西 安 交 通 大 学 验 报 告

实 验 名 称：教学计划编制程序　　课 程 名 称：数据结构与算法

所 在 学 院：抄作业学院　　专　　业：不知道

学 生 姓 名：哈哈哈　　学　　号：111111111

班　　级：不愿意 001　　实 验 日 期：2024 年 6 月 1 日

诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，绝无抄袭。

二〇二四年六月

目录

1 实验目的

1. 掌握拓扑排序，熟悉图的存储结构
2. 掌握 debug 的逐过程和单步调试
3. 初步掌握 LaTeX 的写作方法

1.1 问题图示

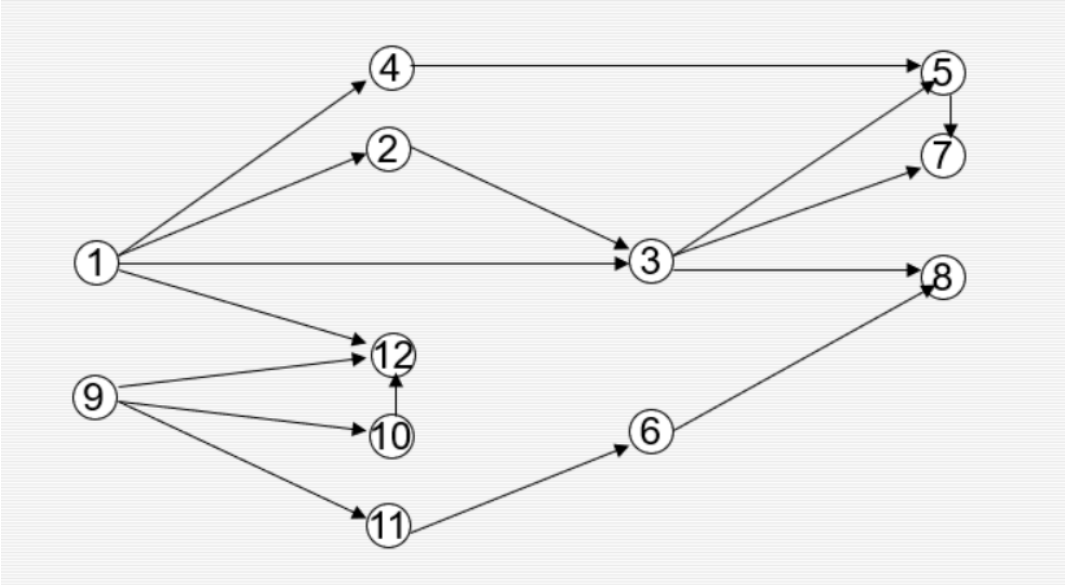

图 1: 问题图示

2 实验环境

硬件配置:

1. 处理器 AMD Ryzen 9 7940H w/ Radeon 780M Graphics，4001 Mhz，8 个内核，16 个逻辑处理器
2. 已安装的物理内存 (RAM) 16.0 GB
3. 系统型号 $ASUSTUFGamingA15FA507XV_FA507XV$

