

## 数据结构课程设计报告

1. 项目背景概述
  - 1.1. 项目结构概述
  - 1.2. 编译构建以及运行
2. 组员贡献
3. 模块和功能
  - 3.1. 用户身份模块
    - 3.1.1. 学生部分
      - 3.1.1.1. 数据结构说明和数据字典
      - 3.1.1.2. 实现的功能
      - 3.1.1.3. 算法及分析
        - 3.1.1.3.1. 二叉搜索树
        - 3.1.1.3.2. 利用端点和中点坐标的冲突检测算法
      - 3.1.1.4. 与其他模块交互
      - 3.1.1.5. 执行效果
    - 3.1.2. 管理员用户
      - 3.1.2.1. 数据结构说明和数据字典
      - 3.1.2.2. 实现的功能
      - 3.1.2.3. 与其他模块交互
      - 3.1.2.4. 执行效果
    - 3.1.3. 教师用户
      - 3.1.3.1. 数据结构说明和数据字典
      - 3.1.3.2. 实现的功能
      - 3.1.3.3. 与其它模块的交互
      - 3.1.3.4. 执行效果
      - 3.1.3.5. 查重算法详述
        - 3.1.3.5.1. AC 自动机引入
        - 3.1.3.5.2. 字典树构建
        - 3.1.3.5.3. 失配指针
        - 3.1.3.5.4. 构建 fail 指针
        - 3.1.3.5.5. 字典树与 Trie 图
        - 3.1.3.5.6. 多模式匹配
        - 3.1.3.5.7. 查重算法实现
        - 3.1.3.5.8. 复杂度分析
  - 3.2. 学生课程模块
    - 3.2.1. 数据结构说明和数据字典
    - 3.2.2. 实现的功能
    - 3.2.3. 算法及分析
      - 3.2.3.1. 通过下标的转换进行快速排序
      - 3.2.3.2. 压缩文件
      - 3.2.3.3. 解压缩文件
    - 3.2.4. 与其他模块交互
    - 3.2.5. 执行效果
  - 3.3. 校园导航模块
    - 3.3.1. 需求实现概述
    - 3.3.2. 核心算法概述
    - 3.3.3. 需求实现代码细节分析
      - 3.3.3.1. 根据课程名称导航
      - 3.3.3.2. 根据上课地点导航
      - 3.3.3.3. 根据上课时间导航
      - 3.3.3.4. 途径多个地点导航（选做算法部分）
4. 总结和心得
5. 参考文献

# 数据结构课程设计报告

- 周 宇 (2020211606)
- 任晓斌 (2020211592)
- 孟浩洋 (2020211591)

## 1. 项目背景概述

此项目为 2022 年春季北京邮电大学数据结构课程设计项目，背景为学校的课程辅助系统，可以帮助学生管理自己的课程和课外活动，具备课程导航功能、课程信息管理和查询功能，以及课外信息管理和查询功能等。每天晚上系统会提醒学生第二天上的课，每门课需要交的作业和需要带的资料，以及考试的信息。快要上课时系统根据该课程的上课地点设计一条最佳线路并输出。学生可以通过系统管理每门课的学习资料、作业和考试信息。在课外，学生可以管理自己的个人活动和集体活动信息，可以进行活动时间的冲突检测和闹钟提醒。

### 1.1. 项目结构概述

本项目使用终端进行交互的方式，通过命令行来进行信息的输入和输出，是一个在本地运行的单机版程序。

项目的文件目录树如下：

```
1  .
2  |-- CMakeLists.txt
3  |-- LICENSE
4  |-- README.md
5  |-- build
6  |   `-- cmake build files
7  |-- doc
8  |   |-- report.md
9  |   |-- course_model_report.md
10 |   `-- doc.md
11 |-- log
12 |   |-- stu
13 |   `-- stu.txt
14 `-- src
15     |-- CMakeLists.txt
16     |-- global_file.h
17     |-- main.cpp
18     `-- model
19         |-- course_model
20         |-- identity_model
21         `-- navigation_model
```

- `src` 目录包含程序的所有源码, 主要为程序的所有模块。
- `log` 为程序的日志文件目录，包含所有程序运行日志文件。
- `doc` 为项目的文档目录，包含课程设计报告和程序文档。
- `build` 为使用 CMake 构建项目后生成的目录。

## 1.2. 编译构建以及运行

项目依赖:

- GNU tool chains for windows (mingw64)
- GCC version support >= C++14
- CMake version >= 3.21
- bash shell for build in terminal

编译构建项目:

- 使用 Cmake 在 bash 终端命令行构建:

```
1 | rm -rf build/ && mkdir build && cd build && cmake -G "MinGW Makefiles" .. &&  
  | cmake --build . -j 24 --target navigation && cd ../
```

- 进入 build 构建目录执行程序:

```
1 | cd build/build/  
2 |  
3 | ./navigation.exe
```

注意如果使用 git bash 在 build/build 目录执行程序 ./navigation.exe 无响应, 请使用其他 shell 比如 powershell 来执行 ./navigation.exe, 此 bug 为 git bash 在 Windows 平台的不兼容所导致。

## 2. 组员贡献

周宇 (40%):

- 编写课外信息管理的全部代码
- 编写除压缩和解压相关算法的课内信息管理的全部代码
- 编写系统时间模拟算法
- 编写 admin\_model 和 teacher\_model 中除了作业查重算法的全部代码
- 编写日志相关代码

任晓斌 (40%):

- 设计构建整个项目的初始框架, 包括登录注册菜单交互等功能
- 编写项目的构建系统 CMake 配置, 设计各模块的依赖关系
- 编写导航模块的全部代码 (包括途径多个地点最短路的选作算法)
- 实现 Aho-Corasick Algorithm (AC自动机算法) 进行学生作业查重

孟浩洋 (20%):

- 实现哈夫曼压缩算法对文件内容进行压缩和解压
- 实现教师批改作业, 学生提交作业, 学生上传资料以及下载资料的功能
- 协助构建系统的各个基本模块, 以及程序的单元测试

## 3. 模块和功能

整个项目按照不同的功能和需求主要划分为如下三个模块:

- 用户身份模块 `identity_model` , 是整个项目的主要模块, 包含所有用户相关逻辑的实现。
- 学生课程模块 `course_model` , 包括所有学生课程相关逻辑和功能的实现。
- 学生导航模块 `navigation_model` , 包含所有学生在校内导航的功能需求实现。

接下来对每个模块进行详细的逻辑实现思路分析以及代码实现细节的讲述。

## 3.1. 用户身份模块

### 3.1.1. 学生部分

#### 3.1.1.1. 数据结构说明和数据字典

- 1. `map<string, string name_to_id`  
在找到正确的名字后通过名字找到对应的课程下标, 通过下标可以读取课程文件创建课程类
- 2. `map<int, pair<string, string time_to_place`  
在找到合理的时间 (大于输入时间的最新上课开始时间) 后必须通过时间查找到对应的地名 (校区+建筑), 这样学生可以实现通过时间进行导航
- 3. `string my_course_table[6][12]`  
存储学生课程表
- 4. `Node t1[1000], Node t2[1000]`  
分别是用来存储课程时间和活动时间的二叉搜索树
- 5. `map<string, vector<single_activity name_to_activity`  
在找到正确的名字后通过名字找到对应的活动
- 6. `map<int, vector<single_activity time_to_activity`  
在找到正确的时间后通过时间找到对应的活动
- 7. `struct single_activity`  
存储活动信息的单元
- 8. `weekly_sys_time`  
模拟时间, 精确到秒
- 9. `weekly_real_time`  
真实时间, 精确到毫秒, 用于计算得到模拟时间
- 10. `map<word_to_sen, map<word_to_par>`  
字符到包含该字符的字符串的map  
这个 `map` 位于学生模块中, 包含  $x$  项, 用于模糊查找, `map` 内部使用红黑树, 在查找时通过对输入的  $k$  个字符作为 `key`, 查找对应的 `value`, 可以以  $O(k \log x)$  时间复杂度实现模糊查找。

#### 3.1.1.2. 实现的功能

- 1. 课程名称模糊查询
- 2. 课表课程查询
- 3. 课程时间查询
- 4. 活动名称模糊查询
- 5. 进入活动菜单 (进入后可以对个人活动增删改, 还可以对活动按照类别进行查找后按照时间排序)
- 6. 活动时间查询
- 7. 进行路径导航
- 8. 模拟时间暂停

- 9.模拟时间快进
- 10.活动闹钟提醒 (以模拟时间为准, 通过输出 "\a" 模拟闹铃)

### 3.1.1.3. 算法及分析

#### 3.1.1.3.1. 二叉搜索树 <sup>1</sup>

场景:

- 对课程和活动时间进行查询时使用

优缺点:

- 用户期望在输入一个时间后, 能够得到这个时间之后离它最近的那个数据。通过二叉搜索树, 这可以在  $O(\ln n)$  时间内实现。
- 用户在期望按照时间进行排序, 而通过二叉搜索树不必建立新的数组进行排序, 只需要进行一次遍历。
- 用户可能对活动进行大量的添加, 树形结构可以保证结点的变化所导致的数据转移消耗较低。
- 在特殊情况下, 二叉搜索树会失衡, 性能将会逼近链表。

分析:

- 二分查找复杂度分析:  
最坏情况: 假设所要查找的数据在树中最后一层, 那么查找次数  $k$  满足  $2^k = n, k = \log n$ 。  
最坏复杂度 =  $O(\log n)$
- 假设数据量为  $n$ 。  
程序中通过 `kth( rank( time+ 1) )` 来找到 `time` 的后继:

```

1  int Student::rank(int x, int root, Node *t) {
2      if (root) {
3          if (x < t[root].value)
4              return rank(x, t[root].left, t);
5          if (x > t[root].value)
6              return rank(x, t[root].right, t) + t[t[root].left].size +
7 t[root].num;
8          return t[t[root].left].size + t[root].num;
9      }
10     return 1;
11 }
12
13 int Student::kth(int x, int root, Node *t) {
14     if (x <= t[t[root].left].size) return kth(x, t[root].left, t);
15     if (x <= t[t[root].left].size + t[root].num) return t[root].value;
16     return kth(x - t[t[root].left].size - t[root].num, t[root].right, t);
17 }

```

`rank(x+1)` 通过分治查找返回 `x` 的后继的排名, `kth(y)` 通过分治查找返回排名 `y` 的数据的值。两个二分查找复杂度均为  $O(\log n)$ , 则只需要  $O(\log n)$  复杂度即可得到结果。

- 通过 `insert(x)` 来添加结点, 依然采用分治, 则可以在  $O(\log n)$  时间内完成活动的添加

```

1  void Student::insert(int x, int root, Node *t, int &cnt) {
2      if (x < t[root].value)
3          if (!t[root].left)
4              t[t[root].left = ++cnt] = Node(0, 0, 1, x);
5          else

```

```

6         insert(x, t[root].left, t, cnt);
7     else if (x > t[root].value)
8         if (!t[root].right)
9             t[t[root].right = ++cnt] = Node(0, 0, 1, x);
10        else
11            insert(x, t[root].right, t, cnt);
12    else
13        t[root].num++;
14    update(root, t);
15 }

```

#### 3.1.1.3.2. 利用端点和中点坐标的冲突检测算法

场景：

- 在用户设置活动或者查看活动时，需要检测是否与课程存在冲突

采用理由：

- 代码简洁高效

```

1 bool Student::interact(int x1, int x2, int y1, int y2) {
2     return (x1 + y1 - x2 - y2 <= x1 + x2 - y1 - y2) && (x1 + x2 - y1 - y2 <=
3     x2 + y2 - x1 - y1);
4 }

```

分析：

- 回忆中学判断两个圆的位置关系，关键在于把握半径和圆心距之间的数量关系。类比到线段，圆心距相当于中点距，直径相当于端点距离。

#### 3.1.1.4. 与其他模块交互

- 1.可以通过课表查询或者课程名称查询创建course实例，完成学生所需要的与课程有关的功能
- 2.与活动相关的功能都在本模块内部实现
- 3.通过路径导航功能可以创建guide实例，实现学生的导航需求

#### 3.1.1.5. 执行效果

- 课程名称查询

- |          |          |
|----------|----------|
| 1.课程名称查询 | 6.活动时间查询 |
| 2.课程课表查询 | 7.进行路径导航 |
| 3.课程时间查询 | 8.时间快进二倍 |
| 4.活动名称查询 | 9.系统时间暂停 |
| 5.进入活动菜单 | 0.注销登出账号 |

请选择您的操作: 1

请输入您要搜索的课程名(请在任意两个字符之间加空格): 语 言

Go语言基础

PHP语言基础

形式语言

现在请输入完整的课程名:

形式语言

请按任意键继续...

- 活动类型查询

请选择您的操作: 5

请输入查询类型: 班级

活动时间: Mon7:2-13:0 活动地点:台球馆 活动名称: 打台球 活动类型: 班级 闹钟属性: circular\_clock

检测到冲突!

活动时间: Mon12:20-14:20 活动地点: class 活动名称: study 活动类型: 班级 闹钟属性: circular\_clock

检测到冲突!

