**数据结构（C++）**

**课程设计报告**

**设计题目: 计算机专业大类分流软件设计**

**院 系: 信息工程学院**

**组 长: 刘文越**

**组 员: 王子卓、肖芷馨、孙翔、康旭**

**指导教师: 慕晨**

**日 期: 2022.12.29**

# 题目

## 设计内容与要求

* 需求：根据学生成绩排名和填报志愿情况将学生分配到各专业
* 输入：专业数、各专业班级数，每班学生人数、各专业的分流报名情况
* 操作：根据输入数据按绩点进行分流
* 输出：分流后分班情况
* 拓展：分流后学生数据的数理统计

## 实验环境

1. 硬件环境：

笔记本电脑

* 处理器：英特尔 Core i7-10870H @ 2.20GHz 八核
* 主 板：微星 MS-17K2
* 内 存：32 GB
* 硬 盘：三星 MZVLB2T0HMLB-00000
* 显 卡：Nvidia GeForce RTX 3070 Laptop GPU

1. 软件环境：

* Windows 10 系统
* Visual Studio 2022 (with Qt Visual Studio Tools插件)
* Qt Creator (with Qt designer)

## 分工及组内得分

（介绍每位组员的角色划分，以及组内得分。组内得分需要符合正态分布）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 组长 | 组员一 | 组员二 | 组员三 | 组员四 |
| 姓名 | 刘文越 | 王子卓 | 肖芷馨 | 孙翔 | 康旭 |
| 学号 | 2021902610 | 2021901374 | 2021901403 | 2021902141 | 2021904571 |
| 组内得分 | A | B | B | C | C |
| 组员分工 | 处理后学生管理类的编写、分流操作的编写、线段树模块的构造 | 预处理类的编写、Qt面板的编写 | 处理后学生管理类的编写、Qt面板的编写 | 链表类的编写、处理后学生管理类的编写 | 链表类的编写、系统稳健性的编写与调试 |

## 系统分析与设计

针对课设需求，首先确定需要进行图形化的UI设计，它具有降低用户学习成本、节省用户时间、功能模块化可视化的优点。对于本系统，我们采用基于C++的Qt框架完成图形化界面的搭建。

针对需求分析软件的功能，我们将其划分为数据处理、数据查看、数据分析、数据修改四个模块：

* **数据处理：**

数据处理即对于原始未分流的专业报名数据进行筛选和合并，依据用户提供的专业名称、专业班级数、班级人数，以平行志愿的准则进行自动分流，最后将分流完成的数据实例化为对应类的一个对象。

* **数据查看：**

以分流完成的数据实例化的对象为基础，对其进行展示，方便用户筛选并查看数据。

* **数据分析：**

以分流完成的数据实例化的对象为基础，对专业或班级的分流后数据进行常见的数理统计学分析，提供给用户分流后的学生分布情况和统计学意义上的分析依据。

* **数据修改**

考虑实际情况中，学生的成绩一般不会进行修改，如果进行修改也需要导入名单重新进行分流，因此学生成绩和排名的修改不应在该分流系统软件的考虑范围内，软件在数据修改方面仅仅提供分流后学生的专业和班级修改。

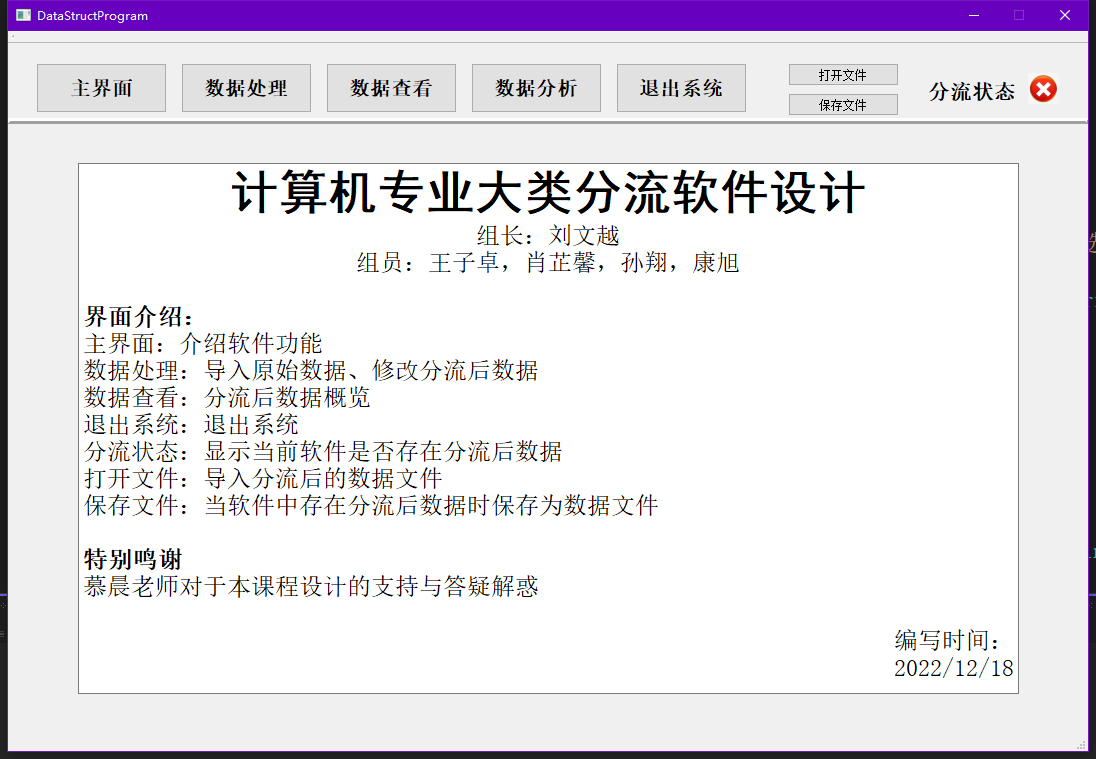
同时本系统需要提供分流后学生信息的存储和再读入功能，数据应该按照固定格式存储于硬盘中，并且支持自选路径存储。

在实际的编写代码过程中，有以下若干细节需要考虑：

* 在文件读写环节，文件读入应支持原始数据的文本文件，文件存储方式上本系统采用特定格式存储于txt文件中，同时系统支持读入本系统自己导出的分流后数据，方便之后再次查询。
* 在数据处理环节，考虑到录入人数一开始是不确定的，提前开辟内存空间的方式显然容易造成内存空间的浪费，本系统解决的方法是，先封装一个用于接收数据预处理类，底层采用链表存储的方式，方便增删节点且不浪费内存空间，并且没有查询操作，链表的缺点也得以避开。
* 在数据处理环节，原始数据中会出现排名相同的学生数据，此时我们应该将其视作同一优先级，对于相同排名、相同志愿且录入后会超出专业人数限制的情况，应同时超出录入以做到公平公正。
* 在数据查看环节，考虑到学生数据较多，仅仅靠浏览难以快速查找特定数据，系统提供了筛选功能，可以依据专业、班级、学生排名、学生学号等条件对分流后学生信息进行筛选。
* 在数据分析环节，数理统计分析往往需要学生的区间特征值，例如区间最大值、区间最小值、区间和等。以暴力的方式维护区间特征值会有极大的时间复杂度，造成时间耗费，而采用动态规划的方式维护区间特征值，虽然节省了不少时间复杂度，但是多层动态规划的情况下需要较高的空间复杂度，需要较大的空间开支。综上，本系统采用线段树的方式维护学生的区间特征值，建树操作为的时间复杂度，而之后的查询和修改操作都为，而总体空间复杂度仍为级别。
* 对于软件的稳健性，在调试阶段中，如果在编辑框内输入错误的学号、专业名等信息，会造成指针访问异常而结束程序，因此我们需要对输入进行限制并使用Qt内置的弹窗对象来提醒用户的输入错误。

