# 实验四

## 1. 问题定义及需求分析

### 1.1 问题描述

##### 利用平衡二叉树设计动态查找表。

### 1.2 实验要求

##### 设计平衡二叉树的动态演示的模拟程序。

###### （1）采用平衡二叉树存储结构。

###### （2）完成平衡二叉树的创建、查找、插入和删除的演示操作。

###### （3）可以考虑两棵平衡二叉树的合并。

### 1.3 数据形式

#### 1.3.1 输入的形式

##### 通过鼠标点击窗体元素输入数据输入值的范围：树中元素的值为int型。

#### 1.3.2 输出的形式

##### 输出到QT窗口

### 1.4 程序的功能

##### 创建平衡二叉树存储结构，通过平衡因子，使二叉排序树达到平衡，提供平衡二叉树的创建、查找和删除，树中元素的查找、插入和删除等基本功能，可以实现创建多棵平衡二叉树，并且能够进行多棵树的合并。通过平衡二叉树的性质，能够使树时刻保持平衡，从而提高在树中遍历数据的速度，具有重要实际意义。

### 1.5 测试数据

#### 1.5.1 输入

##### 输入任意数字，创建新树或添加微现有树的新节点。

#### 1.5.2 输出

##### 在窗口上方显示相应的结果。

## 2.概要设计

### 2.1 抽象数据类型定义

##### ADT AVLTREE

##### {

##### Data int类型；

##### 数据元素操作

##### int calheight(struct AVLNode \*);//计算数的高度

##### int bf(struct AVLNode \*);//计算根节点的平衡因子

##### struct AVLNode \* llrotation(struct AVLNode \*);//LL旋转

##### struct AVLNode \* lrrotation(struct AVLNode \*);//LR旋转

##### struct AVLNode \* rlrotation(struct AVLNode \*);//Rl旋转

##### struct AVLNode \* rrrotation(struct AVLNode \*);//RR旋转

##### struct AVLNode\* insert(struct AVLNode \*,int);//插入节点

##### struct AVLNode\* deleteAVLNode(struct AVLNode \*,int);//删除节点

##### struct AVLNode\* inpre(struct AVLNode\*);//求某个节点的前驱

##### struct AVLNode\* insuc(struct AVLNode\*);//求某个节点的后继

##### struct AVLNode\* find(struct AVLNode\*,int);//查找某个节点

##### }

### 2.2 主程序流程

##### Qt的主程序比较特殊，只是调用主窗口控件。

### 2.3 模块调用关系



## 3.详细设计

### 3.1 定义数据类型及存储结构

/\*平衡树结构声明\*/  
struct AVLNode  
{  
 struct AVLNode \*left,\*right;//左右孩子  
 int data,height;//节点数据和数的高度  
 QLabel lab,Llink,Rlink;  
};  
int calheight(struct AVLNode \*);//计算数的高度  
int bf(struct AVLNode \*);//计算根节点的平衡因子  
struct AVLNode \*llrotation(struct AVLNode\*);//LL旋转  
struct AVLNode \*lrrotation(struct AVLNode\*);//LR旋转  
struct AVLNode \*rlrotation(struct AVLNode\*);//Rl旋转  
struct AVLNode \*rrrotation(struct AVLNode\*);//RR旋转  
struct AVLNode \*insert(struct AVLNode\*,int);//插入节点  
struct AVLNode \*deleteAVLNode(struct AVLNode\*,int,struct AVLNode\*);//删除节点  
struct AVLNode \*inpre(struct AVLNode\*);//求某个节点的前驱  
struct AVLNode \*insuc(struct AVLNode\*);//求某个节点的后继  
struct AVLNode \*FIND(struct AVLNode\*,int);//查找某个节点

### 3.2 每个函数及操作的代码 （代码）

##### (1)计算树的高度

int calheight(struct AVLNode \*p)  
{  
 if(p==NULL)return 0;  
 if(p->left&&p->right)  
 {  
 if(p->left->height<p->right->height)  
 return p->right->height+1;  
 else  
 return p->left->height+1;  
 }  
 else if(p->left&&p->right==NULL)  
 return p->left->height+1;  
 else if(p->left==NULL&&p->right)  
 return p->right->height+1;  
 return 1;  
}

##### (2)计算平衡因子

int bf(struct AVLNode \*n)  
{  
 if(n==NULL)return 0;  
 if(n->left&&n->right)  
 return n->left->height-n->right->height;  
 else if(n->left&&n->right==NULL)  
 return n->left->height;  
 else if(n->left==NULL&&n->right)  
 return -n->right->height;  
 return 0;  
}

##### (3)LL旋转

struct AVLNode \* llrotation(struct AVLNode \*n)//单向左旋平衡处理  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 p=n;  
 tp=p->left;  
  
 p->left=tp->right;  
 tp->right=p;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
  
 return tp;  
}

##### (4)RR旋转

struct AVLNode \* rrrotation(struct AVLNode \*n)//单向右旋平衡处理  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 p=n;  
 tp=p->right;  
  
 p->right=tp->left;  
 tp->left=p;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
  
 return tp;  
}

##### (5)RL旋转

struct AVLNode \* rlrotation(struct AVLNode \*n)//双向旋转（先右后左）  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 struct AVLNode \*tp2;  
 p=n;  
 tp=p->right;  
 tp2=p->right->left;  
  
 p->right=tp2->left;  
 tp->left=tp2->right;  
 tp2->left=p;  
 tp2->right=tp;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
 tp2->height=calheight(tp2);  
  
 return tp2;  
}

##### (6)LR旋转

struct AVLNode \* lrrotation(struct AVLNode \*n)//双向旋转（先左后右）  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 struct AVLNode \*tp2;  
 p=n;  
 tp=p->left;  
 tp2=p->left->right;  
  
 p->left=tp2->right;  
 tp->right=tp2->left;  
 tp2->right=p;  
 tp2->left=tp;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
 tp2->height=calheight(tp2);  
  
 return tp2;  
}

##### (7)插入新节点

struct AVLNode \*insert(struct AVLNode \*r,int data)  
{  
 qDebug()<<"insert addr"<<r;  
 if(r==NULL)//如果当前位置为空，就在此处插入新节点  
 {  
 struct AVLNode \*n;  
 n=new struct AVLNode;  
 n->data=data；  
 r=n;  
 r->left=r->right=NULL;  
 r->height=1;  
 return r;  
 }  
 else  
 {  
 if(data<r->data)  
 r->left=insert(r->left,data);  
 else if(data>r->data)  
 r->right=insert(r->right,data);  
 }  
  
 r->height=calheight(r);//新的节点已经在某个子树中插入，递归更新当前节点的高度  
  
 //如果不平衡，进行旋转  
 if(bf(r)>1&&bf(r->left)>0)  
 r=llrotation(r);  
 else if(bf(r)>1&&bf(r->left)<=0)  
 r=lrrotation(r);  
 else if(bf(r)<-1&&bf(r->right)<=0)  
 r=rrrotation(r);  
 else if(bf(r)<-1&&bf(r->right)>0)  
 r=rlrotation(r);  
  
  
 return r;  
  
}

##### (8)查找节点

struct AVLNode \*FIND(struct AVLNode \*r,int data)  
{  
 if(r==NULL)return NULL;  
 if(data<r->data)  
 return FIND(r->left,data);//在左子树中继续查找  
 else if(data>r->data)  
 return FIND(r->right,data);//在右子树中继续查找  
 else return r;  
}

##### (9)删除节点

struct AVLNode \*deleteAVLNode(struct AVLNode \*p,int data,struct AVLNode \*root)  
{  
  
 if(p->left==NULL&&p->right==NULL)//如果当前节点是叶子节点，直接删除  
 {  
 if(p==root)//如果是AVL树的根  
 root=NULL;  
 delete p;  
 return NULL;  
 }  
  
 struct AVLNode \*q;  
 //递归调用删除函数来在左右子树寻找节点  
 if(p->data<data)  
 p->right=deleteAVLNode(p->right,data,root);  
 else if(p->data>data)  
 p->left=deleteAVLNode(p->left,data,root);  
 else//找到了要删除的节点  
 {  
 if(p->left!=NULL)//如果左子树非空，将当前节点的值与左子树中的最大值交换  
 {  
 q=inpre(p->left);  
 p->data=q->data;  
 //p->lab.setText(QString::number(p->data));  
 p->left=deleteAVLNode(p->left,q->data,root);  
 }  
 else//右子树非空，将当前节点的值与右子树中的最小值交换  
 {  
 q=insuc(p->right);  
 p->data=q->data;  
 //p->lab.setText(QString::number(p->data));  
 p->right=deleteAVLNode(p->right,q->data,root);  
 }  
 }  
  
 p->height=calheight(p);  
  
 qDebug()<<"before raotation node"<<p->data<<"height"<<p->height<<"bf"<<bf(p);  
  
 if(bf(p)>1&&bf(p->left)>0)  
 p=llrotation(p);  
 else if(bf(p)>1&&bf(p->left)<=0)  
 p=lrrotation(p);  
 else if(bf(p)<-1&&bf(p->right)<=0)  
 p=rrrotation(p);  
 else if(bf(p)<-1&&bf(p->right)>0)  
 p=rlrotation(p);  
  
 qDebug()<<"after rotation node"<<p->data<<"height"<<p->height<<"bf"<<bf(p);  
  
 return p;  
}

##### (10)求某节点的前驱

struct AVLNode \*inpre(struct AVLNode\* p)  
{  
 while(p->right!=NULL)  
 p=p->right;//找到左子树上数据最大的节点  
 return p;  
}

