

华中科技大学

# 课程实验报告

课程名称: 数据结构实验

专业班级: 物联网 1601

学 号: U201614890

姓 名: 徐光磊

指导教师: 李国徽

报告日期: 2018 年 1 月 6 日

计算机科学与技术学院

## 目录

<b>1 基于顺序存储结构实现线性表的基本运算</b> .....	2
1. 1 问题描述.....	2
1. 2 顺序表演示系统设计.....	4
1. 3 顺序表演示系统实现与测试.....	8
1. 4 实验小结.....	17
<b>2 基于链式存储结构实现线性表的基本运算</b> .....	19
2. 1 问题描述.....	19
2. 2 单链表演示系统设计.....	19
2. 3 单链表演示系统实现与测试.....	27
2. 4 实验小结.....	36
<b>3 基于二叉链表的二叉树实现</b> .....	38
3. 1 问题描述.....	38
3. 2 二叉树演示系统设计.....	38
3. 3 二叉树演示系统实现与测试.....	44
3. 4 实验小结.....	57
<b>4 基于邻接表的无向图实现</b> .....	58
4. 1 问题描述.....	58
4. 2 无向图演示系统设计.....	58
4. 3 无向图演示系统实现与测试.....	67
4. 4 实验小结.....	72
 参考资料.....	74
附录.....	75

# 1 基于顺序存储结构实现线性表的基本运算

## 1.1 问题描述

叙述实验中线性表的物理结构形式，用物理结构表示数据元素间的逻辑关系以及为此为基础的文件读写功能，设计了如下的数据逻辑结构和基本运算。

### 1.1.1 顺序线性表抽象数据类型：

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种 基本运算，具体数据和运算功能定义如下。

ADT LkList {

    数据对象:  $D = \{a_i \mid a_i \in ElemSet, i = 1, 2, \dots, n, n \geq 0\}$

    数据关系:  $R6 = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 1, 2, \dots, n\}$

(1) 初始化表: InitList(SqList \*\* L)

    初始条件: L 为一个空指针;

    操作结果: 构造了一个空的顺序表，并修改了 L 的值。成功则返回 TRUE，失败返回 FALSE。

(2) 销毁表: DestroyList(SqList \* L)

    初始条件: L 指向一个顺序表;

    操作结果: 清空数据表所占的空间。

(3) 清空表: ClearList (SqList \* L)

    初始条件: L 指向一个顺序表;

    操作结果: 将 L 指向的顺序表数据清空。

(4) 判定空表: ListEmpty(SqList \* L)

    初始条件: L 指向一个顺序表;

    操作结果: 若 L 为空表，则返回 TRUE，否则返回 FALSE。

(5) 求表长: ListLength(SqList \* L)

初始条件: L 指向一个顺序表;

操作结果: 返回 L 中数据元素的个数。

(6) 获得元素: GetElem(SqList \* L, int i, ElemType \* e)

初始条件: L 指向一个顺序表;

操作结果: 将 L 中第 i 个数据元素的值赋值于 e。

(7) 查找元素: LocateElem(SqList \* L, ElemType \* e)

初始条件: 线性表已存在;

操作结果: 返回 L 中第 1 个与 e 相等的元素的索引值, 若不存在, 则返回-1。

(8) 获得前驱: PriorElem(SqList \* L, ElemType \* cur\_e, ElemType \* pre\_e)

初始条件: 线性表 L 已存在;

操作结果: 将线性表 L 中元素 cur\_e 的上一个元素的地址值赋给 pre\_e, 若遇到问题, 则将其赋值为空指针。

(9) 获得后继: NextElem(SqList \* L, ElemType \* cur\_e, ElemType \* next\_e)

初始条件: 线性表 L 已存在;

操作结果: 将线性表 L 中元素 cur\_e 的下一个元素的地址值赋给 next\_e, 若遇到问题, 则将其赋值为空指针。

(10) 插入元素: ListInsert(SqList \* L, int i, ElemType \* e)

初始条件: 线性表 L 已存在;

操作结果: 在 L 的第 i 个位置之前插入新的数据元素 e, 让 e 落在位置 i 上, L 的长度加 1。操作成功返回 TRUE, 失败返回 FALSE

(11) 删除元素: ListDelete(SqList \* L, int i, ElemType \* e)

初始条件: 线性表 L 已存在;

操作结果: 删除 L 的第 i 个数据元素, 用 e 返回其值, L 的长度减 1, 若无该元素。操作成功返回 TRUE, 失败返回 FALSE。

(12) 遍历表: ListTraverse(SqList \* L)

初始条件: 线性表 L 已存在;

操作结果: 依次输出顺序表 L 中的所有数据。

(13) 读取线性表: LoadList(SqList \*\* L, char \* name)

初始条件: 线性表 L 已存在且为空;

操作结果：从名为 name 的文件中读取线性表。

(14) 保存线性表：SaveList (SqList \* L)

初始条件：线性表已存在；

操作结果：保存线性表到文件名为顺序表 L 名字的文件中。

}

### 1.1.2 实验目标

构造成具有功能菜单的系统完成线性表基本操作。通过实验达到：(1) 加深对线性表的概念、基本运算的理解；(2) 熟练掌握线性表的逻辑结构与物理结构的关系；(3) 物理结构采用顺序表，熟练掌握线性表的基本运算的实现。

## 1.2 顺序表演示系统设计

### 1.2.1 系统总体设计

系统的总体架构：由进行展开菜单界面。界面通过 switch 函数进行功能的选择，进入 1.1.1 中的相关功能函数执行相关操作。

菜单界面如下图：



### 1.2.2 有关常量和类型定义

相关常量的定义：

```

typedef int ElemType; //make sure that it's a basic data type

typedef int Status;

#define TRUE 1
#define FALSE 0
#define OK 1

```

```
#define LIST_INIT_SIZE 100 //init length  
#define LISTINCREMENT 10 //step-length when growing and cutting  
#define LISTGAP 15//redundant length needed to decide cutting list or not
```

### 顺序表的存储结构:

```
typedef struct {  
    ElemType * elem;  
    int length;  
    int listsiz;  
    char * name;//length = 30  
} SqList;
```

## 1.2.3 算法设计

### (1) InitList(SqList \*\* L)

设计: 分配存储空间, 并将 length 值设为 0, listsiz 值设为预定义的初始存储容量。

复杂度: 时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

### (2) DestroyList(SqList \* L)

设计: 释放所有内存, 并让 L 指向 NULL。

复杂度: 时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

### (3) ClearList (SqList \* L)

设计: 将表长设为 0, 将 L 的数据清空, 表长置为 0。

复杂度: 时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

### (4) ListEmpty(SqList \* L)

设计: 读取表长 length 的值, 为 0 则返回 TRUE, 否则返回 FALSE。

复杂度: 时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(5) ListLength(SqList \* L)

设计：返回  $L \rightarrow length$  的值。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

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

设计：将  $L \rightarrow elem[i - 1]$  的值赋给  $e$ ，成功则返回 TRUE，失败返回 FALSE。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

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

设计：遍历顺序表，将与给定元素  $e$  相等元素的位序返回。若找不到则返回 -1。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

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

设计：从头遍历，若当前节点后继值为  $cur\_e$ ，则将当前结点的元素赋给  $pre\_e$ ，成功则返回 OK。若未找到，则返回 FALSE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

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

设计：从头遍历，若当前节点值为  $cur\_e$  且非表尾，则将后继结点的元素赋给  $pre\_e$ ，return OK。若未找到，则返回 FALSE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

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

设计：先判断  $i$  值是否符合要求，不符返回 ERROR。 $i$  值合法，则检查空间大小，若空间不够，增加存储容量。然后将插入位置及之后的元素右移，最后插入  $e$ ，表长加一。若插入前表长已经超过存储上限，则调用 growList 增长表长。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

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

设计：先判断  $i$  值是否符合要求，不符返回 ERROR。 $i$  值合法，则将被删除元素的值赋给  $e$ ，然后将被删除元素之后的元素左移，最后表长减 1。若删除后存储空间与表

长相差 15，则调用 cutList 缩小表长。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(12) ListTraverse(SqList \* L)

设计：遍历顺序表，输出每个元素的值。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(13) LoadList(SqList \*\* L, char \* name)

设计：打开需要读取数据对应的文件，若不能打开，返回 ERROR，否则用 fread 函数将数据读入到线性表中，每读一次，线性表长度加 1，读取结束后返回 OK。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(14) SaveList(SqList \* L)

设计：打开或创建要保存的文件，然后使用 fwrite 函数将顺序表保存入文件中。成功返回 TRUE，失败返回 FALSE。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(15) growList (SqList \* L)

设计：当顺序表中表长度不足的时候，及时对顺序表进行扩充。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(16) cutList(SqList \* L)fgets

设计：当顺序表中表长度过长的时候，即表长和开辟空间长度相差 15 个单位以上时，对顺序表进行减少空间的操作，减少空间占用。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(17) fgetsNoN(char \*buf, int bufsize, FILE \*stream)

设计：作为一个 fgets 的重载函数，起到获得用户输入的完整文本，并抛去换行符。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

## 1.3 顺序表演示系统实现与测试

### 1.3.1 实验环境

编程环境：WIN 10 下使用 CodeblocksGNU GCC 进行编译调试，语言为纯 C 语言。

各函数间调用关系：主函数通过 switch 调用各功能函数，各函数间相互独立，只通过主函数中传递指针到各个参数中实现函数内部逻辑。

程序源代码：见附录一。

### 1.3.2 系统测试

使用的测试用例： {1,2,3,4,5,6,7,8,9,10}

(1) InitList 测试：

```
Menu for Linear Table On Sequence Structure
-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:1

Please input the name of new list.
tree
Linear Table has been created
请按任意键继续. . . |
```

创建了一个空表名叫 tree，内部有一定的空间，但没有任何数据。

(2) DestroyList 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:2
请按任意键继续. . .
```

销毁表之前表中有十个整数数据，销毁后，菜单界面便可以见到没有了顺序表名的提示，证明表已经从内存中删除。

(3) ClearList 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:3
请按任意键继续. . . |
```

调用了本函数后，顺序表的表头空间与 elem 指向的顺序表空间仍被保留，但是内部的数据全被抹除，可以通过宏定义一个不常见的数据来作为缺省。

(4) ListEmpty 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:4

----This list is not empty.
请按任意键继续. . .
```

在有数据的情况下测试结果，显然不为空。

(5) ListLength 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:5

----The length of this list is 10.
请按任意键继续. . .
```

可见返回了正确的顺序表长度。

(6) GetElem 测试：

```
Menu for Linear Table On Sequence Structure
List:tree is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:6

Please input the element's position.
5

----The element you get is 5.

请按任意键继续. . .
```

输入该数据的索引位置就可以获得其数据。

(7) LocateElem 测试:

```
Menu for Linear Table On Sequence Structure
List:tree is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:7

Please input the model element.
5

----The index of the first element equal with yours is 5.

请按任意键继续. . .
```

输入数据的值就可以获得其索引位置。

(8) PriorElem 测试:

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----  
1. InitList          8. PriorElem  
2. DestroyList       9. NextElem  
3. ClearList         10. ListInsert  
4. ListEmpty         11. ListDelete  
5. ListLength        12. ListTraverse  
6. GetElem           13. SaveList  
7. LocateElem        14. LoadList  
0. Exit  
-----  
Please select your option [0~12]:8  
Please input the beginning element:4  
----The prior element you get is 3.  
请按任意键继续. . .
```

输入一个起始的数据值，返回其前驱元素的值。

(9) NextElem 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----  
1. InitList          8. PriorElem  
2. DestroyList        9. NextElem  
3. ClearList         10. ListInsert  
4. ListEmpty         11. ListDelete  
5. ListLength        12. ListTraverse  
6. GetElem           13. SaveList  
7. LocateElem        14. LoadList  
0. Exit  
-----  
Please select your option [0~12]:9  
Please input the beginning element:7  
----The next element you get is 8.  
请按任意键继续. . .
```

输入一个起始的数据值，返回其后驱元素的值。

(10) ListInsert 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty          11. ListDelete
5. ListLength         12. ListTraverse
6. GetElem            13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:10

---Please input the position:1
---Please input the element:1
----Successful!
请按任意键继续. . .
-----
```

原数据插入后的结果为：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty          11. ListDelete
5. ListLength         12. ListTraverse
6. GetElem            13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:12

-----all elements-----
1 1 2 3 4 5 6 7 8 9 10
-----end-----
请按任意键继续. . .
-----
```

(11) Listdelete 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:11
---Please input the position:8
----Successfully delete element:9!
请按任意键继续. . .
```

可以看到删除了正确的元素。

(12) ListTraverse 测试:

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:12
-----all elements -----
1 2 3 4 5 6 7 8 9 10
----- end -----
请按任意键继续. . .
```

可见对原始数据集进行了一次完整的遍历。

(13) LoadList 测试:

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:14
---Please input list's name:tree
---Successfully load!
请按任意键继续. . .
```

加载指定名的存档后，可见正常读取了所有数据。

(14) SaveList 测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:13
Successful!
请按任意键继续. . . |
```

程序将自动以该列表的名字为存档名。

(15) 鲁棒性测试：

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:14

---Please input list's name:WrongName

---Loading failed!
请按任意键继续. . .
```

输入了错误的文件名后，程序能进行正确的提醒用户。

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

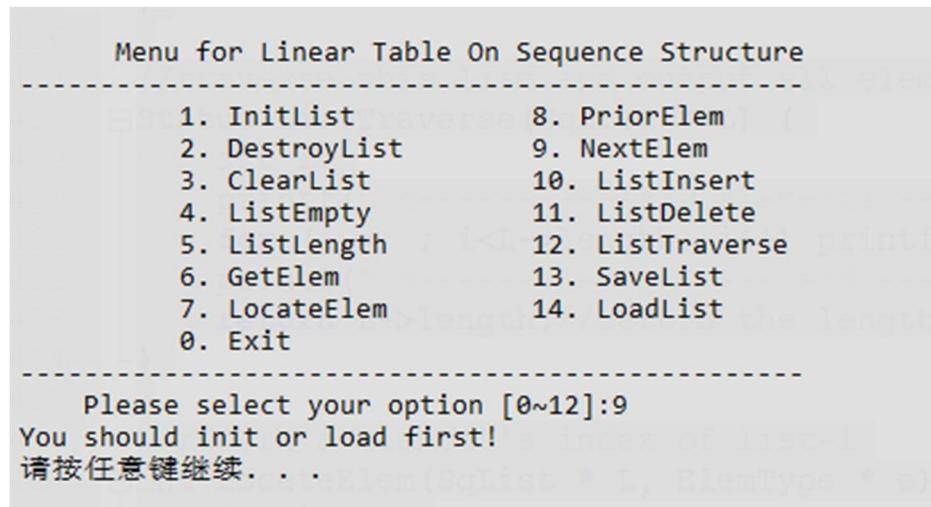
-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:6

Please input the element's position.
77778

----Failed! The position you input is out of bound.
请按任意键继续. . .
```

输入了错误的索引后，程序能够提醒使用者是什么类型的输入错误。



没有创建顺序表的前提进行操作会进行的报错，防止误操作。

## 1.4 实验小结

### 1.4.1 实验中遇到的问题：

实验中遇到了很多 API 定义的问题，书上和任务书里所给的 API 是针对 C++ 进行的函数规范，实际上机操作中，因为使用的是 C 语言，在少了很多高级语言特性的背景下，对所给 API 进行了一定的修改。

### 1.4.2 完成亮点：

- (1) 完成了向磁盘保存和读取顺序表的功能，从而提高了系统的易用性。从这个层面上也间接实现了多表切换同步进行管理的功能。
- (2) 为了降低顺序表对于空间的占用过大问题，学习了 JAVA 8 SDK 中 ArrayList 类的源码实现方法，对顺序表的长度进行动态的扩展与收缩。同时定义了一个扩展与收缩的 Timing 规则，避免了长度在边界震荡时产生的频繁 realloc 造成的额外时空开销。
- (3) 所有变量函数名均遵守骆驼名命名法。误操作均进行了良好的 catch，防止系统崩溃。

### 1.4.3 实验小结：

实验让我又熟悉了 C 语言对于连续内存区域的管理函数，也加深了我对于顺序线性表的逻辑结构和物理结构，从开销程度去了解了程序的运行情况。



## 2 基于链式存储结构实现线性表的基本运算

### 2.1 问题描述

掌握基于链式存储结构，实现线性表的基本的、常见的运算，与以此为基础的多表管理和文件读写功能。

### 2.2 单链表演示系统设计

#### 2.2.1 系统总体设计

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体数据和运算功能定义如下。

```
ADT LkList {
    数据对象:  $D = \{a_i \mid a_i \in ElemSet, i = 1, 2, \dots, n, n \geq 0\}$ 
    数据关系:  $R6 = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 1, 2, \dots, n \}$ 
```

#### 基本函数操作:

(1) InitSqList(SqList \* L)

操作结果：创建链式表组。

(2) InitList(LinkList \* L)

操作结果：构造一个空的链式表

(3) DestroyList(LinkList \* L)

初始条件：链式表 L 已存在。

操作结果：销毁链式表 L。

(4) ClearList (LinkList \* L)

初始条件：链式表 L 已存在。

操作结果：将 L 重置为空链式表。

(5) ListEmpty(LinkList \* L)

初始条件：链式表 L 已存在。

操作结果：若 L 为空链式表，则返回 TRUE，否则返回 FALSE.

(6) ListLength(LinkList \* L)

初始条件：链式表已存在。

操作结果：返回 L 中数据元素的个数。

(7) GetElem(LinkList L,int i,ElemType \* e)

初始条件：单链表已存在， $1 \leq i \leq \text{ListLength}(L)$ 。

操作结果：用 e 返回 L 中第 i 个结点的数据元素值。

(8) LocateElem(LinkList L,ElemType e, Status (\*compare)(ElemType a, ElemType b))

初始条件：链式表已存在。

操作结果：用 e 返回 L 中第 i 个数据元素的值。

(9) PriorElem (LinkList L,ElemType cur\_e,ElemType \* pre\_e)

初始条件：单链表 L 已存在。

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

(10) NextElem (LinkList L,ElemType cur\_e,ElemType \* next\_e)

初始条件：单链表 L 已存在。

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

(11) ListInsert(LinkList \* L,int i,ElemType e)

初始条件：链式表 L 已存在且非空， $1 \leq i \leq \text{ListLength}(L)+1$ 。

操作结果：在 L 的第 i 个结点之前插入新数据元素 e 的结点。

(12) ListDelete(LinkList \* L,int i,ElemType \* e)

初始条件：链式表 L 已存在且非空， $1 \leq i \leq \text{ListLength}(L)$ 。

操作结果：删除 L 第 i 个数据元素的结点，用 e 返回其结点数据元素的值。

(13) ListTraverse(LinkList L, Status (\*visit)(ElemType a))

初始条件：链式表 L 已存在。

操作结果：依次对 L 的每个数据元素调用函数 visit()。一旦调用失败，则操作失败。

(14) ChooseList(LinkList \* L, char s[], SqList L0)

初始条件：链式表 L 已存在

操作结果：选取已有链式表组中的一组链式表

(15) DeleteList(LinkList \* L, char s[], SqList L0)

初始条件：链式表 L 存在

操作结果：删除链式表 L

(16) ReadList(SqList \* L)

初始条件：文件夹中保存有链式表

操作结果：成功读取链式表 L

(17) SaveList(SqList L)

初始条件：已创建链式表 L

操作结果：将链式表 L 保存在创建的文件夹中

## 2.2.2 物理结构

## 1.线性表的数据存储结构

```
//this is the struct taking the ture data
typedef struct lnode {
    ElemType data;
    struct lnode * next;
} LNode;

//this is the struct as the head node taking some infomation of this linked list
typedef struct sqlist {
    int length;
    char * name;
    LNode * head;
} SqList;
```

## 2.相关常量声明

```
typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
```

### 2.2.3 算法设计

(1) InitList(SqList \*\* L)

创建一个新的空链表，并将 L 指向该新空链表，长度设置为 0。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(2) DestroyList(SqList \* L)

