北京工业大学

2023- 2024 学年 第 1 学期

信息学部 计算机科学与技术（实验班）

课程名称：	数据结构课程设计		
报告性质：	课程设计报告		
学号：	21071029	姓名：	王宇飞
任课教师：	杜永萍	课程性质：	必 修
学分：	2.0	学时：	60
班级：	210710	成绩：	
教师评语：			

年 月 日

目录

1.需求分析

1.1 问题描述

以某计算机学院专业课及专业选修课为背景，实现课程计划的辅助编排。学生在一个学期可以同时选学多门课程，同一学期内的各门课程之间必须不能存在“先修课”与“后续课”的次序关系。通过辅助编排系统制定课程计划，使学生可以按照规划完成课程的学习任务。

构造课程的有向无环图，弧表示课程之间必须遵循的优先次序关系。通过扩展已有的拓扑排序算法，进行课程计划的辅助制订，将所列课程划分为子集（不同学期的课程计划），使任意两门有次序关系的课程分属于不同的子集，每个子集中包含的顶点对应着同一学期开设的课程，称这种扩展拓扑排序的划分结果为“拓扑子集的划分”，其结果可以辅助完成教学计划的编排和选课。

1.2 基本要求

调研有关计算机专业应该开设的课程，以此建立构造课程的有向无环图，求解拓扑子集划分的参考解。

可以对拓扑子集划分的结果进行调整。一般情况下，直接的课程编排结果可能过于“密集”（某学期内总课时过多，使学生的负担难以承受），有必要对所求的课程与学期的编排表进行人为的调整，即适度拉长修业的时间，得到既满足课程之间的次序关系，又不使学生负担过重。最终形成满意的学期与课程编排表，以供参考。建立有向无环图以及课程编排结果应当配以图形界面；子集划分结果的人工调整也应提供友好的界面支持。

直接求得的拓扑子集划分方案，每学期课时量总和往往大于学校要求的学期课时数，这就需要把某些课程调整到后续学期。某门课向后调整，有可能引出新的制约冲突，需要再次启动划分算法得出合情理的划分方案。

原始数据的输入可以通过适当的界面，输入有向弧的顶点对的办法来完成。实际的数据应能以文件的形式存储，以备再次启动本辅助编排系统时导入。

1.3 开发环境

Qt Creator 11.0.2 + Visual Studio 2019

1.4 需要处理的数据

项目名称（QString）、课程总数（int）、课程序号（int）、课程名称（QString）、先修课程（QString）、课程学分（int）、课程名称与课程序号的对应关系（QMap<QString, int>）、课程名称与学分的对应关系（QMap<QString, int>）、课程节点（Node）、课程的邻接表（List）、课程之间的关系（Edge）、所有课程构成的有向无环图（Graphl）、文件（QFile）、信号（SIGNAL）、槽函数（SLOT）、Qt 控件等。

1.5 用户界面的设计

该程序主要借助了 Qt 中的 ui 以及控件库完成了用户界面的设计，所包含的主要控件有：QTabWidget（容器控件）、QMessageBox（消息提示框）、QTableWidget（表格）、QLabel

（标签）、QPushButton（常规按钮）、QGridLayout（格栅布局）、QHBoxLayout（水平布局）、QCheckBox（复选框）、QListWidget（列表框）、QLineEdit（单行文本输入框）、Spacer（间距器）、QScrollArea（自动滚动区）、QComboBox（下拉框）、QDoubleSpinBox（浮点计数器）、QSlider（滑动条）、QMenu（菜单）、QMenuBar（菜单栏）、QStatusBar（状态栏）等。

部分用户界面展示如下：

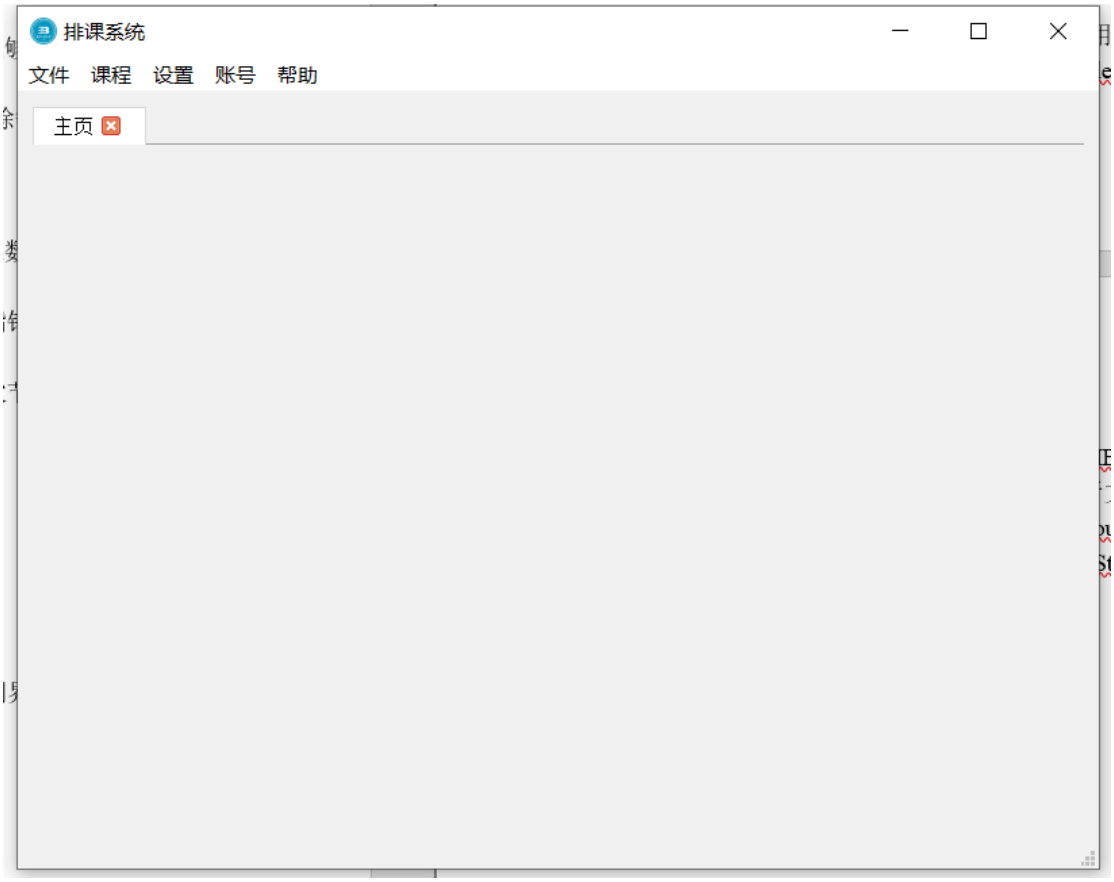

图 1.1 初始界面

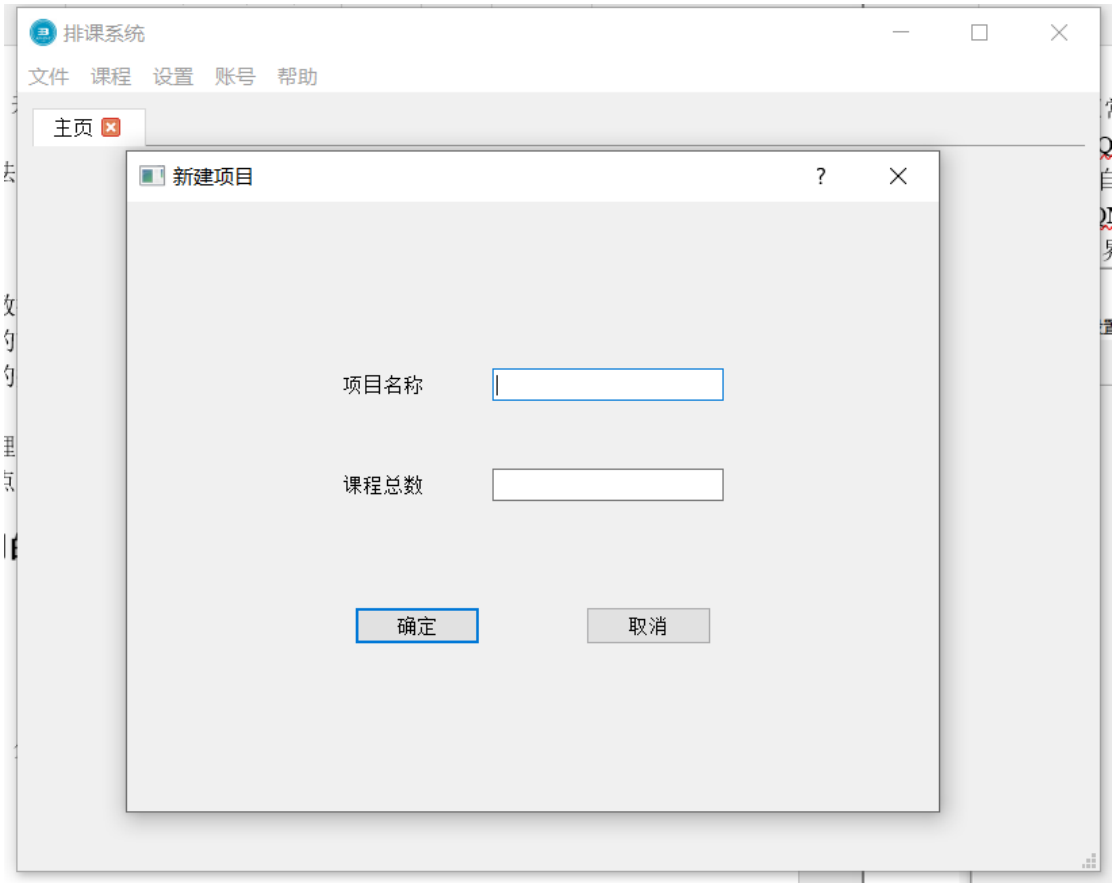
排课系统
文件 课程 设置 账号 帮助
主页
新建项目
项目名称
课程总数
确定
取消

图 1.2 新建项目

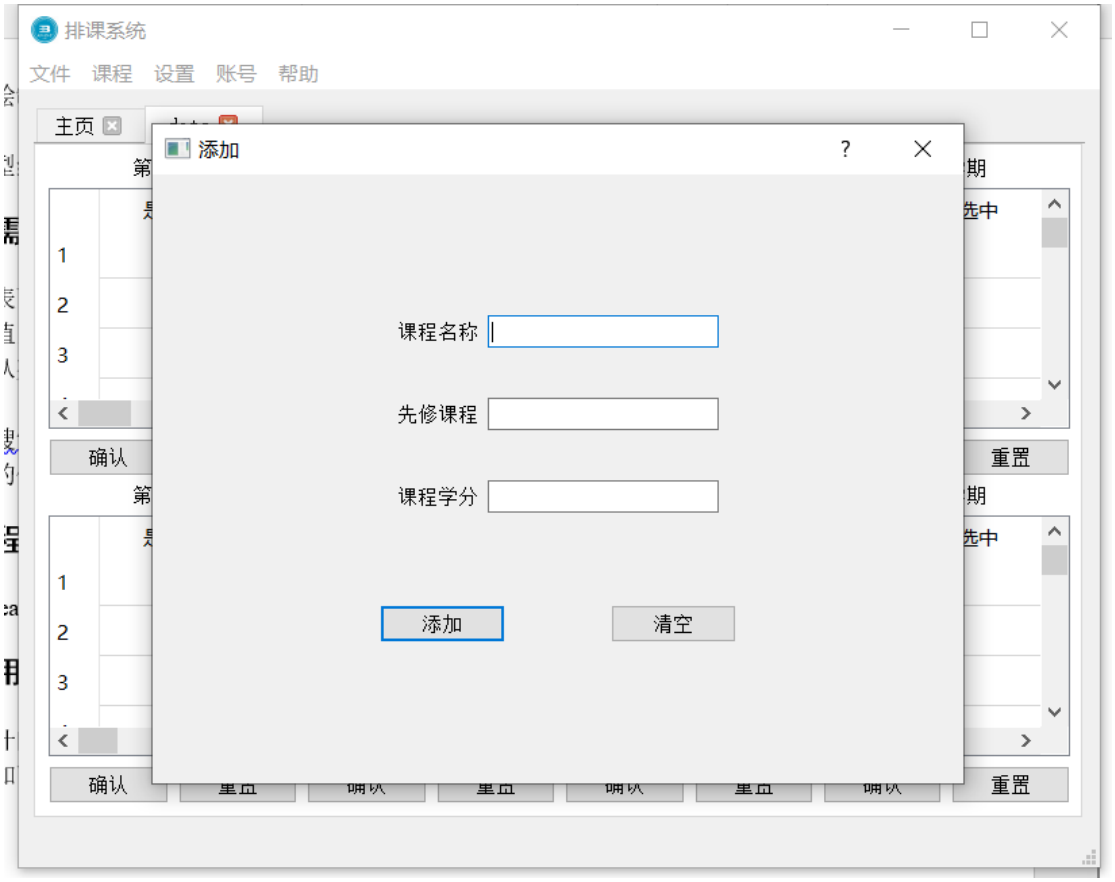
排课系统
文件 课程 设置 账号 帮助
主页
添加
课程名称
先修课程
课程学分
添加
清空
确认
重置