##### (11)求某节点的后继

struct AVLNode \*insuc(struct AVLNode\* p)  
{  
 while(p->left!=NULL)  
 p=p->left;//找到右子树上数据最小的节点  
 return p;  
}

## 4.调试分析

### 4.1 遇到的问题及分析

#### 4.1.1 问题

##### 平衡二叉树中最难处理的是平衡问题，如何保证所创建的树是一个平衡树是关键问题。考虑可以通过递归调用的方式，通过层层递归，来调节各层节点的平衡因子。

##### 在调试过程中经常出现平衡因子数值不对的问题，经过仔细检查，发现是某些部分的代码不能适用于全部的可能情形，当一种新情形出现时，就会产生错误。经过不断的调试和改进，对这部分代码添加if判断，来区分各种不同的情形，进行不同的平衡因子调节和平衡化操作，从而解决了这一问题，通过目前测试来看，程序对于各种数据都有很好的稳定性。

### 4.2 算法时空分析

#### 4.2.1 时间

##### (1)平衡化操作：对于每个节点仅需要进行简单的判断，因此每次平衡复杂度为O(1)；但是考虑到最坏情况，不衡从叶子节点开始，那就需要对每一层都进行平衡化操作，因此总体时间复杂度微O(log(n))

##### (2)删除节点操作：

###### (a)需要查找到该节点位置，考虑树的深度，时间复杂度为O(log(n))。

###### (b)需要查找节点的中序前驱或后继上，时间复杂度又为O(log(n))。

###### (c)删除后，可能需要平衡化，复杂度又为O(log(n))。

###### 综上，时间复杂度为O(log(n))。

##### (3)添加节点操作：

###### (a)需要找到该节点应处的叶子节点位置，时间复杂度为O(log(n))。

###### (b)添加后，可能需要平衡化，复杂度为O(log(n))。

###### 综上，时间复杂度为O(log(n))。

##### (4)合并树：

###### (b)对除了第一棵树以外的每一棵树上的每个节点，都需要需要找到插入位置，时间复杂度为O(n\*log(n))。（n是所有树的节点数之和）

#### 4.2.2 空间

##### (1) 平衡化操作：不需要开辟空间，为O(1)

##### (2) 删除节点操作：O(1)

##### (3) 添加节点操作：O(1)

##### (4) 合并树：直接合并到根节点上，不另开辟空间O(1)

#### 4.2.3 改进

##### 能否在查找删除节点和合并树的操作上进行算法的优化改进，使得查找效率和合并效率更高。

### 4.3 经验和体会

##### 递归调用对于平衡树的平衡调节具有非常重要的作用，这大大减少了代码量，并且使得代码更加简洁。然而递归调用的过程却是非常复杂不容易理解的，稍不留神就有可能出现错误的调用，由于是层层递归，所以对函数运行情况的跟踪并不容易，十分容易出现错误。因此，为了减少错误，应当在编写前弄清楚整个函数所要实现的功能，并且尽量将代码写的规范，以便于调试的时候进行修改。

## 5.使用说明

##### (1)在主界面左下角输入数字。

##### (2)点击”NewTree”建立新树。

##### (3)在右下角表格中点击不同的树可以切换上方窗口中显示的树。

###### (a)在上方已经有树显示的情况下，点击”Find”以在这棵树中查找节点。若找到，该节点会变黄。

###### (b)在上方已经有树显示的情况下，点击”Insert”以在这棵树中插入节点。

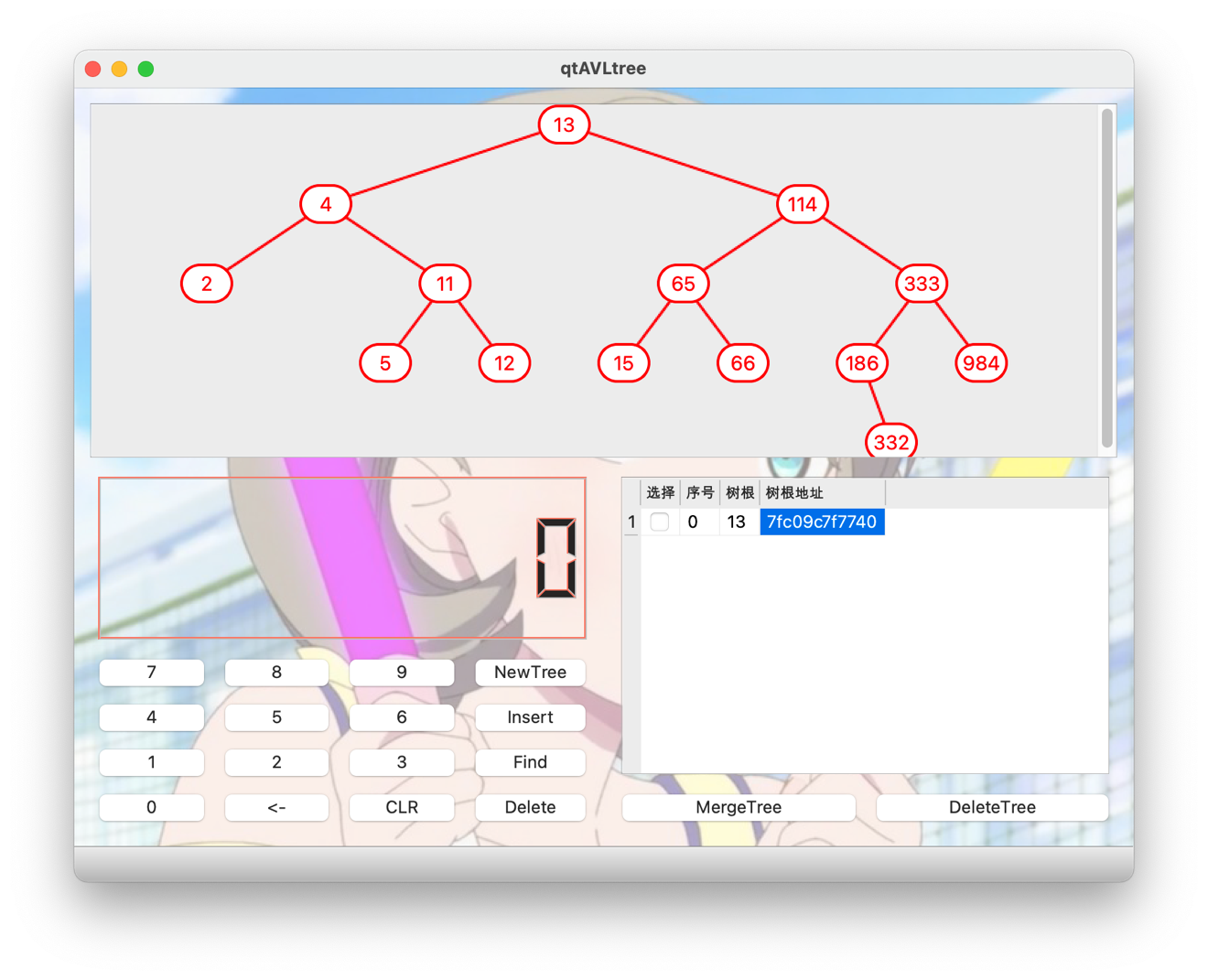
###### (c)在上方已经有树显示的情况下，点击”Delete”以在这棵树中删除节点。

