶点：");

scanf("%s",insertElem.key);

if(InsertVex(\*G,insertElem)==OK){

printf("插入成功！\n");

getchar();getchar();

}

else{

printf("插入失败！\n");

getchar();getchar();

}

break;

case 9:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想删除的顶点：");

scanf("%s",elem.key);

if(DeleteVex(\*G,elem)==OK){

printf("删除成功！\n");

getchar();getchar();

}

else{

printf("删除失败！\n");

getchar();getchar();

}

break;

case 10:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想插入边的尾顶点：");

scanf("%s",elem.key);

printf("请输入你想插入边的头顶点：");

scanf("%s",elem2.key);

if(InsertArc(\*G,elem,elem2)==OK){

printf("插入成功！\n");

getchar();getchar();

}

else{

printf("插入失败！\n");

getchar();getchar();

}

break;

case 11:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

printf("请输入你想删除弧的弧尾顶点：");

scanf("%s",elem.key);

printf("请输入你想删除弧的弧头顶点：");

scanf("%s",elem2.key);

if(DeleteArc(\*G,elem,elem2)==OK){

printf("删除成功！\n");

getchar();getchar();

}

else{

printf("删除失败！\n");

getchar();getchar();

}

break;

case 12:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

count=1;

DFSTraverse(\*G);

getchar();getchar();

break;

case 13:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

InitQueue(Q);

count=1;

BFSTraverse(\*G,Q);

getchar();getchar();

break;

case 14:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

else

{

printf("请输入存储的文件名: ");

scanf("%s", filename);

if (LoadToFile(G,filename) == ERROR){

printf("文件打开失败!\n");

getchar();getchar();

}

else{

printf("成功输出到文件!\n");

getchar();getchar();

}

}

break;

case 15:

if(!Initial){

printf("请先创建图！\n");

getchar();getchar();

break;

}

else{

printf("请输入读取的文件名: ");

scanf("%s", filename);

if (LoadFromFile(G,filename) == ERROR){

printf("文件打开失败!\n");

getchar();getchar();

}

else{

printf("成功载入有向网!\n");

getchar(); getchar();

}

}

break;

case 16:

int m;

printf("请输入你想操作的图[1~3]:");

scanf("%d",&m);

if(m==1){

G=&G1;

if(G->vexnum==0){

Initial=0;

}

else{

Initial=1;

}

printf("切换成功！\n");

getchar();getchar();

}

else if(m==2){

G=&G2;

if(G->vexnum==0){

Initial=0;

}

else{

Initial=1;

}

printf("切换成功！\n");

getchar();getchar();

}

else if(m==3){

G=&G3;

if(G->vexnum==0){

Initial=0;

}

else{

Initial=1;

}

printf("切换成功！\n");

getchar();getchar();

}

else{

printf("切换失败！\n");

getchar();getchar();

}

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()