同济大学

TONGJI UNIVERSITY

《数据结构》课程设计总结

项目名称	尾号为 0 的数据结构课程设计题
学　　院	电子与信息工程学院
专　　业	计算机科学与技术
学生姓名	刘鲲
学　　号	1751740
指导教师	武妍 教授
日　　期	2019 年 8 月 13 日

目 录

1 算法实现设计说明

1.1 题目

哈希表的建立、查找、插入和删除。说明：任意选择开放定址法或者链地址法中的一种。

1.2 软件功能

1.2.1 软件功能设计

- 能够将 txt 存储的数据导入到软件中，建立哈希表
- 能够将开放定址法和链地址法用较好的视觉控件表达
- 能够查找、插入、删除

1.2.2 功能实现方式

QT 下的控件 QTableWidget 能较好地从视觉上表达哈希表的结构，尤其是链地址法的结构，如1.1所示。

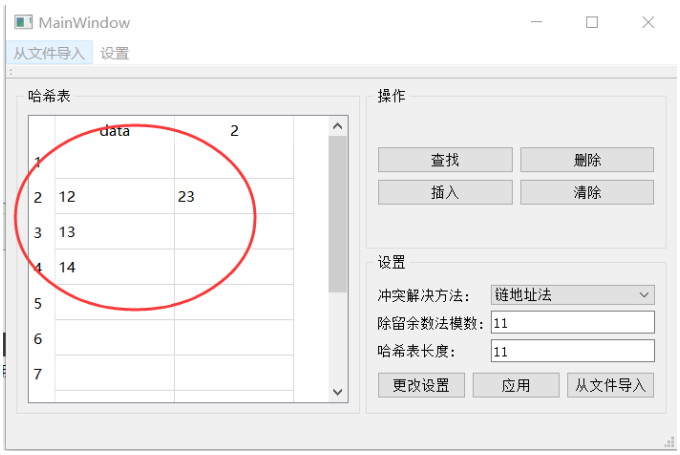


图 1.1 用户界面

按下按钮，就可以进行相应的操作。接下来以插入为例，说明如何实现插入功能。将一个按钮移动到 ui 界面后，右键按钮-> 转到槽，选择 clicked(); 然后编写以下代码：

```
1  void MainWindow::on_pBtnInsert_clicked()
2  {
3      qDebug() << "pBtnSearch";
4
5      QInputDialog dia(this);
6      dia.setFixedHeight(107);
7      dia.setFixedWidth(800);
8      dia.setWindowTitle("插入");
9      dia.setLabelText("请输入关键字: ");
10     dia.setInputMode(QInputDialog::TextInput);
11
12     if(dia.exec() != QInputDialog::Accepted){
```

```
13          return;
14  }
15      qDebug() << dia.textValue();
16      QString key = dia.textValue().trimmed();
17      int addr = key.toInt() % this->mod;
18      QTableWidgetItem *curItem;
19      int cnt = 0;
20      bool isFailed = 1;
21
22      switch (this->conflictMethod) {
23      case OPEN_ADDR:
24          while(cnt < this->tableSize) {
25              curItem = ui->tableWidgetOpen->item(addr,0);
26              if(curItem == nullptr ||
27                  (curItem && curItem->text()==tr(""))){
28                  ui->tableWidgetOpen->setItem(addr,0,new QTableWidgetItem(key))
                        ;
29                  isFailed = 0;
30                  break;
31              }
32              else {
33                  if(curItem->text()==key){
34                      qDebug() << "insert failed(exist)";
35                      isFailed = 1;
36                      break;
37                  }
38                  else {
39                      addr = (addr + 1) % mod;
40                      cnt++;
41                      qDebug() << "Open addressing";
42                  }
43
44              }
45          }
46          break;
47      case LINK_ADDR:
48          while(1) {
49              curItem = ui->tableWidgetOpen->item(addr,cnt);
50              if(curItem == nullptr ||
51                  (curItem && curItem->text()==tr(""))){
52
53                  if(cnt+1 > ui->tableWidgetOpen->columnCount())
```

```
54                  ui->tableWidgetOpen->setColumnCount(cnt+1);
55 ui->tableWidgetOpen->setItem(addr,cnt,new QTableWidgetItem(key));
                            isFailed = 0;
56              break;
57          }
58          else {
59              if(curItem->text()==key){
60                  qDebug() << "insert failed(exist)";
61                  isFailed = 1;
62                  break;
63              }
64              else {
65                  //addr = (addr + 1) cnt++;
66                  qDebug() << "Link addressing";
67              }
68          }
69      }
70      break;
71
72  } // end of switch
73
74  // 满了无法再加
75  if(cnt >= this->tableSize || isFailed){
76      // 错误提示
77      QMessageBox msgBox(QMessageBox::Warning,"错误提示","插入失败",nullptr
            ,this);
78      msgBox.addButton(tr("我知道了"), QMessageBox::AcceptRole);
79  if (msgBox.exec() == QMessageBox::AcceptRole){
80              qDebug() <<"insert failed";
81          }
82      return;
83  }
84
85  QMessageBox msgBox(QMessageBox::Information,"结果提示","插入成功",nullptr
        ,this);
86  msgBox.addButton(tr("我知道了"), QMessageBox::AcceptRole);
87  if (msgBox.exec() == QMessageBox::AcceptRole){
88      qDebug() <<"insert success";
89  }
90  return;
91 }
```

代码 1.1 按钮操作设置代码

代码说明:其中 QInputDialog 是一个输入对话框。case 是根据解决冲突的方法调用不同的操作。QMessageBox 是弹出一个提示框。QTableWidget 则是用于显示哈希表的控件。

由于是右键转到槽，所以不需要信号连接。同样以类似的方式完成其它功能。

注意到文件导入需要添加如下代码：

```
QString curPath = QDir::currentPath();
QString dlgTitle = "打开一个txt文件";
QString filter = "(*.txt)";
QString aFileName=QFileDialog::getOpenFileName(this,dlgTitle,curPath,filter
    );
if (aFileName.isEmpty())
  return;

QFile aFile(aFileName);
if (!aFile.exists()) //文件不存在
  return;
if (!aFile.open(QIODevice::ReadOnly | QIODevice::Text))
  return;
```

