SHANGHAI JIAO TONG UNIVERSITY

教学计划编制问题

吕艺

517021910745

December 30, 2018

1 需求分析

1. 本演示文件的主要目的是制定一个教学计划编制程序。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。

2. 演示文件需要用户输入学期数，没学期的学分上限每门课的课程编号，学分和先修课程的课程号以及编排策略，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。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"schedule.txt" 中。

3. 程序执行的命令包括:
a. 输入用户自定义的命令
b. 选择排序方式
c. 打印错误信息或将信息输出到文件中

2 概要设计

本演示文件中主要采用拓扑排序的方法来进行课程的规划，并采用栈，队列，map 等内置容器来提高程序运行的速度和可读性。

1) 结构体 *Node*

数据对象：char name[10];　　int credit;

2)void create_graph();

操作条件：图还没被建成

操作结果：录入课程信息

2）void solve1(int ans[])；

操作条件：数据已被录入

操作结果：按策略一排序（负担均匀）

3)void solve2(int ans[])

操作条件：数据已被录入

操作结果：按策略二排序(课程尽可能在前几学期)

4)void topo_sort();

操作条件：数据已被录入

操作结果：生成课程信息排序文件

2. 本程序包括两个模块：

1) 主函数模块

int main() {

生成图

拓扑排序

}

2) 拓扑排序 *topo* 单元模块--利用拓扑排序进行课程安排

3 详细设计

1）*topo* 单元模块

```
struct node
{
    char name[10];              //课程名
    int  credit;                //总学分
};

int num_terms, t_max_credit, t_Course, t_E;      //总学期数，学分上限，必修课数，边数
vector<node> G[1000];                            //边表
map<string, int> mp;                             //映射表，从字符型到整型
```

```
void create_graph()
{
    int i;
    cout<<"\t\t\t欢迎使用教学计划编制系统\n";
    cout<<"输入学期总数:␣";
    cin>>num_terms;
    cout<<"请输入学期的学分上限:␣";
    cin>>t_max_credit;
    cout<<"请输入教学计划的课程数:␣" ;
    cin>>t_Course;
    cout<<"请输入各个课程的先修课程的总和(边总数):␣";
    cin>>t_E;
    cout<<"请输入"<<t_Course<<"个课程的课程号(最多30个字符,大写字母+数字如C10)\n";
    node data;
    for(i = 1; i <= t_Course; i++)
    {
        cout<<"请输入第"<<i<<"个";
        cin>>data.name;
        G[i].push_back(data);                    //建立边表
        mp[G[i][0].name] = i;                    //建立映射
    }
    cout<<"请输入"<<t_Course<<"个课程分别对应的学分值";
    for(i = 1; i <= t_Course; i++) cin>>G[i][0].credit;
    cout<<"请输入下列课程的先修课程(输入以#结束)\n";
    char s[30];
    for(i = 1; i <= t_Course; i++)
    {
        cout<<G[i][0].name<<"的先修课程:␣";
        while(true)
        {
            cin>>s;
            if(s[0] == '#') break;               //输入示例: c03 #
            G[i].push_back(G[mp[s]][0]);         //将先修课程插入到对应课程的vector中
        }
    }
    cout<<"\t\t\t录入数据成功\n";
}


void solve1(int ans[])                           //课程数平均
{   ofstream out;
    out.open("Schedule.txt");
    int q = 1, cnt = 0;
    while (q <= num_terms)
    {
        int num = t_Course / num_terms;
        cout<<"\n第"<<q<<"个学期应学课程:␣";
        while(num--)
        {
            cout<<G[ans[cnt++]][0].name<<(num != 0 ? '␣' : '\n');
```

```
        }
        if (q == num_terms) cout<<"OK Over!\n";
        q++;
    }
    out.close();
}

void solve2(int ans[])
{   ofstream out;
    out.open("Schedule.txt");
    int q = 1, cnt = 0;
    while (q <= num_terms)
    {
        int C = G[ans[cnt]][0].credit;
        cout<<"\n第"<<q<<"个学期应学课程: ";
        while(cnt < t_Course && C <= t_max_credit)       //不超过学分上限
        {
            cout<<G[ans[cnt]][0].name;
            if(cnt+1 < t_Course) C = C + G[ans[cnt+1]][0].credit;
            cnt++;
        }
        if(cnt >= t_Course || q == num_terms)
        {
            cout << endl;
            cout<<"OK Over!\n";
            break;
        }
        q++;
    }
    out.close();
}

void topo_sort()
{
    int i, j, Innode[1000];
    for(int i = 0;i < 1000; i++)
        Innode[i] = 0;                          //初始化
    for(i = 1; i <= t_Course; i++)
    {
        int k = G[i].size();                    //vector内置函数
        for(j = 1; j < k; j++)
            Innode[mp[G[i][j].name]]++;         //计算先修课程的门数
    }

    int ans[1000], cnt = 0;                     //初始化
    for(int i = 0;i < 1000; i++)
        ans[i] = 0;

    stack<int> s;
    for(i = 1; i <= t_Course; i++)
```

```
    {
        if(!Innode[i]) s.push(i);                //入度不为零，入栈
    }

    while(!s.empty())
    {
        int cur = s.top(); s.pop();
        ans[cnt++] = cur;
        int k = G[cur].size();
        for(j = 1; j < k; j++)
        {
            int num = mp[G[cur][j].name];        //更新入度状态
            Innode[num]--;
            if(!Innode[num]) s.push(num);            //如果入度为零，入栈
        }
    }
    if(cnt != t_Course) cout<<"Error!";
    else
    {
        puts("OK!");
        while(true)
        {
            cout<<"\n\t\t\t请选择功能:\n";
            cout<<"\t\t\t1.平均分配\n";
            cout<<"\t\t\t2.靠前分配\n";
            cout<<"\t\t\t3.退出\n";
            int strategy;
            cin>>strategy;
            switch(strategy)
            {
                case 1: solve1(ans); break;
                case 2: solve2(ans); break;
            }
            if(strategy == 3) break;
        }
    }
}
```

