

# 校园导航系统的设计

---

## 一、设计任务的描述:

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。

## 二、功能需求说明及分析

### 1) 功能需求

#### (1) 课程信息管理和查询

学生可以输入课程名称，或者通过课程表查询课程的上课时间、上课地点、课程资料、当前进度、已交作业、待交作业、课程群、考试时间和考试地点等信息。

学生可以上传和更新课程资料、上传完成的作业内容。对于课程资料和作业可以进行查询、压缩和去重。

系统管理员统一发布考试时间和考试地点，并可以发布和更改课程上课的时间和上课地点。

#### (2) 课外信息管理和查询

学生可以输入课外活动信息，课外活动包括个人活动和集体活动；个人活动可以包括：自习、锻炼、外出等，集体活动可以包括：班会、小组作业、创新创业、聚餐等。

学生可以设定活动闹钟，闹钟可以是一次性的和周期性的，用于活动提醒。

系统可以检测个人活动、集体活动和课程的时间冲突，并给出提示。

#### (3) 课程导航

学生向系统输入课程名称、上课时间或者上课地点，系统自动进行路径规划。

课程名称可以是《数据结构》，系统会自动根据学生的班级信息和最近的上课时间（未开始）查询上课地点；

上课时间可以是“周五10点”系统会自动根据学生的班级信息和最近的上课时间（未开始）查询上课地点；

上课地点是上课的物理位置，例如“教三楼111教室”；

起点和终点可以在不同校区，需要考虑校区间的交通方式；

校区间的交通方式为：定点班车（可以自行规划班次时刻表）和公共汽车（可等间隔发车）。

#### **(4) 模拟系统时间**

系统依据时钟向前推进，时间精度为小时，且以计算机的10秒作为模拟系统的1小时（可以支持快进）；

人机交互时暂停系统时间推进（例如用户输入信息时）；可以通过加入时钟暂停按钮或者命令来实现。

#### **(5) 建立日志文件**

记录学生课程和活动状态变化，系统提醒的信息，输出的导航信息，以及学生输入的信息和各种查询操作。

## **2) 功能需求分析**

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。

针对以上问题，我们着手设计一款线下教学辅助系统来对学生的课程、学习、日常生活、出行方面提供便捷的服务。

第一点：学习方面。我们的线下课程辅助系统可以统计出用户的课程信息，包括当天的课程安排，上课地点、课程资料、该课程是否有未交作业、课程绑定的课程群。当临近考试时，系统会自动更新，提示用户不同科目考试时间、考试地点。同时该程序也会向用户提供指定科目的课程资料，提供渠道让用户能够提交作业。对所有完成的作业以及提供的资料，系统都会储存，用户可以随时随地地进行查询、压缩、去重、下载，保证用户的课程资源的完整性。

第二点：生活方面。我们的线下客户课程辅助系统会提供课外活动的辅助功能。课外活动分为两种：个人和集体的。个人活动由用户自己设定，同时程序可以让用户设定闹钟，让用户能够及时地完成对应的活动。集体活动由集体负责人发起，用户可以输入自己的学号来查询自己需要参加的集体活动，设定闹钟，提醒用户准时参加。当用户安排的活动在事件或空间上起冲突时，系统会提醒用户活动出现冲突，让用户能够及时进行调整。

第三点：出行方面。在我们大学生活当中，我们每天都要到不同的教室上课，所以出行方面也是一大问题。用户可以输入指定的上课地点，我们系统利用算法，向用户提供多种不同的出行方案，分别是距离最短、时间最短、不同交通工具的最短时间。同时，当用户有其他需求时，用户可以输入出发地点和目的地，系统也会利用算法进行提示。同时，系统会引入学校的公交车的时刻表，方便用户在不同校区之间进行出行。

同时，系统会做好日志文件，记录学生课程和活动状态变化，系统提醒的信息，输出的导航信息，以及学生输入的信息和各种查询操作。我们也会针对不同的功能设置不同的图形化界面。

## **三、总体方案设计说明**

### **1) 软件开发环境**

编程语言：C++

软件环境：Visual Studio ,QtCreator

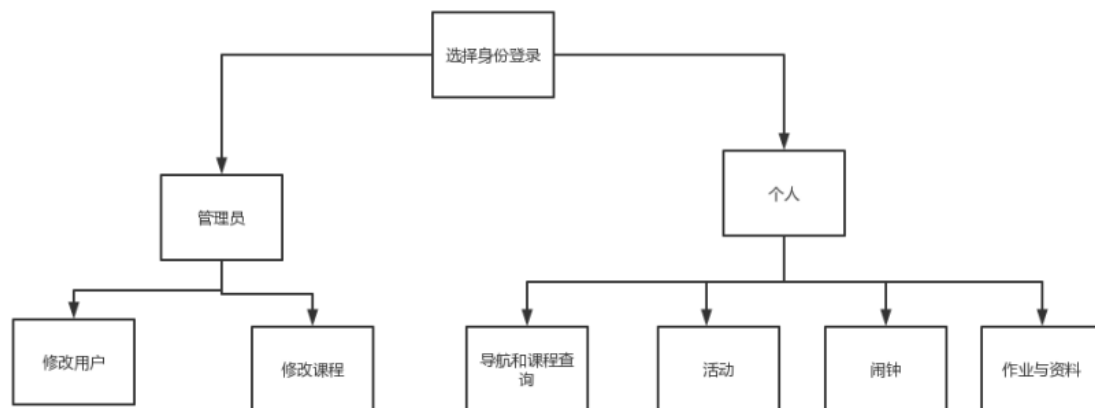
硬件环境：搭载 Windows10 环境的 PC 机

## 2) 总体结构

首先进行系统所需要的变量和结构体等的定义与初始化，之后系统开始运行，用户选择个人的身份并进行登录,由用户选择当前运行的模式，系统初始默认模式为正常运行模式，每 10s 向前推进一个时间点，用户可在所需要的时间点按下相应按键进行相应的模式。

个人用户分为课程查询导航,作业资料,闹钟,活动,四个模块

管理员有控制用户名单和修改课程两个模块



## 3) 模块划分

### Ui模块:

#### 1. 注册

选择身份输入密码用户名注册

#### 2. 登录模式

选择身份输入密码登录

#### 3. 课程导航模式

该模块上班部分是地点和课程查询,采用kmp算法,属于关键字就会跳出对应的结果,在查询课程时可以采用时间,名称,地点三种查询方式,并会根据目前的时间来输入当前的课程状态(本周有或下周有)

下半部分是导航,可以选择无干扰,有干扰,和有干扰加自行车导航,出发地和目的地名称需要和实际一致,可以通过上半部分查询来获取.

