



数 据 结 构 与 算 法 课 程

教学计划编制系统的设计与实现

The Design and Realization of The Teaching Planning System

姓 名：赵哲冕

学 号：17420182200850

学 院： 管理学院

专 业： 管理科学

年 级： 2018 级

二〇二〇年六月十八日

目录

第一章 问题描述与主旨	4
第一节 问题描述	4
第二节 问题主旨	4
第二章 教学计划编制系统需求分析	4
第一节 课程安排需求	4
2.1.1 课程安排需求概述	4
2.1.2 课程安排需求分析	4
第二节 时间安排需求	5
2.2.1 时间安排需求概述	5
2.2.2 时间安排需求分析	5
第三章 教学计划编制系统设计思路	5
第一节 数据结构设计	5
3.1.1 栈的链式存储	5
3.1.2 图的邻接表存储	6
第二节 核心算法设计	7
3.2.1 拓扑排序	7
3.2.2 课程安排策略	9
第三节 关键部分函数模块及其调用图	11
3.3.1 关键部分函数模块	11
3.3.2 关键部分函数模块调用图	12
第四章 教学计划编制系统实现代码	13
第一节 数据结构代码	13
4.1.1 栈的链式存储	13
4.1.2 图的邻接表存储	13
第二节 核心算法代码	14
4.2.1 拓扑排序算法	14
4.2.2 课程安排策略	16

第五章 教学计划编制系统程序测试.....	16
第一节 测试概述	16
第二节 测试数据	16
第三节 输入输出定义	18
第四节 测试过程	20
第五节 测试结果分析	22
第六章 附录	23
第一节 参考书目	23
第二节 源代码	23

第一章 问题描述与主旨

第一节 问题描述

设计一个教学计划编制程序，要求满足如下条件：

大学的每个专业都要制定教学计划。假设任何专业都有固定的学习年限，每学年含两个学期，每学期的时间长度和学分上限值均相等。每个专业开设的课程都是确定的，而且课程在开设时间的安排必须满足先修关系。每门课程由哪些先修课程是确定的，可以有任意多门，也可以没有。每门课恰好占一个学期。

第二节 问题主旨

教学计划的制定是大学每个专业教学的重要组成部分，教学计划制定的合理与否直接关系到该专业学生的教育质量的高低。因此，教学计划的制定理应按照一个项目工程来对待。所有的课程之间，通常都受到一定条件的约束，如每学期学分有一定上限，其中某些课程需在另一些课程完成之后才能开始等。这样一来，在进行设计教学计划编制程序的时候，一定要将所有课程的前驱与后继关系表现出来，最终结果构成一个无环的有向图。

第二章 教学计划编制系统需求分析

第一节 课程安排需求

2.1.1 课程安排需求概述

1. 每个专业开设的课程确定，一共 12 门课程；
2. 每门课程的学分确定，分别为：2, 3, 4, 3, 2, 3, 4, 4, 7, 5, 2, 3，每个学期学生修读的学分上限为 10 学分；
3. 课程开设时间的安排满足前驱与后续关系，某些（个）课程必须在另一些（个）课程完成后才能开始；
4. 生成课程安排的前驱后继图。

2.1.2 课程安排需求分析

1. 针对第一点需求，这是该程序的基础需求，在设计过程中会使用到图的数据结构，因此固定的 12 门课程也规定了程序设计时图的顶点为 12 个。

2. 针对第二点需求，在程序设计过程中需要考虑每学期的学分限制，因此要在满足第三点需求的前提下，保证每学期修读的总学分不能超过规定的 10 分。

3. 针对第三点需求，这是该程序的核心，具体的先修课程安排由程序设计者进行安排，然后需要利用拓扑排序算法来实现在满足前驱与后继的前提下的排序。

4. 针对第三点需求，编制展示函数，显示图各顶点之间的关系，目的是让最终结果更加直观，增强程序的可读性。

第二节 时间安排需求

2.2.1 时间安排需求概述

1. 专业的固定学习年限为 4 年，即 8 个学期；
2. 要求课程安排尽量集中在前三个学年，即前 6 个学期。

2.2.2 时间安排需求分析

1. 针对第一点需求，这是该程序的基础需求，需要在程序设计时进行设定；
2. 针对第二点需求，这是该程序较为重要的核心部分，需要在满足学分限定的条件下，设计一种算法：从第一个学期开始尽可能地使每学期修读的学分总和等于 10，从而完成将课程安排集中在前三个学年的要求。

第三章 教学计划编制系统设计思路

第一节 数据结构设计

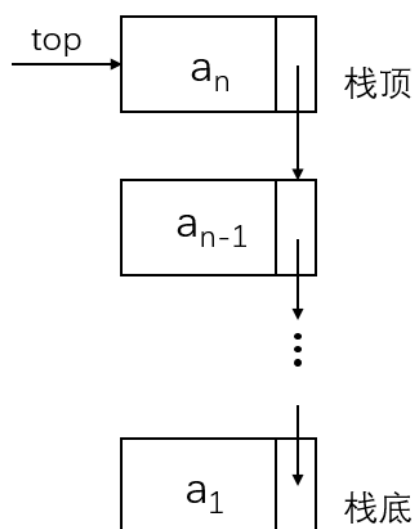
3.1.1 栈的链式存储

在设计教学计划编制系统的过程中，核心算法部分需要调用栈来存储入度为 0 的顶点。在此我选取了栈的顺序存储结构作为栈的数据结构，如下表所示：

栈的元素	链接指针
ElemType data	struct Node *next;

栈顶指针
PNode top;

该数据结构通常用指针来实现栈的链式存储，设一个 top 变量来指示栈顶元素在链表中的位置。对于链栈来说，基本不存在栈满的情况，除非内存已经没有可用空间。对于空栈来说，链栈的空其实就是 top=NULL。栈的结构如下图所示：



3.1.2 图的邻接表存储

在设计教学计划编制系统的过程中，核心算法为拓扑排序算法，该算法是一个对有向图构造拓扑序列的过程，因此需要确定该有向图所使用的数据结构。由于在拓扑排序的过程中，需要进行删除结点的操作，而这一操作对于邻接矩阵来说并不灵活，因此使用邻接表更加方便。具体设计如下：