2）主函数模块

```
int main()
{
    create_graph();
    topo_sort();
    return 0;
}
```

4 调试分析

1. 一开始本演示文件通过遍历寻找对应课程的位置，时间成本高，在改为使用内置容器 map 后代码的可读性大大提升，且时间成本降低。利用栈来模拟递归也降低了空间成本。

2. 算法的复杂度分析

1）时间复杂度

本演示程序的代码中采用了很多 c++ 内置容器操作，时间复杂度较合理。

create_graph，$solve1$,$solve2$ 函数的时间复杂度与元素个数有关，故为 $O(n)$。

topo 函数的时间复杂度主要取决于两层循环嵌套，故为 $O(n^2)$。

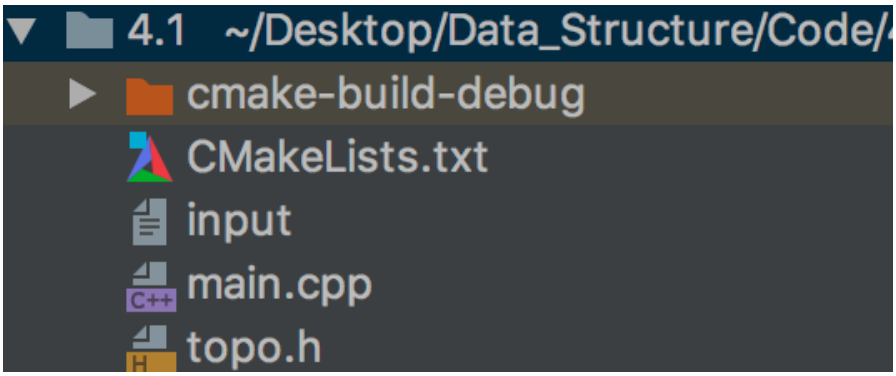
2）空间复杂度

topo 排序模块中利用栈模拟递归过程进行拓扑排序，故其时间复杂度为 O(n)。

主函数模块的复杂度取决于定义主函数作用域 *topo* 模块中的 create_graph 和 topo_sort 函数，故空间复杂度也为 $O(n)$。

5 用户手册

1. 本程序以 Jetbrains Clion 2018.2.5, 采用 C++ 11 标准，程序以项目方式组织 (project)，如图 1 所示：



2. 依次点击菜单"Run"->build, 再点击"Run"，运行程序。

```
                欢迎使用教学计划编制系统
输入学期总数： 6
请输入学期的学分上限： 10
请输入教学计划的课程数： 12
请输入各个课程的先修课程的总和(边总数)： 16
请输入12个课程的课程号(最多30个字符,大写字母+数字如C10)
请输入第1个C01
请输入第2个C02
请输入第3个C03
```

3. 输入用户自定义的信息后输出结果。

6 测试结果

在输入测试数据后，生成以下文件内容。

```
Strategy1:
第1个学期应学课程： C12 C10

第2个学期应学课程： C08 C06

第3个学期应学课程： C11 C09

第4个学期应学课程： C07 C05

第5个学期应学课程： C04 C03

第6个学期应学课程： C02 C01
```

根据以上测试结果数据分析，可以得出模拟数据与理论时间复杂度相符。

7 附录

源程序文件名清单

main.cpp　　//主函数

topo.h　　//排序函数单元模块