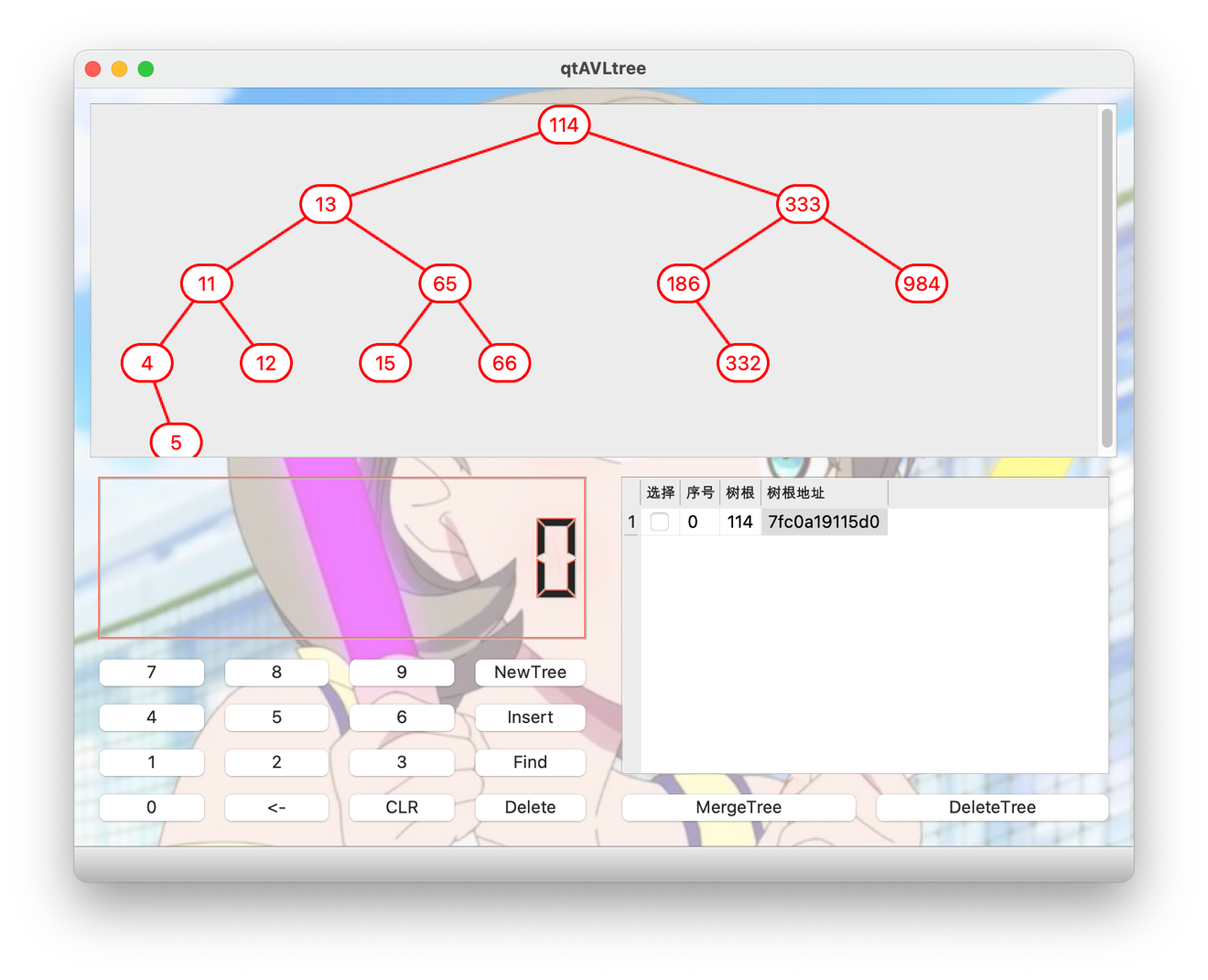
##### (4)在右下角表格的第一列有若干选择框。

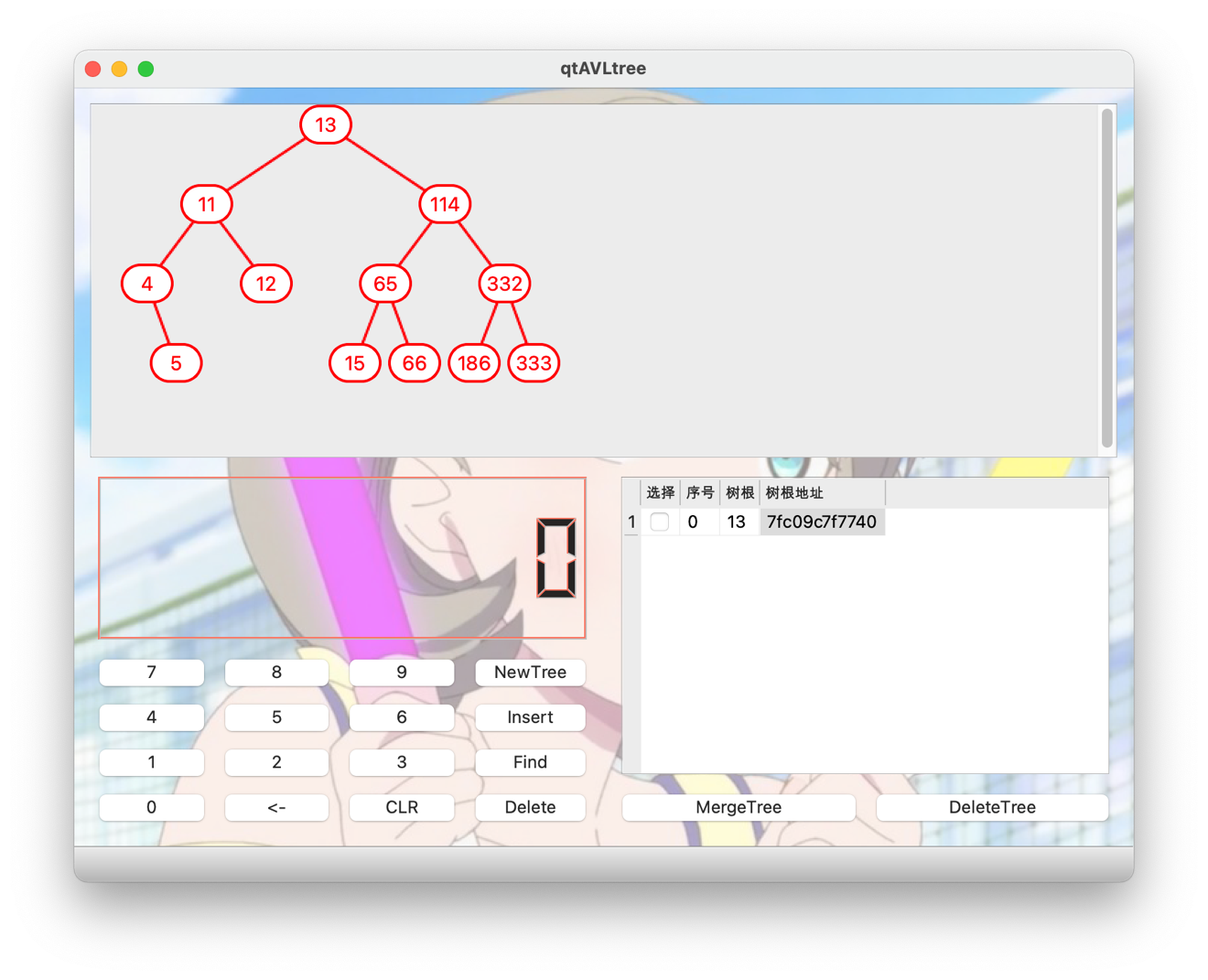
###### (a)如果勾选了某些树后再点击”DeleteTree”按钮，这些树都会被删除。

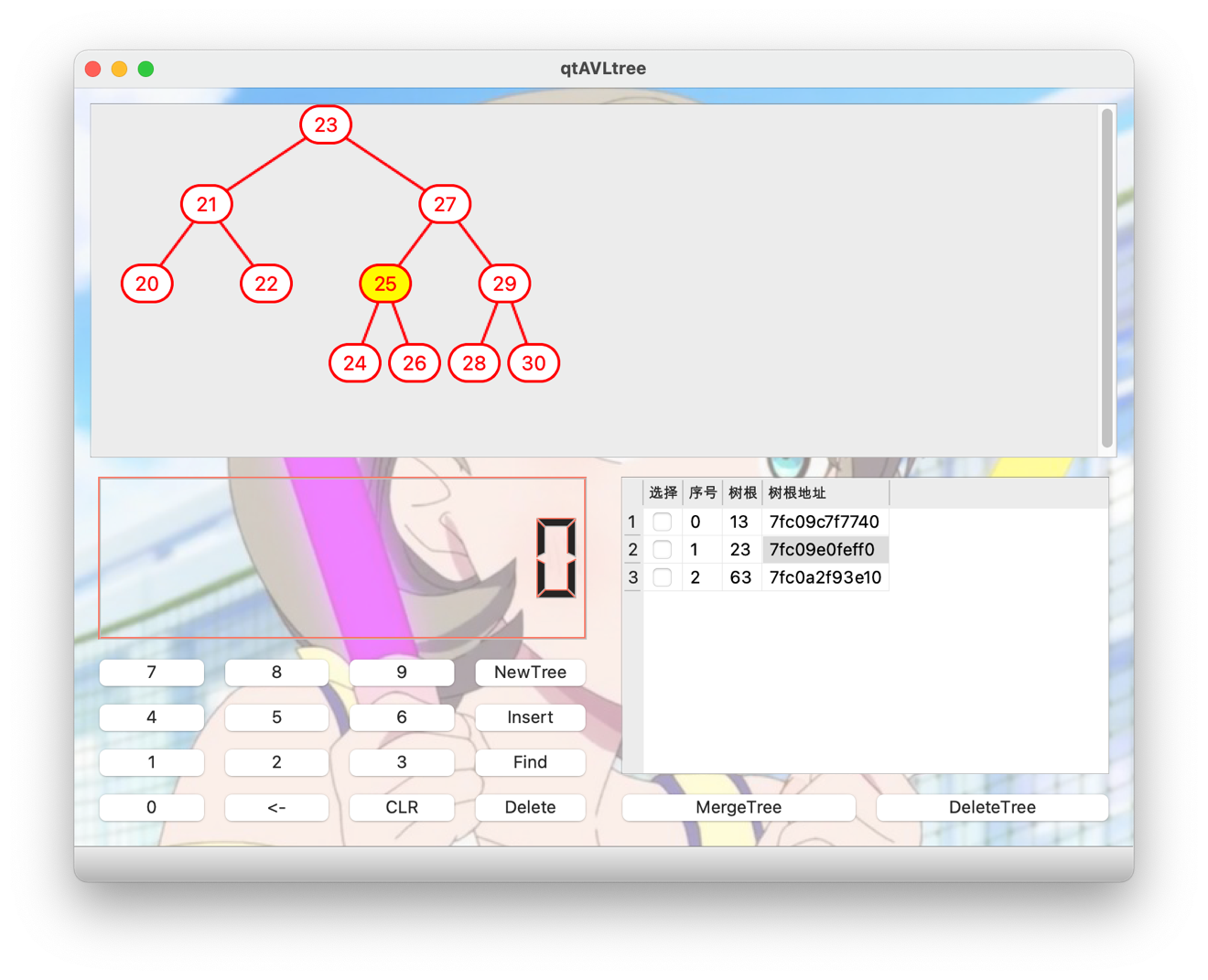
###### (b)如果勾选了大于1棵树再点击”MergeTree”按钮，这些树都会被自动合并到第一棵。