软件配置: Visual Studio Code

3 项目设计和实现

3.1 项目框架

3.2 核心代码阐释

巧妙的是，本次主函数非常的简洁，除此之外 ClassAssignment 类也只有一个 run() 函数。

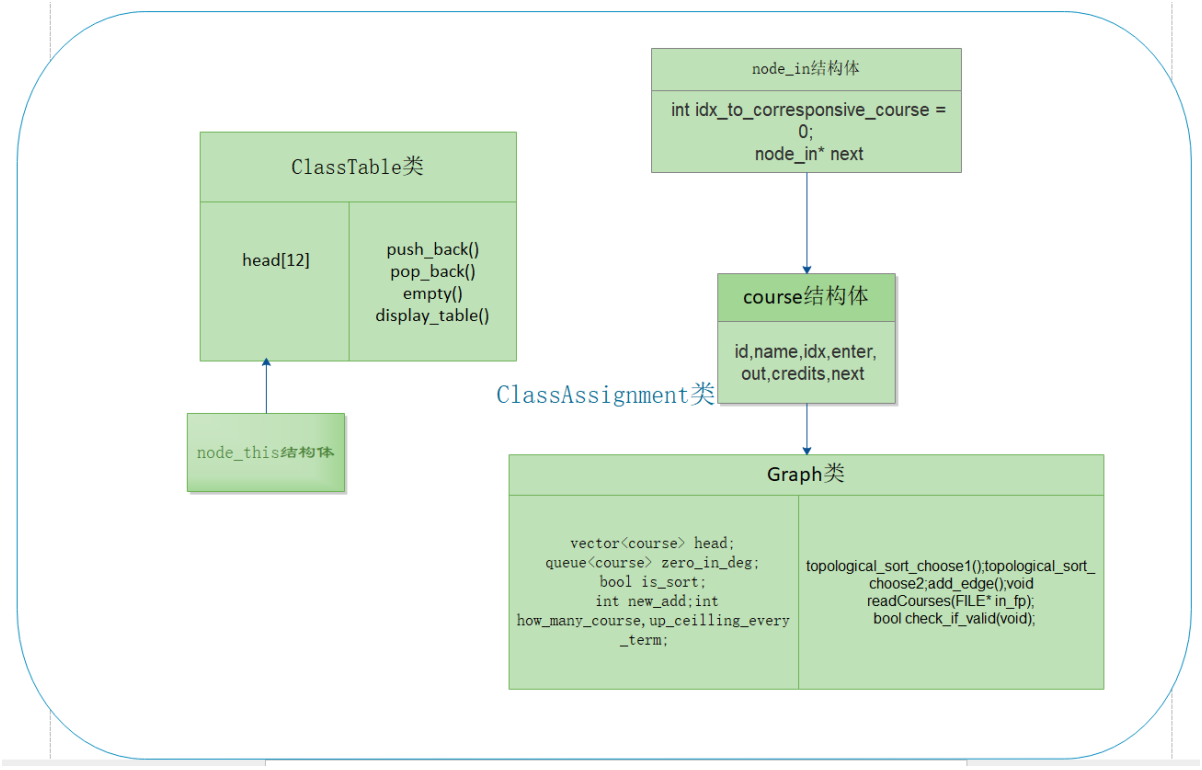

图 2: 项目框架

3.2.1 主函数

```
    #include <iostream>
    #include "ClassAssignment.h"
using namespace std;
int main()
{
        ClassAssignment program;
        program.run();
        return 0;
}
```

3.2.2 Graph 类的参数和一般方法

1. 参数：vector<course> head 为邻接表头节点，head[i] 是第 i 节课；queue<course> zero_in_deg 来存储入度为 0 的结点；is_sort 代表是否执行过拓扑排序；int new_add 代表每次拓扑排序后，zero_in_deg 中新增的入度为零的节点，可以用于下次排序。

2. 方法：topological_sort_choose1 和 topological_sort_choose2 分别代表对应题目要求的两种不同策略的拓扑排序，readCourses(FILE* in_fp) 用于从记事本中读取信息存入 head 中，add_edge() 函数则构建起边的连接，采用的是邻接表的存储方式。对于该邻接表中的某个节点 A，其 node_in* next 的 next 相互连接，共同构成以 A 为先修条件的下一组课程 (可能先修条件不仅仅只有 A)。因为在拓扑排序的过程中，原有图的结构会产生破坏，因而 check_if_valid(void) 用于遍历所有 course 节点的入度，判断入度是否都为零，即可以认为检查拓扑排序的图是否存在环。