1. 邻接表的边表结点的链式存储结构

边表顶点下标	边表课程编号	指向下一个邻接点的指针
int adjvex	char E_data[MAX_NO]	struct EdgeNode* next

2. 邻接表的顶点表结点的链式存储结构

顶点表课程编号	学分信息	边表的头指针
char V_data[MAX_NO]	int grade	EdgeNode* firstedge

3. 图的邻接表结构

邻接表数组	图中当前顶点数
AdjList adjList	int numVertexes

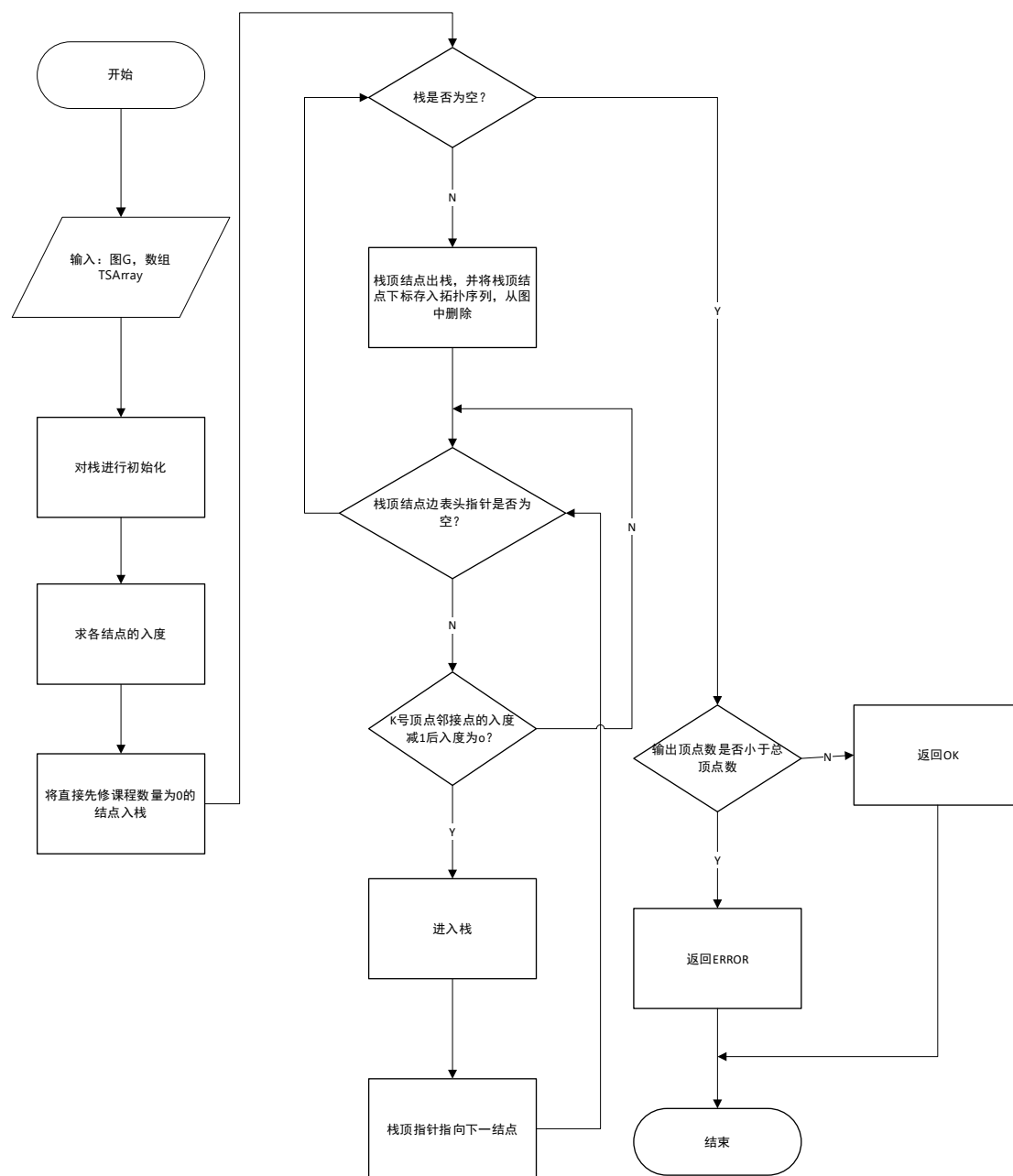
第二节 核心算法设计

3.2.1 拓扑排序

对 AOV 网进行拓扑排序的基本思路是：从 AOV 网中选择一个入度为 0 的顶点输出，然后删去此顶点，并删去以此顶点为尾的弧，继续重复此步骤，直到输出全部顶点或者 AOV 网中不存在入度为 0 的顶点为止。将课程结点进行拓扑排序得到拓扑序列的算法描述如下：

- (1) 对栈进行初始化
- (2) 求各结点的入度
- (3) 将直接先修课程数量为 0 的结点入栈
- (4) 判断栈是否为空，是则跳转 (9) 继续执行，否则跳转 (5) 继续执行
- (5) 栈顶结点出栈，并将栈顶结点的下标存入拓扑序列，并从图中删除
- (6) 遍历栈顶结点的直接后续课程链表，若此结点直接先修课程数量减 1 后入度为 0，并进入栈
- (7) 继续遍历下一直接后续课程链表，直到指针为空
- (8) 判断栈是否为空，是则跳转 (9) 继续执行，否则跳转 (5) 继续执行
- (9) 判断输出顶点数是否小于总顶点数，若小于则说明存在环，返回 ERROR，否则返回 OK；