#### 4. 活动修改查询模式

通过输入关键字和选定活动性质来按复选框查询,可以查询活动,采用kmp算法,属于关键字就会跳出对应的结果,在选定活动性质时间和名称来添加或删除活动

## 5. 闹钟修改查询模式

通过选定闹钟性质来按复选框查询,在选定闹钟性质时间和名称来添加或删除闹钟

## 6. 作业与资料修改查询模式

通过输入关键字来按复选框查询,可以查询已完成作业和未完成作业和资料,采用kmp算法,属于关键字就会跳出对应的结果,通过文件对话框选择文件,或选则目录进行压缩再上传,下载时文件一律下载到download文件夹

## 7. 管理员模式

左边可以搜索用户,并进行删除,右边可以搜索,更新或者添加课程(课程需要在空闲的时间进行添加)

## 功能模块:

### 1.时间模式

取当前时间,按用户要求进行时间模拟并在user面板上进行显示并提示闹钟

### 2.导航模式

搜索最短路径

### 3.匹配模式

用kmp实现关键字搜索

### 4.日志初始化与修改模式

把数据录入数组

### 5.压缩模式

用哈夫曼编码实现对文本的压缩

### 6.文件选择模式

用fileDialog来打开文件对话框

通过的QZipWriter的addFile来实现对目录的压缩

## 4)数据结构说明和数据字典

### 1.全局变量声明

```
int **campus1**=[50][50]; //西土城地图

int **campus2**=[50][50]; //沙河地图

buildings **building**[100];

string **contorlaccount** = "bupt2020211"; //管理员账号

string **contorl_paaword** = "123456"; //管理员密码
```

```
string **nowaccount**;//目前使用者账号

myclas **clas**[100];//课程

int **week**;//当前星期

int **hour**;//当前小时

int **count6** = 0;//记录建筑数目

myact **act**[100];//活动

int **count5** = 0;//记录课程数

note **notice1**[100];//闹钟

int **count7** = 0;//记录闹钟数

string **time1**;//时间

homework **myhomework**[1000];//作业

int **count8** = 0;//作业数

char **way**= 's';//控制多线程

QString **qstr2**;//转换时间

int **waytofind**=0;//判断找路的方式

int **startc**=1;//出发地点校区

int **endc**=1;//目的地校区

classword **word**[1000];//资料

int **count9**;//资料数

int **worh**=0;//判断文件类型

int **kindact**=0;//判断查询活动类型

int **kindact2**=0;//判断修改活动类型

int **kindnot**=0;//判断查找闹钟类型

int **kindnot2**=0;//判断修改闹钟类型

int **conkind**=0;//判断管理员操作类型

myaccount **A**[1000];//个人账号

int **count10**;//个人账号数目
```

## 2.结构体声明\*\*

```
class **buildings** { //建筑

public:

    char **buildingname**[100];

    int **campus** = 0;

    int **index** = 0;

    string **name** = {};

    void **printf**() {

        •    cout << "该建筑名称为" <<name << endl;

    }

};

class **myclas** { //课程

public:

    string **name** ;

    string **time**;

    string **campus**;

    string **buliding**;

    string **classroom**;

    string **timeofexam**;

    string **buildingofexam**;

    string **classroomofexam**;

    string **stage**;

    string **group**;

    char **classname**[1000];

    char **timename**[1000];

    char **buildingname**[1000];

    void **printf**() {
```

- `cout << "该课程上课时间为" <<time<< endl;`
- `cout << "上课及考试校区为" <<campus << endl;`
- `cout << "上课地点为" << buliding<<classroom << endl;`
- `cout << "考试时间为" << timeofexam << endl;`
- `cout << "考试地点为" << buildingofexam<<classroomofexam << endl;`
- `cout << "教学进度为" << stage << endl;`
- `cout << "课程群为" << group << endl;`

`}`

`};`

`class **myact** { //活动`

`public:`

`string **time**;`

`string **kind**;`

`string **name**;`

`char **actname**[100];`

`char **acttime**[100];`

`void **printf**() {`

- `cout << "该活动时间为" << time << endl;`
- `cout << "活动类型为" << kind << endl;`
- `cout << "活动名称为" << name<< endl;`

`}`

`};`

`class **note** { //闹钟`

```

public:

    string **type**;

    string **time**;

    string **kind**;

    void **printf**() {

        •    cout << "闹钟种类为" << time << endl;

        •    cout << "闹钟时间为" << kind << endl;

        •    cout << "闹钟备注为" << kind << endl;

    }

};

class **homework** { //作业

public:

    char **name**[100];

    int **stage**;

    void **printf**() {

        •    cout << "作业名称为" << name << endl;

        •    if(stage==1)cout << "未完成" << endl;

        •    if(stage == 2)cout << "已完成" << endl;

    }

};

class **x** { //kmp中用来记录数据

public:

    int **data**;

    int **way**;

};

class **classword** { //资料

public:

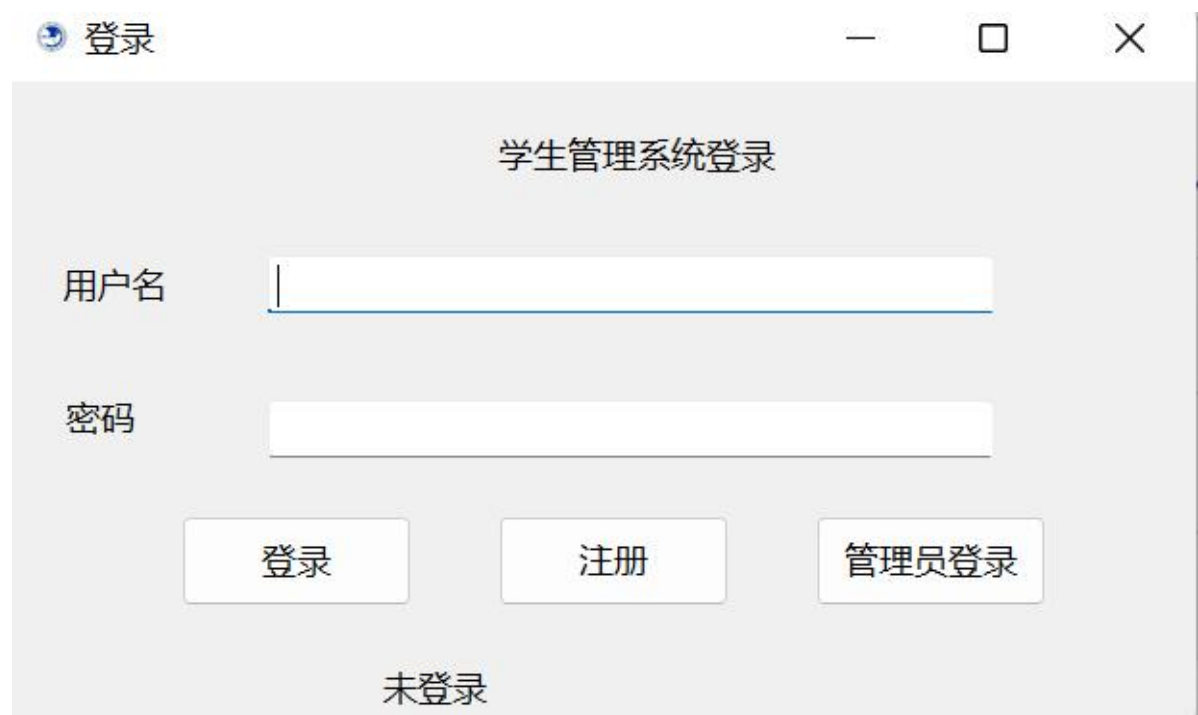
    char **name**[100];