图 1.3 添加课程

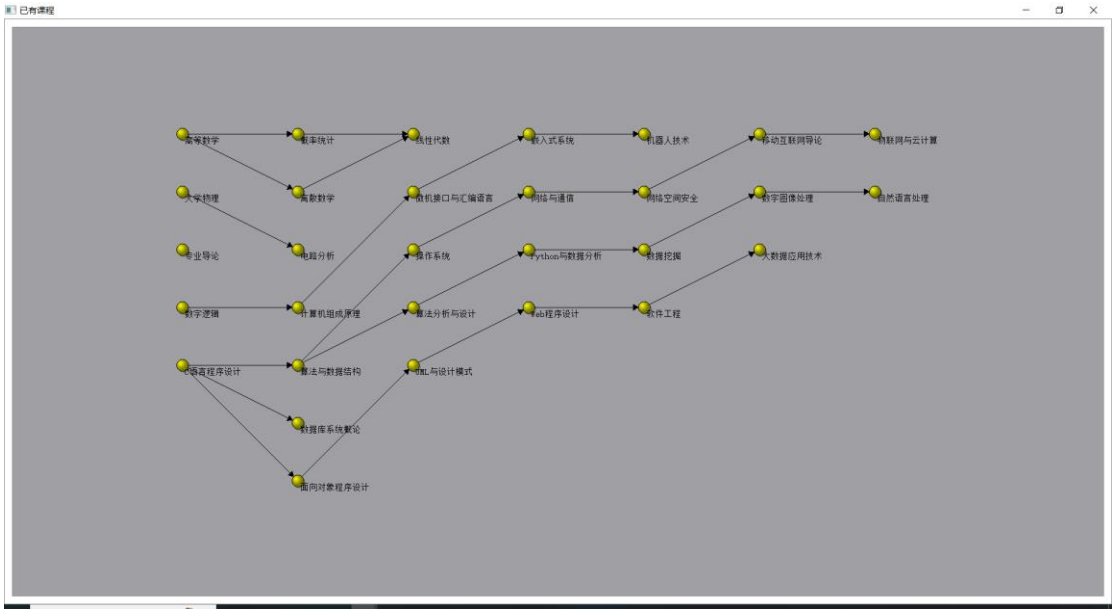

图 1.4 已有课程

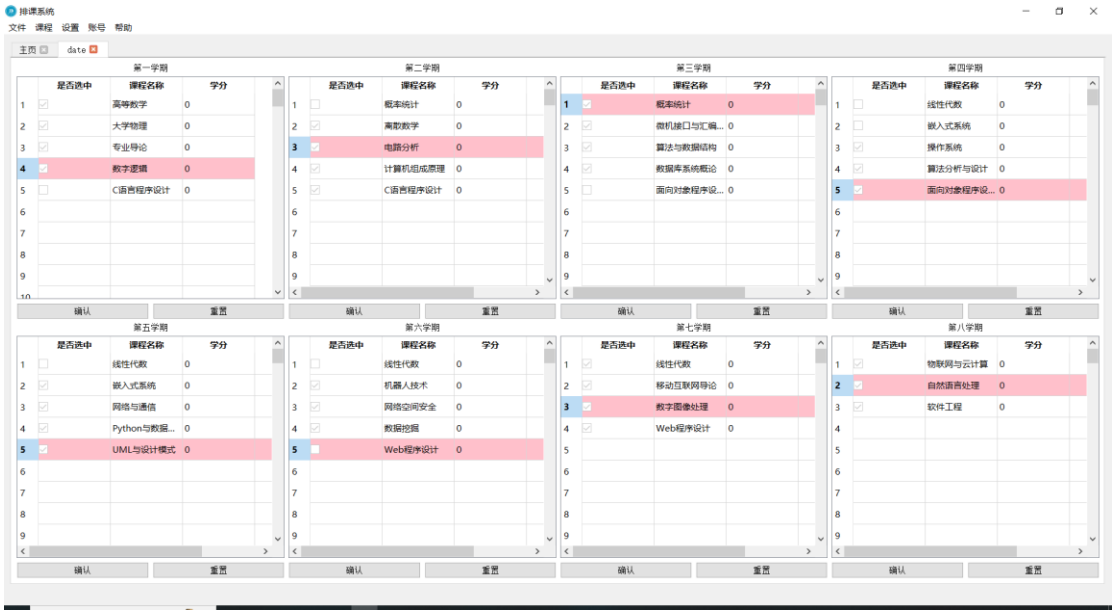

图 1.5 选课

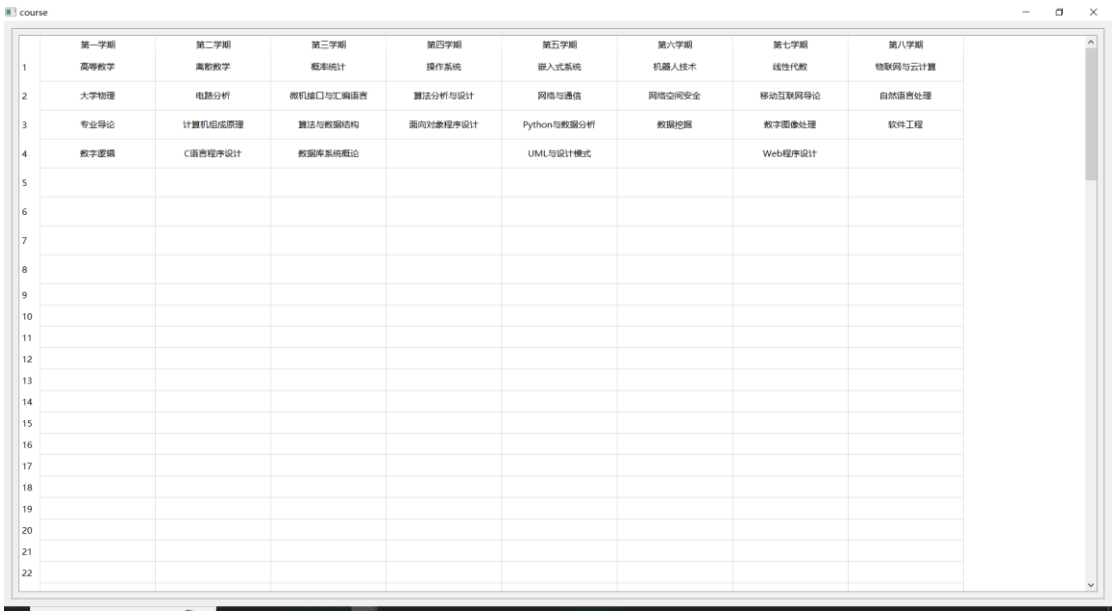

图 1.6 生成课表

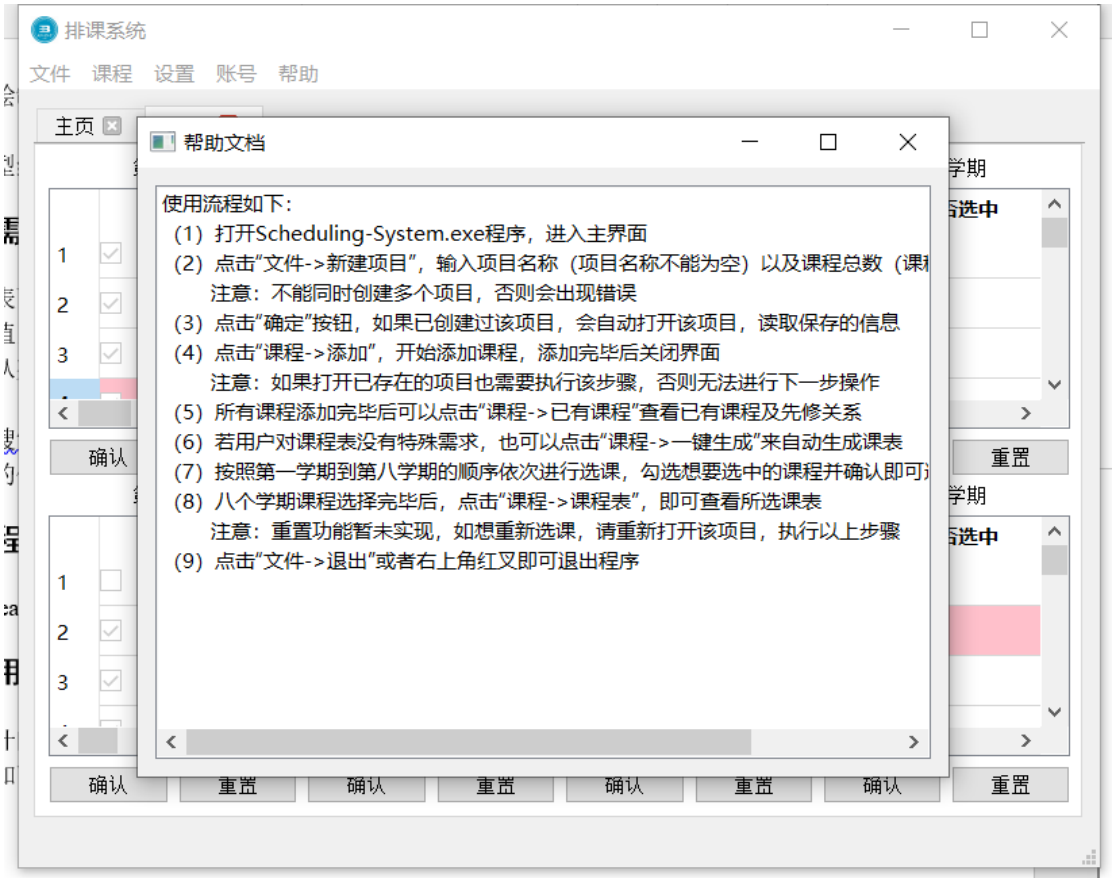

图 1.7 帮助文档

2.数据结构设计

2.1 整体结构及各模块功能描述

2.1.1 整体结构

该程序所包含的代码文件如下所示：

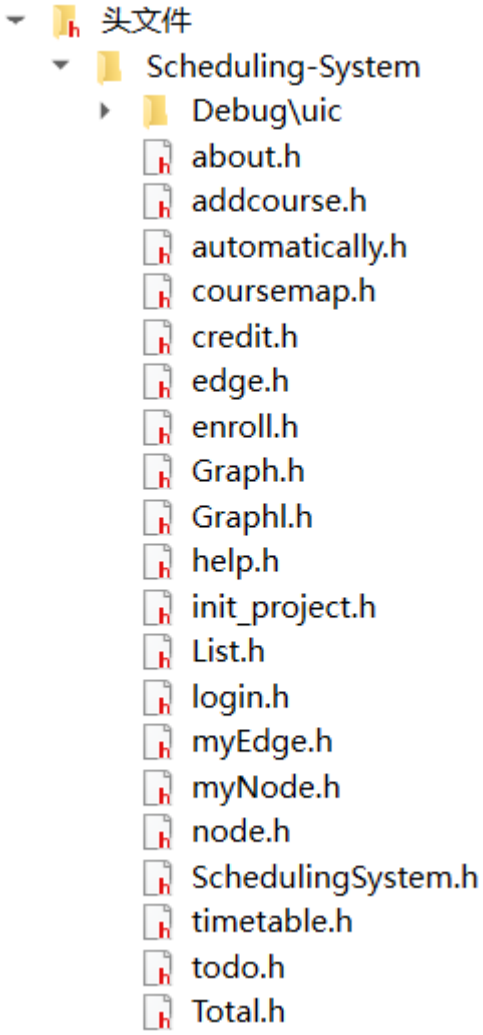

图 2.1 头文件

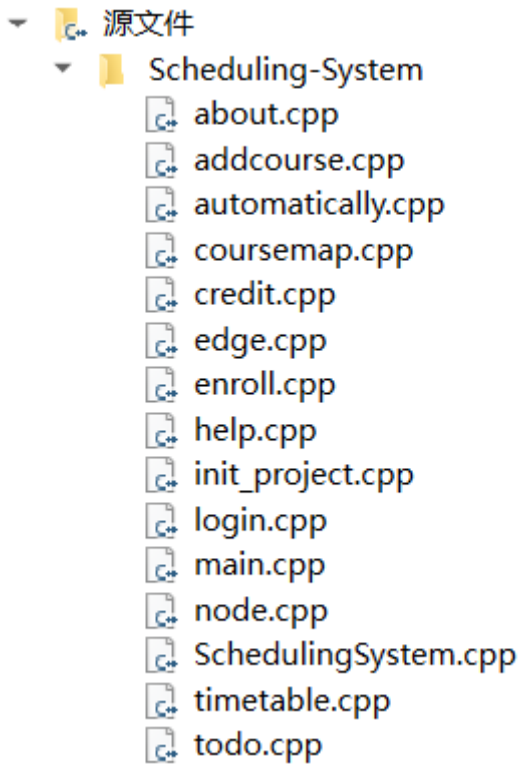

图 2.2 源文件

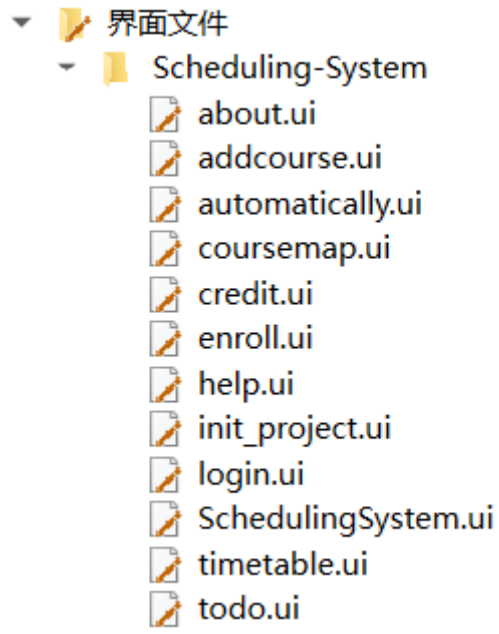

图 2.3 界面文件

2.1.2 模块功能描述

（1）头文件模块

Total.h 为主要数据结构中所使用库的汇总，使用 Total.h 后可不必在不同的文件中反复添加所包含的库。

（2）核心数据结构及算法模块

myNode.h 为课程节点的定义，包含了 myNode 类的成员变量、构造方法、重要函数，

用于表示课程节点的相关信息；

myEdge.h 为有向边的定义，包含了 myEdge 类的成员变量、构造方法、重要函数，用于表示两个 myNode 类之间的前驱后继关系；

List.h 为邻接表的定义，包含了 List 类的成员变量、构造方法、重要函数，实际上是以 Node 类为基础元素的链表，用于存储与某个课程节点相关的所有边，并按照后继节点的序号递增排列；