## 运行结果

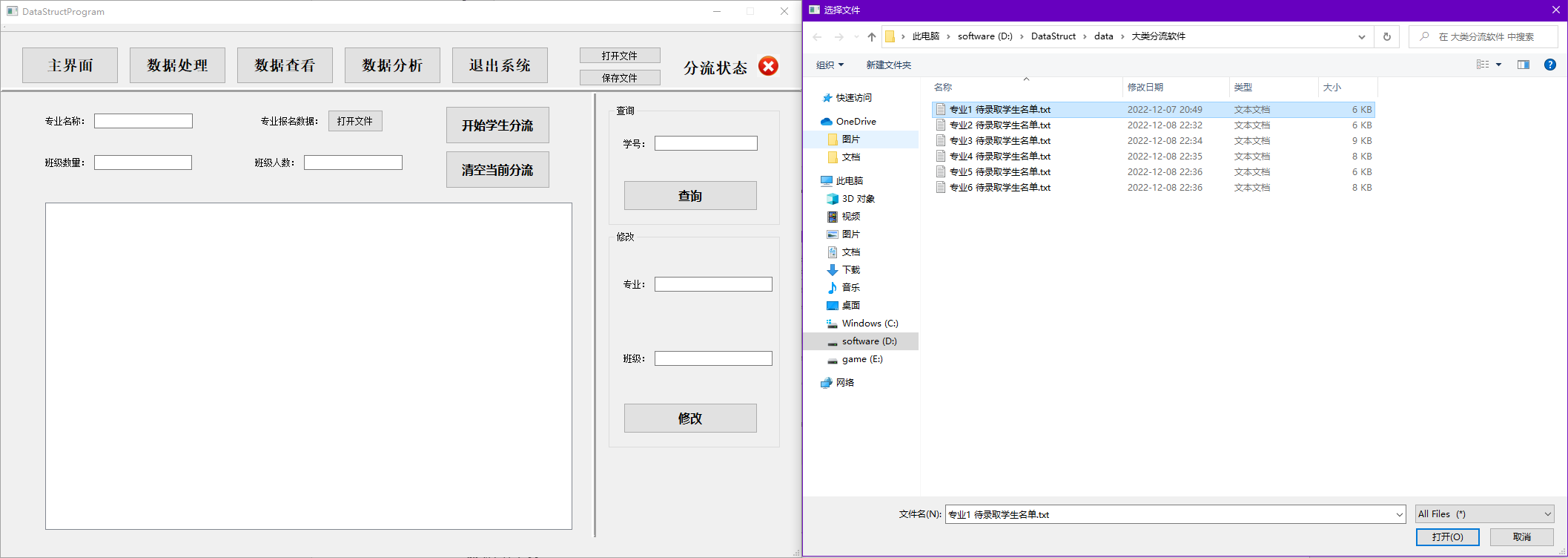
主界面：



数据处理界面：

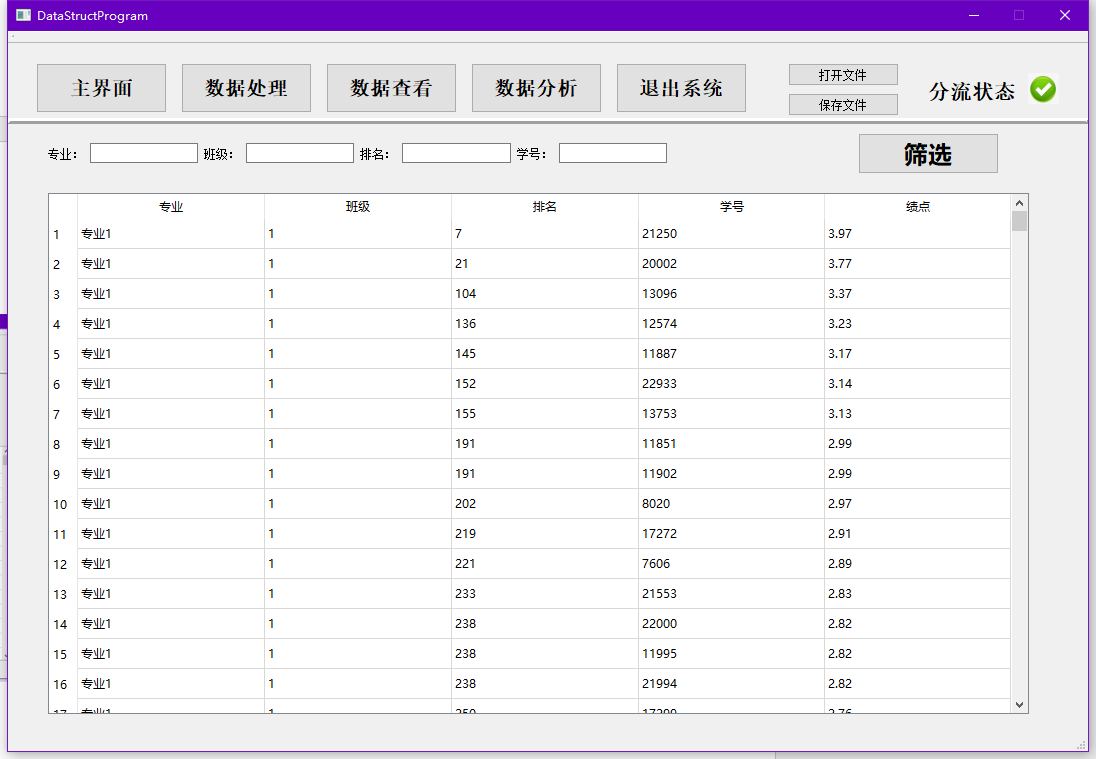


数据导入环节：

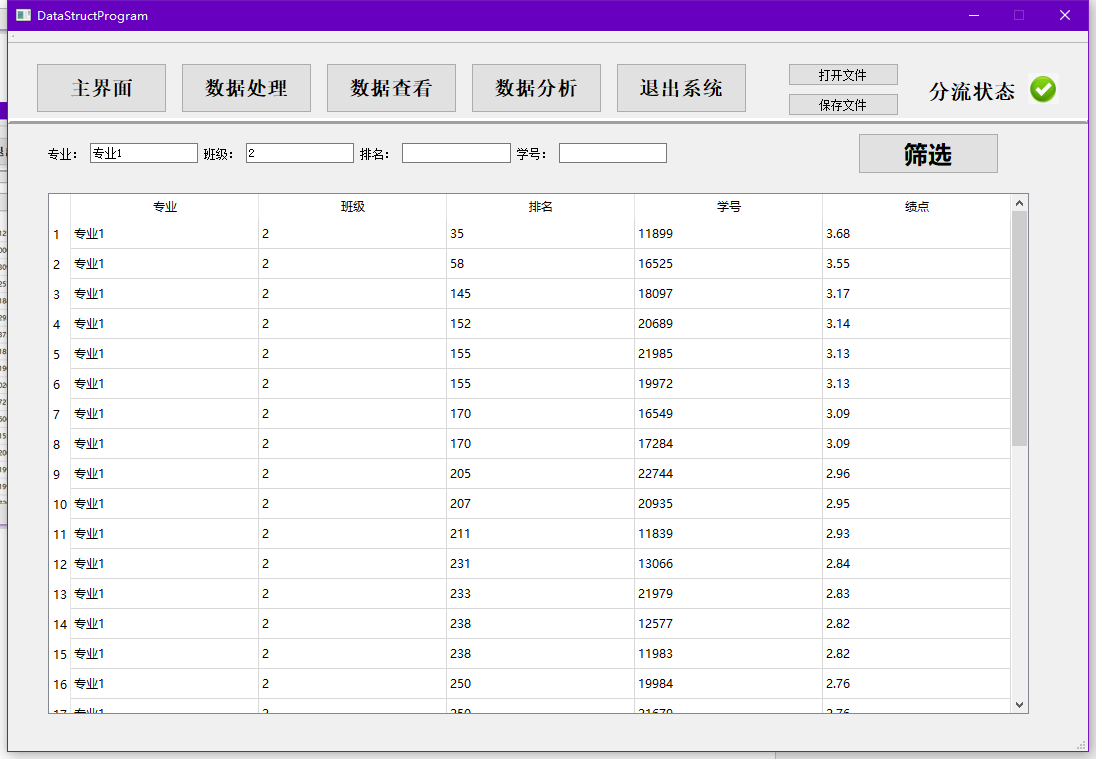




数据查看界面：



数据查看界面（筛选功能）：



数据分析界面：



## 总结

**6.1 课程设计支撑的毕业要求指标点**

数据结构课程设计支撑以下四项毕业要求指标点：

指标点2-4：能够从工程科学的角度，结合文献查阅及研究，对计算机领域复杂工程问题进行系统分析，并获得有效结论。

指标点7-1：知晓和理解环境保护和可持续发展的理念和内涵，正确认识计算机科学技术的发展与环境和可持续发展的关系。

指标点9-3：能够理解个人在团队中的角色划分，且胜任相应的角色职责。

指标点12-1：能在社会发展的大背景下，理解终身学习的必要性，具有自主学习和终身学习的意识。

**6.2心得**

1. 本系统的编写过程中，数据结构与算法课本给我们提供了多种数据结构的思想和实现，帮助我们结合课设需求，使用正确的数据结构去存储和处理数据；《算法竞赛指南》中提供了线段树这种性能优秀的高级数据结构的思想和构造，帮助我们优化了数据分析模块的算法效率；Qt官方文档对于语言内封装类的接口描述清晰，功能丰富，帮助我们实现了图形化的用户界面。
2. 本系统在编写过程中，注重面向对象思想，构造多个封装类，类中封装用于数据交互的接口函数，方便之后对程序的功能维护和拓展，实现了软件的动态维护和可持续发展。在编写过程中，对于每个函数给予了足够的注释进行描述，方便后期人员的调试和维护。
3. 团队在组建时首先确定好小组长，以便划分任务和组织规划。其次小组开会讨论课设的需求，需要完成哪些功能，可能会出现哪些困难等，之后我们将软件依据功能和细节进行模块化划分，基于以上内容划分小组成员各自的内容，在代码书写过程中注意留好供别人调用的接口函数，描述好功能以直接使用，遇到问题及时在交流群内展开讨论。最后，由小组长汇总各自的代码，调试功能，分版本发布在群中方便大家进行bug测试。在此过程中，小组成员学习到了合作与分工的重要性，学会了如何将问题条理细化，认识到了个人与团体是紧密不可分割的，为未来学习与工作生活中的合作打下了良好的基础。
4. 本系统中的自学内容主要是Qt语言的熟悉与使用、使“Qt Visual Studio Tool”插件在VS环境下配置Qt的编译和运行、高级数据结构线段树的思想与构造。  
    在科学技术飞速发展的现在，自学能力显然是当代大学生必不可少的一项技能，而如何在生活中注重培养自学能力，需要我们明确自己的需求和目的、详细认真制定学习计划、妥善利用各种学习渠道、学会自我复盘平时学习的细节和错误、拥有创造性思维和研究新事物的思路。

## 程序源代码

软件采用多文件结构编写，因此源代码前备注对应文件名称

* DataStructProgram.h

#pragma once

#include<iostream>

#include <QtWidgets/QMainWindow>

#include "DivergenceSystem.h"

#include "ui\_DataStructProgram.h"

class DataStructProgram : public QMainWindow

{

Q\_OBJECT

public:

DataStructProgram(QWidget\* parent = Q\_NULLPTR);

private:

Ui::DataStructProgramClass\* ui;

D\_S\_PretreatSystem DSP;

D\_S\_TreatedSystem DSTS;

D\_S\_SegmentTree DSST;

};

* DivergenceSystem.h

#pragma once

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权：长安大学

\* 编写者：刘文越，王子卓，肖芷馨，孙翔，康旭

\*

\* 编写时间：2022/12/04

\* 功能描述：DivergenceSystem头文件。

\* 包括所有自定义类的声明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <QDebug>