活动时间: Mon20:2-13:0 活动地点: 学生活动中心 活动名称: 猜谜 活动类型: 班级 闹钟属性: circular\_clock

未检测到与课程的冲突

请按任意键继续...

- 学生设置活动

请选择您的操作: 1

请输入活动的日期:

1: 星期一

2: 星期二

3: 星期三

4: 星期四

5: 星期五

6: 星期六

7: 星期日

请选择星期: 6

请以格式 小时 : 分钟 输入活动开始时间,输入小时后按回车:

10

:2

请以格式 小时 : 分钟 输入活动结束时间,输入小时后按回车:

11

:2

请输入活动地点: 学生活动中心

请输入活动名字(请将任意字符用空格分离): 活 动

活

动

未检测到与课程的冲突

活动设置完毕

请按任意键继续...

- 课程课表查询

欢迎学生: 王伦 登录! Tue Jun 14 15:41:16 2022

- |           |           |
|-----------|-----------|
| 1. 课程名称查询 | 6. 活动时间查询 |
| 2. 课程课表查询 | 7. 进行路径导航 |
| 3. 课程时间查询 | 8. 时间快进二倍 |
| 4. 活动名称查询 | 9. 系统时间暂停 |
| 5. 进入活动菜单 | 0. 注销登出账号 |

请选择您的操作: 2

课程表										
第1节	第2节	第3节	第4节	第5节	第6节	第7节	第8节	第9节	第10节	第11节
08:00-08:45	08:50-09:35	09:50-10:35	10:40-11:25	11:35-12:15	13:00-13:45	13:50-14:35	14:45-15:30	15:40-16:25	16:35-17:20	17:25-18:10
Mon 计网	计网	计组	计组	计组	计网课设	计网课设	计网课设			
Tue 计网	计网	毛概	毛概							
Wed 计网	计网	数据结构课设			体育	体育	形式语言	形式语言		
Thu Go语言基础	Go语言基础	毛概	毛概	电子电路	电子电路	Java	Java	Java	Python编程	
Fri Go语言基础	Go语言基础	PHP语言基础	PHP语言基础							

请输入想要进入的课程名称:

计网

请按任意键继续. . .

- 活动名称查找

欢迎学生: 王伦 登录! Tue Jun 14 15:52:28 2022

- |           |           |
|-----------|-----------|
| 1. 课程名称查询 | 6. 活动时间查询 |
| 2. 课程课表查询 | 7. 进行路径导航 |
| 3. 课程时间查询 | 8. 时间快进二倍 |
| 4. 活动名称查询 | 9. 系统时间暂停 |
| 5. 进入活动菜单 | 0. 注销登出账号 |

请选择您的操作: 4

请输入您要搜索的活动名 (请在任意两个字符之间加空格): 游

活动时间: Sar9:2-11:2 活动地点: 长城 活动名称: 旅游 活动类型: 个人 闹钟属性: once\_clock

活动时间: Sar19:5-23:5 活动地点: 故宫 活动名称: 旅游 活动类型: 个人 闹钟属性: once\_clock

活动时间: Sun6:10-8:10 活动地点: 颐和园 活动名称: 游园 活动类型: 个人 闹钟属性: no\_clock

请按任意键继续. . .

- 活动页面



活动页面

1. 添加个人活动

2. 删除个人活动

3. 修改个人活动

4. 个人活动闹钟

5. 活动类型查询

6. 返回个人主页

活动表				
活动时间: Sun20:10-22:10	活动地点:S6	活动名称: sleep	活动类型: 个人	闹钟属性: once_clock
未检测到与课程的冲突				
活动时间: Sar21:20-23:20	活动地点:S6	活动名称: take_a_shower	活动类型: 个人	闹钟属性: circular_clock
未检测到与课程的冲突				
活动时间: Mon6:0-7:0	活动地点:playground	活动名称: run	活动类型: 个人	闹钟属性: no_clock
未检测到与课程的冲突				
活动时间: Mon6:0-11:20	活动地点:playground	活动名称: play	活动类型: 个人	闹钟属性: no_clock
检测到冲突!				
活动时间: Tue10:30-11:20	活动地点:篮球场	活动名称: 打篮球	活动类型: 个人	闹钟属性: no_clock
检测到冲突!				
活动时间: Mon3:3-4:4	活动地点:宿舍	活动名称: 睡觉	活动类型: 个人	闹钟属性: no_clock
未检测到与课程的冲突				
活动时间: Mon1:1-1:2	活动地点:排球场	活动名称: 打排球	活动类型: 个人	闹钟属性: no_clock
未检测到与课程的冲突				
活动时间: Wed10:1-10:2	活动地点:宿舍	活动名称: 起床	活动类型: 个人	闹钟属性: no_clock
检测到冲突!				
活动时间: Tue11:1-12:1	活动地点:肯德基	活动名称: 聚餐	活动类型: 个人	闹钟属性: no_clock
检测到冲突!				
活动时间: Tue15:0-16:0	活动地点:麦当劳	活动名称: 喝水	活动类型: 个人	闹钟属性: once_clock
未检测到与课程的冲突				
活动时间: Wed22:0-23:0	活动地点:网吧	活动名称: 开黑	活动类型: 个人	闹钟属性: circular_clock
未检测到与课程的冲突				
活动时间: Thu9:2-10:2	活动地点:playground	活动名称: 升旗	活动类型: 个人	闹钟属性: no_clock
检测到冲突!				
活动时间: Thu18:5-19:5	活动地点:全聚德	活动名称: 烤鸭	活动类型: 个人	闹钟属性: once_clock
未检测到与课程的冲突				
活动时间: Fri8:0-9:0	活动地点:邮局	活动名称: 志愿	活动类型: 个人	闹钟属性: circular_clock
检测到冲突!				
活动时间: Sar9:2-11:2	活动地点:长城	活动名称: 旅游	活动类型: 个人	闹钟属性: once_clock
未检测到与课程的冲突				
活动时间: Sar19:5-23:5	活动地点:故宫	活动名称: 旅游	活动类型: 个人	闹钟属性: once_clock
未检测到与课程的冲突				
活动时间: Sun6:10-8:10	活动地点:颐和园	活动名称: 游园	活动类型: 个人	闹钟属性: no_clock
未检测到与课程的冲突				
活动时间: Sun8:20-11:50	活动地点:圆明园	活动名称: 踏青	活动类型: 个人	闹钟属性: no_clock
未检测到与课程的冲突				
活动时间: Mon12:20-14:20	活动地点:class	活动名称: study	活动类型: 班级	闹钟属性: circular_clock
检测到冲突!				
活动时间: Mon20:2-13:0	活动地点:学生活动中心	活动名称: 猜谜	活动类型: 班级	闹钟属性: circular_clock
未检测到与课程的冲突				
活动时间: Mon7:2-13:0	活动地点:台球馆	活动名称: 打台球	活动类型: 班级	闹钟属性: circular_clock
检测到冲突!				
活动时间: Sar10:2-11:2	活动地点:学生活动中心	活动名称: 活动	活动类型: 个人	闹钟属性: circular_clock
未检测到与课程的冲突				

请选择您的操作: 4  
 请输入你想改变的活动的开始时间:  
 1: 星期一  
 2: 星期二  
 3: 星期三  
 4: 星期四  
 5: 星期五  
 6: 星期六  
 7: 星期日

## 3.1.2. 管理员用户

### 3.1.2.1. 数据结构说明和数据字典

- 1.struct single\_course\_a

一堂课，管理员在修改课程时需要将文件内容存储到该结构体中，在内存中对该结构体进行修改后再重新存入文件

- 2.whole\_course\_a

一门课，管理员在发布课程时需要在内存中对该结构体的变量进行赋值后再新建相关文件

- 3.single\_activity\_a

一次活动，管理员在修改班级活动时需要将班级活动文件内容存储到该结构体中，在内存中对该结构体进行修改后再重新存入文件

### 3.1.2.2. 实现的功能

- 1.班级活动菜单（对班级活动进行增加和删除）
- 2.发布新的课程
- 3.修改原有课程
- 4.根据活动类型进行查找并且将结果按照时间排序

### 3.1.2.3. 与其他模块交互

- 管理员对文件的修改将会影响到其他用户在之后读取各种数据的情况

### 3.1.2.4. 执行效果

- 管理员修改考试时间

欢迎管理员：admin 登录！ Tue Jun 14 11:27:58 2022

1. 班级活动管理

2. 发布新的课程

3. 修改原有课程

0. 注销登出账号

请选择您的操作：3

请输入你想改变的课程的id:0001

请做出选择：

1.修改考试时间 2.修改考试地点 3.删除一堂课 4.增加一堂课 5.提交并返回  
1

请输入考试的新时间： Mon 10 2 11 2

请做出选择：

1.修改考试时间 2.修改考试地点 3.删除一堂课 4.增加一堂课 5.提交并返回  
5

- 管理员发布班级活动

1. 班级活动管理

2. 发布新的课程

3. 修改原有课程

0. 注销登出账号

请选择您的操作: 1

请输入你想改变的班级的id:2020211310

请做出选择:

1. 删除活动 2. 增加活动 3. 提交并返回

2

请输入增加的活动的日期, 时间区间, 地点, 名字长度, 名字 (名字的任意字符之间请加空格)

Mon

6 2

7 2

台球馆

3

打 台 球

请做出选择:

1. 删除活动 2. 增加活动 3. 提交并返回

3

### 3.1.3. 教师用户

#### 3.1.3.1. 数据结构说明和数据字典

- `map<string, string name_to_id`

存储从文件中读到的课程名和id的映射, 用户在输入课程名称后可以进入到课程文件中读取信息。

#### 3.1.3.2. 实现的功能

- 发布作业
- 批改作业

#### 3.1.3.3. 与其它模块的交互

- 发布作业后选了该课程的所有学生在 `homework_set` 中会多出一份作业文件夹, 学生可以在下一次进入该课程页面后发现老师布置了作业
- 批改作业后会吧分数写入文件, 学生可以在下一次进入该课程页面后看到老师给这次作业多少分

#### 3.1.3.4. 执行效果

- 教师布置作业

1.教师布置作业

2.教师批改作业

0.注销登出账号

请选择您的操作：1

请输入该课程的名称：

计网课设

请输入您要布置的作业次数：

4

请输入您要布置的作业的描述(字符请用空格分离)

W S

请按任意键继续...

- 教师批改作业

1.教师布置作业

2.教师批改作业

0.注销登出账号

请选择您的操作：2

请输入要批改作业的课程名称：计网

请输入要批改的作业次数：1

学生 2020211591 与学生 2020211590 本次作业作业重复率为 0%

学生 2020211590 与学生 2020211591 本次作业作业重复率为 0%

请按任意键继续...

请给出学生2020211591的分数：

58

请给出学生2020211590的分数：

85

### 3.1.3.5. 查重算法详述

在教师批改作业功能实现中，需要调用查重算法对学生提交的作业进行查重。本课程设计实现了近似线性的高效查重算法，内部实现采用 Aho-Corasick algorithm (AC自动机算法)，下面为查重算法设计的详细讲解。

#### 3.1.3.5.1. AC 自动机引入

##### 1. AC 自动机概述<sup>2</sup>

在[计算机科学](#)中，**Aho-Corasick算法**是由 [Alfred V. Aho](#) 和 Margaret J. Corasick 发明的字符串搜索算法，用于在输入的一串字符串中匹配有限组“字典”中的子串。它与普通字符串匹配的不同点在于同时与所有字典串进行匹配。算法均摊情况下具有近似于线性的[时间复杂度](#)，约为字符串的长度加所有匹配的数量。

##### 2. AC 自动机简介<sup>3</sup>

AC 自动机是**以 Trie 的结构为基础**，结合 KMP 算法的思想建立的。简单来说，建立一个 AC 自动机有两个步骤：

- 基础的 Trie 结构：将所有的模式串构成一棵 Trie。
- KMP 的思想：对 Trie 树上所有的结点构造失配指针。

然后就可以利用它进行多模式匹配了。

#### 3.1.3.5.2. 字典树构建

AC 自动机在初始时会把若干个模式串丢到一个 Trie 里，然后在 Trie 上建立 AC 自动机。这个 Trie 就是普通的 Trie，该怎么建怎么建。

这里需要仔细解释一下 Trie 的结点的含义，尽管这很小儿科，但在之后的理解中极其重要。Trie 中的结点表示的是某个模式串的前缀。我们在后文也将其称作状态。一个结点表示一个状态，Trie 的边就是状态的转移。

形式化地说，对于若干个模式串  $s_1, s_2 \dots s_n$ ，将它们构建一棵字典树后的所有状态的集合记作  $Q$ 。

#### 3.1.3.5.3. 失配指针

AC 自动机利用一个 fail 指针来辅助多模式串的匹配。

状态  $u$  的 fail 指针指向另一个状态  $v$ ，其中  $v \in Q$ ，且  $v$  是  $u$  的最长后缀（即在若干个后缀状态中取最长的一个作为 fail 指针）。我在这里简单对比一下这里的 fail 指针与 KMP 中的 next 指针：

1. 共同点：两者同样是在失配的时候用于跳转的指针。
2. 不同点：next 指针求的是最长 Border（即最长的相同前后缀），而 fail 指针指向所有模式串的前缀中匹配当前状态的最长后缀。

