

**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CS1806**

**学 号： U201814655**

**姓 名： 杨雨鑫**

**指导教师： 李剑军**

**报告日期： 2019年 12月 24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](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 6](#_Toc458159883)

[1.4 系统测试 7](#_Toc458159884)

[1.5 实验小结 13](#_Toc458159884)

[2 基于链式存储结构的线性表实现 14](#_Toc458159885)

[2.1 问题描述 14](#_Toc458159886)

[2.2 系统设计 14](#_Toc458159887)

[2.3 系统实现 18](#_Toc458159888)

[2.4 系统测试 19](#_Toc458159889)

[2.5 实验小结 24](#_Toc458159884)

[3 基于二叉链表的二叉树实现 26](#_Toc458159890)

[3.1 问题描述 26](#_Toc458159891)

[3.2 系统设计 26](#_Toc458159892)

[3.3 系统实现 30](#_Toc458159893)

[3.4 系统测试 33](#_Toc458159894)

[3.5 实验小结 42](#_Toc458159884)

[4 基于二叉链表的二叉树实现 43](#_Toc458159895)

[4.1 问题描述 43](#_Toc458159896)

[4.2 系统设计 43](#_Toc458159897)

[4.3 系统实现 48](#_Toc458159898)

[4.4 系统测试 49](#_Toc458159899)

[4.5 实验小结 56](#_Toc458159884)

[参考文献 58](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 59](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 72](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 86](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 107](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，实现增删查改等基本运算。

设计的函数需要考虑健壮性，对于一些特殊情况的考虑要到位。额外任务：可以实现多个线性表的管理，并且可以对多个表进行管理操作。需要完成的功能：实验中需要实现顺序表的创建、销毁，顺序表相关的查询与处理操作，元素的查找、插入与删除、文件的读取以及保存。

实验要求：通过实验达到 ⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。(4)熟悉对于文件读取操作的步骤

## 1.2 系统设计

顺序表结构体定义：

typedef struct

{ //顺序表（顺序结构）的定义

ElemType \*elem;

int length;

int listsize;

} SqList;

函数定义：

（1）**status InitList(SqList\* L);**

初始条件：线性表L不存在，输入的命令为1，需要建立新的线性表。

算法设计：分配初始储存空间，如果因内存不足而分配失败则返回OVERFLOW。初始化表长为0，表容量为LIST\_INIT\_SIZE。

操作结果：创建一个空的顺序表。

（2）**status DestroyList(SqList\* L);**

初始条件：线性表L已存在，输入的操作数为2。

算法设计：释放存储空间并销毁顺序表，使L->elem指向NULL，同时表的大小清0。

操作结果：销毁一个已经存在的顺序表L。

（3）**status ClearList(SqList\* L);**

初始条件：线性表L已存在，输入的操作数为3。

算法设计：线性表L的表长元素赋值为0。

操作结果：清空顺序表L中的全部元素。

（4）**status ListEmpty(SqList\* L);**

初始条件：线性表L已存在，输入的操作数为4。

算法设计：根据线性表L的表长，判断表是否为空。

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）**int ListLength(SqList\* L);**

初始条件：线性表L已存在，输入的操作数为5。

算法设计：返回线性表L的表长，即返回SqList中的length元素的值。

操作结果：返回L中数据元素的个数。

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

初始条件：线性表已存在，1≤i≤ListLength(L)，输入的操作数为6。

算法设计：先判断序号i是否在有效范围内（1≤i≤length），根据序号找到线性表L中第i个元素的地址，并将地址储存的元素赋值给指针e指向的值。

操作结果：指针e指向的元素即为L中第i个数据元素。

（7）**int LocateElem(SqList \*L, ElemType e);**

初始条件：线性表L已存在，输入的操作数为7。

算法设计：遍历线性表L，找到第一个和元素e的相等的元素。

操作结果：返回L中第1个与e相等的的数据元素的位置，若与之相等的数据元素不存在，则返回值为 ERROR。

（8）**status PriorElem(SqList\* L, ElemType cur, ElemType\* &pre\_e);**

初始条件：线性表L已存在，输入的操作数为8。

算法设计：遍历线性表L找到第一个和元素cur的相等的元素，如果其有前驱，将前驱的元素赋值给 pre\_e所存储的数据，函数返回OK；否则函数返回ERROR，pre\_e 无意义。

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e 返回它的 前驱元素的地址，否则操作失败，pre\_e 无定义。

（9）**status NextElem(SqList \*L, ElemType cur, ElemType \*next\_e);**

初始条件：线性表L已存在，输入的操作数为9。

算法设计：遍历线性表L找到第一个和元素cur的相等的元素，如果其有后继，将后继的元素赋值给 next\_e所存储的数据，函数返回OK；否则函数返回ERROR，next\_e无意义。

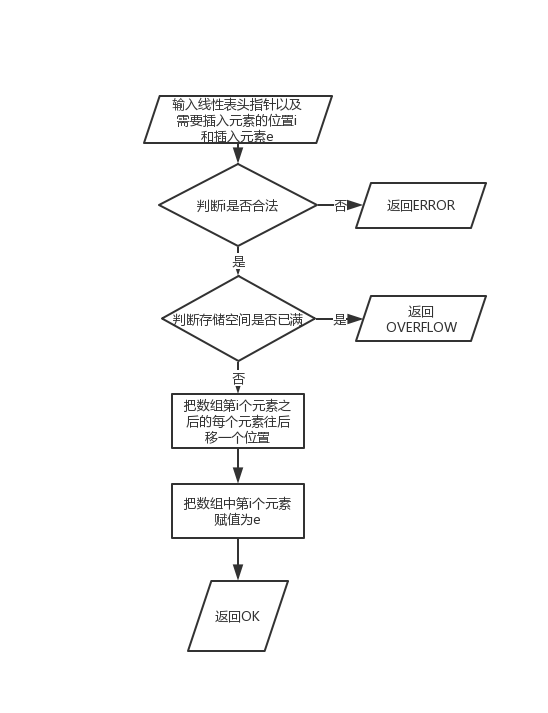
操作结果：若cur是L的数据元素，且不是最后一个，则用next\_e返回它的后继元素的地址，否则操作失败，next\_e无定义。

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

初始条件：线性表已存在，1≤i≤L->length+1，输入的操作数为10。

算法设计：先判断输入的i值是否合法，如果此时线性表的存储空间已满，需要realloc更大的空间之后在第i个位置插入元素e，并且把i位置后的每一个元素都往后移一个位置，最后表长加一。

操作结果：在L的第i个位置上插入新的数据元素e（i为length+1则表示在末尾插入），length的长度加1。（流程图见下图）

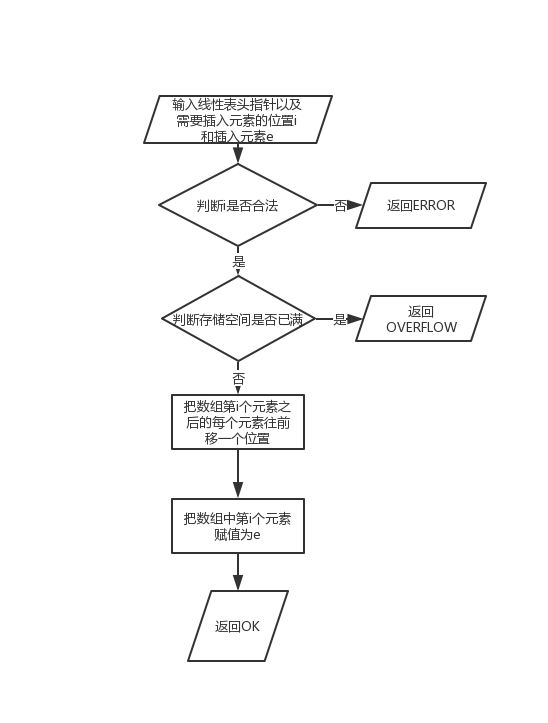


（11）**status ListDelete(SqList \*L, ElemType \*e, int i);**

初始条件：线性表已存在，1≤i≤List->Length,输入的操作数为11。

算法设计：先判断index是否在合理范围之内（1≤i≤Length），然后将第index个位置的值赋给指针引用变量e，之后第i个位置后面的全部元素前移一个位置。

操作结果：删除L的位置为i的数据元素，用e返回其值，同时L的长度减1。（流程图见下图）



（12）**status ListTrabverse(SqList L);**

初始条件：线性表L已存在，输入的操作数为12。

算法设计：遍历表L中的每个元素，并且打印出来。

操作结果：依次输出表L中的每个元素的值。

（13）**status SaveList(SqList \*L);**

初始条件：线性表已存在，输入的操作数为13。

算法设计：遍历以L->elem为头节点的每个顺序表，将表中信息和元素输入到文件并保存。

操作结果：保存当前的操作表。

（14）**status LoadList(SqList \*L);**

初始条件：线性表已存在，输入的操作数为14。

算法设计：读取文件中保存的顺序表的信息，把相应的数据存储到该操作表的数据域中。

操作结果：重新读取已保存的表，给操作表赋值。

（15）**status InputData(SqList \*L);**

初始条件：线性表已存在，输入的操作数为15。

算法设计：输入一串字符串，把其中的数字作为数据读入操作表中。

操作结果：通过直接手写输入数据，直接给操作表赋值。

## 1.3 系统实现

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

{

//在顺序线性表L中第i个位置之前插入新的元素e

//i的合法值为1<=i<=L->length+1

ElemType \*newbase;

ElemType \*p, \*q;

if (!L->elem)

return INFEASTABLE;

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

return ERROR; //i值不合法

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

{ //当前存储空间已满，增加分配

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

if (!newbase)

return OVERFLOW; //存储分配失败

L->elem = newbase; //新的基址

L->listsize += LISTINCREMENT; //增加存储容量

}

q = &(L->elem[i - 1]); //q为插入位置

for (p = &(L->elem[L->length]); p >= q; --p) //插入位置及之后的元素右移

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

\*q = e; //插入e

++L->length; //表长增1

return OK;

}

时间和空间复杂度分析：假设表长为n，则找到第i个元素所需要的时间为1，所以时间复杂度为O(1)。插入元素之后，会将第i个元素后面的每一个元素往后移一位，平均需要移动n-1/2，故时间复杂度为O(n),综上所述，该插入函数的时间复杂度为O(n)。

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

{ //删除表中的第i个元素，并用e返回其值

ElemType \*p, \*q;

if (!L->elem)

return INFEASTABLE;

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

return ERROR; //i值不合法

p = &(L->elem[i - 1]); //p为被删除元素的位置

\*e = \*p; //被删除元素的值赋给e

q = L->elem + L->length - 1; //表尾元素的位置

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

\*(p - 1) = \*p; //被删除元素之后的元素左移

--L->length; //表长减一

return OK;

}

时间和空间复杂度分析：假设表长为n，则找到第i个元素所需要的时间为1，所以时间复杂度为O(1)。插入元素之后，会将第i个元素后面的每一个元素往前移一位，平均需要移动n-1/2，故时间复杂度为O(n),综上所述，该删除函数的时间复杂度为O(n)。

**1.4 系统测试**

测试中使用的测试数据：

表1-1 测试用例

|  |  |
| --- | --- |
| **表ID** | **表中元素（按顺序排列）** |
| **1** | **1，2，3，4，5，6，7，8** |
| **2** | **8，9，3，6，7，4** |
| **3** | **34，12，55，15，9** |

1) 测试函数：InitList

测试步骤及结果如表 1-2 所示

表1-2 InitList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **无线性表时建立表** | **在无表1时，输入命令1** | **建立ID为1的表。** |  |
| **有线性表时建立表** | **在已有表1时，输入命令1** | **重新建立ID为1的表** |  |

2) 测试函数：DestroyList

测试步骤及结果如表 1-3 所示

表1-3 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **有线性表时销毁表** | **在已经创建了id为1的表的基础上，输入命令2** | **销毁ID为1的表。** |  |
| **无线性表时销毁表** | **在没创建id为1的表的基础上，输入命令2** | **无任何操作** |  |

3) 测试函数：ClearList

测试步骤及结果如表 1-4 所示

表1-4 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时清空表** | **在id为1的表中有元素的前提下，输入命令3** | **清空ID为1的表** |  |
| **表存在，无元素时清空表** | **在id为1的表中没有元素的前提下，输入命令3** | **ID为1的表仍为空** |  |
| **表不存在时清空表** | **不存在id为0的表时，输入命令3** | **操作失败** |  |

4) 测试函数：ListEmpty

测试步骤及结果如表 1-5 所示

表1-5 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时判断表是否为空** | **在id为1的表中有元素的前提下，输入命令4** | **得出表1非空** |  |
| **表存在，无元素时判断表是否为空** | **在id为1的表中没有元素的前提下，输入命令4** | **得出表1为空** |  |
| **表不存在时判断表是否为空** | **不存在id为1的表时，输入命令4** | **操作失败** |  |

5) 测试函数：ListLength

测试步骤及结果如表 1-6 所示

表1-6 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在时获得表长** | **在有id为1的表的前提下，输入命令5** | **得出表1的长度为8** |  |
| **表不存在时判断表是否为空** | **不存在id为1的表时，输入命令5** | **操作失败** |  |

6) 测试函数：GetElem

测试步骤及结果如表 1-7 所示

表1-7 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时获得表的首元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为1** | **得到表1的首元素** |  |
| **表存在，有元素时获得表的中间元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为3** | **得到表1的第3个元素** |  |
| **表存在，有元素时获得表的尾元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为8** | **得到表1的尾元素** |  |
| **表存在，有元素时获得超出表范围的元素** | **在id为1的表中有元素的前提下，输入命令6，输入元素的位置为9** | **序号超出范围，获取元素失败** |  |
| **表不存在时获得表的元素** | **不存在id为0的表时，输入命令6，输入元素的位置为1** | **操作失败** |  |

7) 测试函数：LocateElem

测试步骤及结果如表 1-8 所示

表1-8 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有对应元素时，查找相应的元素** | **在id为2的表中有元素7的前提下，输入命令7 7** | **成功在表2中找到元素7，其序号为5** |  |
| **表存在，没有对应元素时，查找相应的元素** | **在id为2的表中没有元素100的前提下，输入命令7 100** | **查找失败** |  |
| **表不存在时查找元素** | **不存在id为2的表时，输入命令7 8** | **操作失败** |  |

8) 测试函数：PriorElem

测试步骤及结果如表 1-9 所示

表1-9 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，元素有前驱时获取对应元素的前驱** | **在表2中元素9 有前驱的前提下，输入命令8 9** | **找到表2中元素9的前驱为8** |  |
| **表存在，元素无前驱时获取对应元素的前驱** | **在表2中元素8没有前驱的前提下，输入命令8 8** | **查找元素8的前驱失败** |  |
| **表不存在时，获取对应元素的前驱** | **不存在id为2的表时，输入命令8 6** | **操作失败** |  |

9) 测试函数：NextElem

测试步骤及结果如表 1-10 所示

表1-10 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，元素有后继时获取对应元素的前驱** | **在表3中元素34有后继的前提下，输入命令9 34** | **找到表3中元素34的后继为12** |  |
| **表存在，元素无后继时获取对应元素的前驱** | **在表3中元素9没有后继的前提下，输入命令9 9** | **查找元素9的后继失败** |  |
| **表不存在时，获取对应元素的后继** | **不存在id为3的表时，输入命令9 55** | **操作失败** |  |