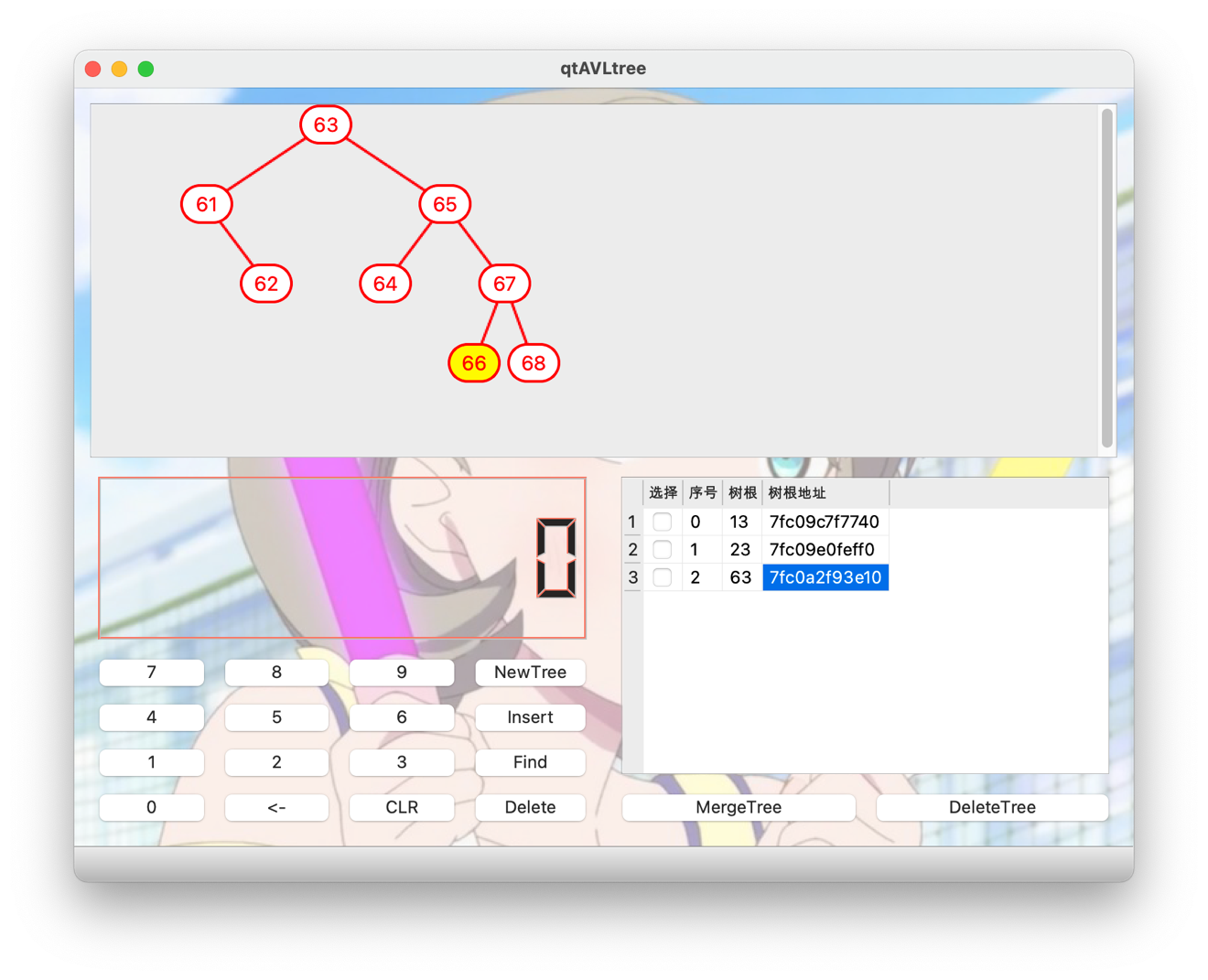
## 6.测试结果

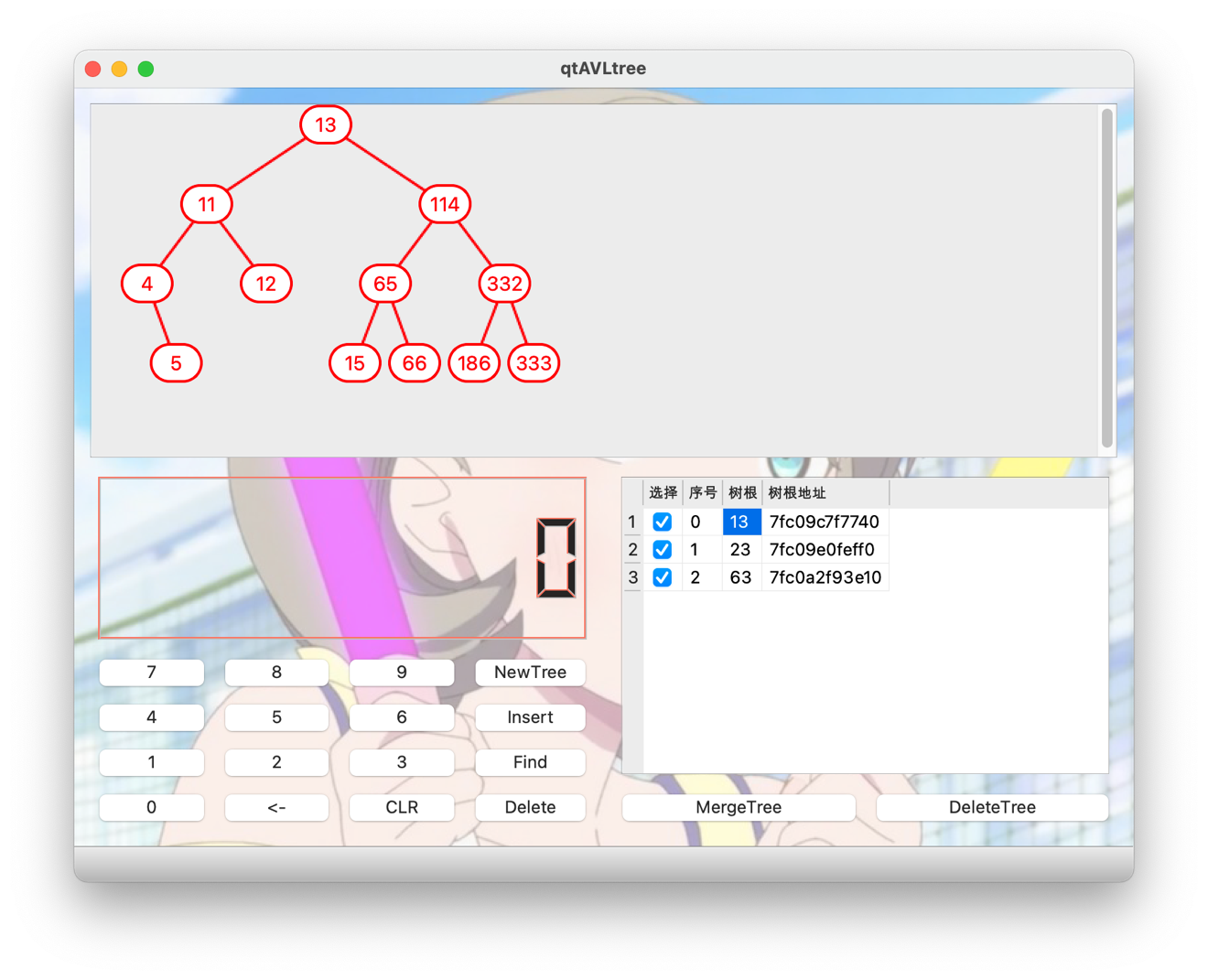


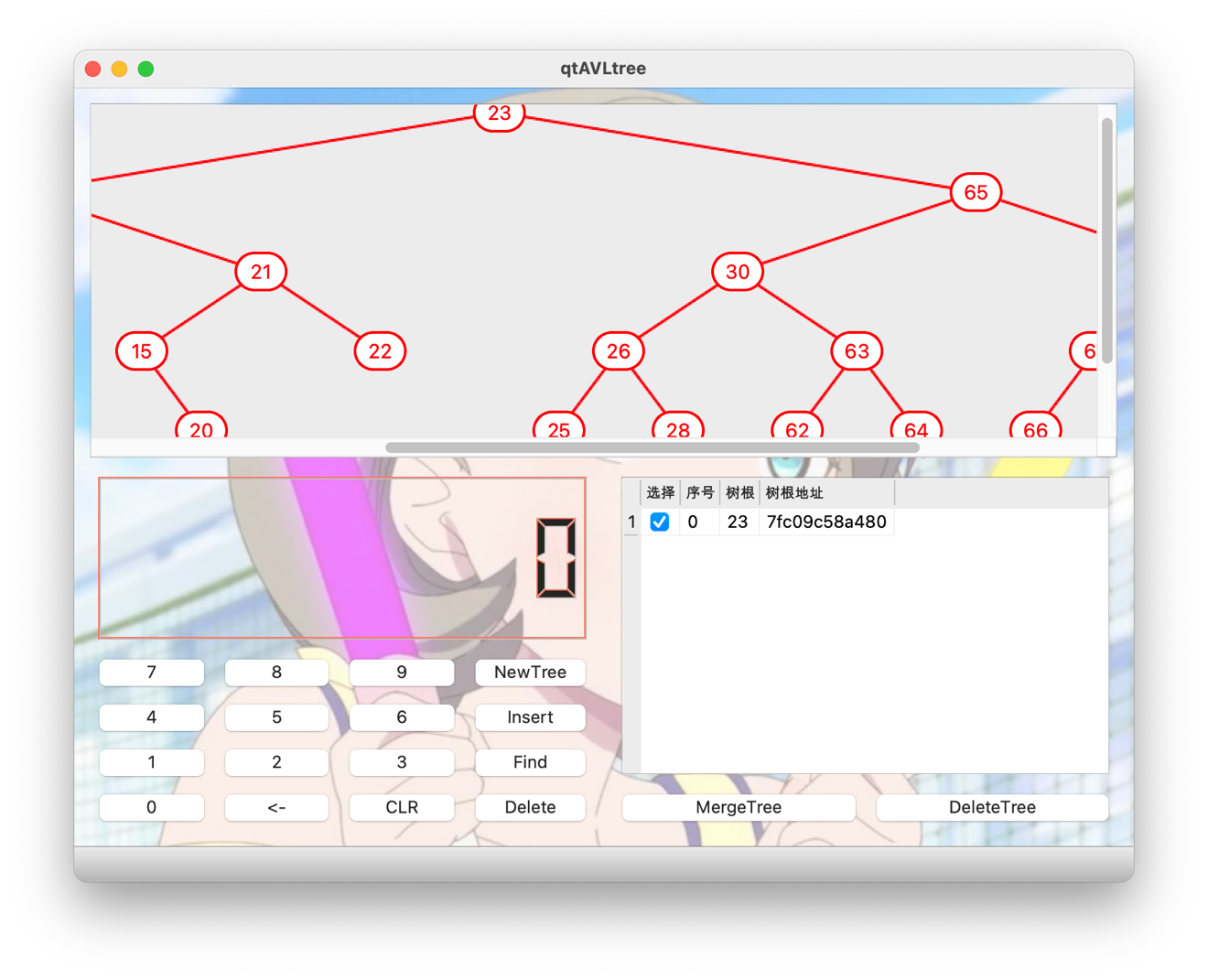


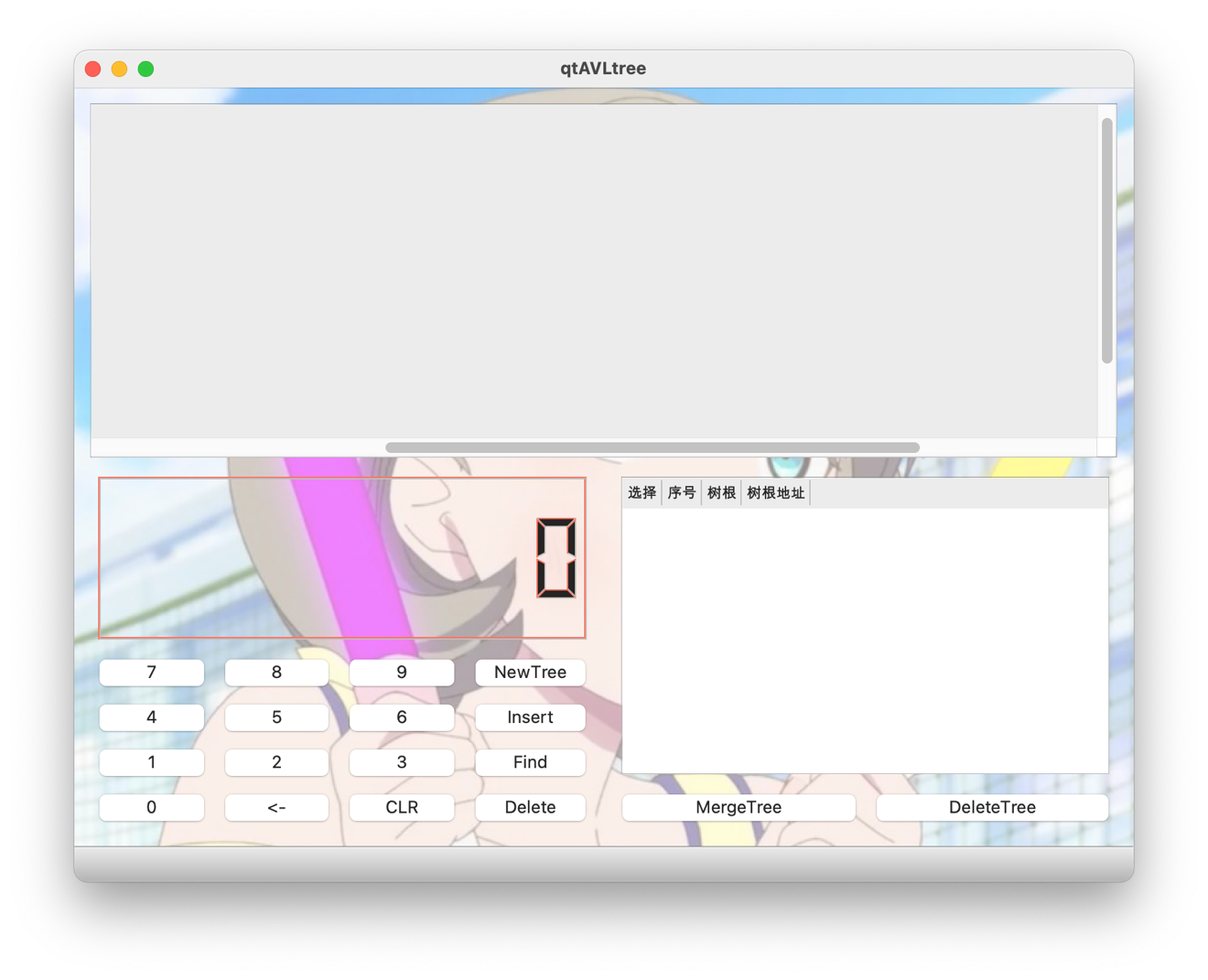












## 7.附录

### 7.1 个人负责的部分

##### 图形界面、树的高度计算、树的平衡因子计算、节点删除。

### 7.2 整个程序

#### 7.2.0 tree.pro

QT += core gui  
  
greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets  
  
CONFIG += c++11  
  
# You can make your code fail to compile if it uses deprecated APIs.  
# In order to do so, uncomment the following line.  
#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0  
  
SOURCES += \  
 avltree.cpp \  
 main.cpp \  
 mainwindow.cpp  
  
HEADERS += \  
 avltree.h \  
 mainwindow.h  
  
FORMS += \  
 mainwindow.ui  
  
# Default rules for deployment.  
qnx: target.path = /tmp/$${TARGET}/bin  
else: unix:!android: target.path = /opt/$${TARGET}/bin  
!isEmpty(target.path): INSTALLS += target  
  
DISTFILES +=  
  
RESOURCES += \  
 res.qrc

#### 7.2.1.1 avltree.h

#ifndef AVLTREE\_H  
#define AVLTREE\_H  
  
#include <qlabel.h>  
#include <qdebug.h>  
  
struct AVLNode  
{  
 struct AVLNode \*left,\*right;  
 int data,height;  
 QLabel lab,Llink,Rlink;  
};  
  
int calheight(struct AVLNode \*);  
int bf(struct AVLNode \*);  
struct AVLNode \*llrotation(struct AVLNode\*);  
struct AVLNode \*lrrotation(struct AVLNode\*);  
struct AVLNode \*rlrotation(struct AVLNode\*);  
struct AVLNode \*rrrotation(struct AVLNode\*);  
struct AVLNode \*insert(struct AVLNode\*,int);  
struct AVLNode \*deleteAVLNode(struct AVLNode\*,int,struct AVLNode\*);  