因为 KMP 只对一个模式串做匹配，而 AC 自动机要对多个模式串做匹配。有可能 fail 指针指向的结点对应着另一个模式串，两者前缀不同。AC 自动机在做匹配时，同一位上可匹配多个模式串。

#### 3.1.3.5.4. 构建 fail 指针

下面介绍构建 fail 指针的**基础思想**：

构建 fail 指针，可以参考 KMP 中构造 Next 指针的思想。

考虑字典树中当前的结点  $u$ ， $u$  的父结点是  $p$ ， $p$  通过字符  $c$  的边指向  $u$ ，即  $trie[p, c] = u$ 。假设深度小于  $u$  的所有结点的 fail 指针都已求得。

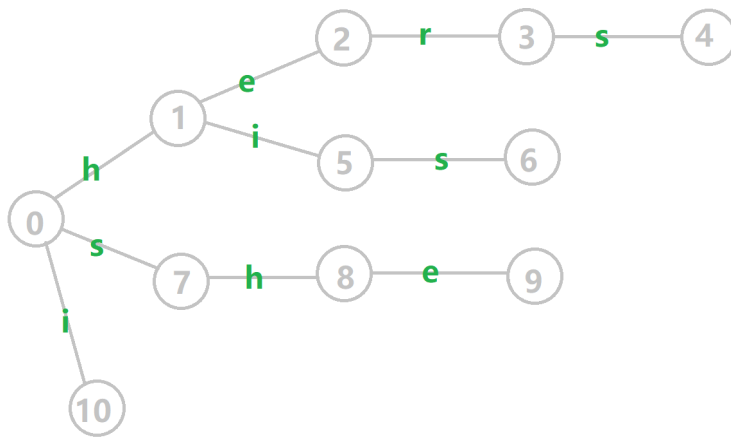
1. 如果  $\text{trie}[\text{fail}[p], c]$  存在：则让  $u$  的 fail 指针指向  $\text{trie}[\text{fail}[p], c]$ 。相当于在  $p$  和  $\text{fail}[p]$  后面加一个字符  $c$ ，分别对应  $u$  和  $\text{fail}[u]$ 。
2. 如果  $\text{trie}[\text{fail}[p], c]$  不存在：那么我们继续找到  $\text{trie}[\text{fail}[\text{fail}[p]], c]$ 。重复 1 的判断过程，一直跳 fail 指针直到根结点。
3. 如果真的没有，就让 fail 指针指向根结点。

如此即完成了  $\text{fail}[u]$  的构建。

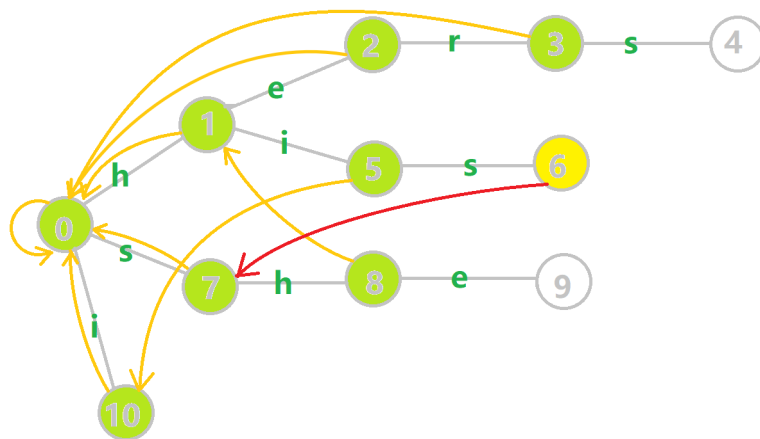
- 例子：

下面放一张 GIF 帮助理解。对字符串 `i he his she hers` 组成的字典树构建 fail 指针：

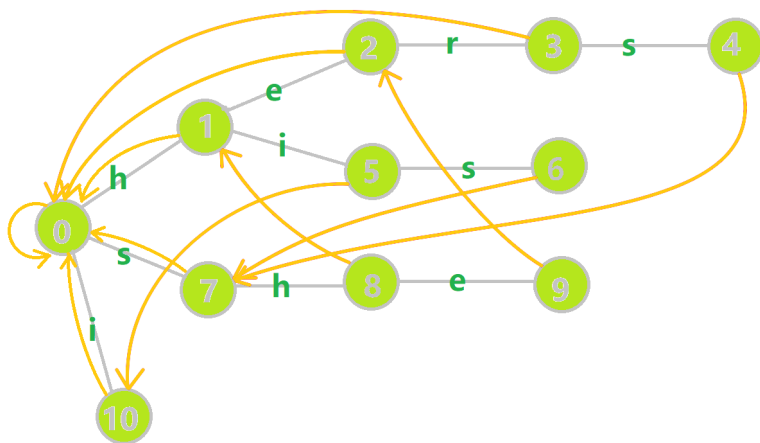
1. 黄色结点：当前的结点  $u$ 。
2. 绿色结点：表示已经 BFS 遍历完毕的结点，
3. 橙色的边：fail 指针。
4. 红色的边：当前求出的 fail 指针。



我们重点分析结点 6 的 fail 指针构建：



找到 6 的父结点 5， $\text{fail}[5] = 10$ 。然而 10 结点没有字母 `s` 连出的边；继续跳到 10 的 fail 指针， $\text{fail}[10] = 0$ 。发现 0 结点有字母 `s` 连出的边，指向 7 结点；所以  $\text{fail}[6] = 7$ 。最后放一张建出来的图



### 3.1.3.5.5. 字典树与 Trie 图

先来看构造函数 `build()`，该函数的目标有两个，一个是构建 fail 指针，一个是构建自动机。参数如下：

1. `tr[u, c]`：有两种理解方式。我们可以简单理解为字典树上的一条边，即 `trie[u, c]`；也可以理解为从状态（结点）`u` 后加一个字符 `c` 到达的状态（结点），即一个状态转移函数 `trans(u, c)`。下文中我们将用第二种理解方式继续讲解。
2. 队列 `q`：用于 BFS 遍历字典树。
3. `fail[u]`：结点 `u` 的 fail 指针。

```

1  // C++ Version
2  void build() {
3      for (int i = 0; i < 26; i++)
4          if (tr[0][i]) q.push(tr[0][i]);
5      while (q.size()) {
6          int u = q.front();
7          q.pop();
8          for (int i = 0; i < 26; i++) {
9              if (tr[u][i])
10                 fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
11             else
12                 tr[u][i] = tr[fail[u]][i];
13         }
14     }
15 }

```

解释一下上面的代码：build 函数将结点按 BFS 顺序入队，依次求 fail 指针。这里的字典树根结点为 0，我们将根结点的子结点——入队。若将根结点入队，则在第一次 BFS 的时候，会将根结点儿子的 fail 指针标记为本身。因此我们将根结点的儿子——入队，而不是将根结点入队。

然后开始 BFS：每次取出队首的结点 `u`（`fail[u]` 在之前的 BFS 过程中已求得），然后遍历字符集（这里是 0-25，对应 a-z，即 `u` 的各个子节点）：

1. 如果 `trans[u][i]` 存在，我们就将 `trans[u][i]` 的 fail 指针赋值为 `trans[fail[u]][i]`。这里似乎有一个问题。根据之前的讲解，我们应该用 while 循环，不停的跳 fail 指针，判断是否存在字符 `i` 对应的结点，然后赋值，但是这里通过特殊处理简化了这些代码。
2. 否则，令 `trans[u][i]` 指向 `trans[fail[u]][i]` 的状态。

这里的处理是，通过 `else` 语句的代码修改字典树的结构。没错，它将不存在的字典树的状态链接到了失配指针的对应状态。在原字典树中，每一个结点代表一个字符串  $S$ ，是某个模式串的前缀。而在修改字典树结构后，尽管增加了许多转移关系，但结点（状态）所代表的字符串是不变的。

而  $\text{trans}[S][c]$  相当于是  $S$  后添加一个字符  $c$  变成另一个状态  $S'$ 。如果  $S'$  存在，说明存在一个模式串的前缀是  $S'$ ，否则我们让  $\text{trans}[S][c]$  指向  $\text{trans}[\text{fail}[S]][c]$ 。由于  $\text{fail}[S]$  对应的字符串是  $S$  的后缀，因此  $\text{trans}[\text{fail}[S]][c]$  对应的字符串也是  $S'$  的后缀。

换言之在 Trie 上跳转的时候，我们只会从  $S$  跳转到  $S'$ ，相当于匹配了一个  $S'$ ；但在 AC 自动机上跳转的时候，我们会从  $S$  跳转到  $S'$  的后缀，也就是说我们匹配一个字符  $c$ ，然后舍弃  $S$  的部分前缀。舍弃前缀显然是能匹配的。那么 fail 指针呢？它也是在舍弃前缀啊！试想一下，如果文本串能匹配  $S$ ，显然它也能匹配  $S$  的后缀。所谓的 fail 指针其实就是  $S$  的一个后缀集合。

`tr` 数组还有另一种比较简单的理解方式：如果在位置  $u$  失配，我们会跳转到  $\text{fail}[u]$  的位置。所以我们可能沿着 fail 数组跳转多次才能来到下一个能匹配的位置。所以我们可以用 `tr` 数组直接记录记录下一个能匹配的位置，这样就能节省下很多时间。

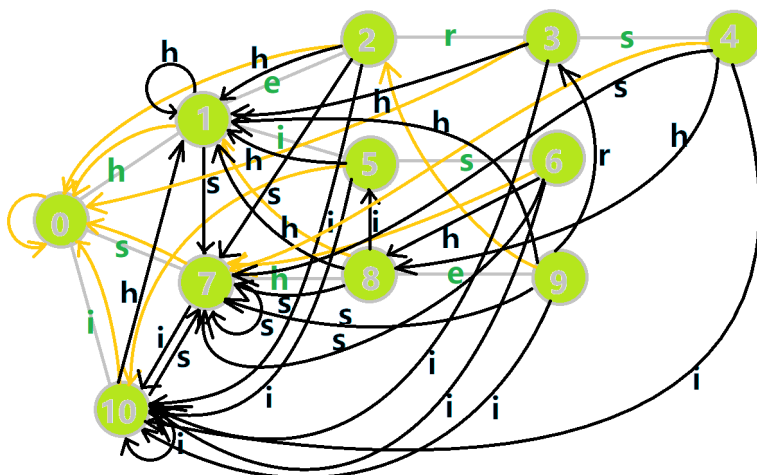
这样修改字典树的结构，使得匹配转移更加完善。同时它将 fail 指针跳转的路径做了压缩（就像并查集的路径压缩），使得本来需要跳很多次 fail 指针变成跳一次。

### 3.1.3.5.6. 多模式匹配

接下来分析匹配函数 `query()`：

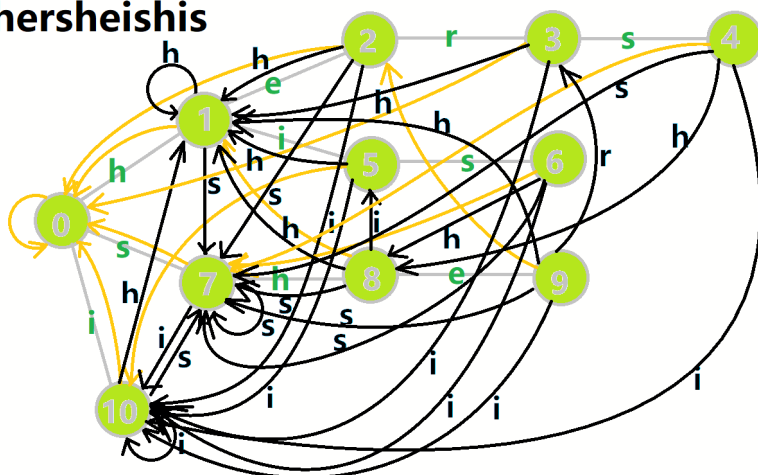
```
1 // C++ Version
2 int query(char *t) {
3     int u = 0, res = 0;
4     for (int i = 1; t[i]; i++) {
5         u = tr[u][t[i] - 'a']; // 转移
6         for (int j = u; j && e[j] != -1; j = fail[j]) {
7             res += e[j], e[j] = -1;
8         }
9     }
10    return res;
11 }
```

这里  $u$  作为字典树上当前匹配到的结点，`res` 即返回的答案。循环遍历匹配串， $u$  在字典树上跟踪当前字符。利用 fail 指针找出所有匹配的模式串，累加到答案中。然后清零。在上文中我们分析过，字典树的结构其实就是一个 `trans` 函数，而构建好这个函数后，在匹配字符串的过程中，我们会舍弃部分前缀达到最低限度的匹配。fail 指针则指向了更多的匹配状态。最后上一份图。对于刚才的自动机：





我们从根结点开始尝试匹配 `ushersheishis`，那么  $p$  的变化将是：



1. 红色结点:  $p$  结点
2. 粉色箭头:  $p$  在自动机上的跳转,
3. 蓝色的边: 成功匹配的模式串
4. 蓝色结点: 示跳 fail 指针时的结点 (状态)。