```



```
};  
  
class **myaccount**{//个人账号  
  
public:  
  
    char **name**[100];  
  
    char **pass**[100];  
  
};
```

在此之列出了非ui类型的结构体,ui类型结构体篇幅太多,不在此展开,详情见源码,具体效果如下:



登录

学生管理系统登录

用户名

密码

登录 注册 管理员登录

未登录



查询任何地点

搜索

查看地图

本部

选择方式

查询课程

搜索

时间

本部

搜索结果

输入出发地

本部

输入目的地

本部

导航

无干扰

未查询

下载作业与资料

查询作业或资料请输入名称

已完成作业

输入目标文件名称

请输入想上传(或想下载到的)文件地址

C://Users//16062//Desktop//zdj.txt

作业

上传

查找活动

活动性质

个人

搜索

输入活动或时间名称

按名称查

性质

个人

### 3.函数声明

```

void **init**();//初始化

void **buildact**();//读取活动文件

void **myclasss**();//读取课程文件

void **personact** (string **act**);//读取活动文件

int **yasuo** (char **j**,char **s**[]);//哈夫曼树压缩

void **controlhomework**();//对作业进行操作

void **setnotice**();//设置闹钟

void **noticestart**();//读取闹钟文件

void **delputhomework** (string **p**);//删除作业

void **searchclass** (char **son**[100]);//搜索作业

void **searchactive** (char **son**[100], string **kind**);//搜索活动

void **searchtime** (char **son**[100]);//搜索时间

void **searchbuilding** (char **son**[100],string **campus**);//搜索建筑

void **searchhomework** (char **son**[100]);//搜索作业

void **searchway** (char **son**[100], int **campus**);//搜索路径

```

```

void **BubbleSort**(x **arr**[], int **length**); //冒泡排序

int **KMP**(const char* **Text**, const char* **Pattern**); //kmp

void **homeworkstart**(); //读取作业文件

void **accountstart**(); //读取账号文件

void guide::guideway(int way) { //控制拥挤度

void control::searchclass(char son[100], string campus) { //搜索课程

void control::searchtime(char son[100], string campus) { //通过时间搜索课程

void control::searchbuilding(char son[100], string campus) { //通过建筑搜索课程

void control::on_comboBox_3_activated(int index) //选择查找方式

void control::addd(string time) //添加课程

void control::on_comboBox_2_activated(int index) //选择修改or添加课程

void control::reload(int con, string time) //把修改后的信息写入本地文本文件

int control::search2(string campus) { //看建筑是否符合输入要求

int checktim(string week, string hour) //看输入时间是否符合规范

int control::search3(string name) //查看输入的课程各方面是否符合要求

int control::search1(string name) { //看输入的课程各方面是否符合规范

int control::search4(string name) { //对符合要求的课程进行删除

void control::on_comboBox_activated(int index) //选择对课程的操作方式

string chan(int a) { //把数字转为字符

int findmin(int D[], int S[]) { //找到未确定点的最小距离

void guide::Dijkstra(int sign, int startpoint, int sign1, int endpoint) { //迪杰斯特拉算法

void guide::yanzhen() { //查找是否有校车可以坐

void findcation::on_comboBox_2_activated(int index) //选择查找模式

void findcation::searchactive(char son[100], string kind, int kindact) //查找活动

void findcation::on_comboBox_activated(int index) //切换查找活动类型

void findcation::on_comboBox_4_activated(int index) //选择对活动操作

int findcation::findcrash(string time) { //查找冲突

```

```
void findcation::setcat(string time)//设置活动

int findcation::change(string time)//修改活动

void findcation::on_comboBox_3_activated(int index)//添加活动

void guide::searchway(char son[100],int campus) { //寻找建筑

void guide::on_comboBox_3_activated(int index)//选择校区

void guide::on_comboBox_activated(int index)//选择找路的方式

int numturn(string num){ //字符转数字

void guide::checktime(char a[]){ //核对今天的课程

void guide::searchclass(char son[100],string campus) { //查找课程按名字

void guide::searchtime(char son[100],string campus) { //查找课程按时间

void guide::searchbuilding(char son[100], string campus) { //查找课程按建筑

void guide::on_comboBox_4_activated(int index)//选择查找校区和方式

void guide::on_comboBox_2_activated(int index)//确定导航的开端和末尾

void guide::on_comboBox_5_activated(int index)//选择起点

void guide::on_comboBox_6_activated(int index)//选择终点

void homework1::searchhomework(char son[100],int stage) { //查找作业

void homework1::searchword(char son[100]){ //查找资料

void homework1::finishwork(){ //将作业字符串转化为数组

void homework1::unfinishwork(){ //查找没完成的作业,字符串化为数组

void homework1::classword(){ //查找资料,字符串化为数组

void homework1::on_comboBox_2_activated(int index)//选择查找对象

int handin(string name,string way){ //查找未上交作业

int reload(string name,string way){ //查找可以更新的资料

int download(string name,string way){ //查找可以下载到资料

void delputhomework(string p,string way,QString l)//删去上交的作业

void homework1::on_comboBox_activated(int index)//上传或下载资料

void homework1::changeback(string str,QString l){ //返回后缀
```

```

string homework1::findsin(string name,string way){//找后缀

void homework1::on_comboBox_3_activated(int index)//选择文件类型

void homework1::on_pushButton_clicked()//设置文件选择框

void login::on_pushButton_clicked()//登录

void login::on_pushButton_2_clicked()//注册

void login::on_pushButton_3_clicked()//管理员

int homework1::yasuo(char j,char s[],char s1[])//压缩

// 初始化哈夫曼树结点
bool homework1::Init(char* filename)

// 构建完整的哈夫曼树
static void CreatHaffmanTree()

// 构建哈夫曼树中寻找已有结点中权值最小的两个结点
static void SearchNode(int Range, int& pos1, int& pos2)

// 得到哈夫曼编码匹配
static void CreatHaffmanCodeMap(int root, char TempCode[CODESIZE + 1], int len)

// 编码
bool homework1:: Encode(char* sourcefile, char* destfile)

// 解码
bool homework1:: Decode(char* sourcefile, char* destfile)

// 清除缓冲区
static void ClearBuffer()

void notice::searchnot(int kind){//设置闹钟

void notice::on_comboBox_activated(int index)//选择设置闹钟类型

void notice::on_comboBox_4_activated(int index)//选择操作类型

int notice::findcrash(string time){//找闹钟时间冲突

void notice::setnot(string time)//找时间冲突

int notice::change(string time) {//修改闹钟

void notice::on_comboBox_3_activated(int index)//输入闹钟内容

void threadd::run()//子线程时间流动

void user::myPrint(){//打印闹钟内容

void user::on_pushButton_3_clicked()//展示导游