struct AVLNode \*inpre(struct AVLNode\*);  
struct AVLNode \*insuc(struct AVLNode\*);  
struct AVLNode \*FIND(struct AVLNode\*,int);  
  
#endif // AVLTREE\_H

#### 7.2.1.2 avltree.cpp

#include"avltree.h"  
  
int calheight(struct AVLNode \*p)  
{  
 //qDebug()<<"calheight"<<endl<<" "<<"val"<<p->data<<"addr"<<p;  
 if(p==NULL)return 0;  
 if(p->left&&p->right)  
 {  
 if(p->left->height<p->right->height)  
 return p->right->height+1;  
 else  
 return p->left->height+1;  
 }  
 else if(p->left&&p->right==NULL)  
 return p->left->height+1;  
 else if(p->left==NULL&&p->right)  
 return p->right->height+1;  
 return 1;  
}  
  
int bf(struct AVLNode \*n)  
{  
 if(n==NULL)return 0;  
  
 if(n->left&&n->right)  
 return n->left->height-n->right->height;  
 else if(n->left&&n->right==NULL)  
 return n->left->height;  
 else if(n->left==NULL&&n->right)  
 return -n->right->height;  
 return 0;//not used  
}  
  
struct AVLNode \*llrotation(struct AVLNode \*n)  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 p=n;  
 tp=p->left;  
  
 p->left=tp->right;  
 tp->right=p;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
  
 return tp;  
}  
  
  
struct AVLNode \*rrrotation(struct AVLNode \*n)  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 p=n;  
 tp=p->right;  
  
 p->right=tp->left;  
 tp->left=p;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
  
 return tp;  
}  
  
  
struct AVLNode \*rlrotation(struct AVLNode \*n)  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 struct AVLNode \*tp2;  
 p=n;  
 tp=p->right;  
 tp2=p->right->left;  
  