#include <map>

class D\_S\_PretreatSystem;

class D\_S\_TreatedSystem;

void string\_split(const std::string& str, const std::string& split, std::vector<int>& rs);

void string\_split(const std::string& str, const std::string& split, std::vector<std::string>& rs);

struct D\_S\_Student {

/\*

原始学生数据的存储节点

\*/

int ID = 0; // 学生学号

int choice = 0; //学生志愿，值为1代表第一志愿

double GPA = -1; //绩点，精度维护在4位有效数字

int rank = 0; //学生排名信息，值为1代表排名第一

friend std::istream& operator>>(std::istream& in, D\_S\_Student& DSS);

friend std::ostream& operator<<(std::ostream& out, const D\_S\_Student& DSS);

};

struct D\_S\_DeLinkedListNode {

D\_S\_Student data;

D\_S\_DeLinkedListNode\* pre = nullptr;

D\_S\_DeLinkedListNode\* nex = nullptr;

};

class D\_S\_MajorList {

/\*

用于接收原始数据的链表类

对应于一个专业的分流前初始文件

\*/

public:

D\_S\_MajorList();

~D\_S\_MajorList();

void deleteMajorList(); //释放链表内存空间

void printList(); //【调试】【控制台】输出链表

void initList(std::string filePath); //从文件读入原始数据，并存储至链表

void addNode(D\_S\_Student DSS); //增加节点

void setListName(std::string name); //设置该专业的名称

int getNodeNum(); //获取该专业人数

friend D\_S\_PretreatSystem;

friend D\_S\_TreatedSystem;

private:

D\_S\_DeLinkedListNode\* head = nullptr;

D\_S\_DeLinkedListNode\* tail = nullptr;

std::string majorName = "NULL";

int nodeNum = 0;

std::pair<int, int> maxNum = { -1, -1 }; //first->专业班级数，second->班级人数

};

class D\_S\_PretreatSystem {

/\*

用于将所有未分流专业报名链表整合至一起并进行分流操作

\*/

public:

void initSystem(); //用于初始化预处理系统

int addMajorList(std::string filePath, std::string majorName, std::pair<int, int> pairNum); //向预处理系统中增添新的专业链表类

D\_S\_Student getNodeByPos(int pos1, int pos2); //通过下标获取单独学生信息

int getListNum(); //得到专业数量

int getListNodeNum(int pos); //通过下标获取对应专业的学生人数

int getSumNum(); //得到学生总数

friend D\_S\_TreatedSystem;

private:

std::vector<D\_S\_MajorList> MajorList;

std::map<int, bool> mp; //用于标记学生学号

int listNum = 0;