```

```

void user::on_pushButton_4_clicked()//展示作业

void user::on_pushButton_5_clicked()//展示闹钟

void user::on_pushButton_6_clicked()//撤回界面

void user::on_comboBox_activated(int index)//暂停时间

void user::on_comboBox_2_activated(int index)//时间加速

```

## 五、模块主要算法设计说明 (ui模块无算法,通过调用下列功能模块进行操作)

### 1) 导航模块

#### (一) 算法思想

整体设计思想是采用Dijkstra算法.

根据两地距离, 求出每一对之间的最短距离, 在得到这些最短距离中最远的一对。利用两个两个函数的调用和双循环嵌套的算法遍历求出每队顶点之间的最短距离并比较得出最远距离, 保存这一个最远距离的起点和终点和距离。输出之前求出的最短距离中最远距离的始点和终点,作为最近路径

#### (二) 算法详解

找最短距离

```

int min = 10000;

int minnum = -1;

for (int count = 0; count < 50; count++) //不断循环得到最小距离
    if (D[count] < min && D[count] != 0 && S[count] != 1) {
        min = D[count];
        minnum = count;

return minnum;//返回最小值

```

使用了一个递归,将不同的校区横跨拆解为同一个校区之间的流动,并输出最短路径

```

//对不同校区的距离进行拆解

•   if(startpoint!=0) Dijkstra(1, startpoint, 1, 0);

•   ui->textBrowser->append(QString::fromStdString("经过40分钟车程到达沙河校区北门"));

•   yanzhen();

•   if (endpoint != 0) Dijkstra(2, 0, 2, endpoint);//拆成两个过程

```



//同一校区找距离

```
•   for (int con = 0; con < 50; con++) {  
  
•       P[con] = startpoint;  
  
•       S1[con] = 0;  
  
•   } //设置起点  
  
•   P[startpoint] = -1;  
  
•   for (int count = 0; count < 50; count++) {  
  
•       D[count] = campus2[startpoint][count];  
  
•   }  
  
•   S1[startpoint] = 1;  
  
•   for (int count = 0; count < 50; count++) { //不断循环找最小距离  
  
•       int minnum = findmin(D, S1);  
  
•       if(minnum >= 0) S1[minnum] = 1;  
  
•       for (int con1 = 0; con1 < 50; con1++) {  
  
•           if (S1[con1] == 0 && (D[con1] > (D[minnum] + campus2[minnum][con1])))  
//更新最短距离  
  
•               D[con1] = D[minnum] + campus2[minnum][con1];  
  
•               P[con1] = minnum;  
  
•           }  
  
•       }  
  
•   }  
  
•   int pre = P[endpoint]; //迭代找上一个结点  
  
•   while (pre != -1) {  
  
•       way[con2] = pre;  
  
•       con2++;  
  
•       pre = P[pre];  
  
•   } //倒置结点顺序
```

```
//输出导航信息
```

```
•   for (con3 = con2 - 2; con3 >= 0; con3--) {  
  
•       ui->textBrowser->append(QString::fromStdString("经过"+chan(campus2[way[con3 + 1]][way[con3]])+ "分钟"));  
  
•   }
```

设置拥挤度:

```
    srand((int)time(0)); //设置随机数  
  
    if (way == 3) {考虑拥挤度  
  
•       for (int count = 0; count < 50; count++) {  
  
•           for (int count1 = count; count1 < 50; count1++) {  
  
•               if (campus1[count][count1] != 10000) {  
  
•                   double r = ((rand() % (6)) + 5); //设置拥挤度为0.5到1之间  
  
•                   double r1 = 10*campus1[count][count1] / r;  
  
•                   campus1[count][count1] = (int)r1;  
  
•               }  
  
•           }  
  
•       }  
  
    if (way == 2) {对于自行车道进行修改//自行车道通过时间为2min  
  
•       campus1[0][6] = 2;  
  
•       .....省下略过,只对自行车道进行操作  
  
    }  
  
    for (int con = 0; con < 50; con++) { //将对角矩阵变完整  
  
•       for (int con1 = con; con1 < 50; con1++) {  
  
•           campus1[con1][con] = campus1[con][con1];  
  
•       }  
  
    }
```

```
}
```

### (三) 算法特点

本算法计算最短路径的常规步骤与传统Dijkstra算法相同，都是寻找权值最小(风险值)最小的边，将对应顶点加入已确认顶点集合，再更新整个map，而之后根据两种不同的方案计算最佳路径是算法最具特点部分，

#### 算法时空分析：

设输入的顶点数为n；输入边数为m；

StructGraph: $O(m \cdot n)$

GetMaximumShortestPath: $O(n^2)$

DFSShortestPath: $O(m \cdot n)$

Output: $(n^2)$

### (四)模块调用

在运行课程查询导航模块时,涉及到导航时调用该模块算法.