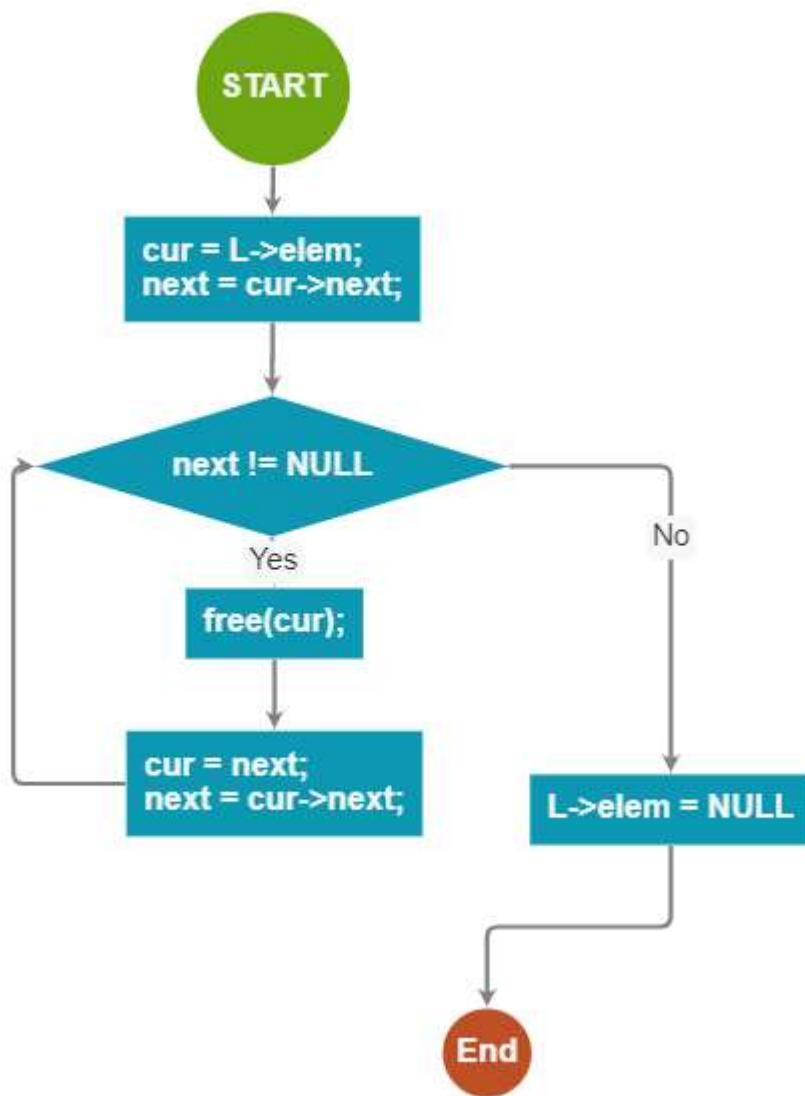
销毁链表的核心在于双指针的前进配合，一个指针负责保存每个 next 针从而不丢失指向下一个数据域的指针，另一个指针负责 free 后面的元素空间，直到完整地遍历完整个数据链表，最后释放链表头的空间。实际操作中，先调用 ClearList，再清空表头，起到代码复用的作用。

时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(3) ClearList (SqList \* L)

清空链表中的元素数据，一个指针负责保存每个 next 针从而不丢失指向下一个数据域的指针，另一个指针负责 free 后面的元素空间，直到完整地遍历完整个数据链表，最后将表头的代表长度的值设为 0。



时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

#### (4) ListEmpty(SqList \* L)

通过读取表头的长度信息，判断链式表是否为空。

时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

#### (5) ListLength(SqList \* L)

通过读取表头的长度信息，返回链式表的长度。

时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

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

利用结点 p 遍历链式表 L，再利用局部变量整形 j 来判断是否等于传入的参数 i，如果

相等，则表示本处元素为外部所要的元素，则向参数 e 赋值该处元素，若找不到则向其赋值为空指针，最后返回操作的结果（TRUE or FALSE）。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(7) LocateElem(SqList \* L, ElemtType \* e)

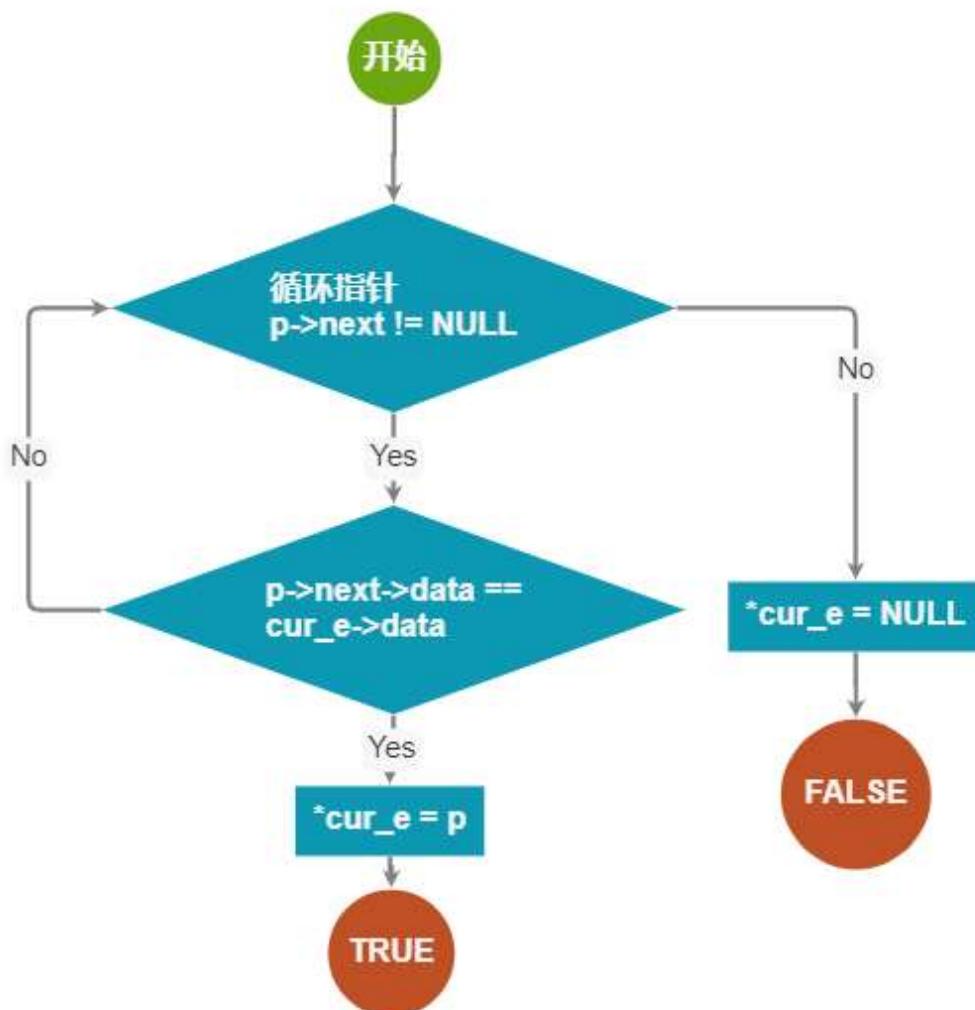
利用 p 指针遍历链表，如果在遍历过程中找到元素 e（相等）则返回此时整形变量 i 的计数值，就是该元素在链表中的索引位置。若找不到则返回-1。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(9) PriorElem (SqList \* L, ElemtType \* cur\_e, ElemtType \* pre\_e)

参数为链表头指针 L，查找的元素 cur\_e，用于返回其前驱的元素 pre\_e。本函数先检测查询的是否为首元素，之后使用遍历指针的下一个是否为空作为 while 循环的条件，从头遍历到倒数第二个，若遍历指针的下一个与 cur\_e 相等，则返回遍历指针指向的值。操作成功则返回 TRUE，失败则返回 FALSE。

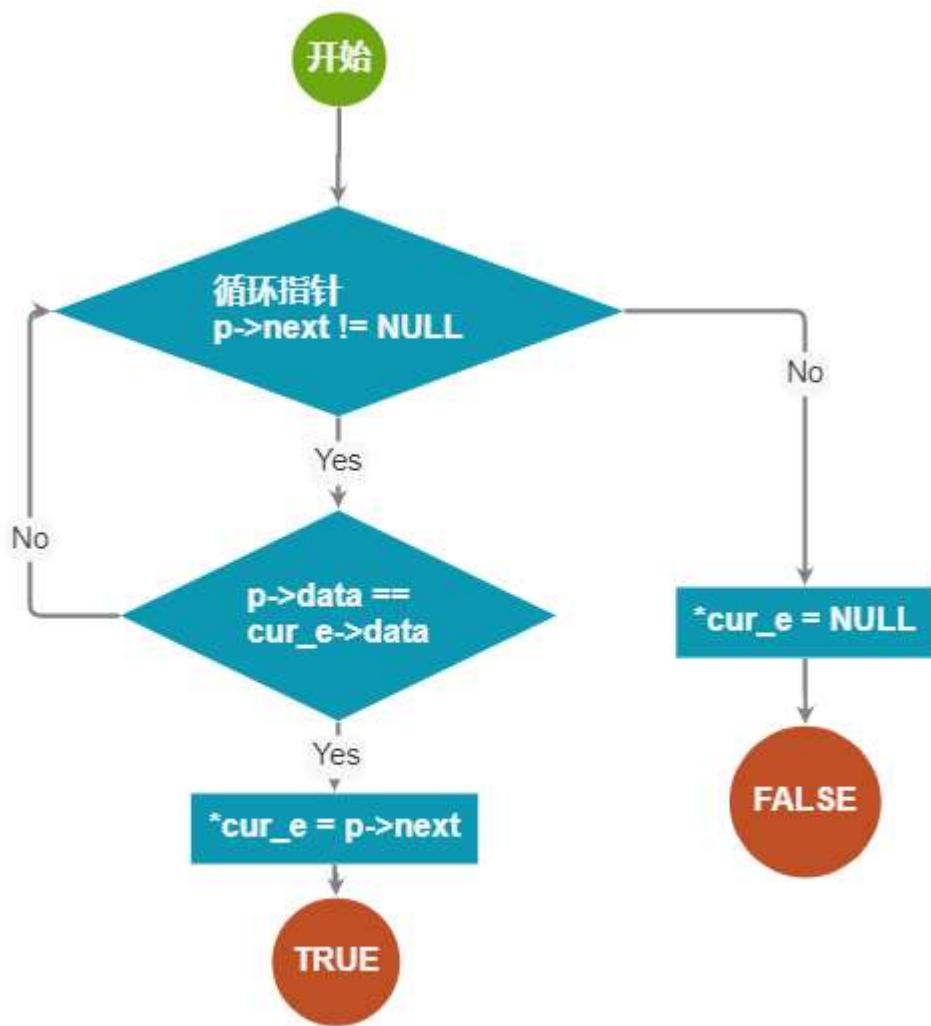


时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(10) NextElem (SqList \* L, ElemtType \* cur\_e, ElemtType \* next\_e)

参数为链表头指针 L, 查找的元素 cur\_e, 用于返回其前驱的元素 pre\_e。本函数先检测查询的是否为首元素, 之后使用遍历指针的下一个是否为空作为 while 循环的条件, 从头遍历到倒数第二个, 若遍历指针指向的元素与 cur\_e 相等, 则返回遍历指针的下一个指针指向的值。操作成功则返回 TRUE, 失败则返回 FALSE。



时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(11) ListInsert(SqList \* L, int i, ElemtType \* e)

输入参数为链表头指针地址 L, 插入位置 i, 插入元素 e, 在 L 的第 i 个结点之前插入新数据元素 e 的结点。使用一个局部变量整形 count 进行计数, 当 count==i 时, 把遍历指针

指向参数 e, e 的指针指向遍历指针的原下一个针指向的元素。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

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

链表头指针 L, 删除位置 i, 用于返回其值的元素 e 的地址。核心算法为使用一个局部变量整形 count 进行计数, 当  $count==i-1$  时, 把当前遍历指针 p 的下一个针指向其下下个元素 (也可能是空指针), 然后 free 原下一个针指向的元素, 最后将其值通过传入的参数 e 完成返回。操作结束后, 根据删除的结果返回 TRUE or FALSE。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(13) ListTraverse(SqList \* L)

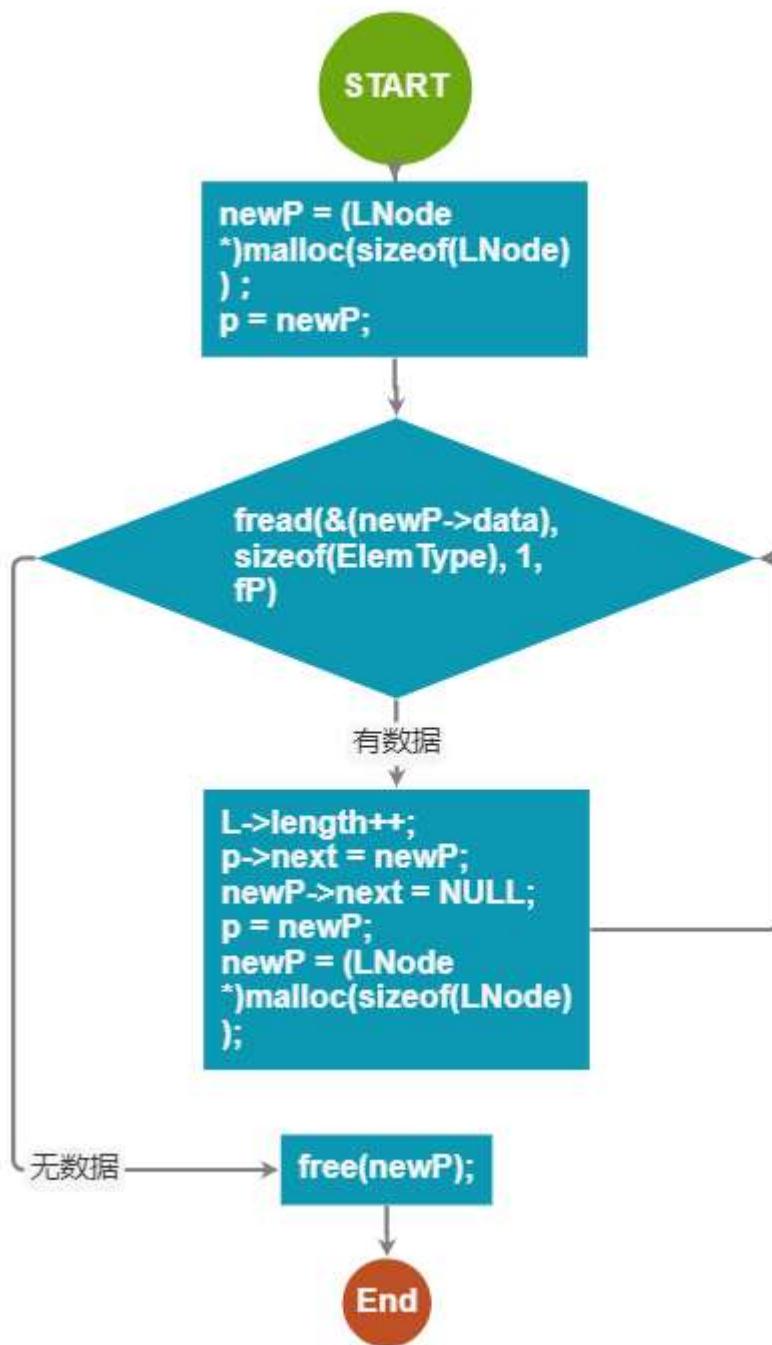
传入参数为链表头指针 L, 通过遍历指针对整个链式表进行遍历输出。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(14) LoadList(SqList \*\* Lp, char \* name)

读取链式表。打开一个名为传入参数 name 的存档文件, 循环 fread 读取文件中的数据, 直到文件尾。算法流程图如下:



时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

#### (15) SaveList(SqList L)

保存链式表。循环使用 `fwrite` 写入内存中链式表的数据。操作结束后，根据结果返回 TRUE or FALSE。

时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

#### (16) fgetsNoN(char \*buf, int bufsize, FILE \*stream)

本方法为 fgets 的一次重载，意图获得一个没有换行符的字符串。通过遍历字符串，将字符’\n’设置为 0，即字符串的结尾标记。

时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

## 2.3 单链表演示系统实现与测试

### 2.3.1 系统实现

程序源代码：见附录 2

### 2.3.2 测试结果

测试用例为{1,2,3,4,5}

(1) InitList 测试：

```

Menu for Linear Table On Sequence Structure
-----
1. InitList          8. PriorElem int
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:1
Please input the name of new list.
list1
Linear Table has been created!
请按任意键继续. . .

```

创建了一个空表名叫 tree，内部有一定的空间，但没有任何数据。

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:12
-----all elements-----
----- end -----

This list is empty!
请按任意键继续. . . |
```

(2) DestroyList 测试:

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList        9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:2
请按任意键继续. . . |
```

销毁表之前表中有五个整数数据，销毁后，菜单界面便可以见到没有了链式表名的提示，证明表已经从内存中删除。

(3) ClearList 测试:

```
Menu for Linear Table On Sequence Structure
List:tree  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:3
请按任意键继续. . .

```

调用了本函数后，链式表中数据链的数据被清除，表头指向的数据指针也被设置为  
空指针。

(4) ListEmpty 测试：

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:4
----This list is not empty.
请按任意键继续. . .

```

在有数据的情况下测试结果，显然不为空。

(5) ListLength 测试：

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:5

----The length of this list is 5.
请按任意键继续. . .
```

可见返回了正确的顺序表长度。

(6) GetElem 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:6

Please input the element's position.
2

----The element you get is 2.
请按任意键继续. . . |
```

输入该数据的索引位置就可以获得其数据。

(7) LocateElem 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:7

Please input the model element.
1

----The index of the first element equal with yours is 1.
请按任意键继续. . .
```

输入数据的值就可以获得其索引位置。

(8) PriorElem 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:8

Please input the beginning element:4

----The prior element you get is 3.
请按任意键继续. . . |
```

输入一个起始的数据值，返回其前驱元素的值。

(9) NextElem 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:9
Please input the beginning element:2
----The next element you get is 3.
请按任意键继续. . . |
```

输入一个起始的数据值，返回其后驱元素的值。

(10) ListInsert 测试：

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList        9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty          11. ListDelete
5. ListLength         12. ListTraverse
6. GetElem            13. SaveList
7. LocateElem         14. LoadList
0. Exit

-----
Please select your option [0~12]:10
---Please input the position:1
---Please input the element:0
----Successful!
请按任意键继续. . . |
```

原数据插入后的结果为：

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:12

-----all elements-----
0 1 2 3 4 5
----- end -----
请按任意键继续. . . |
```

(11) Listdelete 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:11

---Please input the position:3

---Successfully delete element
请按任意键继续. . . |
```

可以看到删除了正确的元素。

```
-----all elements-----
0 1 3 4 5
----- end -----
```

(12) ListTraverse 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:12

-----all elements -----
1 2 3 4 5
----- end -----
请按任意键继续. . .
```

可见对原始数据集进行了一次完整的遍历。

(13) LoadList 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:14

---Please input list's name:list1

---Successfully load!
请按任意键继续. . .
```

加载指定名的存档后，可见正常读取了所有数据。

(14) SaveList 测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:13
Successful!
请按任意键继续. . .
```

程序将自动以该列表的名字为存档名。

(15) 鲁棒性测试:

```
Menu for Linear Table On Sequence Structure
List:list1  is under your control .

-----
1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

-----
Please select your option [0~12]:14
---Please input list's name:1212312312
File open error

---Loading failed!
请按任意键继续. . . |
```

输入了错误的文件名后，程序能进行正确的提醒用户。

```
Menu for Linear Table On Sequence Structure
List:1212312312 is under your control .

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:10

---Please input the position:87

---Please input the element:53

----Failed! The position you input is out of bound.
请按任意键继续. . . |
```

输入了错误的索引后，程序能够提醒使用者是什么类型的输入错误。

```
Menu for Linear Table On Sequence Structure

1. InitList          8. PriorElem
2. DestroyList       9. NextElem
3. ClearList         10. ListInsert
4. ListEmpty         11. ListDelete
5. ListLength        12. ListTraverse
6. GetElem           13. SaveList
7. LocateElem        14. LoadList
0. Exit

Please select your option [0~12]:13
You should init or load first!
请按任意键继续. . .
```

没有创建链式表的前提下进行操作会进行的报错，防止误操作。

## 2.4 实验小结

### 2.4.1 实验问题

本次实验中考察了较多的是针对链式表的 CURD 操作，面对相对上一次实验更为复杂的数据结构时，函数操作后经常会忘记指针指向的位置，从而导致误操作，后来通过图解来帮助自己更好的体会了链式表的数据结构。

### 2.4.2 特色

- (1) 额外完成了链式表数据的向磁盘保存与读取操作，从而间接实现了多表操作的功能，提高了系统的可用性。
- (2) 所有变量函数名均遵守骆驼名命名法，对函数均有注释介绍，界面与逻辑的良好分离。误操作进行了良好的 catch，防止系统崩溃。
- (3) 在判定数据长度函数及其相关依赖函数时，通过一个存储在表头的整数常量节省了  $O(n)$  的查询时间。

#### 2.4.3 实验小结

通过本次实验，我又加深了对于链式表操作的熟悉程度，熟悉了 C 语言中对于链式数据结构在堆栈区的存储逻辑。因为上次实验中对于函数 API 进行了凉的规范，界面与逻辑也做了较好的分离，因此本次实验的工程量并不大，只需要修改逻辑库即可。

### 3 基于二叉链表的二叉树实现

#### 3.1 问题描述

基于二叉链表，通过实验加深对二叉树的概念、基本运算的理解；熟练掌握二叉树的逻辑结构与物理结构的关系；以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

#### 3.2 二叉树演示系统设计

##### 3.2.1 系统总体设计

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种 基本运算，具体数据和运算功能定义如下。

```
ADT BinaryTree {
    数据对象:  $D = D$  是具有相同特性的数据元素的集合
    数据关系:  $R$ :
        若  $D = \emptyset$ , 则  $R = \emptyset$ , 称 BinaryTree 为空二叉树;
        若  $D \neq \emptyset$ , 则  $R = H$ ,  $H$  是如下二元关系:
            (1) 在  $D$  中存在唯一的称为根的数据元素  $root$ , 它在关系  $H$  下无前驱;
            (2) 若  $D - root \neq \emptyset$ , 则存在  $D - root = D_6$ ,  $DS$ , 且  $D_6 \cap DS = \emptyset$ ;
            (3) 若  $D_6 \neq \emptyset$ , 则  $D_6$  中存在唯一的元素  $x_6$ ,  $root, x_6 \in H$ , 且存在  $D_6$  上的关系  $HS \subset H$ ;  $H = root, x_6, root, x_6, DS, H_6, HS$ ;
            (4)  $D_6, H_6$  是一棵符合本定义的二叉树, 成为根的左子树,  $DS, HS$  是一颗符合本定义的二叉树, 成为根的右子树。
```

##### 基本函数操作:

(1) `InitBiTree(&T)`

操作结果：构造空二叉树  $T$ 。

(2) `DestroyBiTree(&T)`

初始条件：二叉树  $T$  存在。

操作结果：销毁二叉树  $T$ 。

(3) `CreateBiTree(&T, definition)`

初始条件：`definition` 给出二叉树  $T$  的定义。

操作结果：按 `definition` 构造二叉树  $T$ 。

(4) `ClearBiTree(&T)`

初始条件：二叉树  $T$  存在。

操作结果：将二叉树  $T$  清为空树。

(5) BiTreeEmpty(T)

初始条件：二叉树 T 存在。

操作结果：若 T 为空二叉树，则返回 TRUE，否则返回 FALSE。.