 p->right=tp2->left;  
 tp->left=tp2->right;  
 tp2->left=p;  
 tp2->right=tp;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
 tp2->height=calheight(tp2);  
  
 return tp2;  
}  
  
struct AVLNode \*lrrotation(struct AVLNode \*n)  
{  
 struct AVLNode \*p;  
 struct AVLNode \*tp;  
 struct AVLNode \*tp2;  
 p=n;  
 tp=p->left;  
 tp2=p->left->right;  
  
 p->left=tp2->right;  
 tp->right=tp2->left;  
 tp2->right=p;  
 tp2->left=tp;  
  
 p->height=calheight(p);  
 tp->height=calheight(tp);  
 tp2->height=calheight(tp2);  
  
 return tp2;  
}  
  
struct AVLNode \*insert(struct AVLNode \*r,int data)  
{  
 qDebug()<<"insert addr"<<r;  
 if(r==NULL)//如果当前位置为空，就在此处插入新节点  
 {  
 struct AVLNode \*n;  
 n=new struct AVLNode;  
 n->data=data;  
 //n->lab.setText(QString::number(data));  
 r=n;  
 r->left=r->right=NULL;  
 r->height=1;  
 //qDebug()<<"newly created addr"<<r;  
 return r;  
 }  
 else  
 {  
 if(data<r->data)  
 r->left=insert(r->left,data);  
 else if(data>r->data)  
 r->right=insert(r->right,data);  
 }  
  
 r->height=calheight(r);//新的节点已经在某个子树中插入，递归更新当前节点的高度  
  
 //qDebug()<<"before raotation node"<<r->data<<"bf"<<bf(r);  
  
 //如果不平衡，进行旋转  
 if(bf(r)>1&&bf(r->left)>0)  
 r=llrotation(r);  
 else if(bf(r)>1&&bf(r->left)<=0)  
 r=lrrotation(r);  
 else if(bf(r)<-1&&bf(r->right)<=0)  
 r=rrrotation(r);  
 else if(bf(r)<-1&&bf(r->right)>0)  
 r=rlrotation(r);  
  
 //qDebug()<<"after rotation node"<<r->data<<"bf"<<bf(r);  
 //r->height=calheight(r);  
  
  
 return r;  
  
}  
  
struct AVLNode \*FIND(struct AVLNode \*r,int data)  
{  
 if(r==NULL)return NULL;  
 if(data<r->data)  
 return FIND(r->left,data);  
 else if(data>r->data)  
 return FIND(r->right,data);  
 else return r;  
}  
  
struct AVLNode \*deleteAVLNode(struct AVLNode \*p,int data,struct AVLNode \*root)  
{  
  
 if(p->left==NULL&&p->right==NULL)//如果当前节点是叶子节点，直接删除  
 {  
 if(p==root)//如果是AVL树的根  
 root=NULL;  
 delete p;  
 return NULL;  
 }  
  
 struct AVLNode \*q;  
 //递归调用删除函数来在左右子树寻找节点  
 if(p->data<data)  
 p->right=deleteAVLNode(p->right,data,root);  
 else if(p->data>data)  
 p->left=deleteAVLNode(p->left,data,root);  
 else//找到了要删除的节点  
 {  
 if(p->left!=NULL)//如果左子树非空，将当前节点的值与左子树中的最大值交换  
 {  
 q=inpre(p->left);  
 p->data=q->data;  
 //p->lab.setText(QString::number(p->data));  
 p->left=deleteAVLNode(p->left,q->data,root);  
 }  
 else//右子树非空，将当前节点的值与右子树中的最小值交换  
 {  
 q=insuc(p->right);  
 p->data=q->data;  
 //p->lab.setText(QString::number(p->data));  
 p->right=deleteAVLNode(p->right,q->data,root);  
 }  
 }  
  
 p->height=calheight(p);  
  
 qDebug()<<"before raotation node"<<p->data<<"height"<<p->height<<"bf"<<bf(p);  
  
 if(bf(p)>1&&bf(p->left)>0)  
 p=llrotation(p);  
 else if(bf(p)>1&&bf(p->left)<=0)  
 p=lrrotation(p);  
 else if(bf(p)<-1&&bf(p->right)<=0)  
 p=rrrotation(p);  
 else if(bf(p)<-1&&bf(p->right)>0)  
 p=rlrotation(p);  
  
 qDebug()<<"after rotation node"<<p->data<<"height"<<p->height<<"bf"<<bf(p);  
  
 return p;  
}  
  
struct AVLNode \*inpre(struct AVLNode\* p)  
{  
 while(p->right!=NULL)  
 p=p->right;  
 return p;  
}  
  
struct AVLNode \*insuc(struct AVLNode\* p)  
{  
 while(p->left!=NULL)  
 p=p->left;  
 return p;  
}

#### 7.2.2.1 mainwindow.h

#ifndef MAINWINDOW\_H  
#define MAINWINDOW\_H  
  
#include <QMainWindow>  
  
QT\_BEGIN\_NAMESPACE  
namespace Ui { class MainWindow; }  
QT\_END\_NAMESPACE  
  
struct Cord  
{  
 int x,y;  
};  
  
class MainWindow : public QMainWindow  
{  
 Q\_OBJECT  
  
public:  
 MainWindow(QWidget \*parent = nullptr);  
 ~MainWindow();  
  
 void CallDrawTree(int);  
 void DrawTree(void\*,Cord,int);  
 void Hide(void\*);  
private slots:  
 #define BUTTONCLICKDECLEAR(n) void on\_Button##n##\_clicked();  
 BUTTONCLICKDECLEAR(9)  
 BUTTONCLICKDECLEAR(8)  
 BUTTONCLICKDECLEAR(7)  
 BUTTONCLICKDECLEAR(6)  
 BUTTONCLICKDECLEAR(5)  
 BUTTONCLICKDECLEAR(4)  
 BUTTONCLICKDECLEAR(3)  
 BUTTONCLICKDECLEAR(2)  
 BUTTONCLICKDECLEAR(1)  
 BUTTONCLICKDECLEAR(0)  
  
 void on\_ButtonBackSpace\_clicked();  
  
 void on\_ButtonCLR\_clicked();  
  
 void on\_ButtonInsert\_clicked();  
  
 void on\_ButtonFind\_clicked();  
  
 void on\_ButtonDelete\_clicked();  
  
 void on\_ButtonNewTree\_clicked();  
  
 void on\_tableWidget\_cellClicked(int row, int column);  
  
 void on\_ButtonMergeTree\_clicked();  
  
 void on\_ButtonDeleteTree\_clicked();  
  
private:  
 Ui::MainWindow \*ui;  
};  
#endif // MAINWINDOW\_H

#### 7.2.2.2 mainwindow.cpp

#include "mainwindow.h"  
#include "ui\_mainwindow.h"  
#include<QDebug>  
#include<QLabel>  
#include<QCheckBox>  
#include<QTableWidgetItem>  
#include<QPainter>  
#include "avltree.h"  
  
#include<cmath>  
  
int Displaying=-1;  
  
std::vector<struct AVLNode\*>TREE;  
  
int D;  
const int W=40,H=30;  
const int LeafOffsetX=5;  
  
MainWindow::MainWindow(QWidget \*parent)  
 : QMainWindow(parent)  
 , ui(new Ui::MainWindow)  
{  
 ui->setupUi(this);  
  
 ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);  
 ui->tableWidget->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);  
}  
  
MainWindow::~MainWindow()  
{  
 while(!TREE.empty())  
 {  
 auto root=\*TREE.begin();  
 while(root)  
 root=deleteAVLNode(root,root->data,root);  
 TREE.erase(TREE.begin());  
 }  
 delete ui;  
}  
  
void MainWindow::Hide(void \*pt)  
{  
 AVLNode\* now\_node=(AVLNode\*)pt;  
 if(now\_node!=NULL)  
 {  
 now\_node->lab.hide();  
 now\_node->Llink.hide();  
 now\_node->Rlink.hide();  
 Hide(now\_node->left);  
 Hide(now\_node->right);  
 }  
}  
  
void MainWindow::DrawTree(void \*pt,Cord now\_cord,int now\_depth)  
{  
 AVLNode\* now\_node=(AVLNode\*)pt;  
 now\_node->lab.setParent(ui->DrawingBoard);  
 now\_node->Llink.setParent(ui->DrawingBoard);  
 now\_node->Rlink.setParent(ui->DrawingBoard);  
 now\_node->Llink.lower();  
 now\_node->Rlink.lower();  
 //now\_node->lab.hide();  
 now\_node->lab.setText(QString::number(now\_node->data));  
 ui->DrawingBoard->setMinimumWidth(std::max(now\_cord.x+W,ui->DrawingBoard->minimumWidth()));  
 ui->DrawingBoard->setMinimumHeight(std::max(now\_cord.y+H,ui->DrawingBoard->minimumHeight()));  
 now\_node->lab.setGeometry(now\_cord.x,now\_cord.y,W,H);  
 now\_node->lab.setStyleSheet("background-color: white; border: 2px solid red;\  
 border-radius: "+QString::number(H/2)+"px;\  
 font-size: "+QString::number(H/2)+"px;\  
 color:red");  
 now\_node->lab.setAlignment(Qt::AlignCenter);  
 now\_node->lab.show();  
  
 if(now\_node->left!=NULL)  
 {  
 Cord L;  
 L.x=now\_cord.x-0.5\*(W+LeafOffsetX)\*(pow(2,D-now\_depth-1));  
 //L.x-=LeafOffsetX\*(D-now\_depth);  
 L.y=now\_cord.y+2\*H;  
  
 int pix\_w=(now\_cord.x+0.5\*W)-(L.x+0.5\*W),pix\_h=2\*H;  
 QImage pix(pix\_w,pix\_h,QImage::Format\_ARGB32\_Premultiplied);  
 QPainter paint(&pix);  
 //pix.fill(Qt::white);  
 pix.fill(qRgba(0,0,0,0));  
 paint.setPen(QPen(Qt::red,2));  
 paint.setRenderHints(QPainter::Antialiasing);  
 paint.setRenderHints(QPainter::HighQualityAntialiasing);  
 paint.setRenderHints(QPainter::SmoothPixmapTransform);  
 paint.drawLine(pix\_w,0,0,pix\_h);  
 paint.end();  
  
 now\_node->Llink.setGeometry(L.x+0.5\*W,now\_cord.y+0.5\*H,pix\_w,pix\_h);  
 now\_node->Llink.setPixmap(QPixmap::fromImage(pix));  
 now\_node->Llink.show();  
  