## 2) 匹配模块

### (一) 算法思想

用kmp算法对关键字进行匹配,再冒泡排序来根据匹配程度来对符合条件的内容进行输出

### (二) 算法详解

KMP算法主要优化字符查找的效率出发，通过观察和假设，将问题转化为寻找一个最小偏移量的问题，之后进一步将问题转化为寻找模式串中前缀和后缀的最大匹配长度。最后通过这个最大匹配长度，反向计算出最小偏移量，得到的问题的解。问题的转化和化简，循序渐进，最终得到了这个问题的一个高效解。

基本过程:

求模式串strM的next数组

遍历比较待匹配的字符串strN（过程=遍历strN+遍历时出现strM[j]的回跳）

比较strN[i]、strM[j]时可能出现的情况为：

2.1 当前字符匹配时，同时移动  $i++$ ， $j++$

2.2 当前字符不匹配，且 $j=0$ 时，只移动  $i++$ ， $j=0$ 不动

2.3 当前字符不匹配，且 $j!=0$ 时， $i$ 不变，strM[j]回跳，当前strN[i]时最多回跳j次

求next数组:

KMP：

```

// 求模式串T的next函数值并存入数组 next。

int **j** = 0, **k** = -1;

next[0] = -1;

while (T[j/*+1*/] != '0')

{
    •   if (k == -1 || T[j] == T[k])
    •   {
    •       ++j; ++k;
    •       if (T[j] != T[k])
    •           next[j] = k;
    •       else
    •           next[j] = next[k];
    •   }// if
    •   else
    •       k = next[k];
} // while

// for(int i=0;i<j;i++)

//{

//   cout<<next[i];

//}

//cout<<endl;

}

```

匹配阶段:

```

if (!Text || !Pattern || Pattern[0] == '0' || Text[0] == '0')//

    •   return -1;//空指针或空串，返回-1。

int **len** = 0;

```

```

const char* **c** = Pattern;

while (*c++ != '0')//移动指针比移动下标快。

{
    • ++len;//字符串长度。

}

int* **next** = new int[len + 1];

get_nextval(Pattern, **next**);//求Pattern的next函数值


int **index** = 0, **i** = 0, **j** = 0;

while (Text[i] != '0' && Pattern[j] != '0')

{
    • if (Text[i] == Pattern[j])

    • {

    • ++i;// 继续比较后继字符

    • ++j;

    • }

    • else

    • {

    • index += j - next[j];

    • if (next[j] != -1)

    • j = next[j];// 模式串向右移动

    • else

    • {

    • j = 0;

    • ++i;

    • }

    • }

}

}

delete[]next;

```

```

    if (Pattern[j] == '0')

        return index;// 匹配成功

    else

        return -1;

}

```

冒泡排序:

```

for (int **i** = 0; i < length; i++)

{

    for (int **j** = 0; j < length - i - 1; j++)

    {

        if (arr[j].way > arr[j + 1].way)

        { //比较,大者往后排

            int **temp**, **temp2**;

            temp = arr[j + 1].way;

            temp2 = arr[j + 1].data;

            arr[j + 1].way = arr[j].way;

            arr[j + 1].data = arr[j].data;

            arr[j].way = temp;

            arr[j].data = temp2;

        }

    }

}

```

### (三) 算法特点

可以较快匹配出对应字符串,效果较好。

KMP的时间复杂度分析

假设m为模式串strM的长度, n为待匹配的字符串strN的长度。

$O(m+n) = O([m, 2m] + [n, 2n])$  = 计算next数组的时间复杂度+遍历比较的复杂度。

也就是：

计算next数组时的比较次数介于 $[m, 2m]$ 。

遍历比较的比较次数介于 $[n, 2n]$ ,最坏情形形如 $T="aaaabaaaab", P="aaaaa"$ 。

所以算法时间复杂度为 $O(m+n)$ 。

## (四)模块调用

在运行课程查询导航,作业资料,闹钟,活动,注册和管理员控制用户名单和修改课程模块时,涉及到对数据查询时调用该模块算法。

## 3)上交下载作业资料模块

### (一) 算法思想

霍夫曼编码使用变长编码符号（如文件中的一个字母）进行编码，其中变长编码表是通过一种评估来源符号出现机率的方法得到的，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。

- 1.按照字符分析要压缩的文件得出结果（有哪些字符，每个字符出现的次数）。
- 2.根据字符出现的次数构建哈夫曼树（得出字符的哈夫曼编码）。
- 3.根据字符的哈夫曼编码进行转换、压缩，然后创建压缩文件。
- 4.读取压缩文件，读出哈夫曼编码和字符的对照表。解压缩。

### (二) 算法详解

设置结构:

```
// 哈夫曼树结点

struct HuffmanNode

{

    char data;                // 每八位二进制视作一个字符读入，最后不足八位的再另外视为一个字符

    long long weight;

    int parent, lchild, rchild;

};

// 哈夫曼树

struct HuffmanTree

{

    HuffmanNode node[TREESIZE - 1];

    int len;
```

```

};

// 数据（单字符对应的数值）和哈夫曼编码的匹配结点

typedef struct MapNode

{

    char data;

    char Bin[CODESIZE + 1];    // 存二进制串

};

// 基于哈夫曼树得到的数据和哈夫曼编码的一整组匹配

struct Map

{

    MapNode node[LEAFSIZE];

    int len;

};

// 文件操作中存储源文件，中间文件，目标文件的字段使用

static char source[READSIZE], temp[READSIZE * CODESIZE + CODESIZE + 8],
dest[READSIZE * CODESIZE + CODESIZE + 8];

// H是哈夫曼树，HuffmanCodeMap是字符和哈夫曼编码的匹配

static HuffmanTree H;

static Map HuffmanCodeMap;

```

使用的函数:

```

static bool Init(char* filename);    // 初始化哈夫曼树结点

static void CreatHaffmanTree(void); // 构建完整的哈夫曼树

static void SearchNode(int Range, int& pos1, int& pos2); // 构建哈夫曼树中寻找已有结点中权值最小的两个结点

static void CreatHaffmanCodeMap(int root, char TempCode[CODESIZE + 1], int len);
// 得到哈夫曼编码匹配

static bool Encode(char* sourcefile, char* destfile);    // 编码

```



```

static bool Decode(char* sourcefile, char* destfile);          // 解码

static void PrintMessage(void);          // 输出哈夫曼树和哈夫曼编码匹配的信息

static void ClearBuffer();          // 清除缓冲区

HuffmanCodeMap.len = 0;

bool OK = true;    // 判断是否正确读入文件，若未能正确读入则直接结束程序

char TempCode[CODESIZE + 1]; // 在CreatHuffmanCodeMap中临时存储在该节点对应的编码
（该节点不是叶子结点则TempCode就不是有效编码）

memset(TempCode, 0, sizeof(TempCode));

char sourcefile[NAMESIZE], encodefile[NAMESIZE], decodefile[NAMESIZE];    // 各
个文件的文件名（文件路径名）

```

功能选择:

```

    if (j == '1')//上传功能
    {
        OK = Init(sourcefile);//判断源文件是否能打开

        if (!OK)

            return 0;

        CreatHuffmanTree();

        CreatHuffmanCodeMap(H.len - 1, TempCode, 0);

        OK = Encode(sourcefile, s);//编码

        if(OK){

            return 1;

        }

        else return 0;

    }

    else if (j == '2')//下载功能

    {

        OK = Decode(s, decodefile);//解压

        return 0;

    }