(6) BiTreeDepth(T)

初始条件：二叉树 T 存在。

操作结果：返回 T 的深度。

(7) Root(T)

初始条件：二叉树 T 存在。

操作结果：返回 T 的根。

(8) Value(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的值。

(9) Assign(T, key, value)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：由 key 知名的节点复制为 value。

(10) Parent(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：若由 key 指明的节点是 T 的非根节点，则返回它的双亲，否则返回 NULL。

(11) LeftChild(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的左孩子。若无左孩子，则返回 NULL。

(12) RightChild(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的右孩子。若无右孩子，则返回 NULL。

(13) LeftSibling(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的左兄弟。若该节点为其双亲的左孩子或无左兄弟，则返回 NULL。

(14) RightSibling(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的右兄弟。若该节点为其双亲的右孩子或无右兄弟，则返回 NULL

(15) InsertChild(T, key, LR, definition)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号，LR 为 0 或 1，由 definition 给定的非空二叉树与 T 不相交且右子树为空。

操作结果：根据 LR 为 0 或 1，插入由 definition 给定的二叉树为 T 中 key 指明的节

点的左或右子树。由 key 指明节点的原有左或右子树则成为 c 的右子树。

(16) DeleteChild(T, key, LR)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号，LR 为 0 或 1。

操作结果：根据 LR 为 0 或 1，删除 T 中 key 指明的节点的左或右子树。

PreOrderTraverse(T) (17) SaveList(SqList L)

初始条件：已创建链式表 L

(16) PreOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：先序遍历 T，并输出每一个节点的值。

(17) InOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：中序遍历 T，并输出每一个节点的值。

(18) PostOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：后序遍历 T，并输出每一个节点的值。

(19) LevelOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：层次遍历遍历 T，并输出每一个节点的值。

} ADT BinaryTree

### 3.2.2 物理结构

#### 1. 二叉树的数据存储结构

**二叉树头的数据类型：**

```
typedef struct node {
    ELEMTYPE key; // a key to find this node
    ELEMTYPE data;
    struct node * left;
    struct node * right;
} Node;
```

**二叉树内数据结点的存储结构：**

```
typedef struct tree {
    int length;
    char * name;

    Node * root;
} Tree;
```

**二叉树数组的存储结构：**

```
Tree ** treeArray = NULL;
```

#### 2. 相关常量声明

```

typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define NULLNODECODE 90017
#define LOAD_LIST_SIZE 31

```

### 3.2.3 算法设计

(1) `InitBiTree(Tree ** T)`

外部传入一个树节点的二级指针，内部进行开辟树头结点的空间，同时初始化各项成员数据，最后向 T 赋值为这个新的二叉树。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(2) `DestroyBiTree(Tree * T)`

先通过调用 `ClearBiTree` 删除二叉树中所有的元素，然后再清除表头空间的数据。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(3) `ClearBiTree(Tree * T)`

利用递归，从二叉树的底部开始清除，任何一个节点的左右子树都被清除之后，自身就会被清除。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(4) `BiTreeEmpty(Tree * T)`

通过查询树表头节点的成员变量 `length`，若不为 0 则不为空。判断的结果通过返回值返回给外部。.

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(5) `BiTreeDepth(Tree * T, int depth)`

利用递归，一层一层的进入子树，通过参数 `depth` 获得当前的层数，外部持有一个全局变量 `depthMax`，若当前 `depth` 超过了 `depthMax` 则更新 `depthMax` 的值。最后返回这个 `depthMax` 值给外部。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(6) `Root(Tree * T)`

访问 T 指向的 `root` 节点，直接通过返回值返回给外部。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(7) **Value(Tree \* T, Node \* e)**

通过递归，前序遍历所有结点，若有一结点的键 key 和 e 的键 key 相同，则返回和这个结点的值，若没有找到，则返回宏定义常量 NULLNODECODE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(8) **Assign(Tree \* T, Node \* e, ElemType value)**

通过递归，前序遍历所有结点，若有一结点的键 key 和 e 的键 key 相同，则对这个结点的数据域进行赋值 value。最后根据操作的结果成功与否，通过返回值返回 TRUE or FALSE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(9) **Parent(Tree \* T, Node \* e)**

通过递归，前序遍历所有非底层的结点，若有一结点的左右子树的键 key 和 e 的键 key 相同，则返回当前结点的指针。若没有找到则返回一个空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(10) **LeftChild(Tree \* T, Node \* e)**

调用 Parent 函数，来获得传入参数 e 的双亲结点，从而获得该双亲节点的左孩子结点，若和 e 不同则将这个左孩子结点返回，若寻找不到双亲节点，或没有左孩子结点，或和 e 相同，则返回空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(11) **RightChild(Tree \* T, Node \* e)**

调用 Parent 函数，来获得传入参数 e 的双亲结点，从而获得该双亲节点的右孩子结点，若和 e 不同则将这个右孩子结点返回，若寻找不到双亲节点，或没有右孩子结点，或和 e 相同，则返回空指针

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(12) **LeftSibling(Tree \* T, Node \* e)**

调用 Parent 函数，来获得传入参数 e 的双亲结点，从而获得该双亲节点的左孩子结点，若这个左孩子结点和 e 不同，且不是空指针，则返回这个左孩子结点指针，否则返回一个空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(13) **RightSibling(Tree \* T, Node \* e)**

调用 Parent 函数，来获得传入参数 e 的双亲结点，从而获得该双亲节点的右孩子结点，

若这个右孩子结点和 e 不同，且不是空指针，则返回这个右孩子结点指针，否则返回一个空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(14) `InsertChild(Tree * T, Node *p, int LR, Tree * c)`

传入的四个参数分别为，插入的二叉树 T，插入的结点 e，e 被插入的位置左右 LR，被插入的二叉树 c。直接将 p 结点的左右子树指针设置为 c 的头指针，然后清空 Tree-c 的表头空间。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(15) `DeleteChild(Tree * T, Node * p, int LR)`

传入的参数为二叉树 T，key 是 T 中某个节点的唯一编号，LR 为 0 或 1，分别对应待删除的左子树或右子树，对被选择的孩子结点调用 `ClearBiTree` 函数来清空其内部的数据。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(16) `PreOrderTraverse(T)`

利用递归来进行前序遍历输出所有数据。在函数内部的逻辑为，传入的结点非空则输出当前参数结点的数据，然后分别对当前结点的左孩子和右孩子递归调用本函数。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(17) `InOrderTraverse(T)`

利用递归来进行中序遍历输出所有数据。在函数内部的逻辑为，传入的结点若为空则退出递归栈，然后对当前参数结点的左孩子调用本函数，再输出当前结点参数的数据，最后对当前结点的右孩子调用本函数。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(18) `PostOrderTraverse(T)`

利用递归来进行后序遍历输出所有数据。在函数内部的逻辑为，传入的结点若为空则退出递归栈，然后输出当前结点参数的数据，再对当前结点的左孩子调用本函数，最后对当前参数结点的右孩子调用本函数。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(19) `LevelOrderTraverse(T)`

本函数常规实现方法应维护一个额外的队列来进行输出。在本程序中，为了减少这部分的存储开销和额外的代码量，采用了多次递归，通过寄存器变量 i 来 for 循环调用函数：`LevelOreder_recurve(Node * node, int depthNow)`（`depthNow` 指当前递归函数所访问的层数），若 `depthNow` 等于外部变量 i 则输出当前结点的数据。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(20) `Node * FindNode(Tree * T, ElemType key)`

利用递归来进行前序遍历，若访问到的结点的键 `key` 与传入的参数 `key` 相等，则返回当前的参数结点，若没有找到的话，则返回一个空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(21) `LoadTree(char * TreeName)`

从磁盘中加载二叉树，打开一个 `TreeName` 为名的存档文件，把所有的树结点数据存入一个数组。因为这个数组中是带了包括空节点的层序遍历序列，所以可以通过当前位置通过运算获得当前结点在数组的位置，从而达到在任何位置都能拿到正确树结点的目的。通过递归可以方便的为树添加这些结点，具体逻辑为先判空，若为空，则返回一个空指针，然后把当前结点的左孩子设为对指定索引调用递归函数的返回值，最后同样对右孩子使用。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(22) `saveTree(Tree * T)`

从磁盘中保存二叉树，保存的序列为带空结点的层序遍历，遍历逻辑同函数 `LevelOrderTraverse`，将这个序列压入一个队列。最后使用 `fwrite` 向磁盘写入出列的树节点数据，文件名为 `Tree-T` 的 `name`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(23) `InsertNode(Tree * T, Node * p, int LR, Node * c)`

向二叉树 `T` 中的结点 `p` 指定位置 `LR` 插入一个新结点 `c`。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(24) `createTreeQuickly(Node** nodeP)`

利用递归来利用一个带空结点的前序遍历字符串来创建新的二叉树，从而方便外部的调用者。每次递归函数时，都打开输入流，从中读取所要的字符串，并用 `atoi` 转换为树结点中的正确数据类型，利用 `malloc` 开辟一个新结点完成初始赋值，再分别对其左右孩子结点调用本函数。因为传入的是一个二级指针，所以可以在某一层递归函数中更新上一层函数的指针值。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(25) `getTreeFromName(char * name)`

因为为了实现多树操作，在 `TreeMain.c` 中有一个全局变量为持有所有树的指针的数组。通过调用本函数可以通过树的名字来获得对应树的指针，从而实现多树操作。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

以上算法函数省略说明了其对应的真正递归调用函数，函数声明为其递归函数的再一层包装。不重复说明包装函数和逻辑抽离函数。

### 3.3 二叉树演示系统实现与测试

### 3.3.1 系统实现

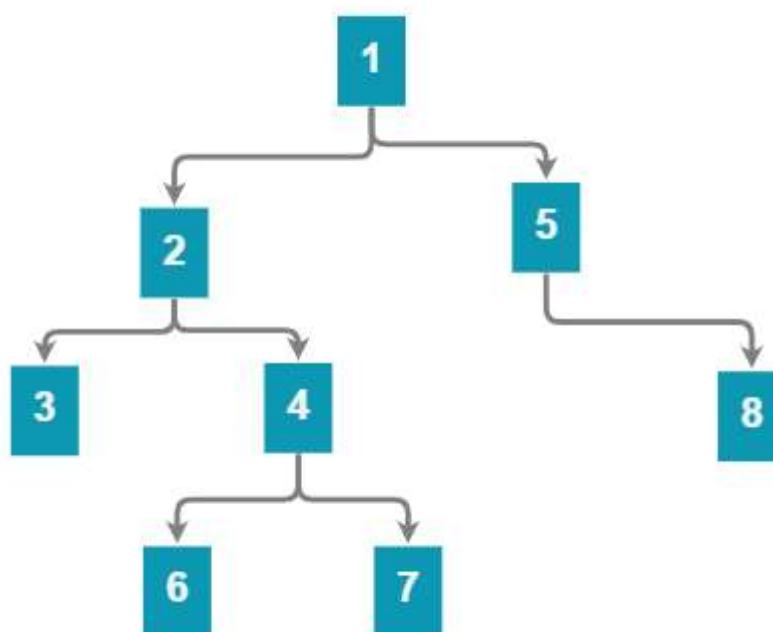
菜单界面：

```
Menu for Binary Tree
-----Trees List-----
-----
1. InitBiTree          2. Assign
3. DestroyTree         4. Parent
5. ClearBiTree         6. LeftChild
7. BiTreeEmpty         8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root              12. RightSibling
13. Value              14. InsertChild
15. DeleteChild        16. PreOrderTraverse
17. InOrderTraverse    18. PostOrderTraverse
19. LevelOrderTraverse 20. LoadTree
21. saveTree           22. InsertNode
23. QuickCreateTree
0. Exit
-----
Please select your option [0~12]:
```

程序源代码：见附录 3

### 3.3.2 测试结果

测试用例为



## (1) InitBiTree(Tree \*\* T)

```

        Menu for Binary Tree
-----Trees List-----
tree ,

-----[1] 1. InitBiTree          [2] 2. Assign
          3. DestroyTree        [4] 4. Parent
          5. ClearBiTree         [6] 6. LeftChild
          7. BiTreeEmpty         [8] 8. RightChild
          9. BiTreeDepth         [10] 10. LeftSibling
          11. Root               [12] 12. RightSibling
          13. Value              [14] 14. InsertChild
          15. DeleteChild        [16] 16. PreOrderTraverse
          17. InOrderTraverse    [18] 18. PostOrderTraverse
          19. LevelOrderTraverse [20] 20. LoadTree
          21. saveTree            [22] 22. InsertNode
          23. QuickCreateTree
          0. Exit

-----[1] Please select your option [0~12]:1

Please input the name of new list:treeNew
Binary Tree has been created
请按任意键继续. . .

```

可以看到成功创建了一颗新的空 树。

## (2) DestroyBiTree(Tree \* T)

```

        Menu for Binary Tree
-----Trees List-----
tree ,treeNew ,

-----[1] 1. InitBiTree          [2] 2. Assign
          3. DestroyTree        [4] 4. Parent
          5. ClearBiTree         [6] 6. LeftChild
          7. BiTreeEmpty         [8] 8. RightChild
          9. BiTreeDepth         [10] 10. LeftSibling
          11. Root               [12] 12. RightSibling
          13. Value              [14] 14. InsertChild
          15. DeleteChild        [16] 16. PreOrderTraverse
          17. InOrderTraverse    [18] 18. PostOrderTraverse
          19. LevelOrderTraverse [20] 20. LoadTree
          21. saveTree            [22] 22. InsertNode
          23. QuickCreateTree
          0. Exit

-----[1] Please select your option [0~12]:3

Please input the name of the tree you want to control:tree

Done!
请按任意键继续. . .

```

成功删除了上面的 tree，界面再次刷新之后，上方的 Trees List 就没有了 tree 的身影。

(3) ClearBiTree(Tree \* T)

```
Please select your option [0~12]:5  
Please input the name of the tree you want to control:tree  
Done!  
请按任意键继续. . .
```

Tree 中原本有样本数据，调用本函数后，系统提示成功被清空了。

(4) BiTreeEmpty(Tree \* T)

```
1. InitBiTree          2. Assign
3. DestroyTree         4. Parent
5. ClearBiTree         6. LeftChild
7. BiTreeEmpty         8. RightChild
9. BiTreeDepth         10. LeftSibling
11. Root              12. RightSibling
13. Value              14. InsertChild
15. DeleteChild        16. PreOrderTraverse
17. InOrderTraverse    18. PostOrderTraverse
19. LevelOrderTraverse 20. LoadTree
21. saveTree           22. InsertNode
23. QuickCreateTree
0. Exit
```

```
Please select your option [0~12]:7  
Please input the name of the tree you want to control:tree  
It's not empty  
请按任意键继续. . . |
```

对样例树调用本函数，返回结果为不为空。

(5) BiTreeDepth(Tree \* T, int depth)

```
Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree          2. Assign
3. DestroyTree         4. Parent
5. ClearBiTree         6. LeftChild
7. BiTreeEmpty         8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root              12. RightSibling
13. Value              14. InsertChild
15. DeleteChild        16. PreOrderTraverse
17. InOrderTraverse    18. PostOrderTraverse
19. LevelOrderTraverse 20. LoadTree
21. saveTree           22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:9

Please input the name of the tree you want to control:tree

It's depth is 4.
请按任意键继续. . .
```

对样例调用本函数，返回了正确的树深度数据。

(6) Root(Tree \* T)

```
Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree          2. Assign
3. DestroyTree         4. Parent
5. ClearBiTree         6. LeftChild
7. BiTreeEmpty         8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root              12. RightSibling
13. Value              14. InsertChild
15. DeleteChild        16. PreOrderTraverse
17. InOrderTraverse    18. PostOrderTraverse
19. LevelOrderTraverse 20. LoadTree
21. saveTree           22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:11

Please input the name of the tree you want to control:tree

Root node: key=1 value=1.
请按任意键继续. . .
```

对样例树调用本函数，成功返回了树里根节点正确的键和值。

(7) Value(Tree \* T, Node \* e)

```
Menu for Binary Tree
-----Trees List-----
tree ,

-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

-----
Please select your option [0~12]:13
Please input the name of the tree you want to control:tree
Please input the key of this node:5
its value is 5请按任意键继续. . . |
```

对样例树调用本函数进行查值，返回了输入的键对应的正确的值。

(8) Assign(Tree \* T, Node \* e, ElemtType value)

```

Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:2

Please input the name of the tree you want to control:tree

Please input the key of this node:1

Please input the new value of this node:10

successful
请按任意键继续. . .

```

对样例树调用本函数，对给定键的树结点的值进行重新赋值。

(9) Parent(Tree \* T, Node \* e)

```

Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:4

Please input the name of the tree you want to control:tree

Please input the key of this node:2
the parent node: key=1 value=10请按任意键继续. . .

```

对样例树调用本函数，函数正确返回了结点 2 的双亲结点 1。

(10) LeftChild(Tree \* T, Node \* e)

```
Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:6

Please input the name of the tree you want to control:tree

Please input the key of this node:2
its left child: key=3, value=3.
请按任意键继续. . .
```

对样例树调用本函数，查询结点 2 的左孩子结点，也返回了正确的结点键值。

(11) RightChild(Tree \* T, Node \* e)

```

        Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:8

Please input the name of the tree you want to control:tree

Please input the key of this node:1
its right child: key=5, value=5.
请按任意键继续. .

```

对样例树调用本函数，查询结点 1 的右孩子结点，也返回了正确的结点键值。

#### (12) LeftSibling(Tree \* T, Node \* e)

```

        Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree           2. Assign
3. DestroyTree          4. Parent
5. ClearBiTree          6. LeftChild
7. BiTreeEmpty          8. RightChild
9. BiTreeDepth          10. LeftSibling
11. Root                12. RightSibling
13. Value               14. InsertChild
15. DeleteChild         16. PreOrderTraverse
17. InOrderTraverse     18. PostOrderTraverse
19. LevelOrderTraverse  20. LoadTree
21. saveTree            22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:10

Please input the name of the tree you want to control:tree

Please input the key of this node:5
its left sibling: key=2 value=2
请按任意键继续. .

```

对样例树调用本函数，查询结点 5 的左兄弟，程序返回了正确的左兄弟结点 2。

(13) RightSibling(Tree \* T, Node \* e)

```

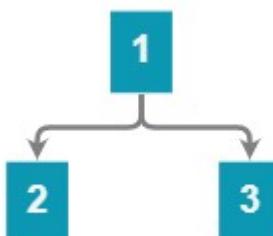
Menu for Binary Tree
-----Trees List-----
tree ,
-----
1. InitBiTree          2. Assign
3. DestroyTree         4. Parent
5. ClearBiTree         6. LeftChild
7. BiTreeEmpty         8. RightChild
9. BiTreeDepth         10. LeftSibling
11. Root              12. RightSibling
13. Value              14. InsertChild
15. DeleteChild        16. PreOrderTraverse
17. InOrderTraverse    18. PostOrderTraverse
19. LevelOrderTraverse 20. LoadTree
21. saveTree           22. InsertNode
23. QuickCreateTree
0. Exit

Please select your option [0~12]:12
Please input the name of the tree you want to control:tree
Please input the key of this node:2
its right sibling: key=5 value=5
请按任意键继续. . .

```

对样例树调用本函数，查询结点 2 的右兄弟，程序返回了正确的右兄弟结点 5。

(14) InsertChild(Tree \* T, Node \*p, int LR, Tree \* c)



传入参数中的新树 Tree-c 为

```

-----
Please select your option [0~12]:14
Please input the name of the tree you want to control:tree
Please input the key of this node:5
Please input the name of the tree you would like to choose to insert:treeNew
Please input 1/0 to choose direction ( 0:L 1:R ):0
successful!
请按任意键继续. . .