算法的执行流程图如图所示：



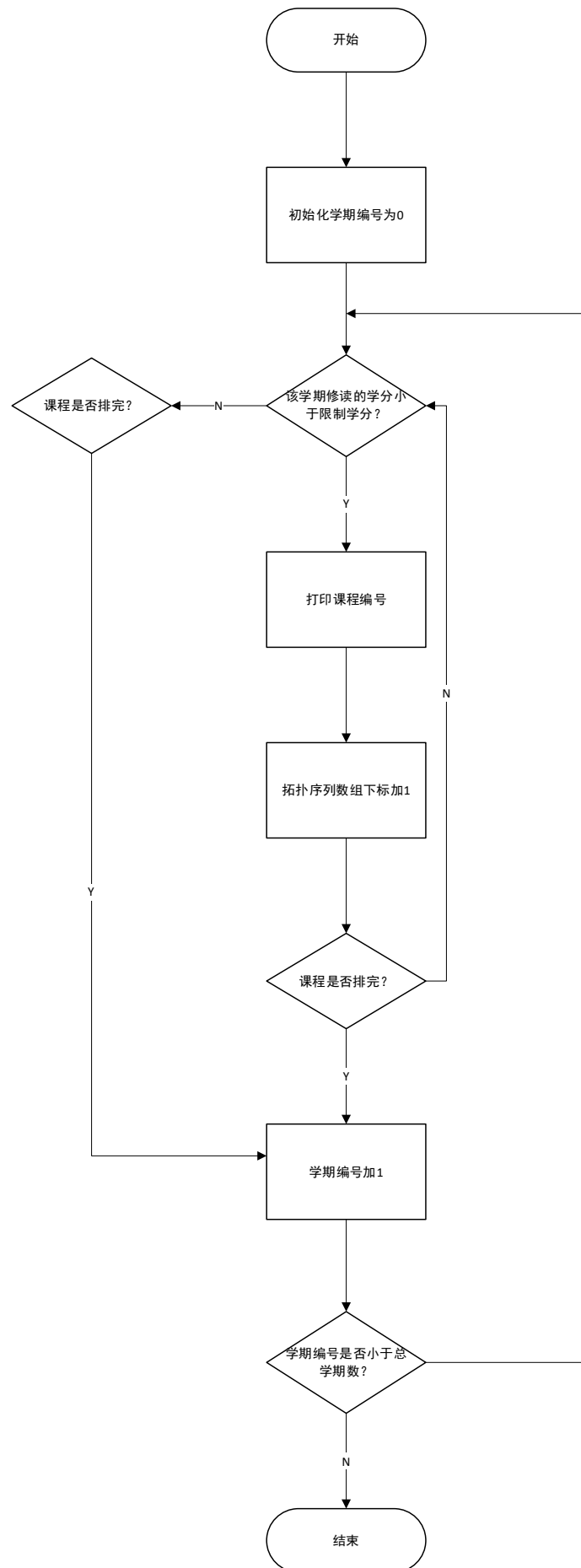
算法时间复杂度分析：

对于一个具有 n 个顶点 e 条弧的 AOV 网来说，整个算法的第（2）步——将入度为 0 的结点入栈操作的时间的时间复杂度为 $O(n)$ ，而之后的 while 循环中，每个顶点进一次栈，出一次栈，入度减 1 的操作供执行的 e 次，所以整个算法的时间复杂度为 $O(n+e)$ 。

3.2.2 课程安排策略

生成拓扑排序后，需要对结果进行排课，实现使课程尽可能地集中在前三个学年，大四安排较少的课程的功能。具体的算法描述如下：

- (1) 初始化学期编号为 0
- (2) 判断该学期修读的学分是否小于限制学分，若是则跳转至 (3)，否则继续判断课程是否排完，若排完则跳转至 (6)
- (3) 打印课程编号
- (4) 拓扑排序数组下标加 1
- (5) 判断课程是否排完，若是则继续执行
- (6) 学期编号加 1
- (7) 判断学期编号是否小于学期数，若是则跳转至 (2)，否则结束循环并退出



算法时间复杂度分析：

算法的第 1~n 个学期循环的过程中需进行学分是否小于限制学分的循环判断，因此算法的时间复杂度为 $O(n^2)$ 。

第三节 关键部分函数模块及其调用图

3.3.1 关键部分函数模块

`PLinkStack CreateEmptyStack(void)`：建立一个空栈

`Status IsEmptyStack(PLinkStack plstack)`：判断是否为空栈

`void PopStack(PLinkStack plstack)`：出栈操作

`void PushStack(PLinkStack plstack, int x)`：进栈操作

`Status GetTop(PLinkStack plstack)`：取栈顶元素

`GraphAdjList CreateGraph_List()`：建立邻接表

`void FindIndegree(GraphAdjList g, int* indegree)`：求各顶点的入度

`Status Locate(GraphAdjList g, char* vertex)`：确定顶点在顶点表中的下标

`Status AddVex(GraphAdjList g, char* vertex, int grade)`：在顶点表中增加结点

`Status AddEdge(GraphAdjList g, char* vertex, char* adjv)`：在边表中增加结点

`Status IsNullGraph_list(GraphAdjList g)`：判断图是否为空

`Status TopologicalSort(GraphAdjList G, int* TSortArray)`：拓扑排序算法

`void RelationshipMap(GraphAdjList g, int* TSortArray)`：课程安排的前驱后继图

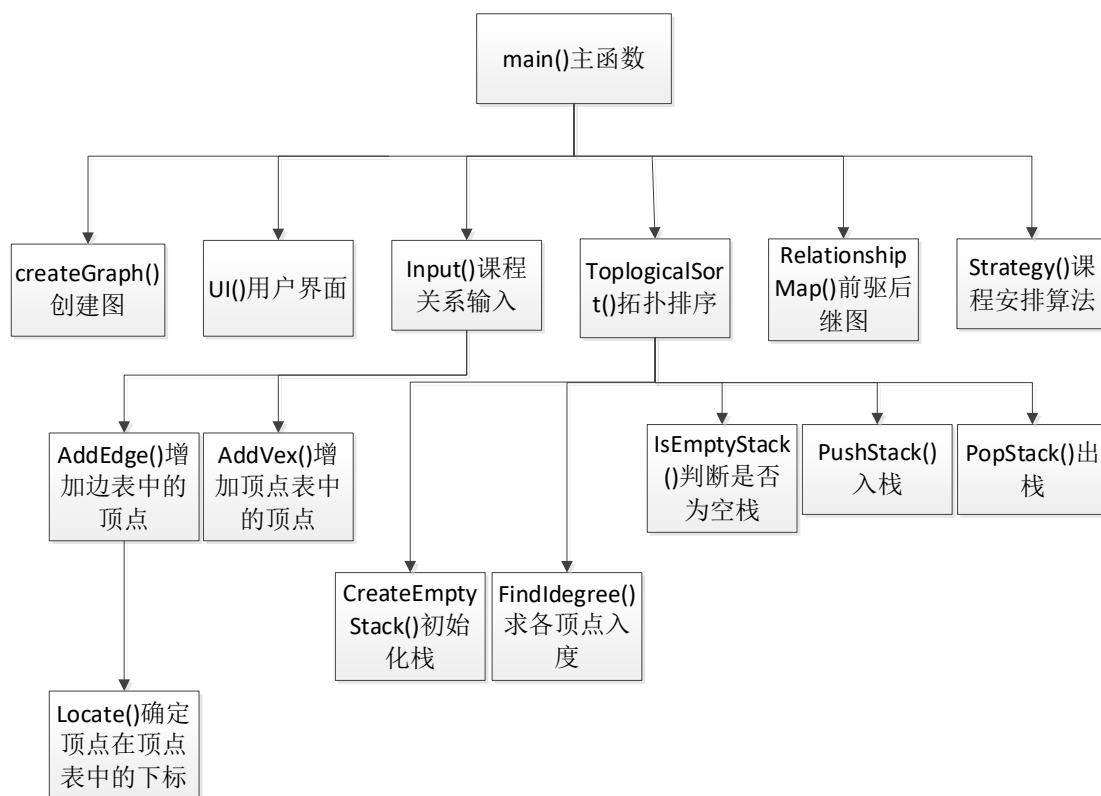
`void Strategy(GraphAdjList g, int* TSortArray)`：课程安排算法