时间复杂度：定义  $|s_i|$  是模板串的长度， $|S|$  是文本串的长度， $|\Sigma|$  是字符集的大小。如果连了 trie 图，时间复杂度就是  $O(\sum |s_i| + n|\Sigma| + |S|)$ ，其中  $n$  是 AC 自动机中结点的数目，并且最大可以达到  $O(\sum |s_i|)$ 。如果不连 trie 图，并且在构建 fail 指针的时候避免遍历到空儿子，时间复杂度就是  $O(\sum |s_i| + |S|)$ 。

### 3.1.3.5.7. 查重算法实现

经过上面对 AC 自动机基本的初步认识，我们可以知道 AC 自动机可以实现字符串的多模式快速匹配，如果你没有看懂具体的细节这不要紧，因为我们的查重算法重点在于应用 AC 自动机。先来看一个非常模板的 AC 自动机问题，题目出处为杭电 OJ **HDU 2222**:

给定  $n$  个长度不超过 50 的由小写英文字母组成的单词，以及一篇长为  $m$  的文章。问其中有多少个单词在文章中出现了。注意：每个单词不论在文章中出现多少次，仅累计 1 次。

输入格式：第一行包含整数  $T$ ，表示共有  $T$  组测试数据。对于每组数据，第一行一个整数  $n$ ，接下去  $n$  行表示  $n$  个单词，最后一行输入一个字符串，表示文章。

输出格式：对于每组数据，输出一个占一行的整数，表示有多少个单词在文章中出现。

数据范围:  $1 \leq n \leq 10^4, 1 \leq m \leq 10^6$

输入样例：

1	1
2	5
3	she
4	he
5	say
6	shr
7	her
8	yasherhs

输出样例:

1 | 3

此问题可以用 AC 自动机实现模式匹配字符串，做到线性复杂度。做法也非常简单，将所有短串插入建立 AC 自动机，然后使用母串在 AC 自动机上进行字符串匹配统计计数即可。

知道如何解决这个问题，那么自然也就可以使用这个思路来进行文本的查重了。假设我们要查重学生 A 以及学生 B 本次作业的重复率，将学生 A 的作业分割为长度为 10 的一些字符串，然后插入构建 AC 自动机。对于学生 B 的作业，就是上面所述问题的母串了，我们只需要对母串在已经构建好的 AC 自动机上做字符串匹配，然后统计有多少学生 A 作业中的长度为 10 的字符串在学生 B 的作业中出现过即可。

假设字符串重复次数为  $X$ ，将学生 A 和学生 B 的作业分别分割为长度为 10 的字符串后，分别得到的字符串数量为  $C_1, C_2$ ，那么定义查重率  $V = 2X / (C_1 + C_2)$ ，此查重率定义较为合理，考虑完全雷同的文本，查重率显然为 100%，对于完全不同的文本，查重率为 0，其余情况介于这个范围内。

### 3.1.3.5.8. 复杂度分析

由于每次查重都要对所有提交学生的作业两两查重，假设  $T$  为两个查重对象作业文本均摊长度， $C$  为学生人数，

那么查重一次作业时间复杂度  $O(TC^2)$ 。

最后附上算法实现的核心代码，完整源码位于教师用户文件 `teacher.cpp` 中：

```
1 void Teacher::check_duplicate(string course_id, int times, vector<string>
   ids) {
2     // cout << "\n查重函数 Debug Result: \n\n";
3     string hw_folder = "../src/model/identity_model/homework_set/" +
4         this->teacher_id + "_teacher/" + course_id +
5         "_course/" + to_string(times) + "_times/";
6     //遍历每个学号
7     for (auto stu_id : ids) {
8         string hw_file = hw_folder + stu_id + "_stu/" + "hw.txt"; //当前学
   生作业
9         ifstream ifs_hw;
10        ifs_hw.open(hw_file, ios::in);
11        if (!ifs_hw.is_open()) {
12            cout << "\n打开学生 " << stu_id << " 作业文件失败!\n";
13            system("pause");
14            return;
15        }
16        string hw_content; //作业中所有字符串拼接起来的结果
17        string hw_lines;
18        while (ifs_hw >> hw_lines) hw_content += hw_lines;
19        // Aho-Corasick algorithm, AC自动机字符串匹配(Trie 图优化)
20        //分割为长度为10的字符串
21        //查重一次作业时间复杂度 $O(T \cdot C^2)$  T 为两个查重对象作业文本均摊长度,C 为学生人
   数
22        /*
23        other_hw_content 待匹配母串
24        hw_content 待分割的子串
25        */
26        vector<string> split_string;
27        set<string> filter1;
28        int cnt = 0;
29        string add_temp = "";
30        for (int i = 0; hw_content[i]; i++) {
31            if (cnt % 10 == 0) {
32                split_string.push_back(add_temp);
```

```

33         filter1.insert(add_temp);
34         add_temp = "";
35         cnt = 0;
36     }
37     add_temp += hw_content[i];
38     cnt++;
39 }
40 if (add_temp.size() != 0) split_string.push_back(add_temp);
41
42 for (auto other_id : ids) {
43     if (other_id != stu_id) {
44         string other_hw_file = hw_folder + other_id + "_stu/" +
"hw.txt"; //其他学生作业
45         ifstream ifs_others;
46         ifs_others.open(other_hw_file, ios::in);
47         string other_hw_content;
48         string other_hw_lines;
49         while (ifs_others >> other_hw_lines)
50             other_hw_content += other_hw_lines;
51         /*other_hw_content 待匹配母串
52         hw_content 待分割的子串
53         split_string 分割后的子串
54         split_pattern 分割后的母串*/
55         vector<string> split_pattern;
56         int cnt2 = 0;
57         set<string> filter2;
58         string add_temp2 = "";
59         for (int i = 0; other_hw_content[i]; i++) {
60             if (cnt2 % 10 == 0) {
61                 split_pattern.push_back(add_temp2);
62                 filter2.insert(add_temp2);
63                 add_temp2 = "";
64                 cnt2 = 0;
65             }
66             add_temp2 += other_hw_content[i];
67             cnt2++;
68         }
69         if (add_temp2.size() != 0)
split_pattern.push_back(add_temp2);
70         // AC自动机实现
71         const int N = 1000, S = 15; //约定作业最长1000*15 字符
72         // const int M = 10000; //母串长度
73         // int n;
74         int tr[N * S][130]; // trie
75         int cnt[N * S]; //每个节点结尾的10长度单词的数量
76         // char str[M];
77         string str;
78         int q[N * S]; // bfs宽搜队列
79         int ne[N * S];
80         int idx = 0; // trie idx
81         char init = 0;
82         // AC自动机插入字符串
83         auto insert = [&](string str) {
84             int p = 0; //根节点
85             for (int i = 0; str[i]; i++) {

```

```

86         int t = str[i] - init;
87         if (!tr[p][t]) tr[p][t] = ++idx;
88         p = tr[p][t];
89     }
90     cnt[p]++;
91 };
92 // bfs构建AC自动机
93 auto build = [&]() {
94     int hh = 0, tt = -1;
95     for (int i = 0; i < 128; i++) {
96         //将根节点的子节点全部入队 就是第一层
97         if (tr[0][i]) q[++tt] = tr[0][i];
98     }
99     while (hh <= tt) {
100         int t = q[hh++];
101         for (int i = 0; i < 128; i++) {
102             int& c = tr[t][i];
103             // trie图优化:
104             //如果j的子节点c不存在 那么就让c直接指向最终的位置
105             //这里的ne[]此时是递归定义的, 如果匹配下一个字母失败,
106             //就会直接跳到最终的位置
107             //不会在每一个fail指针处都判断是否存在该字母
108             if (!c)
109                 tr[t][i] = tr[ne[t]][i];
110             else {
111                 //如果子节点c存在 那就和朴素的写法一样 更新子节点的
112                 ne[c] = tr[ne[t]][i];
113                 q[++tt] = c;
114             }
115         }
116     }
117 };
118
119 // main logic
120 memset(tr, 0, sizeof tr);
121 idx = 0;
122 memset(cnt, 0, sizeof cnt);
123 memset(ne, 0, sizeof ne);
124 memset(q, 0, sizeof q);
125 for (auto s : split_string) {
126     insert(s);
127 }
128 build();
129 str = other_hw_content; //待匹配长母串
130 int res = 0;
131 for (int i = 0, j = 0; str[i]; i++) {
132     int t = str[i] - init;
133     // trie图优化:
134     j = tr[j][t]; //不用再while循环了
135     int p = j;
136     while (p) {
137         res += cnt[p];
138         cnt[p] = 0;
139         p = ne[p];

```

```

140         }
141     }
142     double dup_rate = 1.0 * res * 2 / (filter1.size() +
filter2.size());
143     cout << "\n学生 " << stu_id << " 与学生 " <<
144         other_id << " 本次作业作业重复率为 " << dup_rate * 100 <<
"%\n";
145     }
146 }
147 }
148 system("pause");
149 system("cls");
150 }

```

## 3.2. 学生课程模块

### 3.2.1. 数据结构说明和数据字典

- 1. `struct single_course`  
存储一堂课的信息单元
- 2. `struct material`  
存储一份电子资料的单元（包含权重信息）
- 3. `struct hw`  
存储一份作业的信息单元（包含成绩信息）
- 4. `vector<int order_hws`  
以成绩为关键字进行快速排序时，存放排序结果的下标
- 5. `vector<int order_materials`  
以权重为关键字进行快速排序时，存放排序结果的下标

### 3.2.2. 实现的功能

- 1. 提交课程作业
- 2. 提交课程资料
- 3. 下载课程资料
- 4. 作业名称模糊查询
- 5. 作业成绩排序
- 6. 资料名称模糊查询
- 7. 资料权重排序

### 3.2.3. 算法及分析

#### 3.2.3.1. 通过下标的转换进行快速排序

场景：

- 在进行资料按照权重进行排序，或者作业按照分数进行排序时

优缺点：

- 原始数据完全无序，采用快速排序效率最高

- 考虑到单个作业结构体或者资料结构体所占空间较大，我们在排序时只移动保存在另一个数组中的下标，可以节约无用数据进行转移耗费资源。

分析：

```

1  void Course::qsort_m(int l, int r) {
2      int i = l, j = r, flag = materials[order_materials[(l + r) / 2]].weight,
    tmp;
3      do {
4          while (materials[order_materials[i]].weight > flag) i++;
5          while (materials[order_materials[j]].weight < flag) j--;
6          if (i <= j) {
7              tmp = order_materials[i];
8              order_materials[i] = order_materials[j];
9              order_materials[j] = tmp;
10             i++;
11             j--;
12         }
13     } while (i <= j);
14     if (l < j) qsort_m(l, j);
15     if (i < r) qsort_m(i, r);
16 }

```

- 考虑到最好情况，假设每次都是均匀划分，则运算成本为：

$$T(n) = 2 - T(n/2) + n$$

递归展开后：

$$T(n) = 2 - 2[T(n/4) + n/2] + n = 2^k T(n/(2^k)) + kn$$

最后结束于 $T(1)$ ，即： $2^k = n$

可得：

$$T(n) = Cn + n \log n$$

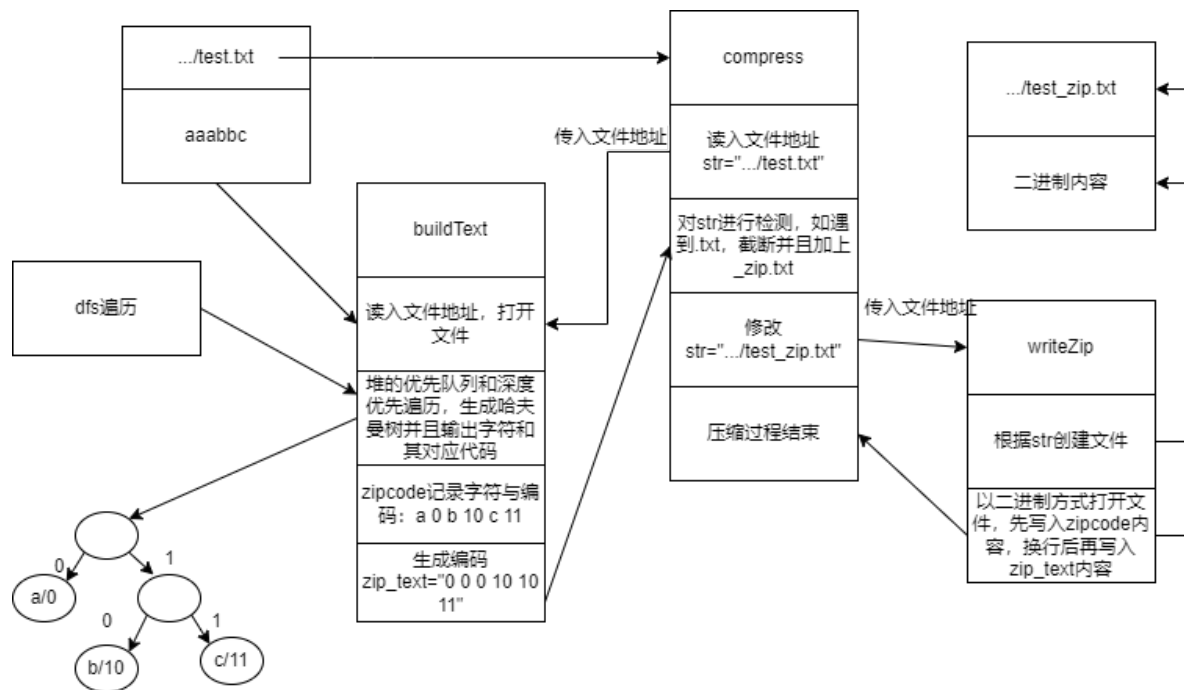
不难看出复杂度为  $O(n \log n)$ 。

### 3.2.3.2. 压缩文件 <sup>4</sup>

场景：

- 学生提交作业和提交材料

示意图：



算法实现：

- 哈夫曼树的结构为

```

1 struct zipnode {
2     int val; //保存字母的权值
3     char ch; //保存字母
4     zipnode -l, -r; //结点的左右孩子，初始时设置为空
5     zipnode(int v, zipnode- lc = nullptr, zipnode- rc = nullptr) : val(v),
6         l(lc), r(rc){};
7 }
8 struct cmp { //使用堆的优先排列生成树
9     bool operator()(zipnode- a, zipnode- b) {
10         return a->val > b->val;
11     };