```

对样例树调用本函数，向 tree 中的结点 5 的左孩子中插入新树 treeNew。系统返回了操

作正常的提示。

(15) DeleteChild(Tree \* T, Node \* p, int LR)

```
Please select your option [0~12]:15
Please input the name of the tree you want to control:tree
Please input the key of this node:1
Please choose to delete L or R of this node(L->0 R->1)
0
successful!
请按任意键继续. . .
```

对样例树调用本函数，删除结点 1 的左子树。系统返回了操作正常的提示。

(16) PreOrderTraverse(T)

```
Please select your option [0~12]:16
Please input the name of the tree you want to control:tree
key=1 value=1
key=2 value=2
key=3 value=3
key=4 value=4
key=6 value=6
key=7 value=7
key=5 value=5
key=8 value=8
请按任意键继续. . .
```

对样例树调用本函数，可见进行了正确的前序遍历输出。

(17) InOrderTraverse(T)

```
Please select your option [0~12]:17
Please input the name of the tree you want to control:tree
key=3 value=3
key=2 value=2
key=6 value=6
key=4 value=4
key=7 value=7
key=1 value=1
key=5 value=5
key=8 value=8
请按任意键继续. . .
```

对样例树调用本函数，可见进行了正确的中序遍历输出。

(18) PostOrderTraverse(T)

```
Please select your option [0~12]:18
```

```
Please input the name of the tree you want to control:tree
key=3 value=3
key=6 value=6
key=7 value=7
key=4 value=4
key=2 value=2
key=8 value=8
key=5 value=5
key=1 value=1
```

```
请按任意键继续. . .
```

对样例树调用本函数，可见进行了正确的后序遍历输出。

(19) LevelOrderTraverse(T)

```
Please select your option [0~12]:19
```

```
Please input the name of the tree you want to control:tree
key=1 value=1
key=2 value=2
key=5 value=5
key=3 value=3
key=4 value=4
key=8 value=8
key=6 value=6
key=7 value=7
```

```
请按任意键继续. . .
```

对样例树调用本函数，可见进行了正确的层序遍历输出。

(20) LoadTree(char \* TreeName)

```
-----  
Please select your option [0~12]:20  
Please input the name of this tree:tree  
successful  
请按任意键继续. . .
```

从硬盘中成功加载了预先保存好的名为 tree 的存档文件

(21) saveTree(Tree \* T)

```
21. saveTree  
23. QuickCreateTree  
0. Exit
```

```
22. InsertNode
```

```
-----  
Please select your option [0~12]:21
```

```
Please input the name of the tree you want to control:tree
Successful!  
请按任意键继续. . . |
```

成功向硬盘中保存了名为 tree 的二叉树。

(22) InsertNode(Tree \* T, Node \* p, int LR, Node \* c)

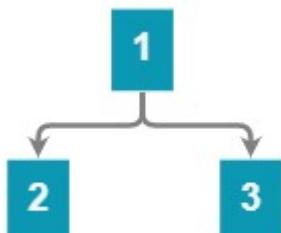
```
Please select your option [0~12]:22  
Please input the name of the tree you want to control:tree  
Please input the key of new node:100  
Please input the value of new node:100  
Please input the key of parent of this node:8  
Please choose left or right( 0:left 1: right ):1  
请按任意键继续. . . |
```

成功地向样例树中插入了一个新的结点。

(23) createTreeQuickly(Node\*\* nodeP)

```
Please select your option [0~12]:23  
Please input the name of new list:treeNew  
Please input the pre-order-traverse( null-># ) data of new list:1 2 # # 3 # #  
Please input the pre-order-traverse key ( split by space ) of new list:1 2 3  
Binary Tree has been created  
请按任意键继续. . . |
```

可见成功的快捷创建了一颗新二叉树，新树为如下图：



(24) 鲁棒性测试

```
Please select your option [0~12]:2  
Please input the name of the tree you want to control:  
There isn't such a tree.  
请按任意键继续. . . |
```

可见输入了错误的树名时，系统会反馈错误的原因。

```
Please select your option [0~12]:14  
Please input the name of the tree you want to control:tree  
Please input the key of this node:89  
There isn't such a node in this tree.  
请按任意键继续. . .
```

可见输入了错误的 key 值时，系统会反馈错误的原因

```
-----  
Please select your option [0~12]:10  
Please input the name of the tree you want to control:tree  
Please input the key of this node:1  
this node don't have a left sibling.  
请按任意键继续. . . |
```

可见操作失败之后也会正常反馈错误信息，起到提醒使用者的作用。

## 3.4 实验小结

### 3.4.1 实验问题

(1) 本次实验中考察了较多的是针对二叉树的 CURD 操作，比前两次线性表的难度要大上不少，由于书本的样例用了较多 C++ 的特性，边花了很多功夫在规范二叉树操作的 API 上。又因为 C 语言较为低级，进行了较多的函数命名进行多层包装。

(2) 由于二叉树的物理结构较为复杂，于是大量地运用了多层的递归，也因此犯了不少错误

### 3.4.2 特色

- (1) 额外完成了二叉树数据的向磁盘保存与读取操作。
- (2) 同一菜单界面下对于多树的同步操作，并利用了二叉树数组对于先后顺序不敏感的特点，将删树加数操作的复杂度变为常数级别。
- (3) 实现了快速创建复杂树的功能，提高了系统对于使用者的易用性。

### 3.4.3 实验小结

本次实验相对之前的实验，无论是代码量或是复杂度都提升了不少，也让我加深了对递归调用的理解和对二叉树物理结构的掌握。

# 4 基于邻接表的无向图实现

## 4.1 问题描述

为实现基于邻接表的图结构以及以此为基础的无向图的管理，按照后文中的形式来设计数据的逻辑结构及基本运算。

## 4.2 无向图演示系统设计

### 4.2.1 系统总体设计

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种 基本运算，具体数据和运算功能定义如下。

ADT ALGraph {

    数据对象  $V$ :  $V$  是具有相同特性的数据元素的集合，称为顶点集。

    数据关系:  $R$ :

$R = \{ VR \}$

$VR = \{ \langle v, w \rangle | v, w \in V \text{ 且 } P(v, w) \}$ ,  $\langle v, w \rangle$  表示从  $v$  到  $w$  的弧，

        谓词  $P(v, w)$  定义了弧  $v, w$  的意义或信息}

**基本函数操作：**

(1) CreateGraph(&G, V, VR)

    初始条件:  $V$  是图的顶点集， $VR$  是图中弧的集合。

    操作结果: 按  $V$  和  $VR$  的定义构造图  $G$ 。

(2) DestroyGraph(&G)

    初始条件: 图  $G$  存在。

    操作结果: 销毁图  $G$ 。

(3) LocateVex(G, u)

    初始条件: 图  $G$  存在， $u$  和  $G$  中顶点有相同特征。

    操作结果: 若  $G$  中存在顶点  $u$ ，则返回该顶点在图中位置；否则返回其他信息。

(4) GetVex(G, v)

    初始条件: 图  $G$  存在， $v$  是  $G$  中某个顶点。

    操作结果: 返回  $v$  的值。

(5) PutVex(&G, v, value)

    初始条件: 图  $G$  存在， $v$  是  $G$  中某个顶点。

操作结果：对  $v$  赋值  $value$ 。.

(6) FirstAdjVex(G, v)

初始条件：图 G 存在， $v$  是 G 中某个顶点。

操作结果：返回  $v$  的第一个邻接顶点。若顶点在 G 中没有邻接顶点，则返回“空”。

(7) NextAdjVex(G, v, w)

初始条件：图 G 存在， $v$  是 G 中某个顶点， $w$  是  $v$  的邻接顶点。

操作结果：返回  $v$  的（相对与  $w$  的）下一个邻接顶点。若  $w$  是  $v$  的最后一个邻接点，则返回“空”。

(8) InsertVex(&G, v)

初始条件：图 G 存在， $v$  和图中顶点有相同特征。

操作结果：在图 G 中增添新顶点  $v$ 。

(9) DeleteVex(&G, v)

初始条件：图 G 存在， $v$  是 G 中某个顶点。

操作结果：删除 G 中顶点  $v$  及其相关的弧。

(10) InsertArc(&G, v, w)

初始条件：图 G 存在， $v$  和  $w$  是 G 中两个顶点。

操作结果：操作结果：在 G 中增添弧 $\langle v, w \rangle$ ，若 G 是无向的，则还增添对称弧 $\langle w, v \rangle$ 。

(11) DeleteArc(&G, v, w)

初始条件：图 G 存在， $v$  和  $w$  是 G 中两个顶点。

操作结果：在 G 中删除弧 $\langle v, w \rangle$ ，若 G 是无向的，则还删除对称弧 $\langle w, v \rangle$ 。

(12) DFSTraverse(G, Visit())

初始条件：图 G 存在，Visit 是顶点的应用函数。

操作结果：对图进行深度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 Visit() 失败，则操作失败。

(13) BFSTraverse(G, Visit())

初始条件：图 G 存在，Visit 是顶点的应用函数。

操作结果：对图进行广度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 Visit() 失败，则操作失败。

} ADT ALGraph

## 4.2.2 物理结构

### 1. 无向图的数据存储结构

**无向图表头的数据类型:**

```
typedef struct graph {
    int vertexCount;
    Vertex * head;
} Graph;
```

**无向图内数据结点的存储结构:**

```
typedef struct vertex {
    ElemType key;
    ElemType data;
    struct vertex * next;
    struct vertex * adj;
} Vertex;
```

**针对无向图数据结点的队列表头存储结构:**

```
typedef struct myqueue {
    int length;
    int listszie;
    Vertex ** head;
} Queue;
```

## 2. 相关常量声明

```
ttypedef int ElemType; //make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define LISTINCREMENT 10 //step-length when growing and cutting
#define LIST_INIT_SIZE 100
#define VERTEX_LIST_SIZE 31
#define NOT_FOUND_CODE 90071
```

### 4.2.3 算法设计

(1) CreateEmptyGraph(Graph \*\* gP)

外部传入一个图头节点的二级指针，内部进行开辟树头结点的空间，同时初始化各项成员数据，最后向 T 赋值为这个新的无向图。最后根据结果返回 TRUE or FALSE。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(2) DestoryGraph(Graph \* g)

遍历一遍  $g->head$  为头的链表，对这条链表中的每个数据调用 `deleteAdjLink`，从而删除

邻接表内所有数据，最后 free 头结点里的空间。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(3) `LocateVex(Graph * g, ElemtType data)`

遍历一遍  $g->head$  为头的链表，若遍历到的结点的数据等于传入的参数 `data`，则返回当前结点的键 `key`，否则返回 `NOT_FOUND_CODE`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(4) `GetVex(Graph * g, ElemtType loc)`

遍历一遍  $g->head$  为头的链表，若遍历到的结点的键值 `key` 等于传入的参数 `loc`，则返回当前结点的数据 `data`，否则返回 `NOT_FOUND_CODE`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(5) `PutVex(Graph * g, Vertex * v, ElemtType value)`

直接为  $v->data$  赋值参数 `value`。若操作成功则返回 `TRUE`，失败则返回 `FALSE`。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(6) `FirstAdjVex(Graph * g, Vertex * v)`

直接返回  $v->adj$  的指针值。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(7) `NextAdjVex(Graph * g, Vertex * v, Vertex * w)`

遍历  $v->adj$  为头的链表，若访问到的结点的键 `key` 等于 `w` 的键 `key`，则返回 `w->adj` 的指针值。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(8) `InsertVex(Graph * g, Vertex * v)`

循环遍历  $g->head$  为头的链表，若这条链表为空，则直接向  $g->head$  赋值为 `v`，若不为空，则在遍历后，在最后一个连接数据的 `adj` 赋值为 `v`。最后 `g->vertexCount` 自加 1。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(9) `DeleteVex(Graph * g, Vertex * v)`

循环遍历  $g->head$  为头的链表，遍历到的结点若 `key` 和参数结点 `v` 的 `key` 不同，则调用函数 `deleteVexFromAdj`，传入参数 `v` 和当前结点，从而删除所有和 `v` 相关的边。再重复循环遍历一次，若遍历到的结点的键 `key` 等于传入的参数 `v` 的键 `key`，则调用函数 `deleteAdjLink`，传入的参数为当前的结点，从而删除这个结点拉出的相连信息。若操作成功返回 `TRUE`，失败则返回 `FALSE`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(10) `InsertArc(Graph * g, Vertex * v, Vertex * w)`

循环遍历  $g->head$  为头的链表，遍历到的结点若 `key` 和参数 `v` 的 `key` 相同，则调用函数 `addVexToAdjEnd`，传入参数为当前遍历到的结点的 `adj` 指针和 `w` 的一个副本数据指针，若和参数 `w` 的 `key` 相同，调用同样的函数，传入参数为当前遍历到的结点的 `adj` 指针和 `v` 的一个副本数据指针。若成功返回 `TRUE`，若失败返回 `FALSE`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(11) `DeleteArc(Graph * g, Vertex * v, Vertex * w)`

循环遍历  $g->head$  为头的链表，遍历到的结点的键 `key` 等于参数 `v` 的 `key`，则调用函数 `deleteVexFromAdj`，传入参数 `w` 和当前遍历到的结点。若遍历到的键 `key` 等于参数 `w` 的 `key`，则调用函数 `deleteVexFromAdj`，传入参数 `v` 和当前遍历到的结点。若成功则返回 `TRUE`，否则返回 `FALSE`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(12) `DFSTraverse(Graph * g)`

先初始化一个长度为图节点数量的数组，用来保存被访问过的结点的键 `key`。递归调用函数 `dfs_recurve(Graph * g, Vertex * v)`，递归逻辑为：向数组中添加参数 `v` 的 `key`，输出这个 `v` 的信息，循环遍历 `v->adj` 为头的链表，若遍历到的结点在数组中没有存储记录（即未被访问过），则递归调用函数 `dfs_recurve`，传入 `g` 和当时遍历到的结点。则程序会从第一个结点开始进行深度遍历访问。

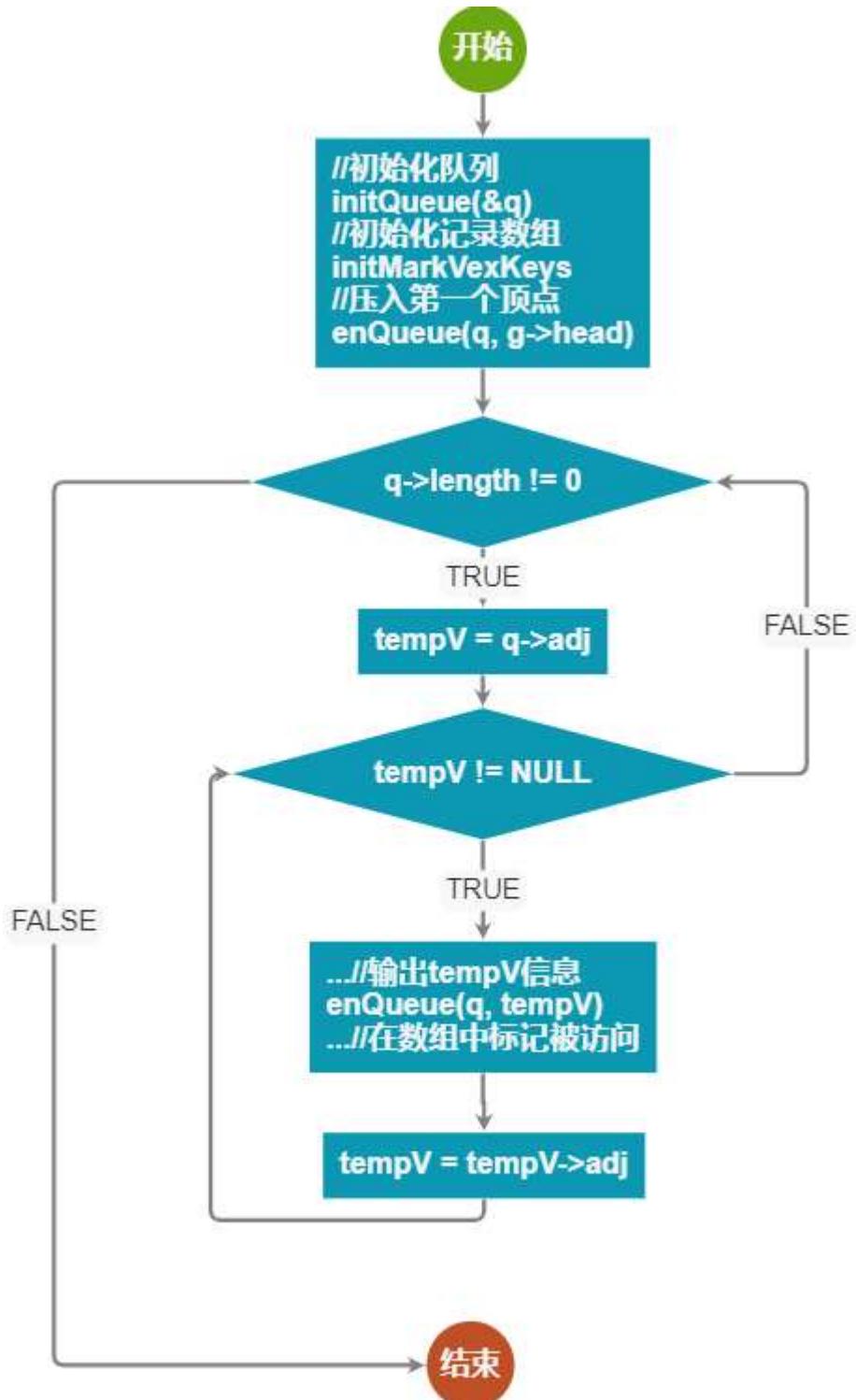
复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

(13) BFSTraverse(Graph \* g)

先初始化一个图结点的队列 q, 再初始化一个长度为图节点数量的数组, 用来保存被访问过的结点的键 key。将第一个定点压入队列, 进入一个 while 循环, 条件为队列 q 不为空, 每次循环中队列出一个元素, 遍历这个元素的连接结点链表, 若被遍历的结点未被访问, 则输出这个结点的信息, 并把这个结点压入队列, 并标记为访问过。最后清空这个队列的空间。

流程图:



复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(n)$

#### (14) deleteVexFromAdj(Vertex \* del, Vertex \* startVex)

传入的参数为待删除的结点 del，待操作的链表来源 startVex。遍历以 startVex->adj 为链表头的链表，若遍历到的结点的下一个数据的键 key 和 del 的键 key 相等，则将遍历到的结点的 adj 针指向下一个数据，再将原 adj 针指向的数据清空。即可删除这条链接信息链表中

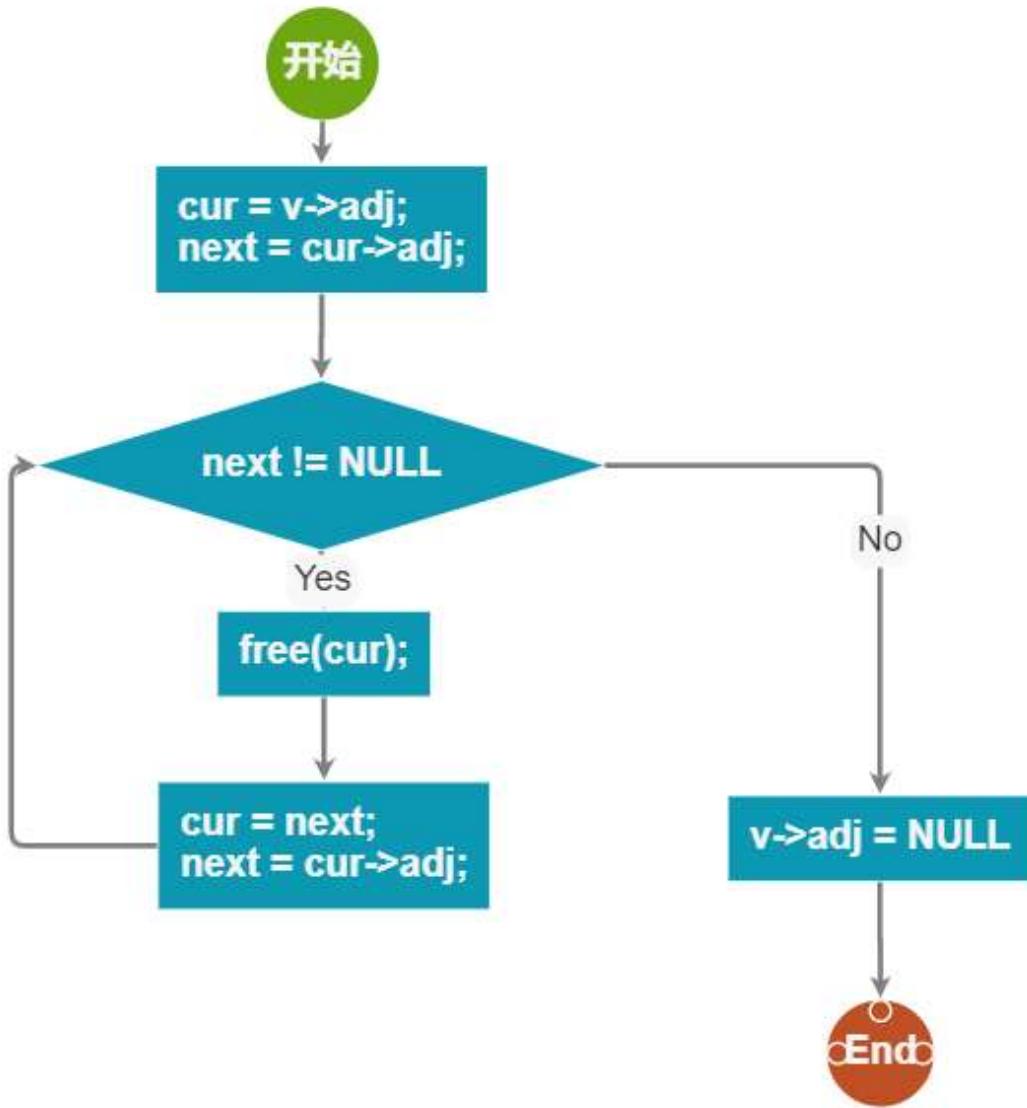
与 del 相关的信息。若操作成功则返回 TRUE，失败则返回 FALSE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(15) `deleteAdjLink(Vertex * v)`