Graph.h 为基础图的定义，包含了 Graph 类的成员变量、构造方法、重要函数，该类包含了 myNode、myEdge、List 三个类，用于存储课程节点以及前驱后继关系等信息，是 Graphl 类的基础；

Graphl .h 为带有邻接表的图的定义，是 Graph 的子类，该类借助 myNode、myEdge、List 三个类以及 Graph 中的基础结构与方法较为完整地表示所有课程的信息及相互关系，是 Graph 类的延伸。

（3）有向无环图绘制模块

node、edge（.h、.cpp）分别用来绘制有向无环图中的节点和边，来源于 Qt 的示例；

coursemap（.h、.cpp、.ui）为“已有课程”界面的定义，该类借助 node 和 edge 两个类，用于绘制所有课程及其相关关系的有向无环图。

（4）界面显示及文件操作模块：

about（.h、.cpp、.ui）为“关于”界面的定义，用于展示该程序的相关信息；

addcourse（.h、.cpp、.ui）为“添加”界面的定义，用于读入用户输入的课程名称、先修课程、学分等信息并写入相应文件；

automatically（.h、.cpp、.ui）为“一键生成”界面的定义，用于从文件中读取课程信息来一键生成课表；

credit（.h、.cpp、.ui）为“学分”界面的定义，用于设置各学期的学分限制；

enroll（.h、.cpp、.ui）为“注册”界面的定义，用于用户注册账号；

help（.h、.cpp、.ui）为“帮助文档”界面的定义，用于为用户提供帮助信息；

init_project（.h、.cpp、.ui）为“新建项目”界面的定义，用于创建新的项目，并对该项目创建独立文件夹，保存相关信息；

login（.h、.cpp、.ui）为“登录”界面的定义，用于用户登录账号；

timetable（.h、.cpp、.ui）为“course”界面的定义，用于读取已选课程并生成课程表；

todo（.h、.cpp、.ui）为“todo”界面的定义，用于显示待优化界面。

（5）主体模块

SchedulingSystem（.h、.cpp）为主界面的定义，包含了以上所有的类，并且定义了大量的信号（SIGNAL）和槽函数（SLOT），用于响应用户的动作以及创建其他窗口，是整个程序的主体文件；

main.cpp 其中包含 main 主函数，该函数中调用了 SchedulingSystem 类，是程序执行的起点。

2.2 主要数据结构

myNode：课程节点，记录了课程的序号、名称、学分、以及 next 指针等信息；

myEdge：课程的前驱后继关系，逻辑上两个 myNode 课程节点构成一条边，记录了前驱以及后继节点的序号；

List：邻接表，某一节点的 List 的 head 指针一定指向该节点，head 的后续节点都为该节点的后继课程，用于整体记录节点的前驱后继关系，该类提供了插入边的函数；

Graph：基础图，包含课程总数、访问标记、添加标记、入度、出度等信息，提供了获

取顶点个数、判断 myEdge 是否合规等函数；

Graphl：带有邻接表的图，在 Graph 的基础上增加了 List 用于记录节点间的前驱后继关系，并且提供了返回某节点的第一条边（第一个前驱后继关系）、返回下一条边、判断给定的边是否为图中的前驱后继关系、设置边、删除指定边、删除入度为 0 的节点及其所有边、进行特定拓扑排序等函数；

node：用于绘制有向无环图中的节点；

edge：用于绘制有向无环图中的有向边。

3.详细设计

3.1 主要函数流程图

3.1.1 List.h

（1）void insert(myEdge e)

该函数用于向 List 链表中插入一条边 e，插入时会按照 e 边的指向节点（也就是后继课程）的序号进行排序，确保序号递增，这样在操作时能确保每一次同样的操作能得到固定的结果。

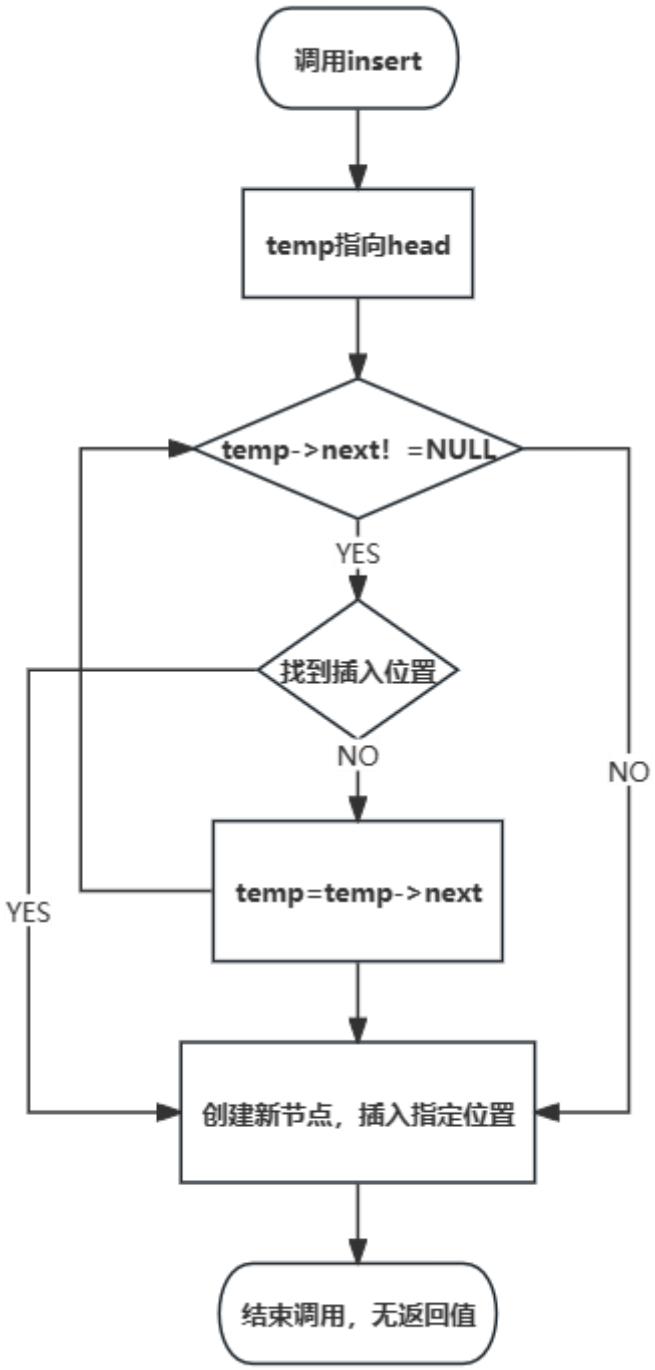

图 3.1 insert 函数流程图

3.1.2 Graph.h

（1）bool isEdge(myEdge e)

传入一个边，判断是否符合要求（课程节点的 index 必须为正整数，且不能大于课程总数）。

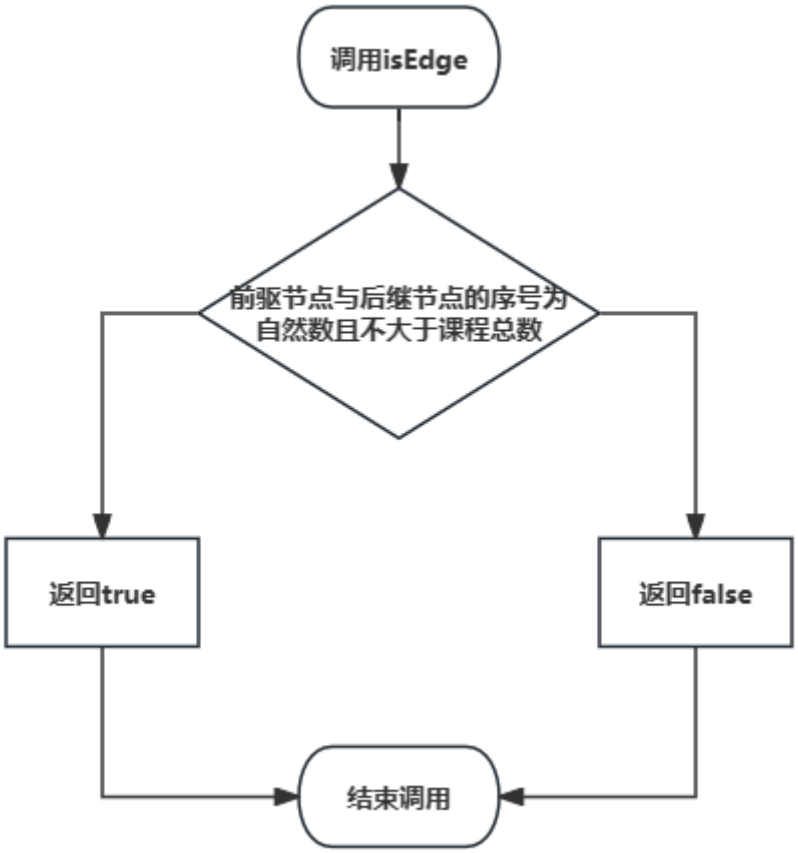

图 3.2 isEdge 函数流程图

3.1.3 Graphl.h

（1）myEdge nextEdge(myEdge e)

传入的参数为一条边（课程前驱后继关系中的一个），返回下一条边（该课程节点与下一个后继节点所构成的前驱后继关系）。

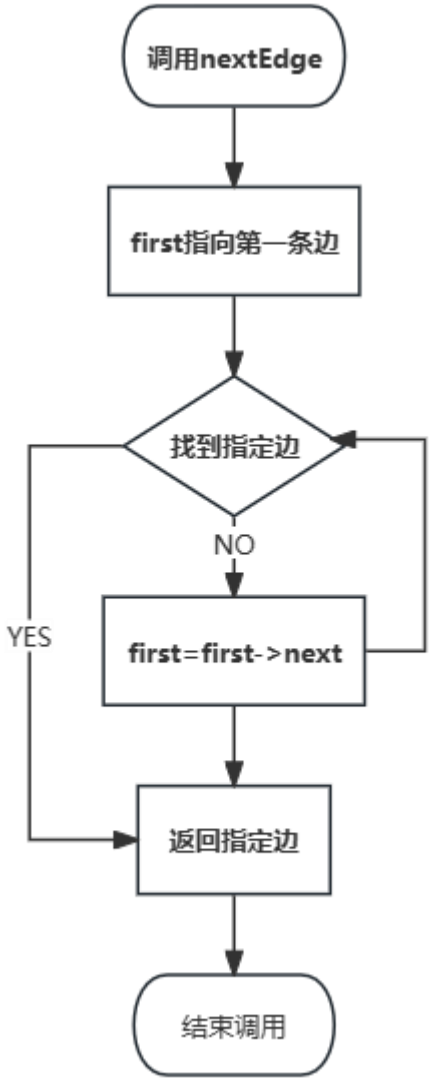

图 3.3 nextEdge 函数流程图

（2）bool isMyEdge(myEdge e)

传入的参数为一条指定边（改边可以是任意的，不做特殊要求），该函数用于判断传入边是否在该 Graphl 中，首先调用 isEdge（）来判断该边是否合规，在循环查找该边，若能在该 Graphl 中查找到该边，则返回 true，否则返回 false。