`Status Input(GraphAdjList g)`：课程关系的手动输入

`Status UI(GraphAdjList g)`：用户界面设计

`int main()`：主函数

3.3.2 关键部分函数模块调用图



第四章 教学计划编制系统实现代码

第一节 数据结构代码

4.1.1 栈的链式存储

由于栈的链式存储是学习过程中较为基础的一种数据结构,故不再做过多的解释,仅提供下列代码:

```
typedef struct Node
{
    ElemType data;//栈的元素
    struct Node *next;//链接指针
}Node,*PNode;

typedef struct LinkStack
{
    PNode top;//栈顶指针
}LinkStack,*PLinkStack;
```

4.1.2 图的邻接表存储

1. 邻接表的边表结点的数据类型定义如下:

```
typedef struct EdgeNode /*边表结点*/
{
    int adjvex;          //邻接点域,存储该顶点对应的下标
    char E_data[MAX_NO]; //边表的课程编号
    struct EdgeNode* next; //指向下一个邻接点
}EdgeNode;
```

对边表结点的数据内容解释如下:

adjvex, 整数类型, 表示边表的邻接点域, 用来存储课程顶点在顶点表数组中的对应下标。

E_data[MAX_NO], 字符型数组, 用来存储对应的课程编号。

next, 指针类型, 表示该课程的直接后继课程链表的指针。

2. 邻接表的顶点表结点的数据类型定义如下：

```
typedef struct VertexNode /*顶点表结点*/
{
    char V_data[MAX_NO];    //存储顶点信息，即课程编号
    int grade;               //存储学分信息
    EdgeNode* firstedge;    //边表的头指针
}VertexNode, * AdjList;
```

对顶点表结点的数据内容解释如下：

V_data[MAX_NO]，字符型数组，用来存储课程编号信息。

grade，整数类型，用来存储课程的学分信息。

firstedge，用于指向顶点边表的头指针。

3. 图的邻接表结构的数据类型定义如下：

```
typedef struct /*邻接表*/
{
    AdjList adjList;        //邻接表数组
    int numVertexes;        //图中当前顶点数
}graphAdjList, * GraphAdjList;
```

对图的邻接表的数据内容解释如下：

adjList，邻接表数组。

numVertexes，整数类型，用于表示图中当前顶点数。

第二节 核心算法代码

在教学计划编制系统中，核心的算法即课程图的拓扑排序以及排序后的排课算法，这里给出算法的实现代码。

4.2.1 拓扑排序算法

```
Status TopologicalSort(GraphAdjList G, int* TSArray)
{
    PEdgeNode p;
```

```

EdgeNode* e;
int i, j, k, nodeno = 0;           //拓扑序列数组中元素的个数
PLinkStack plstack = CreateEmptyStack();

indegree = (int*)malloc(sizeof(int) * G->numVertexes);           //存储各节点的入度
的数组
FindIndegree(G, indegree);           //求各节点的入度

for (i = 0; i < G->numVertexes; i++)
{
    if (indegree[i] == 0)
    {
        PushStack(plstack, i);           //先将所有入度为零的节点的下标
压入栈中
    }
}

while (!IsEmptyStack(plstack))
{
    //当栈不为空时
    j = GetTop(plstack);           //取栈顶元素
    PopStack(plstack);           //出栈
    TArray[nodeno++] = j;           // 将下标存入拓扑序列数组,
个数自加

    for (k = 0; k < G->numVertexes; k++)
    {
        //查找下标为 j 的节点的后继
        p = G->adjList[k].firstedge;           //顶点 k 的边表头指针
        while (p != NULL)
        {
            if (p->adjvex == j)
            {
                //j 为入度为零元素的下标, 当相等时, vex[k]就是该元素的后继
                indegree[k] = indegree[k] - 1; //将该元素的入度减一
                if (indegree[k] == 0)
                {
                    //减一后为零就压入栈中
                    PushStack(plstack, k);
                }
            }

            p = p->next;
        }
    }
}

free(indegree);           //释放空间

```

```

        free(plstack);
        if (nodeno < G->numVertexes) {    //如果拓扑序列数组元素个数小于 AOV 网的
顶点数, 说明拓扑排序失败
            printf("警告: 该 AOV 网中含有回路! 请检查输入的数据! \n");
            return ERROR;
        }
        system("cls");
        return OK;
    }

```

4.2.2 课程安排策略

```

void Strategy(GraphAdjList g, int* TSMArray)
{
    int i, j, grade;
    for (i = 0, j = 0; i < terms; i++)//第 1-8 个学期开始循环
    {
        grade = 0;
        printf("第%d 个学期的课程为: ", i + 1);
        while ((grade += g->adjList[TSMArray[j]].grade) <= gradelimits)//当该学期修的
学分小于限制学分时进行排课
        {
            printf("%s ", g->adjList[TSMArray[j]].V_data);//打印课程编号
            j++;
            if (j > g->numVertexes - 1) break;//如果课程排完了则退出
        }
        if (j > g->numVertexes - 1) break;//如果课程排完了则退出
        printf("\n");
    }
    printf("\n-----\n");
    printf("\n-----\n");
}