遍历 v 为头的链表头，删除整条链表的信息，具体流程见下流程图。



复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(16) `FindVertex(Graph * g, ElemType key)`

遍历 g->head 为头的链表，若遍历到的结点的键 key 和 key 相等，则返回这个结点，若找不到则返回空指针。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(17) `addVexToAdjEnd(Vertex * startVex, Vertex * newVex)`

遍历 `startVex->adj` 为头的链表，当访问到链表尾后将链表尾的结点的 `adj` 针指向 `newVex`，若 `startVex->adj` 为空，则把 `start->adj` 赋值为 `newVex`。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(18) `isAdj(Graph * g, Vertex * v, Vertex * w)`

遍历 `g->head` 为头的链表，若访问到的结点的键 `key` 等于 `v` 的键 `key` 相等，则开始遍历 `v->adj` 为头的链表，若遍历到的结点的 `key` 与 `w` 的键 `key` 相等，则返回 TRUE，若没有找到这样的结点，则返回 FALSE。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(19) `initQueue(Queue ** qP)`

初始化一个空的基于数组的队列，长度为 `LIST_INIT_SIZE`。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(20) `growList(Queue * q)`

使用 `realloc` 扩充队列的长度，步进长度为 `LISTINCREMENT`。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(21) `cutList(Queue * q)`

使用 `realloc` 缩短队列的长度，步进长度为 `LISTINCREMENT`。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(22) `enQueue(Queue * q, Vertex * v)`

首先判断队列已满，调用 `growList` 扩充队列的长度。向队列尾添加顶点 `v`。最后 `q->length` 自加一。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

(23) `deQueue(Queue * q)`

若队列的空间与队列长度相差 15，则调用函数 `cutList` 缩短队列的长度。删除队头的数据，然后通过 `for` 循环把后面的数据批量向前移动一位。

复杂度：时间复杂度  $T(n) = O(n)$

空间复杂度  $S(n) = O(1)$

(24) `destroyQueue(Queue * q)`

清空队列里所有的数据，因为多个数据域皆为连续存储区域，所以连续地调用 `free` 函数即可。

复杂度：时间复杂度  $T(n) = O(1)$

空间复杂度  $S(n) = O(1)$

以上算法函数省略说明了其对应的真正递归调用函数，函数声明为其递归函数的再一层包装。不重复说明包装函数和逻辑抽离函数。

## 4.3 无向图演示系统实现与测试

### 4.3.1 系统实现

菜单界面：

```
----Menu for Undirected and Unweighted Graph----

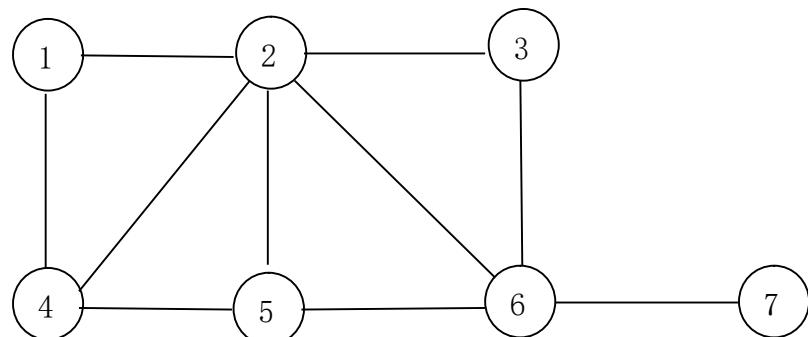
1. CreateEmptyGraph    2. DestoryGraph
3. LocateVex           4. GetVex
5. PutVex              6. FirstAdjVex
7. NextAdjVex          8. InsertVex
9. DeleteVex           10. InsertArc
11. DeleteArc          12. DFSTraverse
13. BFSTraverse        0. Exit

-----
Please select your option [0~13]:|
```

程序源代码：见附录 4

### 4.3.2 测试结果

测试用例为



(1) `CreateEmptyGraph(Graph ** gP)`

----Menu for Undirected and Unweighted Graph----

- |                     |                 |
|---------------------|-----------------|
| 1. CreateEmptyGraph | 2. DestoryGraph |
| 3. LocateVex        | 4. GetVex       |
| 5. PutVex           | 6. FirstAdjVex  |
| 7. NextAdjVex       | 8. InsertVex    |
| 9. DeleteVex        | 10. InsertArc   |
| 11. DeleteArc       | 12. DFSTraverse |
| 13. BFSTraverse     | 0. Exit         |

Please select your option [0~13]:1

Successful!

请按任意键继续. . .

可见成功地创建了一个空的无向图

(2) DestoryGraph(Graph \* g)

----Menu for Undirected and Unweighted Graph----

- |                     |                 |
|---------------------|-----------------|
| 1. CreateEmptyGraph | 2. DestoryGraph |
| 3. LocateVex        | 4. GetVex       |
| 5. PutVex           | 6. FirstAdjVex  |
| 7. NextAdjVex       | 8. InsertVex    |
| 9. DeleteVex        | 10. InsertArc   |
| 11. DeleteArc       | 12. DFSTraverse |
| 13. BFSTraverse     | 0. Exit         |

-----Vertexs' Key List-----

1,2,3,

Please select your option [0~13]:2

Successful!

请按任意键继续. . .

可见成功地删除了整个图。

(3) LocateVex(Graph \* g, ElemType data)

```
----Menu for Undirected and Unweighted Graph----  
1. CreateEmptyGraph    2. DestoryGraph  
3. LocateVex           4. GetVex  
5. PutVex              6. FirstAdjVex  
7. NextAdjVex          8. InsertVex  
9. DeleteVex           10. InsertArc  
11. DeleteArc          12. DFSTraverse  
13. BFSTraverse        0. Exit  
  
-----Vertexs' Key List-----  
1,2,3,4,5,6,7,  
-----  
Please select your option [0~13]:3  
  
Please input "data" of this vertex:7  
data of this vertex is 7.  
请按任意键继续. . .
```

可见成功地输出了数据为 7 的结点 key 值 7。

(4) GetVex(Graph \* g, ElemType loc)

```
-----Vertexs' Key List-----  
1,2,3,4,5,6,7,  
-----  
Please select your option [0~13]:4  
  
Please input "key" of this vertex:7  
key of this vertex is 7.  
请按任意键继续. . .
```

可见可以正常通过 loc 值获得结点的数据值。

(5) PutVex(Graph \* g, Vertex \* v, ElemType value)

```
-----Vertexs' Key List-----  
1,2,3,4,5,6,7,  
-----  
Please select your option [0~13]:5  
  
Please input "key" of this vertex:1  
  
Please input a new value for this vertex:10  
请按任意键继续. . .
```

```
Please select your option [0~13]:4  
  
Please input "key" of this vertex:1  
key of this vertex is 10.  
请按任意键继续. . .
```

可见正常的进行了重新赋值。

(6) FirstAdjVex(Graph \* g, Vertex \* v)

```
-----Vertexs' Key List-----
1,2,3,4,5,6,7,
-----
Please select your option [0~13]:6

Please input "key" of this vertex:1
the first adj-vex: key=2 value=2
请按任意键继续. . . |
```

可见可以正常获得邻接表里 v 的第一个邻接顶点的键值。

(7) NextAdjVex(Graph \* g, Vertex \* v, Vertex \* w)

```
-----Vertexs' Key List-----
1,2,3,4,5,6,7,
-----
Please select your option [0~13]:7

Please input "key" of this vertex:1

Please input "key" of adj-vex of the prior vertex:2
next-adj-vex: key=4 value=4请按任意键继续. . .
```

可见可以正常获得 v 起始 w 方向的下一个顶点的键值。

(8) InsertVex(Graph \* g, Vertex \* v)

```
-----
Please select your option [0~13]:8
Please input a new key of this vertex:8
Please input data of this vertex:8
Successful!
请按任意键继续. . . |
```

可见正常插入了键值为 8,8 的新顶点。

(9) DeleteVex(Graph \* g, Vertex \* v)

----Menu for Undirected and Unweighted Graph----

- |                     |                 |
|---------------------|-----------------|
| 1. CreateEmptyGraph | 2. DestoryGraph |
| 3. LocateVex        | 4. GetVex       |
| 5. PutVex           | 6. FirstAdjVex  |
| 7. NextAdjVex       | 8. InsertVex    |
| 9. DeleteVex        | 10. InsertArc   |
| 11. DeleteArc       | 12. DFSTraverse |
| 13. BFSTraverse     | 0. Exit         |

-----Vertexs' Key List-----

1,2,3,4,5,6,7,8,

Please select your option [0~13]:9

Please input "key" of this vertex:8

Successful!

请按任意键继续. . . |

可见成功删除了顶点。

(10) InsertArc(Graph \* g, Vertex \* v, Vertex \* w)

-----Vertexs' Key List-----

1,2,3,4,5,6,7,8,

Please select your option [0~13]:10

Please input "key" of two vertexs( splited by space ):6 8

Successful!

请按任意键继续. . .

可见成功的插入了一条以 6、8 为顶点的新边。

(11) DeleteArc(Graph \* g, Vertex \* v, Vertex \* w)

-----Vertexs' Key List-----

1,2,3,4,5,6,7,

Please select your option [0~13]:11

Please input "key" of two vertexs( splited by space ):6

7

Successful!

请按任意键继续. . .

```
----DFS starts.----  
key=1 value=1  
key=2 value=2  
key=3 value=3  
key=6 value=6  
key=5 value=5  
key=4 value=4  
----DFS end.----
```

可见完成了一次正确的删边操作

(12) DFSTraverse(Graph \* g)

```
-----Vertexs' Key List-----  
1,2,3,4,5,6,7,  
-----  
Please select your option [0~13]:12  
----DFS starts.----  
key=1 value=1  
key=2 value=2  
key=3 value=3  
key=6 value=6  
key=5 value=5  
key=4 value=4  
key=7 value=7  
----DFS end.----  
请按任意键继续. . . |
```

可见完成了一次正确的深度优先遍历。

(13) BFSTraverse(Graph \* g)

```
-----Vertexs' Key List-----  
1,2,3,4,5,6,7,  
-----  
Please select your option [0~13]:13  
----BFS starts.----  
key=1 value=1  
key=2 value=2  
key=6 value=6  
key=3 value=3  
key=4 value=4  
key=5 value=5  
key=7 value=7  
----BFS end.----  
请按任意键继续. . . |
```

可见进行了一次正常的广度优先遍历。

## 4.4 实验小结

### 3.4.1 实验问题

本次实验中，由于采用了邻接表来进行物理结构来保存图的顶点边关系，提高了操作的复杂程度，在开发时遇到了不少问题，尤其是在维护拉出的链接链表的时候，经常出现指针访问的问题，经过了多次断点调试过程才逐步找到了问题并解决。

### 3.4.2 特色

图的广序优先遍历依赖的队列是一个手动维护的顺序表，有机结合了第一次实验中的顺序表的优点，提高了空间的利用率。

### 3.4.3 实验小结

本次实验并不是很完善，尤其是只实现了无向无权图这一种图，不过减少了维护边集的开销，同时也没有用一个比较优雅的方式在磁盘中保存下图的顶点及边的关系。

## 参考资料

1. 《数据结构 (C 语言版)》 严蔚敏 吴伟民 版
2. 《Algorithms Fourth Edition》 Robert Sedgewick & Kevin Wayne

## 附录

1. 顺序表源代码：

```

SequenceList.c

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
typedef int ElemType; // make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define LIST_INIT_SIZE 100 // init length
#define LISTINCREMENT 10 //step-length when growing and cutting
#define LISTGAP 15 //redundant length needed to decide cutting list or not

typedef struct {
    ElemType * elem;
    int length;
    int listsiz;
    char * name; //length = 30
} SqList;

Status InitList(SqList ** L);
void DestoryList(SqList * L);
void 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_e, ElemType * pre_e);
Status NextElem(SqList * L, ElemType * cur_e, ElemType * next_e);
Status ListInsert(SqList * L, int i, ElemType * e);
Status ListDelete(SqList * L, int i, ElemType * e);
Status ListTraverse(SqList * L);
void growList(SqList * L);
void cutList(SqList * L);
Status SaveList(SqList * L);
Status LoadList(SqList ** L, char * name);
char *fgetsNoN(char *buf, int bufsize, FILE *stream);

//main page
int main() {

```

```

printf("Hello world!\n");
SqList * L = NULL;
int op = 1;
while (op) {
    system("cls"); printf("\n");
    printf("      Menu for Linear Table On Sequence Structure \n");
    if (L != NULL) {
        printf("  List:%s  is under your control .\n", L->name);
    }
    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. ListTraverse\n");
    printf("      6. GetElem       13. SaveList\n");
    printf("      7. LocateElem   14. LoadList\n");
    printf("      0. Exit\n");
    printf("-----\n");
    printf("      Please select your option [0~12]:");
    scanf("%d%c", &op);

    int pos = 0;
    ElemtType * cur = (ElemtType *)malloc(sizeof(ElemtType));

    switch (op) {
        case 1://InitList
            printf("\nPlease input the name of new list.\n");
            char * name = (char *)malloc(30 * sizeof(char));
            fgetsNoN(name, 30, stdin);
            if (InitList(&L) == OK) printf("Linear Table has been
created\n");
            else printf("\nFailed! Because there is no more space in stake's
memory, we can't creat a new table.\n");
            L->name = name;
            break;
        case 2://DestoryList
            if (L == NULL) printf("You should init or load first!\n");
            else DestoryList(L);L=NULL;
            break;
        case 3://ClearList
            if (L == NULL) printf("You should init or load first!\n");
            else ClearList(L);
    }
}

```

```

        break;
case 4://ListEmpty
    if (L == NULL) printf("You should init or load first!\n");
    else {
        if (ListEmpty(L)) {
            printf("\n----This list is empty.\n");
        } else {
            printf("\n----This list is not empty.\n");
        }
    }
    break;
case 5://ListLength
    if (L == NULL) printf("You should init or load first!\n");
    else printf("\n----The length of this list is %d.\n",
ListLength(L));
    break;
case 6://GetElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the element's position.\n");
        scanf("%d%c", &pos);

        ElemType * get = (ElemType *)malloc(sizeof(ElemType));
        if (GetElem(L, pos, get)) {
            printf("\n----The element you get is %d.\n", *get);
        } else {
            printf("\n----Failed! The position you input is out of
bound.\n");
        }
        free(get);
    }
    break;
case 7://LocateElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the model element.\n");
        scanf("%d%c", cur);
        pos = LocateElem(L, cur); pos++;
        if (pos != 0) {
            printf("\n----The index of the first element equal with
yours is %d.\n", pos);
        } else {
            printf("\n----Failed! It seems like that we don't have
such element.\n");
        }
    }
}

```

```

        }
    }
    break;
case 8://PriorElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the beginning element:");
        scanf("%d%c", cur);

        int pos = LocateElem(L, cur);
        if (pos != -1) {
            ElemType * get = (ElemType
*)malloc(sizeof(ElemType));
            if (PriorElem(L, cur, get)) {
                printf("\n----The prior element you get is %d.\n",
*get);
            } else {
                printf("\n----Failed! The element you select don't
have a prior element.\n");
            }
            free(get);
        } else {
            printf("\n----Failed! We can't find such a element.\n");
        }
    }

}
break;
case 9://NextElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the beginning element:");
        scanf("%d%c", cur);

        int pos = LocateElem(L, cur);
        if (pos != -1) {
            ElemType * get = (ElemType
*)malloc(sizeof(ElemType));
            if (NextElem(L, cur, get)) {
                printf("\n----The next element you get is %d.\n",
*get);
            } else {
                printf("\n----Failed! The element you select don't
have a next element.\n");
            }
        }
    }
}

```

```

        free(get);
    } else {
        printf("\n----Failed! We can't find such a element.\n");
    }
}
break;
case 10://ListInsert
if (L == NULL) printf("You should init or load first!\n");
else {
    printf("\n---Please input the position:");
    scanf("%d%c", &pos);
    printf("\n---Please input the element:");
    ElemtType * elemNew = (ElemtType
*)malloc(sizeof(ElemtType));
    scanf("%d%c", elemNew);

    if (ListInsert(L, pos, elemNew)) {
        printf("\n----Successful!\n");
    } else {
        printf("\n----Failed! The position you input is out of
bound.\n");
    }
    free(elemNew);
}
break;
case 11://ListDelete
if (L == NULL) printf("You should init or load first!\n");
else {
    printf("\n---Please input the position:");
    scanf("%d%c", &pos);
    ElemtType * elem = (ElemtType
*)malloc(sizeof(ElemtType));

    if (ListDelete(L, pos, elem)){
        printf("\n---Successfully delete element:%d!\n",
*elem);
    } else{
        printf("\n---Failed!\n");
    }
}

free(elem);
}
break;
case 12://ListTraverse

```

```

        if (L == NULL) printf("You should init or load first!\n");
        else if (!ListTraverse(L)) printf("\nThis list is empty!\n");
        break;
    case 13://SaveList
        if (L == NULL) printf("You should init or load first!\n");
        else {
            if (SaveList(L)) {
                printf("Successful!\n ");
            } else {
                printf("We can't open this file.\n ");
            }
        }
        break;
    case 14://LoadList
        printf("\n---Please input list's name:");
        char * nameIn = (char *)malloc(30 * sizeof(char));
        fgetsNoN(nameIn, 30, stdin);
        if (LoadList(&L, nameIn)) {
            printf("\n---Successfully load!\n");
        } else {
            printf("\n---Loading failed!\n");
        }
        break;
    case 0://Exit
        break;
    }
    free(cur);
//    Sleep(1000);
    system("pause");
}
printf("See you!\n");
return 0;
}

//save a list to disk
Status SaveList(SqList * L) {
    FILE * fP = fopen(L->name, "wb");
    if (fP != NULL) {
        fwrite(L->elem, sizeof(ElemType), L->length, fP);
        fclose(fP);
        return TRUE;
    } else {
        printf("File open error\n ");
        fclose(fP);
    }
}

```

```

        return FALSE;
    }
}

//load a list from disk, and the sqlist pointer will be the parameter-Lp
Status LoadList(SqList ** Lp, char * name) {
    SqList * L = *Lp;
    FILE * fP = fopen(name, "rb");

    if (fP == NULL) {
        return FALSE;
        printf("File open error\n ");
        fclose(fP);
    }

    if (L != NULL) {
        DestoryList(L);
        *Lp = NULL;
    }
    if (InitList(Lp) != OK) return FALSE;
    L = *Lp;
    L->name = name;

    while (fread(&(L->elem[L->length]), sizeof(ElemType), 1, fP)) {
        L->length++;
        if (L->length == L->listsize) {
            growList(L);
        }
    }
    fclose(fP);
    return OK;
}

//initialize this list, give it space and make it empty
Status InitList(SqList ** Lp) {
    *Lp = (SqList *)malloc(sizeof(SqList));
    SqList * L = *Lp;
    L->elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));

    if (!L->elem) {
        exit(0);
    }
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
}

```

```

        return OK;
    }

//grow this list's space
void growList(SqList * L) {
    L->elem    =  (ElemType    *)realloc(L->elem,    L->listsize    +
LISTINCREMENT * sizeof(ElemType));
    L->listsize += LISTINCREMENT;
    if (L->elem == NULL) exit(0);
}

void cutList(SqList * L) {//Pay attention to the opportunity to avoid
frequently cutting and growing
    L->elem    =  (ElemType    *)realloc(L->elem,    L->listsize    -
LISTINCREMENT * sizeof(ElemType));
    L->listsize -= LISTINCREMENT;
    if (L->elem == NULL) exit(0);
}

//Destory the whole list with its space
void DestoryList(SqList * L) {
    free(L->elem);
    free(L->name);
    free(L);
}

//clear the data of this list
void ClearList(SqList * L) {
    int length = L->length;
    int i;
    for (i = 0; i < length; i++) {
        L->elem[i] = 0;
    }
    L->length = 0;
}

//empty->TRUE not-empty->FALSE
Status ListEmpty(SqList * L) {
    if (L->length == 0) {
        return TRUE;
    } else {
        return FALSE;
    }
}

```

```

//return the length of this list
int ListLength(SqList * L) {
    return L->length;
}

//get element by index-i
Status GetElem(SqList * L, int i, ElemType * e) {
    i--;
    if (i < L->length) {
        *e = L->elem[i];
        return TRUE;
    } else { //IndexOutOfBoundsException
        return FALSE;
    }
}

//return the prior item of parameter-cur_e
Status PriorElem(SqList * L, ElemType * cur_e, ElemType * pre_e) {
    int length = L->length;
    int i = 0;
    *pre_e = 0;
    for (i = 0; i < length; i++) {
        if (L->elem[i] == *cur_e) {
            if (i >= 1) {
                *pre_e = L->elem[i - 1];
                return TRUE;
            }
        }
    }
    return FALSE;
}

//return the prior item of parameter-cur_e
Status NextElem(SqList * L, ElemType * cur_e, ElemType * next_e) {
    int length = L->length;
    int i = 0;
    *next_e = 0;
    for (i = 0; i < length; i++) {
        if (L->elem[i] == *cur_e) {
            if (i + 1 < length) {
                *next_e = L->elem[i + 1];
                return TRUE;
            }
        }
    }
}

