**第四部分 第1题： 教学计划编制问题**

**实验报告**

题目：制定一个教学计划编制程序

班级：F1703407 姓名：赵伟基 学号：517021910883 完成日期：2019-01-04

1. **需求分析**

大学的每个专业都有制定教学计划。假设任何专业都有固定的学习年限，每个学年有两个学期，每个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，并且课程在开设时间的安排必须满足先修关系。每门课程有哪些先修课程都是确定的，可以有任意多门，也可以没有。每门课恰好占一个学期。试在这样的前提下设计一个教学计划编制程序。

基本要求如下：

（1）输入参数包括：学期总数，一个学期的学分上限，每门课的课程号（固定占3位 的字母数字串）、学分和直接先修课的课程号。

（2）允许用户指定下列两种编排策略之一：一是使学生在各学期中的学习负担尽量均匀；二是使课程尽可能地集中在前几个学期中。

（3）若根据给定的条件问题无解，则报告适当的信息；否则将教学计划输出到用户指 定的文件中。计划的表格格式自行设计。

1. **概要设计**

本实验根据教学计划先修课程表设计教学计划，需要用到图论中结点和边的定义

定义了结构图node1作为课程节点，node作为课程的边。同时利用队列类来对课程进行拓扑排序；拓扑排序中需要的入度indegree，访问标记vis等为便于使用作为全局变量声明。完成拓扑排序后经过输入选择获得两种策略的结果

1. 主程序模块

int main()

{

手动输入模块

Solve函数

（solve函数包含了对拓扑排序Top\_Sort\_Queue()函数与具体的策略编制和输出模块）

}

1. 队列的抽象类class Queue
2. 数据成员：top 队首 ；rear 队尾；base；
3. 成员函数：
   1. IsFull()：队满判断；
   2. IsEmpty()：队空判断；
   3. Pop()：出队
   4. Push(): 入队
   5. Queue():构造函数
   6. ~Queue():析构函数
   7. Front():取队首
4. 完成拓扑排序的相关定义
   * 1. 课程结点类

数据成员：id 课程编号 cre 学分

k 先修课程总数 pre[N][4] 先修课程编号

* + 1. 边结点类

数据成员： x起点 y终点

* + 1. 其他数据成员：InDegree 入度； res 拓扑排序的序列保存

S 学期总数 C 每学期学分上限 n/num 科目总数

tot 总学分 ave 平均学分

* + 1. 功能函数
       1. Top\_Sort\_Queue（）：利用队对课程进行拓扑排序
       2. solve（）：根据输入产生两种解决方案。

1. **详细设计**
2. Head.h

#ifndef HEAD\_H

#define HEAD\_H

#include <iostream>

#include <fstream>

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

#define N 100

#define M 10010

template<class T>

class Queue //队列类的基本定义

{

public:

bool IsFull() const;

bool IsEmpty() const;

T front();

void Pop();

void Push(T &e);

Queue(int size);

~Queue();

private:

T \*base;

int top,rear;

int maxsize;

};

template<class T>

bool Queue<T>::IsFull() const

{

return rear == maxsize;

}

template<class T>

bool Queue<T>::IsEmpty() const

{

return rear == top;

}

template<class T>

Queue<T>::Queue(int size):maxsize(size)

{

base = new T[maxsize];

if (base == NULL)

{

cout<<"Memory Allocation Failed!"<<endl;exit(0);

}

top = rear = 0;

}

template<class T>

void Queue<T>::Pop()

{

top++;

}

template<class T>

void Queue<T>::Push(T &e)

{

base[rear++] = e;

}

template<class T>

T Queue<T>::front()

{

return base[top];

}

template<class T>

Queue<T>::~Queue()

{

delete [] base;

}

#endif // HEAD\_H

1. Main.cpp

#include "head.h"

using namespace std;

int InDegree[N],res[N];

int S,C,n,l,num,op,ave,s1,tot;

bool flag;

bool vis[N];

struct node1

{

char id[4],pre[N][4];

int cre,k;//cre学分 k先修课程总数

};

node1 course[N];

int head[N];

struct node

{

int x,y,next;