```

- 有关量定义

```

1 unsigned int zip_lenstr; //记录文档的长度
2 zipnode- rt = nullptr; //初始化哈夫曼树
3 map<char, string> zip_code; //记录哈夫曼树生成的字符与对应的编码
4 string zip_text; //压缩后的文件编码以字符串的形式保存

```

- 在处理需要压缩的文件时，采用二进制方式读取文件

```

1 in.open(pathname, ios::binary);

```

- 采取堆的优先队列生成哈夫曼树,时间复杂度为 $O(n\log(n))$ 同时输出字母及其ASCII码对应的数值，以及字母对应的编码

```

1 priority_queue<zipnode-, vector<zipnode->, cmp> Q;
2 for (auto x : cnt) {
3     zipnode- p = new zipnode(x.second);
4     p->ch = x.first;

```

```

5         Q.push(p);
6     }
7     while (Q.size() >= 2) {
8         zipnode- lc = Q.top();
9         Q.pop();
10        zipnode- rc = Q.top();
11        Q.pop();
12        rt = new zipnode(lc->val + rc->val, lc, rc);
13        Q.push(rt);
14    }
15    dfs(rt, ""); //哈夫曼树的优先遍历
16    for (auto x : zip_code) {
17        if (isprint(x.first)) cout << x.first;
18        cout << "/" << (int)x.first << " : " << x.second << endl;
19    }

```

- 对哈夫曼树进行深度优先遍历，时间复杂度为 $O(n)$ ，dfs 函数如下

```

1 void dfs(zipnode- p, string s) {
2     if (p->l == nullptr && p->r == nullptr) {
3         zip_code[p->ch] = s;
4         return;
5     }
6     if (p->l != nullptr) dfs(p->l, s + '0');
7     if (p->r != nullptr) dfs(p->r, s + '1');
8 }

```

- compress 的逻辑如下。str 是输入的作业地址，由学生输入作业次数等信息后自动生成，buildText 函数用于生成哈夫曼树以及对原文件进行二进制读入，writezip 函数将编码以及哈夫曼树的信息写入新生成的压缩文件中。

```

1 void compress(string str) {
2     cout << "compress启动成功" << endl;
3     buildText(str.c_str());
4     int pos = str.find_last_of('.');
5     string str1 = str.substr(0, pos);
6     str = str1 + "_zip.txt"; //将新生成的文件命名为xxx_zip.txt
7     writezip(str.c_str());
8 }

```

优点：

- 由于压缩的文件不需要被看到，而二进制文件储存相对来说占空间小。于是参考了对二进制文件的处理方式，选择用二进制文件存储压缩后的信息，简单高效。

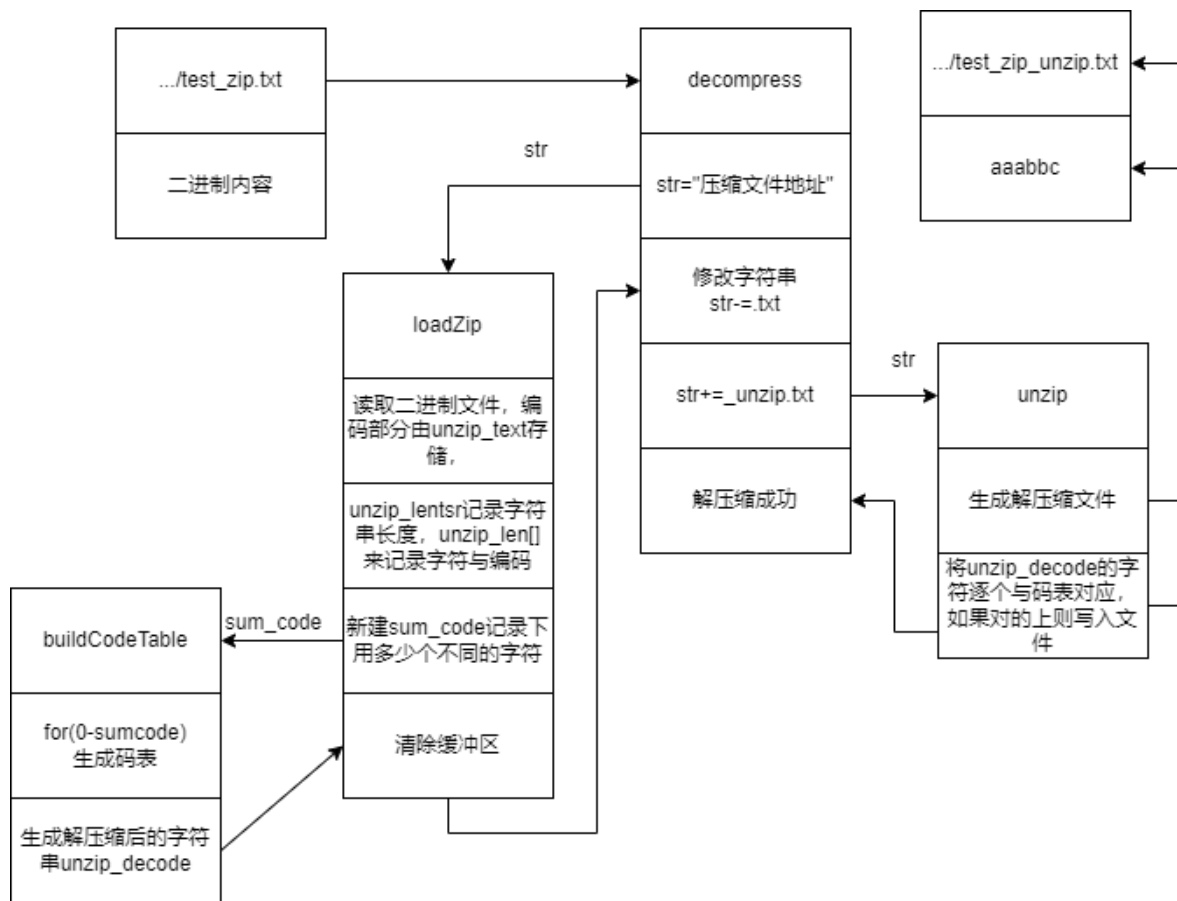
### 3.2.3.3. 解压缩文件

场景：

- 学生下载资料，老师批改作业

示意图：





算法实现：

- 有关变量定义

```

1 unsigned int unzip_lenstr; //记录哈夫曼码的长度
2 pair<char, int> unzip_chlen[257]; //存放生成的码表，包括字符和权值，最多存放257对
   信息
3 map<char, string> unzip_code; //辅助生成unzip_decode
4 map<string, char> unzip_decode; //存放解压缩后的字符信息
5 string unzip_text; //存放解压缩后的字符串
  
```

- `decompress` 逻辑如下，`str` 存放的是需要解压缩的文件地址，`loadZip` 函数读入压缩后的文件，生成码表，并且解压缩；`unzip` 函数将解压缩后的内容存放在 `*zip_unzip.txt` 文件中。

```

1 void decompress() {
2     string str;
3     cin >> str;
4     loadZip(str.c_str());
5     int pos = str.find_last_of('.');
6     string str1 = str.substr(0, pos);
7     str1 += "_unzip.txt"; //命名压缩后的文件。由于是压缩后再解压缩的，因此名字后缀改成
   _zip_unzip.txt
8     unzip(str1.c_str());
9 }
  
```

- 函数 `buildCodeTable` 作用是将压缩文件中的如何解压缩的信息读入并且生成码表，由函数 `loadZip` 调用。程序可以自行生成码表并解码，不需要手动输入，方便操作。

```

1 void buildCodeTable(int n) { //建立编码列表函数
2     int now = 1;
3     for (int i = 0; i < n; ++i) {
4         char now_char = unzip_chlen[now].first;
5         unzip_code[now_char] += unzip_text[i];
6         if (unzip_code[now_char].length() == unzip_chlen[now].second) {
7             unzip_decode[unzip_code[now_char]] = now_char; //生成解压缩字符串
8             now++;
9         }
10    }
11 }

```

- 函数 `unzip` 在写入文件的过程中，将 `unzip_decode` 的字符再次比对，确保正确再写入，相当于一个二次检查,时间复杂度为  $O(n)$ 。

```

1 void unzip(const char* pathname) { //保存
2     ofstream out;
3     out.open(pathname, ios::binary);
4     string now;
5     for (int i = 0; i < unzip_lenstr; ++i) {
6         now += unzip_text[i];
7         if (unzip_decode.count(now)) {
8             out.put(unzip_decode[now]);
9             now.clear();
10        }
11    }
12 }

```

优缺点：

- 解码信息存储在文件中，不必手工输入，操作简单。
- 由于和其他函数交互的原因，一次解压后再次解压会出现乱码。

### 3.2.4. 与其他模块交互

- 学生用户可以在个人主页中通过课程表选定课程或者通过搜索来创建课程类实例并进入课程界面实现课程提供的各种功能

### 3.2.5. 执行效果

- 提交课程资料

请输入资料的名字(任意字符请用空格分离)：

n e w

资料允许上传，请输入资料重要度：

5

compress启动成功

/-128 : 10110

/-29 : 101110

/10 : 1010110

/13 : 0111110

/32 : 111

"/34 : 01110

,/44 : 011110

-/45 : 101010100

./46 : 110011

A/65 : 00101001

B/66 : 101010111

C/67 : 001001

F/70 : 101010101

I/73 : 101010110

M/77 : 1010100

O/79 : 011111100

T/84 : 0010101

W/87 : 011111101

Y/89 : 00101000

a/97 : 1101

b/98 : 10101110

c/99 : 110010

d/100 : 11000

e/101 : 1001

f/102 : 101111

g/103 : 001011

h/104 : 10000

i/105 : 0000

k/107 : 0100000

l/108 : 10100

m/109 : 010001

n/110 : 0011

o/111 : 0110

p/112 : 001000

r/114 : 0001

s/115 : 10001

t/116 : 0101

u/117 : 01001

v/118 : 01111111

w/119 : 0100001

y/121 : 10101111

已提交

请按任意键继续...

- 资料权重排序

请选择您的操作：7

ip 权重：99

stopwait 权重：90

selective 权重：80

gbn 权重：10

udp 权重：2

tcp 权重：1

请按任意键继续...

- 作业名称查询

课程作业：

第1次作业 简单 成绩：0 状态：未交

第2次作业 困难 成绩：0 状态：未交

请选择您的操作：4

请输入您要搜索的作业名(请在任意两个字符之间加空格)：困

作业查询结果：

作业名：困难 成绩：0 状态：未交

请按任意键继续...