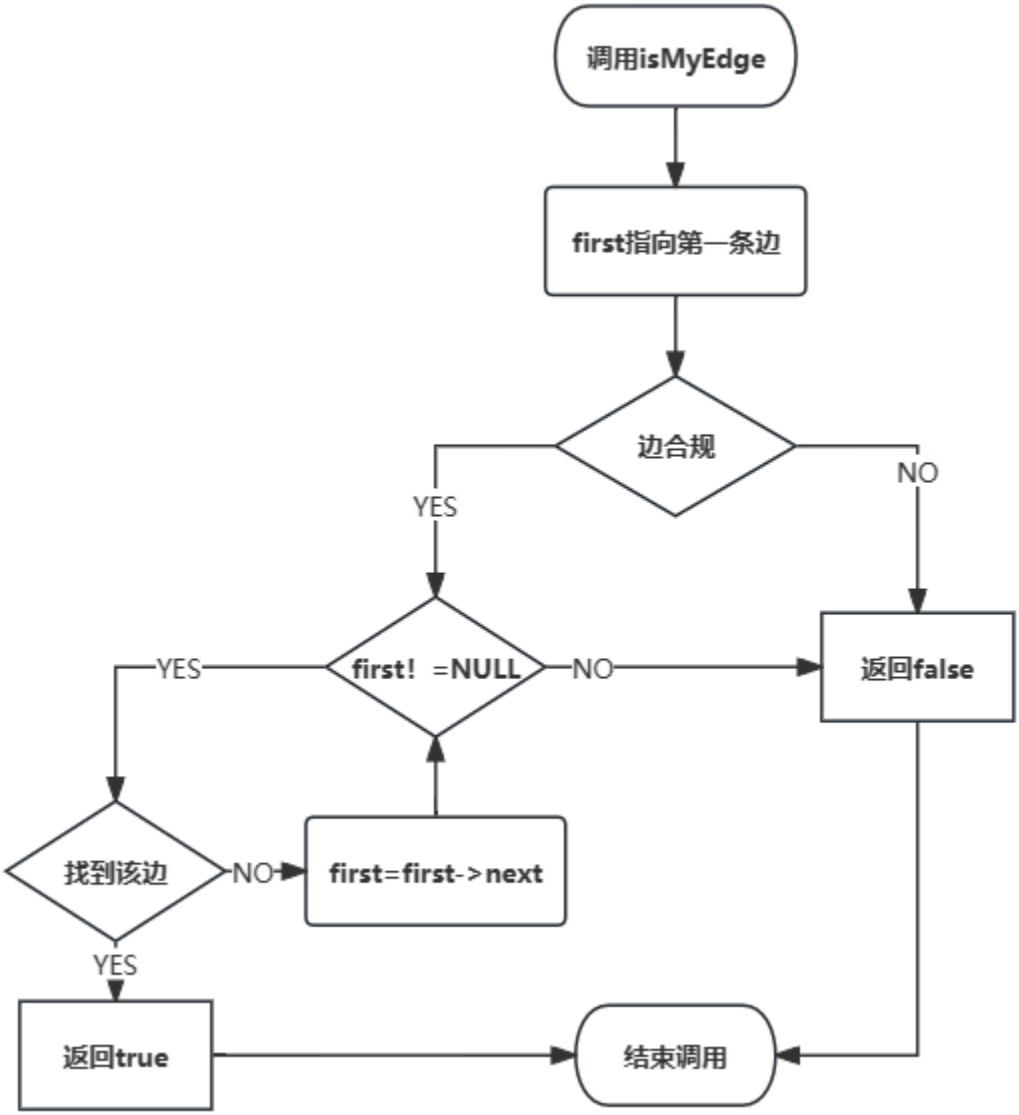

图 3.4 isMyEdge 函数流程图

（3）bool setEdge(int f, int t)

传入的参数为前驱课程的序号以及后继课程的序号，该函数用于向 Graphl 中添加一条边，返回值为 bool 类型，用于判断是否创建成功，如果该边不符合基本限制，则返回 false，反之返回 true。

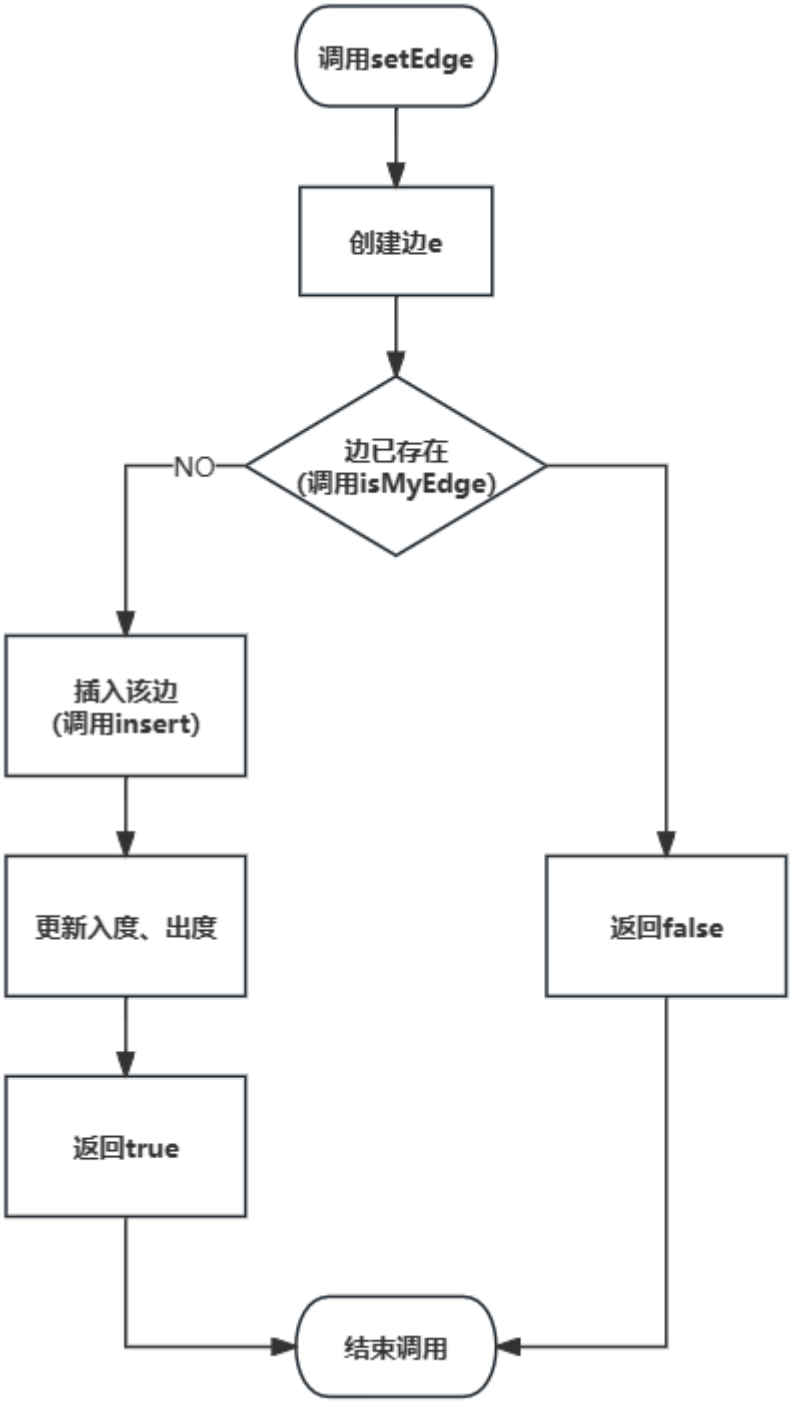

图 3.5 setEdge 函数流程图

（4）bool delEdge(int f, int t)

传入的参数为前驱课程的序号以及后继课程的序号，该函数用于删除 Graphl 中的一条边，返回值为 bool 类型，如果找到该边，删除该边并修改 inDegree 以及 outDegree 数组的值，然后返回 true，反之不执行操作返回 false。

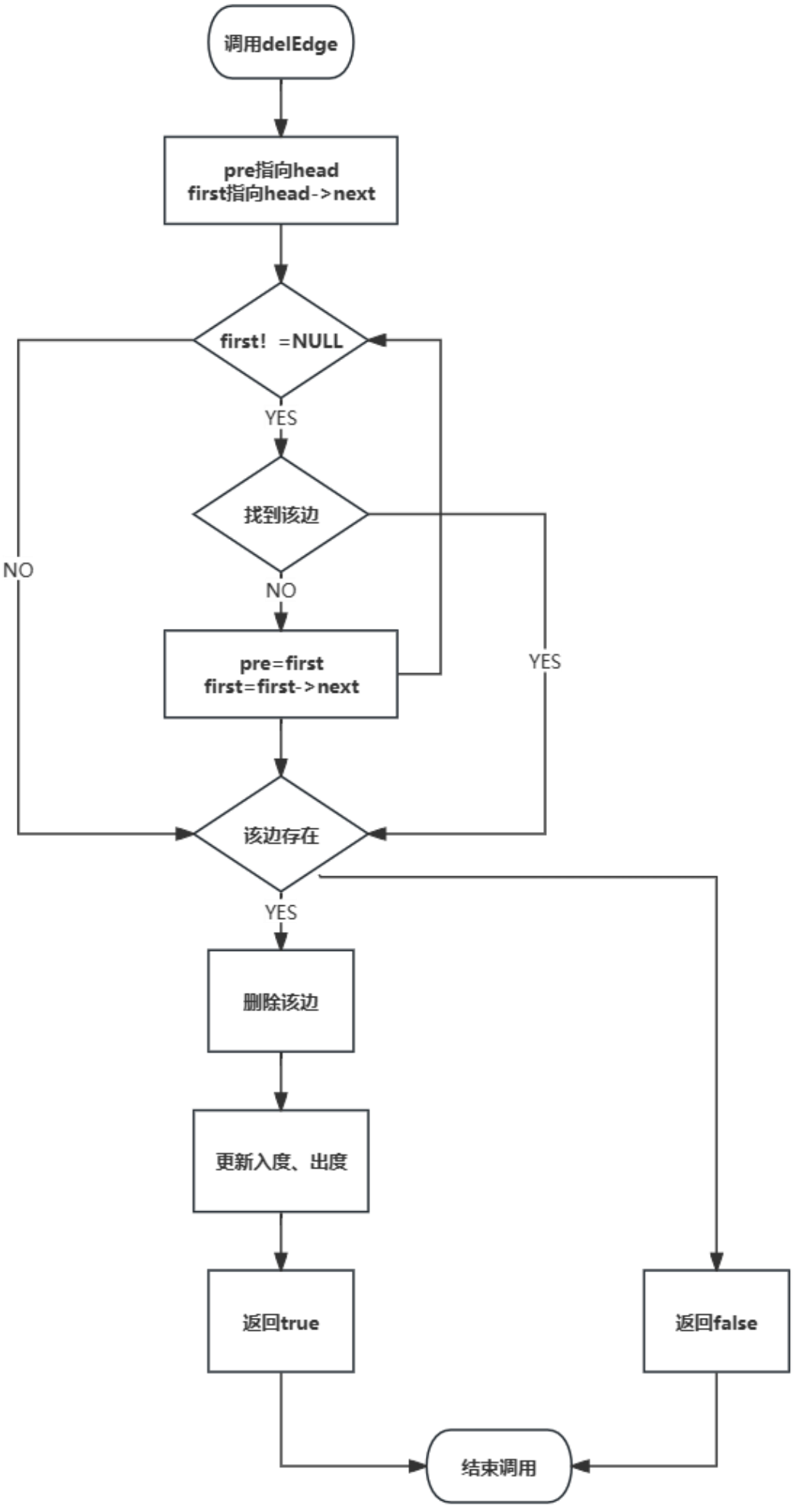
调用delEdge
pre指向head
first指向head->next
first! =NULL
YES
找到该边
NO
NO
pre=first
first=first->next
YES
该边存在
YES
删除该边
更新入度、出度
返回true
返回false
结束调用

图 3.6 delEdge 函数流程图

（5）bool topologySort(QString projectName, int i)

该函数用于执行一轮拓扑排序操作，此处定义的一轮是指将当前 Graphl 中入度为 0 的课程节点全部删除，然后将其写入相应文件，方便后续读取以及查找，返回值为 bool，用于检查该 Graphl 是否包含回路，若在函数执行过程中，发现已经没有入度为 0 的节点且还有剩余的课程节点，则说明包含回路，返回 false，反之返回 true。

