



北京邮电大学

Beijing University of Posts and Telecommunications

数据结构课程实践

学院：计算机学院

[组员：马天成]

[组员：王宸]

[组员：王陆萱]

2022 年 6 月 15 日

目录

1.实验内容和实验环境描述	4
1.1 实验内容	4
1.2 实验环境	4
2.功能设计和模块展示	4
2.1 模块一：登录	4
2.2 模块二：注册	5
2.3 模块三：主界面	6
2.4 模块四：导航	6
2.5 模块五：课程管理	7
2.6 模块六：活动管理	9
3.核心代码设计和算法解析	10
全局变量数据结构	10
3.1 登录模块	11
3.1.1 登陆验证函数	11
3.1.2 规定程序运行路径	11
3.1.3 日志初始化	12
3.2 注册模块	12
3.2.1 搜索历史账号避免冲突函数	13
3.2.2 用户注册表项	13
3.3 桌面模块	13
3.3.1 创建系统时间并开启线程	14
3.3.2 二级用户回档	14
3.4 地图模块	14
3.4.1 数据结构抽象	15
3.4.2 地图搜索算法——贪心	17
3.4.3 搜索算法应用	18
3.5 活动模块	20
3.5.1 数据结构抽象	20
3.5.2 建立个人活动链表	21
3.5.3 建立集体活动链表	22
3.5.4 添加活动	22

3.5.5 查询活动	22
3.5.6 活动筛选	23
3.5.7 活动排序	24
3.6 课程模块	24
3.6.1 数据结构抽象.....	24
3.6.2 获取课程信息.....	25
3.6.3 课程信息	26
3.6.4 添加选修课程——学生功能.....	26
3.6.5 删除课程	27
3.6.6 作业&资料	27
3.6.7 课程模块——管理员特殊功能.....	29
3.7 日志模块	34
3.8 课程作业压缩解压储存.....	35
【一】使用说明	35
【二】细节分析	35
【三】代码截图分析	36
查重函数.....	40
4.实验亮点	43
4.1 选中 Qt 开发，进行良好的 UI 设计.....	43
4.2 极大程度降低用户输入.....	43
4.3 准确性、性能、算法优势	43
4.3.1 贪心地图算法.....	43
4.3.2 不定量规模数据封装.....	44
4.3.3 额外功能多，优化好.....	44
5.实验总结	44
5.1 用时	44
5.2 设计体会	44
5.3 问题评价与改进意见	45

1.实验内容和实验环境描述

1.1 实验内容

大学中每位同学每学期会有多门不同的课程, 课程分布在不同的教学楼甚至不同的校区; 每门课程都会有一些课程资料、作业、考试和课程群等内容; 此外每位同学在课外还会有一些个人或者集体的活动安排。

线下课程辅助系统可以帮助学生管理自己的课程和课外活动, 具备课程导航功能、课程信息管理和查询功能, 以及课外信息管理和查询功能等。每天晚上系统会提醒学生第二天上的课, 每门课需要交的作业和需要带的资料, 以及考试的信息; 快要上课时系统根据该课程的上课地点设计一条最佳线路并输出; 学生可以通过系统管理每门课的学习资料、作业和考试信息; 在课外, 学生可以管理自己的个人活动和集体活动信息, 可以进行活动时间的冲突检测和闹钟提醒。

1.2 实验环境

qt5.15.2、数据库 MySQL (测试开发用的本地数据库, 正式发行是内网穿透的数据库)
开发语言: C++

2.功能设计和模块展示

共六大模块: 登陆、注册、主界面、导航、课程、活动。附有两个小模块: 日志, 压缩。

2.1 模块一: 登录



- **验证功能**

点击 **Sign in** 按钮, 进行身份认证。假如用户名(Account)和密码>Password)根据数据库数据匹配认证成功, 则开启新窗口 **widget_desktop**, 创建桌面窗口, 进行各种功能的展示。

假如输入不正确, 则清空密码输入, 并将光标定位在密码输入位置, 增加用户体验。

- **跳转到注册界面**

假如需要注册新用户, 则点击 **Sign up** 按钮, 跳转到注册界面进行相关操作。假如注册成功, 那么返回此界面时会在账户编辑行注入刚申请的账号, 增加用户体验。

- **回档功能**

根据程序执行目录下的 **Logs/state.log** 数据, 取出用户的登陆数据, 进行回档交互: 假如用户点击确定回档, 则直接对系统注册相关数据, 直接进入桌面服务。此外, 还有**二级回档信息在桌面进行交互申请**, 此界面为登陆信息的回档和主界面的展示。

- **连接数据库**

有时候数据库连接不够稳定, 只需要按下左上角的 link 即可刷新状态。

2.2 模块二: 注册

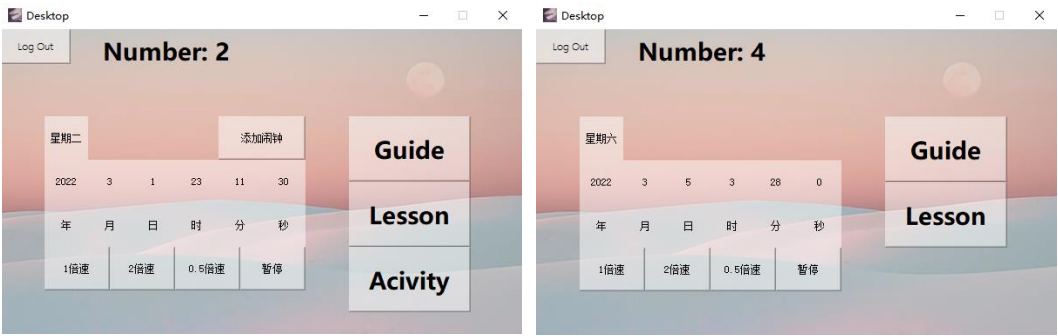


- **注册功能: 用户输入四个数据进行注册实现**

- 账号为学号 账号作为唯一标识不能重复注册
- 密码 0~20 位字符 大小写区分 不能含有空格输入
- 用户名
- 班级 必选

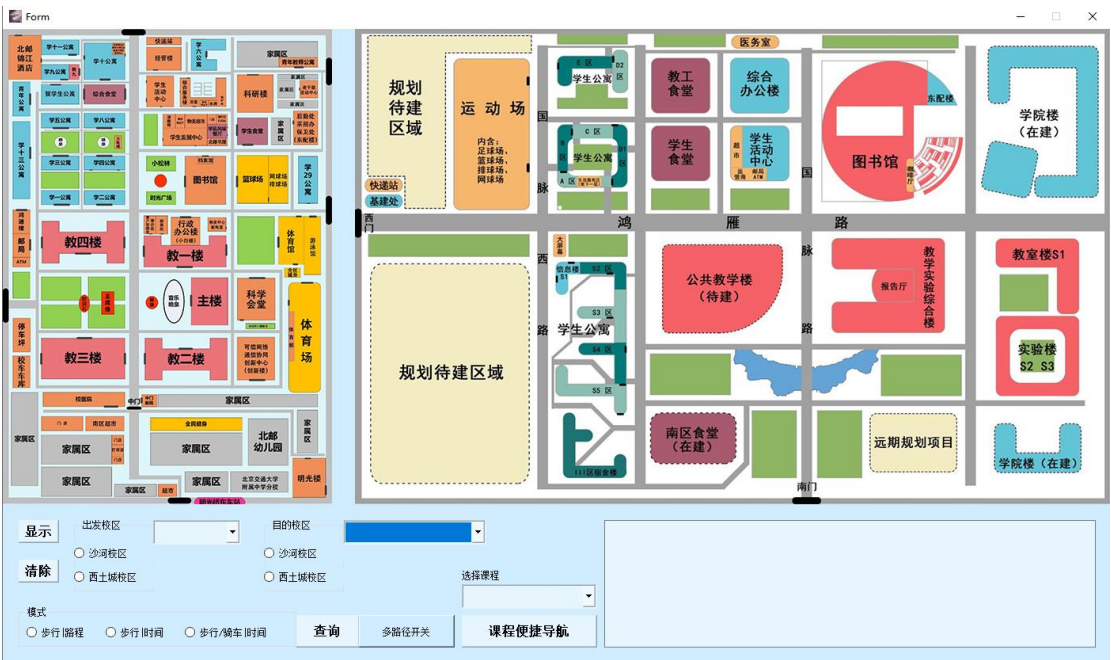
2.3 模块三：主界面

用户界面和管理员界面：



- 点击进入地图、活动、课程模块
点击对应按钮，则进入相应页面（管理员无法进入闹钟、活动，但可以进入导航和课程）
- 时间模块
 - 时间显示 显示系统时间线的年月日时分秒
 - 时间进度管理[1 倍速]、[2 倍速]、[0.5 倍速]、[暂停]
 - 报时设置 系统整点提示框报时
 - 闹钟设置 点击[添加闹钟] 设置数据 点击[添加日常闹钟] 成功返回提醒
- 回档模块
假如当前登录的帐号有历史浏览记录，则询问是否进行继续浏览；选择是则进行继续浏览。

2.4 模块四：导航



- 点位显示与清除

- 显示-显示地图所有点位 (label 形式)
- 清除-清除地图所有点位 (label 形式)

● 导航——普通导航

- 选择出发校区的点位和目的校区的点位
- 选择不同的 mode 进行不同的策略选择

● 导航——课程便捷查询

- 若没有选中任何课程，则自动选择最近的课程进行导航
- 若有选中课程（包含课程名称，时间信息），则导航该课程

● 导航——多路径导航

界面多路径开关可按下，在多路径模式开启状态，用户输入普通导航后，目的地会自动选择为下一个初始地，方便用户多次导航，并将所有的路程信息显示在界面地图上。

2.5 模块五：课程管理

✧ 用户界面

The screenshot shows a web application titled 'Lesson'. It features a weekly course schedule table with columns for days of the week (星期一至星期日) and rows for different courses. The courses listed include '形式语言与自动机', '数据结构课程设计', '计算机网络', '毛概', '数据结构补充课程', '计算机组成原理', '数据结构习题讲解', '数字逻辑课程设计', and '数字逻辑课程'. To the right of the table is a search bar with a '查找' (Search) button and a '添加该选修课程' (Add this elective course) button. Below the table, there are sections for '当前进度: 1 / 16' (Current progress), '上课地点' (Class location), '课程群' (Course group), '未截止作业' (Unsubmitted assignments), and '已截止作业' (Submitted assignments). There are also buttons for '资料功能' (Resource function) and '提交作业' (Submit assignment). At the bottom, there is a '课程信息检索' (Course information search) section with a search bar and buttons for '按名称排序' (Sort by name) and '按时间排序' (Sort by time).