代码 1.2 文件导入设置代码

代码说明：QFileDialog 是一个选取文件的控件，非常方便。打开文件后就是一系列读取文件的操作，不再赘述。

1.3 设计思想

1.3.1 数据结构的选用与设计思想

我开放定址法和链地址法都实现了，我先用纯 C++ 的方式说明开放定址法的算法。由于 QT 的控件封装性较强，很多纯 C++ 的操作都能简化。软件演示部分展示链地址法。

完成一个哈希表，主要要确定：哈希函数和冲突解决方法。为了实现的简易性，数据类型我选取了最简单的 int，哈希函数我使用了最简单的除留余数法。

```
typedef struct {
  vector <string>elem;//存放元素
  int count;//当前元素个数
  int size;//哈希表的大小
  int mod;//计算哈希函数时 mod 几
}HashTable;
```

代码 1.3 数据结构的选用

代码说明：我使用了 vector 作为元素的存储方式。由于选用的是除留余数法作为哈希函数，故需要 mod 来记录余数是多少。

1.3.2 算法设计基本流程

首先，我们先用纯 C++ 方式实现哈希表的基本操作。在这里，我以开放定址法为例。而在软件实例演示中，将演示链地址法。

A. 建立哈希表

建立哈希表相当于多次调用插入操作，故不再进行算法说明。

B. 插入

```
void InsertHash(HashTable *h, KeyType key)
{
  int cnt = 1;
  //用散列函数计算关键词
  int addr = Hash(key, h->mod) % h->mod;
  //int addr0 = addr;

  //当指定元素不为空值时，发生冲突
  while (h->elem[addr].size() != NULLKEY) {
    cnt++;
    addr = (addr + 1) % h->mod;//关键词线性查找散列表后面空位
  }
  h->elem[addr] = key;//散列表指定位置为空，插入值
  cout << addr <<" ";//输出插入位置
}
```

代码 1.4 插入

算法说明：根据开放定址法的原理，addr = (addr + 1) % h->mod; 是这种方法的关键；我使用的是线性探测再散列法。

算法复杂度：$O(n)$

C. 删除

```
Status DeleteHash(HashTable *h, KeyType key)
{
  int cnt = 0;
  //用散列函数计算关键词
  int addr = Hash(key, h->mod) % h->mod;

  //当指定元素不为空值时，发生冲突
  while (cnt < h->size) {
    if(h->elem[addr] == key){
      h->elem.remove(h->elem.begin() + addr);
      break;
    }
    cnt++;
    addr = (addr + 1) % h->mod;//关键词线性查找散列表后面空位
  }

  //删除失败
  if(cnt >= h->size){
```

```
    return Failed;
}
  return OK;
}
```

代码 1.5 删除

算法说明：根据开放定址法的原理，同样是使用 addr = (addr + 1) % h->mod；注意有一个 cnt 计数器，主要是保证在找不到该元素时循环能停止。

算法复杂度：$O(n)$

D. **查找**

```
Status SearchHash(HashTable h, string key, int *addr)
{
  int cnt = 1;
  *addr = Hash(key, h.mod);  //散列地址

  //如果不为关键字则该地址冲突
  while (h.elem[*addr] != key) {
    cnt++;
    *addr = (*addr + 1)% h.mod; //线性查找下面的地址
    if (h.elem[*addr].size() == NULLKEY || *addr == Hash(key,h.mod)){
      //散列地址循环回到原点
      return UNSUCCESS;      //要查找的关键词不存在
    }
  }
  return SUCCESS;
}
```

代码 1.6 查找

算法说明：根据开放定址法的原理，同样是使用 addr = (addr + 1) % h->mod；但这里的参数是 int *addr, 为的是返回它所在的位置。

算法复杂度：$O(n)$

1.4 逻辑结构与物理结构

由于我相当于用纯 C++ 方式和 QT 两次实现了软件功能，所以分别作说明。

1.4.1 纯 C++ 方式

逻辑结构：线性结构。

```
  typedef struct {
    vector <string>elem;//存放元素
    int count;//当前元素个数
    int size;//哈希表的大小
    int mod;//计算哈希函数时 mod 几
```

```
6   }HashTable;
```

代码 1.7 数据结构的选用

物理结构：主要为 C++ STL Vector 中所需的结构。一般为顺序结构。

1.4.2 QT 方式

逻辑结构：线性结构。可以认为 QTableWidget 是像 Excel 表格一样的一行一行为一条数据，每条数据按照顺序排列。

物理结构：索引存储结构。其中主要用到的结构体是 QTableWidgetItem。其主要成员如下：

```
1 class QTableWidgetItem {
2   int row;
3   int column;
4   QString text;
5 }
```

代码 1.8 数据结构的选用

其中，主要通过 row 和 column 来索引。

1.5 开发平台

开发平台：QT 5.13.0 (MSVC 2017)
运行环境：QT Creator 4.9.2 Community
其它的库：C++ STL (纯 C++ 方式实现时)

1.6 系统的运行结果分析说明

1.6.1 应用开发工具进行调试及开发

QT 提供了非常方便的 QDebug 库，可以很方便地在控制台输出结果，例如在一些关键位置写下以下这些代码等：

```
1 qDebug() << "search success";
2 qDebug() << "pBtnSearch";
```

代码 1.9 应用开发工具进行调试及开发

然后打开 debug 调试，就可以看到如1.2这些提示：

```
delete success
delete success
pBtnSearch
"14"
search success
search success
pBtnClear
clear
```