```

```

        }
    }
    return FALSE;
}

//insert a element into this list
Status ListInsert(Sqlist * L, int i, ElemType * e) {
    i--;
    int index = L->length;
    if (i > index) {//IndexOutOfBoundsException
        return FALSE;
    }
    if (L->length == L->listsize) {
        growList(L);
    }
    ElemtType * head = L->elem;
    for (index = L->length; index > i; index--) {
        head[index] = head[index - 1];
    }
    head[i] = *e;
    L->length++;
    return TRUE;
}

//delete a element from this list
Status ListDelete(Sqlist * L, int i, ElemtType * e) {
    if (i >= L->length){
        *e = 0;
        return FALSE;
    }
    int temp = i;
    int length = L->length;
    ElemtType * head = L->elem;
    *e = head[temp];
    for (temp = i; temp < length - 1; temp++) {
        head[temp] = head[temp + 1];
    }
    head[length - 1] = 0;//reset this value
    L->length--;
    if (15 <= L->listsize - length) {//cut list
        cutList(L);
    }
    return TRUE;
}

```

```

//traverse this list and output all element
Status ListTraverse(SqList * L) {
    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;//return the length of list
}

//return a element's index of list-l
int LocateElem(SqList * L, ElemType * e) {
    int i = 0;
    ElemType * head = L->elem;
    for (i = 0; i < L->length; i++) {
        if (*e == head[i]) {
            return i;
        }
    }
    return -1;
}

//Override fgets, so that we can get a string in inputstream without '\n'
char *fgetsNoN(char *buf, int bufsize, FILE *stream) {
    char * returnP = fgets(buf, bufsize, stream);
    int i = 0;
    while (buf[i] != '\n') {
        i++;
    }
    buf[i] = '\0';
    return returnP;
}

```

## 2. 链式表源代码:

LinkedList.c

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1

```

```

//this is the struct taking the ture data
typedef struct Lnode {
    ElemtType data;
    struct Lnode * next;
} LNode;

//this is the struct as the head node taking some infomation of this linked list
typedef struct sqlist {
    int length;
    char * name;
    LNode * head;
} SqList;

Status InitList(SqList ** L);
void DestoryList(SqList * L);
void ClearList(SqList * L);
Status ListEmpty(SqList * L);
int ListLength(SqList * L);
Status GetElem(SqList * L, int i, ElemtType * e);
int LocateElem(SqList * L, ElemtType * e);
Status PriorElem(SqList * L, ElemtType * cur_e, ElemtType * pre_e);
Status NextElem(SqList * L, ElemtType * cur_e, ElemtType * next_e);
Status ListInsert(SqList * L, int i, ElemtType * e);
Status ListDelete(SqList * L, int i, ElemtType * e);
Status ListTraverse(SqList * L);
Status SaveList(SqList * L);
Status LoadList(SqList ** L, char * name);
char * fgetsNoN(char *buf, int bufsize, FILE *stream);

int main() {
    SqList * L = NULL;
    int op = 1;
    while (op) {
        system("cls"); printf("\n");
        printf("      Menu for Linear Table On Sequence Structure \n");
        if (L != NULL) {
            printf("  List:%s  is under your control .\n", L->name);
        }
        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. ListTraverse\n");
printf("      6. GetElem        13. SaveList\n");
printf("      7. LocateElem     14. LoadList\n");
printf("      0. Exit\n");
printf("-----\n");
printf("Please select your option [0~12]:");
scanf("%d%c", &op);

int pos = 0;
ElemType * cur = (ElemType *)malloc(sizeof(ElemType));

switch (op) {
case 1://InitList
    printf("\nPlease input the name of new list.\n");
    char * name = (char *)malloc(30 * sizeof(char));
fgetsNoN(name, 30, stdin);

    if (InitList(&L) == OK) printf("Linear Table has been
created!\n");
    else printf("\nFailed! Because there is no more space in stake's
memory, we can't creat a new table.\n");
    L->name = name;
    break;
case 2://DestoryList
    if (L == NULL) printf("You should init or load first!\n");
    else DestoryList(L); L = NULL;
    break;
case 3://ClearList
    if (L == NULL) printf("You should init or load first!\n");
    else ClearList(L);
    break;
case 4://ListEmpty
    if (L == NULL) printf("You should init or load first!\n");
    else {
        if (ListEmpty(L)) {
            printf("\n----This list is empty.\n");
        } else {
            printf("\n----This list is not empty.\n");
        }
    }
    break;
case 5://ListLength
    if (L == NULL) printf("You should init or load first!\n");
    else printf("\n----The length of this list is %d.\n",

```

```

ListLength(L));
    break;
case 6://GetElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the element's position.\n");
        scanf("%d%c", &pos);

        ElemType * get = (ElemType *)malloc(sizeof(ElemType));
        if (GetElem(L, pos, get)) {
            printf("\n----The element you get is %d.\n", *get);
        } else {
            printf("\n----Failed! The position you input is out of
bound.\n");
        }
        free(get);
    }
    break;
case 7://LocateElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the model element.\n");
        scanf("%d%c", cur);
        pos = LocateElem(L, cur);
        if (pos != 0) {
            printf("\n----The index of the first element equal with
yours is %d.\n", pos);
        } else {
            printf("\n----Failed! It seems like that we don't have
such element.\n");
        }
    }
    break;
case 8://PriorElem
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\nPlease input the beginning element:");
        scanf("%d%c", cur);

        int pos = LocateElem(L, cur);
        if (pos != -1) {
            ElemType      *      get      =      (ElemType
*)malloc(sizeof(ElemType));
            if (PriorElem(L, cur, get)) {

```

```

        printf("\n----The prior element you get is %d.\n",
*get);
    } else {
        printf("\n----Failed! The element you select don't
have a prior element.\n");
    }
    free(get);
} else {
    printf("\n----Failed! We can't find such a element.\n");
}
}

break;
case 9://NextElem
if (L == NULL) printf("You should init or load first!\n");
else {
    printf("\nPlease input the beginning element:");
    scanf("%d%c", cur);

    int pos = LocateElem(L, cur);
    if (pos != -1) {
        ElemtType * get = (ElemtType
*)malloc(sizeof(ElemtType));
        if (NextElem(L, cur, get)) {
            printf("\n----The next element you get is %d.\n",
*get);
        } else {
            printf("\n----Failed! The element you select don't
have a next element.\n");
        }
        free(get);
    } else {
        printf("\n----Failed! We can't find such a element.\n");
    }
}
break;
case 10://ListInsert
if (L == NULL) printf("You should init or load first!\n");
else {
    printf("\n---Please input the position:");
    scanf("%d%c", &pos);
    printf("\n---Please input the element:");
    ElemtType * elemNew = (ElemtType
*)malloc(sizeof(ElemtType));
}

```

```

        scanf("%d%c", elemNew);

        if (ListInsert(L, pos, elemNew)) {
            printf("\n----Successful!\n");
        } else {
            printf("\n----Failed! The position you input is out of
bound.\n");
        }
        free(elemNew);
    }
    break;
case 11://ListDelete
    if (L == NULL) printf("You should init or load first!\n");
    else {
        printf("\n---Please input the position:");
        scanf("%d%c", &pos);
        ElemtType * elem = (ElemtType
*)malloc(sizeof(ElemtType));
        if (ListDelete(L, pos, elem)) {
            printf("\n----Successfully delete element\n");
        } else {
            printf("\n----Failed! The position you input is out of
bound.\n");
        }
        free(elem);
    }
    break;
case 12://ListTraverse
    if (L == NULL) printf("You should init or load first!\n");
    else if (!ListTraverse(L)) printf("\nThis list is empty!\n");
    break;
case 13://SaveList
    if (L == NULL) printf("You should init or load first!\n");
    else {
        if (SaveList(L)) {
            printf("Successful!\n ");
        } else {
            printf("We can't open this file.\n ");
        }
    }
    break;
case 14://LoadList
    printf("\n---Please input list's name:");
    char * nameIn = (char *)malloc(30 * sizeof(char));

```

```

fgetsNoN(nameIn, 30, stdin);
if (LoadList(&L, nameIn)) {
    printf("\n---Successfully load!\n");
} else {
    printf("\n---Loading failed!\n");
}
break;
case 0://Exit
break;
}
free(cur);
//Sleep(1000);
system("pause");
}
printf("See you!\n");
return 0;
}

//save a list to disk
Status SaveList(SqList * L) {
FILE * fP = fopen(L->name, "wb");
if (fP != NULL) {
    LNode * p = L->head;
    while (p != NULL) {
        fwrite(&(p->data), sizeof(ElemType), 1, fP);
        p = p->next;
    }
    fclose(fP);
    return TRUE;
} else {
    printf("File open error\n ");
    fclose(fP);
    return FALSE;
}
}

//load a list from disk, and the sqlist pointer will be the parameter-Lp
Status LoadList(SqList ** Lp, char * name) {
SqList * L = *Lp;
if (L != NULL) {
    DestoryList(L);
    L = NULL;
}
if (InitList(Lp) != OK) return FALSE;

```

```

L = *Lp;
L->name = name;
FILE * fP = fopen(L->name, "rb");
if (fP != NULL) {
    LNode * newP = (LNode *)malloc(sizeof(LNode));
    LNode * p = newP;
    int flag = 1;

    while (fread(&(newP->data), sizeof(ElemType), 1, fP)) {
        L->length++;
        if (flag) {
            L->head = newP;
            flag = 0;
        } else {
            p->next = newP;
        }
        newP->next = NULL;
        p = newP;
        newP = (LNode *)malloc(sizeof(LNode));
    }
    free(newP); //because the last p is useful
    fclose(fP);

    return TRUE;
} else {
    printf("File open error\n ");
    return FALSE;
}
}

//initialize this list, give it space and make it empty
Status InitList(SqList ** Lp) {
    *Lp = (SqList *)malloc(sizeof(SqList));

    SqList * L = *Lp;
    L->length = 0;
    L->head = NULL;
    //the name part space will be spaced by outside
    return OK;
}

//destory the whole list, free all space it takes
void DestoryList(SqList * L) {
    ClearList(L);
}

```

```

        free(L->name);
        free(L);
    }

//clear the data of this list
void ClearList(SqList * L) {
    LNode * p = L->head;
    LNode * pNext = p;
    if (p == NULL) return;
    //free all data in this list
    while (p->next != NULL) {
        pNext = p->next;
        free(p);
        p = pNext;
    }
    free(p);
    L->length = 0;
    L->head = NULL;
}

//empty->TRUE not-empty->FALSE
Status ListEmpty(SqList * L) {
    if (L->length == 0) {
        return TRUE;
    } else {
        return FALSE;
    }
}

//return the length of this list
int ListLength(SqList * L) {
    return L->length;
}

//get element by index-i
Status GetElem(SqList * L, int i, ElemType * e) {
    if (i > L->length || i < 1) {
        return FALSE;
    } //Index out of bound exception

    int count = 0;//this count variable is according to the usual
    LNode * p = L->head;
    while (p != NULL) {
        count++;

```

```

        if (count == i) {
            *e = p->data;
            return OK;
        }
        p = p->next;
    }
    return FALSE;
}

//return the prior item of parameter-cur_e
Status PriorElem(SqList * L, ElemType * cur_e, ElemType * pre_e) {
    LNode * p = L->head;
    while (p->next != NULL) {

        if (p->next->data == *cur_e) {//if the condition is true, then the
point now is the prior one
            *pre_e = p->data;
            return OK;
        }
        p = p->next;
    }
    return FALSE;//including two situation(no such elemType in this list or
cur_e is the head one)
}

//return the prior item of parameter-cur_e
Status NextElem(SqList * L, ElemType * cur_e, ElemType * next_e) {
    LNode * p = L->head;
    while (p != NULL) {
        if (p->data == *cur_e) {
            if (p->next != NULL) {
                *next_e = p->next->data;
                return OK;
            }
        }
        p = p->next;
    }
    return FALSE;//including two situation(no such elemType in this list or
cur_e is the last one)
}

//insert a element into this list
Status ListInsert(SqList * L, int i, ElemType * e) {

```

```

int count = 0;
LNode * p = L->head;
if (i>L->length + 1 || i<1) {
    return FALSE;
}
if (i == 1) {//insert to be the first one
    LNode * newP = (LNode *)malloc(sizeof(LNode));
    newP->data = *e;
    newP->next = L->head;//NULL or the old head
    L->head = newP;
    L->length++;
    return OK;
} else {
    while (p != NULL) {
        count++;//fit as usual
        if (count == i - 1) {//find the prior
            LNode * newP = (LNode *)malloc(sizeof(LNode));
            newP->data = *e;
            LNode * pNext = p->next;
            p->next = newP;
            newP->next = pNext;//exchange the next pointer
            L->length++;
            return OK;
        }
        p = p->next;
    }
}

return FALSE;
}

//delete a element from this list
Status ListDelete(SqList * L, int i, ElemType * e) {
    int count = 0;
    LNode * p = L->head;
    if (i>L->length || i<1) {
        return FALSE;
    }
    if (i == 1) {
        LNode * freeP = L->head;
        L->head = L->head->next;
        free(freeP);
    }
    while (p != NULL) {

```

```

        count++;//fit as usual
        if (count == i - 1) {//find the prior ( p->next would never be null
            LNode * freeP = p->next;
            p->next = p->next->next;
            free(freeP);
            L->length--;
            return OK;
        }

        p = p->next;
    }
    return FALSE;
}

//traverse this list and output all element
Status ListTraverse(SqList * L) {
    printf("\n-----all elements ----- \n");
    LNode * p = L->head;
    while (p != NULL) {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n----- end ----- \n");
    return L->length;//return the length of list
}

//return a element's index of list-l
int LocateElem(SqList * L, ElemType * e) {
    int count = 0;
    LNode * p = L->head;
    while (p != NULL) {
        count++;
        if (p->data == *e) return count;
        p = p->next;
    }
    return -1;
}

//Override fgets, so that we can get a string in inputstream without '\n'
char *fgetsNoN(char *buf, int bufsize, FILE *stream) {
    char * returnP = fgets(buf, bufsize, stream);
    int i = 0;
    while (buf[i] != '\n') {

```

```

        i++;
    }
    buf[i] = '\0';
    return returnP;
}

```

### 3. 二叉树源代码:

TreeMain.c

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <TreeUtil.h>
typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define ARRAY_SIZE 10

Tree ** treeArray = NULL;//init an array for tree array.
int indexTreeArray = 0;//index of empty position for the array above
char *fgetsNoN(char *buf, int bufsize, FILE *stream);
Tree * getTreeFromName(char * name);

//initialize tree array
void initTreeArray() {
    int i = 0;
    for (i = 0; i < ARRAY_SIZE; i++) {
        treeArray[i] = NULL;
    }
}
//move this tree to the end of tree array so we can easily remove it
//if you try to call this func , make sure tree-T has already been in tree array
void removeTreeFromArray(Tree * T) {

    int index = 0;
    for (index = 0; index < indexTreeArray; index++) {
        if (treeArray[index] == T) {
            break;
        }
    }
    //now "index" points at Tree-T
    if (index == indexTreeArray - 1) {//at the end
        treeArray[indexTreeArray - 1] = NULL;
    }
}

```

```

    } else {
        Tree * tail = treeArray[indexTreeArray - 1];
        treeArray[indexTreeArray - 1] = NULL;
        treeArray[index] = tail;
    }
    indexTreeArray--;
}

//as this func's name... print guiding infomation, input and free
Tree * getTree() {
    Tree * T = NULL;
    printf("\nPlease input the name of the tree you want to control:");
    char * name = (char *)malloc(30 * sizeof(char)); fgetsNoN(name, 30,
stdin);
    T = getTreeFromName(name);
    free(name);
    return T;
}
//a logistic func to low down lines of code
Node * getKeyNode(Tree * T) {
    printf("\nPlease input the key of this node:");
    ELEMType key = 0; scanf("%d%c", &key);
    return FindNode(T, key);
}

//InitBiTree
void IBT1() {
    Tree * T = NULL;
    printf("\nPlease input the name of new list:");
    char * name = (char *)malloc(30 * sizeof(char)); fgetsNoN(name, 30,
stdin);
    if (InitBiTree(&T)) {
        printf("Binary Tree has been created\n");
        T->name = name;
        treeArray[indexTreeArray] = T; indexTreeArray++;
    } else { printf("\nFailed! Because there is no more space in stake's
memory, we can't creat a new tree.\n"); }
}

//Assign
void A2() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {

```

```

        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            printf("\nPlease input the new value of this node:");
            ElemType value = 0; scanf("%d%c", &value);
            Assign(T, get, value); printf("\nSuccessful\n");
        }
    }

//Destory
void DT3() {
    Tree * T = getTree();
    if (T) {
        removeTreeFromArray(T);
        DestroyTree(T);
        printf("\nDone!\n");
    } else {
        printERROR(2);
    }
}

//Parent
void P4() {
    Tree * T = getTree();
    if (T) {//not null
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);//not null
        else {
            Node * parent = Parent(T, get);
            if (parent) printf("the parent node: key=%d value=%d",
parent->key, parent->data);
            else printf("no parent-node.\n");

        }
    } else {
        printERROR(2);
    }
}

//ClearBiTree
void CBT5() {
    Tree * T = getTree();
    if (T) {

```

```

        ClearBiTree(T);//just clear all data, but still keep its position
        printf("\nDone!\n");
    } else {
        printERROR(2);
    }
}

//LeftChild
void LC6() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            Node * leftChild = LeftChild(T, get);
            if (!leftChild) printf("this node don't have a left child.\n");
            else printf("its left child: key=%d, value=%d.\n", leftChild->key,
leftChild->data);
        }
    }
}

//BiTreeEmpty
void BTE7() {
    Tree * T = getTree();
    if (T) {
        if (BiTreeEmpty(T)) {
            printf("\nIt's not empty\n");
        } else {
            printf("\nIt's empty\n");
        }
    } else {
        printERROR(2);
    }
}

//RightChild
void RC8() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);

```

```

        else {
            Node * rightChild = RightChild(T, get);
            if (!rightChild) printf("this node don't have a right child.\n");
            else printf("its right child: key=%d, value=%d.\n",
rightChild->key, rightChild->data);
        }
    }
}

//get tree's depth
void BTD9() {
    Tree * T = getTree();
    if (T) {
        int d = BiTreeDepth(T);
        printf("\nIt's depth is %d.\n", d);
    } else {
        printERROR(2);
    }
}

//LeftSibling
void LS10() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            Node * leftSibling = LeftSibling(T, get);
            if (!leftSibling) printf("this node don't have a left sibling.\n");
            else {
                printf("its left sibling: key=%d value=%d\n",
leftSibling->key, leftSibling->data);
            }
        }
    }
}

//get root node
void R11() {
    Tree * T = getTree();
    if (T) {
        Node * root = T->root;
        if (root) {

```

```

        printf("\nRoot node: key=%d value=%d.\n", root->key,
root->data);
    } else {
        printf("\nThis tree is empty.\n");
    }
} else {
    printERROR(2);
}
}

//RightSibling
void RS12() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            Node * rightSibling = RightSibling(T, get);
            if (!rightSibling) printf("this node don't have a right sibling.\n");
            else {
                printf("its right sibling: key=%d value=%d\n",
rightSibling->key, rightSibling->data);
            }
        }
    }
}

//Value
void V13() {
    Tree * T = getTree();
    if (T) {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            printf("its value is %d", Value(T, get));
        }
    } else {
        printERROR(2);
    }
}

//InsertChild
void IC14() {

```

```

Tree * T = getTree();
if (!T) printERROR(2);
else {
    Node * get = getKeyNode(T);
    if (!get) printERROR(3);
    else {
        Tree * Tnew = NULL;
        printf("\nPlease input the name of the tree you would like to
choose to insert:");
        char * name = (char *)malloc(30 * sizeof(char));
        fgetsNoN(name, 30, stdin);
        Tnew = getTreeFromName(name);
        free(name);
        if (!Tnew) printf("this new tree don't exist.\n");
        else {
            printf("Please input 1/0 to choose direction ( 0:L 1:R ):");
            int LR = 0; scanf("%d%c", &LR);
            if (LR == 1 || LR == 0) {
                if (InsertChild(T, get, LR, Tnew)) {
                    removeTreeFromArray(Tnew);
                    printf("successful!\n");
                } else printf("the postion have been occupied.\n");
            } else {
                printf("wrong option!\n");
            }
        }
    }
}

//DeleteChild
void DC15() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        Node * get = getKeyNode(T);
        if (!get) printERROR(3);
        else {
            printf("Please choose to delete L or R of this node(L->0
R->1)\n");
            int LR = 0; scanf("%d%c", &LR);
            if (LR == 1 || LR == 0) {
                if (DeleteChild(T, get, LR)) {
                    printf("successful!\n");
                }
            }
        }
    }
}

```

```
        } else printf("Failed, the position don't have child.\n");
    } else {
        printf("error happened!\n");
    }
}
}

//PreOrderTraverse
void POT16() {
    Tree * T = getTree();
    if (T) {
        Node * root = T->root;
        if (root) {
            PreOrderTraverse(T);
        } else {
            printf("\nThis tree is empty.\n");
        }
    } else {
        printERROR(2);
    }
}

//InOrderTraverse
void IOT17() {
    Tree * T = getTree();
    if (T) {
        Node * root = T->root;
        if (root) {
            InOrderTraverse(T);
        } else {
            printf("\nThis tree is empty.\n");
        }
    } else {
        printERROR(2);
    }
}

//PostOrderTraverse
void POT18() {
    Tree * T = getTree();
    if (T) {
        Node * root = T->root;
```

```

if (root) {
    PostOrderTraverse(T);
} else {
    printf("\nThis tree is empty.\n");
}
} else {
    printERROR(2);
}
}

//LevelOrderTraverse
void LOT19() {
    Tree * T = getTree();
    if (T) {
        Node * root = T->root;
        if (root) {
            LevelOrderTraverse(T);
        } else {
            printf("\nThis tree is empty.\n");
        }
    } else {
        printERROR(2);
    }
}

//Load tree
void LT20() {
    printf("Please input the name of this tree:");
    char * name = (char*)malloc(sizeof(char) * 30); fgetsNoN(name, 30,
    stdin);
    Tree * T = LoadTree(name);
    if (T) {
        treeArray[indexTreeArray] = T; indexTreeArray++;
        printf("successful\n");
    } else {
        free(name);
        printf("there is not such a tree's name in your disk\n");
    }
}

//save tree
void ST21() {
    Tree * T = getTree();
    if (!T) printERROR(2);
}