```

第五章 教学计划编制系统程序测试

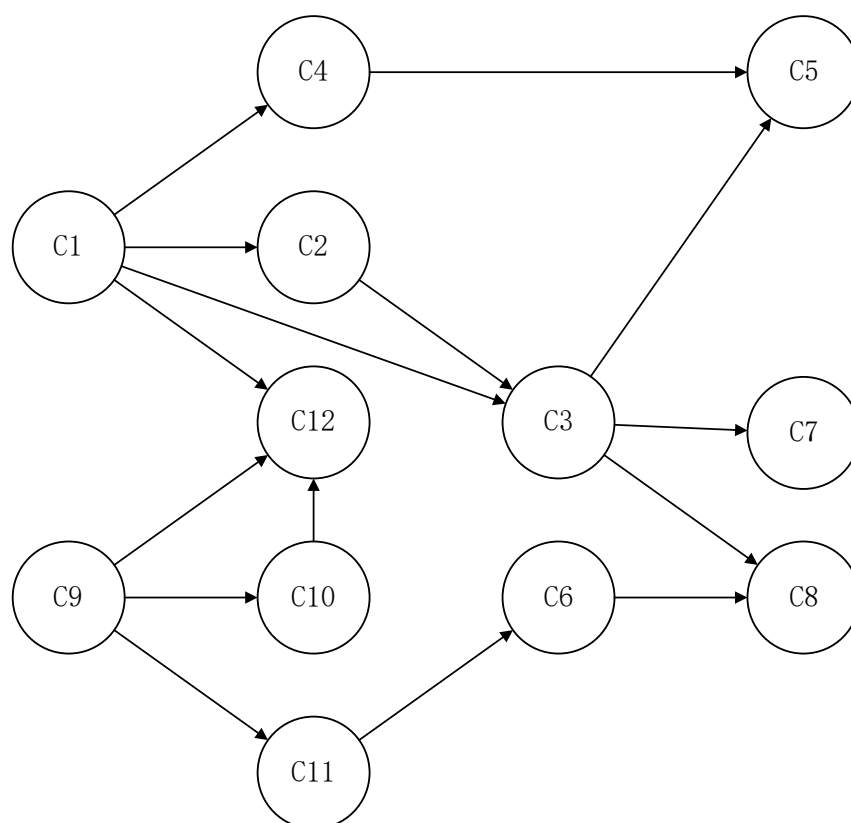
第一节 测试概述

程序的测试与查错是调试程序的必要步骤, 是保证程序可靠性的关键。在程序编码结束后, 程序的作者需要对程序进行必要的测试, 以验证程序是否满足需求分析、系统设计中的相关要求, 同时也是对代码的有力审查。另外, 程序的测试需要尽可能找到程序中存在的漏洞, 并给开发者提出对于程序的改进以及优化之处。

第二节 测试数据

依据题目所提供的数据，我将程序测试数据整理如下：

- (1) 学期总数：8
- (2) 每学期学分上限：10
- (3) 供开设课程数：12
- (4) 课程号：C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12
- (5) 学分顺序：2, 3, 4, 3, 2, 3, 4, 4, 7, 5, 2, 3
- (6) 先修关系：



(7) 测试数据汇总表

课程编号	学分	先修课程数量	先修课程
C1	2	0	无
C2	3	1	C1
C3	4	2	C1, C2
C4	3	1	C1
C5	2	2	C3, C4
C6	3	1	C11
C7	4	2	C5, C3

C8	4	2	C3, C6
C9	7	0	无
C10	5	1	C9
C11	2	1	C9
C12	3	3	C9, C10, C1

第三节 输入输出定义

该程序的运行环境为 Microsoft Windows10 操作系统，开发环境为 Visual Studio 2019。进入程序后 Visual Studio 会自动进行符号的加载，加载就绪后即显示系统用户界面。如下图所示。



对于本程序的测试，定义了规范的输入域输出，以便对测试结果进行较好的分析。

测试的输入定义如下：

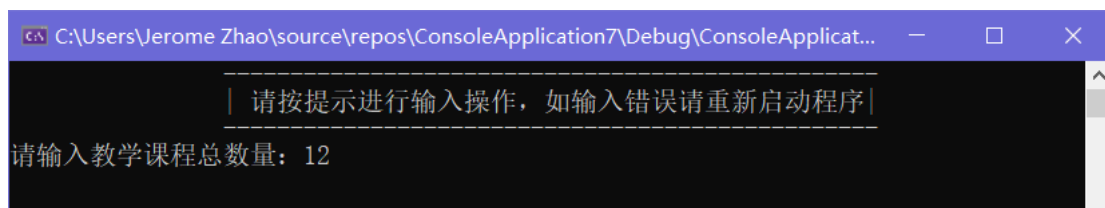
- (1) 输入教学课程总数量：该步骤请输入一个整数类型的数值，表示该专业所有教学课程的总数量，以回车键结束。
- (2) 输入课程号、课程学分、课程的先修课程数量、课程的先修课程号：当输入的课程数小于教学课程的总数量时，循环输入该步骤，并以回车键结束，其中课程号均为字符串类型变量，课程数量为整数类型变量。
- (3) 输入学期总数、每学期学分上限：该步骤均为整数类型的变量，输入后以回车键结束。