图 1.2 调试

这样子我们就能很方便地知道程序何时进入了哪个函数，在这过程中的一些参数是什么，这加快了开发的速度。

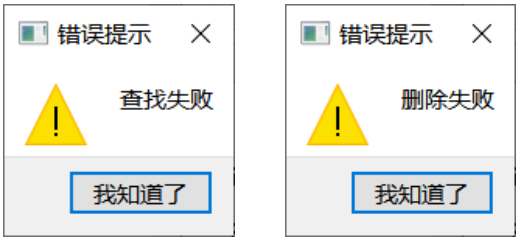
1.6.2 开发软件到达的成果

由于本项目较为简单，运行结果具体可见下一节的运行案例，正确性和稳定性可以得到较为直观的说明。

容错能力中，主要有以下：

- 查找、删除、插入操作失败的错误提示
- 关键字不能重复
- mod 不能超过 size
- 未设置哈希表大小和除留余数法的余数就导入数据
- 哈希表大小和除留余数法的余数不合法，为 0 或字母等等

当遇到以上情况时，都会跳出错误提示或以按钮变灰等方式纠错，例如1.3。



(a) 查找失败　　(b) 删除失败

图 1.3 错误提示

其它纠错设置的错误提示不再一一截图说明。

1.6.3 运行案例的方式说明运行结果

注意：此部分和“操作说明”是完全一样的，故在“操作说明”中不再说明，而直接引用该部分的内容。

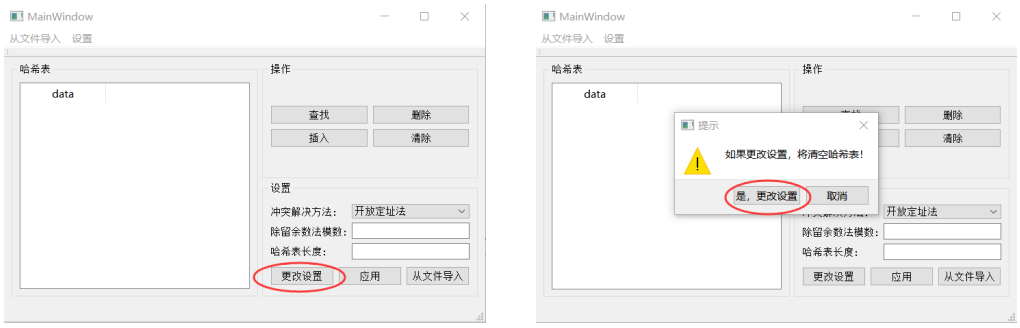
原始界面如1.4。

图 1.4 用户界面

点击“更改设置”，如1.5(a)，会弹出对话框，如1.5(b)，选择“是，更改设置”。

然后设置冲突解决方法、除留余数法的余数和哈希表大小，然后点击“应用”。

然后可以看到左侧已经有东西了。点击“从文件导入”，如1.7(a)。选择 txt 文件，如1.7(b)。txt 文件格式为：第一行是数据个数，从第二行开始，每行一个关键字。



(a) 点击更改设置　　(b) 确认设置

图 1.5 设置

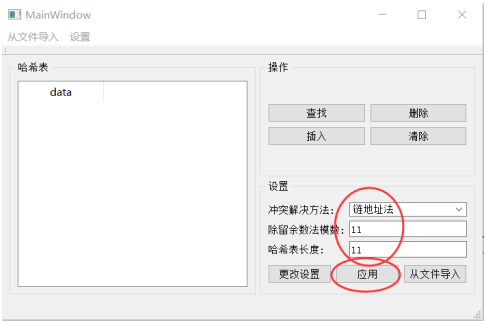
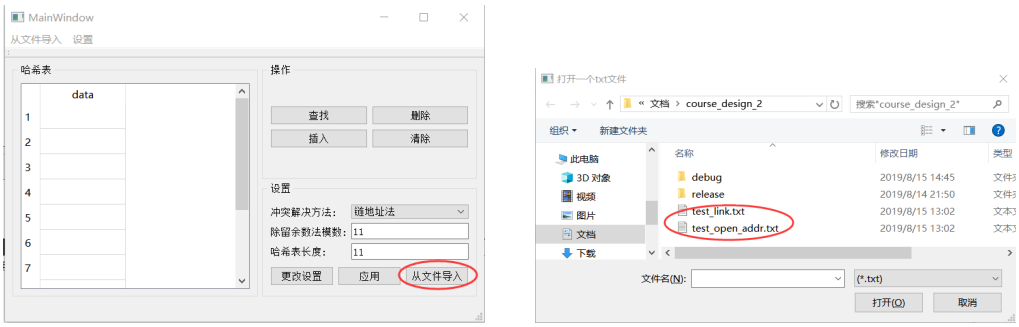


图 1.6 设置具体的方法



(a) 从文件导入　　(b) 选择文件

图 1.7 从文件导入

导入后效果如1.8。

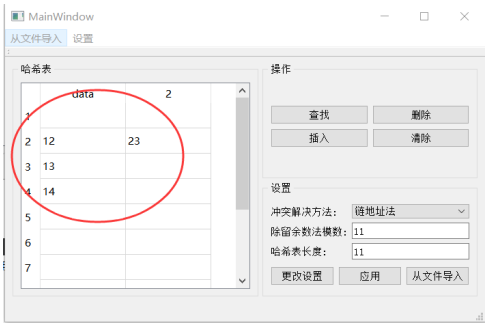
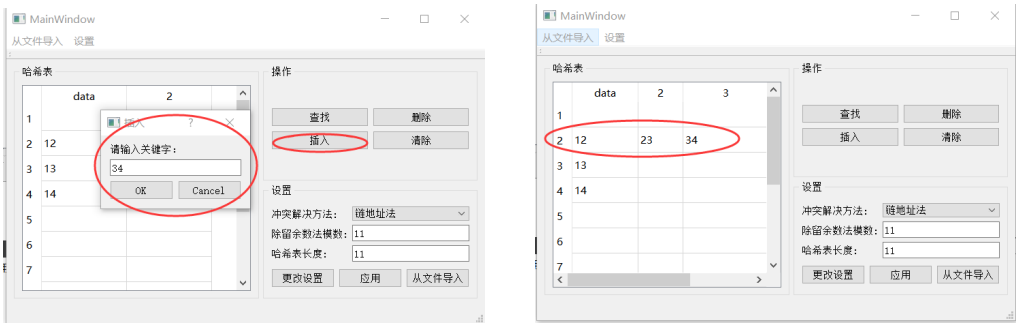