● 查找课程

- 在课程信息检索框中输入检索信息（可为空）回车
- 下方显示查找的课程
- 点击[按时间排序]和[按名称排序]对课程进行检索
- 下方显示检索的课程

● 选课与退选（选修课）

- 输入查询信息 点击[查找]
- 从下拉框中选择课程 点击[添加该选修课]
- 课表中选择课程
- 点击[删除该课程];

● 单项课程信息

- 查看信息 点击课程对应按钮，课程信息显示左下方数据框中
- 提交作业 在未截止作业下拉框中选取待提交作业 点击[提交作业]进入文件功能 选取文件 点击[上传查重]
- 提交资料 点击[提交作业]进入文件功能 选取文件 点击[上传查重]

✧ 管理员界面

Lesson

星期一	星期二	星期三	星期四	星期五	星期六	星期日
数据结构课程设计		数据结构课程设计				数据结构补充课程
					数据结构习题讲解	数据结构
				数据结构讲解课程		

管理员添加课程
管理员修改课程
管理员发布作业
管理员发布考试

当前进度: 1 / 16

上课地点

课程群

未截止作业

已截止作业

考试信息

课程信息检索

按名称排序 按时间排序

● 查看课表信息和检索

- 点击课程按钮，获得相应课程信息

● 添加课程

- 点击[管理员添加课程]进入课程设置模块
- 填写选择相关信息，
- 点击[添加必修课程]或[添加选修课程]实现添加课程功能

● 修改课程

- 选择可修改的课程
- 填写选取修改课程信息
- 点击[修改]实现修改功能

● 发布作业

- 选择有管理权限的课程
- 输入作业名称，截至日期，点击[添加]，完成课程作业添加功能

- 添加完成后，在课程表上点击该课程，在下方未截至作业中可看作获取业情况（已经上交份数）

- **发布考试**

- 选择有管理权限的课程
- 输入考试名称，开始时间，和截至时间,点击[添加],完成课程作业添加功能

2.6 模块六：活动管理

- **查询活动**

- 点击[查询个人活动]显示当前个人活动的时间和内容
- 点击[查询集体活动]显示当前集体活动的时间和内容

- **活动添加**

- 通过面板上设置新建方活动名称、类型、校区和时间
- 点击[添加活动]增加活动

- **活动筛选**

- 活动可作为标签筛选
- 点击[时间排序]和[名称排序], 对个人活动和集体活动进行排序,结果展示在文本框中
- 输入查询内容，点击[关键字段按钮],对个人活动和集体活动进行排序进行查询,结果展示在文本框中

3.核心代码设计和算法解析

全局变量数据结构

✧ 系统时间变量

```
extern FullTime system_Time;  
extern LARGE_INTEGER cpuFreq;  
extern LARGE_INTEGER starter;  
extern LARGE_INTEGER ender;  
extern double v;  
extern bool handle;
```

- FullTime 包含八个数值：年月日时分秒星期周数
- 1 调用 CPU 秒跳数，2、3 分别调用系统 CPU 跳数进行计数，而进行精确的时间模拟
- v 代表时间流逝速度，通过按钮修改相应的值对时间的计数进行控制
- handle 代表句柄。因为用线程跑时间模拟，死循环需要用 handle 来控制

✧ 数据库变量

```
extern QSqlDatabase db;
```

代表数据库变量，全局使用

✧ 全局 HOME 路径变量

```
extern QDir HOME;
```

代表所有文件的存储路径，日志类使用，系统时间初始化使用

✧ 全局用户变量

```
extern int User_Number;  
extern QString User_Account;  
extern int User_IsAdmin;
```

在登陆或者回档时初始化的数据，便于全局使用

✧ 全局日志变量

```
extern Log globalLog;
```

方便调用，记录日志简单

✧ 全局函数

```
bool isEnded(int year, int month, int day, int hour);  
bool isBefore(int y1, int m1, int d1, int h1, int y2, int m2, int d2, int h2);
```

全局判断：

输入时间是否过期（超过系统时间）

两个时间哪个更早

3.1 登录模块

登录窗口

当选择登录的时候,用户输入学号和密码, 进行验证, 连接数据库, 查看输入内容是否和数据库中记录的学号和密码对应。若验证成功, 关联登录界面和主界面, 打开主界面, 清空输入框中内容, 否则的话显示 Wrong account or password!

3.1.1 登陆验证函数

```
bool Widget::check(QString account, QString password)
{
    bool answer = false;
    QSqlQuery query;
    query.exec("select * from people");
    while( query.next() ) {
        if(account == query.value(1).toString())
            if(password == query.value(2).toString()) {
                User_Account = account;
                User_Number = query.value(0).toInt();
                User_IsAdmin = query.value(3).toInt();
                answer = true;
                break;
            }
    }
    return answer;
}
```

在数据库中找到相应的账号和密码, 然后返回搜索结果。一旦匹配到账号, 且密码不正确, 那么会立刻跳出循环而不用遍历整个表项, 节约时间。

思路: 因为用户表是顺序表, 并不能反映账号的顺序, 无法快速查找, 故用遍历。

3.1.2 规定程序运行路径

```
// 规定HOME路径
HOME.setPath( QApplication::applicationDirPath() );
```

首先规定 HOME 路径, HOME 保存运行程序的绝对路径。

3.1.3 日志初始化

```
// 创建日志对象
globalLog.create();
// 说明曾经异常退出, 先直接开启desktop
if(globalLog.getUserNumber() != 0) {
    QMessageBox msgBox;
    msgBox.setText("通知: 有历史登录");
    msgBox.setInformativeText("是否继续上次登录? ");
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Ok);
    int ret = msgBox.exec();

    if(ret == QMessageBox::Ok){
        User_Number = globalLog.getUserNumber();
        QSqlQuery query;
        query.exec("select * from people");
        while( query.next() )
            if(User_Number == query.value(0).toInt()) {
                User_Account = query.value(1).toString();
                User_IsAdmin = query.value(3).toInt();
                break;
            }

        // 验证成功, 开启主界面
        this->desktop = new Widget_Desktop ;
        // close MainPage and show this
        connect(desktop, &Widget_Desktop::exit_return, [=]() {
            delete desktop;
            desktop = nullptr;
            this->show();
        });
        desktop->show() ;
        this->close();

        // 日志 *****
        globalLog.writeLog("打开程序时进行回档操作");
    }
    else
        globalLog.deletePage(); // 不需要回档, 后面的页面数据也没用
}
```

创建日志对象, 即写文件操作, 向 user.log 写用户记录, state.log 当出现异常退出的时候, 保存登录状态, 当 state.log 说明曾经异常退出后, 直接开启 desktop 主界面。

3.2 注册模块

注册窗口

当选择注册的时候, 记录 4 个数据属性, 账号, 密码, 姓名和班级, 当判断该账号有没有注册, 若没有注册, 生成插入语句, 向数据库内插入新注册的学员信息, 生成用户表, 然后记录账号信息, 显示 Success! "Close for continue.这时返回登录界面, 账号输入框自动生成新注册的账号。设置要求账号长度为 10 且密码长度小于 20。如果已经注册弹出 Warning!

This account exists.

3.2.1 搜索历史账号避免冲突函数

```
// 判断该账号有没有被注册
int flag = 0;
query.exec("select * from people");
while(query.next()) {
    if(account == query.value(1).toString()) {
        flag = 1;
        break;
    }
}
```

假如账号被注册，直接返回报错。

3.2.2 用户注册表项

```
// 生成用户表
// _clock
str = "create table "+QString::number(num)+"_clock";
str += "(name VARCHAR(255) NOT NULL, year INT NOT NULL, month INT NOT NULL, day INT NOT NULL, hour INT NOT NULL)";
query.exec(str);
QDebug() << str;

// _clock_dayly
str = "create table "+QString::number(num)+"_clock_dayly";
str += "(name VARCHAR(255) NOT NULL, hour INT NOT NULL)";
query.exec(str);
QDebug() << str;

// _clock_weekly
str = "create table "+QString::number(num)+"_clock_weekly";
str += "(name VARCHAR(255) NOT NULL, week INT NOT NULL, hour INT NOT NULL)";
query.exec(str);
QDebug() << str;

// _once
str = "create table "+QString::number(num)+"_once";
str += "(name VARCHAR(255) NOT NULL, isShaHe INT NOT NULL, location INT NOT NULL, ";
str += "year INT NOT NULL, month INT NOT NULL, day INT NOT NULL, start_hour INT NOT NULL, end_hour INT NOT NULL, label VARCHAR(255))";
query.exec(str);
QDebug() << str;

// _select
str = "create table "+QString::number(num)+"_select";
str += "(number INT NOT NULL, name VARCHAR(255) NOT NULL, teacher INT NOT NULL, ";
str += "isShaHe INT NOT NULL, location INT NOT NULL, dayOfWeek INT NOT NULL, c_index INT NOT NULL, qun VARCHAR(255), extra VARCHAR(255))";
query.exec(str);
QDebug() << str;

// _task
str = "create table "+QString::number(num)+"_task";
str += "(number INT NOT NULL, lesson_number INT NOT NULL, isSelect INT NOT NULL, name VARCHAR(255) NOT NULL, ";
str += "year INT NOT NULL, month INT NOT NULL, day INT NOT NULL, hour INT NOT NULL, isCommit INT NOT NULL)";
query.exec(str);
QDebug() << str;
```

向数据库申请当前用户需要的数据表。

3.3 桌面模块

Desktop 窗口

进入主界面后，有几个按钮到达子界面：guide 打开后即地图界面 lesson 课程界面 clock 设置时钟 activity 活动界面

3.3.1 创建系统时间并开启线程

```
// thread for time counts and update
th = new MyThread(this);
connect(th, &MyThread::refresh, this, &Widget_Desktop::putTime);
connect(th, &MyThread::hourPass, this, &Widget_Desktop::ring);
th->start();
```

对于当前父窗口创建时间线程，然后去跑死循环，时间计数。其工作原理如下：

主界面有时间显示，通过只读模式读取 time.log 文件，如果可以打开，读取数据到全局变量，如果数据数量不等于 8（年、月、日、时、分、秒、周数，星期数）或者内容不合法，删除重开；如果不能通过只读模式打开文件即文件不存在的情况，那么创建 time.log 文件并将初始化数据写入。时间的更新是通过线程来实现的，通过获取 cpu 起始跳数和线程运行后的当前跳数来实现秒数的增加，程序模拟的是 10 秒一跳，返回 refresh 信号，运行 putTime 函数更新时间。程序运行当小时数发生变化，返回一个 hourpass 信号。时间更新对应闹钟功能，程序设置了三种闹钟，单次闹钟对应年月日和小时，日常闹钟对应小时，周期闹钟对应星期数和小时。

3.3.2 二级用户回档

```
// 根据日志进行场景还原
globalLog.maintainState(QString::number(User_Number));
if(!globalLog.getPage().isEmpty() && globalLog.getUserNumber() == User_Number) {
    QMessageBox msgBox;
    msgBox.setText("通知：有历史浏览记录");
    msgBox.setInformativeText("是否继续历史浏览?");
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Ok);
    int ret = msgBox.exec();
    if(ret == QMessageBox::Ok){

```

针对日志类，假如有记录，且匹配当前用户，就进行二次回档。

3.4 地图模块

功能

显示就是显示地图所有点位，清除就是清除地图所有点位。导航选择出发校区的点位和目的校区的点位。选择不同的 mode，进行不同的最短路径选择。

此外，还有多路径导航可以选择。

3.4.1 数据结构抽象

✧ 当前窗口变量创建

```
QString query;
int speedWalk;
int speedRide;
int source;
int sourceIsShaHe;
int destBuilding;
int destIsShaHe;
int mode;

int forLessonGuide; // 是否为课程导航
int restTime;       // 时间策略, 针对课程导航
int timeWeek;       // 课程所在星期
int timeHour;       // 课程要到达的时间 (几点)

bool forMore;       // 多路径开关
```