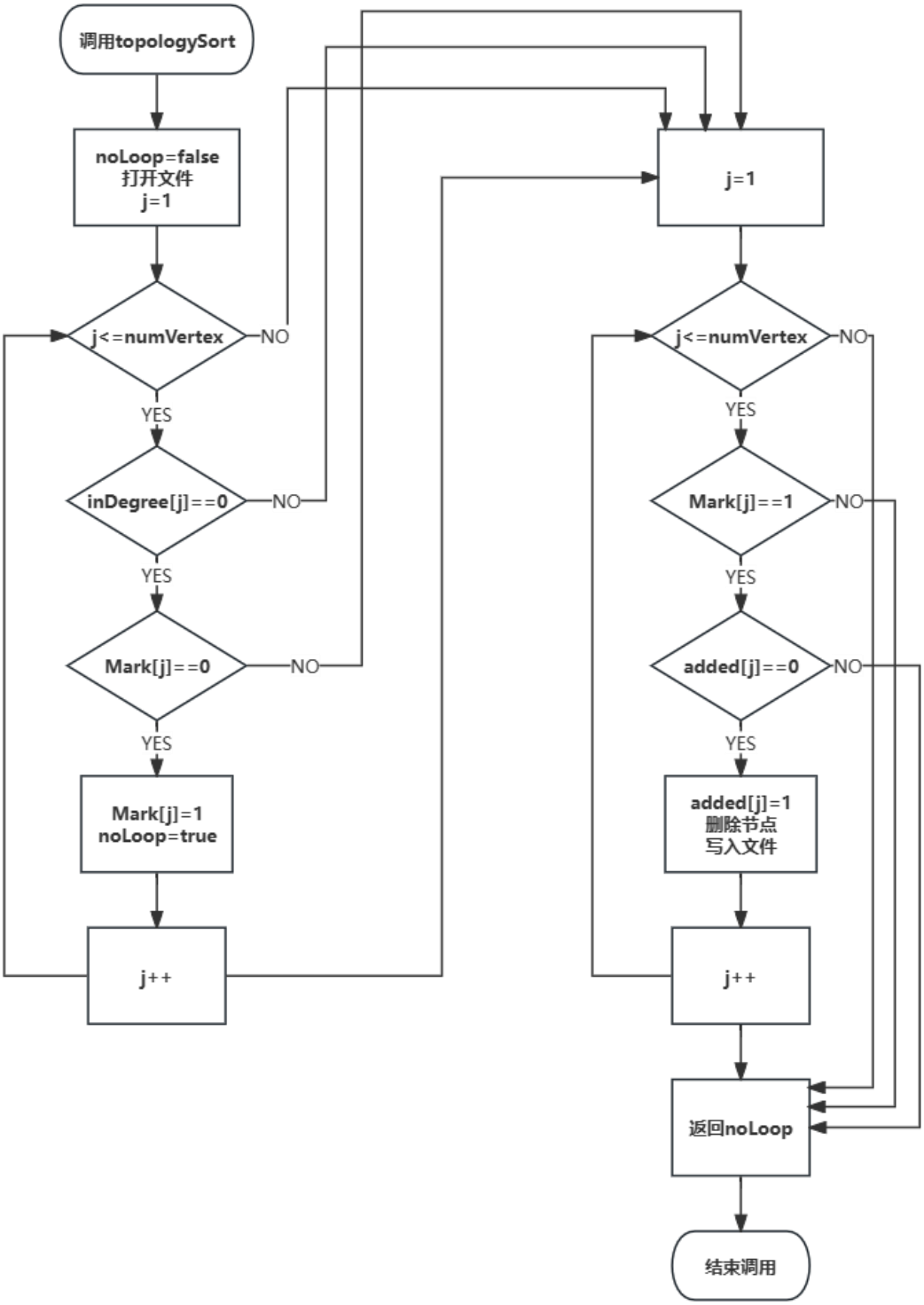

图 3.7 topologySort 函数流程图

3.2 数据结构说明

3.2.1 myNode（课程节点）

代码如下所示：

```
class myNode {
public:
    int index = 0; // 点的索引值
    QString name = ""; // 课程名称
    int weight = 0; // 学分
    myNode* next = NULL; //指向下一条边
    //构造函数
    myNode(int d, myNode* next) {
        index = d;
        this->next = next;
    }

    myNode(int d, int w, QString n, myNode* next) {
        index = d;
        weight = w;
        name = n;
        this->next = next;
    }
};
```

（1）成员变量：

index（int 类型）：用于表示课程的序号（索引），方便后续对课程节点进行其他操作；

name（QString 类型）：用于存储课程名称；

weight（int 类型）：记录课程的学分，在选课过程中会对总学分进行限制；

next（myNode*类型）：记录指向的下一门课程。

（2）构造函数：

myNode(int d, myNode* next)

myNode(int d, int w, QString n, myNode* next)

（3）成员方法：

此类中不包含成员方法。

3.2.2 myEdge（课程的前驱后继关系）

代码如下所示：

```
class myEdge {
    friend class Graph;
    friend class Graphl;
    friend class List;
```

```
private:
    int from, to;//起始点，终止点

public:
    //构造函数
    myEdge() {
        from = 0;
        to = 0;
    }
    myEdge(int f, int t) {
        from = f;
        to = t;
    }
};
```

（1）成员变量：

from（int 类型）：先修课程的序号（index）；

to（int 类型）：后继课程的序号（index）。

（2）构造函数：

myEdge()

myEdge(int f, int t)

（3）成员方法：

此类中不包含成员方法。

说明：

此处 friend class 为友元类，Graph、Graphl、List 分别为基础图、带有邻接表的图、邻接表。友元类的作用是可以使 Graph、Graphl、List 这三个类可以直接使用 myEdge 类中的 private 成员变量。

3.2.3 List（邻接表）

代码如下所示：

```
class List {
public:
    myNode* head = NULL; //头指针
    myNode* tail = NULL; //尾指针
    //构造函数
    List() {}
    //插入操作
    void insert(myEdge e) {
        myNode* temp = head;
        myNode* node;
        while (temp->next != NULL) {
            if (e.to < temp->next->index) break;
            temp = temp->next;
```

```
        }
        if (temp == NULL || temp->next == NULL) {
            node = new myNode(e.to, NULL);
            tail = node;
        }
        else node = new myNode(e.to, temp->next);
        temp->next = node;
    }

    //析构函数
    ~List() {
        myNode* temp = head;
        while (temp != NULL) {
            head = head->next;
            delete temp;
            temp = head;
        }
    }
};
```

（1）成员变量：

head（Node*类型）：头指针，默认值为 NULL；

tail（Node*类型）：尾指针，默认值为 NULL。

（2）构造函数：

List()

（3）成员方法：

void insert(myEdge e)：该函数用于向 List 链表中插入一条边 e，插入时会按照 e 边的指向节点（也就是后继课程）的序号进行排序，确保序号递增，这样在操作时能确保每一次同样的操作能得到固定的结果。

此类中不包含成员方法。

（4）析构函数：

~List()：创建一个 temp 指针（Node*类型），开始指向 head 所在地址，然后将 head 后移，释放 temp 所指向的地址（也就是原 head 的位置），重复以上动作，就可以释放 List 中全部内存。

3.2.4 Graph（基础图）

代码如下所示：

```
class Graph {

public:
    int numVertex;  // 顶点个数
    // int numEdge;  // 边个数
    bool* Mark;  // 标记图的顶点是否被访问过
```

```
7.      bool* added;  // 标记课程是否已经被添加到课表
8.      int* inDegree;  // 存放图中顶点的入度
9.      int* outDegree;  // 存放图中顶点的出度
10.     //构造函数
11.     Graph() {
12.         numVertex = 0;
13.         // numEdge = 0;
14.     }
15.     Graph(int v) {
16.         numVertex = v;
17.         // numEdge = e;
18.         Mark = new bool[numVertex + 1];
19.         added = new bool[numVertex + 1];
20.         inDegree = new int[numVertex + 1];
21.         outDegree = new int[numVertex + 1];
22.         for (int i = 1; i <= numVertex; i++) {
23.             Mark[i] = 0;
24.             added[i] = 0;
25.             inDegree[i] = 0;
26.             outDegree[i] = 0;
27.         }
28.     }
29.
30.     //返回图中顶点个数
31.     int getNumV() {
32.         return numVertex;
33.     }
34.     //判断 e 是否为边
35.     bool isEdge(myEdge e) {
36.         if (e.from >= 1 && e.from <= numVertex && e.to >= 1 && e.to <= numVe
    rtex) return true;
37.         else return false;
38.     }
39. };
```

（1）成员变量：

numVertex（int 类型）：课程数目，所需要创建的 List 数目需要参照该数值；

Mark（bool*类型）：本质上是一个 bool 数组，数组有效大小与 numVertex 相等，该数组用于标记课程节点是否被访问过，如果已被访问则置 1，反之置 0；

added（bool*类型）：与 Mark 类似，作用稍有不同，该数组用于标记课程节点是否已经被添加到课表中并从图中删除，如果已被添加且删除则置 1，反之置 0；

inDegree（bool*类型）：本质上是一个 bool 数组，数组有效大小与 numVertex 相等，该数组用于记录每个课程节点的入度（也就是该课程有多少门先修课程），入度只能为自然数；

outDegree（bool*类型）：功能与 inDegree 基本相同，用于记录出度（也就是该课程有多

少门后继课程）。

（2）构造函数：

Graph()：默认构造函数，不含参数，未对 Mark、added、inDegree、outDegree 数组初始化；

Graph(int v)：传入的参数为课程数，该构造函数会对 numVertex 赋值并初始化各数组。

（3）成员方法：

int getNumV()：返回课程总数，方便在其他.cpp 文件中调用（因为 numVertex 为 private 成员变量，对外部不可见）；

bool isEdge(myEdge e)：传入一个边，判断是否符合要求（课程节点的 index 必须为正整数，且不能大于课程总数）。

3.2.5 Graphl（带有邻接表的图）

代码如下所示：

```
1.  //邻接表表示图
2.  class Graphl : public Graph {
3.  private:
4.
5.  public:
6.      List* list;
7.      QMap<QString, int> name_index;  // 记录图中 Node 的课程名称与索引的对应关系
8.      QMap<QString, int> name_weight;  // 记录图中 Node 的课程名称与学分的对应关系
9.      // 构造函数
10.     Graphl(int v) : Graph(v) {
11.         int i;
12.         list = new List[numVertex + 1];
13.         for (i = 1; i <= numVertex; i++) {
14.             list[i].head = new myNode(i, NULL);
15.             list[i].tail = list[i].head;
16.         }
17.     }
18.
19.     // 拷贝构造函数
20.     Graphl(const Graphl &g) {
21.         numVertex = g.numVertex;
22.         // 根据结点数创建数组
23.         int i;
24.         list = new List[numVertex + 1];
25.         for (i = 1; i <= numVertex; i++) {
26.             list[i].head = new myNode(i, NULL);
27.             list[i].tail = list[i].head;
28.         }
```

```
29.         Mark = new bool[numVertex + 1];
30.         added = new bool[numVertex + 1];
31.         inDegree = new int[numVertex + 1];
32.         outDegree = new int[numVertex + 1];
33.
34.         // 赋值
35.         myNode *temp;
36.         name_index = g.name_index;
37.         name_weight = g.name_weight;
38.         for(int i = 1; i <= numVertex; i++) {
39.             temp = g.list[i].head->next;
40.             while(temp != NULL) {
41.                 setEdge(i, temp->index);
42.                 temp = temp->next;
43.             }
44.             // 不能在此处赋值，因为更新后续的 list[i]时调用 seEdge 会修改之前复制好的 inDegree 与 outDegree
45.             // inDegree[i] = g.inDegree[i];
46.             // outDegree[i] = g.outDegree[i];
47.         }
48.         for(int i = 1; i <= numVertex; i++) {
49.             Mark[i] = g.Mark[i];
50.             added[i] = g.added[i];
51.             inDegree[i] = g.inDegree[i];
52.             outDegree[i] = g.outDegree[i];
53.         }
54.     }
55.
56.     //返回顶点 v 的第一条边
57.     myEdge fistEdge(int v) {
58.         myNode* first = list[v].head->next;
59.         myEdge edge(v, first->index);
60.         return edge;
61.     }
62.     //返回与边 e 有同样初始点的下一条边
63.     myEdge nextEdge(myEdge e) {
64.         myNode* first = list[e.from].head->next;
65.         while (first->index != e.to) {
66.             first = first->next;
67.         }
68.         first = first->next;
69.         myEdge edge(e.from, first->index);
70.         return edge;
71.     }
```

```
72.     //判断是否为图中的边
73.     bool isMyEdge(myEdge e) {
74.         myNode* first = list[e.from].head->next;
75.         if (!isEdge(e)) return false; //如果 e 不为边必定为假
76.         while (first != NULL) {
77.             if (first->index == e.to) return true;
78.             first = first->next;
79.         }
80.         return false;
81.     }
82.     //为图设置一条边
83.     bool setEdge(int f, int t) {
84.         myEdge e(f, t);
85.         if (isMyEdge(e)) return false; //如果该边已经存在，返回 false
86.         else {
87.             list[f].insert(e);
88.             inDegree[t]++;
89.             outDegree[f]++;
90.             return true;
91.         }
92.     }
93.     //删除图的一条边
94.     bool delEdge(int f, int t) {
95.         myNode* first = list[f].head->next;
96.         myNode* pre = list[f].head;
97.         while (first != NULL) {
98.             //若找到该边
99.             if (first->index == t) break;
100.             pre = first;
101.             first = first->next;
102.         }
103.         //找到该边
104.         if (first != NULL) {
105.             pre->next = first->next;
106.             //如果删除的是最后一条边
107.             if (first->next == NULL) list[t].tail = pre;
108.             inDegree[t]--;
109.             outDegree[f]--;
110.             return true;
111.         }
112.         //未找到该边
113.         else return false;
114.     }
115.
```

```
116.     // 删除入度为 0 的 zero 结点
117.     void delZero(int zIndex) {
118.         myNode *tempNode = list[zIndex].head->next;
119.         int tempIndex;
120.         while(tempNode != NULL) {
121.             tempIndex = tempNode->index;
122.             delEdge(zIndex, tempIndex);
123.             tempNode = list[zIndex].head->next;  // 指向的第一个结点
124.         }
125.     }
126.
127.     // 拓扑排序并写入文件，若包含回路则返回 false
128.     bool topologySort(QString projectName, int i) {
129.         bool noLoop = false;  // 如果遍历所有结点找不到入度为 0 的点，则说明图中包含回路
130.         QDir dir("../Scheduling-System/Scheduling-System/myProject");  // 初始化 dir 为当前目录
131.         dir.cd(projectName);
132.         QString s = QString::number(i).append(".txt");  // i.txt
133.         QString filePath = dir.absoluteFilePath(s);
134.         QFile file(filePath);
135.         // 以读写的方式打开文件，成功返回 true，失败返回 false
136.         // 创建 i.txt 文件
137.         file.open(QIODevice::ReadWrite);
138.         file.close();
139.         // 清空文件内容
140.         file.open(QFile::WriteOnly|QFile::Truncate);
141.         file.close();
142.         // 写入内容
143.         file.open(QIODevice::Append | QIODevice::Text);
144.         QTextStream out(&file);
145.         int j;
146.         for(j = 1; j <= numVertex; j++) {
147.             if(inDegree[j] == 0) {
148.                 // 如果入度为 0 且未被访问，说明本轮拓扑暂时未检测到环
149.                 if(Mark[j] == 0) {
150.                     Mark[j] = 1;
151.                     noLoop = true;
152.                 }
153.                 // 不能直接删去边，因为会直接导致本该下一次选的课程在本次入度就为 0
154.             }
155.         }
156.         for(j = 1; j <= numVertex; j++) {
```

```
157.             if(Mark[j] == 1) {
158.                 // 如果该课程未被添加到课程表中
159.                 if(added[j] == 0) {
160.                     added[j] = 1;  // 标记为已添加过，防止再次添加课程
161.                     delZero(j);
162.                     out << name_index.key(j);
163.                     out << "\n";
164.                 }
165.             }
166.         }
167.         return noLoop;
168.     }
169. };
```