图 1.8 导入后效果

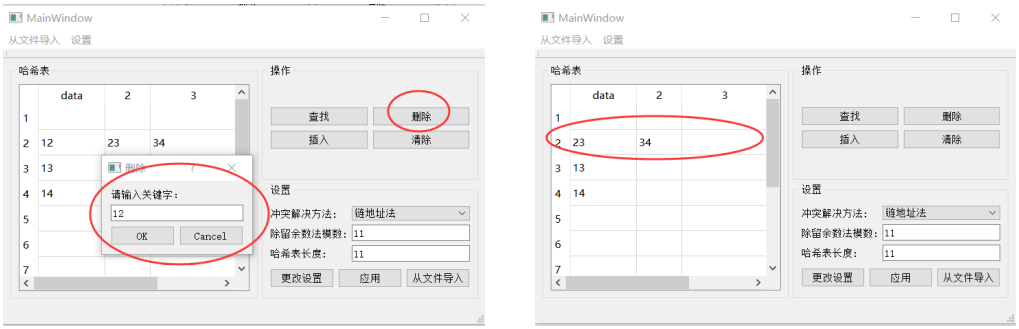
然后，点击“操作”栏中的“插入”，输入关键字，点击确定，如1.9(a)；效果如1.9(b)。可以看到 12、23 的关键字后增加了 34。



(a) 插入时输入关键字　　(b) 插入结果

图 1.9 插入

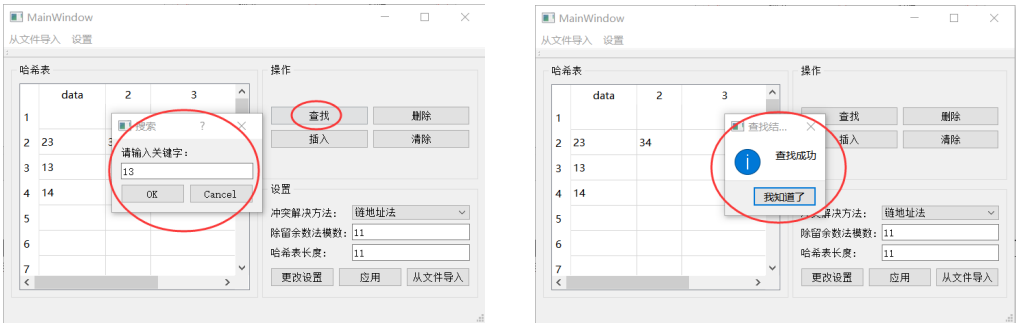
然后，点击“操作”栏中的“删除”，输入关键字，点击确定，如1.10(a)；效果如1.10(b)。可以看到关键字 12 被删除，23、34 顺次前移。如果不存在该关键字，则会给出错误提示，这已在上一节中展示。



(a) 删除时输入关键字　　(b) 删除结果

图 1.10 删除

然后，点击“操作”栏中的“查找”，输入关键字，点击确定，如1.11(a)；效果如1.11(b)。如若失败，则会提示查找失败，失败提示已在上一节中展示。



(a) 查找时输入关键字　　(b) 查找结果

图 1.11 查找

点击“清除”，弹出提示信息后点“我知道了”，即可清除哈希表。
若想再建立一张哈希表，那么点击“更新设置”，重复以上步骤即可。
另外，若想使用开放定址法，则设置冲突解决方法改为“开放地址法”即可。

1.7 操作说明

具体的操作说明已经在上一节1.6.3运行案例的方式说明运行结果中，较详细地列出。故此部分不再赘述。

2 综合应用设计说明

2.1 题目

在关系数据库中，所有数据对象都以表的形式存储，如需在关系数据库中存储树结构，需设置一指向父节点的属性来实现，该过程可以通过如下线性表节点的存储结构模拟：

```
struct Node {
  int id;//该线性表中所有节点的 ID 都唯一
  Elemtype data;//节点的值，自定义数据类型
  int pid;//表示该节点的父节点，值为 0 表示根，对应线性表中其他节点的 id 值
  Node *next;//指向线性表下一个节点
}
```

代码 2.1 线性表节点的存储结构

（1） 请设计一个算法，根据线性表中 pid 的指向，将该线性表中存储的节点转换为树形结构。
（2） 在树结构中插入一个节点，自动将其加入到线性表中。
（3） 在树结构中删除一个节点，自动更新线性表的结构。

2.2 软件功能

2.2.1 软件功能设计

（1） 能够将 txt 存储的数据导入到软件中
（2） 能够将树形结构和线性表结构用较好的视觉控件表达
（3） 在对树形结构的数据进行删除和增加子节点操作时，线性表要能自动更新

2.2.2 功能实现方式

QT 下的控件有 QTreeWidget 和 QTableWidget 分别能较好地从视觉上表达树形（左侧）和线性（右侧）结构，如2.1所示。

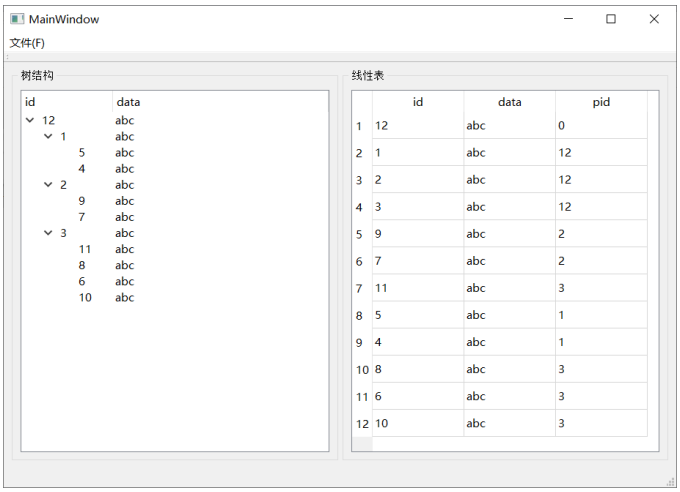


图 2.1 用户界面

因为 QT 的控件有比较好的包装，故通过如下代码配置：

```
addChildNodeAction = new QAction("&添加子节点", this);
deleteNodeAction = new QAction("&删除节点", this);connect(addChildNodeAction
,SIGNAL(triggered()),this,SLOT(addChildNode()));
connect(deleteNodeAction,SIGNAL(triggered()),this,SLOT(deleteNode()));
```

