项目说明文档

数据结构课程设计

——排课软件

作 者 姓 名： 肖杨

学 号： 1950430

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

1 分析 3

2 设计 4

3 实现 11

4 测试 23

1 分析

1.1 背景分析

排课是每个学期都必须面临的一件繁重的日常教务工作，是为下学期教学工作能顺利组织实施的一个前提条件。排课管理是高校每个学期教学教务管理工作的一个主要内容，是整个综合教学教务管理系统中必不可少的部分。

排课也是一项复杂而精细的工作，需要考虑的因素很多，涉及到方方面面，归纳起来主要有五方面相互作用、相互关联的因素，即课程、教师、时间、班级以及教室。如何把各要素科学、规范、高效地进行优化组合，以保证教学秩序的稳定，促进教学质量的提高，这是每所高校都面临的课题。

1.2 功能分析

假设周一至周五上课，每天上10节课，第1大节为第1-2节课，第二大节为第3-5节课，第3大节为第6-7节课，第4大节为8-10节课，在排课时，如一门课程有3节课，则优先安排3节课连续上；如3节课连续无法按排，再优先安排两节课连续上，最后再安排单节课上的情况；如果一门课程需要安排上两天，为教学效果较好，最好不安排在相邻的两天，比如优先安排相隔2天上课。

1 输入数据包括：各学期所开的课程数（必须使每学期所开的课程数之和与课程总数相等），课程编号，课程名称，周学时数，指定开课学期，先决条件。如指定开课学期为0，表示有电脑自行指定开课学期。

2 如输入数据不合理，比如每学期所开的课程数值和与课程总数不相等，应显示适当的提示信息。

3 用文本文件存储输入数据，并且读入计算机。

4 用文本文件存储产生的各学期的课表。

1.3 须知

本项目文件编码均为GBK编码，如果评测时出现乱码，请把输入输出文件编码以及终端显示编码改为GBK编码。

执行exe文件时，应当把测试文件in.txt和out.txt放入exe文件的同一路径中，从而能够正常打开并读写，若无文件则会报错。

2 设计

2.1 算法设计

课程之间存在先修关系，容易看出先修关系应该要满足偏序关系，否则会形成循环依赖，因此考虑对其进行拓扑排序。为减少程序的复杂性，这里不使用图进行拓扑排序，仅使用队列。在排课过程中每排好一学期的课后，将所有课的后继课程入度减1，再判断是否加入队列，直到队列为空时判断排课完毕或出现循环依赖关系。由于数据需要有稳定的保存位置，考虑使用一个Schedule类保存所需的所有数据。课程信息利用重载了[]的哈希表，每门课程作为一个单独的Course进行保存。每学期的排课信息利用Term进行保存，并在Schedule类中保存八个学期的课程。

2.2 数据结构设计

如上功能分析所述，本排课系统需要一个队列来保存当前可排课程。为了利用键值对快速找到对应课程的信息，本项目还利用了哈希表。除此之外不专门利用其他的数据结构功能。

2.3 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将queue类和unordered\_map类的值设计成模板。为了更好的利用模板类，设计了Course类、Term类和Schedule类，分别用于存储单个课程的信息、单个学期的排课情况、整个学期的课程、排课情况和相应接口。

2.4 成员与操作设计

哈希表类

类定义：

template<class v>

class unordered\_map

{

private:

class Node

{

public:

Node(string ikey, v ivalue);

Node();

~Node();

string key;

v value;

Node\* next;

};

int hashSize;

Node\* hashnode;

int hash(string key);

public:

unordered\_map(int size);

~unordered\_map();

Node\* find(string key);

v& operator [] (string key);

};

私有类：

class Node

{

public:

Node(string ikey, v ivalue);

//含参构造函数

Node();

//无参构造函数

string key;//键

v value;//值

Node\* next;//开散列方法，连接的下一个结点

};

私有操作：

int hash(string key);

//哈希化函数

私有成员：

int hashSize;

//哈希表的大小

Node\* hashnode;

//储存哈希表的头结点

公有操作：

unordered\_map(int size);

//含参构造函数

Node\* find(string key);

//查找键是否在表内已有对应结点

v& operator [] (string key);

//利用中括号实现用键访问，方便省去建立结点再插入的过程

队列类

类定义：

template<class T>

class queue

{

private:

int size;

int number;