（1）成员变量：

List（List*类型）：本质上是一个 List 数组（邻接表），数组有效大小与 numVertex 相等，该数组用于记录各课程节点的前驱后继关系，list[i]为序号为 i 的课程节点所对应的 List，list[i]的 head 一定指向该课程节点，而 head 后续所连接的节点都为该课程节点的后继节点；

name_index（QMap<QString, int>类型）：用于记录 Graphl 中课程名称与序号的对应关系；

name_weight（QMap<QString, int>类型）：用于记录 Graphl 中课程名称与学分的对应关系。

（2）构造函数：

Graphl(int v)：传入的参数为课程数量，对 List 数组进行初始化。

Graphl(const Graphl &g)：拷贝构造函数，传入的参数为所要复制的 Graphl，构造函数可以更方便地创建一个完全相同的 Graphl，用于保护原有的 Graphl。

（3）成员方法：

myEdge fistEdge(int v)：传入的参数为指定的课程节点序号，返回该课程节点的第一个后继课程，也就是该 List 的 head->next；

myEdge nextEdge(myEdge e)：传入的参数为一条边（课程前驱后继关系中的一个），返回下一条边（该课程节点与下一个后继节点所构成的前驱后继关系）；

bool isMyEdge(myEdge e)：传入的参数为一条指定边（改边可以是任意的，不做特殊要求），该函数用于判断传入边是否在该 Graphl 中，首先调用 isEdge（）来判断该边是否合规，在循环查找该边，若能在该 Graphl 中查找到该边，则返回 true，否则返回 false；

bool setEdge(int f, int t)：传入的参数为前驱课程的序号以及后继课程的序号，该函数用于向 Graphl 中添加一条边，返回值为 bool 类型，用于判断是否创建成功，如果该边不符合基本限制，则返回 false，反之返回 true；

bool delEdge(int f, int t)：传入的参数为前驱课程的序号以及后继课程的序号，该函数用于删除 Graphl 中的一条边，返回值为 bool 类型，如果找到该边，删除该边并修改 inDegree 以及 outDegree 数组的值，然后返回 true，反之不执行操作返回 false；

void delZero(int zIndex)：传入的参数为入度为 0 的课程节点的序号，该函数用于删除入度为 0 的课程节点，与普通的删除不同的是，由于入度为 0，所以只需要关注后继节点并更新相应的 outDegree 数组即可，该函数是拓扑排序的基础；

bool topologySort(QString projectName, int i)：该函数用于执行一轮拓扑排序操作，此处

定义的一轮是指将当前 Graphl 中入度为 0 的课程节点全部删除，然后将其写入相应文件，方便后续读取以及查找，返回值为 bool，用于检查该 Graphl 是否包含回路，若在函数执行过程中，发现已经没有入度为 0 的节点且还有剩余的课程节点，则说明包含回路，返回 false，反之返回 true。

3.2.6 node

代码如下所示：

```
1.  //! [0]
2.  class Node : public QGraphicsItem
3.  {
4.  public:
5.      Node();
6.      QString name;  // 课程名称
7.
8.      void addEdge(Edge* edge);
9.      QList<Edge*> edges() const;
10.
11.     enum
12.     {
13.         Type = UserType + 1
14.     };
15.     int type() const override { return Type; }
16.
17.     void calculateForces();
18.     bool advancePosition();
19.
20.     QRectF boundingRect() const override;
21.     QPainterPath shape() const override;
22.     void paint(QPainter* painter, const QStyleOptionGraphicsItem* option, QW
    idget* widget) override;
23.
24. protected:
25.     QVariant itemChange(GraphicsItemChange change, const QVariant& value) ov
    erride;
26.
27.     void mousePressEvent(QGraphicsSceneMouseEvent* event) override;
28.     void mouseReleaseEvent(QGraphicsSceneMouseEvent* event) override;
29.
30. private:
31.     QList<Edge*> edgeList;
32.     QPointF newPos;
33. };
34. //! [0]
```

```
35.
36. #endif // NODE_H
```

该类用于绘制有向无环图中的节点，来源于 Qt 的示例，为开源代码，此处不再过多说明。

3.2.7 edge

代码如下所示：

```
1.  class Node;
2.
3.  //! [0]
4.  class Edge : public QGraphicsItem
5.  {
6.  public:
7.      Edge(Node *sourceNode, Node *destNode);
8.
9.      Node *sourceNode() const;
10.     Node *destNode() const;
11.
12.     void adjust();
13.
14.     enum { Type = UserType + 2 };
15.     int type() const override { return Type; }
16.
17. protected:
18.     QRectF boundingRect() const override;
19.     void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QW
    idget *widget) override;
20.
21. private:
22.     Node *source, *dest;
23.
24.     QPointF sourcePoint;
25.     QPointF destPoint;
26.     qreal arrowSize = 10;
27. };
28. //! [0]
29.
30. #endif // EDGE_H
```