测试的输出定义如下：

- (1) 输出课程安排的前驱后继关系图。
- (2) 输出该课程安排的排序结果。
- (3) 输出每个学期的课程安排具体内容。

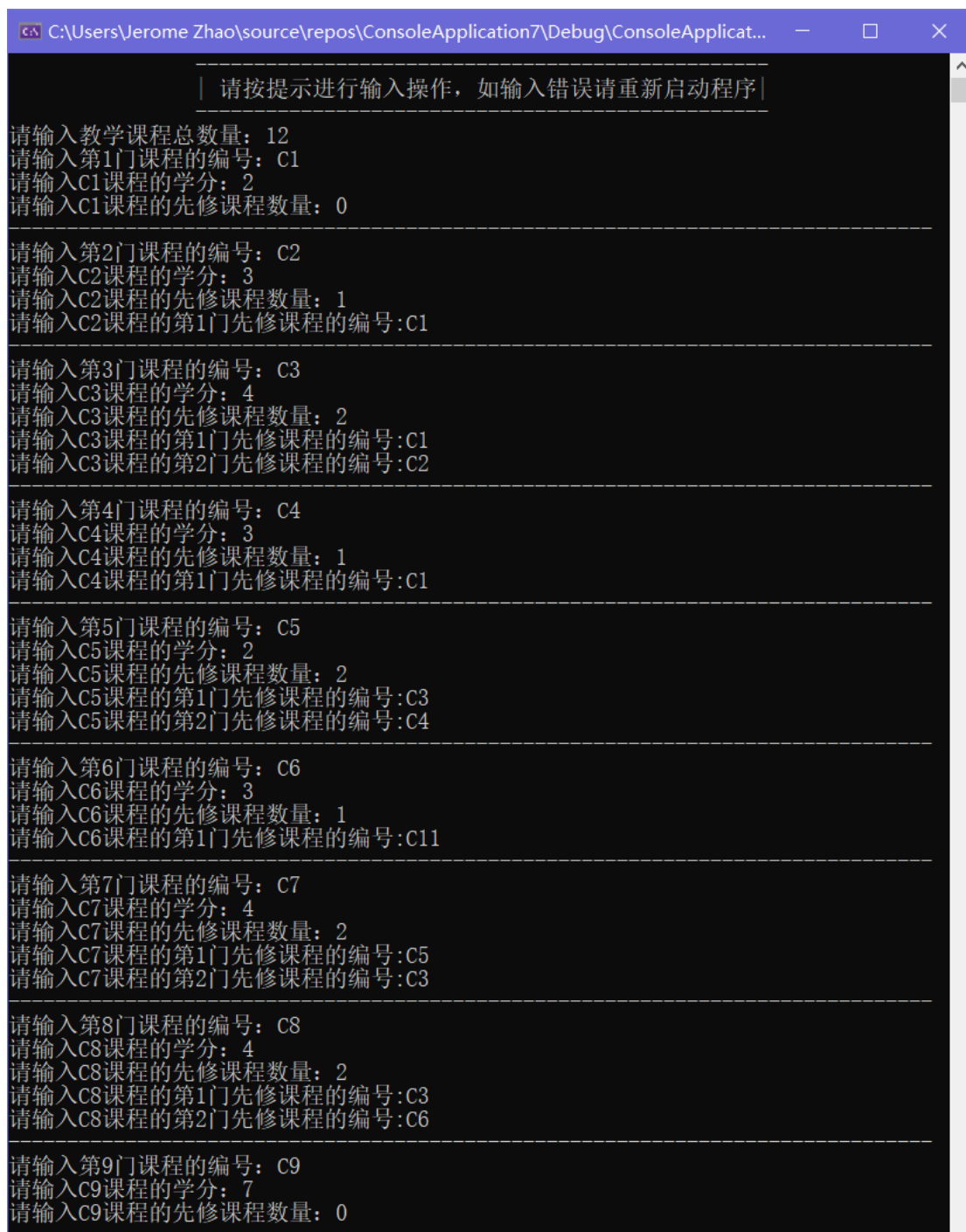
第四节 测试过程

1. 输入教学课程总数量:



A screenshot of a Windows console application window. The title bar shows the file path: C:\Users\Jerome Zhao\source\repos\ConsoleApplication7\Debug\ConsoleApplicat... The window contains a dashed border with the text: | 请按提示进行输入操作，如输入错误请重新启动程序|. Below this, the prompt '请输入教学课程总数量: 12' is displayed.

2. 输入课程号、课程学分、课程的先修课程数量、课程的先修课程号:



A screenshot of a Windows console application window showing the input of course details. The title bar shows the file path: C:\Users\Jerome Zhao\source\repos\ConsoleApplication7\Debug\ConsoleApplicat... The window contains a dashed border with the text: | 请按提示进行输入操作，如输入错误请重新启动程序|. Below this, the following prompts are displayed: '请输入教学课程总数量: 12', '请输入第1门课程的编号: C1', '请输入C1课程的学分: 2', '请输入C1课程的先修课程数量: 0', '请输入第2门课程的编号: C2', '请输入C2课程的学分: 3', '请输入C2课程的先修课程数量: 1', '请输入C2课程的第1门先修课程的编号: C1', '请输入第3门课程的编号: C3', '请输入C3课程的学分: 4', '请输入C3课程的先修课程数量: 2', '请输入C3课程的第1门先修课程的编号: C1', '请输入C3课程的第2门先修课程的编号: C2', '请输入第4门课程的编号: C4', '请输入C4课程的学分: 3', '请输入C4课程的先修课程数量: 1', '请输入C4课程的第1门先修课程的编号: C1', '请输入第5门课程的编号: C5', '请输入C5课程的学分: 2', '请输入C5课程的先修课程数量: 2', '请输入C5课程的第1门先修课程的编号: C3', '请输入C5课程的第2门先修课程的编号: C4', '请输入第6门课程的编号: C6', '请输入C6课程的学分: 3', '请输入C6课程的先修课程数量: 1', '请输入C6课程的第1门先修课程的编号: C11', '请输入第7门课程的编号: C7', '请输入C7课程的学分: 4', '请输入C7课程的先修课程数量: 2', '请输入C7课程的第1门先修课程的编号: C5', '请输入C7课程的第2门先修课程的编号: C3', '请输入第8门课程的编号: C8', '请输入C8课程的学分: 4', '请输入C8课程的先修课程数量: 2', '请输入C8课程的第1门先修课程的编号: C3', '请输入C8课程的第2门先修课程的编号: C6', '请输入第9门课程的编号: C9', '请输入C9课程的学分: 7', '请输入C9课程的先修课程数量: 0'.

```

请输入第10门课程的编号: C10
请输入C10课程的学分: 5
请输入C10课程的先修课程数量: 1
请输入C10课程的第1门先修课程的编号: C9
-----
请输入第11门课程的编号: C11
请输入C11课程的学分: 2
请输入C11课程的先修课程数量: 1
请输入C11课程的第1门先修课程的编号: C9
-----
请输入第12门课程的编号: C12
请输入C12课程的学分: 3
请输入C12课程的先修课程数量: 3
请输入C12课程的第1门先修课程的编号: C9
请输入C12课程的第2门先修课程的编号: C10
请输入C12课程的第3门先修课程的编号: C1
    
```

3. 输出该课程安排的前驱后继关系图

```

该课程安排的前驱后继关系图:
C1---->C2, C3, C4, C12
C2---->C3
C3---->C5, C7, C8
C4---->C5
C5---->C7
C6---->C8

C9---->C10, C11, C12
C10---->C12
C11---->C6
    
```

4. 输出该课程安排的排序方案

```

该课程安排的排序方案为:
C9 C11 C10 C1 C12 C4 C2 C3 C5 C7 C6 C8
-----
                    请按任意键继续...
    
```

5. 输入学期总数、每学期学分上限

```

请输入学期总数: 8
请输入每学期学分上限: 10
    
```

6. 输出每个学期的课程安排具体内容

```

第1个学期的课程为:  C9 C11
第2个学期的课程为:  C10 C1 C12
第3个学期的课程为:  C4 C2 C3
第4个学期的课程为:  C5 C7 C6
第5个学期的课程为:  C8
    
```

教学计划编制系统运行完毕!

请按任意键退出。

第五节 测试结果分析

在程序调试的过程中，遇到了许多常见的语法错误和逻辑错误，这需要我们不断地调试和分析。经过长时间的编码和调试，最终得到了每个学期的教学计划，现将结果分析如下：

该程序满足了题目所要求的限制条件，并成功地输出了相关数据：

- (1) 结果满足课程开设时间安排的前驱与后续关系，即某些（个）课程必须在另一些（个）课程完成后才能开始
- (2) 结果满足学期总数与学分上限的限制
- (3) 结果满足尽量将课程安排在前三个学年的限制

但同时，该程序也存在许多不足之处：

- (1) 在使用 Visual Studio 2019 对程序进行第一次编译时，通常会在用户界面和输入界面等待编译器从微软符号服务器加载相关符号的.pdb 文件，运行速度较慢的硬件可能等待时间较长。
- (2) 在输入课程号、学分和先修关系的过程中用户需大量输入数据，而这样输入极易出错，每次出错都需要重新运行程序，较为不便。针对该问题，我也思考过改进之处，但是由于我个人才疏学浅，依靠目前所掌握的知识难以实现文件输入等操作，因此希望能够在不久的将来继续改进该程序，最终解决用户输入不便的问题。