T\* buffer;

int head;

int tail;

public:

queue(int num);

~queue();

void push(const T& elem);

T pop();

T front();

int getNumber();

bool empty();

bool full();

};

私有成员：

int size;//循环队列的大小

int number;//队列中的元素数量

T\* buffer;//用于保存元素的地址

int head;//队首下标

int tail;//队尾下标

公有操作：

queue(int num);

//用队列大小初始化

void push(const T& elem);

//入队

T pop();

//出队并返回出队值

T front();

//返回队首值

int getNumber();

//得到队列中元素数目

bool empty();

//队列判空

bool full();

//队列判满

Course类

类定义：

class Course

{

public:

Course() = default;

~Course();

void addNext(Course\* ptr);

void delOut(queue<Course\*>& q);

string m\_number;

string m\_name;

int classHours;

int openTerm;

int in = 0;

int out = 0;

Course\*\* next = nullptr;

};

公有成员：

string m\_number;

//课程编号

string m\_name;

//课程名称

int classHours;

//课时

int openTerm;

//开设学期

int in = 0;

//入度

int out = 0;

//出度

Course\*\* next = nullptr;

//指向所有后继课程指针的指针

公有操作：

Course() = default;

//默认构造函数

void addNext(Course\* ptr);

//加入下一个后继课程

void delOut(queue<Course\*>& q);

//将所有后继课程的入度减少1

Term类

类定义：

class Term

{

public:

Term();

~Term();

bool addCourse(Course\* c);

bool isFull();

void arrange();

int courseNum;

int presentClass = 0;

int classHour = 0;

Course\*\* buffer = nullptr;

string termTable[5][10];

bool isAvaiavle[5][4];

};

公有成员：

int courseNum;

//本学期开设的课程数目

int presentClass = 0;

//当前已决定在本学期开设的课程数

int classHour = 0;

//总课时数（避免课时数过多无法排课）

Course\*\* buffer = nullptr;

//存放课程信息

string termTable[5][10];

//课程表

bool isAvaiavle[5][4];

//每节课是否已排课

公有操作：

Term();

//无参构造函数

bool addCourse(Course\* c);

//向该学期加课，返回是否加课成功

bool isFull();

//学期课程数判满

void arrange();

//已经排满课后，对该学期的课程排课

Schedule类

类定义：

class Schedule

{

public:

Schedule(int classnum) :courses(classnum),checkReady(classnum){};

Term singleTerm[8];

unordered\_map<Course> courses;

queue<Course\*> checkReady;

void arrange();

};

公有成员：

Term singleTerm[8];

//八个学期

unordered\_map<Course> courses;

//存放课程信息

queue<Course\*> checkReady;

//队列中存放当前可排课的课程指针

3 实现

3.1 拓扑有序的实现

3.1.1 拓扑有序的实现流程

在读取每门课程的内容时，判断其是否有先修课要求，若无要求则加入队列。在进行总体排课的同时，每当排完一学期的课程，将这学期的课程后继课程入度减1，若入度为0则加入队列。

利用该操作即可在排课过程中使其拓扑有序，有效提升了排课效率。

3.1.2 核心代码

main：

for (i = 0; i < classnum; i++)

{

string s, courseNum, pre;

getline(in, s);

stringstream SS(s);

SS >> courseNum;

Course& c = scheduleTable.courses[courseNum];

c.m\_number = courseNum;

SS >> c.m\_name >> c.classHours >> c.openTerm;

if (c.openTerm)

{

if (scheduleTable.singleTerm[c.openTerm - 1].addCourse(&c))

{

continue;

}

else

{

cout << "无法正常排课" << endl;

system("pause");

return 0;

}

}

while (SS >> pre)

{

scheduleTable.courses[pre].addNext(&c);

c.in++;

}

if (c.in == 0)

{

scheduleTable.checkReady.push(&c);

}

}

Schedule::arrange：

if (singleTerm[term].isFull())

{

singleTerm[term].arrange();

for (int i = 0; i < singleTerm[term].courseNum; i++)

{

singleTerm[term].buffer[i]->delOut(checkReady);

}

term++;

}

void Course::delOut(queue<Course\*>& q)

{

for (int i = 0; i < out; i++)

{

if (--(next[i]->in) == 0)

{

q.push(next[i]);