};

struct D\_S\_StudentPro {

/\*

分流后的学生数据节点

\*/

int ID = -1; //学生学号

double GPA = 0; //学生绩点

int rank = 0; //学生排名

std::vector<std::pair<int, int>> choice; //分流前代表该学生的报考情况，分流后仅有一个元素，first->专业，second->班级

friend std::istream& operator>>(std::istream& in, D\_S\_StudentPro& DSSP);

friend std::ostream& operator<<(std::ostream& out, const D\_S\_StudentPro& DSSP);

};

class D\_S\_TreatedSystem {

/\*

分流后的学生管理系统类

\*/

public:

std::vector<double> initSystem(D\_S\_PretreatSystem& DSP); //导入分流后的预处理系统，用于初始化分流后的学生管理系统类

void initSystem(QString filePath); //用于从文件导入分流后的学生管理系统类

void saveSystem(QString filePath); //向硬盘中导出分流后的学生管理系统类

void deleteSystem(); //释放内存空间

std::vector<D\_S\_StudentPro> getArray(); //获得学生信息

D\_S\_StudentPro getArrayNode(int Pos); //通过下标获得单独学生信息

D\_S\_StudentPro getArrayNodeByID(int tmpID); //通过学生的学号获得单独的学生信息

std::string getMajorName(int Pos); //通过下标获取专业名称

int getMajorPos(std::string str); //通过专业名称获取对应下标

std::vector<int> getClassNum(); //获得当前班级人数

std::pair<int, int> getMajorClassPos(int majorPos); //获得当前专业对应班级的下标

int getStuNum(); //获得当前全部学生人数'

bool modifyArrayNodeByID(int tmpID, int majorPos, int classPos); //修改学生信息

void sortSystem(); //系统排序

int getClassLeftRange(int x); //线段树中的划分范围函数

int getMajorLeftRange(int x); //线段树中的划分范围函数

bool isInit = false; //用于标记当前软件中有无分流后的学生管理系统类

private:

std::vector<D\_S\_StudentPro> DSSParray;

std::vector<std::string> majorName; //专业名字

std::vector<std::pair<int, int>> maxNum; // first->班级数量，second->班级的最大人数

std::vector<int> nowMajorNum; //当前专业人数

std::vector<int> nowClassNum; //当前班级人数

};

struct D\_S\_S\_TreeNode {

/\*

用于维护线段树的数据节点

\*/

double GPASum = 0, GPAMax = -1, GPAMin = 10;

int l = 0, r = 0;

D\_S\_S\_TreeNode\* left = nullptr, \* right = nullptr;

friend std::ostream& operator<<(std::ostream& out, const D\_S\_S\_TreeNode& DSSTN);

};

class D\_S\_SegmentTree {

/\*

线段树封装类

用于维护学生数据的区间特征值

\*/

public:

void build(D\_S\_TreatedSystem& DSTS, D\_S\_S\_TreeNode\*& x, int l, int r); //建树函数，初始化线段树

void build(D\_S\_TreatedSystem& DSTS, int l, int r); //建树函数，初始化线段树

D\_S\_S\_TreeNode query(D\_S\_S\_TreeNode\* x, int l, int r); //查询函数

D\_S\_S\_TreeNode query(int l, int r); //查询函数

double getS(int l, int r, double x); //维护方差

void deleteTree(); //释放内存空间

private:

D\_S\_S\_TreeNode\* root = nullptr;

};

* DataStructProgram.cpp

#include "DataStructProgram.h"

#include <QFileDialog>

#include <QDebug>

#include <QMessageBox>

#include <algorithm>

#include <cstring>

DataStructProgram::DataStructProgram(QWidget\* parent)

: QMainWindow(parent), ui(new Ui::DataStructProgramClass)

{

//初始化操作：

ui->setupUi(this);

ui->stackedWidget->setCurrentIndex(0);

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->P3\_tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->systemState->setCurrentIndex(0);

//主页面切换：

connect(ui->mainPB1, &QPushButton::clicked, [=]()mutable {

ui->stackedWidget->setCurrentIndex(0);

});

connect(ui->mainPB2, &QPushButton::clicked, [=]()mutable {

ui->stackedWidget->setCurrentIndex(1);

});

connect(ui->mainPB3, &QPushButton::clicked, [=]()mutable {

ui->stackedWidget->setCurrentIndex(2);

});

connect(ui->mainPB4, &QPushButton::clicked, [=]()mutable {

ui->stackedWidget->setCurrentIndex(3);

});

connect(ui->mainPB5, &QPushButton::clicked, [=]()mutable {

this->close();

});

//导入导出文件：

connect(ui->systemOpenFile, &QPushButton::clicked, [=]() mutable {

QString filePath = QFileDialog::getOpenFileName(this, QString::fromLocal8Bit(std::string("选择文件").data()), QString::fromLocal8Bit(std::string("D:/DataStruct/data").data()));

if (filePath.isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("路径为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

DSTS.initSystem(filePath);

ui->systemState->setCurrentIndex(1);

});

connect(ui->systemSaveFile, &QPushButton::clicked, [=]()mutable {

QString filePath = QFileDialog::getExistingDirectory(this, QString::fromLocal8Bit(std::string("选择文件夹").data()), "D:/DataStruct/data");

if (filePath.isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("路径为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

DSTS.saveSystem(filePath);

});

//分流界面：

connect(ui->P2\_PBopenFile, &QPushButton::clicked, [=]() mutable {

std::string majorName(ui->P2\_LEname->text().toLocal8Bit());

QString classNumData = ui->P2\_LEclassNum->text();

QString stuNumData = ui->P2\_LEstuNum->text();

if (majorName == "") {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("专业名为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

if (classNumData.isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("班级数量为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

if (stuNumData.isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("班级人数为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

int classNum = classNumData.toInt();

int stuNum = stuNumData.toInt();

ui->P2\_LEclassNum->clear();

ui->P2\_LEstuNum->clear();

ui->P2\_LEname->clear();

QString filePath = QFileDialog::getOpenFileName(this, QString::fromLocal8Bit(std::string("选择文件").data()), QString::fromLocal8Bit(std::string("D:/DataStruct/data/大类分流软件").data()));

if (filePath.isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("路径为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

else {

QByteArray cdata = filePath.toLocal8Bit();

std::pair<int, int> pairNum = { classNum, stuNum };

int nodeNum = this->DSP.addMajorList(std::string(cdata), majorName, pairNum);

ui->tableWidget->setColumnCount(4);

ui->tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("编号").data()) << QString::fromLocal8Bit(std::string("志愿").data()) << QString("GPA") << QString::fromLocal8Bit(std::string("排名").data()));

ui->tableWidget->setRowCount(nodeNum);

for (int i = 0, col = 0; i < nodeNum; i++, col = 0) {

D\_S\_Student now = this->DSP.getNodeByPos(this->DSP.getListNum(), i + 1);

ui->tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(now.ID)));

ui->tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(now.choice)));

ui->tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(now.GPA, 'f', 2)));

ui->tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(now.rank)));

}

}

});

connect(ui->P2\_PBstartDiver, &QPushButton::clicked, [=]() mutable {

if (DSTS.isInit) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("已有分流后数据").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

std::vector<double> grade = DSTS.initSystem(DSP);

std::vector<int> classNum = DSTS.getClassNum();

ui->tableWidget->setColumnCount(3);

ui->tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("专业").data()) << QString::fromLocal8Bit(std::string("班级").data()) << QString::fromLocal8Bit(std::string("GPA均值").data()));

ui->tableWidget->setRowCount(grade.size());

for (int i = 0, nowClass = 0; i < classNum.size(); i++) for (int j = 0, col = 0; j < classNum[i]; j++, nowClass++, col = 0) {

ui->tableWidget->setItem(nowClass, col++, new QTableWidgetItem(QString::fromLocal8Bit(DSTS.getMajorName(i + 1).data())));

ui->tableWidget->setItem(nowClass, col++, new QTableWidgetItem(QString::number(nowClass + 1)));

ui->tableWidget->setItem(nowClass, col++, new QTableWidgetItem(QString::number(grade[nowClass], 'f', 2)));