```

初始化哈夫曼树结点:

```
H.len = 0;

FILE* fptr = fopen(filename, "rb");

if (fptr == NULL)//判断是否能打开

{

    •    ui->textBrowser->append(QString::fromStdString( "找不到待编码的目标文件"));

    •    return false;

}

fseek(fptr, 0, SEEK_END);//重定位流(数据流/文件)上的文件内部位置指针。

long long count = ftell(fptr);    // 除去结尾的EOF

fseek(fptr, 0, SEEK_SET);

while (count)

{

    •    int size = count >= READSIZE ? READSIZE : count;

    •    fread(source, sizeof(char), size, fptr);//从指定文件中读取块数据

    •    for (int i = 0; i < size; i++)

    •    {

    •        bool find = false;

    •        for (int j = 0; j < H.len; j++)

    •            if (H.node[j].data == source[i])//字符写入data中

    •            {

    •                H.node[j].weight++;//权重升级

    •                find = true;

    •                break;

    •            }

    •            if (!find)

    •            {
```

```

    •      H.node[H.len].data = source[i];

    •      H.node[H.len].weight = 1;

    •      H.node[H.len].lchild = H.node[H.len].rchild = H.node[H.len].parent =
-1;

    •      H.len++;

    •      }

    •      }

    •      count -= size;

    }

    fclose(fp);

    return true;//完成权重树的构造

}

```

---

// 构建完整的哈夫曼树

```

static void CreatHaffmanTree()

{

    int count = H.len - 1; // 哈夫曼树只有度为2/0的结点, num(d=2) = num(d=0) - 1

    while (count--)

    {

    •      int pos1, pos2;

    •      SearchNode(H.len, pos1, pos2);

    •      H.node[H.len].lchild = pos1, H.node[H.len].rchild = pos2,
H.node[H.len].parent = -1;//初始化孩子结点

    •      H.node[H.len].weight = H.node[pos1].weight + H.node[pos2].weight;//赋值

    •      H.node[pos1].parent = H.node[pos2].parent = H.len;

    •      H.len++;

    }

}

```

```
// 构建哈夫曼树中寻找已有结点中权值最小的两个结点

static void SearchNode(int Range, int& pos1, int& pos2)
{
    pos1 = -1, pos2 = -1; //权值上pos1为最小,pos2第二小

    for (int i = 0; i < Range; i++)
    {
        // 有双亲的结点不能作为根节点被合并

        if (H.node[i].parent != -1)
        {
            continue;
        }

        // 新结点权值比已有的两个小

        if (pos1 == -1 || H.node[pos1].weight > H.node[i].weight)
        {
            pos2 = pos1, pos1 = i;
        }

        // 新结点权值在两个结点的权值之间

        else if (pos2 == -1 || (H.node[pos1].weight < H.node[i].weight &&
            H.node[i].weight < H.node[pos2].weight))
        {
            pos2 = i;
        }
    }

    return;
}
```

```
// 得到哈夫曼编码匹配,根据树的权重将每一个每一个字符都翻译成二进制数

static void CreatHaffmanCodeMap(int root, char TempCode[CODESIZE + 1], int len)
{
    if (H.node[root].lchild == -1 && H.node[root].rchild == -1)
    {
        HuffmanCodeMap.node[HuffmanCodeMap.len].data = H.node[root].data;

        strcpy(HuffmanCodeMap.node[HuffmanCodeMap.len].Bin, TempCode); //添加结点
    }
}
```

```

    HuffmanCodeMap.len++;

}

if (H.node[root].lchild != -1)

{

    TempCode[len] = '0';

    CreatHuffmanCodeMap(H.node[root].lchild, TempCode, len + 1);

    TempCode[len] = 0;

}

if (H.node[root].rchild != -1)

{

    TempCode[len] = '1';

    CreatHuffmanCodeMap(H.node[root].rchild, TempCode, len + 1);

    TempCode[len] = 0;

}

return// 输出哈夫曼树和哈夫曼编码匹配的信息 编码

}

```

拿编码后的二进制数去代替字符

```

bool homework1:: **Encode**(char* **sourcefile**, char* **destfile**)

{

    FILE* **fptrs** = fopen(sourcefile, "rb");

    FILE* **fptrD** = fopen(destfile, "wb");

    if (fptrs == NULL || fptrD == NULL)//看文件和源文件是否都打得开

    {

        ui->textBrowser->append(QString::fromStdString( "文件名出现错误，编码失败！
n"));

        if (fptrs)

            fclose(**fptrs**);

    }
}

```

```

•   if (fptrD)

•       fclose(**fptrD**);

•   return false;

}

fwrite(&H, sizeof(HuffmanTree), 1, **fptrD**);    // 存入哈夫曼树，方便后续解码//

memset(**source**, 0, sizeof(**source**)); //置0

memset(**temp**, 0, sizeof(**temp**));

memset(**dest**, 0, sizeof(**dest**));

fseek(**fptrS**, 0, SEEK_END);

long long **count** = ftell(**fptrS**);    // 除去EOF的所有字符

fseek(**fptrS**, 0, SEEK_SET);

while (count > 0)

{

•   int **size** = count > READSIZE ? READSIZE : count;

•   fread(**source**, sizeof(char), size, **fptrS**);

•   for (int **i** = 0; i < size; i++)

•   {

•       char **ch** = source[i];

•       for (int **j** = 0; j < HuffmanCodeMap.len; j++)

•           if (HuffmanCodeMap.node[j].data == ch)

•           {

•               strcat(**temp**, HuffmanCodeMap.node[j].Bin);

•               break;

•           }

•   }

•   int **templen** = strlen(temp), **destlen** = 0;

•   int **rev** = templen % 8;

•   for (int **i** = 0; i < templen / 8; i++)

```

```

• {
•     int **ch** = 0;
•     for (int **j** = 0; j < 8; j++)          // 低地址高位的二进制码==>0-255==>字符
•         if (j)
•             ch = ch * 2 + (temp[8 * i + j] - '0');
•         else
•             ch = ch * 2 - (temp[8 * i + j] - '0');
•     dest[destlen++] = (char)ch;
• }
• fwrite(dest, sizeof(char), destlen, **fptrD**);
• count -= size;
• char **copy**[10];                          // 不足八位的留着下次循环补齐
• for (int **i** = 0; i < rev; i++)
•     copy[i] = temp[templen - rev + i];
• memset(**temp**, 0, sizeof(**temp**));
• for (int **i** = 0; i < rev; i++)
•     temp[i] = copy[i];
• memset(**source**, 0, sizeof(**source**));
• memset(**dest**, 0, sizeof(**dest**));
• }
• int **rev** = strlen(temp);
• if (rev)
• {
•     int **ch** = 0;
•     for (int **i** = rev; i < 8; i++)          // 后面应该补8-rev个'0'
•         temp[i] = '0';
•     for (int **i** = 0; i < 8; i++)