看了变量命名也知道是干啥的了。其主要就是针对各种数据的处理信号。

引入数据结构：动态容器 Vector

因为不能直接使用 malloc 模板动态的建立一个容器存储路线数据，所以自行编写了一个模板类 vector，其作用和 malloc 类似，都是形成一个可变化容器装载数据，但是 vector 的实现和功能操作要更为便捷。Vector 有两个类，一个是存储 int 型数据的，一个是存储 double 型数据的。只取 int 类简单说明：

```
class Vector_int
{
private:
    int* obj ;
    int len ;
    int capacity ;
public:
    Vector_int(void) ;
    ~Vector_int(void) ;

    int size(void) ;
    void push( int arg ) ;
    int pop(void) ;
    void clear(void) ;
    void copy(Vector_int& vec);

    int& operator[]( int n ) const ;
};
```

其基本思路是，通过 push 函数像容器内压入数据，初始化会设定容器容量为 5，当容器满了，会在数据入栈的时候创建一个增加 5 容量的新容器，通过 copy 函数将原有数据复制到新容器中，clear 清除旧容器的内容就可实现动态容量。Double 类型 vector 和 int 基本相同只是存储内容是 double 型数据。

引入数据结构：Point, PointPtrList

Point 结构是记录点相关数据的类型，其中包括当前点的序号 number，该点的父亲 index_former，四个子点和记录到达该点的路径长度 data。

```
typedef struct point
{
    int    number;
    int    index_former;
    int    subNum_1;
    int    subNum_2;
    int    subNum_3;
    int    subNum_4;
    double data;
} Point, *PointPtr ;
```

比较巧妙的是我们将 point 划分为每十个一组，并且用 struct 封装进行链表的表示，封装出来的结构就是 PointPtrList：

```
// 每一组10个Point,并且用struct封装一组后进行链表的表示
typedef struct pointptrList {
    PointPtr thisArray; // 当前STRUCT的10个一组
    int size;           // 当前数量
    struct pointptrList* nextPtr;
} PointPtrList, *PointPtrListPtr;
```

PointPtrList 的优点在于摒弃了传统链表需要一个一个找的复杂情况，尤其是数据多的情况当需要找某个点时，只需要去对应十个为一组的组中寻找即可，大大方便了寻找。

引入数据结构：搜索地图手动栈类 MapStack

```
class MapStack
{
private:
    PointPtrListPtr obj ;
    PointPtrListPtr cur ;
    int len ;
public:
    void create(int number, int subNum_1, int subNum_2, int subNum_3, int subNum_4) ;
    void burden();
    int size(void) ;
    void push(int number, int index_former, int subNum_1, int subNum_2, int subNum_3, int subNum_4, double data) ;
    Point& operator[] (int n) const ;
};
```

MapStack 是模拟数据入栈的栈类，当查询某个点(位置)到另一个点（位置）的路线时，将会从出发点以水波纹的形式散发去寻找其他点，进而寻找目的点。其查找操作类似于树的查找，出发点是根。从根寻找的点将顺序存储在 MapStack 这个建立的栈中。通过上面的 PointPtrList 的形式实现链接。

3.4.2 地图搜索算法——贪心

✧ 首先有两个基本的路径搜索算法，都是贪心

```
double quick_findPath(int source, int dest, int mode, Vector_int& path, QString table);  
double findPath(int source, int dest, int mode, Vector_int& path, QString table);
```

分别是快速寻找算法和全面寻找算法

快速寻找算法

循环全局数据，类似于递推的全局变量。首先初始化将起点入栈，建立数组准备记录路径。开始查找循环，**只向目的地方向寻找。怎么做呢？**

引入一个函数：这个点是否比上一个点更靠近目的地，来判断当前节点是否值得入栈。

```
bool becomeCloser(int subNum, double curAbscissa, double curOrdinate, double destAbscissa, double destOrdinate);
```

● 跳过循环的方式有三种

- 一种是找到目的地，那么就更新答案，直接跳过
- 另一种方式是虽然没到达目的地，但是**已经走的路径长度以及超过了记录的到达目的地的路径的长度。**
- 最后一种方式是虽然没到达目的地，但是**过程中会记录点位距离起始地点的度量。当度量大于已经记录的值时，那么没必要入栈。**

● 继续寻找路径

查询该层 stack 的所有信息，便于生成子节点的查查询，当一圈查找完毕，开始为这一圈的子层建立新站，如果该节点直接与当前栈有路径相连，则创建，设置一个变量使不会往回找。生成四个子点的子点中更加靠近目的地的栈点。假如节点已经到达并且距离更短，则不入站；否则更新节点。

● 终止时间

在没有遍历完整个栈之前，一直寻找，直到遍历完全。这个时候运算已经完成。

✧ 运算结果

当前可能并不存在一直能靠近目的地的路径，所以这个算法是快速算法。**当返回值是-1 时，继续全面寻找算法的使用。**

全面寻找算法

循环全局数据，类似于递推的全局变量。首先初始化将起点入栈，建立数组准备记录路径。开始查找循环，不同的是，有节点可放则入栈。跳过循环的方法和快速寻找相同。针对子点集查询路径并生成数据压栈，开始为四个子层建立新站，如果该节点直接与当前栈有路径直接相连，则创建，同样的会设置变量数据使之不会往回找。继续往前找点，加入节点已经到达并且距离更短，则直接跳过，否则就更新节点。

这个做法就是一般的搜索算法，遍历整个节点网络。（不会记录回头类的路径）

- 跳过循环的方式有三种

- 一种是找到目的地，那么就更新答案，直接跳过
- 另一种方式是虽然没到达目的地，但是已经走的路径长度以及超过了记录的到达目的地的路径的长度。
- 最后一种方式是虽然没到达目的地，但是过程中会记录点位距离起始地点的度量。当度量大于已经记录的值时，那么没必要入栈。

- 继续寻找路径

查询该层 stack 的所有信息，便于生成子节点的查查询，当一圈查找完毕，开始为这一圈的子层建立新站，如果该节点直接与当前栈有路径相连，则创建，设置一个变量使不会往回找。生成四个子点的子点中更加靠近目的地的栈点。假如节点已经到达并且距离更短，则不入站；否则更新节点。

- 终止时间

在没有遍历完整个栈之前，一直寻找，直到遍历完全。这个时候运算已经完成。


3.4.3 搜索算法应用

班车设计和导航策略如下：

```
// 打车45min
// 班车信息默认沙河到本部是9, 11, 13, 15；本部到沙河是8, 10, 12, 14, 时长45min
// 公共汽车默认10分钟一班，8-18点都有，1h
// 策略：不紧急的时候，优先班车，再公交，最后打车（省钱）；紧急时时间短的优先（一定是打车竞争力大），即使是到不了会迟到也一定选择打车尽早去
```

课程便捷导航

首先选择出发地点；然后选择课程便捷导航。

The image shows a web interface for course navigation. On the left, there is a section titled '出发校区' (Departure Campus) with three radio button options: '沙河校区' (Shah Campus), '西土城校区' (Xitucheng Campus), and an unlabeled option. To the right of this is a '选择课程' (Select Course) dropdown menu. Below these elements is a large button labeled '课程便捷导航' (Course Convenience Navigation).

- ✧ 课程输入框没有输入的时候

选择时间最近的课程导航，查找课程相应的点位进行目的初始化，保留这两个暂存信息和结果答案，先找必修课，如果不是当前班级的直接跳过；再找选修课假如当前没答案，那么直接进行赋值，假如当前离此时更近，那么进行更新。假如没有课程，那么就报错；当有课程时，生成目的校区，选中相应的建筑信息，计算出相应的剩余时间，然后点击进行正常搜索。

- 课程时间比较算法

```

// 假如当前离此时更近, 那么进行更新(星期数转化为mod比较)
QString tmpName = query.value(1).toString();
int tmpIsShaHe = query.value(3).toInt();
int tmpLoc = query.value(4).toInt();
int tmpWeekDay = (query.value(5).toInt()-weekDay+7)%7; /* 转化 0-6 */
int tmpHour = 6+2*query.value(6).toInt();
if(tmpWeekDay == 0 && tmpHour < hour)
    tmpWeekDay = 7; // 因为要过一周才能到

if(tmpWeekDay < ansWeekDay || (tmpWeekDay == ansWeekDay && tmpHour < ansHour)) {
    ansName = tmpName;
    ansIsShaHe = tmpIsShaHe;
    ansLoc = tmpLoc;
    ansWeekDay = tmpWeekDay;
    ansHour = tmpHour;
}

```

时间比较最重要的就是**星期数**:

● 星期数算法

```

int tmpWeekDay = (query.value(5).toInt()-weekDay+7)%7;
if(tmpWeekDay == 0 && tmpHour < hour)
    tmpWeekDay = 7; // 因为要过一周才能到

```

将数据转换为 mod, 方便进行比较。

这里采用巧妙的办法, 将星期数转化为相应的 mod, 十分方便比较。

✧ 课程框有输入的时候

当有输入就导航选入的课程地点先找必修再找选修, 当找到时, 选中响应的建筑信息, 点击进行正常搜索。

✧ 课程导航调用一般导航, 只不过加入班车信息计算

一般导航

首先判断出发点和目的地是否选中, 如果没有直接退出。输出清零。

路线一共有四种情况: 出发点分别是沙河和本部, 目的地分别是沙河和本部。如果出发点和目的地相同, 那么基本算法为记录建筑出入口数组, 根据数组找最短路径, 然后更新答案, 逆向输出答案, 画点并输出额外信息是否能到达。当出发点和目的地不同时, 算法稍有差别, 需要先生成出发点路线, 后续框架相同, 只是增加班车的相关内容。

设计了两个算法:

- 一个是快速入栈, 可能出现找不到路径的情况但是计算速度快;
- 一个是一定能找到路径, 但是速度较慢。

使用逻辑是先用第一个, 当第一个没有找到解决路径时, 再用第二个算法。

3.5 活动模块

功能

实现活动查询（个人、集体），活动添加（个人、集体）；点击查询个人活动显示当前个人活动的时间和内容；点击查询集体活动显示当前集体活动的时间和内容。

活动可作为标签筛选，点击**时间排序和名称排序**，可对个人活动和集体活动的时间进行排序。也可通过关键字段进行查询，同样也可进行时间排序和名称排序

通过面板上方活动名称、活动类型、校区和时间的填写，点击添加活动即可实现增加活动的功能。

点击活动删除可对个人活动或集体活动删除（注意：**当集体活动不是个人创建时，不能实现删除**）

3.5.1 数据结构抽象

- 活动的结构体

```
typedef struct activity
{
    QString name;
    int isShahe;
    int location;
    int year;
    int month;
    int day;
    int startHour;
    int endHour;
    QString label;

    struct activity* nextPtr;
} Activity, *AcivityPtr;
```

- 本窗口内部变量

```
// 添加活动需要的变量
int isShaHe;
int forPeople;

// 活动需要的指针变量和指针数组
AcivityPtr single; // 个人活动链表头
int sizeSingle;
AcivityPtr* singleSelect; // 个人活动指针数组（筛选结果）
int sizeSingleSelect;

AcivityPtr team; // 集体活动链表头
int sizeTeam;
AcivityPtr* teamSelect; // 集体活动指针数组（筛选结果）
int sizeTeamSelect;

int type; // 展示模式 (1-个人, 2-集体)
```