}

ui->systemState->setCurrentIndex(1);

});

connect(ui->P2\_PBclearDiver, &QPushButton::clicked, [=]() mutable {

DSTS.deleteSystem();

ui->systemState->setCurrentIndex(0);

});

connect(ui->P2\_PBQuery, &QPushButton::clicked, [=]() mutable {

if (!DSTS.isInit) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("未检测到分流后数据").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

int nowID = ui->P2\_LE\_IDQuery->text().toInt();

D\_S\_StudentPro now = DSTS.getArrayNodeByID(nowID);

if (now.ID == -1) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("未找到该学号学生").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

ui->P2\_LEmajorModify->setText(QString::fromLocal8Bit(DSTS.getMajorName(now.choice[0].first).data()));

ui->P2\_LEclassModify->setText(QString::number(now.choice[0].second));

});

connect(ui->P2\_PBModify, &QPushButton::clicked, [=]() mutable {

if (!DSTS.isInit) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("未检测到分流后数据").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

int nowID = ui->P2\_LE\_IDQuery->text().toInt();

if (nowID == 0) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("未找到该学号学生").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

if (ui->P2\_LEmajorModify->text().isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("专业名不可以为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

if (ui->P2\_LEclassModify->text().isEmpty()) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("班级不可以为空").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

int majorPos = DSTS.getMajorPos(std::string(ui->P2\_LEmajorModify->text().toLocal8Bit()));

if (majorPos == 0) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("专业名有误").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

std::pair<int, int> classPos = DSTS.getMajorClassPos(majorPos);

if (ui->P2\_LEclassModify->text().toInt() < classPos.first + 1 || ui->P2\_LEclassModify->text().toInt() > classPos.first + classPos.second) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("班级号有误").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

DSTS.modifyArrayNodeByID(nowID, majorPos, ui->P2\_LEclassModify->text().toInt());

DSTS.sortSystem();

ui->P2\_LEmajorModify->clear();

ui->P2\_LEclassModify->clear();

});

//数据查看界面：

connect(ui->mainPB3, &QPushButton::clicked, [=]() mutable {

if (!DSTS.isInit) {

ui->P3\_tableWidget->setColumnCount(1);

ui->P3\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("未检测到分流后学生列表").data()));

return;

}

std::vector<D\_S\_StudentPro> DSSarray = DSTS.getArray();

ui->P3\_tableWidget->setColumnCount(5);

ui->P3\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("专业").data()) << QString::fromLocal8Bit(std::string("班级").data()) << QString::fromLocal8Bit(std::string("排名").data()) << QString::fromLocal8Bit(std::string("学号").data()) << QString::fromLocal8Bit(std::string("绩点").data()));

ui->P3\_tableWidget->setRowCount(DSSarray.size());

for (int i = 0, col = 0; i < DSSarray.size(); i++, col = 0) {

ui->P3\_tableWidget->setItem(i, col++, new QTableWidgetItem(QString::fromLocal8Bit(DSTS.getMajorName(DSSarray[i].choice[0].first).data())));

ui->P3\_tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(DSSarray[i].choice[0].second)));

ui->P3\_tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(DSSarray[i].rank)));

ui->P3\_tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(DSSarray[i].ID)));

ui->P3\_tableWidget->setItem(i, col++, new QTableWidgetItem(QString::number(DSSarray[i].GPA, 'f', 2)));

}

});

connect(ui->P3\_PBstartFilt, &QPushButton::clicked, [=]()mutable {

if (!DSTS.isInit) {

ui->P3\_tableWidget->setColumnCount(1);

ui->P3\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("未检测到分流后学生列表").data()));

return;

}

int lmt\_major = -1, lmt\_class = -1, lmt\_ID = -1, lmt\_rank = -1, nowRow = 0;

std::vector<D\_S\_StudentPro> DSSarray = DSTS.getArray();

std::string majorNameStr(ui->P3\_LElimitedMajor->text().toLocal8Bit());

QString lmt\_classStr = ui->P3\_LElimitedClass->text();

QString lmt\_IDStr = ui->P3\_LElimitedID->text();

QString lmt\_rankStr = ui->P3\_LElimitedRank->text();

if (!majorNameStr.empty()) lmt\_major = DSTS.getMajorPos(majorNameStr);

if (!lmt\_classStr.isEmpty()) lmt\_class = lmt\_classStr.toInt();

if (!lmt\_IDStr.isEmpty()) lmt\_ID = lmt\_IDStr.toInt();

if (!lmt\_rankStr.isEmpty()) lmt\_rank = lmt\_rankStr.toInt();

ui->P3\_tableWidget->setColumnCount(5);

ui->P3\_tableWidget->setRowCount(nowRow);

ui->P3\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("专业").data()) << QString::fromLocal8Bit(std::string("班级").data()) << QString::fromLocal8Bit(std::string("排名").data()) << QString::fromLocal8Bit(std::string("学号").data()) << QString::fromLocal8Bit(std::string("绩点").data()));

for (int i = 0, col = 0; i < DSSarray.size(); i++, col = 0) {

if (lmt\_major != -1 && lmt\_major != DSSarray[i].choice[0].first) continue;

if (lmt\_class != -1 && lmt\_class != DSSarray[i].choice[0].second) continue;

if (lmt\_ID != -1 && lmt\_ID != DSSarray[i].ID) continue;

if (lmt\_rank != -1 && lmt\_rank != DSSarray[i].rank) continue;

ui->P3\_tableWidget->setRowCount(++nowRow);

ui->P3\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::fromLocal8Bit(DSTS.getMajorName(DSSarray[i].choice[0].first).data())));

ui->P3\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSarray[i].choice[0].second)));

ui->P3\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSarray[i].rank)));

ui->P3\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSarray[i].ID)));

ui->P3\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSarray[i].GPA, 'f', 2)));

}

});

//数据分析：

connect(ui->P4\_PBanalyse, &QPushButton::clicked, [=]()mutable {

if (!DSTS.isInit) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("未检测到分流后数据").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

int sumClassNum = 0, nowRow = 0;

std::vector<int> intVec;

std::vector<int> classNumVec = DSTS.getClassNum();

for (auto& x : classNumVec) sumClassNum += x;

DSST.build(DSTS, 1, DSTS.getStuNum());

std::string qry\_classStr(ui->P4\_LEclass->text().toLocal8Bit());

std::string qry\_majorStr(ui->P4\_LEmajor->text().toLocal8Bit());

if (!(qry\_majorStr.empty() ^ qry\_classStr.empty())) {

QMessageBox::information(NULL, QString::fromLocal8Bit(std::string("错误").data()), QString::fromLocal8Bit(std::string("查询值异常").data()), QMessageBox::Ok | QMessageBox::Cancel, QMessageBox::Ok);

return;