```

```

    •    if (i)

    •        ch = ch * 2 + (temp[i] - '0');

    •    else

    •        ch = ch * 2 - (temp[i] - '0');

    •    dest[0] = (char)ch;

    •    fwrite(dest, sizeof(char), 1, **fptrD**);

    }

    fputc((char)rev, **fptrD**);          // 存储最后数位剩余的信息，多占一个字节

    string **d**=destfile;

    ui->textBrowser->append(QString::fromStdString( "上传成功！压缩文件为"+d));

    fclose(**fptrS**);

    fclose(**fptrD**);

    return true;

}

```

```

// 解码

bool homework1:: Decode(char* sourcefile, char* destfile)

{

    FILE* fptrS = fopen(sourcefile, "rb");

    FILE* fptrD = fopen(destfile, "wb");

    if (fptrS == NULL || fptrD == NULL)

    {

    •    ui->textBrowser->append(QString::fromStdString( "文件名出现错误，解码失败！"));

    •    if (fptrS)

    •        fclose(fptrS);

    •    if (fptrD)

    •        fclose(fptrD);

    •    return false;

    }

}

```



```

}

memset(source, 0, sizeof(source));

memset(temp, 0, sizeof(temp));

memset(dest, 0, sizeof(dest));

fseek(fptrs, -1, SEEK_END);    // 定位到辅助信息处, 倒数顺序: EOF 辅助信息 存储的最后一位(可能只有部分有用)

long long count = ftell(fptrs); // count为有用信息的字节数(最后几个bit可能为无效信息)

int rev = (int)fgetc(fptrs); // 最后一个字节的有效比特位数

fseek(fptrs, 0, SEEK_SET);

fread(&H, sizeof(HuffmanTree), 1, fptrs);

count -= ftell(fptrs);

while (count)                // 若feof(fptr)则count=0
{
    int size = count > READSIZE ? READSIZE : count;

    fread(source, sizeof(char), size, fptrs);

    for (int i = 0; i < size; i++)
    {
        int ch = (unsigned char)source[i];

        char copy[10];

        memset(copy, 0, sizeof copy);

        for (int j = 7; j >= 0; j--)        //最先输出低位,低位在高地址
        {
            copy[j] = '0' + (ch % 2);

            ch = ch / 2;
        }

        strcat(temp, copy);
    }

    int templen = strlen(temp), destlen = 0;

```

```

•   int index = 0;

•   while (templen - index >= CODESIZE)

•   {

•       int root = H.len - 1;

•       while (H.node[root].lchild != -1 && H.node[root].rchild != -1)

•       {

•           if (temp[index] == '0')

•               root = H.node[root].lchild;

•           else if (temp[index] == '1')

•               root = H.node[root].rchild;

•           index++;

•       }

•       dest[destlen++] = H.node[root].data;

•   }

•   fwrite(dest, sizeof(char), destlen, fptrD);

•   count -= size;

•   char copy[CODESIZE];

•   for (int i = 0; i < templen - index; i++) // 不足CODESIZE位的留着下次循环再找

•       copy[i] = temp[index + i];

•   memset(temp, 0, sizeof(temp));

•   for (int i = 0; i < templen - index; i++)

•       temp[i] = copy[i];

•   memset(source, 0, sizeof(source));

•   memset(dest, 0, sizeof(dest));

•   }

int templen = strlen(temp) - (8 - rev); // 8-rev位是人工添加的‘0’，是无效信息

int index = 0, destlen = 0;

```

```

while (index < templen)

{

    •   int root = H.len - 1;

    •   while (H.node[root].lchild != -1 && H.node[root].rchild != -1)

    •   {

    •       if (temp[index] == '0')

    •           root = H.node[root].lchild;

    •       else if (temp[index] == '1')

    •           root = H.node[root].rchild;

    •       index++;

    •   }

    •   dest[destlen++] = H.node[root].data;

}

fwrite(dest, sizeof(char), destlen, fptrD);

string d=destfile;

ui->textBrowser->append(QString::fromStdString( "下载成功! 解压文件为"+d));

fclose(fptrS);

fclose(fptrD);

return true;

}

```

```

// 清除缓冲区

static void ClearBuffer()

{

    while (getchar() != 'n');

}

```

### (三) 算法特点

有效的减小了文件体积

算法时空分析

设输入文件大小为n

Init o(n)

CreateHuffmanTree o(1)

//新构建的非叶子结点最多有255个

SearchNode o(1) //结点最多有511个，树的高度最大也只能是256，搜索次数也为常数级

CreatHaffmanCodeMap o(1)

Encode o(n)

Decode o(n)

PrintMessage o(1)

ClearBuffer o(m) //m为用户在命令行键入的字符数

### (四)模块调用

在运行课程作业资料模块时,涉及到对资料上传和下载时使用该模块.

## 4)时间线程模块

### (一) 算法思想

信号与槽是qt的基础，也是 Qt 的一大创新。因为有了信号与槽的编程机制，在 Qt 中处理界面各个组件的交互操作时变得更加直观和简单。

由于qt不允许在子线程对ui进行修改,所以在这里创建thread句柄来计算时间,再通过qt的connect信号槽实现在ui上的打印

### (二) 算法详解

```
//触发信号

while(true){

    extern string time1;

    qstr2 = QString::fromStdString(time1);

    pauseLock.lock();//线程锁控制停止

    emit finishs();

    •    if(way=='e') break;

    •    if (way == 's') { //改变速度

    •        sleep(10 * 1000);
```

```

•   }

•   if (way == '2') {

•       sleep(2 * 1000);

•   }

•   if (way == '3') {

•       sleep(500);

•   }

•   if (way == 's' || way == '2' || way == '3') {

•       hour = hour + 1;

•       if (hour % 24 == 0) {

•           hour = 0;

•           week = (week + 1) % 7;

•       }

•       string day;

•       if (week == 0) day = "7";

•       if (week == 1) day = "1";

•       if (week == 2) day = "2";

•       if (week == 3) day = "3";

•       if (week == 4) day = "4";

•       if (week == 5) day = "5";

•       if (week == 6) day = "6";

•       ostringstream oss;//打印为字符串

•       oss<< "星期" << day<<"_" << hour << ": 00" ;

•       time1 = oss.str();

•       pauseLock.unlock();

```

建立信号槽:

```
connect(&t,SIGNAL(finishs()),this,SLOT(myPrint()),Qt::QueuedConnection);
```

控制开始或停止:

```
暂停或继续
void user::on_comboBox_activated(int index)
{
    if(t.lockk==1 && index==0) {t.pauseLock.lock();t.lockk=0 ;} //控制线程锁
    if(t.lockk==0 && index==1) {t.pauseLock.unlock();t.lockk=1;}
    cout<<"okk"<<endl;
}
```

### (三) 算法特点

规避了qt无法在非主线程对ui进行修改的不足

信号与槽机制是 Qt GUI 编程的基础，使用信号与槽机制可以比较容易地将信号与响应代码关联起来

### (四) 模块调用

进入使用者模块时,点击时钟开始,该模块被启用,点击暂停后停止

## 5) 初始化日志模块

### (一) 算法思想

将txt文本中的值放到数组中,若要修改就对数组进行修改,修改后的数组再通过输入输出流对txt进行修改.