 DrawTree(now\_node->left,L,now\_depth+1);  
 }  
 if(now\_node->right!=NULL)  
 {  
 Cord R;  
 R.x=now\_cord.x+0.5\*(W+LeafOffsetX)\*(pow(2,D-now\_depth-1));  
 //R.x+=LeafOffsetX\*(D-now\_depth);  
 R.y=now\_cord.y+2\*H;  
  
 int pix\_w=(R.x+0.5\*W)-(now\_cord.x+0.5\*W),pix\_h=2\*H;  
 QImage pix(pix\_w,pix\_h,QImage::Format\_ARGB32\_Premultiplied);  
 QPainter paint(&pix);  
 //pix.fill(Qt::white);  
 pix.fill(qRgba(0,0,0,0));  
 paint.setPen(QPen(Qt::red,2));  
 paint.setRenderHints(QPainter::Antialiasing);  
 paint.setRenderHints(QPainter::HighQualityAntialiasing);  
 paint.setRenderHints(QPainter::SmoothPixmapTransform);  
 paint.drawLine(0,0,pix\_w,pix\_h);  
 paint.end();  
  
 now\_node->Rlink.setGeometry(now\_cord.x+0.5\*W,now\_cord.y+0.5\*H,pix\_w,pix\_h);  
 now\_node->Rlink.setPixmap(QPixmap::fromImage(pix));  
 now\_node->Rlink.show();  
  
 DrawTree(now\_node->right,R,now\_depth+1);  
 }  
  
}  
  
void MainWindow::CallDrawTree(int whichTree)  
{  
 D=calheight(TREE[whichTree]);  
 qDebug()<<"tree height"<<D;  
 int Xroot=0.5\*(W+LeafOffsetX)\*(pow(2,D-1)-1);  
 Cord CordRoot;  
 CordRoot.x=Xroot,CordRoot.y=0;  
 if(Displaying!=-1)  
 {  
 qDebug()<<"hiding"<<TREE[Displaying];  
 Hide(TREE[Displaying]);  
 Displaying=-1;  
 }  
 DrawTree(TREE[whichTree],CordRoot,1);  
 Displaying=whichTree;  
 update();  
}  
  
#define BUTTONCLICKFUNC(n)\  
void MainWindow::on\_Button##n##\_clicked()\  
{\  
 if((long long)ui->lcdNumber->intValue()\*10+n<=1e9)\  
 ui->lcdNumber->display(ui->lcdNumber->intValue()\*10+n);\  
}\  
  
BUTTONCLICKFUNC(9)  
BUTTONCLICKFUNC(8)  
BUTTONCLICKFUNC(7)  
BUTTONCLICKFUNC(6)  
BUTTONCLICKFUNC(5)  
BUTTONCLICKFUNC(4)  
BUTTONCLICKFUNC(3)  
BUTTONCLICKFUNC(2)  
BUTTONCLICKFUNC(1)  
BUTTONCLICKFUNC(0)  
  
void MainWindow::on\_ButtonBackSpace\_clicked()  
{  
 ui->lcdNumber->display(ui->lcdNumber->intValue()/10);  
}  
  
void MainWindow::on\_ButtonCLR\_clicked()  
{  
 ui->lcdNumber->display(0);  
}  
  
void MainWindow::on\_ButtonInsert\_clicked()  
{  
 int x=ui->lcdNumber->intValue();  
 qDebug()<<"Inserting:"<<endl<<" "<<x;  
  
 if(Displaying==-1)  
 {  
 qDebug()<<"no tree being displayed";  
 on\_ButtonCLR\_clicked();  
 return;  
 }  
 qDebug()<<"to tree"<<Displaying;  
 TREE[Displaying]=insert(TREE[Displaying],x);  
  
 ui->tableWidget->setItem(Displaying,2,new QTableWidgetItem(QString::number(TREE[Displaying]->data)));  
 ui->tableWidget->setItem(Displaying,3,new QTableWidgetItem(QString::number((long long)TREE[Displaying],16)));  
  
 CallDrawTree(Displaying);  
  
 on\_ButtonCLR\_clicked();  
}  
  
  
void MainWindow::on\_ButtonFind\_clicked()  
{  
 int x=ui->lcdNumber->intValue();  
 qDebug()<<"Looking For:"<<endl<<" "<<x;  
  
 if(Displaying==-1)  
 {  
 qDebug()<<"no tree being displayed";  
 on\_ButtonCLR\_clicked();  
 return;  
 }  
 AVLNode\* res=FIND(TREE[Displaying],x);  
 if(res==NULL)  
 {  
 qDebug()<<"not found";  
 on\_ButtonCLR\_clicked();  
 return;  
 }  
 else  
 {  
 CallDrawTree(Displaying);  
 res->lab.setStyleSheet("background-color: yellow; border: 2px solid red;\  
 border-radius: "+QString::number(H/2)+"px;\  
 font-size: "+QString::number(H/2)+"px;\  
 color:red");  
 }  
  
 on\_ButtonCLR\_clicked();  
}  
  
  
void MainWindow::on\_ButtonDelete\_clicked()  
{  
 int x=ui->lcdNumber->intValue();  
 qDebug()<<"Trying to Delete:"<<endl<<" "<<x;  
  
 if(Displaying==-1)  
 {  
 qDebug()<<"no tree being displayed";  
 on\_ButtonCLR\_clicked();  
 return;  
 }  
  
 if(FIND(TREE[Displaying],x)==NULL)  
 {  
 qDebug()<<"not found";  
 on\_ButtonCLR\_clicked();  
 return;  
 }  
 TREE[Displaying]=deleteAVLNode(TREE[Displaying],x,TREE[Displaying]);  
  
 if(TREE[Displaying]!=NULL)  
 {  
 ui->tableWidget->setItem(Displaying,2,new QTableWidgetItem(QString::number(TREE[Displaying]->data)));  
 ui->tableWidget->setItem(Displaying,3,new QTableWidgetItem(QString::number((long long)TREE[Displaying],16)));  
  
 CallDrawTree(Displaying);  
 }  
 else  
 {  
 ui->tableWidget->removeRow(Displaying);  
 TREE.erase(TREE.begin()+Displaying);  
 for(int i=Displaying;i<(int)TREE.size();++i)  
 ui->tableWidget->setItem(i,1,new QTableWidgetItem(QString::number(i)));  
 Displaying=-1;  
 }  
  
 on\_ButtonCLR\_clicked();  
}  
  
  
void MainWindow::on\_ButtonNewTree\_clicked()  
{  
 int x=ui->lcdNumber->intValue();  
 //qDebug()<<"New Tree:"<<endl<<" "<<x;  
  
 struct AVLNode \*NewTree=NULL;  
 NewTree=insert(NewTree,x);  
  
 qDebug()<<"got new addr"<<NewTree;  
  
 TREE.push\_back(NewTree);  
  
 ui->tableWidget->insertRow(TREE.size()-1);  
 QTableWidgetItem \*checkBoxItem = new QTableWidgetItem();  
 checkBoxItem->setCheckState(Qt::Unchecked);  
 ui->tableWidget->setItem(TREE.size()-1,0,checkBoxItem);  
 //ui->tableWidget->setCellWidget(TREE.size()-1,0,new QCheckBox);  
 ui->tableWidget->setItem(TREE.size()-1,1,new QTableWidgetItem(QString::number(TREE.size()-1)));  
 ui->tableWidget->setItem(TREE.size()-1,2,new QTableWidgetItem(QString::number(NewTree->data)));  
 ui->tableWidget->setItem(TREE.size()-1,3,new QTableWidgetItem(QString::number((long long)NewTree,16)));  
  
  
 on\_ButtonCLR\_clicked();  
}  
  
  
void MainWindow::on\_tableWidget\_cellClicked(int row, int column)  
{  
 CallDrawTree(row);  
}  
  
void MainWindow::on\_ButtonMergeTree\_clicked()  
{  
 qDebug()<<"merging trees";  
 int FirstSelected=-1;  
 for(int i=0;i<(int)TREE.size();++i)  
 {  
 if(ui->tableWidget->item(i,0)->checkState())  
 {  
 FirstSelected=i;  
 break;  
 }  
 }  
 if(FirstSelected==-1)  
 {  
 qDebug()<<"nothing selected!";  
 return;  
 }  
 for(int i=FirstSelected+1;i<(int)TREE.size();++i)  
 {  
 if(!ui->tableWidget->item(i,0)->checkState())  
 continue;  
 qDebug()<<i<<"is going to be merged with"<<FirstSelected;  
 while(TREE[i])  
 {  
  