该类用于绘制有向无环图中的有向边，来源于 Qt 的示例，为开源代码，此处不再过多说明。

4.测试

4.1 正确运行程序的用例

（1）打开程序

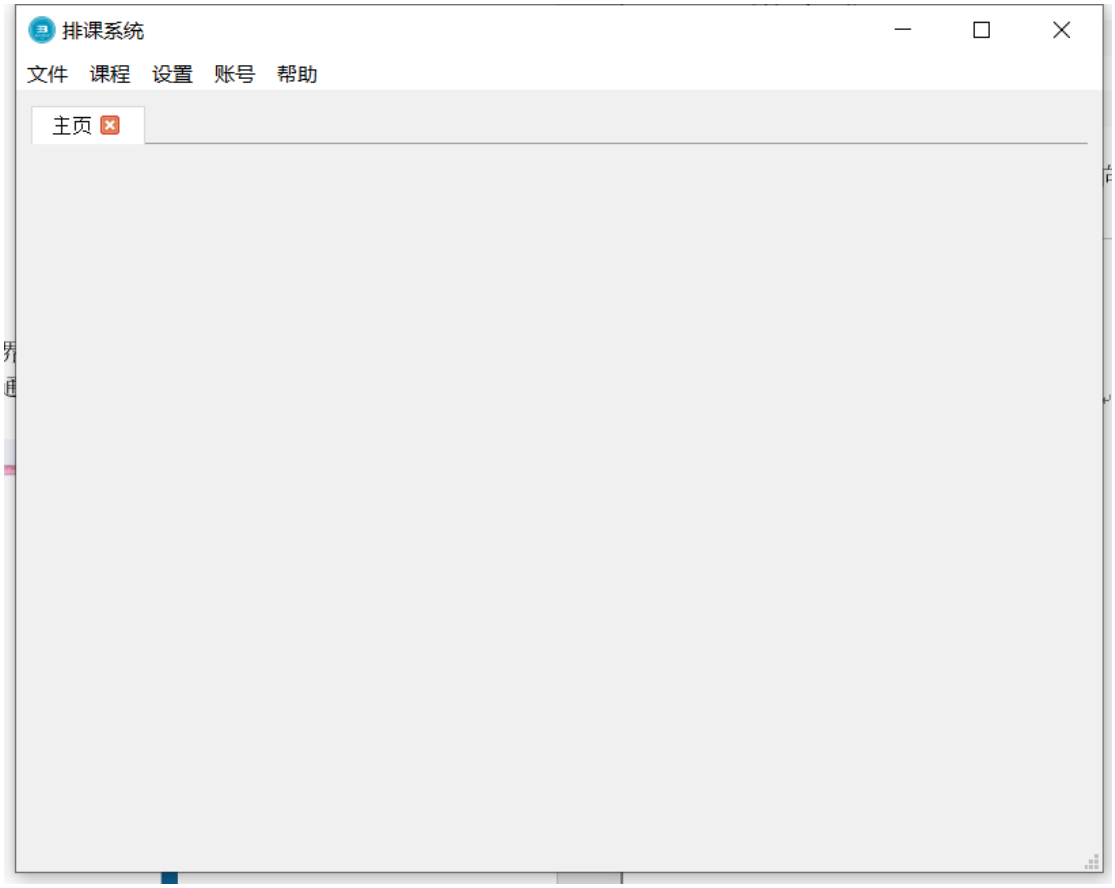

图 4.1 主界面

（2）新建项目

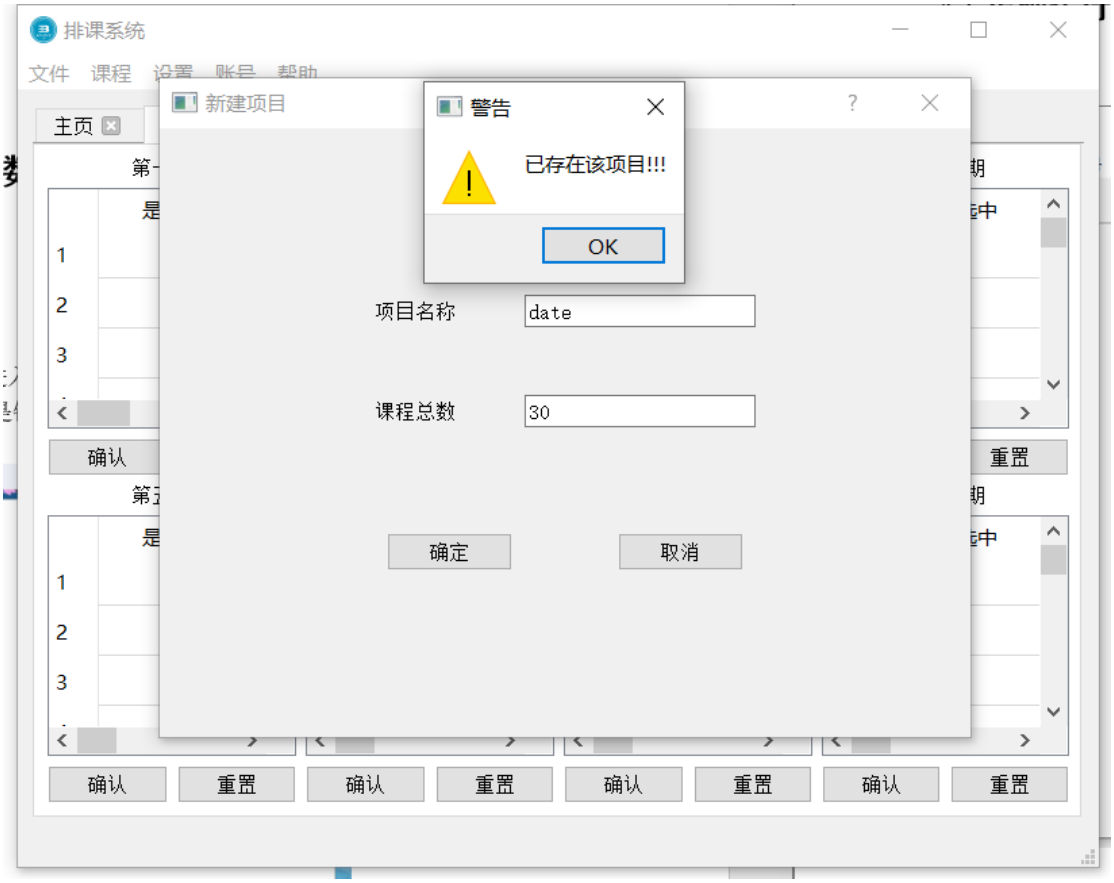

图 4.2 新建项目（已有项目）

（3）添加课程

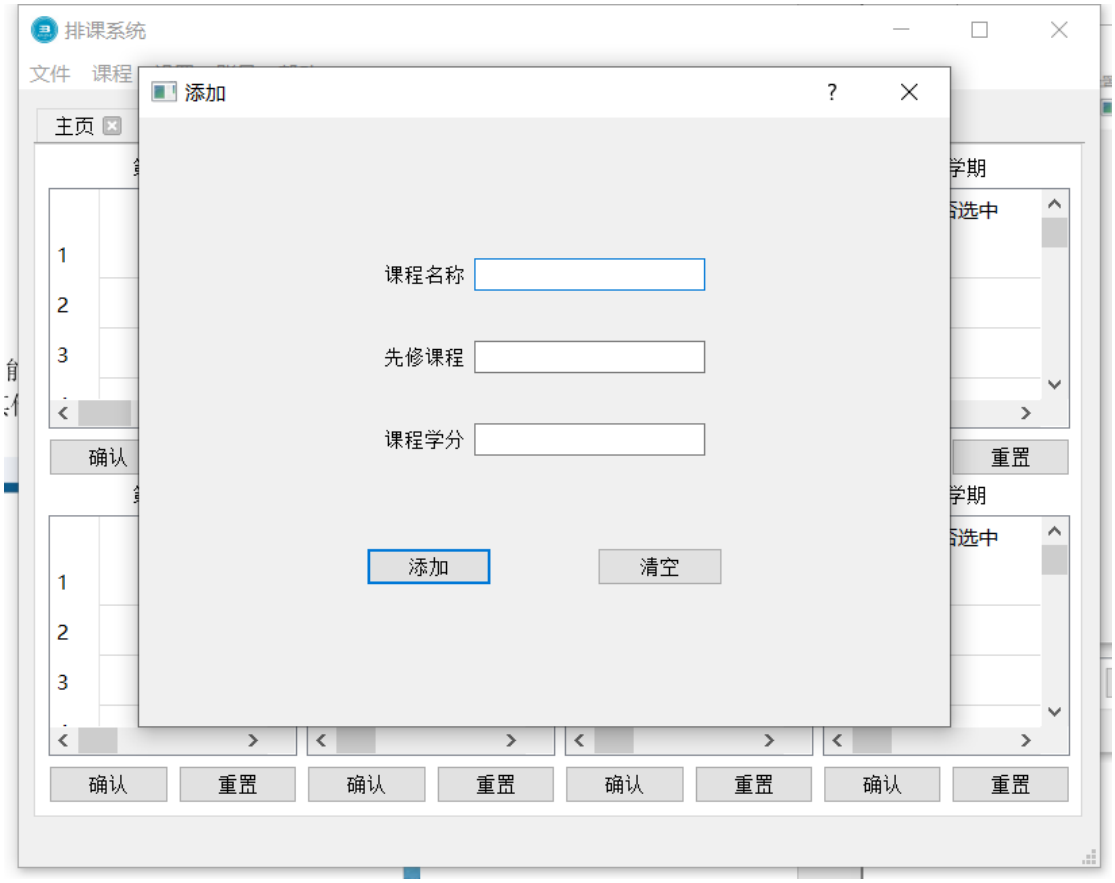

图 4.3 添加课程

（4）查看已有课程

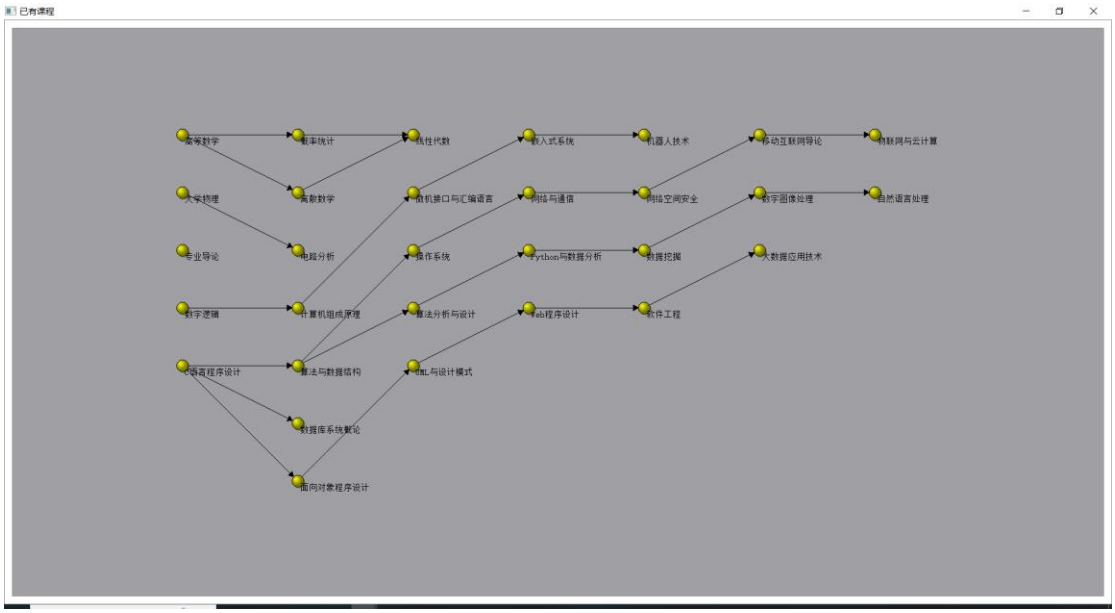

图 4.4 查看已有课程

（5）一键生成课表

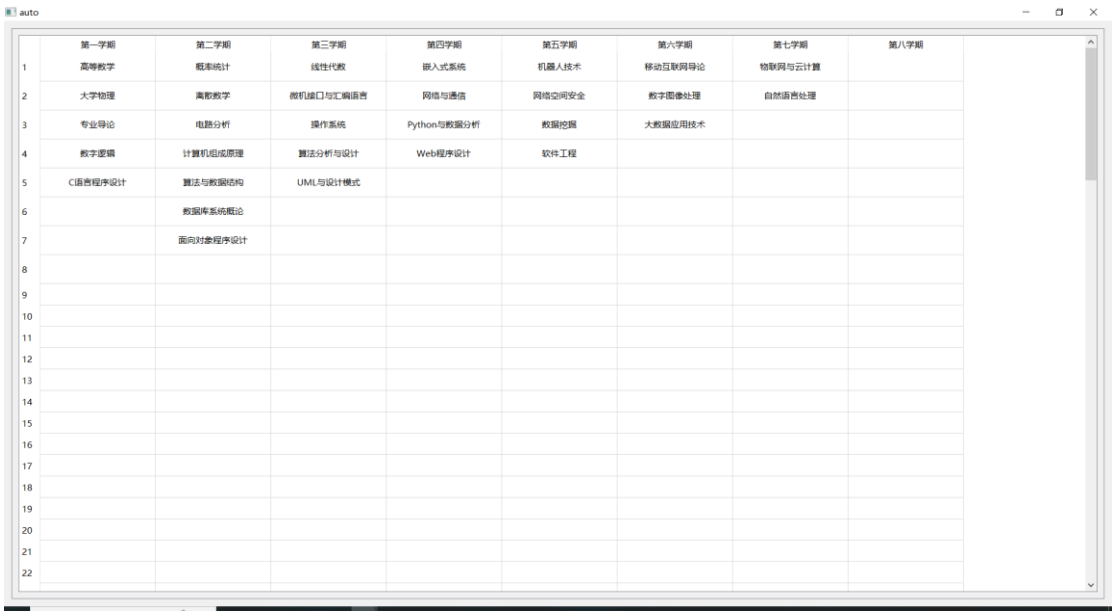

图 4.5 一键生成课表

（6）自主选课

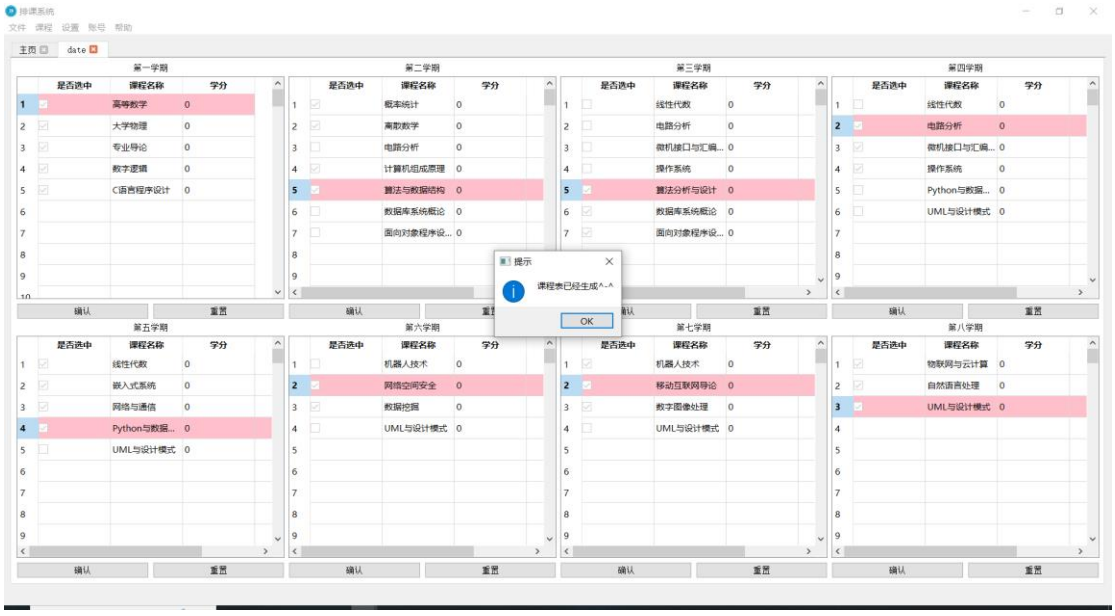

图 4.6 自主选课

（7）查看课表

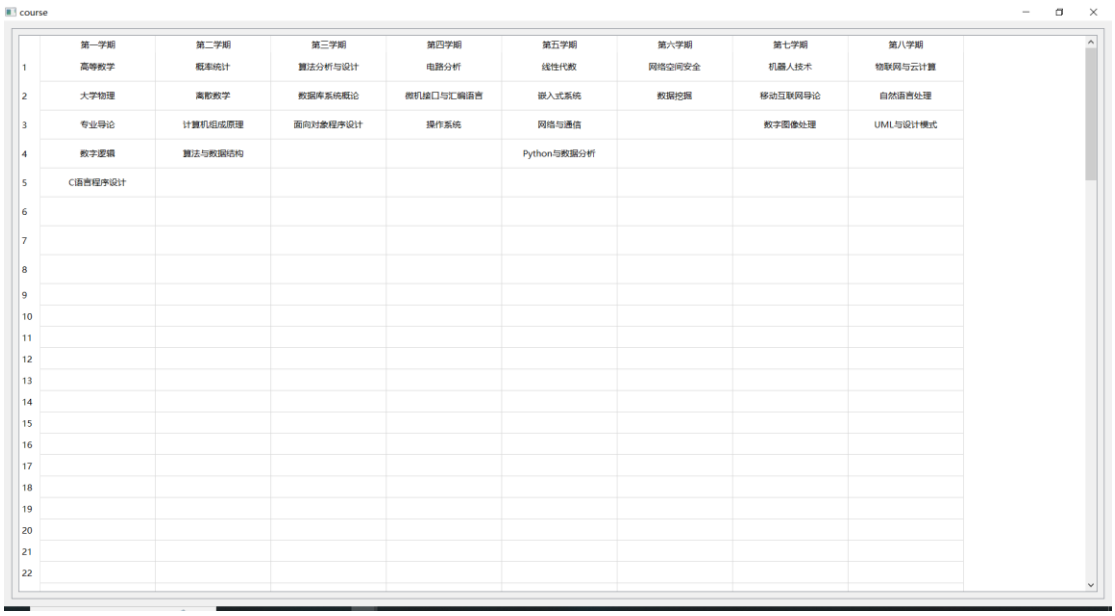

图 4.7 查看课表

4.2 导致程序运行错误的用例

（1）项目名输入为空

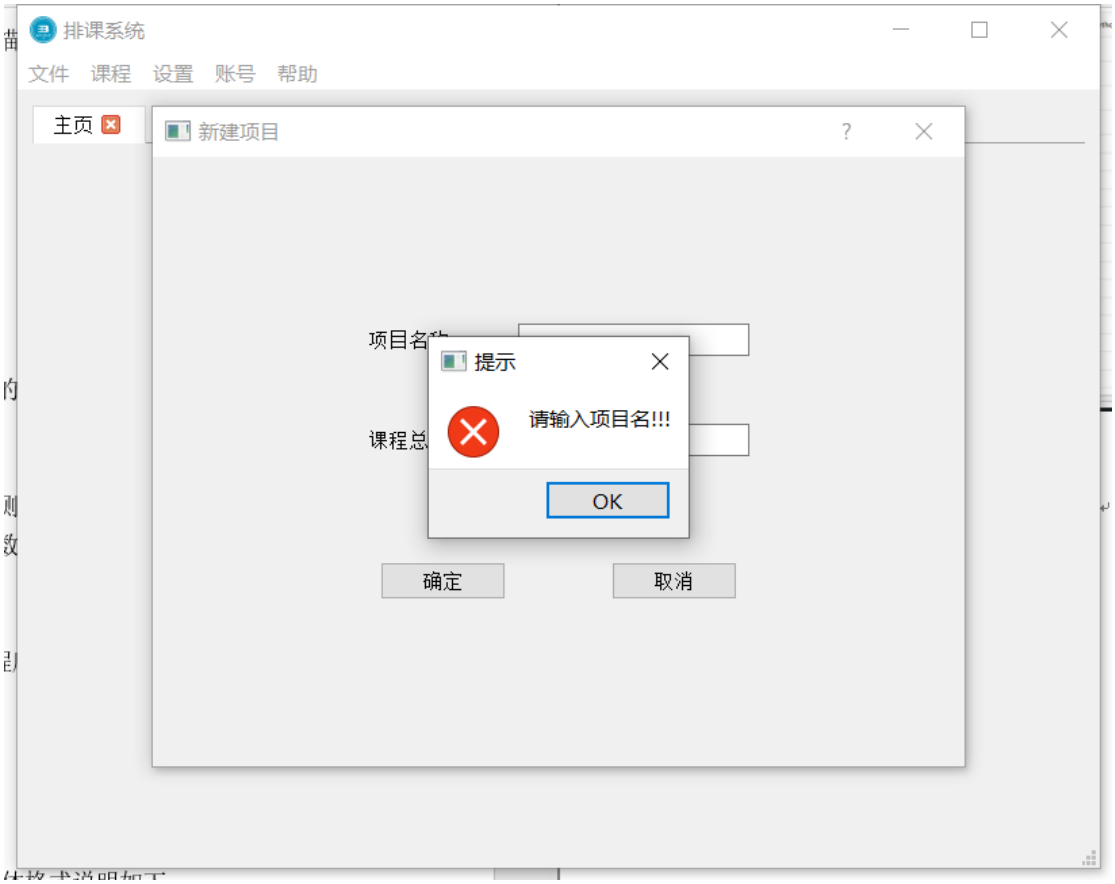

图 4.8 项目名输入为空

（2）课程总数输入为非正整数

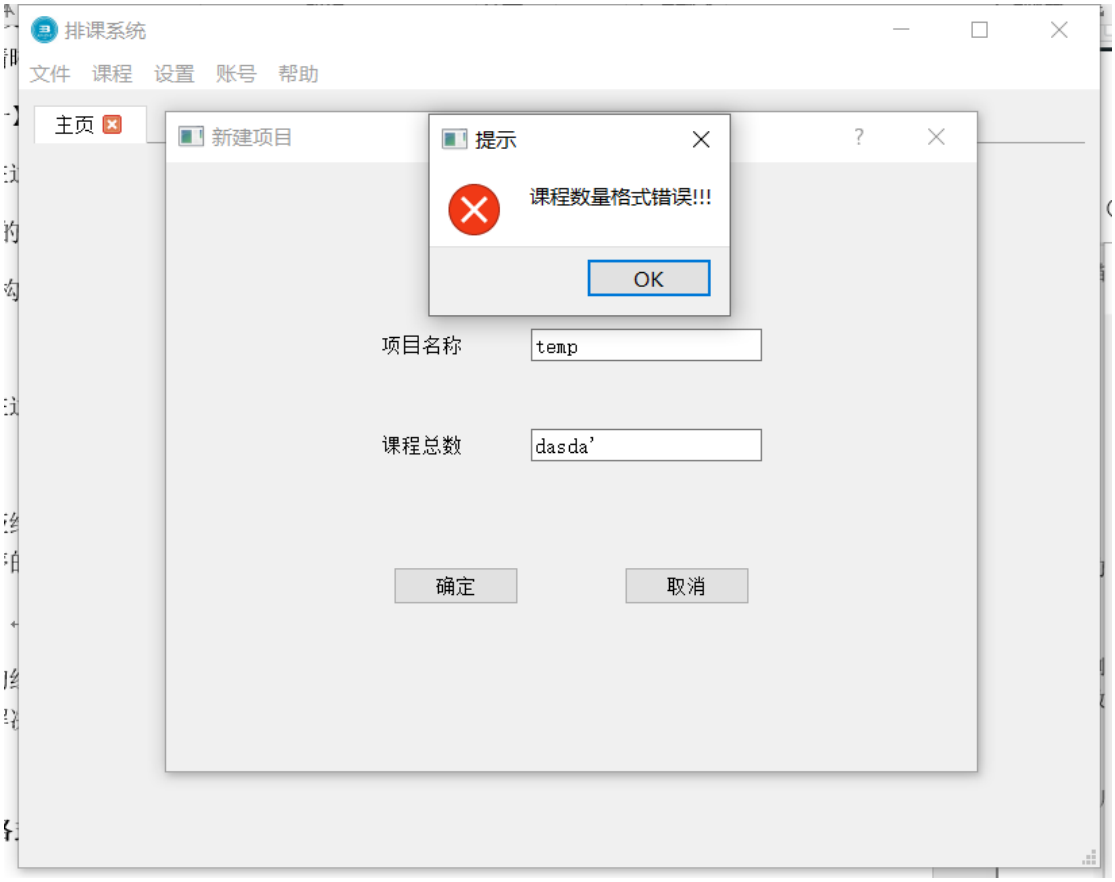

图 4.9 课程总数输入为非正整数

（3）课程学分输入为非自然数

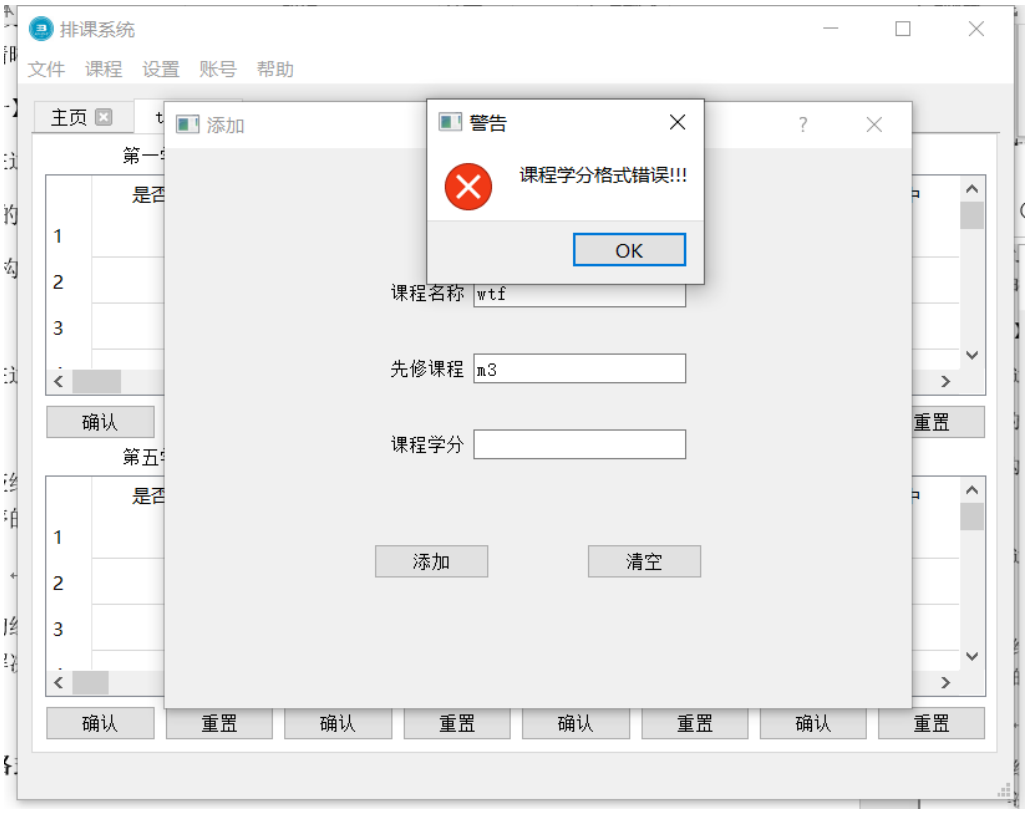

图 4.10 课程学分输入为非自然数

5.总结与提高

5.1 遇到的问题及解决情况

（1）界面关闭信号无法发出

在该程序中，我想在添加课程结束后触发后续的排课操作，于是利用信号机制发送接收关闭信号，但是发现没有作用。原因是点击窗口自带的关闭选项后，该窗口会直接关闭，而无法发出信号，因此接收方永远无法接收，所以需要重写关闭事件。

代码如下所示：

```
// 重写关闭事件，否则关闭后无法发出 addClose 信号
    void closeEvent ( QCloseEvent * e )
    {
        if( QMessageBox::question(this,
                                  tr("Quit"),
                                  tr("Are you sure to quit this application?
"),
                                  QMessageBox::Yes, QMessageBox::No )
            == QMessageBox::Yes){
            e->accept();  // 不会将事件传递给组件的父组件
            emit addClose();
        }
        else
            e->ignore();
    }
```

（2）复制构造函数出错

在写 Graphl 类的复制构造函数时出现了错误，在同一轮循环中同时更新了入度出度，并同时调用了 setEdge（该函数用于向 Graphl 添加边，会修改入度出度的值），从而导致入度出度的数组发生错误，导致排课失败。

赋值部分代码如下所示：

```
// 赋值
        myNode *temp;
        name_index = g.name_index;
        name_weight = g.name_weight;
        for(int i = 1; i <= numVertex; i++) {
            temp = g.list[i].head->next;