```

```

    else {
        if (saveTree(T)) {
            printf("Successful!\n");
        } else {
            printf("Failed to open file\n");
        }
    }
}

//insert a new node to a given parent node
void IN22() {
    Tree * T = getTree();
    if (!T) printERROR(2);
    else {
        printf("Please input the key of new node:");
        ElemType key = 0; scanf("%d%c", &key);
        if (FindNode(T, key)) {
            printf("there has been a node with the same key\n");
            return;
        }
        printf("Please input the value of new node:");
        ElemType value = 0; scanf("%d%c", &value);
        Node * newNode = (Node*)malloc(sizeof(Node));
        newNode->key = key;
        newNode->data = value;
        newNode->left = NULL; newNode->right = NULL;
        if (!BiTreeEmpty(T)) {//it's empty now
            T->root = newNode;
            T->length++;
            printf("successful\n");
        } else {
            printf("Please input the key of parent of this node:");
            ElemType parentKey = 0; scanf("%d%c", &parentKey);
            Node * parentNode = FindNode(T, parentKey);
            if (!parentNode) { printf("there isn't such a node with this
key.\n"); return; }

            printf("Please choose left or right( 0:left 1:right ):");
            int LR = 0; scanf("%d%c", &LR);
            if (LR == 0 && parentNode->left == NULL) {
                parentNode->left = newNode;
                T->length++;
            } else if (LR == 1 && parentNode->right == NULL) {
                parentNode->right = newNode;
            }
        }
    }
}

```

```

        T->length++;
    }
}
}
}

//Quickly Create a binary tree with a pre-order array
void QCT23(){
    Tree * T = NULL;
    printf("\nPlease input the name of new list:");
    char * name = (char *)malloc(30 * sizeof(char)); fgetsNoN(name, 30,
stdin);

    T = (Tree*)malloc(sizeof(Tree));
    if(!T) exit(1);

    T->name = name;
    T->root = NULL;
    T->length = 0;

    printf("\nPlease input the pre-order-traverse( null-># ) data of new
list:");
    createTreeQuickly(&(T->root));
    printf("\nPlease input the pre-order-traverse key ( split by space ) of new
list:");
    valueKeyFromQuick(T->root);
    getchar();
    if (T->root) {
        printf("Binary Tree has been created\n");
        T->name = name;
        treeArray[indexTreeArray] = T; indexTreeArray++;
    } else { printERROR(1); }
}

//print the top-layer menu
void printMenu() {
    system("cls");    printf("\n");
    printf("          Menu for Binary Tree\n");
    printf("-----Trees List-----\n");
    int i = 0;
    for(i = 0; i < indexTreeArray; i++){
        printf("%s ,",treeArray[i]->name);
    }
    printf("\n-----\n");
    printf("      1. InitBiTree\t2. Assign\n");
}

```

```

printf("      3. DestroyTree\t\t4. Parent \n");
printf("      5. ClearBiTree\t\t6. LeftChild\n");
printf("      7. BiTreeEmpty\t\t8. RightChild\n");
printf("      9. BiTreeDepth\t\t10. LeftSibling\n");
printf("     11. Root\t\t12. RightSibling\n");
printf("     13. Value\t\t14. InsertChild\n");
printf("     15. DeleteChild\t\t16. PreOrderTraverse\n");
printf("     17. InOrderTraverse\t\t18. PostOrderTraverse\n");
printf("     19. LevelOrderTraverse\t\t20. LoadTree\n");
printf("     21. saveTree\t\t22. InsertNode\n");
printf("     23. QuickCreateTree\n");
printf("     0. Exit\n");
printf("-----\n");
}

int main() {

    treeArray = (Tree **)malloc(sizeof(Tree*) * ARRAY_SIZE);//init an
array for tree array.(max size is 10)
    initTreeArray();
    indexTreeArray = 0;//index for the array above
    int op = 1;
    while (op) {
        printMenu();//print the top-layer menu
        printf("      Please select your option [0~12]:");
        scanf("%d%c", &op);

        switch (op) {
        case 1://InitBiTree
            IBT1(); break;
        case 2://Assign
            A2(); break;
        case 3://Destroy
            DT3(); break;
        case 4://Parent
            P4(); break;
        case 5://ClearBiTree
            CBT5(); break;
        case 6://LeftChild
            LC6(); break;
        case 7://Empty
            BTE7(); break;
        case 8://RightChild
            RC8(); break;
        }
    }
}

```

```

case 9://Depth
    BTD9(); break;
case 10://LeftSibling
    LS10(); break;
case 11://Root
    R11(); break;
case 12://RightSibling
    RS12(); break;
case 13://get value
    V13(); break;
case 14://InsertChild
    IC14(); break;
case 15://DeleteChild
    DC15(); break;
case 16://PreOrderTraverse
    POT16(); break;
case 17://InOrderTraverse
    IOT17(); break;
case 18://PostOrderTraverse
    POT18(); break;
case 19://LevelOrderTraverse
    LOT19(); break;
case 20://LoadTree
    LT20();
    break;
case 21://Save tree
    ST21();
    break;
case 22://InsertNode
    IN22(); break;
case 23://create a binary tree in a fast way
    QCT23();break;
case 0:
    op = 0;
    break;
}
system("pause");
}
printf("\nSee you\n");
return 0;
}

char *fgetsNoN(char *buf, int bufsize, FILE *stream) {

```

```

char * returnP = fgets(buf, bufsize, stream);
int i = 0;
while (buf[i] != '\n') {
    i++;
}
buf[i] = '\0';
return returnP;
}

//get tree node
Tree * getTreeFromName(char * name) {
    int i = 0;
    for (i = 0; i < indexTreeArray; i++) {
        if (!strcmp(treeArray[i]->name, name)) {
            return treeArray[i];
        }
    }
    return NULL;
}

```

## TreeUtil.h

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <math.h>
#include <assert.h>

typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define NULLNODECODE 90017
#define LOAD_LIST_SIZE 31

```

```

typedef struct node {
    ElemType key;//a key to find this node
    ElemType data;
    struct node * left;
    struct node * right;
}Node;

```

```

typedef struct tree {
    int length;

```

```

char * name;

Node * root;
} Tree;

Status InitBiTree(Tree ** T);
void DestroyTree(Tree * T);
//Status CreateTree(Tree * T); //WTF for definition
void ClearBiTree(Tree * T);
Status BiTreeEmpty(Tree * T);
int BiTreeDepth(Tree * T);
Node * Root(Tree * T);
ElemType Value(Tree * T, Node * e);
Status Assign(Tree * T, Node * e, ElemType value);
Node * Parent(Tree * T, Node * e);
Node * LeftChild(Tree * T, Node * e);
Node * RightChild(Tree * T, Node * e);
Node * LeftSibling(Tree * T, Node * e);
Node * RightSibling(Tree * T, Node * e);
Status InsertChild(Tree * T, Node * p, int LR, Tree * c); //c is a new tree to
insert
Status DeleteChild(Tree * T, Node * p, int LR);
Node * FindNode(Tree * T, ElemType key);

Status PreOrderTraverse(Tree * T);
Status InOrderTraverse(Tree * T);
Status PostOrderTraverse(Tree * T);
Status LevelOrderTraverse(Tree * T);

Tree * LoadTree(char * TreeName);
Status saveTree(Tree * T);
void LevelOreder_recurve(Node * node, int depthNow);
void BiTreeDepth_recurve(Node * node, int i);
Status InsertNode(Tree * T, Node * p, int LR, Node * c);
void PostOrderTraverse_recurve(Node * n);
void InOrderTraverse_recurve(Node * n);
void PreOrderTraverse_recurve(Node * n);
int Parent_recurve(Node * start, Node * key, Node ** parent);
void ClearBiTree_recurve(Node * n);
void printERROR(int code); //print error information
int FindNode_recurve(Node * node, ElemType key, Node ** result);
void saveTree_recurve(Node * node, int i);
Node * LoadTree_recurve(int i);
int BiTreeLength(Node * node);

```

```

/*
    Quickly Create a binary tree with a pre-order array
    when calling this function outside, parameter "node" must be root-node.
*/
void createTreeQuickly(Node** nodeP) {
    Node * node = *nodeP;
    char * temp = (char*)malloc(sizeof(char)*10);
    scanf("%s", temp);

    if (!strcmp(temp, "#"))return;//equal
    //else
    int result = atoi(temp);

    *nodeP = (Node*)malloc(sizeof(Node)); node = *nodeP;
    if(!node) {
        printERROR(1);
        exit(1);
    }
    node->data = result; node->left = NULL; node->right = NULL;
    free(temp);
    createTreeQuickly(&node->left);
    createTreeQuickly(&node->right);
}
//Set values in one time for a tree just being created quickly
void valueKeyFromQuick(Node* node){
    if(!node) return;
    char * temp = (char*)malloc(sizeof(char)*10);
    scanf("%s", temp);
    int result = atoi(temp);
    free(temp);
    node->key = result;
    valueKeyFromQuick(node->left);
    valueKeyFromQuick(node->right);
}

//Create a new and empty binary tree
Status InitBiTree(Tree ** T) {

    Tree * newT = (Tree *)malloc(sizeof(Tree));
    newT->length = 0;
    newT->name = NULL;//set its name outside
    newT->root = NULL;
    if (newT != NULL) {

```

```

        *T = newT;
        return OK;
    } else {
        printERROR(1);
        return FALSE;
    }

}

//destroy this tree and free all space it had taken.
void DestroyTree(Tree * T) {
    ClearBiTree(T);
    free(T->name);
    free(T);
}

//clear all data of this tree
void ClearBiTree(Tree * T) {
    ClearBiTree_recurve(T->root);
    T->root = NULL;
    T->length = 0;
}
void ClearBiTree_recurve(Node * n) {//for recursion
    if (!n) return;
    ClearBiTree_recurve(n->left); n->left = NULL;
    ClearBiTree_recurve(n->right); n->right = NULL;
    // printf("key=%d value=%d\n",n->key,n->data);
    free(n);//and set old position of n as null outside
}

//get the root node of tree-T
Node * Root(Tree * T) {
    return T->root;
}

//assign a value for node-e of tree-T
Status Assign(Tree * T, Node * e, ElemType value) {
    //Assign_recurve(T->root, e, &value);
    if (e == NULL) {
        return FALSE;
    } else {
        e->data = value;
        return OK;
    }
}

```

```

}

//get value of node-e of tree-T
ElemType Value(Tree * T, Node *e) {
    if (!T || !e) return FALSE;
    return e->data;
}

//just print error info
void printERROR(int code) {

    switch (code) {
        case 1:
            printf("\nStack overflow\n");
            break;
        case 2:
            printf("\nThere isn't such a tree.\n");
            break;
        case 3:
            printf("\nThere isn't such a node in this tree.\n");
            break;
    }
}

//find the node of tree-T by key
Node * FindNode(Tree * T, ElemType key) {
    Node * result = NULL;
    FindNode_recurve(T->root, key, &result);
    return result;
}

int FindNode_recurve(Node * node, ElemType key, Node ** result) {
    if (!node) return 0;
    if (node->key == key) {
        *result = node;
        return 1;//1 means that we found the element, so we can set a flag to
stop recursion
    }
    // FindNode_recurve(node->left, key, result);
    // FindNode_recurve(node->right, key, result);
    // return 1;
    if (FindNode_recurve(node->left, key, result)) return 1;
    if (FindNode_recurve(node->right, key, result)) return 1;
    return 0;
}

```

```

//get left child of tree-T
Node * LeftChild(Tree * T, Node * e) {
    return e->left;
}

//get right child of tree-T
Node * RightChild(Tree * T, Node * e) {
    return e->right;
}

//find the parent node of node-e
Node * Parent(Tree * T, Node * e) {
    Node * parent = NULL;
    if(e->key == T->root->key){
        return NULL;
    }
    Parent_recurve(T->root, e, &parent);
    return parent;
}
int Parent_recurve(Node * start, Node * key, Node ** parent) {
    if (!start) return 0;
    if (start->right == key || start->left == key) {
        *parent = start;
        return 1;
    }
    if (Parent_recurve(start->left, key, parent)) return 1;
    if (Parent_recurve(start->right, key, parent)) return 1;
    return 0;
}

//find the right sibling of node-e in tree-T
Node * RightSibling(Tree * T, Node * e) {
    Node * parent = Parent(T, e);
    if (parent) {//have parent node
        if (parent->right) {//and parent have right node
            if (parent->right != e){// and this right node is not the
parameter node
                return parent->right;
            }
        }
    }
    return NULL;
}

```

```

//find the left sibling of node-e in tree-T
Node * LeftSibling(Tree * T, Node * e) {
    Node * parent = Parent(T, e);
    if (parent) {//have parent node
        if (parent->left) {//and parent have right node
            if (parent->left != e) {// and this right node is not the parameter
node
                return parent->left;
            }
        }
    }
    return NULL;
}

//PreOrderTraverse
Status PreOrderTraverse(Tree * T) {
    if (T == NULL) return FALSE;
    PreOrderTraverse_recurve(T->root);
    printf("\n");
    return OK;
}
void PreOrderTraverse_recurve(Node * n) {//for recursion
    if (!n) return;
    printf("key=%d value=%d\n", n->key,n->data);
    PreOrderTraverse_recurve(n->left);
    PreOrderTraverse_recurve(n->right);
}

//InOrderTraverse
Status InOrderTraverse(Tree * T) {
    if (T == NULL) return FALSE;
    InOrderTraverse_recurve(T->root);
    printf("\n");
    return OK;
}
void InOrderTraverse_recurve(Node * n) //for recursion
{
    if (!n) return;
    InOrderTraverse_recurve(n->left);
    printf("key=%d value=%d\n", n->key,n->data);
    InOrderTraverse_recurve(n->right);
}

//PostOrderTraverse

```

```

Status PostOrderTraverse(Tree * T) {
    if (T == NULL) return FALSE;
    PostOrderTraverse_recurve(T->root);
    printf("\n");
    return OK;
}
void PostOrderTraverse_recurve(Node * n) { //for recursion
    if (!n) return;
    PostOrderTraverse_recurve(n->left);
    PostOrderTraverse_recurve(n->right);
    printf("key=%d value=%d\n", n->key,n->data);
}

//insert a subTree-c to the tree-T, 0->L 1->R
Status InsertChild(Tree * T, Node * p, int LR, Tree * c) {

    int flag = 0;
    if (LR == 0) {
        Node * left = p->left;
        if (!left) {//left is null
            p->left = c->root;
            flag = 1;
        }
    } else {
        Node * right = p->right;
        if (!right) {//right is null
            p->right = c->root;
            flag = 1;
        }
    }
    if (flag) {
        T->length += c->length;
        free(c);//but we preserve the data part
        return OK;
    } else {
        return FALSE;
    }
}

//insert a new node to the tree-T, 0->L 1->R
Status InsertNode(Tree * T, Node * p, int LR, Node * c) {

    if (!LR) {
        Node * left = p->left;

```

```

        if (!left) {//left is null
            p->left = c;
            T->length++;
            return OK;
        }
    } else {
        Node * right = p->right;
        if (!right) {//right is null
            p->right = c;
            T->length++;
            return OK;
        }
    }
    return FALSE;
}

//delete a child-tree
Status DeleteChild(Tree * T, Node * p, int LR) {
    if (!LR) {
        Node * left = p->left;
        if (left) {//left is not null
            T->length -= BiTreeLength(left);
            ClearBiTree_recurve(left);
            p->left = NULL;
            return OK;
        }
    } else {
        Node * right = p->right;
        if (right) {//right is not null
            T->length -= BiTreeLength(right);
            ClearBiTree_recurve(right);
            p->right = NULL;
            return OK;
        }
    }
    return FALSE;
}

//this tree is empty or not
Status BiTreeEmpty(Tree * T) {
    return T->length;
}

```

```

//get the depth of this binary tree
int depthMax = 0;
int BiTreeDepth(Tree * T) {
    depthMax = 0;
    Node * head = T->root;
    BiTreeDepth_recurve(head, 1);
    return depthMax;
}
void BiTreeDepth_recurve(Node * node, int i) {
    if (!node) return;
    if (i > depthMax) depthMax = i;

    BiTreeDepth_recurve(node->left, i + 1);
    BiTreeDepth_recurve(node->right, i + 1);

}

//traverse in level, kind of BST, without an aux-quene to save memory
int i = 0;
Status LevelOrderTraverse(Tree * T) {
    i = 0; //reset two flags
    if (T) {
        Node * head = T->root;
        for (i = 1; i <= BiTreeDepth(T); i++) {
            //each i means that we are trying to print all elements of the i'th
level
            LevelOreder_recurve(head, 1);
        }
        printf("\n");
        return OK;
    } else {
        return FALSE;
    }
}
void LevelOreder_recurve(Node * node, int depthNow) {
    if (!node) return;

    if (depthNow == i) {
        printf("key=%d value=%d\n", node->key, node->data);//print the go
back in recursion
    } else {
        LevelOreder_recurve(node->left, depthNow + 1);
        LevelOreder_recurve(node->right, depthNow + 1);
    }
}

```

```

    }

}

//save this tree in disk in a simple way
Node ** saveList = NULL;
Status saveTree(Tree * T) {

    FILE * fP = fopen(T->name, "wb");
    if (!fP) return FALSE;
    int depth = BiTreeDepth(T);
    int LISTSIZE = pow(2, depth) - 1;
    saveList = (Node **)malloc(sizeof(Node*)*LISTSIZE);
    int i = 0;
    for (i = 0; i < LISTSIZE; i++) {
        saveList[i] = NULL;
    }//initialize save list

    Node * nullNode = (Node*)malloc(sizeof(Node));
    nullNode->data = NULLNODECODE;
    nullNode->key = NULLNODECODE;

    saveTree_recurve(T->root, 1);//call this function to load nodes to save
list, including null nodes

    for (i = 0; i < LISTSIZE; i++) {
        if (saveList[i]) {
            fwrite(saveList[i], sizeof(Node), 1, fP);
        } else {//null node
            fwrite(nullNode, sizeof(Node), 1, fP);
        }
    }

}//save node list to disk
free(saveList); saveList = NULL;//reset
fclose(fP);
return TRUE;
}

void saveTree_recurve(Node * node, int i) {
    if (!node) return;
    saveList[i - 1] = node;

    saveTree_recurve(node->left, 2 * i);
    saveTree_recurve(node->right, 2 * i + 1);
}

```

```

//Load tree data from disk
Node ** loadList = NULL;
int maxIndex_loadTree = 0;

Tree * LoadTree(char * TreeName) {

    maxIndex_loadTree = 0;
    loadList = (Node ** )malloc(sizeof(Node *) * LOAD_LIST_SIZE); //we
set a init-num, in fact it's not elegant.
    register int i = 0;
    for (i = 0; i < LOAD_LIST_SIZE; i++) {
        loadList[i] = NULL;
    }

    FILE * fP = fopen(TreeName, "rb");
    if (!fP) return NULL;
    else { //this file exists
        i = 0;
        Node * newNode = (Node *)malloc(sizeof(Node));

        while (fread(newNode, sizeof(Node), 1, fP)) {
            loadList[i] = newNode; i++;
            newNode = (Node *)malloc(sizeof(Node));
        }
        free(newNode);
    }

    //now we got a list full of nodes
    //int i is length of this list
    //    int depth = (int)sqrt(i+1);
    maxIndex_loadTree = i;
    Node * root = LoadTree_recurve(1); //load
    Tree * T = (Tree *)malloc(sizeof(Tree));
    if (T) {
        T->root = root;
        T->length = i;
        T->name = TreeName;
    }
    free(loadList);
    return T;
}

Node * LoadTree_recurve(int i) {
    Node * node = loadList[i - 1];

```

```

assert(node); //variable "node" here is not a null node
if (node->data == NULLNODECODE) {
    return NULL;
}
if (i * 2 > maxIndex_loadTree) {//at the bottom of this sub-tree
    node->left = NULL;
} else {
    node->left = LoadTree_recurve(i * 2);
}
if (i * 2 + 1 > maxIndex_loadTree) {//at the bottom of this sub-tree
    node->right = NULL;
} else {
    node->right = LoadTree_recurve(i * 2 + 1);
}
return node;
}

int BiTreeLength(Node * node) {
    if (!node) return 0;
    return BiTreeLength(node->left) + BiTreeLength(node->right) + 1;
}

```

#### 4. 无向图源代码:

GraphMain.c

```

#include <windows.h>
#include <Graph.h>
typedef int ElemType; //make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0

/*
    print some information to users
    1. Stack overflow
    2. There is not such vertex
    3. Successful!
    4. You should create a graph first
*/
void printInfo(int code) {
    switch (code) {
        case 1:
            printf("Stack overflow.\n");
            break;
        case 2:
            printf("There is not such vertex.\n");
    }
}

```

```

        break;
    case 3:
        printf("Successful!\n");
        break;
    case 4:
        printf("You should create a graph first.\n");
        break;
    }
}