}

}

}

3.2 分配单门课程的实现

3.2.1 分配单门课程的实现流程

首先，从队列中取出首个课程。考虑到每个学期有一定数量的课，且对课时有限制，因此可能出现当前排课的学期无法排课的情况，这时应向后遍历学期尝试排课，直到找到未完成排课且可进行排课的学期，此时将课程加入该学期。

每当成功分配一门课程后，判断该学期是否满课，若已满课则对该学期进行排课，并将其中的所有课程的后继课程入度减一以满足拓扑排序。

3.2.2 核心代码

void Schedule::arrange()

{

int term = 0;

bool isError = false;

while (term < 8)

{

if (checkReady.empty())

{

cout << "课程循环依赖！" << endl;

isError = true;

break;

}

auto course = checkReady.front();

if (singleTerm[term].addCourse(course))

{

checkReady.pop();

}

else

{

int i = term + 1;

while (i < 8)

{

if (singleTerm[i].addCourse(course))

{

checkReady.pop();

break;

}

else

{

i++;

}

}

if (i == 8)

{

cout << "无法排开课" << endl;

isError = true;

break;

}

}

if (singleTerm[term].isFull())

{

singleTerm[term].arrange();

for (int i = 0; i < singleTerm[term].courseNum; i++)

{

singleTerm[term].buffer[i]->delOut(checkReady);

}

term++;

}

}

if (isError)

{

cout << "无法正常排课" << endl;

system("pause");

exit(-1);

}

}

3.3 单学期排课的实现

3.3.1 单学期排课的实现流程

每次进行单学期排课的时候，该学期已分配满课。从第一门课程开始，将其课时进行拆解，使其拆分为3节课、2节课、1节课并计数。

首先排3节课的情况，这种情况每天仅有两个课时可能有空闲，对每个合理时间进行遍历判断是否有课，若无课则排课。若无法排三节课，则会将其拆分为2节课+1节课的形式。

其次是2节课的情况，这种情况则遍历所有课时，直到找到合理位置进行排课，1节课同理。

在排课过程中，利用isScheduled数组来判断当天是否排课，从而避开排课日期相同，若当天已排课，则优先隔天排课。由于隔天排课可能会使有日期未正常遍历，这里选择使用isVisited数组进行判断，并对j模5，直到数组全部被访问且未排课后count归零。

若count不为0，说明已正常排课，否则说明排课失败，3节课时进行拆分，2节课和1节课时说明排不开课应提示并退出程序。

3.3.2 核心代码

void Term::arrange()

{

int i, j;

for (i = 0; i < courseNum; i++)

{

auto course = buffer[i];

int hours = course->classHours;

int one = 0, two = 0, three = 0;

bool isScheduled[5] = { false,false,false,false,false };

while (hours > 0)

{

if (hours % 3 == 0)

{

three++;

hours -= 3;

}

else if (hours % 2 == 1)

{

if (hours >= 3)

{

three++;

hours -= 3;

}

else

{

one++;

hours -= 1;

}

}

else

{

two += hours / 2;

hours = 0;

}

}

if (one + two + three > 5)

{

cout << "某节课课时过多，排不开课" << endl;

system("pause");

exit(-1);

}

while (three || two || one)

{

bool isVisited[5] = { false,false,false,false,false };

int count = 5;

if (three)

{

for (j = 0; count > 0; j %= 5)

{

if (isScheduled[j])

{

isVisited[j] = true;

j += 2;

count--;

continue;

}

if (isVisited[j])

{

j++;

continue;

}

isVisited[j] = true;

if (isAvaiavle[j][1])

{

isScheduled[j] = true;

isAvaiavle[j][1] = false;

termTable[j][2] = course->m\_name;

termTable[j][3] = course->m\_name;

termTable[j][4] = course->m\_name;

three--;

break;

}

if (isAvaiavle[j][3])

{

isScheduled[j] = true;

isAvaiavle[j][3] = false;

termTable[j][7] = course->m\_name;

termTable[j][8] = course->m\_name;

termTable[j][9] = course->m\_name;

three--;

break;

}

count--;

j++;

}

if (count == 0)

{

three--;

two++;

one++;

if (one == 2)

{

one = 0;

two++;

}

}

}

else if (two)

{

for (j = 0; count > 0; j %= 5)

{

if (isScheduled[j])

{

isVisited[j] = true;

j += 2;

count--;

continue;

}

if (isVisited[j])