3.5.2 建立个人活动链表

```
// 查询个人活动
single = new Activity; // 先创建头部, 不放信息
single->nextPtr = nullptr;
sizeSingle = 0;

ActivityPtr tmp = single;
query.exec("select * from "+QString::number(User_Number)+"_once");
while(query.next()) {
    tmp->nextPtr = new Activity;

    tmp->nextPtr->name      = query.value(0).toString();
    tmp->nextPtr->isShahe   = query.value(1).toInt();
    tmp->nextPtr->location  = query.value(2).toInt();
    tmp->nextPtr->year      = query.value(3).toInt();
    tmp->nextPtr->month     = query.value(4).toInt();
    tmp->nextPtr->day       = query.value(5).toInt();
    tmp->nextPtr->startHour = query.value(6).toInt();
    tmp->nextPtr->endHour   = query.value(7).toInt();
    tmp->nextPtr->label     = query.value(8).toString();

    tmp->nextPtr->nextPtr = nullptr;
    tmp = tmp->nextPtr;
    sizeSingle++;
}

// 生成指针数组
singleSelect = new ActivityPtr[sizeSingle];
sizeSingleSelect = sizeSingle;
tmp = single->nextPtr;
for(int i=0; i < sizeSingleSelect; i++) {
    singleSelect[i] = tmp;
    tmp = tmp->nextPtr;
}
```

这里的意思就是以链表作为存储结构。但是这里用另一个指针数组将地址赋值, 使得后续在排序的时候十分方便。

3.5.3 建立集体活动链表

```
// 查询集体活动
team = new Activity; // 先创建头部, 不放信息
team->nextPtr = nullptr;
sizeTeam = 0;

ActivityPtr tmp = team;
query.exec("select * from activity");
while(query.next()) {
    tmp->nextPtr = new Activity;

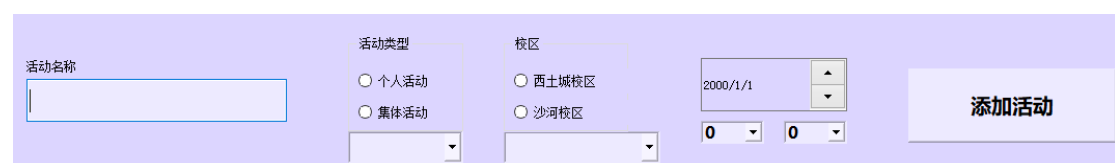
    tmp->nextPtr->name      = query.value(0).toString();
    tmp->nextPtr->isShahe   = query.value(1).toInt();
    tmp->nextPtr->location  = query.value(2).toInt();
    tmp->nextPtr->year      = query.value(3).toInt();
    tmp->nextPtr->month     = query.value(4).toInt();
    tmp->nextPtr->day       = query.value(5).toInt();
    tmp->nextPtr->startHour = query.value(6).toInt();
    tmp->nextPtr->endHour   = query.value(7).toInt();
    tmp->nextPtr->label     = query.value(8).toString();

    tmp->nextPtr->nextPtr = nullptr;
    tmp = tmp->nextPtr;
    sizeTeam++;
}

// 生成指针数组
teamSelect = new ActivityPtr[sizeTeam];
sizeTeamSelect = sizeTeam;
tmp = team->nextPtr;
for(int i=0; i < sizeTeamSelect; i++) {
    teamSelect[i] = tmp;
    tmp = tmp->nextPtr;
}
```

这里的意思就是以链表作为存储结构。但是这里用另一个指针数组将地址赋值, 使得后续在排序的时候十分方便。

3.5.4 添加活动



首先选择地点编号添加选择标签 (集体活动和个人活动的标签), 需要保证添加的活动是合法的 (开始时间在当前时间之前等), 将添加的活动写入数据库中

- 1) 个人活动添加 (班会、小组作业、创新创业活动、聚餐)
- 2) 集体活动添加 (自习、锻炼、外出)

3.5.5 查询活动

- 个人活动查询

首先进行初始数据处理, 进行记录的删除, 活动结构中通过链表结构存储的, 删除对应

的指针数组。创建头部，进行复制生成指针数组顺便直接展示到查询结果板。个人活动的查询是在个人活动表中。

● 集体活动查询

集体活动的查询和个人活动的查询步骤基本相同，只不过集体活动的查询表是在 activity 表中。

● 合理性检测（冲突避免）

```
// *****核心算法*****
int flag = 0;
// 添加检测机制-1: 判断是否和课程有冲突 (直接看所在的课程里是不是有冲突)
int weekDay = CaculateWeekDay(year, month, day);
int left, right; // 下界和上界 (1, 2, 3, 4, 边界为0, 5), 针对课程节数使用, 判断是否落在活动中。如果落在(left ≤ index ≤ right), 则冲突
```

1. 对本人课程进行避免:

通过这个策略将活动将活动时间转化为活动窗口。因为一天四节课，所以有 0-5 和数值设置：

left=0: 在 8 点前开始; =1 在 10 点前开始..=5, 在 16 点后开始;

right=0: 在 8 点前结束; ..=5, 在 16 点后结束;

直接通过判断当日的是否落在活动窗口内：如果落在，则冲突；否则，不冲突。

```
if(start_Hour < 8) left = 0;
else if(start_Hour < 10) left = 1;
else if(start_Hour < 12) left = 2;
else if(start_Hour < 14) left = 3;
else if(start_Hour < 16) left = 4;
else left = 5;
if(end_Hour > 16) right = 5;
else if(end_Hour > 14) right = 4;
else if(end_Hour > 12) right = 3;
else if(end_Hour > 10) right = 2;
else if(end_Hour > 8) right = 1;
else right = 0;
```

```
while(!flag && query.next()) {
    if(query.value(8).toString() == userClass && query.value(5).toInt() == weekDay
        && left ≤ query.value(6).toInt() && query.value(6).toInt() ≤ right )
        flag = 1;
}
```

当然，这里有个星期转换的函数：`int iWeek=(d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;`

2. 对个人，集体活动进行避免:

```
if(start1 ≥ end2 || end1 ≤ start2)
    return false;
return true;
```

这个就是简单的时间判断了：

3.5.6 活动筛选

活动存储后链表和指针数组都存在，筛选是用指针数组。**展示个人活动，遍历向前覆盖，如果有标签选择，那么进行第二次筛选，也是向前覆盖的操作。**个人和集体的活动筛选类似，只是类型 Type 不同和向前覆盖的规模不同。

```
if(ui→comboBox_labelSelect→currentIndex() ≠ -1) {
    QString input = ui→comboBox_labelSelect→currentText();

    int tmpSize = sizeSingleSelect;
    sizeSingleSelect = 0;
    for(int i=0; i < tmpSize; i++) // 往前覆盖
        if(input == singleSelect[i]→label)
            singleSelect[sizeSingleSelect++] = singleSelect[i];
}
```

3.5.7 活动排序

```
void Widget_Activity::quickSort_Name(ActivityPtr* array, int l, int r)
{
    if(l >= r) return;
    int i = l; // 从左边界起始
    int j = r; // 从右边界起始
    ActivityPtr key = array[i]; // 最左边作为key
    while (i<j)
    {
        // 右边向左滑
        while( i < j && array[j]→name >= key→name )
        {
            array[i] = array[j];
            // 左边向右滑
            while( i < j && array[i]→name <= key→name )
            {
                array[j] = array[i];
            }
            array[i] = key; // i == j
            quickSort_Name(array, l, i-1);
            quickSort_Name(array, i+1, r);
        }
    }
}
```

有时间排序和名称排序两种排序，通过快速排序进行排序。

时间排序是把年月日时取出来，名称排序是按照名称长度排序。快速排序通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

快速排序算法通过多次比较和交换来实现排序，其排序流程如下：

- 1、首先设定一个分界值，通过该分界值将数组分成左右两部分。
- 2、将大于或等于分界值的数据集中到数组右边，小于分界值的数据集中到数组的左边。此时，左边部分中各元素都小于或等于分界值，而右边部分中各元素都大于或等于分界值。
- 3、然后，左边和右边的数据可以独立排序。对于左侧的数组数据，又可以取一个分界值，将该部分数据分成左右两部分，同样在左边放置较小值，右边放置较大值。右侧的数组数据也可以做类似处理。
- 4、重复上述过程，可以看出，这是一个递归定义。通过递归将左侧部分排好序后，再递归排好右侧部分的顺序。当左、右两个部分各数据排序完成后，整个的排序也就完成了。

3.6 课程模块

3.6.1 数据结构抽象

- 课程结构体

```

typedef struct lesson {
    int isSelect;
    int number;
    QString name;
    int teacher;
    int isShaHe;
    int location;
    int dayOfWeek;
    int index;
    QString qun;
    QString extra;
    struct lesson* nextptr;
} Lesson, *LessonPtr;

```

● 窗口内部变量

```

// 课程表功能 *****
QPushButton** format; // 课程表
int* formatIndex; // 课程表链表对应的format下标 (空间大小为size, 数值范围为0-27)
int selectIndex; // 被选中的format下标 (0-27), 通过鼠标选择

// 课程表链表
LessonPtr lessons;
int size; // 课程规模
LessonPtr* lessonSelect; // 筛选后的课程链表: 直接指向相应的空间
int sizeSelect; // 筛选过后的size

QDir filePath; // 当前课程表对应的文件夹

void setInformation(); // 初始化所有信息
void quickSort_Name(LessonPtr* array, int l, int r);
void quickSort_Time(LessonPtr* array, int l, int r);

// 选修课功能 *****
LessonPtr lessonXuanxiu;
int sizeXuanxiu; // 选修课程规模
LessonPtr* lessonXuanxiuSelect; // 筛选后的选修课程链表: 直接指向相应的空间
int sizeXuanxiuSelect; // 筛选过后的size

```

3.6.2 获取课程信息

根据 lesson 结构体最后的 lesson 指针可知, 课程情况是链表存储的。首先新建一个 lesson 头部, 不存储信息。(数值赋值, 没必要附图片)

对于学生课程表来说, 首先找出学生班级, 先找必修课程, 再找选过的选修课程; 对于老师来说, 也是先寻找必修课程, 再找教授的选修课程。接着排开课程表, 遍历课程信息链表, 放入 28 个课程框按钮中。生成选修课链表, 首先也是建立一个 lesson 头部, 但不储存信息, 初始化, 建立指针空间, 点击一下显示选修课程按钮进行显示。

```

// 画出课程表
format = new QPushButton*[28];
// 先画出按钮
for(int i=0; i < 7; i++) {
    for(int j=0; j < 4; j++) {
        format[4*i+j] = new QPushButton(this);
        format[4*i+j]→setGeometry(50+150*i, 50+100*j, 150, 100);
        format[4*i+j]→setFont(QFont("Microsoft YaHei",10,75));
        format[4*i+j]→setStyleSheet("background: rgba(255,255,255,50%)");
        format[4*i+j]→show();
        connect(format[4*i+j],&QPushButton::clicked,[=]() {
            int x=this→mapFromGlobal(QCursor().pos()).x();
            int y=this→mapFromGlobal(QCursor().pos()).y();
            selectIndex = 4*((x-50)/150) + (y-50)/100;
            qDebug() << "select" << selectIndex;
            showInformation();
        });
    }
}
}