//bro-function of func-"fgets", which can get string input without '\n'
char *fgetsNoN(char *buf, int bufsize, FILE *stream) {
    char * returnP = fgets(buf, bufsize, stream);
    int i = 0;
    while (buf[i] != '\n') {
        i++;
    }
    buf[i] = '\0';
    return returnP;
}

/*
    a logistic func to low down lines of code
*/
Vertex * getKeyNode(Graph * g) {
    printf("\nPlease input the key of this vertex:");
    ElemType key = 0; scanf("%d%c", &key);
    return FindVertex(g, key);
}

/*
    print the top-layer menu
*/
void printMenu(Graph * g) {
    system("cls");

    printf("----Menu for Undirected and Unweighted Graph----\n\n");
    printf("      1. CreateEmptyGraph\t2. DestoryGraph\n");
    printf("      3. LocateVex\t4. GetVex \n");
    printf("      5. PutVex\t6. FirstAdjVex\n");
    printf("      7. NextAdjVex\t8. InsertVex\n");
    printf("      9. DeleteVex\t10. InsertArc\n");
    printf("      11. DeleteArc\t12. DFSTraverse\n");
    printf("      13. BFSTraverse\t0. Exit\n");
}

```

```

if (g) {
    printf("\n-----Vertexs' Key List-----\n");
    Vertex * startVex = g->head;
    while (startVex) {
        printf("%d,", startVex->key);
        startVex = startVex->next;
    }
    printf("\n-----\n");
} else {
    printf("\n-----\n");
}

}

int main() {

    int op = 1;
    Graph * g = NULL;
    while (op) {
        printMenu(g);//print the top-layer menu
        printf("\tPlease select your option [0~13]:");
        scanf("%d%c", &op);

        switch (op) {
        case 1://CreateEmptyGraph
            if (!CreateEmptyGraph(&g)) printInfo(1);
            else printInfo(3);
            break;
        case 2://DestoryGraph
            if (!g) printInfo(4);
            else {
                DestoryGraph(g);
                printInfo(3);
                g = NULL;
            }
            break;
        case 3://LocateVex
            if (!g) printInfo(4);
            else {
                printf("\nPlease input \"data\" of this vertex:");
                ElemType data = 0; scanf("%d%c", &data);
                int key = LocateVex(g, data);
            }
        }
    }
}

```

```

        if (key == NOT_FOUND_CODE) printf("we don't find
such a vertex having this data.\n");
        else printf("data of this vertex is %d.\n", key);
    }
    break;
case 4://GetVex
    if (!g) printInfo(4);
    else {
        printf("\nPlease input \"key\" of this vertex:");
        ElemType key = 0; scanf("%d%c", &key);
        int data = GetVex(g, key);
        if (data == NOT_FOUND_CODE) printf("we don't find
such a vertex having this key.\n");
        else printf("key of this vertex is %d.\n", data);
    }
    break;
case 5://PutVex
    if (!g) printInfo(4);
    else {
        printf("\nPlease input \"key\" of this vertex:");
        ElemType key = 0; scanf("%d%c", &key);
        Vertex * v = FindVertex(g, key);
        if (!v) printInfo(2);
        else {
            printf("\nPlease input a new value for this vertex:");
            ElemType newValue = 0; scanf("%d%c",
&newValue);
            PutVex(g, v, newValue);
        }
    }
    break;
case 6://FirstAdjVex
    if (!g) printInfo(4);
    else {
        printf("\nPlease input \"key\" of this vertex:");
        ElemType key = 0; scanf("%d%c", &key);
        Vertex * v = FindVertex(g, key);
        if (!v) printInfo(2);
        else {
            Vertex * getVer = FirstAdjVex(g, v);
            if (!getVer) printf("this vertex don't have adj-vex\n");
            else {
                Vertex * trueV = FindVertex(g, getVer->key);
                printf("the first adj-vex: key=%d value=%d\n",

```

```

trueV->key, trueV->data);
}
}
}
break;
case 7://NextAdjVex
if (!g) printInfo(4);
else {
    printf("\nPlease input \"key\" of this vertex:");
    ElemType key = 0; scanf("%d%c", &key);
    Vertex * v = FindVertex(g, key);
    if (!v) printInfo(2);
    else {
        printf("\nPlease input \"key\" of adj-vex of the prior vertex:");
        ElemType keyAdj = 0; scanf("%d%c", &keyAdj);
        Vertex * vAdj = FindVertex(g, keyAdj);
        if (!vAdj) printInfo(2);
        else {
            Vertex * nextV = NextAdjVex(g, v, vAdj);
            if (!nextV) printf("We don't find such a next-adj-vex.\n");
            else {
                Vertex * trueV = FindVertex(g, nextV->key);
                printf("next-adj-vex: key=%d value=%d", trueV->key,
trueV->data);
            }
        }
    }
}
break;
case 8://InsertVex
if (!g) printInfo(4);
else {
    printf("Please input a new key of this vertex:");
    ElemType key = 0; scanf("%d%c", &key);
    if (FindVertex(g, key)) printf("This key has been occupied.\n");
    else {
        printf("Please input data of this vertex:");
        ElemType data = 0; scanf("%d%c", &data);
        Vertex * newVertex = (Vertex *)malloc(sizeof(Vertex));
        newVertex->key = key; newVertex->data = data; newVertex->next =
NULL;
        if (!newVertex) {
            printInfo(1);
            free(newVertex);
        }
    }
}

```

```

        }
        else {
            InsertVex(g, newVertex);
            printInfo(3);
        }
    }
}
break;

case 9://DeleteVex
if (!g) printInfo(4);
else {
    printf("\nPlease input \"key\" of this vertex:");
    ElemType key = 0; scanf("%d%c", &key);
    Vertex * v = FindVertex(g, key);
    if (!v) printInfo(2);
    else {
        DeleteVex(g, v); printInfo(3);
    }
}
break;

case 10://InsertArc
if (!g) printInfo(4);
else {
    printf("\nPlease input \"key\" of two vertexs( splited by space ):");
    ElemType keyV = 0; ElemType keyW = 0;
    scanf("%d%c%d%c", &keyV, &keyW);
    Vertex * v = FindVertex(g, keyV); Vertex * w = FindVertex(g, keyW);
    if (v && w) {
        if (InsertArc(g, v, w)) printInfo(3);
        else printf("perhaps these two vertexs have been connected already
or some error else happen!");
    } else {
        printInfo(2);
    }
}
break;

case 11://DeleteArc
if (!g) printInfo(4);
else {
    printf("\nPlease input \"key\" of two vertexs( splited by space ):");
    ElemType keyV = 0; ElemType keyW = 0;
    scanf("%d%c%d%c", &keyV, &keyW);
    Vertex * v = FindVertex(g, keyV); Vertex * w = FindVertex(g, keyW);
    if (v && w) {
}
}

```

```

        if (DeleteArc(g, v, w)) printInfo(3);
        else printf("Error!");
    } else {
        printInfo(2);
    }
}
break;
case 12://DFSTraverse
if (!g) printInfo(4);
else {
    printf("----DFS starts.----\n");
    DFSTraverse(g);
    printf("----DFS end.----\n");
}
break;
case 13://BFSTraverse
if (!g) printInfo(4);
else {
    printf("----BFS starts.----\n");
    BFSTraverse(g);
    printf("----BFS end.----\n");
}
break;
case 0://Exit
op = 0;
break;
}
system("pause");
}
printf("\nSee you\n");
return 0;
}

Graph.h
#include <QueueForGraph.h>
```

```

typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define VERTEX_LIST_SIZE 31
#define NOT_FOUND_CODE 90071
```

```
/**
```

```

* by adjacency list. All parameter vertexs are in the first line
*/
typedef struct graph {
    int vertexCount;
    Vertex * head;
} Graph;

//Status CreateGraph(Graph ** g, Vertex ** vertexList, Edge ** edgeList);//better a
more convient way to create a new graph
Status CreateEmptyGraph(Graph ** gP);
Status DestoryGraph(Graph * g);
ElemType LocateVex(Graph * g, ElemType data);//Confused about the meaning of this
function
ElemType GetVex(Graph * g, ElemType loc);
Status PutVex(Graph * g, Vertex * v, ElemType value);
Vertex * FirstAdjVex(Graph * g, Vertex * v);
Vertex * NextAdjVex(Graph * g, Vertex * v, Vertex * w);
Status InsertVex(Graph * g, Vertex * v);
Status DeleteVex(Graph * g, Vertex * v);
Status InsertArc(Graph * g, Vertex * v, Vertex * w);
Status DeleteArc(Graph * g, Vertex * v, Vertex * w);
void DFSTraverse(Graph * g);
void BFSTraverse(Graph * g);

Status deleteVexFromAdj(Vertex * del, Vertex * startVex);
void deleteAdjLink(Vertex * v);
void dfs_recurve(Graph * g, Vertex * v);
Status isMarked(Vertex * v);
Vertex * FindVertex(Graph * g, ElemType key);
Vertex * addVexToAdjEnd(Vertex * startVex, Vertex * newVex);
Status isAdj(Graph * g, Vertex * v, Vertex * w);

//Create a graph with no vertex and edge
Status CreateEmptyGraph(Graph ** gP) {
    if (*gP) DestoryGraph(*gP);

    Graph * g = (Graph *)malloc(sizeof(Graph));
    if (!g) return FALSE;
    *gP = g;

    g->vertexCount = 0;
    g->head = NULL;
    return TRUE;
}

```

```

}

/*
Find a vertex by a key in graph-g
*/
Vertex * FindVertex(Graph * g, ElemType key) {
    Vertex * startVex = g->head;
    while (startVex) {
        if (startVex->key == key) return startVex;
        startVex = startVex->next;
    }
    return NULL;
}

//this func is used to find node's key from value
ElemType LocateVex(Graph * g, ElemType data) {
    Vertex * startVex = g->head;
    while (startVex) {
        if (startVex->data == data) {//find the vertex we want
            return startVex->key;
        }
        startVex = startVex->next;
    }
    return NOT_FOUND_CODE;
}

//return the first vextex linked with Vretex-v
Vertex * FirstAdjVex(Graph * g, Vertex * v) {
    return v->adj;
}

//this is a WTF function
Vertex * NextAdjVex(Graph * g, Vertex * v, Vertex * w) {
    Vertex * tempV = v;
    while (tempV) {
        //cause w is in the first link, so w don't have adj vertex
        if (tempV->key == w->key) {
            return tempV->adj;
        }

        tempV = tempV->adj;
    }
    return NULL;
}

```

```

//this func is used to find node's value from key
ElemType GetVex(Graph * g, ElemType loc) {
    Vertex * startVex = g->head;
    while (startVex) {
        if (startVex->key == loc) {//find the vertex we want
            return startVex->data;
        }
        startVex = startVex->next;
    }
    return NOT_FOUND_CODE;
}

/*
put a value for a specialized vertex
*/
Status PutVex(Graph * g, Vertex * v, ElemType value) {
    if (v) {
        v->data = value;
        return TRUE;
    } else {
        return FALSE;
    }
}

/*
Insert a nude vex in this graph, which would be the last vertex in first line
*/
Status InsertVex(Graph * g, Vertex * v) {

    Vertex * lastVex = g->head;
    if (lastVex) {
        while (lastVex->next) {
            lastVex = lastVex->next;
        }
        //now the lastVex is the last vertex in this graph
        lastVex->next = v;
    } else {
        g->head = v;
    }
    v->next = NULL;
    v->adj = NULL;

    g->vertexCount++;
}

```

```

//then we should update v's linking infomation, but v is nude???
return TRUE;
}

/***
*      calling this func when you need to delete a vertex from a adj-link.
*/
Status deleteVexFromAdj(Vertex * del, Vertex * startVex) {
    Vertex * tempV = startVex->adj;
    if(!tempV) return OK;
    if (tempV->key == del->key) {//lucky enough, it's the first one
        startVex->adj = tempV->adj;
        free(tempV);
        return TRUE;
    }
    while (tempV->adj != NULL) {

        if (tempV->adj->key == del->key) {
            Vertex * newAdj = tempV->adj->adj;
            free(tempV->adj);
            tempV->adj = newAdj;
            return TRUE;
        }

        tempV = tempV->adj;
    }
    return FALSE;
}

```

```

//Destory the whole graph
Status DestoryGraph(Graph * g) {
    Vertex * startVex = g->head;
    while (startVex) {
        Vertex * tempV = startVex->next;
        deleteAdjLink(startVex);
        startVex = tempV;
    }
    free(g);
    return TRUE;
}

```

```

//delete a adj-link
void deleteAdjLink(Vertex * startVex) {

```

```

Vertex * v = startVex; Vertex * vAdj = v;
while (v->adj) {
    vAdj = v->adj;
    free(v);
    v = vAdj;
}

/*
delete a vertex and clean all edge concered about it
*/
Status DeleteVex(Graph * g, Vertex * v) {
    Vertex * startVex = g->head;

    while(startVex){
        if(startVex->key != v->key){
            deleteVexFromAdj(v, startVex);
        }
        startVex = startVex->next;
    }

    startVex = g->head;

    if(v->key == g->head->key){ //delete head
        g->head = g->head->next;
        deleteAdjLink(startVex);
    }else{
        while(startVex->next){
            if(startVex->next->key == v->key){
                Vertex * tempV = startVex->next->next;
                deleteAdjLink(startVex->next);
                startVex->next = tempV;
                break;
            }
            startVex = startVex->next;
        }
    }

    g->vertexCount--;
    return TRUE;
}

/*

```

```

find out v and w is connected or not.
return: TRUE or FALSE
*/
Status isAdj(Graph * g, Vertex * v, Vertex * w) {
    Vertex * startVex = g->head;
    int flag = 1;
    while (startVex) {
        if (startVex->key == v->key) {//find the vertex we want
            if (flag) {
                Vertex * tempV = startVex->adj;
                while (tempV) {
                    if (tempV->key == w->key) return TRUE;
                    tempV = tempV->adj;
                }
                flag = 0;
            }
        } else if (startVex->key == w->key) {
            if (flag) {
                Vertex * tempV = startVex->adj;
                while (tempV) {
                    if (tempV->key == v->key) return TRUE;
                    tempV = tempV->adj;
                }
                flag = 0;
            }
        }
        startVex = startVex->next;
    }
    return FALSE;
}

/*
insert a arc into this graph and update lots of relationships....
*/
Status InsertArc(Graph * g, Vertex * v, Vertex * w) {
    Vertex * startVex = g->head;
    if (isAdj(g, v, w)) return FALSE;//check they are connected or not

    Vertex * vCopy = (Vertex *)malloc(sizeof(Vertex)); Vertex * wCopy = (Vertex
*)malloc(sizeof(Vertex));
    vCopy->next = NULL; wCopy->next = NULL;
    vCopy->key = v->key; wCopy->key = w->key;
    vCopy->adj = NULL; wCopy->adj = NULL;
}

```

```

//in fact we don't have to value data, so that we don't have to update all same-vertex in
all adj-links

//all vertex in adj-link is a copy of the origin

int flag = 0;

while (startVex) {
    if (startVex->key == v->key) {//find the vertex we want
        startVex->adj = addVexToAdjEnd(startVex->adj, wCopy);
        flag++;
    } else if (startVex->key == w->key) {
        startVex->adj = addVexToAdjEnd(startVex->adj, vCopy);
        flag++;
    }//we don't check v or w is already have linking or not
    startVex = startVex->next;
}

if (flag == 2) {
    return TRUE;
} else {
    return FALSE;
}

/*
A tool func to help add a vertex to a adj-link's tail
Return: the new head of adj-link after adding
*/
Vertex * addVexToAdjEnd(Vertex * firstAdjVex, Vertex * newVex) {

    Vertex * tempV = firstAdjVex;
    if (tempV) {
        while (tempV->adj) {
            tempV = tempV->adj;
        }//make tempV become the last one in this adj-link
        tempV->adj = newVex;
        return firstAdjVex;
    } else {
        return newVex;
    }
}

/*

```

```

delete a arc from this graph
*/
Status DeleteArc(Graph * g, Vertex * v, Vertex * w) {
    Vertex * startVex = g->head;
    int flag = 0;
    while (startVex) {
        if (startVex->key == v->key) {
            if (deleteVexFromAdj(w, startVex)) flag++;
        } else if (startVex->key == w->key) {
            if (deleteVexFromAdj(v, startVex)) flag++;
        }
        startVex = startVex->next;
    }
    if (flag == 2) return TRUE;
    else return FALSE;
}

//some data to support two search functions
ElemType * markVertexKeys;//save keys
int size_markVertex;

//figure out parameter Vertex-v is marked or not, so that we can make decision outside
Status isMarked(Vertex * v) {
    int i = 0;
    for (i = 0; i < size_markVertex; i++) {
        if (markVertexKeys[i] == v->key) {
            return TRUE;
        }
    }
    return FALSE;
}

/*
DFS depth first search
*/
void DFSTraverse(Graph * g) {
    size_markVertex = 0;
    markVertexKeys = (ElemType *)malloc(sizeof(ElemType) * g->vertexCount);
    int i = 0;
    for (i = 0; i < g->vertexCount; i++) {
        markVertexKeys[i] = NOT_FOUND_CODE;//init this array
    }
    //start dfs
    dfs_recurve(g, g->head);
}

```

```

//end dfs
free(markVertexKeys);
}
void dfs_recurve(Graph * g, Vertex * v) {
markVertexKeys[size_markVertex] = v->key; size_markVertex++;
//do sth
Vertex * trueV = FindVertex(g, v->key);
if(!trueV) printf("error!");
else {
    printf("key=%d value=%d\n", trueV->key, trueV->data);
    Vertex * tempV = trueV->adj;
    while (tempV) {
        if (!isMarked(tempV)) {
            dfs_recurve(g, tempV);//not be marked so we traverse this node
        }
        tempV = tempV->adj;
    }
}

/*
BFS breadth first search
*/
void BFSTraverse(Graph * g) {
if (!g->head) return;
Queue * q;
if (!initQueue(&q)) return;

size_markVertex = 0;
markVertexKeys = (ElemType *)malloc(sizeof(ElemType) * g->vertexCount);
if (!markVertexKeys) return;

int i = 0;
for (i = 0; i < g->vertexCount; i++) {
    markVertexKeys[i] = 0;//init this array
}
//init-work is done

enQueue(q, g->head);
printf("key=%d value=%d\n", g->head->key, g->head->data);//startpoint
markVertexKeys[size_markVertex] = g->head->key; size_markVertex++;

```

```

while (q->length) {//when Queue is not empty
    Vertex * v = deQueue(q);
    Vertex * startVex = g->head;
    while (startVex) {
        if (startVex->key == v->key) {//found this vertex's adj-vertices
            Vertex * tempV = startVex->adj;
            while (tempV) {
                if (!isMarked(tempV)) {
                    Vertex * trueV = FindVertex(g, tempV->key);
                    if (!trueV) printf("error!\n");
                    else{
                        printf("key=%d      value=%d\n", trueV->key,
trueV->data);
                        enQueue(q, tempV);
                        markVertexKeys[size_markVertex] = tempV->key;
size_markVertex++;
                    }
                }
                tempV = tempV->adj;
            }
            break;
        }
        startVex = startVex->next;
    }
}

//free memory
free(markVertexKeys);
destroyQueue(q);
}

```

## QueueForGraph.h

```

#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;//make sure that it's a basic data type
typedef int Status;
#define TRUE 1
#define FALSE 0
#define OK 1
#define LISTINCREMENT 10 //step-length when growing and cutting
#define LIST_INIT_SIZE 20

typedef struct vertex {

```

```

ElemType key;
ElemType data;
struct vertex * next;
struct vertex * adj;
} Vertex;

typedef struct myqueue {
    int length;
    int listszie;
    Vertex ** head;
} Queue;

/*
Initialize this queue with 100 size.
*/
Status initQueue(Queue ** qP) {
    Queue * q = (Queue *)malloc(sizeof(Queue));
    if (!q) return FALSE;
    *qP = q;
    q->length = 0;
    q->listszie = LIST_INIT_SIZE;
    q->head = (Vertex **)malloc(sizeof(Vertex*)*LIST_INIT_SIZE);
    return OK;
}

/*
when this gragh is too large, we have to grow this list to support BFS
*/
void growList(Queue * q) {
    q->head = (Vertex **)realloc(q->head, q->listszie + sizeof(Vertex*)*LISTINCREMENT);
    q->listszie += LISTINCREMENT;
    if (q->head == NULL) exit(0);
}

/*
on the other hand...cut its size.
*/
void cutList(Queue * q) { //Pay attention to the opportunity to avoid frequently cutting
and growing
    q->head = (Vertex **)realloc(q->head, q->listszie - sizeof(Vertex*)*LISTINCREMENT);
    q->listszie -= LISTINCREMENT;
    if (q->head == NULL) exit(0);
}

```

```
}

/*
 add to the end
*/
void enQueue(Queue * q, Vertex * v) {
    Vertex ** verArray = q->head;
    if (q->length == q->listsize) {
        growList(q);
    }
    verArray[q->length] = v;
    q->length++;
}

/*
 delete item from the head position
 return the vertex dequeued
*/
Vertex * deQueue(Queue * q) {

    if (q->length == 0) return NULL;
    Vertex ** verArray = q->head;
    Vertex * returnVer = verArray[0];
    int i = 1;
    for (i = 1; i < q->length; i++) {
        verArray[i - 1] = verArray[i];//@override
    }
    q->length--;
    return returnVer;
}

//destroy queue and free memory
void destroyQueue(Queue * q) {
    free(q->head);
    free(q);
}
```