第六章 附录

第一节 参考书目

[1] 严蔚敏, 吴伟民. 《数据结构 (C 语言版)》. 清华大学出版社, 2014

[2] Stephen Prata 著, 姜佑译. 《C Primer Plus 中文版 (第六版)》. 人民邮电出版社, 2019

第二节 源代码

```
// 17420182200850赵哲冕第12题.cpp : 此文件包含 "main" 函数。
/*   Preparation of teaching plan
   教学计划编制系统——design by 赵哲冕*/

#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<math.h>

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR -1
#define OVERFLOW -1
#define MAX_NO 4           //最大编号值
#define MAX_VERTEX_NUM 100 //最大顶点数

int terms;//学期数
int gradelimits;//每学期学分限制
int* indegree;//入度数组

typedef char VertexType;
typedef int ElemType;
typedef int Status;

/*****-----数据结构部分-----*****/
/* -----栈的链式存储----- */
typedef struct Node
{
    ElemType data;//栈的元素
    struct Node* next;//链接指针
}Node, * PNode;
typedef struct LinkStack
```

```

{
    PNode top;//栈顶指针
}LinkStack, * PLinkStack;

/* -----栈的操作----- */
//建一个空栈
PLinkStack CreateEmptyStack(void)
{
    PLinkStack plstack;

    plstack = (PLinkStack)malloc(sizeof(struct LinkStack));
    if (plstack != NULL)
        plstack->top = NULL;
    else
        printf("Out of space!");
    return plstack;
}

//判断是否为空栈
Status IsEmptyStack(PLinkStack plstack)
{
    return(plstack->top == NULL);
}

//出栈操作
void PopStack(PLinkStack plstack)
{
    PNode p;

    if (IsEmptyStack(plstack))
    {
        printf("Stack Is Empty!\n");
    }
    else
    {
        p = plstack->top;
        plstack->top = plstack->top->next;
        free(p);
    }
}

//进栈操作
void PushStack(PLinkStack plstack, int x)
{

```



```

PNode p;
p = (PNode)malloc(sizeof(struct Node));

if (p == NULL)
{
    printf("Out Of Space!");
}
else
{
    p->data = x;
    p->next = plstack->top;
    plstack->top = p;
}
}

//取栈顶元素
Status GetTop(PLinkStack plstack)
{
    if (IsEmptyStack(plstack))
    {
        printf("Stack Is Empty!");
    }
    else
        return (plstack->top->data);
}

/*-----邻接表的构建-----*/
typedef struct EdgeNode* PEdgeNode;
typedef struct EdgeNode //边表结点
{
    int adjvex;           //邻接点域，存储该顶点对应的下标
    char E_data[MAX_NO]; //边表的课程编号
    struct EdgeNode* next; //指向下一个邻接点
}EdgeNode;

typedef struct VertexNode //顶点表结点
{
    char V_data[MAX_NO]; //存储顶点信息，即课程编号
    int grade;           //存储学分信息
    EdgeNode* firstedge;  //边表的头指针
}VertexNode, * AdjList;

typedef struct /*邻接表*/
{

```

```

AdjList adjList;           //邻接表数组
int numVertexes;           //图中当前顶点数
}graphAdjList, * GraphAdjList;

/*-----邻接表的操作-----*/
GraphAdjList CreateGraph_List()
{
    //创建一个空图
    GraphAdjList g = (GraphAdjList)malloc(sizeof(graphAdjList));

    if (g == NULL)
    {
        printf("OVERFLOW!");
    }
    else
    {
        g->adjList = (AdjList)malloc(sizeof(VertexNode) * MAX_VERTEX_NUM);
        if (g->adjList)
        {
            g->numVertexes = 0;
            return g;
        }
        else
            free(g);
    }

    return 0;
}

//求各顶点的入度
void FindIndegree(GraphAdjList g, int* indegree)
{
    int i;
    PEdgeNode p;

    for (i = 0; i < g->numVertexes; i++) indegree[i] = 0; //先初始化，全为零
    for (i = 0; i < g->numVertexes; i++) {
        p = g->adjList[i].firstedge;
        while (p != NULL) {
            indegree[i]++;
            p = p->next;
        }
    }
}

```

```
}
```

//确定顶点在顶点表中的下标

```
Status Locate(GraphAdjList g, char* vertex)
{
    int i;

    for (i = 0; i < g->numVertexes; i++)
    {
        if (strcmp(g->adjList[i].V_data, vertex) == 0)
            break;
    }

    return i;
}
```

//在顶点表中增加结点

```
Status AddVex(GraphAdjList g, char* vertex, int grade)
{
    if (g->numVertexes < MAX_VERTEX_NUM)
    {
        strcpy_s(g->adjList[g->numVertexes].V_data, vertex);
        g->adjList[g->numVertexes].grade = grade;
        g->adjList[g->numVertexes].firstedge = NULL;
        g->numVertexes = g->numVertexes + 1;

        return OK;
    }
    else {
        printf("OVERFLOW!\n");

        return OVERFLOW;
    }
}
```

//在边表中增加结点

```
Status AddEdge(GraphAdjList g, char* vertex, char* adjv)//图, 当前课程编号, 插入课程编号
{

    int i;
    EdgeNode* q;
    PEdgeNode pe = (PEdgeNode)malloc(sizeof(struct EdgeNode));
```

```

    if (pe != NULL) //分配空间成功
    {
        i = Locate(g, adjv);
        pe->adjvex = i;
        strcpy_s(pe->E_data, adjv);
        i = Locate(g, vertex);
        q = g->adjList[i].firstedge;
        g->adjList[i].firstedge = pe;
        pe->next = q;

        return OK;
    }
    else { //分配空间失败
        printf("OVERFLOW!\n");

        return OVERFLOW;
    }
}

/* --判断是否为空图-- */
Status IsNullGraph_list(GraphAdjList g)
{
    return (g->numVertexes == 0);
}

/*****-----核心算法部分-----*****/
/*-----拓扑排序算法-----*/
Status TopologicalSort(GraphAdjList G, int* TSArray)
{
    PEdgeNode p;
    EdgeNode* e;
    int i, j, k, nodeno = 0; //拓扑序列数组中元素的个数
    PLinkStack plstack = CreateEmptyStack();

    indegree = (int*)malloc(sizeof(int) * G->numVertexes); //存储各节点的入度的数
    组
    FindIndegree(G, indegree); //求各节点的入度

    for (i = 0; i < G->numVertexes; i++)
    {
        if (indegree[i] == 0)
        {
            PushStack(plstack, i); //先将所有入度为零的节点的下标压入栈中
        }
    }
}