```

● 点击后显示所有信息

这里通过鼠标点击的坐标判断点击的是哪一个课程（0-27）。点击课程表中课程在链表中循环找，找到就显示地点、显示群、显示考试信息设置信息沙河还是西土城以及时间，然后跳出；没找到那就遍历完了都没有显示，并将一些信息归零。

3.6.3 课程信息

首先清空并获取信息，建立筛选链表，假如输入为空，则直接通过获得所有课程的链表数据显示全部课程结果；如果有输入，那么开始筛选所有课程，根据输入的词语筛选，建立链表，获得所有课程的链表数据。先进行名字字段筛选，选择的是 kmp 算法，然后按序号进行筛选，同样是 kmp 算法。筛选完毕进行展示。

3.6.4 添加选修课程——学生功能

如果没进行选择，那么返回，否则获取对应选项，查找该课程信息，检查是否冲突，首先找出学生班级先找必修课，再找选修课，如果重复了显示 Error! 和必修课有冲突!或者 Error! 和选修课有冲突!没问题后进行添加到数据库中，回归初始状态。

● 冲突检测

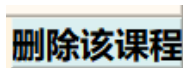
```

// 先找必修课
query.exec("select * from lesson_compulsory");
while(query.next()) {
    // 重复了
    if(query.value(9).toString() == userClass && query.value(5).toInt() == obj.dayOfWeek && query.value(6).toInt() == obj.index) {
        flag = 1;
        break;
    }
}
if(flag == 1) {
    // 再找选修课
    query.exec("select * from "+QString::number(User_Number)+"_select");
    while(query.next()) {
        // 重复了
        if(query.value(5).toInt() == obj.dayOfWeek && query.value(6).toInt() == obj.index) {
            flag = 1;
            break;
        }
    }
}
}

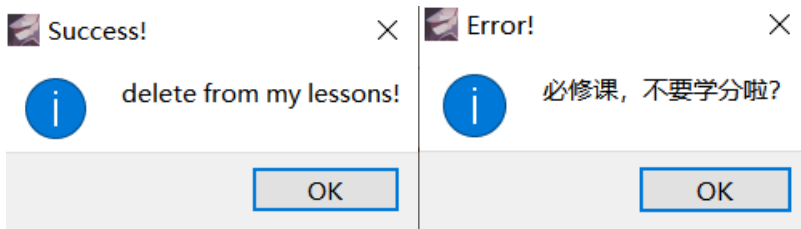
```

主要寻找有没有课程重复。这里只需要判断节数相等即可。

3.6.5 删除课程



点击右下角的删除课程，就可以把相应的课程删除：



此外，在删除和添加课程后，我们会及时对链表进行修改并肢解成现在页面上，不用刷新！

3.6.6 作业&资料

选中一个课程的情况下，系统会自动设定文件传输路径（相应的文件夹下）。并将压缩窗口的路径修改成需要的路径。

```

// 赋值filePath
if(tmp->isSelected == 0) {
    filePath.setPath(HOME.absolutePath()+"/files/c/"+QString::number(tmp->number));
    if(!filePath.exists()) {
        qDebug() << "不存在该路径";
        filePath.mkpath(HOME.absolutePath()+"/files/c/"+QString::number(tmp->number));
        filePath.setPath(HOME.absolutePath()+"/files/c/"+QString::number(tmp->number));
    }
} else {
    filePath.setPath(HOME.absolutePath()+"/files/s/"+QString::number(tmp->number));
    if(!filePath.exists()) {
        qDebug() << "不存在该路径";
        filePath.mkpath(HOME.absolutePath()+"/files/s/"+QString::number(tmp->number));
        filePath.setPath(HOME.absolutePath()+"/files/s/"+QString::number(tmp->number));
    }
}
qDebug() << filePath.absolutePath();
zip->setFilepath(filePath.absolutePath());

```

未截止作业

已截止作业

资料功能

提交作业

未截止作业：当前时间在截止时间之前；

已截止作业：当前时间在截止时间之后。

● 学生

1. 在学生个人表中根据作业号以及 isSelect 把符合的选择出来，生成单个作业信息。

2. 根据选中作业选项，已交作业显示已提交；未交作业显示未提交，然后分类。

未截止作业

1 课程设计验收 2022-6-16 9 (已提交)

5 验收实验报告 2022-6-20 9 (未提交)

3. 此外，在提交作业的时候，系统会自动命名文件名，使得管理方便。

DS Project_Release > DS-Localhost > files > s > 1

在 1 中搜索

名称	修改日期	类型	大小
2_验收实验报告.Nzip	2022/6/10 19:39	NZIP 文件	1,894 KB

其中，DS-Location 之前的都是 HOME 路径。

4. 当然，没选中未提交作业时，点击作业，显示 Error! No task selected!；输入不为空先获取编号，接着直接更新个人作业状态，再更新作业总数状态。

5. 在上传作业的时候，会自动查重。（全字符匹配和 MD5 查重）如果重复则报错，不予更改，老师可以直接在文件夹下查重，看哪个学生抄袭了。

● 老师

老师添加作业信息功能和学生基本类似，只是表不同。当同学交上作业后，在老师的课程页面作业中可发现，已提交作业数目+1。

未截止作业

1 课程设计验收 2022-6-16 9 1/1

5 验收实验报告 2022-6-20 9 1/1

（作业有唯一标识，不会重复）

● 资料上传

直接套用 zip 压缩窗口，只不过不用额外功能，只是简单的上传和覆盖。

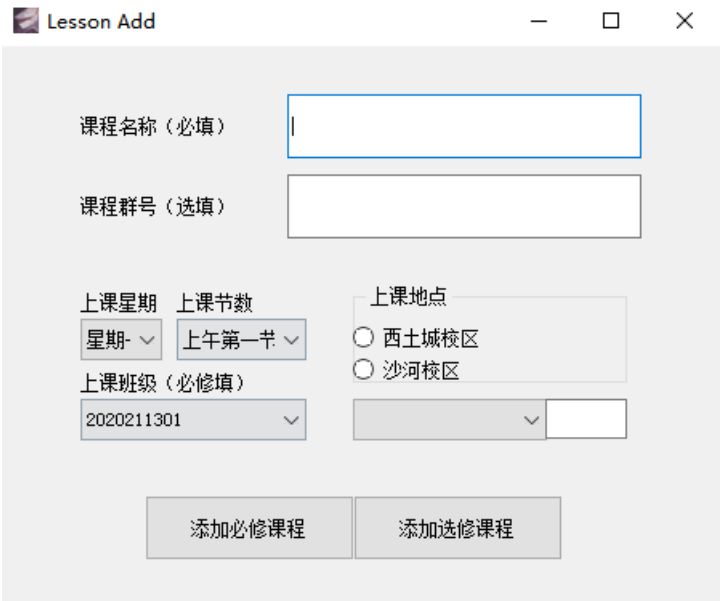
（不会系统命名，但在固定文件夹内传输）

3.6.7 课程模块——管理员特殊功能



管理员进行课程管理总共有四个功能：添加课程、修改课程、发布作业、发布考试，分别点击右上角四个按钮可进入功能界面实现功能。可见界面中有课程表是 7*4 的规格，我们规定上午有两节大课分别是 8：00-10：00 和 10：00-12：00 下午有两节大课分别是 1：00-3：00 和 3：00-5：00，通过点击课程表上的课程可在课程表下方显示课程信息，包括课程当前进度即进行了几周，课程地点，课程群号。还可以显示未截至作业和已截至作业的情况，右边可显示当前课程考试信息。在界面右下角是及逆行课程信息检索的位置，输入课程名，可显示课程信息，还可对教授课程进行名称排序或时间排序。

添加课程



(1) 功能

对课程名称、课程群号填写，对上课星期（星期一、星期二、星期三、星期四、星期五、星期六、星期日），上课节数（上午第一大节，上午第二大节、下午第一大节、下午第二大节），上课班级（2020211301、2020211302、2020211303、2020211304、2020211305）的选择，地点的选择和地点额外信息进行录入，再点击添加必修课程或添加选修课程实现添加课程功能。

(2) 算法实现

首先先将课程名称和群号记录下来，接着检测是否冲突：学生课程冲突或者是老师课程冲突，如果发现冲突，显示 Error!和已有课有冲突!如果没有冲突，进行添加课程功能，将输入的数据写入数据库中，如果添加课程成功，显示 Success! Added!，添加选修课和必修课算法几乎相同，只是写入的数据库表不同，选修课写入选修表，必修课写入必修表。

● 合法性检测：（冲突检测，以必修课为例）

```
// 检测是否冲突
int flag = 0;
query.exec("select * from lesson_compulsory");
while(query.next()) {
    // 要么学生冲突，要么我自己冲突
    if((query.value(2).toInt() == User_Number || query.value(9).toString() == c)
        && query.value(5).toInt() == week && query.value(6).toInt() == index ) {
        flag = 1;
        break;
    }
}
query.exec("select * from lesson_select");
while(query.next()) {
    if(query.value(2).toInt() == User_Number && query.value(5).toInt() == week && query.value(6).toInt() == index ) {
        flag = 1;
        break;
    }
}
if(flag == 1) {
    QMessageBox mess(QMessageBox::Information, "Error!", "和已有课有冲突!");
    mess.setWindowIcon(QIcon(":/images/02.jpg"));
    mess.exec();
    return;
}

// 添加必修
query.exec("select * from lesson_compulsory");
int size = query.size();
QString cmd = "insert into lesson_compulsory(number, name, teacher, isShaHe, location, dayOfWeek, c_index, qun, extra, class) values(";
cmd += QString::number(size+1)+"','"+name+"',"+ QString::number(User_Number)+"','"+QString::number(isShaHe)+"',"
+QString::number(location)+"','"+QString::number(week)+"','"+QString::number(index)+"','"+qun+"','"+extra+"','"+c+"'";
qDebug() << cmd;
query.exec(cmd);
```

必修课的冲突检测策略是：

- 不和班级内的其他课程冲突；
- 不和自己已经带的课冲突。

选修课的冲突检测策略是：

- 不和自己已经带的课冲突。

修改课程

Lesson Change

可修改的课程

可更新的字段 (可多选)

课程名称

课程群号

星期数

地点

☐ 西土城校区

☐ 沙河校区

节数

修改

可修改的课程

数据结构7-3 必修

数据结构补充课程7-2 必修

数据结构习题讲解6-3 必修

数据结构课程设计3-2 选修

数据结构课程设计1-2 选修

数据结构讲解课程5-4 选修

- (1) 功能
- 对可修改的课程进行选择后，在课程名称，课程群号，星期数，节数或者地点进行填写或选择，点击修改按钮实现修改功能。
- (2) 算法实现
- 首先选中信息，判断是否选中，如果没有选中，即非法输入那么直接退出并显示 Warning Event: nothing selected!接着选择课程信息，修改名称，修改群号，修改位置，修改星期数，修改节数并写入数据库中。最后初始化变量，均置为空。