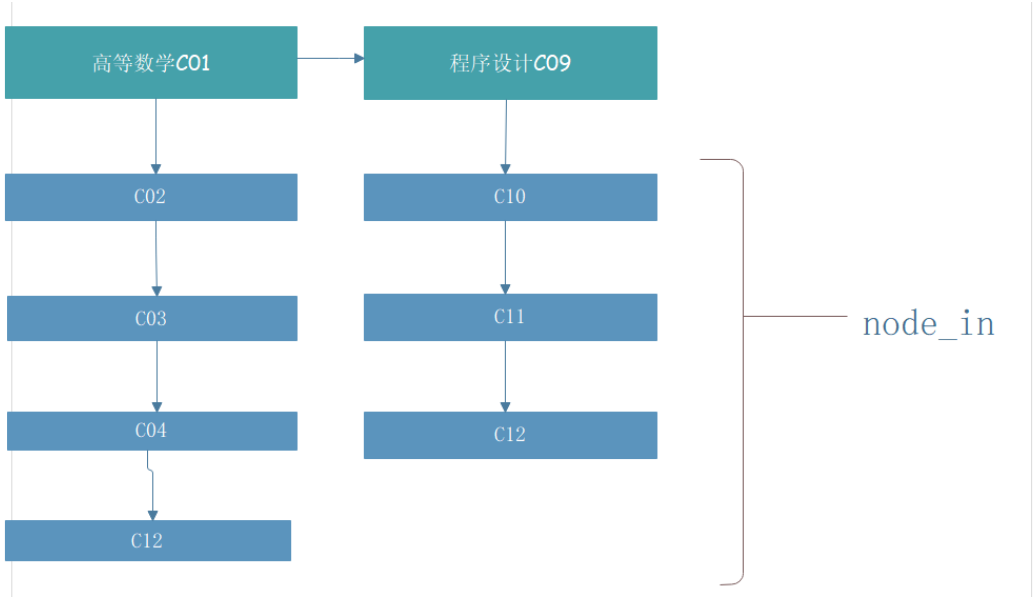

图 3: 邻接表图示

```
class Graph{
public:
       int how_many_course,up_ceilling_every_term;
       vector<course> head; //邻接表头节点，head[i] 是第 i 节课
       queue<course> zero_in_deg; //存储入度为 0 的结点
        bool is_sort = false; //是否执行过拓扑排序
  int new_add = -1;
  Graph()
       {
        cout<<" 请输入一共要学多少门课程 ，以及每学期的学分上限："<<endl;
        cin>>how_many_course>>up_ceilling_every_term;
        cout<<"---------------------------------成功-------------------------------------"<<endl;
        cout<<"--------------------------C:\\Users\\xixiboliya\\Desktop\\course.txt---------------------"<<endl;
        };
        void topological_sort_choose1(int xuefen,int term,ClassTable& class_table); //拓扑排序，均匀
        void topological_sort_choose2(int xuefen,int term,ClassTable& class_table); //拓扑排序，集中
       void add_edge(string num, vector<string> prerequisite); //添加一条有向边到图中，num 是该节点的课序号，prerequsite 是先置课程的序
       void readCourses(FILE* in_fp);
  bool check_if_valid(void);
};

```

3.2.3 ClassTable 类的参数和一般方法

ClassTable 用于存放最终每学期所排的课程，node_this head[12] 中，head[i] 上挂的是第 i+1 学期的课程，采用邻接表存储方式，每一个 head[i] 中每一个是一个虚拟节点，data 为空，其每一个虚拟节点的 next 接出该学期应该的所修课程。

```
   class ClassTable
{
public:
```

```
^^Inode_this head[12]; //表头，head[i] 上挂的是第 i+1 学期的课
  ClassTable() {
    for (auto &i : head) {
        i = node_this();   //进行初始化，并使用了虚拟头节点。
    }
  }
  void push_back(int term, string name) //尾插法
  {
    node_this* tmp_ptr = &head[term];
    while (tmp_ptr->next)
    tmp_ptr = tmp_ptr->next; //寻找尾节点
    tmp_ptr->next = new node_this; //开辟新空间
    tmp_ptr->next->data=name;
  }
void pop_back(int term, string name) //尾部弹出一个结点
  {
    node_this* tmp_ptr = &head[term];
    node_this* tmp_ptr_previous = tmp_ptr;
    while (tmp_ptr->next)
    {
      tmp_ptr_previous = tmp_ptr;
      tmp_ptr = tmp_ptr->next; //找尾节点
    }
    if (tmp_ptr != &head[term])
    {
    name=tmp_ptr->data; //取出课程名
    if (tmp_ptr != &head[term])
     {
        delete tmp_ptr; //释放空间
        tmp_ptr_previous->next = nullptr; //防止野指针
    }
    }
   }
   bool empty(int term)
  {
    return (bool)(!(head[term].next));
    }
^^Ivoid display_table(int terms)
  {
^^I  for(int i = 0;i <= terms ; i++)
    {
     cout<<" 第"<<i+1<<" 学期课程如下: "<<endl;
     node_this* ptr= head[i].next;
     while(ptr!=nullptr)
     {
        cout<<ptr->data<<endl;
        ptr=ptr->next;
      }
    }
   }
};
```