```

```

    }
}

while (!IsEmptyStack(plstack))
{
    //当栈不为空时
    j = GetTop(plstack);           //取栈顶元素
    PopStack(plstack);           //出栈
    TSArray[nodeno++] = j;        // 将下标存入拓扑序列数组，个数自加

    for (k = 0; k < G->numVertexes; k++)
    {
        //查找下标为j的节点的后继
        p = G->adjList[k].firstedge;    //顶点k的边表头指针
        while (p != NULL)
        {
            if (p->adjvex == j)
            {
                //j为入度为零元素的下标，当相等时，vex[k]就是该元素的后继
                indegree[k] = indegree[k] - 1; //将该元素的入度减一
                if (indegree[k] == 0)
                {
                    //减一后为零就压入栈中
                    PushStack(plstack, k);
                }
            }

            p = p->next;
        }
    }

    free(indegree);    //释放空间
    free(plstack);
    if (nodeno < G->numVertexes) {    //如果拓扑序列数组元素个数小于AOV网的顶点数，说明
        拓扑排序失败
        printf("警告：该AOV网中含有回路！请检查输入的数据！\n");

        return ERROR;
    }

    system("cls");

    return OK;
}

/*-----课程安排的前驱后继图-----*/
void RelationshipMap(GraphAdjList g, int* TSArray)
{

```

```

int i, j;
PEdgeNode p;//定义边表结点指针

printf("-----\n");
printf("该课程安排的前驱后继关系图为: \n");
for (i = 0; i < g->numVertexes; i++)//循环顶点表中的每一个顶点, 即每门课程
{
    int k = 0;

    for (j = 0; j < g->numVertexes; j++)//循环边表中的每一个顶点, 即每门课程
    {
        p = g->adjList[j].firstedge;//顶点表中指向边表的头指针
        while (p != NULL)//如果有后继的课程的话则进入循环
        {
            if (strcmp(p->E_data, g->adjList[i].V_data) == 0)//若边表中的课程号与
            顶点表中的课程号相等
            {
                printf("%s", g->adjList[i].V_data);//打印顶点表中的课程号
                printf("---->%s", g->adjList[j].V_data);//打印边表中的第一个顶点
                break;
            }
            p = p->next;//指向边表中的下一个结点
        }
        if (p != NULL && strcmp(p->E_data, g->adjList[i].V_data) == 0)//若有后继结
        点且边表中的课程号与顶点表中的课程号相等
        {
            k = j;//将边表中后面的结点序号赋值给k
            break;
        }
    }
    for (j = k + 1; j < g->numVertexes; j++)//打印边表中后续的结点编号
    {
        p = g->adjList[j].firstedge;//顶点表中指向边表的头指针
        while (p != NULL)
        {
            if (strcmp(p->E_data, g->adjList[i].V_data) == 0)
            {
                printf(", %s", g->adjList[j].V_data);
            }
            p = p->next;
        }
    }
    printf("\n");
}

```

```

    }
    printf("该课程安排的排序方案为: \n");
    for (i = 0; i < g->numVertexes; i++)
    {
        printf("%s ", g->adjList[TArray[i]].V_data);
    }
    printf("\n");
    printf("-----\n");
}

/*-----课程安排算法-----*/
void Strategy(GraphAdjList g, int* TArray)
{
    int i, j, grade;

    printf("-----\n");
    for (i = 0, j = 0; i < terms; i++)//第1-8个学期开始循环
    {
        grade = 0;
        printf("第%d个学期的课程为: ", i + 1);
        while ((grade += g->adjList[TArray[j]].grade) <= gradelimits)//当该学期修的学
        分小于限制学时进行排课
        {
            printf("%s ", g->adjList[TArray[j]].V_data);//打印课程编号
            j++;
            if (j > g->numVertexes - 1) break;//如果课程排完了则退出
        }
        if (j > g->numVertexes - 1) break;//如果课程排完了则退出
        printf("\n");
    }
    printf("\n-----\n");
}

/*-----课程关系手动输入-----*/
Status Input(GraphAdjList g)
{
    int i, j, n, grade, in;
    char str[100], str2[100];

    printf("请输入教学课程总数量: ");
    scanf_s("%d", &n); getchar();

```

```

for (i = 0; i < n; i++)
{
    printf("请输入第%d门课程的编号: ", i + 1);
    gets_s(str);
    printf("请输入%s课程的学分: ", str);
    scanf_s("%d", &grade);
    printf("请输入%s课程的先修课程数量: ", str);
    scanf_s("%d", &in); getchar();
    AddVex(g, str, grade);
    if (in)
    {
        for (j = 0; j < in; j++)
        {
            printf("请输入%s课程的第%d门先修课程的编号:", str, j + 1);
            gets_s(str2);
            AddEdge(g, str, str2);
        }

        printf("-----\n");
    }

    return OK;
}

/*-----用户界面设计-----*/
Status UI(GraphAdjList g)
{
    printf("\t\t-----\n");
    printf("\t\t|          欢迎使用教学计划编制系统          |\n");
    printf("\t\t|          design by 赵哲冕          |\n");
    printf("\t\t-----\n");
    printf("\t\t|          请按任意键继续...          |");

    getchar();
    system("cls");
    printf("\t\t-----\n");
    printf("\t\t| 请按提示进行输入操作, 如输入错误请重新启动程序 |\n");
    printf("\t\t-----\n");

```



```

    return 0;

}

/*-----主函数-----*/
int main()
{
    GraphAdjList g = CreateGraph_List();
    int* TSA;
    TSA = (int*)malloc(sizeof(int) * g->numVertexes);

    UI(g); //用户界面
    Input(g); //手动输入
    if (TopologicalSort(g, TSA) != 1)
    { //拓扑排序
        return -1;
    }

    printf("\t\t-----\n");
    printf("\t\t|          教学计划编制系统正在运行          |\n");
    printf("\t\t-----\n");
    RelationshipMap(g, TSA); //显示顶点间的前驱后继关系
    printf("\t\t          请按任意键继续...\n");
    getchar();
    system("cls");

    printf("\t\t-----\n");
    printf("\t\t| 请按提示进行输入操作，如输入错误请重新启动程序 |\n");
    printf("\t\t-----\n");
    printf("\n请输入学期总数：");
    scanf_s("%d", &terms); getchar();
    printf("\n请输入每学期学分上限：");
    scanf_s("%d", &gradelimits); getchar();
    system("cls");
    printf("\t\t-----\n");
    printf("\t\t|          教学计划编制系统正在运行          |\n");
    printf("\t\t-----\n");
    Strategy(g, TSA); //使得课程尽可能地集中在前面的学期中

    printf("\n");
    printf("\t\t-----\n");
    printf("\t\t|      教学计划编制系统运行完毕!      |\n");
    printf("\t\t-----\n");
    printf("\t\t          请按任意键退出.\n"); getchar();
}

```