- 合法性检测
- 没啥检测，就是看输入是不是空

发布作业

Lesson task

可选课程

作业名称

截止日期

2000/1/1

添加

1 课程设计验收 2022-6-16 9 1/1

5 验收实验报告 2022-6-20 9 6/1

- (1) 功能
- 对老师管理员教授的课程进行选择，输入作业名称，截至日期，点击添加按钮，完成课

程作业添加功能。当添加完成作业后，老师在课程表上点击该课程后，在下方未截至作业中可看到作业情况已经上交份数状况。

(2) 算法实现

首先先将各种信息写入或选择，判断时间是否合适，如果时间非法，即作业时间在课程开始时间之前等情况，那么结束退出并显示 Warning! Time is not invalid!，如果没有输入，那么结束退出并显示 Warning Event: nothing selected!。接着选择必修课程信息，将信息添加到学生作业中去，找到课程班级，通过数组找到所有学生，添加到学生的 task 里面。选修课程信息与必修课程信息类似，首先找到所有学生，找到他们的选修课表，将作业添加到课程中去。作业添加结束后，更新作业总人数，返回 message，显示 Success Event: task added! 接着将页面清空结束。

关键是得把作业信息添加给学生！

这里有两种：必修课根据班级来，可以固定人数；选修课要找谁选了课，所以要对所有选课学生进行寻找。

● 添加作业到学生

这是必修课：

```
int count = 0;
// 添加到学生中去 (必修课)
if(isSelect[lessonIndex] == 0) {
    // 找到课程班级
    QString c;
    query.exec("select * from lesson_compulsory limit "+QString::number(number[lessonIndex]-1)+"",1");
    while(query.next()) c = query.value(9).toString();
    qDebug() << c;

    // 找到所有学生
    Vector<int> student; // 自己的数组
    query.exec("select * from people");
    while(query.next()) {
        if(query.value(5).toString() == c) {
            student.push(query.value(0).toInt());
            count++;
        }
    }

    // 添加到学生的task里
    for(int i=0; i < student.size(); i++) {
        QString cmd = "insert into "+QString::number(student[i])+"_task(number, lesson_number, isSelect, name, year, month, day, hour, isCommit) values(";
        cmd += QString::number(size+1)+"",
            +QString::number(number[lessonIndex])+",",
            +QString::number(isSelect[lessonIndex])+",",
            +""+name+"",
            +QString::number(year)+"",+QString::number(month)+"",+QString::number(day)+"",+QString::number(hour)+"",0);
        qDebug() << cmd;
        query.exec(cmd);
    }
}
```

这是选修课：

```
// 添加到学生中去 (选修课)
else if(isSelect[lessonIndex] == 1) {
    // 找到所有学生
    Vector<int> student;
    query.exec("select * from people");
    while(query.next())
        if(query.value(3).toInt() == 0) student.push(query.value(0).toInt());

    // 找选修课表
    for(int i=0; i < student.size(); i++) {
        query.exec("select * from "+QString::number(student[i])+"_select");
        int flag = 0;
        while(!flag && query.next())
            if(query.value(0).toInt() == number[lessonIndex]) flag = 1;

        // 添加课程表
        if(flag == 1) {
            count++;
            QString cmd = "insert into "+QString::number(student[i])+"_task(number, lesson_number, isSelect, name, year, month, day, hour, isCommit) values(";
            cmd += QString::number(size+1)+"",
                +QString::number(number[lessonIndex])+",",
                +QString::number(isSelect[lessonIndex])+",",
                +""+name+"",
                +QString::number(year)+"",+QString::number(month)+"",+QString::number(day)+"",+QString::number(hour)+"",0);
            qDebug() << cmd;
            query.exec(cmd);
        }
    }
}
```

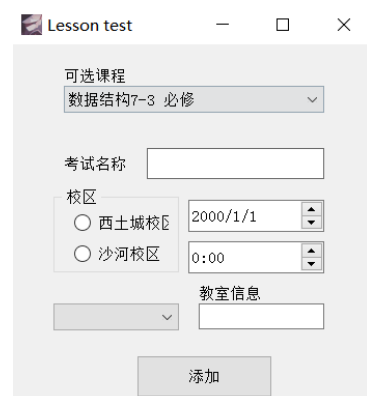
添加完毕后要更新作业总人数：

```
// 更新作业总人数
query.exec("update lesson_task set all_number = "+QString::number(count)+" where number = "+QString::number(size+1));
```

- 冲突检测

没啥检测，就是看输入是不是空，还有时间是否合法：

发布考试



(1) 功能

课程考试功能界面和课程作业功能界面基本相同，对老师管理员教授的课程进行选择，输入考试名称，开始时间，截至时间和教室点击添加按钮，完成课程作业添加功能。

(2) 算法实现

首先先将考试相关信息录入界面，接着判断时间是否合适，如出现考试时间冲突或者考试时间在课程时间之前等的情况，那么说明时间不合法，退出结束并显示 Warning! Time is not invalid!再判断其他输入情况，如果出现没选中的情况，那么退出并显示 Warning! Event: something not selected or empty!当所有的一切都合法，找到课程班级，通过数组找到所有学生，添加到学生的 test 里面,将考试写入数据库考试表中。考试添加结束后，更新作业总人数，返回 message，显示 Success Event: test added! 接着将页面清空结束。

- 冲突检测

没啥检测，就是看输入是不是空，还有时间是否合法：

3.7 日志模块

```
class Log
{
public:
    Log();
    void create(); // 创建初始化 (需要先初始化HOME)
    void writeLog(QString str);

    // 每次进页面保存状态
    void maintainState(QString page);
    // 正常退出时移除状态记录
    void removeState();

    // 信息跳转
    int getUserNumber() {return number;}
    void deletePage() {page = "";}
    QString getPage() {return page;}

private:
    // 从保留状态文件里取出来的东西
    int number;
    QString page; // 看打开来的是什么界面
};
```

其基本思路就是提供一个全局的记录存储器和写入日志的接口。

● 写日志

```
// 写日志
void Log::writeLog(QString str)
{
    QDir tempDir(HOME.absolutePath()+"/Logs");
    if(!tempDir.exists()) {
        qDebug() << "不存在该路径";
        tempDir.mkpath(HOME.absolutePath()+"/Logs");
    }
    QFile tempFile(tempDir.absolutePath()+"/user.log");
    if(tempFile.open(QIODevice::ReadWrite | QIODevice::Append)) {
        QTextStream txtOutput(&tempFile);

        // 先生成时间
        QString curTime = QString::number(system_Time.get_BigTime().get()[0])
            + "-" + QString::number(system_Time.get_BigTime().get()[1])
            + "-" + QString::number(system_Time.get_BigTime().get()[2])
            + ":" + QString::number(system_Time.get_SmallTime().get()[0])
            + ":" + QString::number(system_Time.get_SmallTime().get()[1])
            + "." + QString::number(system_Time.get_SmallTime().get()[2]);

        txtOutput.setCodec(QTextCodec::codecForName("utf-8"));
        txtOutput << curTime << "    User: " << QString::number(User_Number) << "    content: " << str << "\n";
    }
    // 没有文件, 说明正常退出, 无需操作
    else {
        qDebug() << "log: File ERROR!";
    }
    tempFile.close();
}
```

● 创建历史登录和浏览信息, 供回档模块使用

```
// 创建属性, 并检测曾经状态
void Log::create()
{
    QDir tempDir(HOME.absolutePath()+"/Logs");
    if(!tempDir.exists()) {
        qDebug() << "不存在该路径";
        tempDir.mkpath(HOME.absolutePath()+"/Logs");
    }

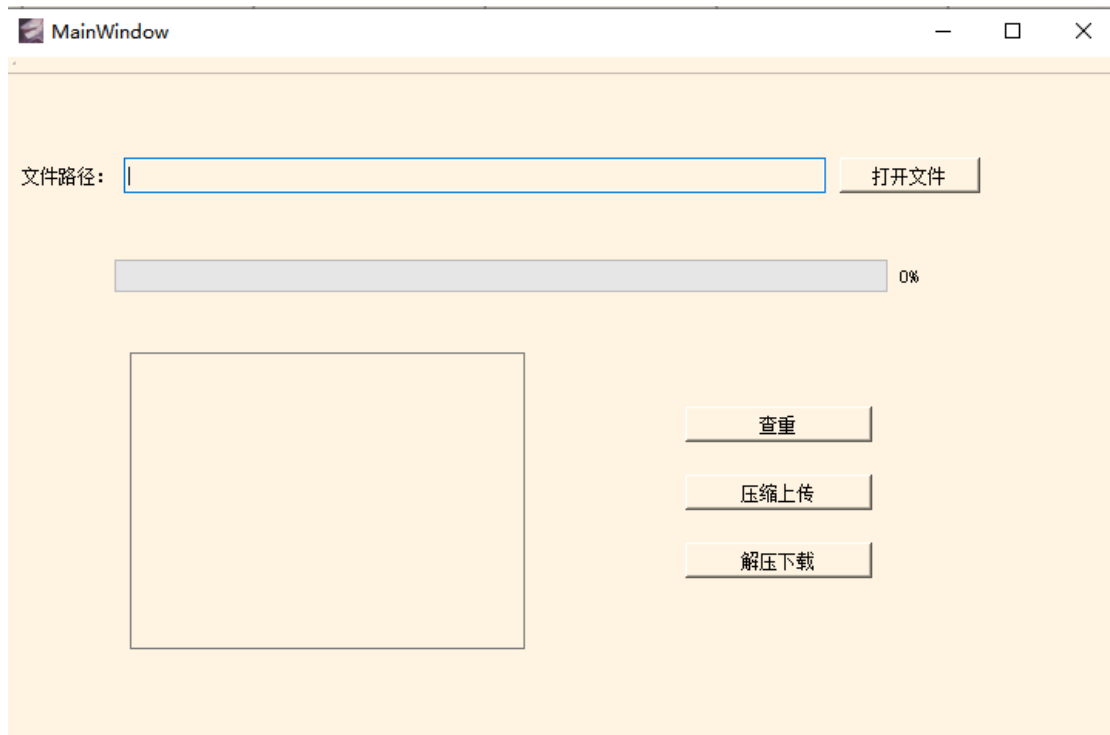
    QFile tempFile(tempDir.absolutePath()+"/state.log");
    // 能够成功打开
    if(tempFile.open(QIODevice::ReadOnly)) {
        // 录入数据
        QByteArray array = tempFile.readAll();
        QByteArrayList list = array.split(' ');

        number = list.at(0).toInt();
        // 没有后续数据了
        if(list.size() == 1)
            page = "";
        else
            page = list.at(1);
    }
    // 没有文件, 说明正常退出
    else {
        qDebug() << "state File ERROR!";
        number = 0;
        page = "";
    }
    tempFile.close();
}
```

3.8 课程作业压缩解压储存

【一】使用说明

(1) 界面展示



(2) 功能:

- {1}查重 (点击按钮后跳出文件管理器, 选择文件夹, 对文件夹内文件进行查重)
- {2}压缩上传 (先打开文件, 选择要压缩上传的文件, 点击压缩上传, 进度条显示到 100% 时, 返回信息框提示压缩上传成功自动在上传到的指定路径进行查重, 返回重复文件信息框)
- {3}解压下载 (打开文件选择解压下载的文件, 点击解压下载, 自动弹出文件资源管理器, 选择路径进行解压下载, 进度条显示到 100% 时, 返回提示框解压下载成功)