10) 测试函数：ListInsert

测试步骤及结果如表 1-11 所示

表1-11 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，在表尾插入元素** | **在已构建好的表3中表头插入元素，输入命令10 6 18** | **表3中的序列由34→12→55→15→9**  **变为34→12→55→15→18→9** |  |
| **表存在，在表中间插入元素** | **在已构建好的表3中表头插入元素，输入命令10 3 18** | **表3中的序列由34→12→55→15→9**  **变为34→12→18→55→15→9** |  |
| **表存在，在表头插入元素** | **在已构建好的表3中表头插入元素，输入命令10 1 18** | **表3中的序列由34→12→55→15→9**  **变为18→34→12→55→15→9** |  |
| **表存在，在超出范围时插入元素** | **在已构建好的表3中表头插入元素，输入命令10 8 18** | **操作错误** |  |
| **表不存在时，插入元素** | **不存在id为3的表时，输入命令10 2 18** | **操作失败** |  |

11) 测试函数：ListDelete

测试步骤及结果如表 1-12 所示

表1-12 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，在表头删除元素** | **在已构建好的表3中表头插入元素，输入命令11 1** | **表3中的序列由34→12→55→15→9**  **变为12→55→15→9** |  |
| **表存在，在表中间删除元素** | **在已构建好的表3中表中间插入元素，输入命令11 3** | **表3中的序列由34→12→55→15→9**  **变为34→12→15→9** |  |
| **表存在，在表末尾删除元素** | **在已构建好的表3中表尾插入元素，输入命令11 5** | **表3中的序列由34→12→55→15→9**  **变为34→12→55→15** |  |
| **表存在，在超出范围时删除元素** | **在已构建好的表3中插入元素，输入命令11 8** | **操作失败** |  |
| **表不存在时，删除元素** | **不存在id为3的表时，输入命令11 1** | **操作失败** |  |

12) 测试函数：ListTraverse

测试步骤及结果如表 1-13 所示

表1-13 ListTraverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，**遍历**表中元素** | **在表1存在的时，输入命令12** | **遍历打印ID为1的表中元素** |  |
| **表不存在时，遍历表中元素** | **不存在表1时，输入命令12** | **操作失败** |  |

13) 测试函数：SaveList & LoadList

测试步骤及结果如表 1-14 所示

表1-14 保存/读取函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **测试保存功能** | **存在表1、表2、表3时，输入 14 文件名命名为yyx** | **将全部线性表保存到文件yyx** |  |
| **测试读取功能** | **退出程序，重新打开程序，输入13，读取的文件名为yyx，输入12** | **从文件中读入保存的线性表，读入后的表1的元素应该与之前完全相同** |  |

结果分析：实验测试结果说明了这次试验所编写的程序可以实现任务书上所有的功能，并且额外实现的多表管理。

## 1.5 实验小结

这次试验是基于顺序表实现的线性表，大部分功能函数编写过程中，自己开始并没有考虑到一些特殊情况，导致开始调试的时候经常遇到程序报错死机的情况。后来，在仔细阅读了书上的函数后，发现书上的代码考虑的情况更加全面，自己也逐渐学会如何编写较为严谨的函数，对测试用例的理解更加重视了。

这次主要的难点是在文件读取这个环节上，通过老师给我的部分函数以及查阅一些网上的资料之后，我成功编写了二进制文件的读写函数。另外，在实现多表管理这个额外功能时，我只用了静态的结构体数组来实现，并没有使用指针实现动态的多表管理，对于空间的利用率还不是考虑的特别到位，准备在下一次链表实现线性表的实验中使用指针来优化这个缺点。

最后，十分感谢老师和助教对我的帮助。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用单链表作为线性表的物理结构，实现增删查改等基本运算。

设计的函数需要考虑健壮性，对于一些特殊情况的考虑要到位。额外任务：可以实现多个线性表的管理，并且可以对多个表进行管理操作。需要完成的功能：实验中需要实现单链表的创建、销毁，顺序表相关的查询与处理操作，元素的查找、插入与删除、文件的读取以及保存。

实验要求：通过实验达到 ⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用单链表,熟练掌握线性表的基本运算的实现。(4)熟悉对于文件读取操作的步骤

## 2.2 系统设计

单链表节点定义：

typedef struct ListNode

{ //链表元素结构体

ElemType elem;

struct ListNode \*next;

} ListNode,\*ListNodePtr;

其中表头结点的数据域用来存储表的大小。

函数定义：

（1）**status InitList(ListNodePtr L);**

初始条件：单链表L不存在，输入的命令为1，需要建立新的单链表。

算法设计：分配初始储存空间，如果因内存不足而分配失败则返回OVERFLOW。初始化的表只有一个空表头。

操作结果：创建一个空的顺序表。

（2）**status DestroyList(ListNodePtr L);**

初始条件：单链表L已存在，输入的操作数为2。

算法设计：释放存储空间并销毁顺序表，使表头结点的next指针指向NULL，同时表的大小清0。

操作结果：销毁一个已经存在的顺序表L。

（3）**status ClearList(ListNodePtr L);**

初始条件：单链表L已存在，输入的操作数为3。

算法设计：单链表L的表长元素赋值为0。

操作结果：清空顺序表L中的全部元素。

（4）**status ListEmpty(ListNodePtr L);**

初始条件：单链表L已存在，输入的操作数为4。

算法设计：根据单链表L的表长，判断表是否为空。

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）**int ListLength(ListNodePtr L);**

初始条件：单链表L已存在，输入的操作数为5。

算法设计：返回单链表L的表长，即返回表头结点中的elem元素的值。

操作结果：返回L中数据元素的个数。

（6）**status GetElem(ListNodePtr L, int i, ElemType\* e);**

初始条件：单链表已存在，1≤i≤ListLength(L)，输入的操作数为6。

算法设计：先判断序号i是否在有效范围内（1≤i≤length），根据序号找到单链表L中第i个元素的地址，并将地址储存的元素赋值给指针e指向的值。

操作结果：指针e指向的元素即为L中第i个数据元素。

（7）**int LocateElem(ListNodePtr L, ElemType e);**

初始条件：单链表L已存在，输入的操作数为7。

算法设计：遍历单链表L，找到第一个和元素e的相等的元素。

操作结果：返回L中第1个与e相等的的数据元素的位置，若与之相等的数据元素不存在，则返回值为 ERROR。

（8）**status PriorElem(ListNodePtr L, ElemType cur, ElemType\* &pre\_e);**

初始条件：单链表L已存在，输入的操作数为8。

算法设计：遍历单链表L找到第一个和元素cur的相等的元素，如果其有前驱，将前驱的元素赋值给 pre\_e所存储的数据，函数返回OK；否则函数返回ERROR，pre\_e 无意义。

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e 返回它的 前驱元素的地址，否则操作失败，pre\_e 无定义。

（9）**status NextElem(ListNodePtr L, ElemType cur, ElemType \*next\_e);**

初始条件：单链表L已存在，输入的操作数为9。

算法设计：遍历单链表L找到第一个和元素cur的相等的元素，如果其有后继，将后继的元素赋值给 next\_e所存储的数据，函数返回OK；否则函数返回ERROR，next\_e无意义。

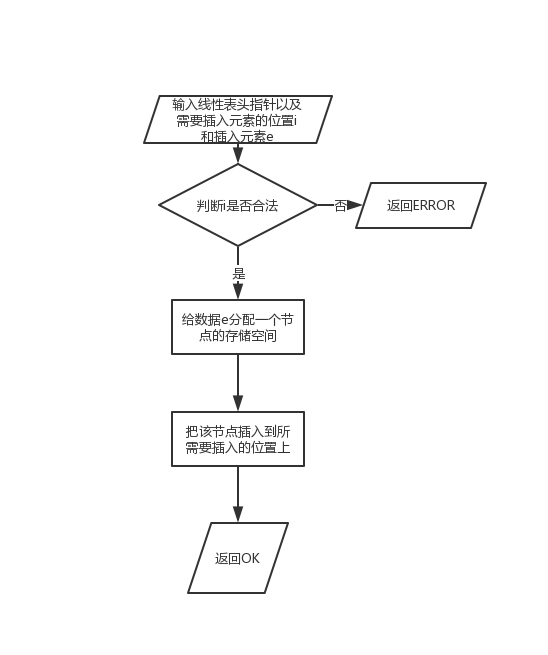
操作结果：若cur是L的数据元素，且不是最后一个，则用next\_e返回它的后继元素的地址，否则操作失败，next\_e无定义。

（10）**status ListInsert(ListNodePtr L, int i, ElemType e);**

初始条件：单链表已存在，1≤i≤L->length+1，输入的操作数为10。

算法设计：先判断输入的i值是否合法。接着给需要插入的数据分配一个节点的内存，再把该节点插入到所在的位置上。

操作结果：在L的第i个位置上插入新的数据元素e（i为length+1则表示在末尾插入），单链表的长度加1。（流程图见下图）

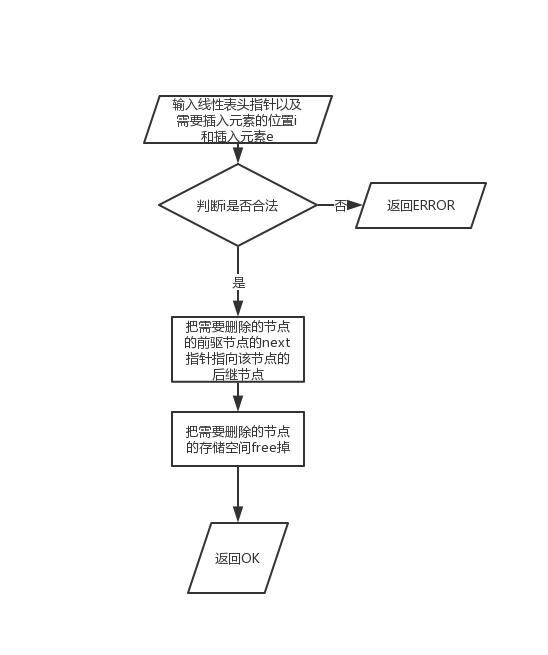


（11）**status ListDelete(ListNodePtr L, ElemType \*e, int i);**

初始条件：单链表已存在，1≤i≤List->Length,输入的操作数为11。

算法设计：先判断index是否在合理范围之内（1≤i≤Length），然后把需要删除的节点的前一个结点的next指针指向该节点的下一个节点。

操作结果：删除L的位置为i的数据元素，用e返回其值，同时L的长度减1。（流程图见下图）



（12）**status ListTrabverse(SqList L);**

初始条件：单链表L已存在，输入的操作数为12。

算法设计：遍历表L中的每个元素，并且打印出来。

操作结果：依次输出表L中的每个元素的值。

（13）**status SaveList(ListNodePtr L);**

初始条件：单链表已存在，输入的操作数为13。

算法设计：遍历以L为表头节点的每个顺序表，将表中信息和元素输入到文件并保存。

操作结果：保存当前的操作表。

（14）**status LoadList(ListNodePtr L);**

初始条件：单链表已存在，输入的操作数为14。

算法设计：读取文件中保存的顺序表的信息，把相应的数据存储到该操作表的数据域中。

操作结果：重新读取已保存的表，给操作表赋值。

（15）**status InputData(ListNodePtr L);**

初始条件：单链表已存在，输入的操作数为15。

算法设计：输入一串字符串，把其中的数字作为数据读入操作表中。

操作结果：通过直接手写输入数据，直接给操作表赋值。

## 2.3 系统实现

（1）**status ListInsert(ListNodePtr L, int i, ElemType e)**{

if(!L)

return INFEASTABLE;

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

return ERROR;

ListNode \*p=L,\*q=NULL;

while(i>1){ //把p指针移动到需要添加元素的位置

p=p->next;

i--;

}

q=(ListNodePtr)malloc(sizeof(ListNode));

q->elem=e;

q->next=p->next;

p->next=q;

L->elem++;

return OK;

}

时间和空间复杂度分析：假设表长为n，则找到第i个元素所需要的时间为i，所以平均需要的步骤为n/2，所以时间复杂度为O(n)。接下来进行的插入的操作，时间复杂度为O(2),故函数的时间复杂度为O(n),综上所述，该插入函数的时间复杂度为O(n)。

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

if(!L)

return INFEASTABLE;

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

return ERROR;

ListNode \*p=L;

while(i>1){ //把p指针移动到需要删除的位置

p=p->next;

i--;

}

\*e=p->next->elem;

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

L->elem--;

return OK;

}

时间和空间复杂度分析：假设表长为n，则找到第i个元素所需要的时间为i，所以平均需要的步骤为n/2，所以时间复杂度为O(n)。之后需要进行删除的操作，时间复杂度为O(2)故函数的时间复杂度为O(n),综上所述，该插入函数的时间复杂度为O(n)。

## 2.4 系统测试

测试中使用的测试数据：

表2-1 测试用例

|  |  |
| --- | --- |
| **表ID** | **表中元素（按顺序排列）** |
| **1** | **1，2，3，4，5，6，7，8** |
| **2** | **8，9，3，6，7，4** |
| **3** | **34，12，55，15，9** |

1) 测试函数：InitList

测试步骤及结果如表 2-2 所示

表2-2 InitList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **无线性表时建立表** | **在无表1时，输入命令1** | **建立ID为1的表。** |  |
| **有线性表时建立表** | **在已有表1时，输入命令1** | **重新建立ID为1的表** |  |

2) 测试函数：DestroyList

测试步骤及结果如表 2-3 所示

表2-3 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **有线性表时销毁表** | **在已经创建了id为1的表的基础上，输入命令2** | **销毁ID为1的表。** |  |
| **无线性表时销毁表** | **在没创建id为1的表的基础上，输入命令2** | **无任何操作** |  |

3) 测试函数：ClearList

测试步骤及结果如表 2-4 所示

表2-4 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时清空表** | **在id为1的表中有元素的前提下，输入命令3** | **清空ID为1的表** |  |
| **表存在，无元素时清空表** | **在id为1的表中没有元素的前提下，输入命令3** | **ID为1的表仍为空** |  |
| **表不存在时清空表** | **不存在id为0的表时，输入命令3** | **操作失败** |  |

4) 测试函数：ListEmpty

测试步骤及结果如表 2-5 所示

表2-5 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时判断表是否为空** | **在id为1的表中有元素的前提下，输入命令4** | **得出表1非空** |  |
| **表存在，无元素时判断表是否为空** | **在id为1的表中没有元素的前提下，输入命令4** | **得出表1为空** |  |
| **表不存在时判断表是否为空** | **不存在id为1的表时，输入命令4** | **操作失败** |  |

5) 测试函数：ListLength

测试步骤及结果如表 2-6 所示

表2-6 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在时获得表长** | **在有id为1的表的前提下，输入命令5** | **得出表1的长度为8** |  |
| **表不存在时判断表是否为空** | **不存在id为1的表时，输入命令5** | **操作失败** |  |

6) 测试函数：GetElem

测试步骤及结果如表 2-7 所示

表2-7 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有元素时获得表的首元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为1** | **得到表1的首元素** |  |
| **表存在，有元素时获得表的中间元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为3** | **得到表1的第3个元素** |  |
| **表存在，有元素时获得表的尾元素** | **在id为1的表中有元素的前提下，输入命令6，输入的元素位置为8** | **得到表1的尾元素** |  |
| **表存在，有元素时获得超出表范围的元素** | **在id为1的表中有元素的前提下，输入命令6，输入元素的位置为9** | **序号超出范围，获取元素失败** |  |
| **表不存在时获得表的元素** | **不存在id为0的表时，输入命令6，输入元素的位置为1** | **操作失败** |  |