{

j++;

continue;

}

isVisited[j] = true;

if (isAvaiavle[j][0])

{

isScheduled[j] = true;

isAvaiavle[j][0] = false;

termTable[j][0] = course->m\_name;

termTable[j][1] = course->m\_name;

two--;

break;

}

if (isAvaiavle[j][1])

{

isScheduled[j] = true;

isAvaiavle[j][1] = false;

termTable[j][2] = course->m\_name;

termTable[j][3] = course->m\_name;

two--;

break;

}

if (isAvaiavle[j][2])

{

isScheduled[j] = true;

isAvaiavle[j][2] = false;

termTable[j][5] = course->m\_name;

termTable[j][6] = course->m\_name;

two--;

break;

}

if (isAvaiavle[j][3])

{

isScheduled[j] = true;

isAvaiavle[j][3] = false;

termTable[j][7] = course->m\_name;

termTable[j][8] = course->m\_name;

two--;

break;

}

count--;

}

if (count == 0)

{

cout << "某节课课时过多，排不开课" << endl;

system("pause");

exit(-1);

}

}

else

{

for (j = 0; count > 0; j %= 5)

{

if (isScheduled[j])

{

isVisited[j] = true;

j += 2;

count--;

continue;

}

if (isVisited[j])

{

j++;

continue;

}

isVisited[j] = true;

if (isAvaiavle[j][0])

{

isScheduled[j] = true;

isAvaiavle[j][0] = false;

termTable[j][0] = course->m\_name;

one--;

break;

}

if (isAvaiavle[j][1])

{

isScheduled[j] = true;

isAvaiavle[j][1] = false;

termTable[j][2] = course->m\_name;

one--;

break;

}

if (isAvaiavle[j][2])

{

isScheduled[j] = true;

isAvaiavle[j][2] = false;

termTable[j][5] = course->m\_name;

one--;

break;

}

if (isAvaiavle[j][3])

{

isScheduled[j] = true;

isAvaiavle[j][3] = false;

termTable[j][7] = course->m\_name;

one--;

break;

}

count--;

}

if (count == 0)

{

cout << "某节课课时过多，排不开课" << endl;

system("pause");

exit(-1);

}

}

}

}

}

3.4 总体系统的实现

3.4.1 总体系统的实现流程

对于该系统，首先利用fstream打开in.txt读入课程数目，得到课程数目后建立Schedule，并向其中的每学期读入课程数目，为Term分配相应空间储存所有课程的指针。

随后，向Schedule中的哈希表读入所有课程信息，并对有指定学期开课的课程直接分配学期，对没有指定学期的学期在判断其入度为0后加入队列，否则不加入队列。

之后在Schedule中对每个课程分配学期，并对每个学期进行排课。此时若正常排课完成，将会向out.txt中输出排课内容，在屏幕上显示提示信息，否则将会报错。

3.4.2 核心代码

int main()

{

fstream in, out;

in.open("in.txt", ios\_base::in | ios\_base::binary);

out.open("out.txt", ios\_base::out | ios\_base::binary);

if (in.is\_open() == false || out.is\_open() == false)

{

cout << "文件打开失败，请检查文件是否正确命名！" << endl;

system("pause");

return 0;

}

int classnum, i, j, k;

in >> classnum;

Schedule scheduleTable(classnum);

int sum = 0;

for (i = 0; i < 8; i++)

{

in >> scheduleTable.singleTerm[i].courseNum;

sum += scheduleTable.singleTerm[i].courseNum;

scheduleTable.singleTerm[i].buffer = new Course\*[scheduleTable.singleTerm[i].courseNum];

}

while (in.get() != '\n') continue;

if (sum != classnum)

{

cout << "输入出错，请检查文件" << endl;

system("pause");

return 0;

}

for (i = 0; i < classnum; i++)

{

string s, courseNum, pre;

getline(in, s);

stringstream SS(s);

SS >> courseNum;

Course& c = scheduleTable.courses[courseNum];

c.m\_number = courseNum;

SS >> c.m\_name >> c.classHours >> c.openTerm;

if (c.openTerm)

{

if (scheduleTable.singleTerm[c.openTerm - 1].addCourse(&c))

{

continue;

}

else

{

cout << "无法正常排课" << endl;

system("pause");

return 0;

}

}

while (SS >> pre)