}

if (!qry\_classStr.empty()) {

string\_split(qry\_classStr, ";", intVec);

std::sort(intVec.begin(), intVec.end());

intVec.erase(std::unique(intVec.begin(), intVec.end()), intVec.end());

ui->P4\_tableWidget->setColumnCount(6);

ui->P4\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("专业").data()) << QString::fromLocal8Bit(std::string("班级").data()) << QString::fromLocal8Bit(std::string("GPA均值").data()) << QString::fromLocal8Bit(std::string("GPA最大值").data()) << QString::fromLocal8Bit(std::string("GPA最小值").data()) << QString::fromLocal8Bit(std::string("方差").data()));

for (auto& x : intVec) {

if (x <= 0 || x > sumClassNum) continue;

int majorCnt = 0, temp = x;

while (temp - classNumVec[majorCnt] > 0) temp -= classNumVec[majorCnt], majorCnt++;

int l = DSTS.getClassLeftRange(x) + 1, r = DSTS.getClassLeftRange(x + 1), col = 0;

D\_S\_S\_TreeNode DSSTN = DSST.query(l, r);

double averGPA = DSSTN.GPASum / double(r - l + 1);

double GPA\_S = DSST.getS(l, r, averGPA);

ui->P4\_tableWidget->setRowCount(++nowRow);

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::fromLocal8Bit(DSTS.getMajorName(majorCnt + 1).data())));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(x)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(averGPA, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSTN.GPAMax, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSTN.GPAMin, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(GPA\_S, 'f', 4)));

}

}

std::vector<std::string> strVec;

if (!qry\_majorStr.empty()) {

string\_split(qry\_majorStr, ";", strVec);

std::sort(strVec.begin(), strVec.end());

strVec.erase(std::unique(strVec.begin(), strVec.end()), strVec.end());

ui->P4\_tableWidget->setColumnCount(5);

ui->P4\_tableWidget->setHorizontalHeaderLabels(QStringList() << QString::fromLocal8Bit(std::string("专业").data()) << QString::fromLocal8Bit(std::string("GPA均值").data()) << QString::fromLocal8Bit(std::string("GPA最大值").data()) << QString::fromLocal8Bit(std::string("GPA最小值").data()) << QString::fromLocal8Bit(std::string("方差").data()));

for (auto& x : strVec) {

int majorPos = DSTS.getMajorPos(x);

if (!majorPos) continue;

int l = DSTS.getMajorLeftRange(majorPos) + 1, r = DSTS.getMajorLeftRange(majorPos + 1), col = 0;

D\_S\_S\_TreeNode DSSTN = DSST.query(l, r);

double averGPA = DSSTN.GPASum / double(r - l + 1);

double GPA\_S = DSST.getS(l, r, averGPA);

ui->P4\_tableWidget->setRowCount(++nowRow);

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::fromLocal8Bit(DSTS.getMajorName(majorPos).data())));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(averGPA, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSTN.GPAMax, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(DSSTN.GPAMin, 'f', 2)));

ui->P4\_tableWidget->setItem(nowRow - 1, col++, new QTableWidgetItem(QString::number(GPA\_S, 'f', 4)));

}

}

DSST.deleteTree();

});

}

* DivergenceSystem.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权：长安大学

\* 编写者：刘文越，王子卓，肖芷馨，孙翔，康旭

\*

\* 编写时间：2022/12/06

\* 功能描述：DivergenceSystem头文件中所有函数的实现

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "DivergenceSystem.h"

#include <ctime>

#include <algorithm>

//Fuctions:

void string\_split(const std::string& str, const std::string& split, std::vector<int>& rs) {

char\* Cstr = new char[str.size() + 1];

strcpy(Cstr, str.c\_str());

char\* temp = strtok(Cstr, split.c\_str());

while (temp != NULL) {

rs.push\_back(stoi(std::string(temp)));

temp = strtok(NULL, split.c\_str());

}

delete[] Cstr;

}

void string\_split(const std::string& str, const std::string& split, std::vector<std::string>& rs) {

char\* Cstr = new char[str.size() + 1];

strcpy(Cstr, str.c\_str());

char\* temp = strtok(Cstr, split.c\_str());

while (temp != NULL) {

rs.push\_back(std::string(temp));

temp = strtok(NULL, split.c\_str());

}

delete[] Cstr;

}

//D\_S\_Student:

std::istream& operator>>(std::istream& in, D\_S\_Student& DSS) {

in >> DSS.ID >> DSS.choice >> DSS.GPA >> DSS.rank;

return in;

}

std::ostream& operator<< (std::ostream& out, const D\_S\_Student& DSS) {

out << "test: " << DSS.ID << ' ' << DSS.choice << ' ' << DSS.GPA << ' ' << DSS.rank << '\n';

return out;

}

//D\_S\_DeLinkedList:

D\_S\_MajorList::D\_S\_MajorList() {

head = new D\_S\_DeLinkedListNode;

tail = new D\_S\_DeLinkedListNode;

this->head->nex = tail;

this->tail->pre = head;

}

D\_S\_MajorList::~D\_S\_MajorList() {

//D\_S\_DeLinkedListNode\* now = this->head, \* lst = nullptr;

//while (now) {

// lst = now;

// now = now->nex;

// delete lst;

//}

}

void D\_S\_MajorList::deleteMajorList() {

D\_S\_DeLinkedListNode\* now = this->head->nex, \* lst = nullptr;

while (now != this->tail) {

lst = now;

now = now->nex;

delete lst;

}

}

void D\_S\_MajorList::printList() {

D\_S\_DeLinkedListNode\* now = this->head->nex;

while (now != this->tail) {

std::cout << now->data;

now = now->nex;

}

}

void D\_S\_MajorList::initList(std::string filePath) {

D\_S\_Student now;

std::ifstream fin(filePath, std::ios::in);

if (!fin) {

std::cout << "Failed to open the file !\n";

return;

}

std::string str;

fin >> str >> str >> str >> str;

while (fin >> now.ID) {

fin >> str;

if (str == "第一志愿") now.choice = 1;

else if (str == "第二志愿") now.choice = 2;

else if (str == "第三志愿") now.choice = 3;

else if (str == "第四志愿") now.choice = 4;

else if (str == "第五志愿") now.choice = 5;

else now.choice = 0;

fin >> now.GPA >> now.rank;

D\_S\_DeLinkedListNode\* nowptr = new D\_S\_DeLinkedListNode;

nowptr->data = now;

nowptr->nex = this->tail, nowptr->pre = this->tail->pre;

this->tail->pre->nex = nowptr, this->tail->pre = nowptr;

this->nodeNum++;

}

}

void D\_S\_MajorList::addNode(D\_S\_Student DSS) {

D\_S\_DeLinkedListNode\* nowptr = new D\_S\_DeLinkedListNode;

nowptr->data = DSS;

nowptr->nex = this->tail, nowptr->pre = this->tail->pre;

this->tail->pre->nex = nowptr, this->tail->pre = nowptr;

this->nodeNum++;

}

void D\_S\_MajorList::setListName(std::string name) {

this->majorName = name;

}

int D\_S\_MajorList::getNodeNum() {

return this->nodeNum;

}

//D\_S\_PretreatSystem:

void D\_S\_PretreatSystem::initSystem() {

for (int i = 0; i < this->listNum; i++) this->MajorList[i].deleteMajorList();

this->MajorList.clear();

this->mp.clear();

this->listNum = 0;

}

int D\_S\_PretreatSystem::addMajorList(std::string filePath, std::string majorName, std::pair<int, int> pairNum) {

D\_S\_MajorList DSM;

DSM.initList(filePath);

DSM.setListName(majorName);

DSM.maxNum = pairNum;

this->MajorList.push\_back(DSM);

this->listNum++;

return DSM.getNodeNum();

}

D\_S\_Student D\_S\_PretreatSystem::getNodeByPos(int pos1, int pos2) {

D\_S\_DeLinkedListNode\* now = this->MajorList[pos1 - 1].head->nex;

int posCnt = 0;

while (now != this->MajorList[pos1 - 1].tail) {

if (++posCnt == pos2) return now->data;

else now = now->nex;