7) 测试函数：LocateElem

测试步骤及结果如表 2-8 所示

表2-8 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，有对应元素时，查找相应的元素** | **在id为2的表中有元素7的前提下，输入命令7 7** | **成功在表2中找到元素7，其序号为5** |  |
| **表存在，没有对应元素时，查找相应的元素** | **在id为2的表中没有元素100的前提下，输入命令7 100** | **查找失败** |  |
| **表不存在时查找元素** | **不存在id为2的表时，输入命令7 8** | **操作失败** |  |

8) 测试函数：PriorElem

测试步骤及结果如表 2-9 所示

表2-9 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，元素有前驱时获取对应元素的前驱** | **在表2中元素9 有前驱的前提下，输入命令8 9** | **找到表2中元素9的前驱为8** |  |
| **表存在，元素无前驱时获取对应元素的前驱** | **在表2中元素8没有前驱的前提下，输入命令8 8** | **查找元素8的前驱失败** |  |
| **表不存在时，获取对应元素的前驱** | **不存在id为2的表时，输入命令8 6** | **操作失败** |  |

9) 测试函数：NextElem

测试步骤及结果如表 2-10 所示

表2-10 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，元素有后继时获取对应元素的前驱** | **在表3中元素34有后继的前提下，输入命令9 34** | **找到表3中元素34的后继为12** |  |
| **表存在，元素无后继时获取对应元素的前驱** | **在表3中元素9没有后继的前提下，输入命令9 9** | **查找元素9的后继失败** |  |
| **表不存在时，获取对应元素的后继** | **不存在id为3的表时，输入命令9 55** | **操作失败** |  |

10) 测试函数：ListInsert

测试步骤及结果如表 2-11 所示

表2-11 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，在表尾插入元素** | **在已构建好的表3中表头插入元素，输入命令10 6 18** | **表3中的序列由34→12→55→15→9**  **变为34→12→55→15→18→9** |  |
| **表存在，在表中间插入元素** | **在已构建好的表3中表头插入元素，输入命令10 3 18** | **表3中的序列由34→12→55→15→9**  **变为34→12→18→55→15→9** |  |
| **表存在，在表头插入元素** | **在已构建好的表3中表头插入元素，输入命令10 1 18** | **表3中的序列由34→12→55→15→9**  **变为18→34→12→55→15→9** |  |
| **表存在，在超出范围时插入元素** | **在已构建好的表3中表头插入元素，输入命令10 8 18** | **操作错误** |  |
| **表不存在时，插入元素** | **不存在id为3的表时，输入命令10 2 18** | **操作失败** |  |

11) 测试函数：ListDelete

测试步骤及结果如表 2-12 所示

表2-12 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，在表头删除元素** | **在已构建好的表3中表头插入元素，输入命令11 1** | **表3中的序列由34→12→55→15→9**  **变为12→55→15→9** |  |
| **表存在，在表中间删除元素** | **在已构建好的表3中表中间插入元素，输入命令11 3** | **表3中的序列由34→12→55→15→9**  **变为34→12→15→9** |  |
| **表存在，在表末尾删除元素** | **在已构建好的表3中表尾插入元素，输入命令11 5** | **表3中的序列由34→12→55→15→9**  **变为34→12→55→15** |  |
| **表存在，在超出范围时删除元素** | **在已构建好的表3中插入元素，输入命令11 8** | **操作失败** |  |
| **表不存在时，删除元素** | **不存在id为3的表时，输入命令11 1** | **操作失败** |  |

12) 测试函数：ListTraverse

测试步骤及结果如表 2-13 所示

表2-13 ListTraverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **表存在，**遍历**表中元素** | **在表1存在的时，输入命令12** | **遍历打印ID为1的表中元素** |  |
| **表不存在时，遍历表中元素** | **不存在表1时，输入命令12** | **操作失败** |  |

13) 测试函数：SaveList & LoadList

测试步骤及结果如表 2-14 所示

表2-14 保存/读取函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **测试保存功能** | **存在表1、表2、表3时，输入 14 文件名命名为yyx** | **将全部线性表保存到文件yyx** |  |
| **测试读取功能** | **退出程序，重新打开程序，输入13，读取的文件名为yyx，输入12** | **从文件中读入保存的线性表，读入后的表1的元素应该与之前完全相同** |  |

## 2.5 实验小结

这次试验，我完成了利用单链表实现表，这次试验基本思路与上一次实验相同，不过在存储结构上，单链表的空表头用来存储表的长度，算法实现起来更加简洁而高效。本人在单向链表的插入与删除的函数编写过程中体会到了指针操作的重要性。

最后谈一下自己对于这两种实现线性表的方法的思考。我认为用单链表编写的线性表对于空间的利用率更加高效，而顺序表的存储方式相对死板，不能实现有多少数据就申请多少存储空间，因此本人更加偏向于用单链表实现线性表。

最后，十分感谢老师和助教对我的帮助。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

本次实验需要通过运用二叉链表作为物理结构，建立二叉树。该二叉树需要有增删查改等功能。实现功能的过程中需要用到堆栈，队列等数据结构，在实现遍历，建立二叉树的过程中需要用到递归的知识，最终的输出要求有一个完整的菜单，每个功能需要有输入提示指令。

## 3.2 系统设计

二叉树节点定义：

typedef struct BiNode

{

ElemType elem;

struct BiNode \*pre;

struct BiNode \*left;

struct BiNode \*right;

int key;

} BiNode;

存储二叉树的结构体定义：

typedef struct Treehead

{

char name[100];

BiNode \*root;

int id; //表示该树的id值

struct Treehead \*next;

struct Treehead \*pre;

int nodes;

} Treehead;

二叉树存储实例如下图：

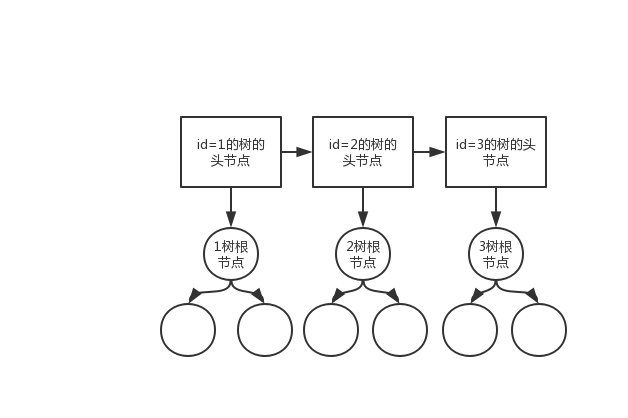


表3-1

函数定义：

（1）**BiNode \*CreateBiTree(BiNode \*T);**

初始条件：二叉树表头已经初始化，传入二叉树的表头的根节点指针。

算法设计：根据读取的字符，如果字符为“#”，则返回NULL指针，如果不是，则分配给该字符一个节点的存储空间，递归调用create函数，传入的节点为T节点的左孩子。

操作结果：通过先序遍历构建一个二叉树。

（2）**status DestroyBiTree(Treehead \*T);**

初始条件：二叉树存在，传入该二叉树的表头。

算法设计：先调用clear函数把表头结点所存储的二叉树清空，再把表头节点从双向链表中删除。

操作结果：销毁一个已经存在的二叉树，该树可以为空树。

（3）**status ClearBiTree(BiNode \*T);**

初始条件：二叉树存在，传入二叉树的根节点。

算法设计：利用递归遍历二叉树的每一个节点，把每一个节点的存储空间都释放，最后把T节点置为NULL值，让该树的头结点的root指针指向T。

操作结果：清空顺序表L中的全部元素，返回T节点。

（4）**status BiTreeEmpty(BiNode \*T);**

初始条件：二叉树存在，传入二叉树的根节点。

算法设计：通过判断二叉树的根节点是否为空。

操作结果：若L为空树，则返回TRUE,否则返回FALSE。

（5）**status BiTreeDepth(BiNode \*T);**

初始条件：二叉树存在，传入二叉树的根节点

算法设计：递归查找根节点的左右子树的高度，以最大高度加一作为返回值。

操作结果：返回二叉树的深度。

（6）**BiNode \*LocateNode(BiNode \*T, int e);**

初始条件：二叉树已存在，传入二叉树的根节点和需要定位的节点的键值e。

算法设计：递归查找根节点的左右子树，直到搜索到键值为e的节点，返回该节点，如果该节点为空则返回NULL指针。

操作结果：获取键值为e的节点。

（7）**status Assign(BiNode \*T, int e, ElemType value);**

初始条件：二叉树已存在，传入二叉树的跟节点，需要修改的节点的键值e和修改后的节点值value。

算法设计：调用Locate函数定位键值为e的节点，把该节点的值改为value。

操作结果：二叉树中键值为e的节点数据被改为value。

（8）**BiNode \*GetSibling(BiNode \*T, int e);**

初始条件：二叉树已存在，传入需要获取兄弟节点的键值。

算法设计：调用Locate函数定位键值为e的节点，再获取其兄弟节点。

操作结果：若键值为e的节点有兄弟节点，则返回兄弟节点，如果不存在，则返回NULL。

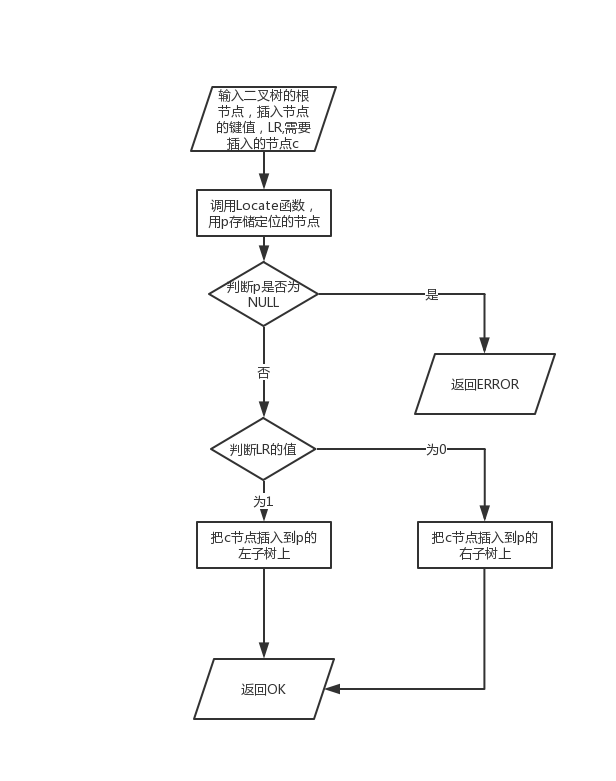
（9）**status InsertNode(BiNode \*T, int e, int LR, BiNode \*c);**

初始条件：二叉树已存在，传入需要插入节点的键值e和在该节点的什么部位插入LR，再传入需要插入的节点c。

算法设计：调用Locate函数定位键值为e的节点，如果LR为0，则在左树插入节点c，如果LR为1，则再右树插入节点c。

操作结果：节点c成功被插入到相应的位置。

函数流程图如下图：



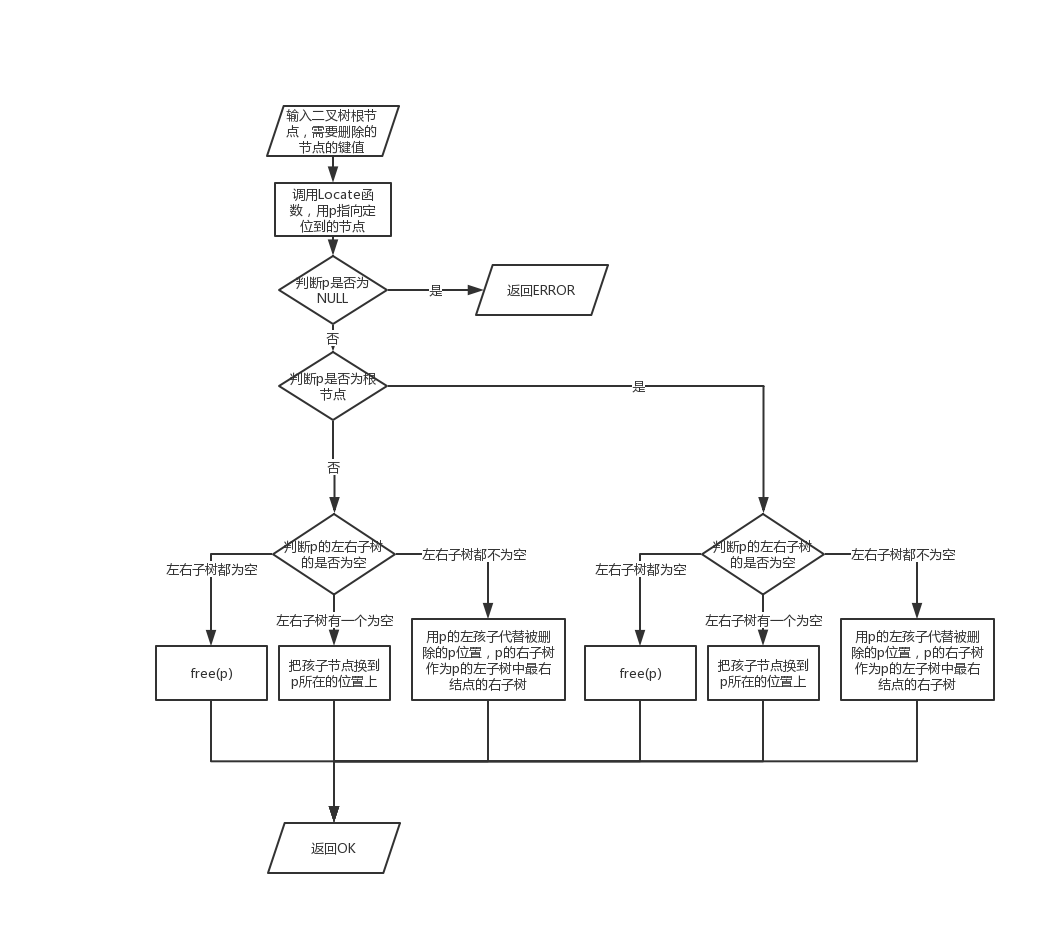
（10）**BiNode \*DeleteNode(BiNode \*T, int e);**

初始条件：二叉树以存在，传入二叉树的根节点和需要删除的节点的键值e。

算法设计：先调用Locate函数定位键值为e的节点，再对该节点的删除类型进行判断并进行删除操作。

操作结果：键值为e的节点被成功删除。

函数流程图如下图：



（11）**status PreOrderTraverse(BiNode \*T);**

初始条件：二叉树以存在，传入二叉树的根节点。

算法设计：先把根节点压入栈中，弹出根节点并把该节带你的左右子树同时压入栈中，重复以上操作直到栈中元素个数为0时返回OK。

操作结果：输出二叉树素的先序遍历。

（12）**status InOrderTraverse(BiNode \*T);**

初始条件：二叉树已存在，传入二叉树的根节点。

算法设计：先递归访问左子树，再输出根节点，最后递归访问右子树。

操作结果：输出二叉树的中序遍历。

（13）**status PostOrderTraverse(BiNode \*T);**

初始条件：二叉树已存在，传入二叉树的根节点。

算法设计：先递归访问左子树，再递归访问右子树，最后输出根节点。

操作结果：输出二叉树的后序遍历。

（14）**status LevelOrderTraverse(BiNode \*T);**

初始条件：二叉树已存在，传入二叉树的根节点。

算法设计：创建一个队列，先让根节点进队，然后每一个节点出队的时候，都让该节点的左右节点依次入队，知道队的长度为0时，返回OK。

操作结果：输出二叉树的层序遍历。

（15）**status SaveBiTree(Treehead \*T);**

初始条件：二叉树书已存在，传入二叉树的表头节点。

算法设计：按照先序遍历的方式依次访问二叉树的节点，把二叉树的先序遍历存储在二进制文件中。

（16）**BiNode \*LoadBiTree(BiNode \*T);**

初始条件：二叉树已存在，并且该二叉树为空。

算法设计：调用CreateBiTree2函数，把二进制文件中的前序遍历构建成一个二叉树。

（17）**BiNode \*CreateBiTree2(BiNode \*T, ElemType \*\*p);**

初始条件：二叉树已存在，传入二叉树的根节点和指向一个字符串数组的二级指针。

算法设计：通过移动二级指针实现递归建立二叉树。

## 3.3 系统实现

（1）**status InsertNode(BiNode \*T, int e, int LR, BiNode \*c)**{

//根据传入的键值，以及LR判断插入左树还是右树，c为需要插入的节点

BiNode \*p;

p = LocateNode(T, e);

if (p == NULL)

return ERROR;

if (LR == 0)