};

node edge[M];

void add(int x,int y)

{

l++;

edge[l].x = x;edge[l].y = y;

edge[l].next = head[x];head[x] = l;

}

int TopSort\_Queue()//对课程进行拓扑排序

{

memset(vis,0,sizeof(vis));//访问记录清零

Queue<int> Q(N);

for (int i=1;i<=n;i++)

if (InDegree[i] == 0) Q.Push(i);

while (!Q.IsEmpty())

{

int v = Q.front();

res[++num] = v;

vis[v] = 1;

Q.Pop();

for (int p=head[v];p;p = edge[p].next)

{

int u = edge[p].y;

InDegree[u]--;

if (InDegree[u] == 0) Q.Push(u);

}

}

for (int i=1;i<=n;i++)

if (!vis[i]) return 0;

return 1;

}

void solve()

{

memset(head,0,sizeof(head)); //数据清零

memset(InDegree,0,sizeof(InDegree));

l = 0;

for (int i=1;i<=n;i++)//添加表示直接先修课程的有向边

{

for (int j = 1;j<=course[i].k;j++)

{

flag = 0;

for (int i1 = 1;i1<=n;i1++)

if (!strcmp(course[i1].id, course[i].pre[j]))

{

add(i1,i);

InDegree[i]++;

flag = 1;

}

if (!flag)

{

cout<<"构建失败，课程编号不存在!\n";

return ;

}

}

}

int tmp = TopSort\_Queue();

if (!tmp)

{

cout<<"构建失败，在先修课程中存在逻辑错误\n";

return ;

}

tot = 0;

s1 = 0;

for (int i=1;i<=num;i++)

{

tot += course[res[i]].cre;

if (tot > C)

{

s1++;

tot = course[res[i]].cre;

}

if (course[res[i]].cre > C)

{

cout<<"编制失败 下列课程的学分大于每学期学分限制\n"<<course[res[i]].id;

return ;

}

cout<<course[res[i]].id<<endl;

}

if (s1 > S)

{

cout<<"编制失败 学期数及学分限制不足以满足该教学计划的编制！\n";

return ;

}

cout<<"请输入你的编制目标\n";

cout<<"方案1是使每个学期的学习负担尽量均匀\n";

cout<<"方案2是使课程尽量集中在前面几个学期\n";

cout<<"输入1或2来完成方案制定，输入0退出程序\n";

ofstream outFile;

outFile.open("result.txt", ios::out);//建立输出文件result.txt

while (~scanf("%d", &op))

{

if (op == 1)

{

outFile<<"策略1:\n";

cout<<"策略1:\n";

ave = num / S;//学分平均值

s1 = 0;//表示各个学期

int i = 1;

while (i<=num)

{

s1++;

int tot = 0;//累计一个学期的学分总和

int j = i;

while (j<=num && j-i+1<=ave && tot <= C)//保证每个学期安排的课程数目不超过平均数且在学分限制以内

tot += course[res[j++]].cre;

outFile<<"第"<<s1<<" 学期: ";

cout<<"第"<<s1<<" 学期: ";

for (int k = i;k<j;k++)

{

outFile<<course[res[k]].id<<' ';

cout<<course[res[k]].id<<' ';

}

outFile<<endl;

cout<<endl;

i = j;

}

}

else if (op == 2)

{

outFile<<"策略2:\n";

cout<<"策略2:\n";

s1 = 0;

int i = 1;

while (i<=num)

{

s1++;

int tot = 0;

int j = i;

while (j<=num && tot <= C)//尽可能多的将课程安排在前几个学期

tot += course[res[j++]].cre;

outFile<<"第"<<s1<<" 学期: ";

cout<<"第"<<s1<<" 学期: ";

for (int k = i;k<j;k++)

{

outFile<<course[res[k]].id<<' ';

cout<<course[res[k]].id<<' ';

}

outFile<<endl;

cout<<endl;

i = j;