### (二) 算法详解

写入数组(以活动数组为例):

```
while (ifs >> act[count6].acttime) {

    •   act[count6].time = act[count6].acttime;

    •   ifs >> b;

    •   act[count6].kind = b;

    •   ifs >> act[count6].actname;

    •   act[count6].name = act[count6].actname;

    •   count6++;
```

写回文本(以课程为例):

```

for (int count1 = 0; count1 < count5; count1++) {
    ofs<<clas[count1].name<<endl;
    ofs << clas[count1].time << endl;
    ofs << clas[count1].campus << endl;
    ofs << clas[count1].buliding << endl;
    ofs << clas[count1].classroom << endl;
    ofs << clas[count1].timeofexam << endl;
    ofs << clas[count1].buildingofexam << endl;
    ofs << clas[count1].classroomofexam << endl;
    ofs << clas[count1].stage << endl;
    ofs << clas[count1].group << endl;
} //写入更新后数组

```

### (三) 算法特点

便于操作,直接对文本操作和查找过于复杂

### (四)模块调用

在程序开始时就行进行初始化,把日志里的数据全都还原到数组里,修改以后录入即可.

在运行课程查询导航,作业资料,闹钟,活动,注册和管理员控制用户名单和修改课程模块时,涉及到对数据修改时调用该模块算法.

## 6)文件读取目录压缩模块

### (一)算法思想

用fileDialog来打开文件对话框

通过的QZipWriter的addFile来实现对目录的压缩

### (二) 算法详解

文件选择:

```

//定义文件对话框类
QFileDialog *fileDialog = new QFileDialog(this);

//定义文件对话框标题
fileDialog->setWindowTitle(QStringLiteral("选择文件"));

//设置打开的文件路径
fileDialog->setDirectory("./");

fileDialog->setNameFilter(tr("File(*.doc* *.bmp* *.zip*)"));

//设置可以选择多个文件,默认为只能选择一个文件QFileDialog::ExistingFiles
fileDialog->setFileMode(QFileDialog::ExistingFiles);

//设置视图模式
fileDialog->setViewMode(QFileDialog::Detail);

```

```

//获取选择的文件的路径
QStringList fileNames;
if (fileDialog->exec()) {
    fileNames = fileDialog->selectedFiles();
}

QString s=fileNames.join("");//拼凑出地址

```

目录压缩:

压缩:

```

QDir dir(tmpPath);//读目录下的文件
foreach (QFileInfo info, dir.entryInfoList())//遍历文件
{
    if (info.fileName() == "." || info.fileName() == "..")
        continue;
    if (info.isFile())
    {
        QFile upfile(info.filePath());
        upfile.open(QIODevice::ReadOnly);
        QString fileName =
info.filePath().mid(basePath.size()+1,info.filePath().size()); //取文件名
        writer->addFile(fileName,upfile.readAll()); //把文件加入到zip写入流
        qDebug()<<fileName<<tmpPath<<basePath;
        upfile.close();
    }
    else if (info.isDir())//如果还是文件则继续往下遍历
    {
        QZipWriterEx(writer,info.filePath(),basePath);
    }
}
return true;
}

```

选择压缩对象和储存位置:

```

bool ret;
QZipWriter *writer = new QZipWriter(SavePath);//选择保存位置
if(QZipWriterEx(writer,dirPath,dirPath))//dirPath为了方便回调，所以传了两次。
    ret=true;
else
    ret=false;

writer->close();
delete writer;

```



### (三)算法特点

选择较为方便,通过对多文件压缩可以实现更好上传更多的资料

### (四)模块调用

在对资料或作业进行上传或更新时使用

## 6)范例执行结果及测试情况说明

见用户使用手册

## 7)评价和改进建议

**本次数据结构课设，根据需求书的需求，我实现了所有的基础功能和ui设计以及bmp和目录压缩**

主要采用的算法是Dijkstra算法，辅助以QT图形库的相关函数。整个系统的开发和设计主要经历了下列几个时间点：

第5周：详细进行需求分析

第6周：校区路线图的构建

第7周：设计程序整体架构，将各模块的框架明确

第8周：设计各模块内函数的基本框架并明确一部分函数之间的调用情况以及各变量的设计及调用

第9周：各模块函数的具体代码实现，并测试各模块运行情况

第10周：继续完善各模块函数的具体实现，分析可能存在的可用但存在风险的操作

第11周：着手QT的编写工作

第12周：完善QT用户界面的交互性与稳定性及初步调试，根据用户初步反馈调整各主要位置存在的不足之处

第14周：终版报告书写，并做最终测试，提交最终版报告书以及源代码

### 感想：

1. 刚开始的想法是觉得qt也是cpp写的,所以就一开始没考虑qt,直接写cpp版本后期转成qt,后来发现有点多此一举,因为qt和设计思路和cpp很多地方都不一样,有些自认很复杂的工作在qt化时发现是白做的(比如退出按钮啥的),有些则需要大范围修改(比如QString和string的转化),最让我痛苦的就是多线程的问题,qt无法在子线程中修改ui,所以学了信号槽,但在做信号槽的时候遇到很多离谱的错误,花了很多的时间.万幸写的各种查找算法还可以用,大概花了一个多星期完成了整个图形化的界面.
2. 这次作业涉及到算法也都不算很难,难得kmp和迪杰斯特拉和哈夫曼编码在上学期已经接触过了,难的是如何组合起来,这次最大的收获可能就是从一个设计者的角度来想怎么让用户用的舒服,而不是像oj一样单纯判断正误,因为测试用例总是对的,而你永远不知道用户会输入什么特别离谱数据.但我也只是有限的做了一些测试,不能保证没有bug(毕竟连bat也不敢包票一个程序开始没有bug),但收获是实打实的.

## 分工:

倪玮昊,刘义红,缪奇志,共同完成了界面设计,程序开发和文档撰写工作,三人付出相同.