{

c->right = p->left;

if (c->right != NULL)

c->right->pre = c;

p->left = c;

c->pre = p;

}

else if (LR == 1)

{

c->right = p->right;

if (c->right != NULL)

c->right->pre = c;

p->right = c;

c->pre = p;

}

return OK;

}

时间和空间复杂度分析：假设二叉树的节点个数为n，则找到键值为e的节点所需要的时间为n，所以平均需要的步骤为n/2，所以时间复杂度为O(n)。接下来进行的插入的操作，把定位到的节点的原左子树或右子树作为插入节点的右子树综上所述，该插入函数的总时间复杂度为O(n)。空间上在插入一个节点后该子树的高度增加了1，总结点数加一。

（2）**BiNode \*DeleteNode(BiNode \*T, int e)**{

//根据传入的键值删除相应的节点

BiNode \*p;

p = LocateNode(T, e);

if (p == NULL)

return T;

if (p->pre == NULL)

{

//该节点为根节点，分为四种情况，第一种左右子树都为空，第二种左子树为空，第三种右子树为空，第四种左右子树都不为空

if (p->left == NULL && p->right == NULL)

{

free(p);

T = NULL;

}

if (p->right != NULL && p->left == NULL)

{

T = p->right;

T->pre = NULL;

free(p);

}

if (p->right == NULL && p->left != NULL)

{

T = p->left;

T->pre = NULL;

free(p);

}

else if (p->right != NULL && p->left != NULL)

{

T = p->left;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

q->right->pre = q;

T->pre = NULL;

free(p);

}

} //若删除的为根节点，该节点没有父亲节点，故需要单独分情况讨论

else

{ //删除的节点不为根节点

if (p->left == NULL && p->right == NULL)

{

if (p->pre->left == p)

p->pre->left = NULL;

else

p->pre->right = NULL;

free(p);

}

if (p->right != NULL && p->left == NULL)

{

if (p->pre->left == p)

p->pre->left = p->right;

else

p->pre->right = p->right;

free(p);

}

if (p->right == NULL && p->left != NULL)

{

if (p->pre->left == p)

p->pre->left = p->left;

else

p->pre->right = p->left;

free(p);

}

else if (p->right != NULL && p->left != NULL)

{

if (p->pre->left == p)

{

p->pre->left = p->left;

p->left->pre = p->pre;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

p->right->pre = q;

free(p);

}

if (p->pre->right == p)

{

p->pre->right = p->left;

p->left->pre = p->pre;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

p->right->pre = q;

free(p);

}

}

}

printf("该节点成功删除!\n");

return T;

}

时间和空间复杂度分析：假设二叉树节点数为n，则找到键值为e的节点所需要的时间为i，所以平均需要的步骤数为n/2，所以时间复杂度为O(n)。之后需要进行删除的操作，需要先搜索需要删除的节点的左子树的最右节点，然后把需要删除的节点的原右子树接在该节点后面，这个操作所需要的时间复杂度也为O(n),综上所述，该删除节点函数的时间复杂度为O(n)。

备注：该函数首先需要考虑被删除的节点是否为根节点，下一步需要考虑该节点的左右子树分别是否为空的情况，对于自己编程思路的严谨性要求很高。

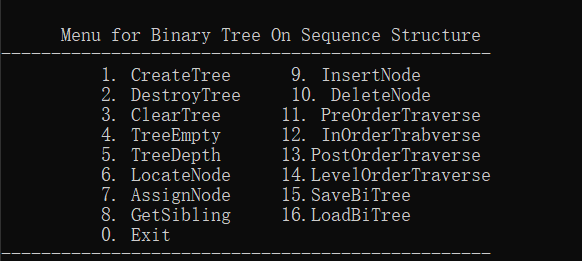
## 3.4 系统测试

测试中使用的测试数据：

表3-2 测试用例

|  |  |
| --- | --- |
| **树名** | **树定义序列** |
| **A** | **a b # c d # # # e # f # #** |
| **B** | **a b # # c # #** |
| **C** | **a b c d e f # # # # # # #** |

菜单实例：



1) 测试函数：**BiNode \*CreateBiTree(BiNode \*T);**

测试步骤及结果如表 3-3 所示

表3-3 CreateBiTree函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **无二叉树时建立树** | **在无二叉树的时候输入一串先序序列构建二叉树** | **建立树名为A的树，树A的id为1** |  |
| **在有二叉树时继续建立二叉树** | **在二叉树A已经建立的情况下，再输入一段先序序列建立二叉树B** | **建立树名为B的树，树B的id为2** |  |
| **在有二叉树时继续建立二叉树** | **在二叉树A，B已经建立的情况下，再输入一段先序序列建立二叉树C** | **建立树名为C的树，树C的id为3** |  |

2) 测试函数：**status DestroyBiTree(Treehead \*T);**

测试步骤及结果如表 3-4 所示

表3-4 DestroyBiTree函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **有二叉树时销毁树** | **在已有树A时，输入该树的id为1** | **销毁树名为A的树。** |  |
| **无二叉树时销毁树** | **在无树A时，输入id为4** | **无任何操作** |  |

3) 测试函数：**status ClearBiTree(BiNode \*T);**

测试步骤及结果如表 3-5 所示

表3-5 ClearBiTree函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树非空时清空树** | **在已有树A时，输入树A的id为1** | **清空A树。** |  |
| **无二叉树时清空树** | **在摧毁树A后，输入命令树A的id为1** | **无任何操作** |  |

4) 测试函数：**status BiTreeEmpty(BiNode \*T);**

测试步骤及结果如表 3-6 所示

表3-6 BiTreeEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树非空时判断树是否为空** | **在已有树A时，输入命令4** | **A树非空。** |  |
| **二叉树为空时判断树是否为空** | **在把树A时clear后，输入命令4** | **A树为空树。** |  |
| **无二叉树时判断树是否为空** | **在无树A时，输入命令4** | **操作失败** |  |

5) 测试函数：**status BiTreeDepth(BiNode \*T);**

测试步骤及结果如表 3-7 所示

表3-7 BiTreeDepth函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树非空时获取树深度** | **在已有树A时，输入命令5** | **获取A树的深度，应为4** |  |
| **二叉树为空时获取树深度** | **在清空树A后，输入命令5** | **获取B树的深度，应为0** |  |
| **无二叉树时获取树深度** | **在摧毁树A时，输入命令5** | **操作失败** |  |

6) 测试函数：**BiNode \*LocateNode(BiNode \*T, int e);**

测试步骤及结果如表 3-8 所示

表3-8 LocateNode函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树非空时获得根节点** | **在已有树A时，输入命令6,键值1** | **获得A树的根节点，应为a** |  |
| **二叉树为空时获得根节点** | **在已有空树A时，输入命令6** | **A树为空树，获取失败** |  |
| **无二叉树时获得根节点** | **在无树A时，输入命令6** | **操作失败** |  |

7) 测试函数：**status Assign(BiNode \*T, int e, ElemType value);**

测试步骤及结果如表 3-9 所示

表3-9 Assign函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树中某节点存在时赋值给该节点** | **在已有树A时，输入命令7，把value赋值为m** | **给A树节点1赋值m,先序遍历序列为mbcdef** |  |
| **二叉树中某节点不存在时赋值给该节点** | **在已有树A时，输入命令7，把value赋值为e** | **A树中不存在节点10，赋值失败** |  |
| **无二叉树时获得根节点** | **在无树A时，输入命令Assign A 1 50** | **操作失败** |  |

8) 测试函数：**BiNode \*GetSibling(BiNode \*T, int e);**

测试步骤及结果如表 3-10 所示

表3-10 GetSibling函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树中某节点存在兄弟节点时获得兄弟节点** | **在已有树A时，输入命令8 2** | **获得A树节点2的兄弟节点，应为e** |  |
| **二叉树中某节点不存在兄弟节点时获得兄弟节点** | **在已有树A时，输入命令8 3** | **A树中节点3没有兄弟节点，获取失败** |  |
| **二叉树中某节点不存在时获得兄弟节点** | **在已有树A时，输入命令8 10** | **获取兄弟节点失败** |  |
| **无二叉树时获得兄弟节点** | **在无树A时，输入命令8 2** | **操作失败** |  |

9) 测试函数：**status InsertNode(BiNode \*T, int e, int LR, BiNode \*c);**

测试步骤及结果如表 3-11 所示

表3-11 InsertNode函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **向二叉树中某节点插入子节点** | **在已有树A和非空树C时，输入命令9，键值为3，LR为0，值为h** | **在A树的2节点的左子树插入孩子c,先序遍历序列为abchdef** |  |
| **二叉树中某节点不存在时插入子树** | **在已有树A的情况下，在不存在的结点的左子树插入w** | **操作失败** |  |
| **子树不存在时插入子树** | **操作树A不存在** | **操作失败** |  |

10) 测试函数：**BiNode \*DeleteNode(BiNode \*T, int e);**

测试步骤及结果如表 3-12 所示

表3-12 DeleteNode函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **二叉树中某节点存在且要删除的节点两个子树存在时，删除该节点** | **在已有树A时，输入命令10 1** | **删除A树根节点1，先序遍历序列为bcdef** |  |
| **二叉树中某节点孩子节点不存在，删除该节点** | **在已有树A时，输入命令10 1 4** | **删除d节点，二叉树的先序遍历为abcef** |  |
| **无二叉树时删除该节点的子树** | **在无树A时，输入命令10 10** | **操作失败** |  |

11) 测试函数：**status PreOrderTraverse(BiNode \*T);**

**status InOrderTraverse(BiNode \*T);**

**status PostOrderTraverse(BiNode \*T);**

测试步骤及结果如表 3-13 所示

表3-13 Traverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **测试前序遍历** | **在已有树A时，输入命令11 1** | **前序遍历树A，应为abcdef** |  |
| **测试中序遍历** | **在已有树A时，输入命令12 1** | **中序遍历树A，应为bdcaef** |  |
| **测试后序遍历** | **在已有树A时，输入命令13 1** | **后序遍历树A，应为dcbfea** |  |
| **测试层序遍历** | **在已有树B时，输入命令14 1** | **层序遍历树B，应为abecfd** |  |

12) 测试函数：**SaveTree & LoadTree**

测试步骤及结果如表 3-14 所示

表3-14 保存/读取函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **测试保存功能** | **存在树A** | **将二叉树A保存到文件test** |  |
| **测试读取功能** | **先清空二叉树A，再从test文件中加载生成二叉树A** | **从文件中读入保存的二叉树，读入后的树A应为保存之前完全相同** |  |

所有功能函数完成测试！

## 3.5 实验小结

这次基于二叉链表的二叉树实验是我花的时间最长的一次实验，实验中编写的函数大量的使用了递归，队列，堆栈，链表等数据结构，让我对以上的数据结构加深了理解并能够灵活的运用。此外，本次实验我成功的把第一次实验中没有完成的利用动态链表实现多表管理成功的实现了一遍，把内存资源更好的利用了起来，不过实现起来也会相对复杂一些。

这次试验我也发现了很多不足之处，感觉自己在编写函数的时候由于之前没有考虑到一些测试用例，导致在之后修改的时候只能用if函数来处理特殊情况，最后把自己完全搞晕，只能重构函数，这反映出了我自己在设计编写函数的时候考虑不充分，可以自己先在演算纸上把所有可能的偏难怪的测试用例总结一遍之后再开始编写函数。在debug的过程中，我发现free掉一个指针之后如果不把该指针赋值为NULL，编译器就会报错，即使没有报错，在之后自己意想不到的地方就会发生问题，因此拥有一个良好的编程习惯也是一个很重要的事情。

# 4 基于邻接表的图实现

## 4.1 问题描述

本次实验需要通过运用邻接表作为物理结构，建立一个图。对于生成的图需要能够进行增删查改等功能实现。实现这些功能的时候会运用到以前的数据结构以及文件读取方面的知识，最终目标要求输出一个完整的菜单，可以对每一种命令进行操作。

## 4.2 系统设计

本次实验我实现的是有向图。

图的定义：

typedef struct Graph

{ //图定义

struct Vertex elem[MAXVERTEXNUM];

char name[100];

int vernum; //顶点数

int arcnum; //弧度数

int id; //表示该图的id值

struct Graph \*next;

struct Graph \*pre;

}Graph;

说明：elem为顶点数组，用来存储顶点的信息，name字符串用来存储图的名称，vernum用来存储该图的顶点数，arcnum用来存储弧度数，id用来存储图的id，next指针指向下一个图，pre指针指向前一个图。

图的顶点定义：

typedef struct Vertex

{ //顶点定义

char data;

int key;

struct ArcNode \*firstArc;

}Vertex;

说明：data用来存储每个顶点的数据，key用来存储每个顶点的键值，一个弧度指针指向该节点的第一邻接点。

图的弧的定义：

typedef struct ArcNode

{ //弧度定义

int rad;

struct ArcNode \*next;

}ArcNode;

说明：rad用来存储弧度所指向的下一个顶点的键值，next指针指向下一条从该顶点出发的弧。

图的邻接表存储实例如下图：

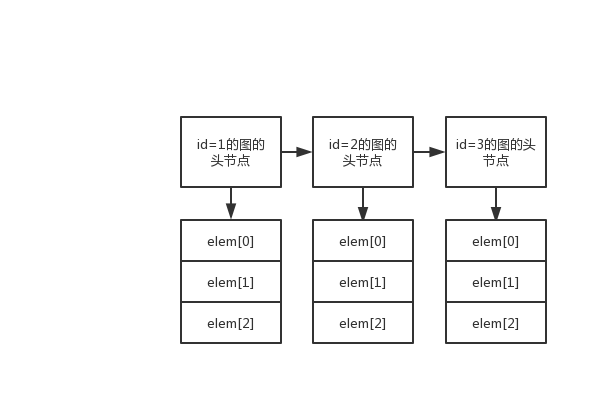


表4-1

函数定义：

（1）**status CreateGraph(char value[],int arcnode[],int vnum,int arcnum,Graph \*a);**