            while(temp != NULL) {
                setEdge(i, temp->index);
                temp = temp->next;
            }
```

```
            // 不能在此处赋值，因为更新后续的 list[i]时调用 seEdge 会修改之前复制好的 inDegree 与 outDegree
            // inDegree[i] = g.inDegree[i];
            // outDegree[i] = g.outDegree[i];
        }
        for(int i = 1; i <= numVertex; i++) {
            Mark[i] = g.Mark[i];
            added[i] = g.added[i];
            inDegree[i] = g.inDegree[i];
            outDegree[i] = g.outDegree[i];
        }
```

（3）QTableWidgetItem 居中显示导致崩溃

在对课表进行图形化显示时，考虑到美观，所以将其居中显示，但是在实现的过程中出现了程序崩溃的现象，最后发现原因是不能对空白的单元格（未赋值）进行居中显示操作，否则就会导致程序崩溃。

（4）无法读取到项目名称

用户创建项目后，该程序会创建一个 title 为项目名称的窗口，但是尝试多次后始终为空，原因是创建窗口的操作未在接收名称的操作之后执行。因此我引入了信号与槽函数机制，接收到项目名称后，会发送一个接收成功的信号，只有在接收到该信号后，窗口才会被创建，这样就可以确保新建窗口的 title 为输入的项目名称。

（5）打包.exe 文件后文件读写失效

由于代码中文件读写部分用了相对路径“../”，所以.exe 所在的上级目录要与之相同，否则无法找到对应文件完成相应操作。

5.2 自己对完成课设情况的评价

完成了所需的基本功能，做了部分拓展，但是仍有很多不足之处。主要是因为前期统筹工作没有做好，一开始想做一个功能较多且界面美观的程序，但是没有思考所需的代码能力以及时间成本，导致后期只能压缩工作量，导致最后程序的效果与预想差距较大。之后我会更注意前期的规划，评估好自身的能力和时间，以便更好地完成工作。

5.3 体会与收获

一学期的数据结构设计让我受益匪浅，本次课程设计是对上学期数据结构与算法课程的巩固与提高，既需要对算法进行设计，又要提供给用户友好的界面和交互体验。在上学期学习数据结构与算法时，我在学习完每章之后会自己用代码编写所学的数据结构，以求加深我对数据结构的认识与理解。在本次算法编写过程中，我沿用了上学期所保留的代码，但却发现了部分问题，我认识到过去对某些数据结构以及算法仍存在误区，于是我不断推敲逻辑、修改代码，在这过程中我的编程能力和动手能力得到了提升。除了代码，还需要对界面进行精心设计，在大一的时候我们曾做过 C 语言课设，当时使用 easyX 完成了简单的界面设计，而这次数据结构课设我采用了 Qt 对界面进行设计，Qt 设计的界面更加美观，但是难度也相应上升。由于未学习过 Qt，一切都是从 0 开始，刚开始使用的时候，连创建项目都很困难，对其中的代码和函数都不熟悉，处处遇到困难，于是我查阅相关资料，不断修改代码不断试错，渐渐较熟悉地掌握了 Qt 控件、信号、槽函数等机制的使用。

通过这段时间的课程设计，我认识到数据结构是一门比较难的课程，需要多花时间上机练习。这次的程序训练培养了我实际分析问题、编程和动手能力，使我掌握了程序设计的基本技能，提高了我适应实际、实践编程的能力。

最后，要衷心感谢杜永萍老师这两学期对我们学习上的指导，每次遇到问题杜老师都会给我们耐心解答，如果没有杜老师的帮助，我可能很难完成这次数据结构的课程设计。祝杜老师身体健康，工作顺利!!!