}

}

else

{

cout<<"Thank you!\n";

return ;

}

}

}

int main()

{

cout<<"请输入学期总数S 每学期学分限制C 以及学科总数N\n";

cout<<"S = ";

scanf("%d", &S);

printf("C = ");

scanf("%d", &C);

printf("N = ");

scanf("%d", &n);

cout<<"请输入课程编号 学分 先修课程的总数目和各课程编号\n";

for (int i=1;i<=n;i++)

{

scanf("%s %d", course[i].id, &course[i].cre);

scanf("%d", &course[i].k);

for (int j=1;j<=course[i].k;j++)

scanf("%s", course[i].pre[j]);

}

solve();

return 0;

}

1. **调试分析**
2. 为方便，该实验没有规定图类，输入的数据均以全局变量数组形式存储，数据安全性缺乏保证。
3. 该算法未对项目进行很好的面向对象保证，代码的可重复性一般
4. 在算法中对各种特殊情况以进行报错处理，容错性较好。
5. 算法复杂度分析

（1）：时间复杂度

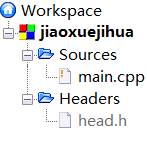
1. O(n^3):solve();
2. O(n^2):main();Top\_Sort\_queue()
3. O(1):add();

（2）：空间复杂度

O(n): prim();remove();insert();adjListGraph();

1. **用户手册**

1．本程序使用的 Code::Blocks 16.01 IDE，程序以项目（project）方式组织，如图 1 所示



2． 用户点击main.cpp->build and run,手动按照交互提示输入数据即可。

输入样例如下：

C01 2 0

C02 3 1 C01

C03 4 2 C01 C02

C04 3 1 C02

C05 2 2 C03 C04

C06 3 1 C11

C07 4 2 C05 C03

C08 4 2 C03 C06

C09 7 0

C10 5 1 C09

C11 2 1 C09

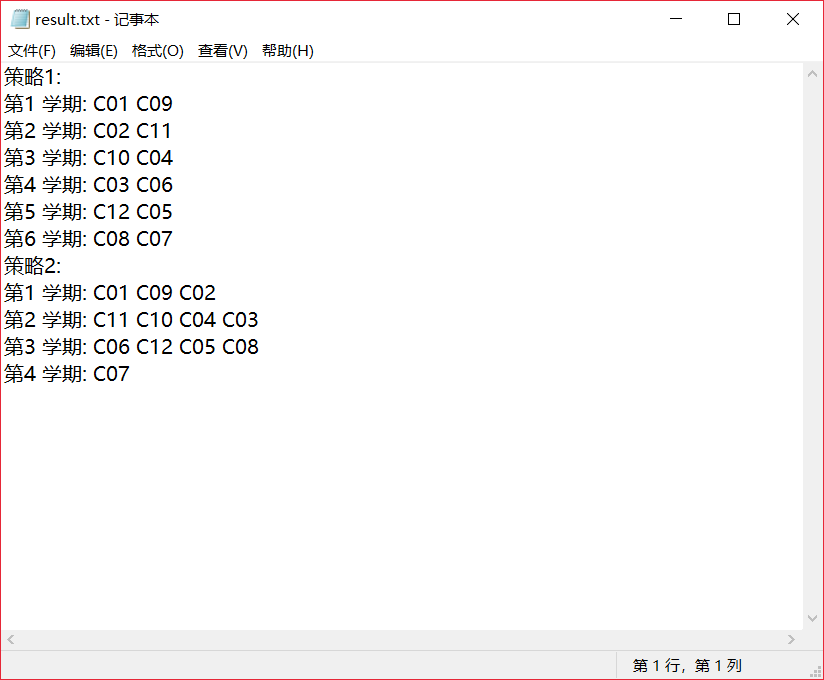
C12 3 3 C09 C10 C01

1. **测试结果**
2. 依照测试数据



1. 手动输入数据

依照图片所示输入数据，即可生成方案并传至文件“result.txt”中去。



1. **附录**

main.cpp //主函数 功能函数模块

head.h //头文件 定义函数