初始条件：图已经初始化，传入value字符串和int型数组arcnode。

算法设计：根据传入的value字符串依次给每个顶点赋值，每两个arcnode的值代表一条弧，依次读取每一条弧的信息，插入到对应的顶点的第一邻接点处。

操作结果：成功建立一个有向图。

（2）**status DestroyGraph(Graph \*a);**

初始条件：图存在，传入该图的结构体。

算法设计：把该图从图的单链表中删除，并且把该图的存储空间释放。

操作结果：销毁一个已经存在的有向图。

（3）**Vertex \*LocateVex(Graph \*a,int key);**

初始条件：图存在，传入图的结构体和需要定位的顶点的键值。

算法设计：利用循环访问图中的每一个节点，直到找到键值与key相同的顶点，返回该顶点，如果没有搜索到该顶点则返回NULL。

操作结果：返回需要定位的顶点的指针。

（4）**status PutVex (Graph \*a,int key,char value);**

初始条件：图存在，传入图的结构体和需要修改值的的顶点的键值，修改后的值为value传入。

算法设计：调用Locate函数先根据键值定位顶点，再把顶点的值修改为value。

操作结果：修改成功返回OK，修改失败返回ERROR。

（5）**Vertex \*FirstAdjVex(Graph \*a,int key);**

初始条件：图存在，传入图的结构体和需要寻找第一邻接点的顶点的键值。

算法设计：先调用Locate函数定位键值为key的顶点，返回该顶点的第一邻接点的所在顶点。

操作结果：返回第一邻接点。

（6）**Vertex \*NextAdjVex(Graph \*a,int v,int w);**

初始条件：图存在，传入图的结构体和两个顶点键值。

算法设计：调用Locate函数定位第一个顶点，再遍历该顶点的弧，找到键值为w的弧，返回该弧的下一条弧，即第二邻接点。

操作结果：获取键值为v的弧w的下一邻接点。

（7）**status InsertVex(Graph \*a,char v);**

初始条件：图存在，传入图的结构体和插入顶点的值。

算法设计：在原有的顶点后面加入一个新的节点，并且把v的值赋给该节点。

操作结果：插入了一个新的节点并且把该节点赋值。

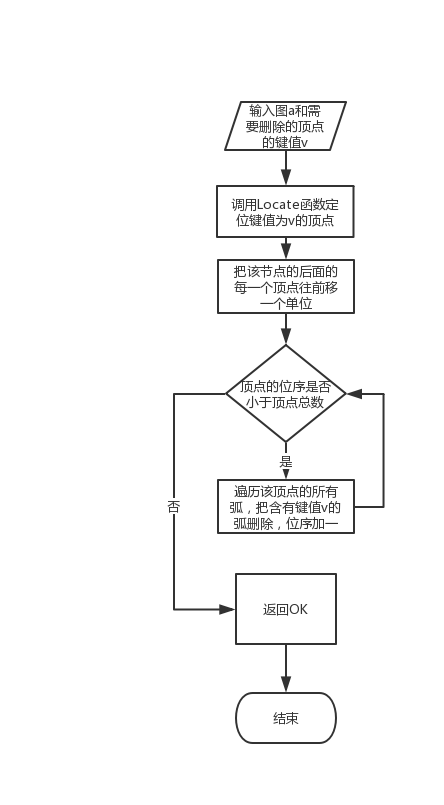
（8）**status DeleteVex(Graph \*a,int v);**

初始条件：图存在，传入图的结构体和需要删除的节点的键值。

算法设计：调用Locate函数定位键值为v的顶点，把该顶点以及其邻接点全部删除，遍历所有的弧，把所有含有v的弧删除。

操作结果：把顶点v所有信息从图中删除。

函数的操作流程图如下图：



Delete函数流程图

（9）**status InsertArc(Graph \*a,int v,int w);**

初始条件：图存在，传入图的结构体和需要插入弧的信息。

算法设计：调用Locate函数定位键值为v的顶点，在该顶点的第一邻接点后面插入rad值为w的弧。

操作结果：弧成功插入。

（10）**status DeleteArc(Graph \*a,int v,int w);**

初始条件：图存在，传入图的结构体和需要删除弧的信息。

算法设计：先调用Locate函数定位键值为v的顶点，在遍历该顶点的所有弧，找到rad值为w的弧，删除该弧。

操作结果：弧成功删除。

（11）**status DFSTraverse(Graph \*a);**

初始条件：图存在，传入图的结构体。

算法设计：利用visit函数来检验该节点是否访问，其余利用递归实现深度优先搜索输出每个节点的值，最后把没有连通的节点的值依次输出。

操作结果：输出图的深度优先遍历。

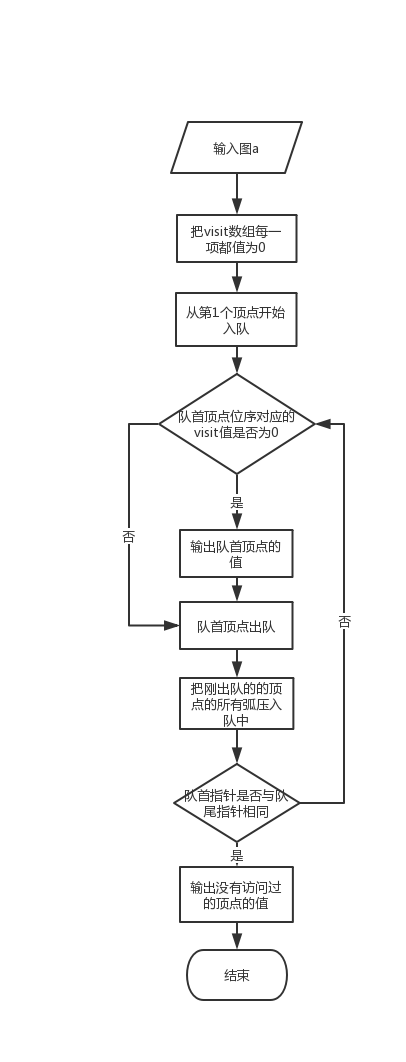
（12）**status BFSTraverse(Graph \*a);**

初始条件：图存在，传入图的结构体。

算法设计：利用队列先让头顶点入队，每弹出一个顶点，便把该顶点的所有没有访问过的弧的节点压入队中，直到队列长度为0，最后输出没有连通的顶点的值。

操作结果：输出图的广度优先遍历。

函数的流程图操作如下图：



BFS函数流程图

（13）**status SaveGraph(Graph \*a);**

初始条件：图已存在，传入图的结构体。

算法设计：先把顶点数和弧数写入文件，再依次按照位序存储每个顶点的键值，最后存储每个弧的信息，保存在自命名的二进制文件中。

操作结果：成功的把图的信息存储到二进制文件中。

（14）**status LoadGraph(Graph \*a);**

初始条件：图存在。

算法设计：把二进制中的数据读取出来，按照create函数的流程创建图。

操作结果：成功把二进制文件中存储的图加载出来。

## 4.3 系统实现

（1）**status DeleteVex(Graph \*a,int v)** {

int i;

for(i=1;i<=a->vernum;i++){

DeleteArc(a,v,i);

DeleteArc(a,i,v);

}

Vertex \*p=NULL;

p=LocateVex(a,v);

if(p==NULL)

return ERROR;

while(a->elem[a->vernum].key!=p->key){

p->data=(p+1)->data;

p->key=(p+1)->key;

p->firstArc=(p+1)->firstArc;

p=p+1;

}

a->vernum--;

return OK;

}

时间和空间复杂度分析：需要寻找到需要删除的顶点的时间复杂度为O(n),遍历访问每一条弧的时间复杂度为O(n^2),因此删除一个顶点所需要的总时间复杂度为O(n^2)。该算法调用了递归函数，但是删除弧的操作为遍历操作，故空间复杂度为O(n)。

（2）**status BFSTraverse(Graph \*a);**

{

int v=0,w;

for(v=0;v<a->vernum;v++)

visit[v]=0;

Vertex \*queue[20],\*pt=NULL,\*front=queue,\*back=queue;

for(v=0;v<a->vernum;v++){

if(!visit[v]){

\*back=a->elem[v];

back++;

while(front!=back){

ArcNode \*q=front->firstArc;

for(w=0;w<a->vernum;w++){

if(a->elem[w].key==front->key)

break;

}

if(!visit[w])

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

visit[w]=1;

front++;

while(q!=NULL){

\*back=\*LocateVex(a,q->rad);

back++;

q=q->next;

}

}

}

}

return OK;

}

时间和空间复杂度分析：先对visit数组进行赋值，时间复杂度为O(n),接下来依次访问每个顶点并且对每个顶点进行入队和出队的操作，时间复杂度为O(n),故总的时间复杂度为O(n)。空间复杂度与图的顶点数有关，故空间复杂度为O(n)。

备注：该函数的实现运用了队列数据结构，与二叉树的层序遍历操作比较相像。

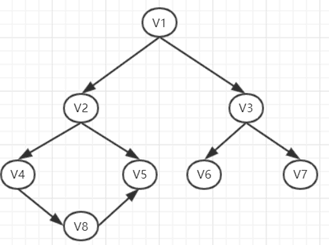
## 4.4 系统测试

菜单界面如下图：

菜单显示图



测试图实例：



测试用例图

1）测试函数：**CreateGraph;**

表4-2 CreateGraph函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图不存在的时候建立图** | **在无图的时候输入图的信息初始化图** | **建立图名为A的有向图** |  |

2）测试函数：**status DestroyGraph;**

表4-3 DestroyGraph函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图已经存在的时候销毁图** | **对已经存在的图执行销毁操作** | **图成功被销毁** |  |
| **在图不存在的时候销毁图** | **对不存在的图执行销毁操作** | **销毁图失败** |  |

3）测试函数：**Vertex \*LocateVex;**

表4-4 LocateVex函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图已经存在的时候根据键值寻找顶点** | **键值输入为4** | **返回需要定位的顶点的值** |  |
| **在图不存在的时候寻找顶点** | **访问图的id为2** | **定位顶点失败** |  |
| **在图存在的时候寻找不存在的顶点** | **键值输入19** | **定位顶点失败** |  |

4）测试函数：**status PutVex;**

表4-5 PutVex函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的情况下修改顶点的值** | **输入键值为5，修改后的值为Z** | **顶点的值被修改为Z** |  |
| **在图存在的时候修改不存在的顶点的值** | **输入的键值为100** | **修改失败** |  |
| **在图不存在的时候修改某个顶点的值** | **输入的图的id为3** | **修改失败** |  |

5）测试函数：**Vertex \*FirstAdjVex;**

表4-6 FirstAdjVex函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的时候，寻找顶点的第一邻接点** | **输入键值为1** | **输出的第一邻接点值为3** |  |
| **在图存在的时候，寻找的顶点没有第一邻接点** | **输入键值为5** | **没有第一邻接点** |  |
| **在图不存在的时候寻找第一邻接点** | **输入的图id为3** | **查找失败** |  |

6）测试函数：**Vertex \*NextAdjVex;**

表4-7 NextAdjVex函数测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | | **理论结果** | **运行结果** | | |
| **在图存在的时候，寻找一个顶点的下一邻接点** | **键值输入1，3** | | **下一邻接点的值为2** |  | | |
| **在图存在的时候寻找一个没有下一邻接点的顶点** | **键值输入1，2** | | **没有下一邻接点** |  | | |
| **在图不存在的时候寻找一个顶点的下一邻接点** | **图的id输入为3** | | **操作失败** |  | | |
| **在图不存在的时候寻找第一邻接点** | | **输入的图id为3** | | | **查找失败** |  |

7）测试函数：**status InsertVex;**

表4-8 InsertVex函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的时候插入一个顶点** | **插入的顶点的值为9** | **顶点成功插入** |  |
| **在图不存在的时候插入一个顶点** | **输入的图的id为4** | **顶点插入失败** |  |

8）测试函数：**status DeleteVex;**

表4-9 DeleteVex函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的时候删除一个顶点** | **删除的顶点的键值为5** | **2的第一邻接点变为4** |  |
| **在图存在的时候，删除一个不存在的顶点** | **删除顶点的键值为10** | **删除失败** |  |
| **在图不存在的时候删除一个顶点** | **输入的图的id为3** | **删除失败** |  |

9）测试函数：**status InsertArc;**

表4-10 InsertArc函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的时候插入一条弧** | **插入的弧为5 8** | **5的第一邻接点变为8** |  |
| **在图存在的时候插入一条顶点不存在的弧** | **插入的弧信息为10 1** | **插入弧失败** |  |
| **在图不存在的时候插入一条新的弧** | **图的id为4** | **插入弧失败** |  |

10）测试函数：**status DeleteArc;**

表4-11 DeleteArc函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **在图存在的时候删除一条弧** | **删除的弧信息为5 8** | **5的第一邻接点变为空** |  |
| **在图存在的时候删除一条不存在的弧** | **删除弧的信息为11 2** | **删除失败** |  |
| **在图不存在的时候删除一条弧** | **输入的图的id为5** | **删除失败** |  |

11）测试函数：**status DFSTraverse;**

表4-12 DFSTraverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **图存在的时候深度优先遍历整个图** | **输入图A** | **1 3 7 6 2 5 4 8** |  |
| **图不存在的时候深度优先遍历整个图** | **输入图A** | **输出失败** |  |

12）测试函数：**status BFSTraverse;**

表4-13 BFSTraverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **图存在的时候深度优先遍历整个图** | **输入图A** | **1 3 2 7 6 5 4 8** |  |
| **图不存在的时候深度优先遍历整个图** | **输入图A** | **输出失败** |  |

13）测试函数：**status SaveGraph;**

表4-14 SaveGraph函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **图存在的时候把图存储到二进制文件中** | **把id为1的图存储到yyx文件中** | **文件存储成功** |  |
| **图不存在** | **把id为4的图存储** | **文件存储失败** |  |

14）测试函数：**status LoadGraph;**

表4-15 LoadGraph函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试目的** | **测试条件&输入** | **理论结果** | **运行结果** |
| **图存在的时候加载文件中的图** | **把yyx文件中的图加载到id为2的图中** | **文件成功加载，BFS遍历结果为1 2 3 4 5 6 7 8** |  |
| **图不存在** | **加载图yyx** | **加载失败** |  |

所有的功能函数完成测试！

## 4.5 实验小结

本次实验是基于邻接表实现对图的操作。这次实验总体来说是对之前三个实验的一次综合实验，既用到了单链表的存储结构，同时也使用了十字链表的数据结构，当然我在这次实验中操作熟练度有了很大的提升，对于之前犯过的一些小的细节上的错误都有了很深的印象，对于自我来说是一次很好的总结与提升的机会。

这次试验，我选择了有向图，当然由于自己插入弧的方法和助教检查的样例先后顺序不一样，还是麻烦了助教细心的帮我检查了程序。印象最深的就是自己刚开始并没有考虑到删除了图的头节点形成了多个不连通图的情况，导致在遍历的过程中出现了错误，还是暴露了自己编写程序的严谨性不足。相信自己在这四次实验之后能够产生对编写程序的独到的经验和见解，从而在今后的编程道路上走得更远。

最后，由衷的感谢老师和助教在实验过程中帮助我理解实验报告上的要求以及对我在程序的编写思路和样例测试上所提供的帮助。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#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 InitList(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);

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

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

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

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

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

status ListTrabverse(SqList L);

status LoadList(SqList \*L);

status SaveList(SqList \*L);

status InputData(SqList \*L);

/\*功能函数声明\*/

int main(void)

{

SqList L[3] = {0};

int op = 1, renumber, flag, mail, id = 0;

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" 1. InitList 8. PriorElem\n");

printf(" 2. DestroyList 9. NextElem\n");

printf(" 3. ClearList 10. ListInsert\n");

printf(" 4. ListEmpty 11. ListDelete\n");

printf(" 5. ListLength 12. ListTrabverse\n");

printf(" 6. GetElem 13.LoadList\n");

printf(" 7. LocateElem 14.SaveList\n");

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

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

printf(" 请选择你的操作[0~15]:");

scanf("%d", &op);

switch (op)

{

case 1:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if (InitList(L + id - 1) == OK)

{

printf("线性表创建成功！\n");

}

else

printf("线性表创建失败！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 2:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = DestroyList(L + id - 1)) == OK)