{

scheduleTable.courses[pre].addNext(&c);

c.in++;

}

if (c.in == 0)

{

scheduleTable.checkReady.push(&c);

}

}

in.close();

scheduleTable.arrange();

for (i = 0; i < 8; i++)

{

out << "第" << i + 1 << "学期课表为:" << endl;

out << " 周一 周二 周三 周四 周五" << endl;

for (j = 0; j < 10; j++)

{

out << "第" << j + 1 << "节" << " ";

for (k = 0; k < 5; k++)

{

out << scheduleTable.singleTerm[i].termTable[k][j] << " ";

}

out << endl;

}

}

out.close();

cout << "课表已安排完毕" << endl;

system("pause");

return 0;

}

4 测试

4.1 正常测试

**测试用例**：

38

5 5 5 2 5 5 6 5

c01 程序设计基础 5 0

c02 离散数学 6 0 c01

c03 数据结构算法 4 0 c01 c02

c04 汇编语言 5 0 c01

c05 算法设计 4 0 c03 c04

c06 计算机组成原理 6 0

c07 微机原理 4 0 c03

c08 单片机应用 3 0 c03

c09 编译原理 5 0 c03

c10 操作系统原理 4 0 c03

c11 数据库原理 5 0 c03

c12 高等数学 6 0

c13 线性代数 6 0

c14 数值分析 6 0 c12

c15 普通物理 4 0 c12

c16 计算机文化 3 0

c17 计算机系统结构 6 0 c06

c18 计算机网络 5 0 c03

c19 数据通信 6 0

c20 面向对象程序设计 3 0 c01 c03

c21 Java 3 0 c01 c03

c22 C#.net 5 0 c01 c03

c23 PowerBuilder 5 0 c01 c03

c24 VC++ 3 0 c01 c03

c25 ASP程序设计 5 0 c01 c03

c26 JSP程序设计 5 0 c01 c03

c27 VB.net 5 0 c01 c03