代码 2.2 小菜单设置代码

以及：

```
void MainWindow::on_treeWidget_customContextMenuRequested(const QPoint &pos
    )
{
  curItem = ui->treeWidget->itemAt(pos); //获取当前被点击的节点
  if(curItem != nullptr){
    QVariant var = curItem->data(0,Qt::UserRole);
    QMenu *popMenu =new QMenu(this);
    popMenu->addAction(addChildNodeAction);
    popMenu->addAction(deleteNodeAction);
    popMenu->exec(QCursor::pos());
  }
}
```

代码 2.3 小菜单设置代码

用以上两段代码就可以右键显示具体操作的菜单，如2.2所示。

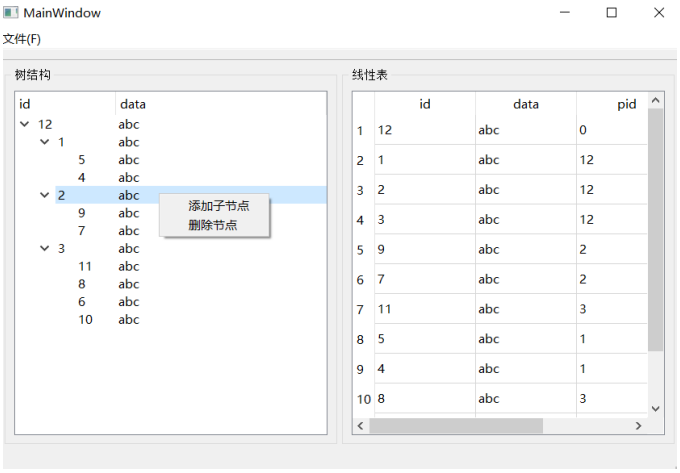


图 2.2 小菜单

还有其它的操作已经在上一章的算法实现中说明。

2.3 设计思想

2.3.1 软件实现思路

软件的实现主要分两部走：第一步，我先用纯 C++ 方式完整实现算法。环境是 Visual Studio 2017。第二步，使用 QT 下的控件实现标准交互式图形界面，加载输入框、按钮、功能菜单等内容。QT 控件较

好地封装了一些查找、插入、删除的函数，可供我们使用。我们可以将接下来的代码看作是和纯 C++ 方式的一一对应。

而第二步又可以分为两步：一、导入数据文件，将线性表型数据文件转为树形结构并显示；二、在树形结构上完成添加和删除操作时，同步对线性表执行该操作。

2.3.2 数据结构的选用与设计思想

题目已经限制了数据结构为树形和线性。但注意到，我需要一种方式来记录转换后的树。否则通过了链表生成了一颗树后，每当我们想用一种图形化界面来展现树的时候，都需要重新生成一遍，这是十分不合理的。

题目要求的线性表的节点存储结构如下：

```
1 struct Node {
2   int id;//该线性表中所有节点的 ID 都唯一
3   Elemtype data;//节点的值，自定义数据类型
4   int pid;//表示该节点的父节点，值为 0 表示根，对应线性表中其他节点的 id 值
5   Node *next;//指向线性表下一个节点
6 }
```

代码 2.4 线性表节点的存储结构

我经过思考，认为仅仅有这些属性是没有足够的信息一次性记录一颗树的父子节点信息，使得能在 $O(n)$ 的时间打印一颗树。

接下来，我们考虑选取哪种结构存储一棵树。在《数据结构》这门课[1] 中，我们知道一棵树有三种表达方式：

- 双亲表示法
- 孩子表示法
- 孩子兄弟表示法

双亲表示法在求节点的孩子时需要遍历整个结构，而 pid 就相当于双亲信息；孩子表示法将每个节点的孩子节点排列起来，看成线性表，但是不适用于求父节点的操作。孩子兄弟表示法这种存储结构便于实现各种树的操作，在这里就需要新增两个指针：struct Node *firstchild, nextsbling;

综合这三种传统方法，以及从代码的复杂程度上考虑，如果使用双亲表示 + 孩子表示法较为合理。int pid 已经给出了父节点信息，而孩子的表示可以使用 Vector 数据结构。

除此之外，我通过网上搜索，还看了许多其它语言如 Python、Javascript[2] 等的线性结构转树结构的代码，它们都使用了类似列表的数据结构来记录子节点，如以下代码第 17 行：

```
1  function Tree(datas){
2      var tree={},
3          parent_id;
4      for(var i=0;i<datas.length;i++){
5          var item=datas[i];
6          tree[item.id]=item
7      }
8      var root=null;
9      for(var i=0;i<datas.length;i++){
10         var obj=datas[i];
```

```
        if(obj.in==null){
root=tree[obj.id]       }else{
            parent_id=obj.in;
            if(!tree[parent_id].children){
                tree[parent_id].children=[]
            }
            tree[parent_id].children.push(tree[obj.id])
        }
    }
    return root;
}
```

代码 2.5 Javascript 线性结构转树结构代码

由以上分析，我认为，我使用双亲表示 + 孩子表示法是合理的。最后我选取的存储结构如下：

```
struct Node {
  int id;//该线性表中所有节点的 ID 都唯一
  Elemtype data;//节点的值，自定义数据类型
  int pid;//表示该节点的父节点，值为 0 表示根，对应线性表中其他节点的 id 值
  Node *next;//指向线性表下一个节点
  vector <Node *> children;// 指向孩子的指针
}
```

代码 2.6 最后选取的线性表存储结构

相对于题目要求的数据结构，新增了 vector <Node *> children 这一成员来记录指向孩子的指针。

2.3.3 算法设计基本流程

我设计了四个函数，分别针对我们需要的四个操作。

A. **线性链表初始化**

```
int initLinearList(Node *L)
{
  int n; //个数
  Node *p = L;
  int i;

  cin >> n;
  for (i = 0; i < n; i++) {
    p->next = new Node;
    p = p->next;
    cin >> p->id >> p->pid;
  }
  p->next = NULL;
  return 0;
```

```
15 }
```