{

printf("线性表删除成功！\n");

L->elem = NULL;

}

else if (renumber == INFEASTABLE)

printf("线性表不存在！\n");

else

printf("线性表删除失败！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 3:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = ClearList(L + id - 1)) == OK)

printf("线性表清除成功！\n");

else if (renumber == INFEASTABLE)

printf("线性表是空的！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 4:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = ListEmpty(L + id - 1)) == INFEASTABLE)

printf("线性表不存在！\n");

else if (renumber == TRUE)

printf("线性表是空的！\n");

else

printf("线性表不是空的！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 5:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = ListLength(L + id - 1)) != INFEASTABLE)

printf("线性表长度为:%d\n", renumber);

else

printf("线性表不存在！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 6:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入元素位置");

scanf("%d", &flag);

if ((renumber = GetElem(L + id - 1, flag, &mail)) == OK)

printf("位置%d的元素是%d\n", flag, mail);

else if (renumber == ERROR)

printf("该位置越界！\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 7:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要定位的元素:");

scanf("%d", &flag);

if ((renumber = LocateElem(L + id - 1, flag)) > 0)

printf("匹配成功，该元素位置为:%d\n", renumber);

else if (renumber == ERROR)

printf("匹配失败！\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 8:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要寻找前驱的元素:");

scanf("%d", &flag);

if ((renumber = PriorElem(L + id - 1, flag, &mail)) == OK)

printf("该数据的前驱元素为:%d\n", mail);

else if (renumber == ERROR)

printf("该元素没有前驱元素\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 9:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要寻找后继的元素:");

scanf("%d", &flag);

if ((renumber = NextElem(L + id - 1, flag, &mail)) == OK)

printf("该数据的后继元素为:%d\n", mail);

else if (renumber == ERROR)

printf("该数据没有后继元素\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 10:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入需要插入元素的位置:");

scanf("%d", &flag);

printf("请输入插入元素的值");

scanf("%d", &mail);

if ((renumber = ListInsert(L + id - 1, flag, mail)) == OK)

printf("插入成功!\n");

else if (renumber == OVERFLOW)

printf("存储分配失败\n");

else if (renumber == ERROR)

printf("插入元素的位置输入不合法\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 11:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入需要删除的元素的位置:");

scanf("%d", &flag);

if ((renumber = ListDelete(L + id - 1, &mail, flag)) == OK)

printf("该节点删除成功,删除的节点的值为:%d\n", mail);

else if (renumber == ERROR)

printf("删除元素的位置输入不合法\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 12:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if (ListTrabverse(L[id - 1]))

printf("线性表不存在！\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 13:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = LoadList(L + id - 1)) == OK)

printf("线性表成功赋值\n");

else if (renumber == 1)

printf("无法打开文件\n");

else

printf("线性表不存在\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 14:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = SaveList(L + id - 1)) == OK)

printf("文件成功保存\n");

else if (renumber == 1)

printf("无法打开文件\n");

else

printf("线性表不存在\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 15:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

{

printf("超出id范围\n");

getchar();

}

else if (InputData(L + id - 1) == OK)

printf("数据成功输入\n");

else

{

printf("线性表不存在\n");

getchar();

}

printf("请输入任意键继续\n");

getchar();

break;

case 0:

break;

} //end of switch

} //end of while

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

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

status InitList(SqList \*L) //初始化表

{

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

if (!L->elem) //如果L分配失败，返回overflow

exit(OVERFLOW);

L->length = 0;

L->listsize = LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \*L)

{

if (!L->elem)

return INFEASTABLE;

L->listsize = L->length = 0; //把表长和空间清0

free(L->elem); //释放存储空间

return OK;

}

status ClearList(SqList \*L)

{ //把表中的数据清除

if (!L->elem)

return INFEASTABLE;

L->length = 0;

return OK;

}

status ListEmpty(SqList \*L)

{ //检测线性表是否为空

if (!L->elem)

return INFEASTABLE;

if (L->length == 0)

{

return TRUE;

}

else

{

return FALSE;

}

}

int ListLength(SqList \*L)

{ //输出线性表长度

if (!L->elem)

return INFEASTABLE;

return L->length;

}

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

{

if (!L->elem)

return INFEASTABLE;

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

return ERROR; // 判断位置是否越界，越界则返回错误

\*e = \*(L->elem + i - 1); //e变量用来存储该元素的值

return OK;

}

int LocateElem(SqList \*L, ElemType e)

{ //e为需要定位的元素

if (!L->elem)

return INFEASTABLE;

int i;

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

{

if (L->elem[i - 1] == e) //如果匹配成功，则返回该元素的位置

return i;

}

return ERROR; //匹配失败，返回ERROR

}

status PriorElem(SqList \*L, ElemType cur, ElemType \*pre\_e)

{ //pre\_e用来存储前驱存储的元素

if (!L->elem)

return INFEASTABLE;

int index = LocateElem(L, cur) - 1; //定义位置变量

if (index <= 0)

return ERROR;

\*pre\_e = \*(L->elem + index - 1); //pre\_e指针指向前驱元素

return OK;

}

status NextElem(SqList \*L, ElemType cur, ElemType \*next\_e)

{ //next\_e用来存储后继的元素

if (!L->elem)

return INFEASTABLE;

int index = LocateElem(L, cur); //定义位置变量

if (index < 0 || index >= L->length)

return ERROR;

\*next\_e = \*(L->elem + index); //next\_e指针指向后继元素

return OK;

}

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

{

//在顺序线性表L中第i个位置之前插入新的元素e

//i的合法值为1<=i<=L->length+1

ElemType \*newbase;

ElemType \*p, \*q;

if (!L->elem)

return INFEASTABLE;

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

return ERROR; //i值不合法

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

{ //当前存储空间已满，增加分配

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

if (!newbase)

return OVERFLOW; //存储分配失败

L->elem = newbase; //新的基址

L->listsize += LISTINCREMENT; //增加存储容量

}

q = &(L->elem[i - 1]); //q为插入位置

for (p = &(L->elem[L->length]); p >= q; --p) //插入位置及之后的元素右移

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

\*q = e; //插入e

++L->length; //表长增1

return OK;

}

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

{ //删除表中的第i个元素，并用e返回其值

ElemType \*p, \*q;

if (!L->elem)

return INFEASTABLE;

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

return ERROR; //i值不合法

p = &(L->elem[i - 1]); //p为被删除元素的位置

\*e = \*p; //被删除元素的值赋给e

q = L->elem + L->length - 1; //表尾元素的位置

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

\*(p - 1) = \*p; //被删除元素之后的元素左移

--L->length; //表长减一

return OK;

}

status ListTrabverse(SqList L)

{

if (!L.elem)

return INFEASTABLE;

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;

}

status SaveList(SqList \*L)

{

if (!L->elem)

return INFEASTABLE;

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename); // 写文件

if ((fp = fopen(filename, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return 1;

}

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

fclose(fp);

return OK;

}

status LoadList(SqList \*L)

{

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

//读文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

L->length = 0;

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

{

printf("File open error\n ");

return 1;

}

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

L->length++;

fclose(fp);

return OK;

}

status InputData(SqList \*L)

{

if (!L->elem)

return INFEASTABLE;

char str[100];

int i = 0, count = 0, a[100] = {0}, symbol = 0;

printf("请输入一串元素\n");

getchar();

gets(str);

while (1)

{

while (str[i] && (str[i] < '0' || str[i] > '9') && str[i] != '-') //判断负数的情况

i++;

if (str[i])

{

if (str[i] == '-')

{

symbol = 1;

i++;

}

while (str[i] >= '0' && str[i] <= '9')

{

a[count] = 10 \* a[count] + str[i] - '0';

i++;

}

if (symbol == 1)

a[count] = -a[count];

symbol = 0;

count++;

}

else

break;

}

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

{ //接着之前的数据在之后继续加新的数据

L->elem[L->length + i] = a[i];

}

L->length += i;

return OK;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Linked List Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#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 ListNode

{ //链表元素结构体

ElemType elem;

struct ListNode \*next;

} ListNode,\*ListNodePtr; //ListNode为结构体，\*ListNodePtr为结构体元素指针

status InitList(ListNode \*\*L);

status DestroyList(ListNodePtr L);

status ClearList(ListNodePtr L);

status ListEmpty(ListNodePtr L);

int ListLength(ListNodePtr L);

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

int LocateElem(ListNodePtr L, ElemType e);

status PriorElem(ListNodePtr L, ElemType cur, ElemType \*pre\_e);

status NextElem(ListNodePtr L, ElemType cur, ElemType \*next\_e);

status ListInsert(ListNodePtr L, int i, ElemType e);

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

status ListTrabverse(ListNodePtr L);

status LoadList(ListNodePtr L);

status SaveList(ListNodePtr L);

status InputData(ListNodePtr L);

/\*功能函数声明\*/

int main(void)

{

ListNodePtr L[3];

int i;

for(i=0;i<3;i++)

L[i]=NULL;

int op = 1, renumber, flag, mail, id = 1;

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" 1. InitList 8. PriorElem\n");

printf(" 2. DestroyList 9. NextElem\n");

printf(" 3. ClearList 10. ListInsert\n");

printf(" 4. ListEmpty 11. ListDelete\n");

printf(" 5. ListLength 12. ListTrabverse\n");

printf(" 6. GetElem 13.LoadList\n");

printf(" 7. LocateElem 14.SaveList\n");

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

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

printf(" 请选择你的操作[0~15]:");

scanf("%d", &op);

switch (op)

{

case 1:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if(id<0||id>3)

printf("超出id范围\n");

else{

if (InitList(L+id-1) == OK)

{

printf("线性表创建成功！\n");

}

else

printf("线性表创建失败！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 2:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id < 0||id>3)

printf("超出id范围\n");

else{

if ((renumber = DestroyList(L[id-1])) == OK)

{

printf("线性表删除成功！\n");

L[id-1] = NULL;

}

else if (renumber == INFEASTABLE)

printf("线性表不存在！\n");

else

printf("线性表删除失败！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 3:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else{

if ((renumber = ClearList(L [id - 1])) == OK)

printf("线性表清除成功！\n");

else if (renumber == INFEASTABLE)

printf("线性！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 4:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else{

if ((renumber = ListEmpty(L [ id - 1])) == INFEASTABLE)

printf("线性表不存在！\n");

else if (renumber == TRUE)

printf("线性表是空的！\n");

else

printf("线性表不是空的！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 5:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else{

if ((renumber = ListLength(L [id - 1])) != INFEASTABLE)

printf("线性表长度为:%d\n", renumber);

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 6:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入元素位置");

scanf("%d", &flag);

if ((renumber = GetElem(L [ id - 1], flag, &mail)) == OK)

printf("位置%d的元素是%d\n", flag, mail);

else if (renumber == ERROR)

printf("该位置越界！\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 7:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要定位的元素:");

scanf("%d", &flag);

if ((renumber = LocateElem(L [id - 1], flag)) > 0)

printf("匹配成功，该元素位置为:%d\n", renumber);

else if (renumber == ERROR)

printf("匹配失败！\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 8:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要寻找前驱的元素:");

scanf("%d", &flag);

if ((renumber = PriorElem(L [id - 1], flag, &mail)) == OK)

printf("该数据的前驱元素为:%d\n", mail);

else if (renumber == ERROR)

printf("该元素没有前驱元素\n");

else

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 9:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("输入需要寻找后继的元素:");

scanf("%d", &flag);

if ((renumber = NextElem(L [id - 1], flag, &mail)) == OK)

printf("该数据的后继元素为:%d\n", mail);

else if (renumber == ERROR)

printf("该数据没有后继元素\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 10:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入需要插入元素的位置:");

scanf("%d", &flag);

printf("请输入插入元素的值");

scanf("%d", &mail);

if ((renumber = ListInsert(L [id - 1], flag, mail)) == OK)

printf("插入成功!\n");

else if (renumber == OVERFLOW)

printf("存储分配失败\n");

else if (renumber == ERROR)

printf("插入元素的位置输入不合法\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 11:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else

{

printf("请输入需要删除的元素的位置:");

scanf("%d", &flag);

if ((renumber = ListDelete(L [id - 1], &mail, flag)) == OK)

printf("该节点删除成功,删除的节点的值为:%d\n", mail);

else if (renumber == ERROR)

printf("删除元素的位置输入不合法\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 12:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else{

if (ListTrabverse(L[id - 1])==INFEASTABLE)

printf("线性表不存在！\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 13:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else if ((renumber = LoadList(L[id - 1])) == OK)

printf("线性表成功赋值\n");

else if (renumber == 1)

printf("无法打开文件\n");

else

printf("线性表不存在\n");

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 14:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

printf("超出id范围\n");

else{

if ((renumber = SaveList(L [id - 1])) == OK)

printf("文件成功保存\n");

else if (renumber == 1)

printf("无法打开文件\n");

else

printf("线性表不存在\n");

}

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 15:

printf("请输入表的id值[1~3]:");

scanf("%d", &id);

if (id > 3 || id < 0)

{

printf("超出id范围\n");

getchar();

}

else if (InputData(L [id - 1]) == OK)

printf("数据成功输入\n");

else

{

printf("线性表不存在\n");

getchar();

}

printf("请输入任意键继续\n");

getchar();

break;

case 0:

break;

} //end of switch

} //end of while

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

status InitList(ListNode \*\*L){

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

if(!L)

return OVERFLOW;

(\*L)->elem=0; //把表长清0

(\*L)->next=NULL; //把指针域赋值为NULL

return OK;

}

status DestroyList(ListNodePtr L){

if(!L)

return INFEASTABLE;

ListNodePtr p=L;

while(!p){

p=p->next; //两个指针，一个用来判断是否到了表的尾部，另外一个用来释放存储空间

free(L);

L=p;

}

return OK;

}

status ClearList(ListNodePtr L){

if(!L)

return INFEASTABLE;

DestroyList(L->next); //调用销毁表的函数，把除了表头以外的节点全部摧毁掉

L->next=NULL;

L->elem=0;

return OK;

}

status ListEmpty(ListNodePtr L){

if(!L)

return INFEASTABLE;

if(L->elem==0)

return TRUE;

else

return FALSE;

}

int ListLength(ListNodePtr L){

if(!L)

return INFEASTABLE;

return L->elem; //把表长赋值给e元素的数据域

}

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

if(!L)

return INFEASTABLE;

ListNode \*p=L;

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

return ERROR;

while(i>0){ //遍历整个线性表，直到找到所输入的位置

p=p->next;

i--;

}

\*e=p->elem; //把该位置的元素赋值给e的数据域

return OK;

}

int LocateElem(ListNodePtr L, ElemType e){

if(!L)

return INFEASTABLE;

int count=0;

ListNode \*p=L;

while(p){

if(p->elem==e) //判断该节点存储的元素是否为需要寻找的元素

return count; //该元素的的位置

p=p->next;

count++;

}

return ERROR;

}

status PriorElem(ListNodePtr L, ElemType cur, ElemType \*pre\_e){

if(!L)

return INFEASTABLE;

ListNode \*p=L->next;

while(p->next!=NULL){

if(p->next->elem==cur){ //如果该节点的后继节点的存储值和cur变量相同，则把该节点的存储值赋值给pre\_e

\*pre\_e=p->elem;

return OK;

}

p=p->next;

}

return ERROR;

}

status NextElem(ListNodePtr L, ElemType cur, ElemType \*next\_e){

if(!L)

return INFEASTABLE;

ListNode \*p=L->next;

while(p&&p->next){ //注意要排除p不是尾节点

if(cur==p->elem){

\*next\_e=p->next->elem;

return OK;

}

p=p->next;