3.2.4 Graph 类 add_edge() 函数的使用

这个函数是 readCourses(FILE *in_fp) 之后调用的，作用在于建立一个以 <course> 为主主体的 vector 数组，并采用邻接表的形式存储，笔者精力憔悴，不想

```
void Graph::add_edge(string num, vector<string> prerequisite)
{
  int class_idx = (num[1] - '0') * 10 + num[2] - '0' - 1; // 获得欲添加的课程在 head 数组中的位置
  head[class_idx].idx = class_idx;      // 把课程号作为下标写入
  for (auto &pre : prerequisite)
  {
    head[class_idx].enter++; // 节点入度 ++
    int pre_idx = (pre[1] - '0') * 10 + pre[2] - '0' - 1;
    head[pre_idx].out++;       // 结点出度 ++
    node_in *cur_ptr = head[pre_idx].next; // 在前置课程的邻接表尾部插入当前课程
    if (!cur_ptr)
    {
      // 邻接表当前行只有头节点
      head[pre_idx].next = new node_in; // 添加新节点
      head[pre_idx].next->idx_to_corresponsive_course = class_idx;
    }
    else
    {
      while (cur_ptr->next)
      ^^Icur_ptr = cur_ptr->next;^^I // 寻找尾巴节点
      cur_ptr->next = new node_in; // 添加新节点
      cur_ptr->next->idx_to_corresponsive_course = class_idx;
    }
  }
  return;
}
```

3.2.5 拓扑排序的实现

因为两种实现的拓扑排序大同小异，第一种只是多了当前学分和是否大于每学期所限制的学分的判断，每次调用的时候进行一轮拓扑排序，并将结果存入 class_table 当中，并进行相关节点入度的自减，将自减后入度为零的节点加入到缓冲区 buffer 中，并在拓扑排序结束时一次性加入 zero_in_deg

```
      void Graph::topological_sort_choose1(int xuefen, int term, ClassTable &class_table)
{
       int buffer[30] = {0}; // 缓冲区，用于存储每次新产生的 0 入度结点，在函数结束前将缓冲区内的结点一次性入栈
       int buf_tail = -1;    // 队列结构，指示队列尾部
       new_add = 0;
       if (!is_sort) // 首次运行
    {
    is_sort = true;
    for (auto &i : head) // 把所有入度为 0 的顶点入栈
      if (i.enter == 0 && i.id != "")
          zero_in_deg.push(i);
  }

     while (zero_in_deg.size())
     {
```

```
    course temp; // 临时变量，用于存储出队元素
    temp = zero_in_deg.front();
    if (xuefen + temp.credits > up_ceilling_every_term)
      break;
    xuefen += temp.credits;
    class_table.push_back(term, temp.name);
    zero_in_deg.pop();
    node_in *p_front = head[temp.idx].next; // 双指针法释放链表空间
    node_in *p_back = head[temp.idx].next;
    while (p_front) // 搜索一行邻接表
    {
      p_back = p_front;
      p_front = p_front->next;
      if (--head[p_back->idx_to_corresponsive_course].enter == 0) // 检查这个结点入度是否为 0
      {
      buffer[++buf_tail] = p_back->idx_to_corresponsive_course; // 新的 0 入度结点写入缓冲区
      }
      delete p_back;
    }
^^I}
  for (int i = 0; i <= buf_tail; i++) // 将缓冲区内的结点入栈
  {
    zero_in_deg.push(head[buffer[i]]);
    new_add++;
  }
  return;
}

bool Graph::check_if_valid(void)
{
  bool valid = true;
  for (auto i : head)
  {
    if (i.enter != 0)
    {
      valid = false;
      break;
    }
   }
  return valid;
}

void Graph::topological_sort_choose2(int xuefen, int term, ClassTable &class_table)
{
  int buffer[30] = {0}; // 缓冲区，用于存储每次新产生的 0 入度结点，在函数结束前将缓冲区内的结点一次性入栈
  int buf_tail = -1;    // 队列结构，指示队列尾部
  new_add = 0;
  if (!is_sort) // 首次运行
  {
  is_sort = true;
    for (auto &i : head) // 把所有入度为 0 的顶点入栈
      if (i.enter == 0 && i.id != "")
            zero_in_deg.push(i);
  }