代码 2.7 纯 C++ 实现的算法

这一部分主要是将输入的线性数据记录下来，生成一个链表。没有其它特殊的地方。

算法复杂度：$O(n)$

B. 将线性链表转为树形结构，记录子节点信息

```
1  int list2tree(Node *L)
2  {
3    Node *p = L->next;
4    Node *q;
5    int cur_pid;
6
7    while (p) {
8      cur_pid = p->pid;
9      if (cur_pid == 0) {
10       p = p->next;
11       continue;
12     }
13
14     q = L->next;
15     while (q) {
16       // 找到父亲节点
17       if (q->id == cur_pid) {
18         q->children.push_back(p);//将子节点的指针记录下来
19       }
20       q = q->next;
21     }
22     p = p->next;
23   }
24
25   return 0;
26 }
```

代码 2.8 纯 C++ 实现的算法

算法说明：主要部分是两个 while 循环。对于每一个节点，都遍历一次链表，将它的子节点的指针加入到 children 里。

算法复杂度：$O(n^2)$

C. 添加节点

```
1  int addNode(Node *L, Node &e)
2  {
3    Node *p;
4    Node *t = new Node;
```

```
  memcpy(&(t->data), &(e.data), sizeof(ElemType));
t->id = e.id;  t->pid = e.pid;

  t->next = L->next->next;
  L->next->next = t;
  p = t->next;

  while (p){
    if (p->id == e.pid) {
      p->children.push_back(t);
      return 0;
    }
    p = p->next;
  }
  return 0;
}
```

代码 2.9 纯 C++ 实现的算法

算法说明：主要部分是 while 循环。先将要添加的节点用头插法加入到链表中；再根据要添加的节点的 pid，遍历一次链表，找到它的父节点，然后将要添加的节点的指针加入到它的父节点的 children 里。

算法复杂度：$O(n)$

D. 删除节点

```
int deleteNode(Node *L, int id)
{
  Node *q = L;
  Node *p = L->next;
  while (p) {
    if (p->id == id) {
      q->next = p->next;

      for (auto i : p->children) {
        deleteNode(L, i->id);
      }
      delete p;

      p = q->next;

    }
    else {
      q = q->next;
      p = q->next;
    }
```

```
21   }
22 return 0;}
```

代码 2.10 纯 C++ 实现的算法

算法说明：主要部分是 while 循环和 for 循环。注意删除一个树节点的话，要连带着删除它的子节点，故在 for 循环中遍历要删除的节点的子节点，递归调用 deleteNode 删除它的子节点。

算法复杂度：$O(n^2)$

E. 展示树结构

```
1  int printTree(Node *p, int level)
2  {
3    if (p->children.size() == 0) {
4      return 0;
5    }
6
7    for (auto i : p->children){
8      for (int j = 0; j < level; j++) {
9        cout << " ";
10     }
11     cout << i->id << endl;
12     printTree(i, level + 1);
13   }
14
15   return 0;
16 }
```

代码 2.11 纯 C++ 实现的算法

算法说明：递归调用，通过 id 号前面的空格的多少来展示层级关系。

算法复杂度：$O(n)$

控制台效果如2.3：

图 2.3 展示树结构

2.4 逻辑结构与物理结构

同样，这部分我分为两块说明，一部分是纯 C++ 方式，另一部分是 QT 方式。

2.4.1 纯 C++ 方式

逻辑结构：树形结构。因为题目要求是将线性表转为树形结构。
物理结构：链式结构。结构如下：

```
struct Node {
  int id;//该线性表中所有节点的 ID 都唯一
  Elemtype data;//节点的值，自定义数据类型
  int pid;//表示该节点的父节点，值为 0 表示根，对应线性表中其他节点的 id 值
  Node *next;//指向线性表下一个节点
  vector <Node *> children;// 指向孩子的指针
}
```

代码 2.12 线性表节点的存储结构

2.4.2 QT 方式

逻辑结构：树形结构和线性结构。因为题目要求是将线性表转为树形结构，并且我需要展示线性结构。
物理结构：链式结构。主要是 QTableWidgetItem 和 QTreeWidgetItem，它们主要的成员如下：

```
class QTreeWidgetItem {
  QTreeWidgetItem * parent;
  QList <QTreeWidgetItem *> children;
  QString text;
}
```

代码 2.13 QTreeWidgetItem

```
class QTableWidgetItem {
  int row;
  int column;
  QString text;
}
```

代码 2.14 QTableWidgetItem

2.5 开发平台

开发平台：QT 5.13.0 (MSVC 2017)
运行环境：QT Creator 4.9.2 Community
其它的库：C++ STL (纯 C++ 方式实现时)

2.6 系统的运行结果分析说明

2.6.1 应用开发工具进行调试及开发

QT 提供了非常方便的 QDebug 库，可以很方便地在控制台输出结果，例如在一些关键位置写下以下这些代码等：

```
1 qDebug() << "Err";
2 qDebug() << "deleteNode";
```

代码 2.15 应用开发工具进行调试及开发

然后打开 debug 调试[3]，就可以看到如2.4这些提示：

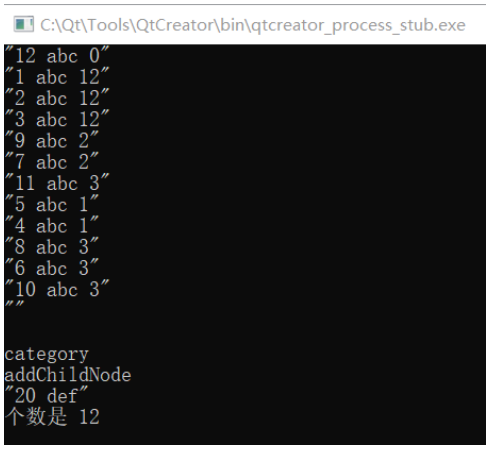


图 2.4 调试

这大大加快了开发的速度。

开发中比较重要的 QT 控件是 QTreeWidget 和 QTableWidget[4]，以及 QT 的核心信号与槽机制[5]。但由于不是本课设核心知识点，故不作说明，仅列出参考文献。