}

return ERROR;

}

status ListInsert(ListNodePtr L, int i, ElemType e){

if(!L)

return INFEASTABLE;

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

return ERROR;

ListNode \*p=L,\*q=NULL;

while(i>1){ //把p指针移动到需要添加元素的位置

p=p->next;

i--;

}

q=(ListNodePtr)malloc(sizeof(ListNode));

q->elem=e;

q->next=p->next;

p->next=q;

L->elem++;

return OK;

}

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

if(!L)

return INFEASTABLE;

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

return ERROR;

ListNode \*p=L;

while(i>1){ //把p指针移动到需要删除的位置

p=p->next;

i--;

}

\*e=p->next->elem;

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

L->elem--;

return OK;

}

status ListTrabverse(ListNodePtr L){

if (!L)

return INFEASTABLE;

ListNode \*p=L->next;

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

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

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

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

return OK;

}

status LoadList(ListNodePtr L)

{

if (!L) // 线性表未创建

return INFEASTABLE;

//读文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

L->elem = 0;

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

{

printf("File open error\n ");

return 1;

}

ListNode \*p=L,\*q=NULL;

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

while (fread(&(q->elem), sizeof(ElemType), 1, fp)){

p->next=q;

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

p=p->next;

p->next=NULL;

L->elem++;

}

fclose(fp);

return OK;

}

status SaveList(ListNodePtr L)

{

if (!L)

return INFEASTABLE;

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename); // 写文件

if ((fp = fopen(filename, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return 1;

}

int a[100],i;

ListNode \*p=L;

for(i=0;i<L->elem&&p->next!=NULL;i++){ //把链表中的元素存储到数组中

p=p->next;

a[i]=p->elem;

}

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

fclose(fp);

return OK;

}

status InputData(ListNodePtr L){

if(!L)

return INFEASTABLE;

ListNode \*p=L;

while(p->next!=NULL)

p=p->next;

char str[100];

int i = 0, count = 0, a[100] = {0}, symbol = 0;

printf("请输入一串元素\n");

getchar();

gets(str);

while (1)

{

while (str[i] && (str[i] < '0' || str[i] > '9') && str[i] != '-') //判断负数的情况

i++;

if (str[i])

{

if (str[i] == '-')

{

symbol = 1;

i++;

}

while (str[i] >= '0' && str[i] <= '9')

{

a[count] = 10 \* a[count] + str[i] - '0';

i++;

}

if (symbol == 1)

a[count] = -a[count];

symbol = 0;

count++;

}

else

break;

}

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

{ //接着之前的数据在之后继续加新的数据

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

p=p->next;

p->elem=a[i];

p->next=NULL;

}

L->elem += i;

return OK;

}附录C 基于二叉链表二叉树实现的源程序

/\* Binary Tree On sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include<math.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef char ElemType; //数据元素类型定义

int key\_value = 0;

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct BiNode

{

ElemType elem;

struct BiNode \*pre;

struct BiNode \*left;

struct BiNode \*right;

int key;

} BiNode; //BiNode为二叉树结构体

typedef struct Treehead

{

char name[100];

BiNode \*root;

int id; //表示该树的id值

struct Treehead \*next;

struct Treehead \*pre;

int nodes;

} Treehead; //Treehead为二叉树的头部，用来存储树的名字以及树的节点个数

BiNode \*CreateBiTree(BiNode \*T); //创建二叉树

status DestroyBiTree(Treehead \*T); //销毁二叉树

status ClearBiTree(BiNode \*T); //清空二叉树

status BiTreeEmpty(BiNode \*T); //检查二叉树是否为空

status BiTreeDepth(BiNode \*T); //检查树的深度

BiNode \*LocateNode(BiNode \*T, int e); //根据键值定位节点

status Assign(BiNode \*T, int e, ElemType value); //修改节点的值

BiNode \*GetSibling(BiNode \*T, int e); //获得兄弟节点

status InsertNode(BiNode \*T, int e, int LR, BiNode \*c); //插入节点

BiNode \*DeleteNode(BiNode \*T, int e); //删除节点

status PreOrderTraverse(BiNode \*T); //先序遍历

status InOrderTraverse(BiNode \*T); //中序遍历

status PostOrderTraverse(BiNode \*T); //后序遍历

status LevelOrderTraverse(BiNode \*T); //层序遍历

status SaveBiTree(Treehead \*T); //保存二叉树

BiNode \*LoadBiTree(BiNode \*T); //加载二叉树

BiNode \*CreateBiTree2(BiNode \*T, ElemType \*\*p);

BiNode visualprint(BiNode \*T);

/\*功能函数声明\*/

int main(void)

{

Treehead \*head = NULL, \*tail = NULL, \*p = NULL, \*q = NULL;

int op = 1, id, renumber, key, LR;

ElemType value;

while (op)

{

system("cls");

printf("\n\n");

printf(" Menu for Binary Tree On Sequence Structure \n");

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

printf(" 1. CreateTree 9. InsertNode\n");

printf(" 2. DestroyTree 10. DeleteNode\n");

printf(" 3. ClearTree 11. PreOrderTraverse\n");

printf(" 4. TreeEmpty 12. InOrderTrabverse\n");

printf(" 5. TreeDepth 13.PostOrderTraverse\n");

printf(" 6. LocateNode 14.LevelOrderTraverse\n");

printf(" 7. AssignNode 15.SaveBiTree\n");

printf(" 8. GetSibling 16.LoadBiTree\n");

printf(" 0. Exit \n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d", &op);

switch (op)

{

case 1:

printf("请输入二叉树的前序序列:\n");

if (head == NULL)

{

head = (Treehead \*)malloc(sizeof(Treehead));

tail = head;

head->id = 1;

p = head;

head->pre = NULL;

head->next = NULL;

}

else if (head != NULL)

{

tail = (Treehead \*)malloc(sizeof(Treehead));

p->next = tail;

tail->pre = p;

tail->id = p->id + 1;

p = tail;

tail->next = NULL;

}

tail->root = NULL;

tail->nodes = 0;

q = head;

if ((tail->root = CreateBiTree(tail->root)) != NULL)

{

tail->root->pre = NULL;

printf("请输入二叉树的名称:\n");

fflush(stdin);

gets(p->name);

printf("二叉树创建成功！\n");

}

else

printf("二叉树创建失败！\n");

q->nodes = key\_value;

key\_value = 0;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 2:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

if (q == head)

head = head->next;

if (q == tail)

tail = tail->pre;

DestroyBiTree(q);

printf("二叉树删除成功！\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 3:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

ClearBiTree(q->root);

q->root = NULL;

printf("二叉树清空成功!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 4:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

if (BiTreeEmpty(q->root) == OK)

printf("二叉树%s是空树!\n", q->name);

else

printf("二叉树%s不是空树!\n", q->name);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 5:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("二叉树的深度为:%d\n", BiTreeDepth(q->root));

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 6:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("请输入需要查找节点的键值:\n");

scanf("%d", &key);

BiNode \*node;

node = LocateNode(q->root, key);

if (node != NULL)

printf("该节点的值为:%c\n", node->elem);

else

printf("该节点不存在!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 7:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("请输入需要修改的节点的键值:\n");

scanf("%d", &key);

printf("输入修改后的节点值:\n");

fflush(stdin);

scanf("%c", &value);

if (Assign(q->root, key, value) == OK)

printf("节点值成功修改!\n");

else

printf("节点值修改失败!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 8:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("请输入需要获取兄弟结点的键值:\n");

scanf("%d", &key);

BiNode \*bro;

bro = GetSibling(q->root, key);

if (bro == NULL)

printf("该节点的兄弟节点不存在!\n");

else

printf("该节点的兄弟节点的节点值为:%c\n", bro->elem);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 9:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("请输入需要插入的的节点的键值:\n");

scanf("%d", &key);

printf("请输入需要插入的子树，0代表左树，1代表右树\n");

scanf("%d", &LR);

printf("请输入需要插入节点的值:\n");

fflush(stdin);

scanf("%c", &value);

BiNode \*c = NULL;

c = (BiNode \*)malloc(sizeof(BiNode));

c->elem = value;

c->left = NULL;

c->right = NULL;

c->pre = NULL;

c->key = ++q->nodes;

if (InsertNode(q->root, key, LR, c) == OK)

printf("节点成功插入!\n");

else

{

printf("节点插入失败!\n");

q->nodes--;

}

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 10:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

printf("请输入需要删除结点的键值:\n");

scanf("%d", &key);

q->root = DeleteNode(q->root, key);

q->nodes--;

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 11:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

PreOrderTraverse(q->root);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 12:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

InOrderTraverse(q->root);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 13:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

PostOrderTraverse(q->root);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 14:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

LevelOrderTraverse(q->root);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 15:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

SaveBiTree(q);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 16:

printf("请输入二叉树的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("二叉树不存在！\n");

else

{

q->root = LoadBiTree(q->root);

}

q = head;

printf("请输入任意键继续\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");

return 0;

} //end of main()

BiNode \*CreateBiTree(BiNode \*T)

{

char ch;

BiNode \*trans = NULL; //用来存储函数的返回值的指针

fflush(stdin); //清空缓存区

scanf("%c", &ch);

if (ch != '#')

{

if (!(T = (BiNode \*)malloc(sizeof(BiNode))))

return NULL;

T->elem = ch;

T->key = ++key\_value;

trans = CreateBiTree(T->left);

T->left = trans;

if (trans != NULL)

trans->pre = T;

trans = CreateBiTree(T->right);

T->right = trans;

if (trans != NULL)

trans->pre = T;

}

else

{

T = NULL; //若输入为‘#’，则返回NULL指针

}

return T;

}

status DestroyBiTree(Treehead \*T)

{

ClearBiTree(T->root); //调用清空函数把二叉树清空

if (T->pre != NULL) //删除存储树的表头

T->pre->next = T->next;

if (T->next != NULL)

T->next->pre = T->pre;

free(T);

T = NULL;

return OK;

}

status ClearBiTree(BiNode \*T)

{

if (T == NULL)

return INFEASTABLE;

else

{

ClearBiTree(T->left);

ClearBiTree(T->right);

free(T);

T = NULL;

}

return OK;

}

status BiTreeEmpty(BiNode \*T) //检查二叉树的头节点来判断是否为空树

{

if (!T)

return OK;

else

return ERROR;

}

status BiTreeDepth(BiNode \*T)

{

int HL = 0, HR = 0, MAXH = 0;

if (T != NULL)

{ //递归查找左右子树的深度，选取深度较大的子节点的深度加一作为返回值

HL = BiTreeDepth(T->left);

HR = BiTreeDepth(T->right);

MAXH = (HL > HR) ? HL : HR;

return (MAXH + 1);

}

return MAXH;

}

BiNode \*LocateNode(BiNode \*T, int e)

{

if (T == NULL)

return NULL;

else

{

if (T->key == e)

return T;

else

{ //递归查找左右子树

BiNode \*pl = NULL, \*pr = NULL, \*p;

pl = LocateNode(T->left, e);

pr = LocateNode(T->right, e);

p = pl ? pl : pr;

return p;

}

}

}

status Assign(BiNode \*T, int e, ElemType value)

{

BiNode \*p;

p = LocateNode(T, e); //根据键值定位需要修改的节点

if (p == NULL)

return ERROR;

p->elem = value;

return OK;

}

BiNode \*GetSibling(BiNode \*T, int e)

{

BiNode \*p;

p = LocateNode(T, e);

if (p == NULL || p->pre == NULL) //判断该节点是否在树中

return NULL;

if (p->pre->left == p)

return p->pre->right;

else if (p->pre->right == p)

return p->pre->left;

}

status InsertNode(BiNode \*T, int e, int LR, BiNode \*c)

{ //根据传入的键值，以及LR判断插入左树还是右树，c为需要插入的节点

BiNode \*p;

p = LocateNode(T, e);

if (p == NULL)

return ERROR;

if (LR == 0)

{

c->right = p->left;

if (c->right != NULL)

c->right->pre = c;

p->left = c;

c->pre = p;

}

else if (LR == 1)

{

c->right = p->right;

if (c->right != NULL)

c->right->pre = c;

p->right = c;

c->pre = p;

}

return OK;

}

BiNode \*DeleteNode(BiNode \*T, int e)

{ //根据传入的键值删除相应的节点

BiNode \*p;

p = LocateNode(T, e);

if (p == NULL)

return T;

if (p->pre == NULL)

{ //该节点为根节点，分为四种情况，第一种左右子树都为空，第二种左子树为空，第三种右子树为空，第四种左右子树都不为空

if (p->left == NULL && p->right == NULL)

{

free(p);

T = NULL;

}

if (p->right != NULL && p->left == NULL)

{

T = p->right;

T->pre = NULL;

free(p);

}

if (p->right == NULL && p->left != NULL)

{

T = p->left;

T->pre = NULL;

free(p);

}

else if (p->right != NULL && p->left != NULL)

{

T = p->left;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

q->right->pre = q;

T->pre = NULL;

free(p);

}

} //若删除的为根节点，该节点没有父亲节点，故需要单独分情况讨论

else

{ //删除的节点不为根节点

if (p->left == NULL && p->right == NULL)

{

if (p->pre->left == p)

p->pre->left = NULL;

else

p->pre->right = NULL;

free(p);

}

if (p->right != NULL && p->left == NULL)

{

if (p->pre->left == p)

p->pre->left = p->right;

else

p->pre->right = p->right;

free(p);

}

if (p->right == NULL && p->left != NULL)

{

if (p->pre->left == p)

p->pre->left = p->left;

else

p->pre->right = p->left;

free(p);

}

else if (p->right != NULL && p->left != NULL)

{

if (p->pre->left == p)

{

p->pre->left = p->left;

p->left->pre = p->pre;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

p->right->pre = q;

free(p);

}

if (p->pre->right == p)

{

p->pre->right = p->left;

p->left->pre = p->pre;

BiNode \*q = p->left;

while (q->right != NULL)

q = q->right;

q->right = p->right;

p->right->pre = q;

free(p);

}

}

}

printf("该节点成功删除!\n");

return T;

}

status PreOrderTraverse(BiNode \*T)

{ //利用堆栈实现

BiNode \*ptr[100];

int top = -1;

BiNode \*pt = T;

ptr[++top] = pt;

while (top != -1)

{

pt = ptr[top--]; //每次栈顶元素出栈

printf("%c ", pt->elem); //输出

if (pt->right)

ptr[++top] = pt->right;

if (pt->left)

ptr[++top] = pt->left;

}

return OK;

}

status InOrderTraverse(BiNode \*T)

{ //利用递归实现

if (T != NULL)

{

InOrderTraverse(T->left);

printf("%c ", T->elem);

InOrderTraverse(T->right);

}

return OK;

}

status PostOrderTraverse(BiNode \*T)

{ //利用递归实现

if (T != NULL)

{

PostOrderTraverse(T->left);

PostOrderTraverse(T->right);

printf("%c ", T->elem);

}

return OK;

}

status LevelOrderTraverse(BiNode \*T)

{ //利用队列实现层序遍历

BiNode queue[100], \*front = queue, \*back = queue;

if (T != NULL)

{

\*back = \*T;

back++;

while (front != back)

{

if (front->left != NULL)

{

\*back = \*front->left;

back++;

}

if (front->right != NULL)

{

\*back = \*front->right;

back++;

}

printf("%c ", front->elem);

front++;

}

}

return OK;

}

status SaveBiTree(Treehead \*T)

{

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename); // 写文件

if ((fp = fopen(filename, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return 1;

}

ElemType a;

BiNode \*ptr[100];

int top = -1;