```

```
 while (zero_in_deg.size())
  {
    course temp; // 临时变量，用于存储出队元素
    temp = zero_in_deg.front();
    zero_in_deg.pop(); // 最大元素出队
    class_table.push_back(term, temp.name);
    node_in *p_front = head[temp.idx].next; // 双指针法释放链表空间
    node_in *p_back = head[temp.idx].next;

    while (p_front)
    {
      p_back = p_front;
      p_front = p_front->next;
      if (--head[p_back->idx_to_corresponsive_course].enter == 0) // 检查这个结点入度是否为 0
      {
      ^^Ibuffer[++buf_tail] = p_back->idx_to_corresponsive_course; // 新的 0 入度结点写入缓冲区
      }
      delete p_back;
   }
  }

  for (int i = 0; i <= buf_tail; i++) // 将缓冲区内的结点入栈
  {
    zero_in_deg.push(head[buffer[i]]);
    new_add++;
  }
  return;
}
```

3.2.6 主函数设计

主函数是 ClassAssignment_run() 函数，而不是传统的 int main()。在主函数中先后完成：输入记事本文件的绝对位置、调用 add_edge() 生成图，通过循环调用拓扑排序的函数一轮一轮进行调用、最后 display 运行结果。

4 实验结果

4.1 实验界面

```
请输入一共要学多少门课程 , 以及每学期的学分上限：
12 9
---------------------------------成功-------------------------------------
--------------------------C:\Users\xixiboliya\Desktop\course.txt----------------------
请输入课程信息文件的路径：C:\Users\xixiboliya\Desktop\course.txt
-------------------------------------------肘你 , man !-----------------------------------------------
请选择你需要的方式，一是使学生在各学期中的学习负担尽量均匀(输入1)；二是使课程尽可能地集中在前几个学期中
(输入2): 1
这个图是正确的,可以进行拓扑排序! clever!
```