【二】细节分析

- (1) **压缩按钮**: click()函数-->从 textline 中获得地址-->改变 compression 类中 coding 和 decoding 的枚举类 condition 的值-->compression 类继承 QThread 线程类调用 start()函数-->进程 start()函数调用 compression 类的 run()函数-->通过判断枚举类的值来选择压缩部分或者解压部分-->调用 creatinfo()函数建立哈夫曼表和哈夫曼树-->调用 coding()函数二进制写入哈夫曼表结束压缩-->对函数传入的上传路径的文件夹中函数调用查重函数进行查重[见 3]
- (2) **解压按钮**: click()函数-->打开资源管理器选择地址-->改变 compression 类中 coding 和 decoding 的枚举类 condition 的值-->compression 类继承 QThread 线程类调用 start()函数-->进程 start()函数调用 compression 类的 run()函数-->通过判断枚举类的值来选择压

缩部分或者解压部分-->调用 decoding()函数读取配置函数建树搜索替换重新写入

(3) **查重按钮**: -->使用两种查询匹配方式-->文件属性 MD5 值匹配-->获得文件路径-->读取文件内容-->存入 QCryptographicHash{生成二进制或文本数据的加密 (不可逆的) 哈希数据}-->遍历元素 (使用 iterator 迭代器检查并遍历 qhash 关联容器内的元素) -->返回重复值-->文件内容全字符匹配 (按照要求的算法) -->获得文件路径-->读取文件内容-->遍历元素-->返回重复值

● 功能特点

[1]可以传入上传路径, 即文件压缩查重存储路径

[2]可重规格化命名上传文件

[3]查重功能可独立使用

[4]通过哈夫曼树顺序构造法(不断从优先队列中取出最小权值的两个结点, 构造一棵新的树, 该树根结点权值为其左右儿子结点权值之和。而其根结点字符数据等于左儿子字符数据, 再把树根节点放入优先序列)构造哈夫曼树

【三】代码截图分析

(1) 压缩函数

```
void MainWindow::on_codeButton_clicked()
{
    ui->progressBar->setValue(0);
    z=0;
    process->MyPath=path;
    process->condition=coding;
    process->Filepath=Filepath;
    if(Filename.length()!=0){
        process->Filename=Filename;
    }
    else{
        process->Filename="";
    }
    process->start();
    qDebug()<<"开始压缩";
    QMessageBox::information(this,"warn","压缩成功 正在进行查重");
    on_checkButton_clicked(Filepath);
    ui->progressBar->setValue(0);
}
```


- 进程 start()函数调用 compression 类的 run()函数-->通过判断枚举类的值来选择压缩部分或者解压部分

```
void Compression::run()
{
    isoK=true;
    if(condition==coding)
    {
        if(this->CreatInfo(MYpath))
        {
            this->Coding(MYpath,Filepath);
        }
    }
    else if(condition==decoding)
        this->Decoding(MYpath,Filepath);
}
```

- 读取文件建立结点数组，通过哈夫曼树的构造函数建立字符结点的哈夫曼树

```
bool Compression::CreatInfo(QString path)
{
    this->clear(); //清空内容
    for (int i = 0; i<256; i++)
    {
        My_Infos[i].ch = (unsigned char)i;
    }
    quint8 ch=0;

    QFile openfile(path); //读取二进制文件
    if(!openfile.open(QIODevice::ReadOnly)) {...}
    //读取路径下对应文件
    QByteArray a;
    while(!openfile.atEnd()) {...}
    //遍历统计字符类型和个数
    openfile.close();
    bInfo = new Info[size];
    for (int i = 0, j = 0; i<256; i++) {...}
    //建立结点数组
    percentage=sum/100;
    my_Tree = new HuffmanTree<Info>(bInfo, size); //创建树
    this->setCode(my_Tree->getRoot()); //完善数组中的信息
    return true;
}
```

- Info 节点的声明

```
struct Info
{
    unsigned char ch; //对应字符
    quint64 count; //出现的次数 23
    QString str; //代替的二进制编码 (5 位)
    quint16 len; //编码长度为5

    Info(long count = 0) //构造函数
    {...}

    ~Info() //析构函数
    {...}

    bool operator<=(Info R) //运算符重载 {...}

    bool operator >(Info R) {...}

    Info operator+(Info &R) {...}

    bool operator<=(Info *R) //运算符重载 {...}

    bool operator >(Info *R) {...}

    Info operator+(Info *R) {...}
};
```

- 模板类哈夫曼树的定义

```
template<class T>
class HuffmanTree
{
public:
    HuffmanTree(T w[], int n); // 构造函数根据数组构造树
    ~HuffmanTree(); // 析构函数
    HuffmanNode<T> *getRoot() // 获取根节点 {...}
protected:
    HuffmanNode<T> *root; // 根节点
    // void deleteTree(HuffmanNode<T> *t); // 删除子树
    void mergeTree(HuffmanNode<T> &ht1, HuffmanNode<T> &ht2, HuffmanNode<T> *&parent); // 融合函数
};
```

- 使用哈夫曼树类的构造函数 存储在堆中

```
template<class T>
HuffmanTree<T>::HuffmanTree(T w[], int n) // 通过数组构造哈夫曼树
{
    Heap<HuffmanNode<T>*> hp(256); // 使用最小堆存放森林

    for (int i = 0; i < n; i++) {...}
    // 创建所有树的结点

    HuffmanNode<T> *parent = NULL;

    while (hp.size() > 1) {...}
    // 将堆中森林按照层次遍历存放
    hp.RemoveMin(root);
}
```

- 将字符集存入排序好的层次遍历结构存储在堆中的树

```
void Compression::setCode(HuffmanNode<Info> *node)
{
    QStack<HuffmanNode<Info>*> s;
    QStack<QString> s1;
    s.push(node);
    QString str;
    s1.push(str);
    HuffmanNode<Info> *p = NULL;
    while (!s.empty()) {...}
}
```

- 进入把文本和哈夫曼表二进制写入文件

```
void Compression::Coding(QString path, QString Filepath)
{
    QFile openfile(path); // 读文件
    if (!openfile.open(QIODevice::ReadOnly)) // QIODevice类是所有输入输出IO类的基础类 {...}
    quint8 ch = 0;
    QString str;
    if (Filename.length() != 0) {...}
    else {...}
    // 文件命名 (外界传入)
    path = Filepath + "/" + str + QString(".Nzip"); // 写文件
    QFile savefile(path);
    if (!savefile.open(QIODevice::WriteOnly)) {...}
    QDataStream fout(&savefile); // QDataStream是数据流, 相当于数据管道, 屏蔽了数据转换过程, 读取数据

    // 将配置文件写进去
    fout<<sum; // 个数
    fout<<size; // 字符种类
    for (int i = 0; i < size; i++) {...}

    // 进行压缩
    quint64 temp = 0; // 存储的单位
    int pos = 0;

    QByteArray a;
    while (!openfile.atEnd()) {...}

    if (pos) // 最后的编码不满足8个字节, 填充0 {...}

    openfile.close();
    savefile.close();
}
```

- 压缩上传完成

(2) 解压函数

```
void MainWindow::on_decodeButton_2_clicked()
{
    QMessageBox::information(this, "warn", "选择下载到的地址");
    QString filepath = QFileDialog::getExistingDirectory(this);
    ui->progressBar->setValue(0);
    z=0;
    process->condition=decoding;
    process->MyPath=path;
    process->Filepath=filepath;
    process->start();
    qDebug()<<"开始解压";
    QMessageBox::information(this, "warn", "解压成功");
    ui->progressBar->setValue(0);
}
```

- 进入进程类的 strat () 函数, 调用 run ()

```
void Compression::run()
{
    isoK=true;
    if(condition==coding)
    {
        if(this->CreatInfo(MYpath))
        {
            this->Coding(MYpath,Filepath);
        }
    }
    else if(condition==decoding)
        this->Decoding(MYpath,Filepath);
}
```

- 进入解压函数 Decoding (), 读取配置文件即哈夫曼编码, 根据结点数创建树, 遍历替换为字符, 写入函数

```
void Compression::Decoding(QString path,QString Filepath)
{
    Filepath="C:/Users/todn/Desktop";
    this->clear();
    QFile openfile(path); //读文件
    if(!openfile.open(QIODevice::ReadOnly)) {...}
    QDataStream fin(&openfile);

    // 读配置文件
    fin>>sum;
    fin>>size;
    percentage=sum/100;
    bInfo = new Info[size];

    for(int i=0;i<size;i++) {...}

    my_Tree = new HuffmanTree<Info>(bInfo, size); //创建树
    HuffmanNode<Info> *p = my_Tree->getRoot();
    QString str;
    path=path.remove(QString(".Nzip"));
    str=path.section('/',-1);
    path=Filepath+"/"+str;
    //path="myfile.txt";
    QFile savefile(path);
    if(!savefile.open(QIODevice::WriteOnly)) {...}
    QDataStream fout(&savefile);

    unsigned char ch = 0;
    quint64 a=0;
    quint64 temp=0;
    int count = 0; // 字符数量的记录
    while (sum>count) {...}
    // 搜索树字符替换
    openfile.close();
    savefile.close();
}
```

查重函数

```
void MainWindow::on_checkButton_clicked()
{
    QString filepath = QFileDialog::getExistingDirectory(this);
    QVector<QString> fileall;
    QVector<QString> vector;
    //
    QStringList fileList = getFiles(filepath);
    for(int i= 0; i < fileList.count(); i++){ ... }
    //遍历获得文件路径

    //全字符匹配
    for(int i=0;i<fileall.size();i++){ ... }

    //md5 值查重
    for(QHash<QByteArray, QStringList>::iterator it = fileMd5.begin();it!=fileMd5.end();++it){ ... }
    //使用iterator迭代器检查并遍历qhash关联容器【QHash{QByteArray}=Qstring】内的元素
    fileMd5.clear();
}
```

- 获得当前路径下所有文件的路径和 md5 值（进行不可逆的压缩）

```
QStringList MainWindow::getFiles(const QString &path)
{
    QStringList ret;
    QDir dir(path);
    //文件目录操作类 对文件目录进行操作
    //QDir::Dirs碰到隐藏目录会报错,所以使用Dir::AllDirs
    QFileInfoList fileInfoList = dir.entryInfoList(QDir::Files|QDir::NoDotAndDotDot|QDir::Dirs);
    for(int i = 0; i < fileInfoList.count(); ++i){
        if(fileInfoList[i].isDir()){
            QStringList files = getFiles(fileInfoList[i].absoluteFilePath());
            ret.append(files);
        }
        else{
            ret.append(fileInfoList[i].absoluteFilePath());
        }
    }
    return ret;
}

QByteArray MainWindow::getFileMd5(const QString &fileName)
{
    //QCryptographicHash 可用于生成二进制或文本数据的加密（不可逆的）哈希数据（不同长度的输入得到相同长度的输出）
    QCryptographicHash hash(QCryptographicHash::Md5);
    QFile file(fileName);
    if(!file.open(QIODevice::ReadOnly)){
        QMessageBox::information(this,"infomation","打开文件失败");
        return QByteArray();
    }
    while (!file.atEnd()) {
        QByteArray text = file.read(1024*10);
        hash.addData(text);
        QApplication->processEvents();
        //界面处理各种事件 避免假死
    }

    QByteArray fileMd5 = hash.result();

    return fileMd5;
}
```