c28 Delphi 5 0 c01 c03

c29 C++Builder 5 0 c01 c03

c30 英语 5 1

c31 英语 5 2

c32 英语 5 3

c33 英语 5 4

c34 英语 5 5

c35 英语 5 6

c36 英语 5 7

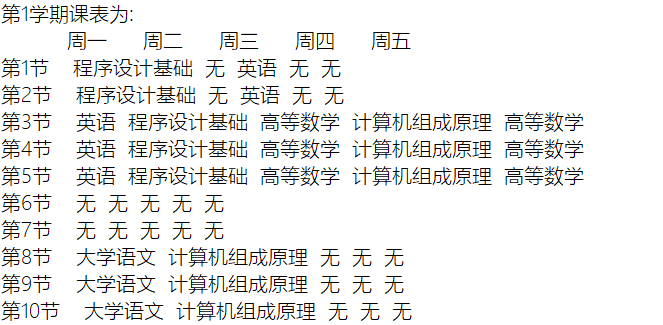
c37 英语 5 8

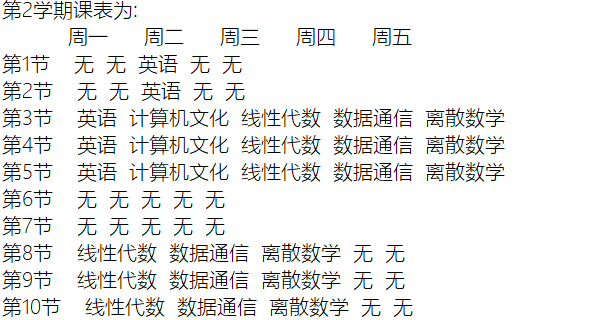
c38 大学语文 3 1

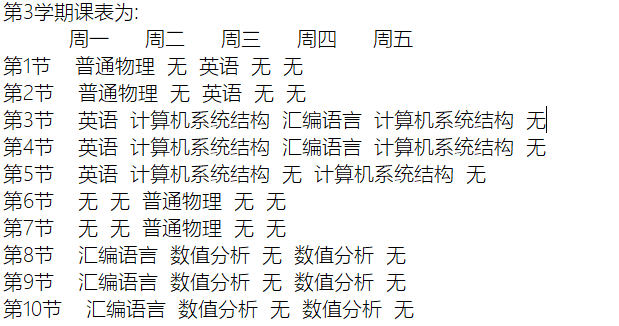
**预期结果**：

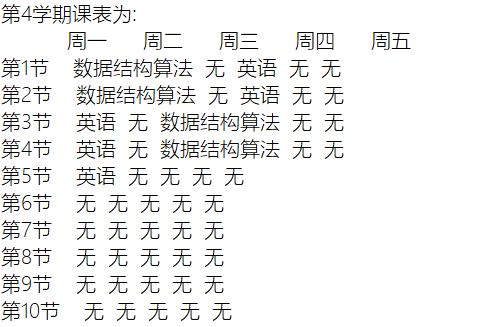
排课成功，out.txt文件中写入八个学期的课表

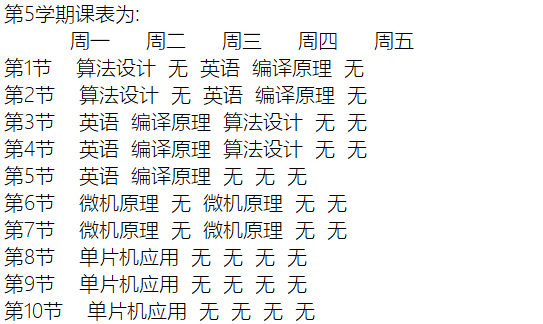
**实验结果：**

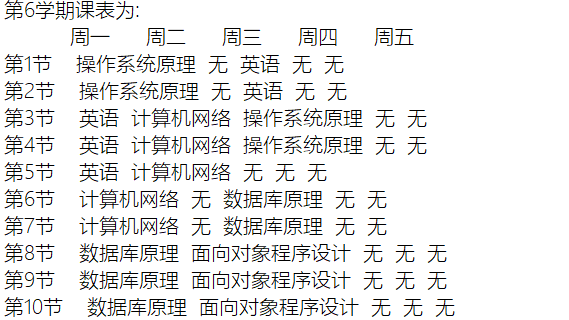


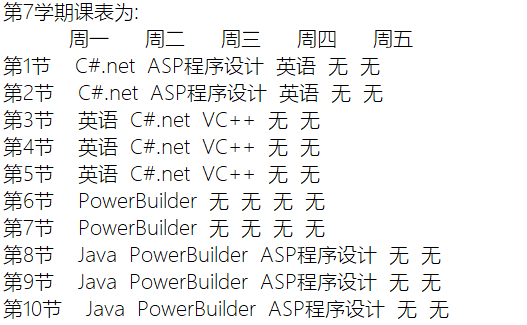


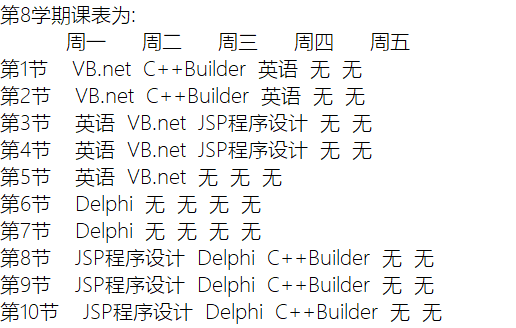












4.2 错误测试

4.2.1 每学期开课数之和不为课程总数

**测试用例：**

7

1 1 1 1 1 1 1 1

c01 程序设计基础 5 0

c02 离散数学 6 0 c01

c03 数据结构算法 4 0 c01 c02

c04 汇编语言 5 0 c01

c05 算法设计 4 0 c03 c04

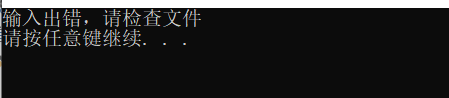
c06 计算机组成原理 6 0

c07 微机原理 4 0 c03

c08 单片机应用 3 0 c03

**预期结果：**给出错误提示，输出文件为空。

**实验结果：**



4.2.2 课程存在循环依赖

**测试用例：**

8

1 1 1 1 1 1 1 1

c01 程序设计基础 5 0 c02

c02 离散数学 6 0 c01v c03

c03 数据结构算法 4 0 c04

c04 汇编语言 5 0 c01 c05

c05 算法设计 4 0 c03 c06

c06 计算机组成原理 6 0 c07

c07 微机原理 4 0 c03 c08

c08 单片机应用 3 0 c01

**预期结果：**给出错误提示，输出文件为空

**实验结果：**