}

}

int D\_S\_PretreatSystem::getListNum() {

return this->listNum;

}

int D\_S\_PretreatSystem::getListNodeNum(int cnt) {

return this->MajorList[cnt - 1].getNodeNum();

}

int D\_S\_PretreatSystem::getSumNum() {

int rs = 0;

for (int i = 0; i < this->listNum; i++) {

D\_S\_DeLinkedListNode\* now = this->MajorList[i].head->nex;

while (now != this->MajorList[i].tail) {

if (!mp[now->data.ID]) rs++, mp[now->data.ID] = true;

now = now->nex;

}

}

mp.clear();

return rs;

}

//D\_S\_StudentPro:

std::istream& operator>>(std::istream& in, D\_S\_StudentPro& DSSP) {

in >> DSSP.choice[0].first >> DSSP.choice[0].second >> DSSP.rank >> DSSP.ID >> DSSP.GPA;

return in;

}

std::ostream& operator<<(std::ostream& out, const D\_S\_StudentPro& DSSP) {

out << DSSP.choice[0].first << ' ' << DSSP.choice[0].second << ' ' << DSSP.rank << ' ' << DSSP.ID << ' ' << DSSP.GPA << '\n';

return out;

}

//D\_S\_TreatedSystem:

std::vector<double> D\_S\_TreatedSystem::initSystem(D\_S\_PretreatSystem& DSP) {

//分流操作：

int sumNum = DSP.getSumNum(), nowPos = 1;

std::map<int, int> ID2Pos;

D\_S\_StudentPro fir;

fir.choice.push\_back({ 0, 0 });

DSSParray.push\_back(fir);

for (int i = 0; i < DSP.getListNum(); i++) {

D\_S\_DeLinkedListNode\* now = DSP.MajorList[i].head->nex;

while (now != DSP.MajorList[i].tail) {

if (!ID2Pos[now->data.ID]) {

ID2Pos[now->data.ID] = DSSParray.size();

D\_S\_StudentPro DSSP;

DSSP.ID = now->data.ID, DSSP.GPA = now->data.GPA, DSSP.rank = now->data.rank;

DSSP.choice.push\_back({ now->data.choice, i + 1 });

DSSParray.push\_back(DSSP);

}

else DSSParray[ID2Pos[now->data.ID]].choice.push\_back({ now->data.choice, i + 1 });

now = now->nex;

}

}

for (auto& x : this->DSSParray) std::sort(x.choice.begin(), x.choice.end());

std::sort(DSSParray.begin(), DSSParray.end(), [](D\_S\_StudentPro& a, D\_S\_StudentPro& b) {

return a.rank < b.rank;

});

for (int i = 0; i < DSP.getListNum(); i++) this->majorName.push\_back(DSP.MajorList[i].majorName);

for (int i = 0; i < DSP.getListNum(); i++) this->maxNum.push\_back(DSP.MajorList[i].maxNum);

for (int i = 0; i < DSP.getListNum(); i++) nowMajorNum.push\_back(0);

for (int i = 0; i < DSP.getListNum(); i++) for (int j = 0; j < DSP.MajorList[i].maxNum.first; j++) nowClassNum.push\_back(0);

while (nowPos < DSSParray.size()) {

bool flag = true;

std::vector<bool> st(DSP.getListNum(), false);

for (int i = 0; i < DSP.getListNum(); i++) if (maxNum[i].first \* maxNum[i].second > nowMajorNum[i]) st[i] = true;

while (flag) {

flag = false;

int defChoice = -1;

for (auto& x : DSSParray[nowPos].choice) if (st[x.second - 1]) {

nowMajorNum[x.second - 1]++;

defChoice = x.second;

break;

}

DSSParray[nowPos].choice.clear();

if (defChoice != -1) DSSParray[nowPos].choice.push\_back({ defChoice, 0 });

if (nowPos != DSSParray.size() - 1 && DSSParray[nowPos].rank == DSSParray[nowPos + 1].rank) flag = true;

nowPos++;

}

}

for (int i = 1; i < DSSParray.size(); i++) if (DSSParray[i].choice.size() == 0) {

int minPos = -1, minNum = 0x3f3f3f3f;

for (int j = 0; j < DSP.getListNum(); j++) if (minNum > nowMajorNum[j]) minNum = nowMajorNum[j], minPos = j;

DSSParray[i].choice.push\_back({ minPos + 1, 0 });

nowMajorNum[minPos]++;

}

//分班操作：

srand(unsigned(time(nullptr)));

std::vector<int> classID;

int startPos = 1, randStartPos = 0;

for (int i = 0; i < DSP.getListNum(); i++) {

for (int j = 1; j <= nowMajorNum[i]; j++) {

classID.push\_back(j % maxNum[i].first + startPos);

nowClassNum[j % maxNum[i].first + startPos - 1]++;

}

std::random\_shuffle(classID.begin() + randStartPos, classID.end());

startPos += maxNum[i].first, randStartPos += nowMajorNum[i];

}

sort(DSSParray.begin() + 1, DSSParray.end(), [](D\_S\_StudentPro& a, D\_S\_StudentPro& b) {

return a.choice[0].first < b.choice[0].first;

});

for (int i = 1; i < DSSParray.size(); i++) DSSParray[i].choice[0].second = classID[i - 1];

sort(DSSParray.begin() + 1, DSSParray.end(), [](D\_S\_StudentPro& a, D\_S\_StudentPro& b) {

if (a.choice[0].second != b.choice[0].second) return a.choice[0].second < b.choice[0].second;

else return a.rank < b.rank;

});

int classSum = 0;

for (int i = 0; i < DSP.getListNum(); i++) classSum += maxNum[i].first;

std::vector<double> classGrade(classSum, 0.0);

std::vector<int> classNum(classSum, 0);

for (int i = 1; i < DSSParray.size(); i++) {

classGrade[DSSParray[i].choice[0].second - 1] += DSSParray[i].GPA;

classNum[DSSParray[i].choice[0].second - 1]++;

}

for (int i = 0; i < classSum; i++) classGrade[i] /= double(classNum[i]);

this->isInit = true;

return classGrade;

}

void D\_S\_TreatedSystem::initSystem(QString filePath) {

this->isInit = true;

std::ifstream fcin(filePath.toStdString());

int Size = 0;

fcin >> Size;

while (Size--) {

D\_S\_StudentPro now;

std::pair<int, int> nowPair;

fcin >> nowPair.first >> nowPair.second >> now.rank >> now.ID >> now.GPA;

now.choice.push\_back(nowPair);

this->DSSParray.push\_back(now);

}

fcin >> Size;

while (Size--) {

std::string now;

fcin >> now;

this->majorName.push\_back(now);

}

fcin >> Size;

while (Size--) {

std::pair<int, int> now;

fcin >> now.first >> now.second;

this->maxNum.push\_back(now);

}

fcin >> Size;

while (Size--) {

int now;

fcin >> now;

this->nowMajorNum.push\_back(now);

}

fcin >> Size;

while (Size--) {

int now;

fcin >> now;

this->nowClassNum.push\_back(now);

}

fcin.close();