- 比较不同路径下文件的值是否相等

```
int MainWindow::CompareMsg(char* msg1, char* msg2)
{
    // 将2个消息各自读取到buffer中
    FILE *pOne, *pTwe;
    long lsize1, lsize2;
    char* buffer1 = NULL;
    char* buffer2 = NULL;
    size_t result1, result2;

    pOne = fopen(msg1, "rb");
    pTwe = fopen(msg2, "rb");
    if ((pOne == NULL) || (pTwe == NULL)) { ... }

    fseek(pOne, 0, SEEK_END);
    lsize1 = ftell(pOne);
    //rewind(pOne);
    fseek(pOne, 0, SEEK_SET);
    fseek(pTwe, 0, SEEK_END);
    lsize2 = ftell(pTwe);
    rewind(pTwe);

    if (lsize1 != lsize2) { ... }
    // 比较大小

    buffer1 = new char[lsize1];
    buffer2 = new char[lsize2];
    if ((buffer1 == NULL) || (buffer2 == NULL)) { ... }
    // 有一方为空

    result1 = fread(buffer1, 1, lsize1, pOne);
    result2 = fread(buffer2, 1, lsize2, pTwe);
    if ((result1 != lsize1) || (result2 != lsize2)) { ... }

    int result = memcmp(buffer1, buffer2, lsize1); // 比较buf1buf2前ls字节

    fclose(pOne);
    fclose(pTwe);
    delete[] buffer1;
    delete[] buffer2;

    return result;
}
```

- (3) 建树的时候用到了树节点的树构造函数和堆的整理函数

树类构造函数：树节点声明，层次遍历存放到堆中

```
template<class T>
HuffmanTree<T>::HuffmanTree(T w[], int n) // 通过数组构建哈夫曼树
{
    Heap<HuffmanNode<T>*> hp(256); // 使用最小堆放森林

    for (int i = 0; i < n; i++) { ... }
    // 创建所有树的结点

    HuffmanNode<T> *parent = NULL;

    while (hp.size() > 1) { ... }
    // 将堆中森林按照层次遍历存放
    hp.RemoveMin(root);
}
```

- 树结点连接函数：把左右子节点和父母节点连接起来

```
template<class T>
void HuffmanTree<T>::mergeTree(HuffmanNode<T> &ht1, HuffmanNode<T> &ht2, HuffmanNode<T> *&parent)
{
    parent = new HuffmanNode<T>( ht1.data + ht2.data);
    parent->leftChild = &ht1;
    parent->rightChild = &ht2;
    ht1.parent = parent;
    ht2.parent = parent;
}
```

- 堆的构造函数:把结点数组中的结点读入堆中并对堆中结点的哈夫曼顺序进行调整形成用堆存储的层次结构存储的哈夫曼树

```
template<class T>
Heap<T>::Heap(T arr[], int n)
{
    maxHeapsize = n;
    heap = new T[maxHeapsize];
    for (int i = 0; i++; i<n) //复制数组元素到堆中
        heap[i] = arr[i];
    currentsize = n;
    int currentPos = (currentsize - 2) / 2;
    while (currentPos >= 0)
    {
        siftDown(currentPos, currentsize - 1); //从上到下调整
        currentPos--;
    }
}
```

- 堆中哈夫曼树的构造函数：把小频率结点往后（下层次）调整，大频率节点留在原来层次

```
template<class T> //下滑函数
void Heap<T>::siftDown(int start, int m)
{
    int i = start; //开始的节点
    int j = 2 * i + 1; //开始节点的左子女
    T temp = heap[i];
    while (j <= m) //检查是否在最后的位置
    {
        if (j < m && compare2(heap[j], heap[j+1]))
            j++; //让j指向两者中较小者
        if (compare1(temp, heap[j])) //小则不调整
            break;
        else
        {
            heap[i] = heap[j]; i = j; j = 2 * j + 1;
        }
    }
    heap[i] = temp;
}

template<class T>
void Heap<T>::siftUp(int start) //上调整函数
{
    int j = start, i = (j - 1) / 2; //j为子女, i为父节点
    T temp = heap[j];
    while (j > 0)
    {
        if (compare1(heap[i], temp))
            break;
        else
        {
            heap[j] = heap[i];
            j = i;
            i = (i - 1) / 2;
        }
    }
    heap[j] = temp;
}
```

4.实验亮点

4.1 选中 Qt 开发，进行良好的 UI 设计

1. 稳定

不闪退，bug 少，没有非预计的弹出对话框，无页面控件属性的细微变化，无测试数据问题。

2. 色彩舒适度高

采用单色面板——不仅创建和谐统一的视觉效果，而且把所有的视觉焦点的聚集在内容而非色彩搭配上。

3. 交互性与易用性强

图案合理布局和简洁化表达使复杂的内容精简化，增加用户的接受度。按钮风格与整体格调保持一致，达到整个界面的统一和谐，使用户无障碍融入到交互情境中。前置展示用户目标，根据用户的操作行为，推导/预判出用户下一步即将要执行的目标，提前将用户的操作目标展示出来，进一步地减少用户的操作路径。人性化，充分考虑用户的使用情景和使用方式。对系统的每一个细节进行相应的优化

4. 信息可读性强

排版优美：使用常规字体，浅色背景，适量留白和字体大小差异。提高用户的阅读体验，突出文字内容。留白是版式构成中重要的组成元素，字间距和行间距就是留白的具体表现，它决定着文本的可读性。如果文字间隔太近或太松，都会影响阅读。采用合理的留白使图像与其他元素进行区分。选择合理的字体大小，能达到主次分明。

5. UI 识别性强

用户能快速了解每个按钮的用处，符合大众认知习惯，让用户不用思考就可以作出反应。图标风格统一、整体性强传达含义清晰、准确、容易记忆。轮廓清晰，形状边缘棱角分明。

4.2 极大程度降低用户输入

凡是程序中可以给出选项的我们都给出选项了。其目的是提高用户的使用体验；同时也能使得无用的输入数据减少，提高程序的可用性。

4.3 准确性、性能、算法优势

4.3.1 贪心地图算法

在地图算法中，我们摒弃了 Dijkstra。它是点到所有点的最短路径算法，而我们使用的

算法是点到点的算法。Dijkstra 算法的复杂度是 $O(n^2)$ ，对比于我们使用的最短路径算法的复杂度 $O(\log_2 n)$ 来说，我们使用的算法复杂度更低一些，算法的执行效率更高。

采用贪心的算法，有以下优点：

- 需要获取的数据量少了几个数量级：因为只需要寻找联系的点即可，不需要全图搜索；算法速度快：Dijkstra 需要所有点，这显然。。。数据量非常大的时候计算量极为离谱。
- 手动入栈，不用递归，也能大大改善性能，更是针对超大数据规模的唯一方针。
- 有两种策略，但是**第一种快速寻找成功率较高，更能进一步提升性能！**

4.3.2 不定量规模数据封装

在各种事件，课程的功能中，我们都选择了链表和指针数组相结合的方案。这样不仅有效保存了数据，也能针对数据进行快速的排序；不用查询全部数据，对于数据较多的情况，跟传统链表遍历比较，无需一个一个查找。

在地图算法中，我们甚至使用了链表和数组相结合的方法（为了提高性能）：

PointPtrList 十个数据为一个数组进行封装；然后再对很多这种数组进行链表封装：用什么查什么，找到对应十组，继而在十个里面找，大大降低了查找的复杂程度，内存占用小。

4.3.3 额外功能多，优化好

- 整体代码设计：尽量减少用户输入，且对用户输入的合法性做了大量规定；
- 地图模块中：我们采用了连续的多路径功能（上一个地点直接在这一次路径中被选为起点，且显示在窗口上），大大增加了用户体验；
- 课程模块中：我们更是以按钮触发，用户只需要点击课程名，便可以进行该课程有关的所有操作。非常方便！

5.实验总结

5.1 用时

大致在 150h~200h，整体花费时间较长，可以看出我们小组对这次实验的重视程度。

5.2 设计体会

经过这次实验，我们充分了解了整个软件的开发流程。虽然走了不少弯路，也经常因为初始数据设计不完整完全而返工；对于用户优化体验，那更是折磨至深。

在本次课程设计的过程中遇到了很多困难与艰辛，但是也拥有了很多的努力与收获。本次课程设计的先导学科是数据结构，同时这部分的内容也叫做数据结构课程设计。在经过上学期的数据结构课程的学习之后，我们对数据结构部分的基础知识有了了解和认识，但是对于代码的书写以及在实际的软件应用方面所掌握的还是相对薄弱。在本次课程设计的过程中借此机会，一方面能够重新温习回顾在理论课上所学习的内容，并且在课程设计的过程中对

这些知识有了更加深刻、更为直观具体的了解能力。

作为一门软件设计实践类的课程,在此过程中我们得以数据结构相关的主要知识进一步进行运用,考察的不仅仅是知识理论的理解掌握,更是对实际应用和动手能力的一种强化、锻炼。经过本学期本次课程的学习,我们对数据结构这门重要课程的知识内容有了更深刻的理解,提高了我们的编码能力以及问题解决能力。将理论知识扩展到实际应用是课程设计的重要目的,因此在经历本次课程设计之后,我的这一方面能力也有了明显的提高。我将会认真总结反思本次课程设计中的得失与问题,在以后的各项设计和学习生活中起到重要的参考价值。课程初期经过题目的布置到需求的分析,从概要设计到命令程序的编写,再到最终的图形化用户交互界面的制作,最终进行调试测试与总结。经过一路跌跌撞撞,总算解决了很多问题,在开发流程上和设计上也积累了一定的经验。针对功能设计,我们知道了数据该如何整理,如何复用数据和代码,提高复用性和性能,以及常用数据要怎么处理,不常用数据该怎么处理。

还有一点深刻的体会:图形界面适配有点难做。

5.3 问题评价与改进意见

● 地图搜索算法

一开始我们便选用了贪心算法(根本没想到迪杰斯特拉,算法题做多了基本就记得贪心),但是在设计过程中出现了很多问题:怎么手动入栈?要存哪些数据?当前栈和下一个栈有什么联系?对于这个问题,我们画了很多设计图,最终确定这一版(我们认为最好的这一版)。

在编写过程中,也不断进行优化:比如已经走过的节点,再次到达时,是否有某些性质呢?当然是有的:可以和前一次进行比较,来确定当前节点是否入栈。这样维护了一个节点到起始地点的度量,来保证尽量少的重复计算。而对于子节点,当他对刚才走过来的节点方向走的时候,这又是毫无意义的。所以这种情况也要避免。在这种思路下,我们呢形成了第一个普通寻找方案。

但是一定不能停止,还要想着性能优化:于是开发出了快速算法。以前基本在几百个栈的规模,这一下子就减少到了200以内了。确实很棒!

实际上,在写这篇报告的时候,我们还有改进思路:写一个折中算法。当当前节点找不到离目的地更近的相邻点位的时候,那么对他的所有相邻节点进行入栈。这样来代替一般算法,又有着速度提升!

● 冲突检测问题

在写活动与课程冲突的时候,不知为什么想到了滑动窗口。于是我们便沿用了这个思路,做出了简单的,易懂的,方便的判断方式。

其他检测基本没有什么亮点了,基本就是常规的思路。

● 用户体验优化问题

这里也走了很多弯路。基本都在第一版的UI上大改而来。但是经过一番折腾,我们总算了解了整个开发流程,但是在窗口的大小变化上,界面调换上还有很大进步空间。