 TREE[FirstSelected]=insert(TREE[FirstSelected],TREE[i]->data);  
 TREE[i]=deleteAVLNode(TREE[i],TREE[i]->data,TREE[i]);  
  
 ui->tableWidget->setItem(FirstSelected,2,new QTableWidgetItem(QString::number(TREE[FirstSelected]->data)));  
 ui->tableWidget->setItem(FirstSelected,3,new QTableWidgetItem(QString::number((long long)TREE[FirstSelected],16)));  
  
 CallDrawTree(FirstSelected);  
 }  
 TREE.erase(TREE.begin()+i);  
 ui->tableWidget->removeRow(i);  
 --i;  
 }  
}  
  
  
void MainWindow::on\_ButtonDeleteTree\_clicked()  
{  
 qDebug()<<"deleting trees";  
 for(int i=0;i<(int)TREE.size();++i)  
 {  
 if(!ui->tableWidget->item(i,0)->checkState())  
 continue;  
 if(Displaying!=-1)  
 {  
 qDebug()<<"hiding"<<TREE[Displaying];  
 Hide(TREE[Displaying]);  
 Displaying=-1;  
 }  
 qDebug()<<i<<"is going to be deleted";  
 while(TREE[i])  
 TREE[i]=deleteAVLNode(TREE[i],TREE[i]->data,TREE[i]);  
 TREE.erase(TREE.begin()+i);  
 ui->tableWidget->removeRow(i);  
 --i;  
 }  
 for(int i=0;i<(int)TREE.size();++i)  
 ui->tableWidget->setItem(i,1,new QTableWidgetItem(QString::number(i)));  
}

#### 7.2.2.3 mainwindow.ui

<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>600</height>  
 </rect>  
 </property>  
 <property name="windowTitle">  
 <string>qtAVLtree</string>  
 </property>  
 <property name="autoFillBackground">  
 <bool>false</bool>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">MainWindow{  
background-image: url(:/tkk.png);  
}</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <layout class="QGridLayout" name="gridLayout\_3">  
 <item row="1" column="0">  
 <layout class="QGridLayout" name="gridLayout">  
 <item row="2" column="2">  
 <widget class="QPushButton" name="Button6">  
 <property name="text">  
 <string>6</string>  
 </property>  
 </widget>  
 </item>  
 <item row="3" column="0">  
 <widget class="QPushButton" name="Button1">  
 <property name="text">  
 <string>1</string>  
 </property>  
 </widget>  
 </item>  
 <item row="4" column="2">  
 <widget class="QPushButton" name="ButtonCLR">  
 <property name="text">  
 <string>CLR</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="1">  
 <widget class="QPushButton" name="Button8">  
 <property name="text">  
 <string>8</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="0">  
 <widget class="QPushButton" name="Button7">  
 <property name="text">  
 <string>7</string>  
 </property>  
 </widget>  
 </item>  
 <item row="4" column="0">  
 <widget class="QPushButton" name="Button0">  
 <property name="text">  
 <string>0</string>  
 </property>  
 </widget>  
 </item>  
 <item row="4" column="1">  
 <widget class="QPushButton" name="ButtonBackSpace">  
 <property name="text">  
 <string>&lt;-</string>  
 </property>  
 </widget>  
 </item>  
 <item row="4" column="3">  
 <widget class="QPushButton" name="ButtonDelete">  
 <property name="text">  
 <string>Delete</string>  
 </property>  
 </widget>  
 </item>  
 <item row="3" column="3">  
 <widget class="QPushButton" name="ButtonFind">  
 <property name="text">  
 <string>Find</string>  
 </property>  
 </widget>  
 </item>  
 <item row="2" column="0">  
 <widget class="QPushButton" name="Button4">  
 <property name="text">  
 <string>4</string>  
 </property>  
 </widget>  
 </item>  
 <item row="2" column="3">  
 <widget class="QPushButton" name="ButtonInsert">  
 <property name="text">  
 <string>Insert</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="3">  
 <widget class="QPushButton" name="ButtonNewTree">  
 <property name="text">  
 <string>NewTree</string>  
 </property>  
 </widget>  
 </item>  
 <item row="3" column="2">  
 <widget class="QPushButton" name="Button3">  
 <property name="text">  
 <string>3</string>  
 </property>  
 </widget>  
 </item>  
 <item row="2" column="1">  
 <widget class="QPushButton" name="Button5">  
 <property name="text">  
 <string>5</string>  
 </property>  
 </widget>  
 </item>  
 <item row="3" column="1">  
 <widget class="QPushButton" name="Button2">  
 <property name="text">  
 <string>2</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="2">  
 <widget class="QPushButton" name="Button9">  
 <property name="text">  
 <string>9</string>  
 </property>  
 </widget>  
 </item>  
 <item row="0" column="0" colspan="4">  
 <widget class="QLCDNumber" name="lcdNumber">  
 <property name="palette">  
 <palette>  
 <active>  
 <colorrole role="Button">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>240</red>  
 <green>120</green>  
 <blue>105</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Light">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>255</red>  
 <green>122</green>  
 <blue>96</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Midlight">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>247</red>  
 <green>48</green>  
 <blue>58</blue>  
 </color>  
 </brush>  
 </colorrole>  
 </active>  
 <inactive>  
 <colorrole role="Button">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>240</red>  
 <green>120</green>  
 <blue>105</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Light">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>255</red>  
 <green>122</green>  
 <blue>96</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Midlight">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>247</red>  
 <green>48</green>  
 <blue>58</blue>  
 </color>  
 </brush>  
 </colorrole>  
 </inactive>  
 <disabled>  
 <colorrole role="Button">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>240</red>  
 <green>120</green>  
 <blue>105</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Light">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>255</red>  
 <green>122</green>  
 <blue>96</blue>  
 </color>  
 </brush>  
 </colorrole>  
 <colorrole role="Midlight">  
 <brush brushstyle="SolidPattern">  
 <color alpha="255">  
 <red>247</red>  
 <green>48</green>  
 <blue>58</blue>  
 </color>  
 </brush>  
 </colorrole>  
 </disabled>  
 </palette>  
 </property>  
 <property name="digitCount">  
 <number>10</number>  
 </property>  
 <property name="segmentStyle">  
 <enum>QLCDNumber::Filled</enum>  
 </property>  
 <property name="intValue" stdset="0">  
 <number>0</number>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item row="1" column="1">  
 <layout class="QGridLayout" name="gridLayout\_2">  
 <item row="1" column="0">  
 <widget class="QPushButton" name="ButtonMergeTree">  
 <property name="text">  
 <string>MergeTree</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="1">  
 <widget class="QPushButton" name="ButtonDeleteTree">  
 <property name="text">  
 <string>DeleteTree</string>  
 </property>  
 </widget>  
 </item>  
 <item row="0" column="0" colspan="2">  
 <widget class="QTableWidget" name="tableWidget">  
 <property name="styleSheet">  
 <string notr="true"/>  
 </property>  
 <property name="rowCount">  
 <number>0</number>  
 </property>  
 <property name="columnCount">  
 <number>4</number>  
 </property>  
 <column>  
 <property name="text">  
 <string>选择</string>  
 </property>  
 <property name="textAlignment">  
 <set>AlignLeading|AlignVCenter</set>  
 </property>  
 </column>  
 <column>  
 <property name="text">  
 <string>序号</string>  
 </property>  
 <property name="textAlignment">  
 <set>AlignLeading|AlignVCenter</set>  
 </property>  
 </column>  
 <column>  
 <property name="text">  
 <string>树根</string>  
 </property>  
 <property name="textAlignment">  
 <set>AlignLeading|AlignVCenter</set>  
 </property>  
 </column>  
 <column>  
 <property name="text">  
 <string>树根地址</string>  
 </property>  
 <property name="textAlignment">  
 <set>AlignLeading|AlignVCenter</set>  
 </property>  
 </column>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item row="0" column="0" colspan="2">  
 <widget class="QScrollArea" name="scrollArea">  
 <property name="minimumSize">  
 <size>  
 <width>0</width>  
 <height>0</height>  
 </size>  
 </property>  
 <property name="styleSheet">  
 <string notr="true"/>  
 </property>  
 <property name="verticalScrollBarPolicy">  
 <enum>Qt::ScrollBarAsNeeded</enum>  
 </property>  
 <property name="horizontalScrollBarPolicy">  
 <enum>Qt::ScrollBarAsNeeded</enum>  
 </property>  
 <property name="widgetResizable">  
 <bool>true</bool>  
 </property>  
 <widget class="QWidget" name="DrawingBoard">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>772</width>  
 <height>257</height>  
 </rect>  
 </property>  
 <property name="styleSheet">  
 <string notr="true"/>  
 </property>  
 </widget>  
 </widget>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>24</height>  
 </rect>  
 </property>  
 </widget>  
 <widget class="QStatusBar" name="statusbar"/>  
 </widget>  
 <resources/>  
 <connections/>  
</ui>

#### 7.2.3 main.cpp

#include "mainwindow.h"  
  
#include <QApplication>  
#include<QLabel>  
  
int main(int argc, char \*argv[])  
{  
 QApplication a(argc, argv);  
 MainWindow w;  
 w.show();  
  
  
 return a.exec();  
}

#### 7.2.4 res.qrc

<RCC>  
 <qresource prefix="/">  
 <file>tkk.png</file>  
 </qresource>  
</RCC>