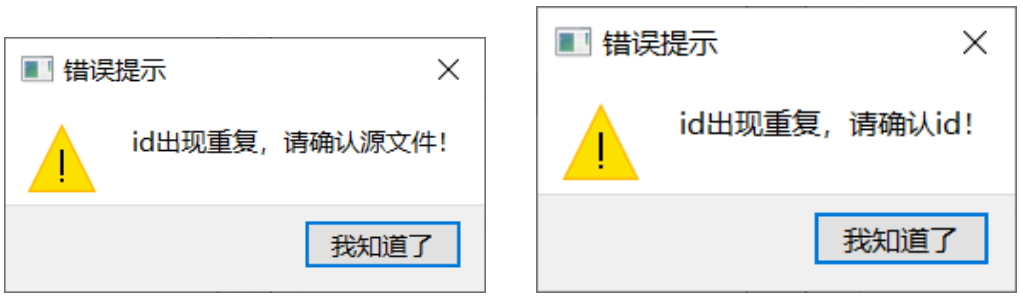
2.6.2 开发软件到达的成果

由于本项目较为简单，运行结果的正确性和稳定性可以得到较为直观的说明。

容错能力中，主要有以下几点：

- 输入文件存在 id 重复
- 插入的节点 id 重复

当遇到以上这两种情况时，会跳出错误提示，如2.5。



(a) 源文件出现重复　　(b) 插入节点 id 重复

图 2.5 错误提示

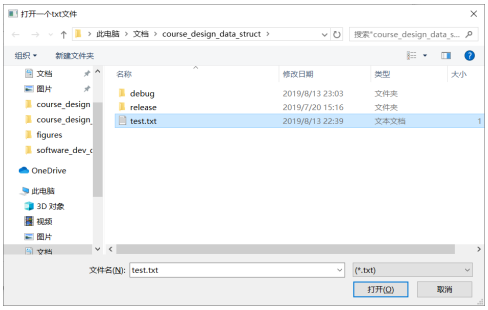
2.6.3 运行案例的方式说明运行结果

首先，我们先要准备一个 txt 文件，格式如下：第一行是节点个数，第二行是根节点的 id,data,pid。pid=0 表示为根。从第三行开始是其它节点信息，依次为 id，data，pid。如2.6。

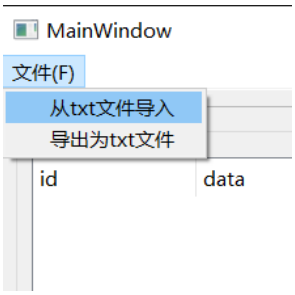
```
12
12 abc 0
1 abc 12
2 abc 12
3 abc 12
9 abc 2
7 abc 2
11 abc 3
5 abc 1
4 abc 1
8 abc 3
6 abc 3
10 abc 3
```

图 2.6 导入数据格式

选中准备好的 txt 文件，如2.7(a)。点击文件-> 从 txt 文件导入，在导航中选择准备好的 txt 文件，如2.7(b)。



(a) 选中 txt 文件



(b) 通过 txt 文件导入链表数据

图 2.7 删除节点

然后就可以看到如2.8界面。可通过拖拉窗口使控件显示完整。

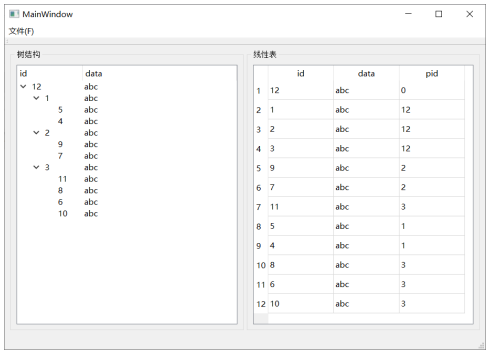


图 2.8 导入后的结果

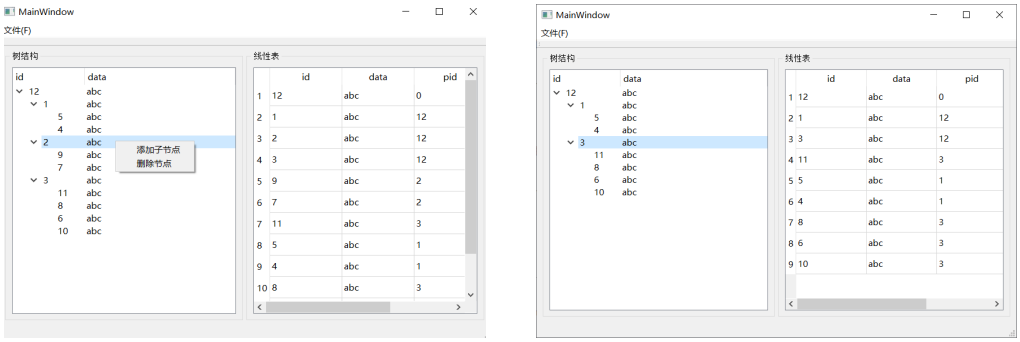
A. **删除节点**

在要删除的节点上右键，选择删除节点，如2.10(a)。单击删除节点后，可以看到左侧树结构中，所有该节点以及该节点的子节点都删除了；而在右侧，对应的节点也都同步删除了。如2.9(b)，id 为 2 的节点及其 id=9 和 7 的节点都被删除了。