BiNode \*pt = T->root;

ptr[++top] = pt;

while (top != -1)

{

pt = ptr[top--]; //每次栈顶元素出栈

if (pt)

a = pt->elem;

else

a = '#';

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

if (pt->right)

ptr[++top] = pt->right;

else if (pt->elem != '#' && pt->right == NULL)

{

ptr[++top] = (BiNode \*)malloc(sizeof(BiNode));

ptr[top]->elem = '#';

ptr[top]->left = NULL;

ptr[top]->right = NULL;

}

if (pt->left)

ptr[++top] = pt->left;

else if (pt->elem != '#' && pt->left == NULL)

{

ptr[++top] = (BiNode \*)malloc(sizeof(BiNode));

ptr[top]->elem = '#';

ptr[top]->left = NULL;

ptr[top]->right = NULL;

}

}

fclose(fp);

printf("文件成功保存!\n");

return OK;

}

BiNode \*LoadBiTree(BiNode \*T)

{

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

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

{

printf("File open error\n ");

return NULL;

}

BiNode \*p = T;

ElemType a[100], \*b;

b = a;

while (fread(b, sizeof(ElemType), 1, fp))

{

b++;

}

b = a;

T = CreateBiTree2(T, &b);

fclose(fp);

return T;

}

BiNode \*CreateBiTree2(BiNode \*T, ElemType \*\*p)

{ //利用递归实现二叉树的加载，该函数使用了二级指针

char ch;

BiNode \*trans = NULL;

ch = \*\*p;

ElemType \*q = \*p;

q = q + 1;

\*p = q;

if (ch != '#')

{

if (!(T = (BiNode \*)malloc(sizeof(BiNode))))

return NULL;

T->elem = ch;

T->key = ++key\_value;

trans = CreateBiTree2(T->left, p);

T->left = trans;

if (trans != NULL)

trans->pre = T;

trans = CreateBiTree2(T->right, p);

T->right = trans;

if (trans != NULL)

trans->pre = T;

}

else

{

T = NULL;

}

return T;

}

BiNode visualprint(BiNode \*T){

int i=pow(2,BiTreeDepth(T)),n=2,k=0;

BiNode queue[100], \*front = queue, \*back = queue;

if (T != NULL)

{

\*back = \*T;

back++;

while (front != back)

{

if (front->left != NULL)

{

\*back = \*front->left;

back++;

}

if (front->right != NULL)

{

\*back = \*front->right;

back++;

}

while(k<i/n)

printf(" ");

printf("%c", front->elem);

front++;

}

}

**}** 附录D 基于邻接表图实现的源程序

/\*Graph On Adjacency List\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#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 DG 1

#define DN 2

#define UDG 3

#define UDN 4 //常量定义

#define INITLENGTH 80

#define INCREMENT 20

#define MAXVERTEXNUM 20

int visit[20];

typedef int status;

typedef int ElemType;

typedef int KeyType;//数据元素类型定义

typedef struct ArcNode { //弧度定义

int rad;

struct ArcNode \*next;

}ArcNode;

typedef struct Vertex { //顶点定义

char data;

int key;

struct ArcNode \*firstArc;

}Vertex;

typedef struct Graph{ //图定义

struct Vertex elem[MAXVERTEXNUM];

char name[100];

int vernum; //顶点数

int arcnum; //弧度数

int id; //表示该图的id值

struct Graph \*next;

struct Graph \*pre;

}Graph;

status CreateGraph(char value[],int arcnode[],int vnum,int arcnum,Graph \*a);

status DestroyGraph(Graph \*a);

Vertex \*LocateVex(Graph \*a,int key);

status PutVex (Graph \*a,int key,char value);

Vertex \*FirstAdjVex(Graph \*a,int key);

Vertex \*NextAdjVex(Graph \*a,int v,int w);

status InsertVex(Graph \*a,char v);

status DeleteVex(Graph \*a,int v); //v是删除节点的键值

status InsertArc(Graph \*a,int v,int w);

status DeleteArc(Graph \*a,int v,int w);

status DFSTraverse(Graph \*a);

void DFS(Graph \*a,int v);

status BFSTraverse(Graph \*a);

void BFS(int i,Graph \*a);

status SaveGraph(Graph \*a);

status LoadGraph(Graph \*a);

int main(void)

{

Graph \*head=NULL,\*tail=NULL,\*p=NULL,\*q=NULL;

int op = 1,vnum=0,anum=0,i=0,id=0,key\_value;

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" 1. CreateGraph 9. InsertArc\n");

printf(" 2. DestroyGraph 10. DeleteArc\n");

printf(" 3. LocateVex 11. DFSTraverse\n");

printf(" 4. PutVex 12. BFSTrabverse\n");

printf(" 5. FirstAdjVex 13.SaveGraph\n");

printf(" 6. NextAdjVex 14.LoadGraph\n");

printf(" 7. InsertVex \n");

printf(" 8. DeleteVex \n");

printf(" 0. Exit \n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d", &op);

switch (op)

{

case 1:

if(head==NULL){

head=(Graph \*)malloc(sizeof(Graph));

tail=head;

head->id = 1;

p=head;

head->pre=NULL;

head->next=NULL;

}

else if (head != NULL)

{

tail = (Graph \*)malloc(sizeof(Graph));

p->next = tail;

tail->pre = p;

tail->id = p->id + 1;

p = tail;

tail->next = NULL;

}

q = head;

printf("请输入图的名称:\n");

fflush(stdin);

gets(tail->name);

printf("请输入顶点数和边数:\n");

scanf("%d %d",&vnum,&anum);

char a[1000];

int b[1000];

printf("请输入顶点的值:\n");

fflush(stdin);

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

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

fflush(stdin);

}

printf("请输入弧的信息(a,b):\n");

for(i=0;i<anum\*2;i++){

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

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

}

CreateGraph(a,b,vnum,anum,tail);

printf("图%s创建成功!\n",tail->name);

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 2:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

if (q == head)

head = head->next;

if (q == tail)

tail = tail->pre;

DestroyGraph(q);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 3:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

printf("请输入需要查找的顶点的键值:\n");

scanf("%d",&key\_value);

Vertex \*node=NULL;

node=LocateVex(q,key\_value);

if(node==NULL||node->key==-1)

printf("该节点不存在!\n");

else

printf("该顶点的值为%c\n",node->data);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 4:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

char ch;

printf("请输入需要修改的顶点的键值:\n");

scanf("%d",&key\_value);

printf("请输入修改后的节点的值:\n");

fflush(stdin);

scanf("%c",&ch);

if(PutVex(q,key\_value,ch)==ERROR)

printf("修改失败!\n");

else

printf("修改成功\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 5:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

Vertex \*p=NULL;

printf("请输入需要寻找的弧度的顶点的键值:\n");

scanf("%d",&key\_value);

p=FirstAdjVex(q,key\_value);

if(p==NULL)

printf("不存在第一邻接点!\n");

else

printf("第一邻接点值为:%c\n",p->data);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 6:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

Vertex \*p=NULL;

int v,w;

printf("请输入需要寻找下一邻接点的两个顶点的键值:(a,b)\n");

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

p=NextAdjVex(q,v,w);

if(p==NULL)

printf("不存在下一邻接点!\n");

else

printf("下一邻接点值为:%c\n",p->data);

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 7:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

char ch;

int re;

printf("请输入需要添加的点的值:\n");

fflush(stdin);

scanf("%c",&ch);

re=InsertVex(q,ch);

if(re==OK)

printf("节点成功添加!\n");

else

printf("节点添加失败\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 8:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

int re,key;

printf("请输入需要删除的顶点的键值:\n");

scanf("%d",&key);

re=DeleteVex(q,key);

if(re==OK)

printf("该节点成功删除!\n");

else

printf("该节点删除失败!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 9:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

int v,w,re;

printf("请输入需要插入的弧的信息:(a,b)\n");

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

re=InsertArc(q,v,w);

if(re==OK)

printf("弧成功插入!\n");

else

printf("弧插入失败!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 10:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

int v,w,re;

printf("请输入需要删除的弧的信息:(a,b)\n");

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

re=DeleteArc(q,v,w);

if(re==OK)

printf("弧成功删除!\n");

else

printf("弧删除失败!\n");

}

q = head;

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 11:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

DFSTraverse(q);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 12:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

BFSTraverse(q);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 13:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

SaveGraph(q);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 14:

printf("请输入图的id:\n");

scanf("%d", &id);

while (q != NULL && q->id != id)

q = q->next;

if (q == NULL)

printf("图不存在！\n");

else

{

LoadGraph(q);

}

q = head;

printf("\n请输入任意键继续\n");

getchar();

getchar();

break;

case 15:

printf("请输入任意键继续\n");

getchar();

getchar();

break;

case 16:

printf("请输入任意键继续\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");

return 0;

} //end of main()

status CreateGraph(char value[],int arcnode[],int vnum,int arcnum,Graph \*a){

if(a==NULL)

return ERROR;

int i;

ArcNode \*p=NULL;

a->vernum=vnum;

a->arcnum=arcnum;

for(i=0;i<a->vernum;i++){

(a->elem+i)->data=value[i];

(a->elem+i)->firstArc=NULL;

(a->elem+i)->key=i+1;

}

for(i=0;i<a->arcnum\*2;i++){

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

p->next=a->elem[arcnode[i]-1].firstArc;

a->elem[arcnode[i]-1].firstArc=p;

p->rad=arcnode[++i];

}

return OK;

}

status DestroyGraph(Graph \*a){

if (a->pre != NULL) //删除存储树的表头

a->pre->next = a->next;

if (a->next != NULL)

a->next->pre = a->pre;

printf("图%s删除成功！\n",a->name);

free(a);

a = NULL;

return OK;

}

Vertex \*LocateVex(Graph \*a,int key){

int i;

for(i=0;i<a->vernum;i++)

{

if(key==(a->elem+i)->key)

return (a->elem+i);

}

return NULL;

}

status PutVex (Graph \*a,int key,char value){

Vertex \*p=NULL;

p=LocateVex(a,key);

if(p==NULL)

return ERROR;

p->data=value;

return OK;

}

Vertex \*FirstAdjVex(Graph \*a,int key){

Vertex \*p=NULL,\*q=NULL;

p=LocateVex(a,key);

if(p==NULL)

return NULL;

if(p->firstArc==NULL)

return NULL;

else{

q=LocateVex(a,p->firstArc->rad);

return q;

}

}

Vertex \*NextAdjVex(Graph \*a,int v,int w){

Vertex \*p=LocateVex(a,v);

if(p==NULL)

return NULL;

ArcNode \*q=p->firstArc;

while(q!=NULL){

if(q->rad==w){

if(q->next==NULL)

return NULL;

else

return LocateVex(a,q->next->rad);

}

else

q=q->next;

}

if(q==NULL)

return NULL;

}

status InsertVex(Graph \*a,char v){

if(a->vernum>=MAXVERTEXNUM)

return ERROR;

a->elem[a->vernum].data=v;

a->elem[a->vernum].firstArc=NULL;

a->elem[a->vernum].key=a->elem[a->vernum-1].key+1;

a->vernum++;

return OK;

}

status DeleteVex(Graph \*a,int v){

int i;

for(i=1;i<=a->vernum;i++){

DeleteArc(a,v,i);

DeleteArc(a,i,v);

}

Vertex \*p=NULL;

p=LocateVex(a,v);

if(p==NULL)

return ERROR;

while(a->elem[a->vernum].key!=p->key){

p->data=(p+1)->data;

p->key=(p+1)->key;

p->firstArc=(p+1)->firstArc;

p=p+1;

}

a->vernum--;

return OK;

}

status InsertArc(Graph \*a,int v,int w){

Vertex \*q=LocateVex(a,v);

if(q==NULL)

return ERROR;

ArcNode \*p=q->firstArc;

while(p!=NULL){

if(p->rad==w)

return ERROR;

p=p->next;

}

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

p->rad=w;

p->next=q->firstArc;

q->firstArc=p;

a->arcnum++;

return OK;

}

status DeleteArc(Graph \*a,int v,int w){

Vertex \*q=LocateVex(a,v);

if(q==NULL)

return ERROR;

else{

ArcNode \*p=q->firstArc,\*ql=NULL;

if(p==NULL)

return OK;

if(p->rad==w){

q->firstArc=p->next;

free(p);

a->arcnum--;

p=NULL;

return OK;

}

while(p->next!=NULL){

if(p->next->rad==w)

break;

p=p->next;

}

if(p->next==NULL)

return OK;

if(p->next->rad==w){

ql=p->next;

p->next=ql->next;

free(ql);

a->arcnum--;

ql=NULL;

}

return OK;

}

}

status DFSTraverse(Graph\* a)

{

int v=0;

for(v=0;v<a->vernum;v++)

visit[v]=0;

for(v=0;v<a->vernum;v++){

if(!visit[v])

DFS(a,v);

}

return OK;

}

void DFS(Graph \*a,int v){

visit[v]=1;

printf("%c ",a->elem[v].data);

int w=0;

ArcNode \*p=a->elem[v].firstArc;

while(p!=NULL){

Vertex \*q=NULL;

q=LocateVex(a,p->rad);

for(w=0;w<a->vernum;w++){

if(a->elem[w].key==q->key)

break;

}

if(!visit[w])

DFS(a,w);

p=p->next;

}

}

status BFSTraverse(Graph\* a)

{

int v=0,w;

for(v=0;v<a->vernum;v++)

visit[v]=0;;

Vertex \*queue[20],\*pt=NULL,\*front=queue,\*back=queue;

for(v=0;v<a->vernum;v++){

if(!visit[v]){

\*back=a->elem[v];

back++;

while(front!=back){

ArcNode \*q=front->firstArc;

for(w=0;w<a->vernum;w++){

if(a->elem[w].key==front->key)

break;

}

if(!visit[w])

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

visit[w]=1;

front++;

while(q!=NULL){

\*back=\*LocateVex(a,q->rad);

back++;

q=q->next;

}

}

}

}

return OK;

}

status SaveGraph(Graph \*a)

{

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename); // 写文件

if ((fp = fopen(filename, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return 1;

}

fwrite(&a->vernum, sizeof(ElemType), 1, fp);

fwrite(&a->arcnum, sizeof(ElemType), 1, fp);

int i;

for(i=0;i<a->vernum;i++)

{

fwrite(&(a->elem[i].data),sizeof(char),1,fp);

fwrite(&(a->elem[i].key),sizeof(int),1,fp);

}

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

ArcNode \*p=a->elem[i].firstArc;

while(p!=NULL)

{

fwrite(&(a->elem[i].key),sizeof(int),1,fp);

fwrite(&(p->rad),sizeof(int),1,fp);

p=p->next;

}

}

fclose(fp);

printf("文件成功保存!\n");

return OK;

}

status LoadGraph(Graph \*a){

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

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

{

printf("File open error\n ");

return ERROR;

}

int i,vnum,anum;

fread(&(a->vernum), sizeof(int), 1, fp);

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

if(a==NULL)

return ERROR;

ArcNode \*p=NULL;

for(i=0;i<a->vernum;i++){

fread(&((a->elem+i)->data), sizeof(char), 1, fp);

fread(&((a->elem+i)->key), sizeof(int), 1, fp);

(a->elem+i)->firstArc=NULL;

}

int b[100];

for(i=0;i<a->arcnum\*2;i++){

fread(&b[i],sizeof(int),1,fp);

}

for(i=0;i<a->arcnum\*2;i++){

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

p->next=LocateVex(a,b[i])->firstArc;

LocateVex(a,b[i])->firstArc=p;

p->rad=b[++i];

}

fclose(fp);

return OK;

}