- 压缩解压测试

```
new.txt x tcp.txt udp.txt ip.txt 0002_course.txt M stu_info.txt 0003_course.txt 0004_course... 压缩上传...
src > model > identity_model > material_set > upload > new.txt
1 ude and Cold
2 A patron in Montreal cafe turned on a tap in the washroom and got scalded. "This is an outrage
3 "But, Monsieur, C stands for chaude - French for hot. You should know that if you live in Mont
4 "Wait a minute," roared the patron. "The other tap is also marked C."
5 "Of course," said the manager, "It stands for cold. After all, Montreal is a bilingual city."

rse.txt 0011_course.txt 0012_course.txt student.cpp 2020211591_activity_table.txt guide.cpp stu.txt M new_zip_unzip.txt X
src > model > identity_model > material_set > upload > new_zip_unzip.txt
1 Chaude and Cold
2 A patron in Montreal cafe turned on a tap in the washroom and got scalded. "This is an outrage
3 "But, Monsieur, C stands for chaude - French for hot. You should know that if you live in Mc
4 "Wait a minute," roared the patron. "The other tap is also marked C."
5 "Of course," said the manager, "It stands for cold. After all, Montreal is a bilingual city."
```

### 3.3. 校园导航模块

此模块代码实现和报告/文档撰写由 (2020211592 任晓斌) 实现

#### 3.3.1 需求实现概述

校园导航模块实现了所有课程设计中需求的功能，学生可根据多种方式进行导航，方式如下：

- 学生根据课程名称导航
- 学生根据上课时间导航
- 学生根据上课地点导航

同时学生导航可以有多种导航策略，而且考虑了跨校区等细节情况，方式如下：

- 最短距离策略导航
- 最短时间策略导航（考虑道路的拥挤度）
- 交通工具的最短时间策略（考虑不同的交通工具方式）
- 实现跨校区导航，自动识别是否需要跨校区进行路径规划

具体每个功能的实现细节，以及代码细节分析，文档后续会详细说明。

#### 3.3.2. 核心算法概述

在各种不同策略和要求的最短路实现代码中，均采用 [Floyd 算法](#)<sup>5</sup> 来预处理出全源最短路，且以文件保存所有预处理好的最短路结果，每次导航只需快速查询即可，不用每次导航都跑一次最短路算法。

为什么采用预预处理全源最短路，而不是每次导航都跑一次 Dijkstra 或者其它单源最短路算法呢？我们先分析一下每次导航都要跑 Dijkstra 算法的时间和空间复杂度。朴素实现的 Dijkstra 算法时间复杂度为  $O(mn)$ ，其中  $m$  为图中的边数， $n$  为图中的点数。考虑每次导航时如果在线建图一次，其 I/O 所用时间已经是近似  $O(n^2)$  时间复杂度，且导航在系统中是较为常用的部分，需要频繁调用。

所以综合考虑，使用 Floyd 算法只需要一次预处理以  $O(n^3)$  的时间花费即可求出整个地图的全源最短路，因为地图在后续的系统运行中是不会再被修改的，所以我们将预处理出的结果保存下来查询即可。

#### 3.3.3. 需求实现代码细节分析

接下来对导航模块中所有项目需求详细讲述源代码，实现思路，以及执行效果。在地图中，我们约定保留 1 号建筑物为校门，所有建筑设施的编号均在 1 ~ 20 内，所有课程导航**只按照在课表中有课的建筑之间进行路径导航**。

导航模块源码目录树：



### 3.3.3.1. 根据课程名称导航

在这个需求中，学生输入需要查询的上课名称，然后系统即可规划同时输出最短路策略。

首先来看程序的演示执行效果，在导航菜单中键入 1 进行根据课程名称导航：

导航页面
1. 根据课程名称导航
2. 根据上课地点导航
3. 根据上课时间导航
4. 经过固定地点导航
0. 退出当前导航系统

请输入您的操作：1\_

回车之后，系统首先会检索当前学生 2020211591 的课表，全部列出该学生本学期的课程，如下：

请输入您的操作：1

您的当前课程可选项为：

Go语言基础	Java	PHP语言基础	Python编程
电子电路	计网	计网课设	计组
毛概	数据结构课设	体育	形式语言

请输入您正在上的课程名称：计网

请输入您将要上课的课程名称：毛概

比如我们输入当前正在上的课程《计网》，然后输入接下来要上的课程《毛概》，键入回车之后，系统首先会自动识别是否需要跨校区上课，然后再输出导航规划的路径，对于这两门课，根据上面说的课表文件可知，是不需要跨校区的，都在沙河校区上课，所以输出结果如下：

已查询到您不需要跨校区上课，当前为沙河校区内导航：

.....  
已识别到您当前在：沙河校区 5号教学楼  
导航目的教学楼为：沙河校区 7号教学楼  
.....

最短步行距离路线如下：

5 号教学楼 → 6 号教学楼 → 沙河校门 → 3 号教学楼 → 7 号教学楼

最短步行路线总长度：433 米

最短步行时间路线如下：

5 号教学楼 → 8 号教学楼 → 16 号教学楼 → 7 号教学楼

最少步行所需时长约 201.333 秒

自行车道最短时间路线如下：

5 号教学楼 → 10 号教学楼 → 9 号教学楼 → 7 号教学楼

最少骑车所需时长约 92.75 秒

请按任意键继续. . . \_

导航会同时输出所有情况和策略下的最短路规划方案，供学生选择，比如最短步行距离策略，最短步行时间策略，以及通过骑自行车的交通方式，走自行车道的路线和所需时间。

然后再测试一个需要跨校区上课的例子，比如输入当前正在上的课程为《毛概》，接下来要上的课程为《Java》，输出结果如下：

```
请输入您正在上的课程名称：毛概
请输入您将要上课的课程名称：Java
.....
已查询到您需跨校区上课，将从沙河校区出发前往西土城校区：
.....
已识别到您当前在：沙河校区 7号教学楼
导航目的教学楼为：西土城校区 8号教学楼
.....

您可选乘每日定点校车路线如下：
-----
北邮沙河校区→沙河公交站→京藏高速沙河收费站→京藏高速→马甸桥→北邮海淀校区
-----

您可选乘公交车路线如下(乘坐12站 43分钟)：
-----
北邮沙河校区→沙河公交站→马甸桥东公交站→蓟门桥南公交站→北邮海淀校区
-----

您到达西土城校区之后的校内导航路线如下：
最短步行距离路线如下：
-----
西土城校门 → 18 号教学楼 → 14 号教学楼 → 8 号教学楼
-----
最短步行路线总长度：175 米
-----

最短步行时间路线如下：
-----
西土城校门 → 8 号教学楼
-----
最少步行所需时长约 83.3333 秒
-----

自行车道最短时间路线如下：
-----
西土城校门 → 28 号教学楼 → 18 号教学楼 → 9 号教学楼 → 15 号教学楼 → 8 号教学楼
-----
最少骑车所需时长约 132.5 秒
-----

请按任意键继续。 . . .
```

此时系统会同时加上跨校区导航，也就是从沙河校区至西土城校区，或者从西土城校区至沙河校区的导航规划。

同时提供定点班车路线: 北邮沙河校区->沙河公交站->马甸桥东公交站->蓟门桥南公交站->北邮海淀校区

校车途径路线: 北邮沙河校区->沙河公交站->京藏高速沙河收费站->京藏高速->马甸桥->北邮海淀校区

同样地，选择当在西土城校区上课，然后接下来要上的课输入一门沙河校区的，会以同样的逻辑进行导航，这里就不再赘述。以上就是以课程名称导航的执行效果，接下来分析这部分实现的代码和思路。

- `Guide` 导航类的声明, 源码位于当前模块目录下的 `guide.h`

```
1  class Guide {
2      public:
3          int build_now;           //现在所处的建筑位置
4          string campus_now;       //现在所处的校区 沙河 西土城
5          string stu_id;           //当前操作的学生的学号
6
7          Guide();
8          Guide(string in_campus, string stu_id); //现在所处的校区 当前操作学生的学号
9          Guide(string stu_id);               //当前操作学生的学号
10
11         void build_graph(string campus_now); //每次导航时候都要先建图 放入内存中
12
13         void guideOperMenu();                //导航选择菜单
14         void print_path_by_course();          //根据课程名称导航
15         void print_path_by_location();        //根据上课地点导航 考虑不同校区
16         void print_path_by_time();           //根据上课时间导航
17         void print_path_by_fixed_building(); //选做算法 经过固定建筑(同校区内的建筑)
18     };

```

这里声明了所有导航模块中的函数，功能由其后的注释说明，这一部分先讲述根据课程名称导航的函数细节。

- `void Guide::print_path_by_course()` 根据课程名称导航

在函数开始，我们先读入需要的数据，比如学生的课表信息，课表里面包含了每门课的名称，上课地点，建筑的编号等等，我们分别使用哈希表 `std::map` 来映射存储，这样每次查询都可以  $O(\log N)$  复杂度快速查询信息。

```

1 //课程表信息
2 //存入课程对应的建筑编号
3 map<string, int> mp;
4 map<string, string> campus_map; //课程 课程所在校区
5 set<string> all_course;
6 //星期 第几节 教室 课程名称 所在校区 课程编号 教室所在建筑
7 string file_date, file_classroom, file_course_name, file_campus;
8 int file_class_number, name_length;
9 string file_course_id, garbage;
10 int file_building_id;
11 while (ifs >> file_date >> file_class_number >> file_classroom >>
file_course_name >> file_campus >> file_course_id >> file_building_id >>
name_length) {
12     for (int z = 1; z <= name_length; z++) {
13         ifs >> garbage;
14     }
15     mp[file_course_name] = file_building_id;
16     campus_map[file_course_name] = file_campus; //所在校区
17     all_course.insert(file_course_name);
18 }

```

存储信息的核心代码如下，使用哈希表 `map<string, int> mp` 以及 `map<string, string> campus_map` 分别存储课程名称对应的建筑编号，以及课程名称对应的校区之间的映射关系。

接下来就是学生用户输入当前正在上课的名称，以及接下来要上课的名称，由于课程名称和其所在的建筑编号都已经保存在哈希表中，所以此时可以快速查询出这两门课所在的建筑编号，这样就可以查询已经求出的最短路径，然后输出就行了，代码如下：

- 首先根据校区关键字判断是否需要跨校区导航

```

1 cout << "\n请输入您正在上的课程名称：";
2 string course_go_on;
3 cin >> course_go_on;
4
5 cout << "请输入您将要上课的课程名称：";
6 string course_name;
7 cin >> course_name;
8
9 bool flag_shahe_xitu = false; //沙河->西土城 跨校区
10 bool flag_xitu_shahe = false; //西土城 -> 沙河 跨校区
11 string cps_now = campus_map[course_go_on];
12 string cps_next = campus_map[course_name];
13 if (cps_now == cps_next && cps_now == "沙河") {
14     cout << "\n已查询到您不需要跨校区上课，当前为沙河校区内导航：\n";
15     this->campus_now = "沙河";
16 } else if (cps_now == cps_next && cps_now == "西土城") {
17     cout << "\n已查询到您不需要跨校区上课，当前为西土城校区内导航：\n";
18     this->campus_now = "西土城";
19 } else if (cps_now == "沙河" && cps_next == "西土城") {
20     flag_shahe_xitu = true;
21     this->campus_now = "西土城";
22     cout << "\n已查询到您需跨校区上课，将从沙河校区出发前往西土城校区：\n";
23
24     // cout << "已到达西土城\n\n";
25 } else if (cps_now == "西土城" && cps_next == "沙河") {

```



```

26     flag_xitu_shahe = true;
27     this->campus_now = "沙河";
28     cout << "\n已查询到您需跨校区上课，将从西土城校区出发前往沙河校区：\n";
29
30     // cout << "已到达沙河\n\n";
31 }

```

- 读取地图数据文件，然后找到建筑编号

```

1  int now_build_id = mp[course_go_on]; //当前所在校区的建筑
2  int next_build_id = mp[course_name]; //目的校区所在建筑
3
4  ifstream iifs;
5  string path_file;
6
7  if (this->campus_now == "沙河")
8      path_file = "../../src/model/navigation_model/path1.txt"; //沙河校区
9  else
10     path_file = "../../src/model/navigation_model/path2.txt"; //西土城

```

接下来就是输出不同的导航策略下的路径规划方案，由于最短路径我们是已经预处理好的，所以只需要读取对应导航策略的路径数据文件然后输出方案即可，比如走自行车道的最短路方案在文件 `*bike_path.txt` 中。

- 最短距离策略

没有任何限制，没有拥挤度的最短距离路径预处理保存在文件 `path1.txt` (沙河校区)，以及 `path2` (西土城校区)

```

1  for (std::string line; std::getline(iifs, line);) {
2      vector<string> v; //去掉空格分开之后的所有单独建筑编号
3      string temp = "";
4
5      for (int i = 0; line[i]; i++) {
6          if (!isspace(line[i]))
7              temp += line[i];
8          else {
9              while (isspace(line[i])) i++;
10             i--;
11             v.push_back(temp);
12             temp = "";
13         }
14     }
15     v.push_back(temp);
16
17     if (stoi(v[0]) == now_build_id && stoi(v[1]) == next_build_id) {
18         // cout << line << endl;
19         for (int i = 2; i < (int)(v.size() - 2); i++)
20             cout << (v[i] == "1" ? (cps_next + "校门 -> ") : v[i] + " 号教学楼
-> ");
21         cout << v[v.size() - 2] + " 号教学楼" << endl;
22         cout << "-----
\n";
23         cout << "最短步行路线总长度：" << v[v.size() - 1] << " 米" << endl;
24         break;

```

```

25     }
26 }

```

输出规划路径的代码如上所示，本质逻辑就是线性复杂度  $O(n)$  遍历最短路方案，然后输出即可。

- 最短时间策略

在最短时间策略中，我们对地图道路附上拥挤度属性，此部分导航按照课设要求不考虑跨校区。道路的拥挤度采用随机数赋予，道路拥挤程度使用整数范围在  $[1, 3]$  内表示，生成具有拥挤度的源码位于 `gen_map_crowd.cpp`，生成地图的核心代码如下：

```

1 void gen() {
2     std::random_device seed;
3     std::mt19937 rand(seed());
4     std::uniform_int_distribution<int> dist(0, 999); //距离，单位为米，随机距离
5     std::uniform_int_distribution<int> building_id(0, 19); //建筑编号 [1,20]
6     std::uniform_int_distribution<int> crowdedness(0, 2); //道路拥挤程度
7     // [1,3]
8     ofstream ofs;
9     ofs.open("map2_crowd.txt");
10
11     //选210条边
12     //建筑距离>50m的才符合实际
13     int cnt = 0;
14     while (1) {
15         int build1 = building_id(rand) + 1;
16         int build2 = building_id(rand) + 1;
17         int random_dist = (dist(rand) + 1) * (crowdedness(rand) + 1); //距离
18         *拥挤度
19         if (random_dist < 50) continue;
20         if (build1 == build2) continue; //排除自环的情况
21         cnt++;
22         ofs << build1 << ' ' << build2 << ' ' << random_dist << endl;
23         if (cnt >= 210) break;
24     }
25     ofs.close();
26 }