B. **添加子节点**

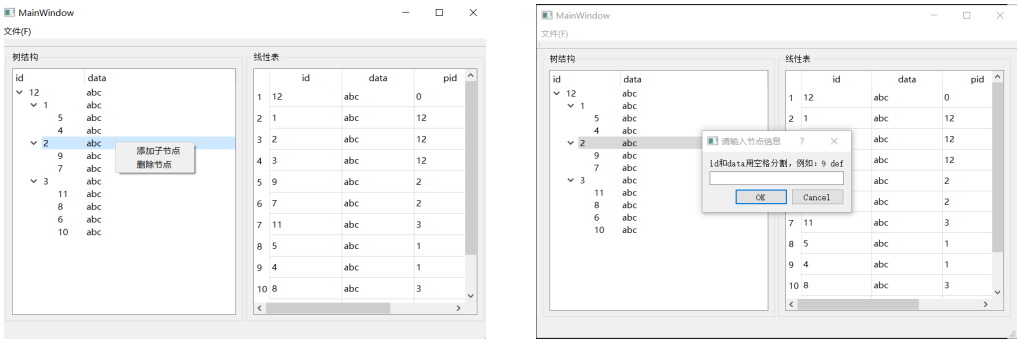
在要添加子节点的节点上右键，选择添加子节点节点，如2.10(a)。之后会弹出对话框，输入 id 和 data，用空格分割，如2.10(b)。

这里我们输入 20 def，点击 OK。然后就可以看到树结构中 id=2 的节点有了 id=20，data=def 的节点，右侧线性表的尾部新增了一个 id=20，data=def 的节点，如2.11 中红圈中所示，从而做到线性表和树形结构同步变化。



(a) 选择删除节点　　(b) 删除节点后

图 2.9 删除节点



(a) 选择添加节点　　(b) 添加子节点

图 2.10 添加子节点

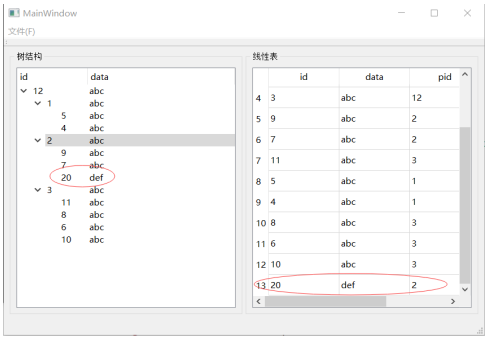


图 2.11 添加子节点

2.7 操作说明

具体的操作说明已经在上一节2.6.3运行案例的方式说明运行结果中，较详细地列出。

3 实践总结

3.1 所做的工作

在这次课程设计中，我主要完成了两款软件。一款能够模拟哈希表的建立、删除、新增、查找操作。一款能够模拟线性表转为树形结构的操作。

在实践过程中，我复习了哈希表的各种哈希函数的选取和冲突的解决方法；哈希表的建立、查找、插入和删除操作。我通过查阅资料，分析选取了较好的线性表转树形结构的数据结构和方法。

还有就是自学了 QT，了解掌握了 QT 的基本信号与槽的原理，以及部分关键控件如 QTreeWidget、QTableWidget 等的操作。

3.2 总结与收获

这是我第一次自己动手完成一款软件。在一开始选取平台的时候我纠结了很久。QT 相对来说上手比较慢，学习曲线较为陡峭。但是考录到 QT 良好的跨平台的特性，我还是选择了 QT。我在前期花了大量的时间熟悉 QT，了解 QT 的基本原理和核心机制。在后续的实现中，我又陆陆续续碰到不少问题。

比如，综合应用当中，我做的是 QTreeWidget 和 QTableWidget 时时保持同步，这就牵扯到信号的传递问题。我原来各个自定义控件封装较好，但是为了能够保证同步，而个人水平又有限，还是把一些封装的东西拆开了。

最后该份报告是使用同济大学本科毕业论文 latex 模板[6] 完成的，在图片的位置调整上花了挺大的功夫，就目前而言图片往往会和它的说明文字相隔较远的距离。这是 latex 的内部机制导致的; 另外还存在引用的代码会侵占页眉和页脚的问题，我一直尝试解决，但恕水平有限，无法完全解决，仍不美观，望老师见谅！

该套同济大学本科毕业论文 latex 模板中，加载了开源的参考文献的国标。所以只需手动把必要的信息放上即可。在我们的《数据结构》课程设计计划及题目当中，参考文献没有符号标明文献类别，是不规范的，也不符合《同济大学毕业设计（论文）模板（理工类）》的规定。

经过此次课设后，我自学能力提高了很多。从完全不懂 QT 到基本掌握 QT，其实是搜索了无数次网页的。另外 QT 还有较为完善的文档[4]，大大地方便了我的学习。这次课设还帮我复习了很多数据结构的知识，我认为我在这次课设中收获很大。

参考文献

[1] 严蔚敏, 吴伟民. [M]. 北京: 清华大学出版社, 2018.

[2] XINQILELA. 将给定格式的线性结构转换为树形结构[EB/OL]. China: CSDN(2017-11-25)[2019-08-15]. https://blog.csdn.net/xinqilela/article/details/78631074.

[3] LUSANSHUI. Qt 程序显示控制台调试信息[EB/OL]. China: CSDN(2018-12-10)[2019-08-15]. https://blog.csdn.net/lusanshui/article/details/84935278.

[4] QTGROUP. Qt documentation[EB/OL]. America: Qt Group(2019)[2019-08-15]. https://doc.qt.io/qt-5/qtwidgets-module.html.

[5] ZERO. Qt5 开发学习总结 (一)——信号与槽机制[EB/OL]. China: CSDN(2018-02-01)[2019-08-15]. https://blog.csdn.net/kilotwo/article/details/79231759.

[6] SXKDZ. Latex template for tongji undergaduate thesis[EB/OL]. America: GitHub Inc.(2019-04)[2019-08-15]. https://github.com/SXKDZ/tongjithesis.

[7] ZHANGPY. 【Qt 开发】Qt5.7 中文显示乱码解决方法两种[EB/OL]. China: CSDN(2016-09-06)[2019-08-15]. https://blog.csdn.net/LG1259156776/article/details/52450483.

[8] URIELCHIANG. Qt 添加新类时出现 error: LNK2019: 无法解析的外部符号解决[EB/OL]. China: CSDN(2018-03-27)[2019-08-15]. https://blog.csdn.net/uriel_chiang/article/details/79708288.