图 4: 界面

4.2 输入

采用记事本输入，并对输入中的注释进行了去除操作，采用 fget() 和 ungetc() 函数读取一个字符看一看是不是‘/’或回车，是的话放回并丢弃该行。

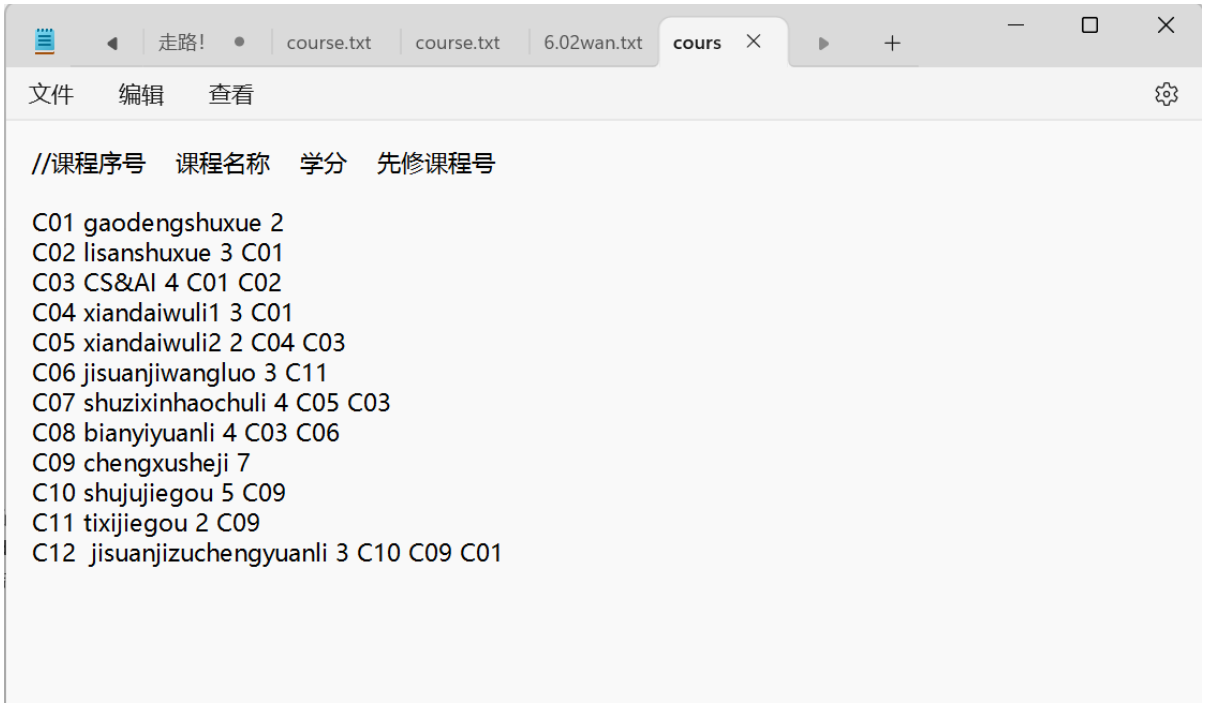

图 5: 输入界面

4.3 均匀排课

```
第1学期课程如下:
gaodengshuxue
chengxusheji
第2学期课程如下:
lisanshuxue
xiandaiwuli1
第3学期课程如下:
shujujiegou
tixijiegou
第4学期课程如下:
CS&AI
jisuanjizuchengyuanli
第5学期课程如下:
jisuanjiwangluo
xiandaiwuli2
第6学期课程如下:
bianyiyuanli
shuzixinhaochuli
PS C:\my VSC\2024.6.01>
```

图 6: 均匀排课

4.4 集中排课

```
第1学期课程如下：
gaodengshuxue
chengxusheji
第2学期课程如下：
lisanshuxue
xiandaiwuli1
shujujiegou
tixijiegou
第3学期课程如下：
CS&AI
jisuanjizuchengyuanli
jisuanjiwangluo
第4学期课程如下：
xiandaiwuli2
bianyiyuanli
第5学期课程如下：
shuzixinhaochuli
PS C:\my VSC\2024.6.01>
```

图 7: 集中排课

可以看到，这两种排课方式对于第六学期的安排并不相同，说明两个拓扑排序的代码都管用。

5 实验总结

5.1 收获

1. 学会了因为多个 cpp 文件联合使用而修改 setting.json 文件的方法
2. 初步掌握了用 LaTeX 写文档的方法
3. 对防止野指针的出现做出了顽强的挣扎
4. 对输入进行了创新，从记事本中读入数据并且去掉注释

5.2 存在的问题

1. 程序健壮性较弱，不能对不法输入进行 assert 警告，虽然使用了 exit(1) 进行一定的修补
2. 对相关类的规定仍然不太清楚，感觉可以将 Graph 类和 ClassTable 类分开装进两个 cpp 文件里
3. 对类内、外函数进行的区分完全根据实际需要确定，没有太多逻辑