```

根据上述有拥挤度的地图，预处理最短路方案的文件为 `path*_crowd.txt` 所以在查询最短路方案时查询这些文件。输出最短时间方案的核心代码如下：

```

1 //最短时间策略，重新生成地图考虑拥挤度 不考虑跨校区 步行不考虑交通方式
2 //输出步行时间 步速 1.5m/s
3
4 string crowd_path_file;
5 if (this->campus_now == "沙河")
6     crowd_path_file = "../src/model/navigation_model/path1_crowd.txt";
7     //沙河校区
8 else
9     crowd_path_file = "../src/model/navigation_model/path2_crowd.txt";
10
11 ifstream ifs_crowd;
12 ifs_crowd.open(crowd_path_file, ios::in);

```

```

12  if (!ifs_crowd.is_open()) {
13      cout << "拥挤度路径寻找文件不存在!" << endl;
14      system("pause");
15      return;
16  }
17
18  cout << "最短步行时间路线如下: \n";
19  cout << "-----\n";
20  for (std::string line; std::getline(ifs_crowd, line);) {
21      vector<string> v; //去掉空格分开之后的所有单独建筑编号
22      string temp = "";
23
24      for (int i = 0; line[i]; i++) {
25          if (!isspace(line[i]))
26              temp += line[i];
27          else {
28              while (isspace(line[i])) i++;
29              i--;
30              v.push_back(temp);
31              temp = "";
32          }
33      }
34      v.push_back(temp);
35
36      if (stoi(v[0]) == now_build_id && stoi(v[1]) == next_build_id) {
37          // cout << line << endl;
38          for (int i = 2; i < (int)(v.size() - 2); i++) cout << (v[i] == "1" ?
(cps_next + "校门 -> ") : v[i] + " 号教学楼 -> ");
39          cout << v[v.size() - 2] + " 号教学楼" << endl;
40          cout << "-----\n";
41          cout << "最少步行所需时长约 " <<
42              stoi(v[v.size() - 1]) * 1.0 / 1.5 << " 秒" << endl;
43          break;
44      }
45  }

```

- 交通工具策略导航

此导航策略要求校区内选择自行车时，只能走自行车道路，默认自行车在校区任何地点都有；在考虑不同拥挤度的情况下时间最短；对应的地图文件位于 `bike_map*.txt`，预处理求出的最短路方案位于文件 `bike_path*.txt` 中，输出走自行车道的导航方案核心代码如下：

```

1  //交通工具策略 考虑新地图 自行车道 考虑拥挤度
2  //自行车道重新生成地图
3  //只需要最短时间，行驶速度为 4m/s
4  //自行车道地图 bike_map1.txt bike_map2.txt bike_path1.txt bike_path2.txt
5  string bike_path_file;
6  if (this->campus_now == "沙河")
7      bike_path_file = "../src/model/navigation_model/bike_path1.txt"; //
沙河校区
8  else
9      bike_path_file = "../src/model/navigation_model/bike_path2.txt";
10
11  ifstream ifs_bike;
12  ifs_bike.open(bike_path_file, ios::in);

```

```

13  if (!ifs_bike.is_open()) {
14      cout << "自行车道路径寻找文件不存在!" << endl;
15      system("pause");
16      return;
17  }
18
19  cout << "自行车道最短时间路线如下: \n";
20  cout << "-----\n";
21  for (std::string line; std::getline(ifs_bike, line);) {
22      vector<string> v; //去掉空格分开之后的所有单独建筑编号
23      string temp = "";
24
25      for (int i = 0; line[i]; i++) {
26          if (!isspace(line[i]))
27              temp += line[i];
28          else {
29              while (isspace(line[i])) i++;
30              i--;
31              v.push_back(temp);
32              temp = "";
33          }
34      }
35      v.push_back(temp);
36
37      if (stoi(v[0]) == now_build_id && stoi(v[1]) == next_build_id) {
38          // cout << line << endl;
39          for (int i = 2; i < (int)(v.size() - 2); i++)
40              cout << (v[i] == "1" ? (cps_next + "校门 -> ") : v[i] + " 号教学楼
-> ");
41          cout << v[v.size() - 2] + " 号教学楼" << endl;
42          cout << "-----\n";
43          cout << "最少骑车所需时长约 " <<
44              stoi(v[v.size() - 1]) * 1.0 / 4.0 << " 秒" << endl;
45          break;
46      }
47  }

```

以上就是学生用户根据课程名称关键字进行导航，其核心代码逻辑的分析和说明。

### 3.3.3.2. 根据上课地点导航

在此功能需求中，学生需要输入上课的地点，系统为学生输出路径规划方案。

- 上课地点是上课的物理位置（例如 沙河 2号楼208）
- 起点和终点可以在不同校区，需要考虑校区间的交通方式；
- 校区间的交通方式为：定点班车（可以自行规划班次时刻表）和公共汽车（可等间隔发车）。

首先是程序执行的演示效果如下，和根据课程名称导航一样，系统会自动识别是否需要跨校区进行导航，如果不需要跨校区就输出在本校区内的导航，我们就以一个比较复杂的跨校区例子来演示，同样来到导航选择菜单：

导航页面
1. 根据课程名称导航
2. 根据上课地点导航
3. 根据上课时间导航
4. 经过固定地点导航
0. 退出当前导航系统

请输入您的操作：2  
请输入您当前所在教室(例如 沙河 2号楼208)：沙河 2号楼208  
请输入您将要去的教室(例如 沙河 9号楼510)：西土城 8号楼104

我们输入操作选项 2 进行根据上课地点来导航，然后接着输入当前所在的上课地点，以及接下来所要去的上课地点。在图示的测试样例中，我们输入当前位于沙河校区 2 号楼 208，将要抵达西土城校区 8 号楼 104 进行上课。

导航自动识别到需要跨校区，先输出校区间的导航路径，包括校车以及定点班车的方案。

然后分别输出所有导航策略，包括最短路径，最短时间，不同交通方式策略下的最短路径规划方案：

```
识别到您需要跨校区导航，请先沿如下路线步行至沙河校门口进行乘车：
-----
2 号教学楼 → 6 号教学楼 → 沙河校区门口
-----
最短步行路线总长度：334 米
-----

您可选乘每日定点校车路线如下：
-----
北邮沙河校区→沙河公交站→京藏高速沙河收费站→京藏高速→马甸桥→北邮海淀校区
-----

您可选乘公交车路线如下(乘坐12站 43分钟)：
-----
北邮沙河校区→沙河公交站→马甸桥东公交站→蓟门桥南公交站→北邮海淀校区
-----

您到达西土城校区之后的校内导航路线如下：
-----
最短步行距离路线如下：
西土城校门 → 18 号教学楼 → 14 号教学楼 → 8 号教学楼
-----
最短步行路线总长度：175 米
-----

最短步行时间路线如下：
-----
西土城校门 → 8 号教学楼
-----
最少步行所需时长约 83.3333 秒
-----

自行车道最短时间路线如下：
-----
西土城校门 → 20 号教学楼 → 10 号教学楼 → 9 号教学楼 → 15 号教学楼 → 8 号教学楼
-----
最少骑车所需时长约 132.5 秒
-----

请按任意键继续. . .
```

接下来我们讲述此功能实现的代码细节，同样的，和根据课程名称导航的函数类似，我们先读取学生课程表信息，然后使用哈希表分别存储上课教室和其所在对应的建筑编号，以及上课教室和其所在上课的校区之间的映射关系，这样对于 C++ 中的 `std::map` 我们每次都可以高效  $O(\log N)$  时间复杂度查询到建筑编号，进行导航。

```
1 //课程表信息
2 //存入课程对应的建筑编号
3 map<string, int> mp; //具体教室 教室所在建筑编号
4 // map<string, string> campus_map; //具体教室 教室所在校区
5 set<string> all_course;
6 //星期 第几节 教室 课程名称 所在校区 课程编号 教室所在建筑
7 string file_date, file_classroom, file_course_name, file_campus;
8 int file_class_number, name_length;
9 string file_course_id, garbage;
10 int file_building_id;
```

```

11 while (ifs >> file_date >> file_class_number >> file_classroom >>
    file_course_name >> file_campus >> file_course_id >> file_building_id >>
    name_length) {
12     for (int z = 1; z <= name_length; z++) {
13         ifs >> garbage;
14     }
15     mp[file_classroom] = file_building_id;
16     // campus_map[file_course_name] = file_campus; //所在校区
17     all_course.insert(file_course_name);
18 }

```

这部分代码其实也是和上面根据课程名称导航部分唯一不同地方，我们使用哈希表维护的查询信息不同，一个是根据课程名称，此处是根据上课地点。其余的代码均完全复用即可，因为导航最本质的逻辑都是要归到建筑的编号上，不论根据什么导航，我们先想办法找到目的的建筑编号，然后其余的代码逻辑完全照搬复用即可，此功能后续实现部分完全参考上面根据课程名称导航的讲述，以及源码的注释即可。

还有就是需要注意一点，对于课表中没有课的建筑，导航系统是读取不到建筑信息的，所以也就无法输出路线，在测试样例中**请务必根据学生课表信息的课程进行导航，请勿随意输入建筑设施的编号。**

### 3.3.3.3. 根据上课时间导航

在此功能需求中，需要学生输入打算上课的时间，然后系统为学生查询距离此时间最近的课程。如果当天已无课程，会提示学生输入改天的时间。

上课时间可以是“周五10点30分（Fri 10:30）”系统会自动更据学生的班级信息和最近的上课时间（未开始）查询上课地点。

首先是程序执行的演示效果如下，和根据课程名称导航一样，系统会自动识别是否需要跨校区进行导航，如果不需要跨校区就输出在本校区内的导航，我们就以一个比较复杂的跨校区例子来演示，同样来到导航选择菜单：

导航页面
1. 根据课程名称导航
2. 根据上课地点导航
3. 根据上课时间导航
4. 经过固定地点导航
0. 退出当前导航系统

请输入您的操作：3

请先输入您当前所在校区以及建筑/教学楼编号(例如 沙河 5)：沙河 5

请输入您要查询的上课时间(例如 Fri 10:30)：Fri 10:30

学生用户先输入当前所在校区以及当前所在的建筑编号，方便导航系统定位。然后学生需要输入自己打算上课的时间，比如周五早上 10:30 分，此处注意输入格式，使用英文星期，随后系统便开始导航规划路径输出如下：

```
.....
当前您所在校区为：沙河
当前所在校区建筑：5
查询课程的校区为：西土城
目的校区建筑编号：3
查询课程名结果为：PHP语言基础
.....

识别到您需要跨校区导航，请先沿如下路线步行至沙河校门口进行乘车：
-----
5 号教学楼 → 6 号教学楼 → 沙河校区门口
-----
最短步行路线总长度：269 米
-----

您可选乘每日定点校车路线如下：
-----
北邮沙河校区→沙河公交站→京冀高速沙河收费站→京冀高速→马甸桥→北邮海淀校区
-----

您可选乘公交车路线如下(乘坐12站 43分钟)：
-----
北邮沙河校区→沙河公交站→马甸桥东公交站→蓟门桥南公交站→北邮海淀校区
-----

您到达西土城校区之后的校内导航路线如下：

最短步行距离路线如下：
-----
西土城校门 → 18 号教学楼 → 3 号教学楼
-----
最短步行路线总长度：265 米
-----

最短步行时间路线如下：
-----
西土城校门 → 15 号教学楼 → 14 号教学楼 → 3 号教学楼
-----
最少步行所需时长约 262 秒
-----

自行车道最短时间路线如下：
-----
西土城校门 → 20 号教学楼 → 10 号教学楼 → 3 号教学楼
-----
最少骑车所需时长约 117.5 秒
-----

请按任意键继续. . . ■
```

导航效果如上图所示，系统首先会根据学生用户输入的上课时间，为学生查询到最近还未开始上的课程，包括这门课的校区（上图显示为西土城），这门课在目的校区所在的建筑编号（上图显示为西土城校区 3号楼），以及这门课的名称（上图显示为 PHP 语言基础）。接下来会根据是否需要跨校区导航，分别输出所有导航策略，包括最短路径，最短时间，不同交通方式策略下的最短路径规划方案。

接下来我们讲述这部分代码实现的细节和逻辑。学生课表的作息时间表完全使用北邮计算机学院大二下学期的课表作息，这里使用我自己的课表作息时间来模拟系统时间，如下图所示：



系统首先根据学生输入的时间，来查找最近未开始上的课，如果当天已无课程，会提示学生输入改天的时间。对于世界的查找，代码具体实现的逻辑就是很简单的判断输入的时间落入哪个课程时间段：

```

1  int class_number;
2  if (query_hour < 8)      //小于第一节课的时间
3      class_number = 1;  //每天的第一节课
4  else if (query_hour <= 8) {
5      if (query_minute == 0)
6          class_number = 1;
7      else if (query_minute <= 50)
8          class_number = 2;
9      else
10         class_number = 3;
11     else ...
12 }
13 ...