}

void D\_S\_TreatedSystem::saveSystem(QString filePath) {

filePath += "/DivergenceSystemDataFile.txt";

std::ofstream fout(std::string(filePath.toLocal8Bit()));

fout << this->DSSParray.size() << '\n';

for (auto& x : this->DSSParray) fout << x;

fout << this->majorName.size() << '\n';

for (auto& x : this->majorName) fout << x << '\n';

fout << this->maxNum.size() << '\n';

for (auto& x : this->maxNum) fout << x.first << ' ' << x.second << '\n';

fout << this->nowMajorNum.size() << '\n';

for (auto& x : this->nowMajorNum) fout << x << '\n';

fout << this->nowClassNum.size() << '\n';

for (auto& x : this->nowClassNum) fout << x << '\n';

fout.close();

}

void D\_S\_TreatedSystem::deleteSystem() {

this->isInit = false;

this->DSSParray.clear();

this->majorName.clear();

this->maxNum.clear();

this->nowMajorNum.clear();

this->nowClassNum.clear();

}

std::vector<D\_S\_StudentPro> D\_S\_TreatedSystem::getArray() {

std::vector<D\_S\_StudentPro> rs;

for (int i = 1; i < this->DSSParray.size(); i++) rs.push\_back(DSSParray[i]);

return rs;

}

D\_S\_StudentPro D\_S\_TreatedSystem::getArrayNode(int Pos) {

return this->DSSParray[Pos];

}

D\_S\_StudentPro D\_S\_TreatedSystem::getArrayNodeByID(int tmpID) {

for (auto& x : this->DSSParray) if (x.ID == tmpID) return x;

return DSSParray[0];

}

std::string D\_S\_TreatedSystem::getMajorName(int pos) {

return majorName[pos - 1];

}

int D\_S\_TreatedSystem::getMajorPos(std::string str) {

for (int i = 0; i < this->majorName.size(); i++) if (this->majorName[i] == str) return i + 1;

return 0;

}

std::vector<int> D\_S\_TreatedSystem::getClassNum() {

std::vector<int> rs;

for (auto& x : this->maxNum) rs.push\_back(x.first);

return rs;

}

std::pair<int, int> D\_S\_TreatedSystem::getMajorClassPos(int majorPos) {

std::pair<int, int> rs = { 0, -1 };

for (int i = 0; i < majorPos - 1; i++) rs.first += this->maxNum[i].first;

rs.second = maxNum[majorPos - 1].first;

return rs;

}

int D\_S\_TreatedSystem::getStuNum() {

int rs = 0;

for (auto& x : nowMajorNum) rs += x;

return rs;

}

bool D\_S\_TreatedSystem::modifyArrayNodeByID(int tmpID, int majorPos, int classPos) {

for (auto& x : this->DSSParray) if (x.ID == tmpID) {

x.choice[0].first = majorPos, x.choice[0].second = classPos;

return true;

}

return false;

}

void D\_S\_TreatedSystem::sortSystem() {

sort(DSSParray.begin() + 1, DSSParray.end(), [](D\_S\_StudentPro& a, D\_S\_StudentPro& b) {

if (a.choice[0].first != b.choice[0].first) return a.choice[0].first < b.choice[0].first;

else if (a.choice[0].second != b.choice[0].second) return a.choice[0].second < b.choice[0].second;

else return a.rank < b.rank;

});

}

int D\_S\_TreatedSystem::getClassLeftRange(int x) {

int rs = 0;

for (int i = 0; i < x - 1; i++) rs += this->nowClassNum[i];

return rs;

}

int D\_S\_TreatedSystem::getMajorLeftRange(int x) {

int rs = 0;

for (int i = 0; i < x - 1; i++) rs += this->nowMajorNum[i];

return rs;

}

void pushup(D\_S\_S\_TreeNode\*& x) {

x->GPASum = x->left->GPASum + x->right->GPASum;

x->GPAMax = std::max(x->left->GPAMax, x->right->GPAMax);

x->GPAMin = std::min(x->left->GPAMin, x->right->GPAMin);

}

void deleteNode(D\_S\_S\_TreeNode\*& x) {

if (x->left != nullptr) deleteNode(x->left);

if (x->right != nullptr) deleteNode(x->right);

delete x;

}

//D\_S\_S\_TreeNode:

std::ostream& operator<<(std::ostream& out, const D\_S\_S\_TreeNode& DSSTN) {

out << DSSTN.GPASum << ' ' << DSSTN.GPAMax << ' ' << DSSTN.GPAMin << '\n';

return out;

}

//D\_S\_SegmentTree:

void D\_S\_SegmentTree::build(D\_S\_TreatedSystem& DSTS, int l, int r) {

if (r == 0) return;

root = new D\_S\_S\_TreeNode;

root->l = l, root->r = r;

if (l == r) {

root->GPASum = DSTS.getArrayNode(l).GPA;

root->GPAMin = DSTS.getArrayNode(l).GPA;

root->GPAMax = DSTS.getArrayNode(l).GPA;

}

else {

int mid = l + r >> 1;

build(DSTS, root->left, l, mid);

build(DSTS, root->right, mid + 1, r);

pushup(root);

}

}

void D\_S\_SegmentTree::build(D\_S\_TreatedSystem& DSTS, D\_S\_S\_TreeNode\*& x, int l, int r) {

x = new D\_S\_S\_TreeNode;

x->l = l, x->r = r;

if (l == r) {

x->GPASum = DSTS.getArrayNode(l).GPA;

x->GPAMin = DSTS.getArrayNode(l).GPA;

x->GPAMax = DSTS.getArrayNode(l).GPA;

}

else {

int mid = l + r >> 1;

build(DSTS, x->left, l, mid);

build(DSTS, x->right, mid + 1, r);

pushup(x);

}

}

D\_S\_S\_TreeNode D\_S\_SegmentTree::query(D\_S\_S\_TreeNode\* x, int l, int r) {

if (l <= x->l && r >= x->r) return \*x;

D\_S\_S\_TreeNode rs, son;

int mid = x->l + x->r >> 1;

if (l <= mid) {

son = query(x->left, l, r);

rs.GPASum += son.GPASum;

rs.GPAMax = std::max(rs.GPAMax, son.GPAMax);

rs.GPAMin = std::min(rs.GPAMin, son.GPAMin);

}

if (r > mid) {

son = query(x->right, l, r);

rs.GPASum += son.GPASum;

rs.GPAMax = std::max(rs.GPAMax, son.GPAMax);

rs.GPAMin = std::min(rs.GPAMin, son.GPAMin);

}

return rs;

}

D\_S\_S\_TreeNode D\_S\_SegmentTree::query(int l, int r) {

if (l <= this->root->l && r >= this->root->r) return \*(this->root);

D\_S\_S\_TreeNode rs, son;

int mid = this->root->l + this->root->r >> 1;

if (l <= mid) {

son = query(this->root->left, l, r);

rs.GPASum += son.GPASum;

rs.GPAMax = std::max(rs.GPAMax, son.GPAMax);

rs.GPAMin = std::min(rs.GPAMin, son.GPAMin);

}

if (r > mid) {

son = query(this->root->right, l, r);

rs.GPASum += son.GPASum;

rs.GPAMax = std::max(rs.GPAMax, son.GPAMax);

rs.GPAMin = std::min(rs.GPAMin, son.GPAMin);

}

return rs;

}

double D\_S\_SegmentTree::getS(int l, int r, double x) {

double Sum = 0;

for (int i = l; i <= r; i++) Sum += (this->query(i, i).GPASum - x) \* (this->query(i, i).GPASum - x);

return Sum / double(r - l + 1);

}

void D\_S\_SegmentTree::deleteTree() {

deleteNode(this->root);

this->root = nullptr;

}

* main.cpp

#include "DataStructProgram.h"

#include <QtWidgets/QApplication>

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

{

QApplication a(argc, argv);

DataStructProgram w;

w.show();

return a.exec();

}