```

变量 `class_number` 存储根据学生输入的时间，查询得到的课程节号，以便于进一步查找更多上课的信息，比如课程所在的校区，课程所在的建筑编号，等等。



在这个需求中，我们需要使用哈希表维护的映射关系就更麻烦一些。因为需要维护的信息有：

- 课程所在星期及课程节号
- 课程名称及其对应所在建筑编号
- 课程名称及其对应所在的校区

所以需要在哈希表中维护更多信息，我们使用如下代码来实现初始数据的存储映射关系：

```
1  map<pair<string, int>, pair<string, int> > mp;    //{Mon, 1}, {计网, 5}> 星期
   课程节号 课程名称 建筑编号
2  map<string, string> campus_map;                  //课程 课程所在校区
3  set<string> all_course;
4  //星期 第几节 教室 课程名称 所在校区 课程编号 教室所在建筑
5  string file_date, file_classroom, file_course_name, file_campus;
6  int file_class_number, name_length;
7  string file_course_id, garbage;
8  int file_building_id;
9  while (ifs >> file_date >> file_class_number >> file_classroom >>
   file_course_name >> file_campus >> file_course_id >> file_building_id >>
   name_length) {
10     for (int z = 1; z <= name_length; z++) {
11         ifs >> garbage;
12     }
13     mp[{file_date, file_class_number}] = {file_course_name,
   file_building_id};
14     campus_map[file_course_name] = file_campus;    //所在校区
15     all_course.insert(file_course_name);
16 }
```

哈希表 `mp` 内嵌两个 C++ `std::pair` 来同时维护课程日期，课程节号，课程名称，以及上课所在建筑编号，哈希表 `campus_map` 映射课程名称以及课程所在的校区。

上面已经说过了，如何维护课程信息之间的映射关系是不同条件导航实现思路的唯一区别，最后只需要转换为查询建筑编号就行了，所以后面的代码，完全复用根据课程名称导航，或者根据课程地点导航部分。

### 3.3.3.4. 途径多个地点导航（选做算法部分）

在此功能需求中，需要实现通过学生用户指定必须经过的建筑编号来规划从起点到终点的最短路。

此算法实现源代码位于导航模块目录下的 `bonus_algo.cpp`，我们通过证明算法的正确性，分析算法的复杂度，最后再讲解源码的过程来依次讲述这部分实现的内容。

#### 1. 算法思路概述<sup>6 7</sup>：

首先将问题抽象为这样的图论问题：给定一个无向无负权图  $G(V, E)$ ，规定起点  $S$ ，终点  $T$ ，且给定  $K$  个图中互不相同的点  $a_1, a_2, a_3, \dots, a_k$ ，求出从  $S$  出发到  $T$ ，同时经过所有点  $a_i (i = 1 \sim k)$  的一条最短路。

初步分析此问题的最坏情况，即在  $G$  上求一条最短路，从  $S$  出发到  $T$  且必须经过所有点。此问题即为经典的 [Hamiltonian path problem](#)，在 TCS 学术界早已证明为 NP-complete 问题，目前无多项式时间复杂度解法。所以此问题最坏情况可归约至 NP 完全问题，下面考虑一般情况下的此问题。

对于只求出经过有限数量固定点的情况，将图  $G$  分割为两个点集  $G_1, G_2$ , 令

$G_1 = \{V_1 | V_1 \in a_i (i = 1 \sim k)\}$ ,  $G_2 = G \setminus G_1$ , 定义最短路  $Path(S, T, K)$  表示在此问题约束下的最短路状态空间。定义图  $G$  子集  $G_s$  的排列

$P(G_s) = \{permutation(g_1, g_2, g_3 \dots g_k) | g_1, g_2, g_3 \dots g_k \in G_s\}$ , 则最短路最优解的状态空间:

$$Path(S, T, K) = \min\{Path(S, g'_1, K) + \sum_{i=2}^{k-1} Path(g_i, g'_{i+1}, K) + Path(g'_k, T, K), k \geq 2\}$$

其中  $\{g'_1, g'_2, g'_3 \dots g'_k \in P(G_1)\}$  表示约束点集子图的一个排列,  $Path(S, T, K)$  为求得最优解的状态集合。

## 2. 算法正确性证明:

假设  $Path(S, T, K')$  同为此问题的最优解空间, 讨论约束集合  $K$  与  $K'$  的关系:

- $K' \subset K$ , 由  $\min\{Path(S, T, K)\} = \min\{Path(S, T, K')\}$  可知  $K' = K$  即最优解唯一, 为  $\min\{Path(S, T, K)\}$
- $K' \not\subset K$  且  $K' \cap K \neq \emptyset$ , 令  $K'' = K' \cap K$ , 则存在子图  $G_{K''}$  的一个排列  $P(G_{K''}) \notin P(G_1)$ , 这与所有的约束条件  $P(G_k) \in P(G_1)$  矛盾, 所以  $K' \not\subset K$  且  $K' \cap K \neq \emptyset$  不成立.
- $K' \cap K = \emptyset$ , 同第二种集合关系, 推导可知与所有的约束条件  $P(G_k) \in P(G_1)$  矛盾, 不成立.
- $K \subset K'$ , 同第一种集合关系. 由条件  $\min\{Path(S, T, K)\} = \min\{Path(S, T, K')\}$  可知  $K' = K$  即最优解唯一, 为  $\min\{Path(S, T, K)\}$

综上推导可知此最短路问题的唯一最优解状态空间为  $\min\{Path(S, T, K)\}$  不存在其它的最优解情况. QED.

## 3. 算法复杂度分析:

根据问题最优解的状态转移表达式来逐项分析:

$$Path(S, T, K) = \min\{Path(S, g'_1, K) + \sum_{i=2}^{k-1} Path(g_i, g'_{i+1}, K) + Path(g'_k, T, K), k \geq 2\}$$

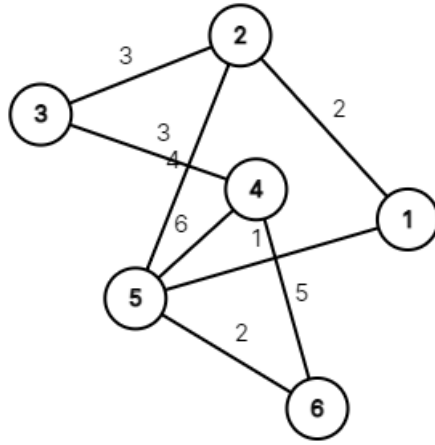
对于求解  $\sum_{i=2}^{k-1} Path(g_i, g'_{i+1}, K)$ , 需要  $O(K!)$  时间复杂度,  $K$  为约束集合大小, 也就是图中必须经过的点数集合大小。对于上式整体的求和结果, 必须考虑到排列  $P(G_1)$  的所有排列情况, 通俗来讲为  $g'_1$  以及  $g'_k$  的所有情况都需要考虑, 同时不断更新最优解。仔细思考一下, 这本质上就是要求任意两点之间的最短路, 那么可用的算法通常来说有 Floyd 算法求解全源最短路, 或者对于每个点都执行一次 Dijkstra 算法, 但是后者实现起来较为麻烦, 相比 Floyd 算法复杂度只有常数级别的优化, 故此处采用 Floyd 算法来实现。

所以算法最终的复杂度为  $O(N^3 + K!)$ , 其中  $N = |V|$ ,  $K = |G_1|$ 。

## 4. 算法代码实现和样例解释:

由于项目的地图中有超过20个建筑设施, 所以不便于使用校园地图来举例讲述。为了直观演示算法的正确性和效果, 我们此处构建一个较小的样例来说明, 此测试样例对应的地图文件位于导航模块下的

`shortest_path_test_map.txt`, 我们首先绘制此样例对应的无向带权图如下所示:



使用 [csacademy.com/app/graph\\_editor](https://csacademy.com/app/graph_editor) 绘制

考虑从 1 号点出发，终点为 6 号点，途中必须经过 2, 3, 4, 5 号点的测试样例。算法在开始执行前先使用 Floyd 算法预处理一次全源最短路，并记忆化保存，此样例中  $G_1 = \{2, 3, 4, 5\}$ ,  $|G_1| = 4$ . 令  $P(G_1) = \{a_1, a_2 \dots a_k\}$ , 则对于  $P(G_1)$  的每种排列，都求一次  $\sum_{i=2}^{k-1} Path(g_i, g_{i+1}, K)$ , 再加上  $Path(S, a_1, K) + Path(a_k, T, K)$  的值来持续更新最优解即可，用伪代码描述如下：

```

1 //Precomputation: Find all pairs shortest paths, e.g. using Floyd-warshall
2 n = number of nodes
3 for i=1 to n: for j=1 to n: d[i][j]=INF
4 for k=1 to n:
5     for i=1 to n:
6         for j=1 to n:
7             d[i][j] = min(d[i][j], d[i][k] + d[k][j])
8
9 //Now try all permutations
10 shortest = INF
11 for each permutation a[1],a[2],...a[k] of the 'mustpass' nodes:
12     shortest = min(shortest, d['start'][a[1]]+d[a[1]][a[2]]+...+d[a[k]]
13     ['end'])
14 print shortest

```

接下来为此功能的演示效果：

首先进入导航菜单，键入选项 4 选择经过固定点的导航，然后回车：

导航页面
1. 根据课程名称导航
2. 根据上课地点导航
3. 根据上课时间导航
4. 经过固定地点导航
0. 退出当前导航系统

请输入您的操作：4  
 请输入您当前所在校区：沙河  
 请输入您所在的建筑编号(例如 5)：5  
 请输入您的目的建筑编号(例如 3)：8

请输入您必须要经过的固定建筑物编号，  
 例如 3 2 3 5，第一个 3 代表个数，表示必须经过 2, 3, 5 三个点：4 1 2 3 4

已为您规划沙河校区最短路径如下：

5号建筑 → 6号建筑 → 1号建筑 → 3号建筑 → 4号建筑 → 2号建筑 → 14号建筑 → 11号建筑 → 8号建筑

最短路径距离总和为：1113 米

请按任意键继续。 . . .

首先输入当前所在校区，便于系统定位。接着输入当前所在建筑，以及目的所在建筑的编号，最后输入必须经过的建筑：先输入必须经过建筑的数量，然后依次输入每个必须经过的建筑编号。如上图所示样例，学生当前在 5 号建筑，导航前往 8 号建筑，途中必须经过 4 个点，分别为 1, 2, 3, 4 号建筑。最后导航输出规划好的最短路径。

最后附上我们在项目中对于此算法完整的代码实现：

```
1 void Guide::print_path_by_fixed_building() { //选做算法 经过固定地点
2     cout << "请输入您当前所在校区： ";
3     string campus_now;
4     cin >> campus_now;
5     cout << "请输入您所在的建筑编号(例如 5)： ";
6     int startp;
7     cin >> startp;
8
9     cout << "请输入您的目的建筑编号(例如 8)： ";
10    int endp;
11    cin >> endp;
12
13    //读入地图中的边
14    string path_file;
15    // test
16    // path_file =
17    "../src/model/navigation_model/shortest_path_test_map.txt";
18
19    if (campus_now == "沙河")
20        path_file = "../src/model/navigation_model/map1.txt"; //沙河校区
21    else
22        path_file = "../src/model/navigation_model/map2.txt"; //西土城
23
24    ifstream ifs;
25    ifs.open(path_file, ios::in);
26    if (!ifs.is_open()) {
27        cout << "\n打开地图路径文件失败!" << endl;
28        system("pause");
29        return;
30    }
31
32    const int N = 220, INF = 1e9;
33    int n = 210;
34    int d[N][N];
35    int P[N][N]; //记录DP过程中间点，还要记录路径
36    memset(P, 0, sizeof P);
37    // init dist
38    for (int i = 1; i <= n; i++)
39        for (int j = 1; j <= n; j++)
40            if (i == j)
41                d[i][j] = 0;
42            else
43                d[i][j] = INF;
44
45    auto floyd = [&]() {
46        for (int k = 1; k <= n; k++)
47            for (int i = 1; i <= n; i++)
48                for (int j = 1; j <= n; j++)
```

```

48         if (d[i][k] + d[k][j] < d[i][j]) {
49             d[i][j] = d[i][k] + d[k][j];
50             P[i][j] = k;
51         }
52     };
53
54     std::function<void(int, int)> output_path = [&](int u, int v) -> void {
55
56         // 输出 u v 最短路经过的所有点，输出不包括起点终点
57         if (P[u][v] == 0) return;
58         output_path(u, P[u][v]);
59         cout << P[u][v] << "号建筑 -> ";
60         output_path(P[u][v], v);
61         return;
62     };
63
64     //读入地图边
65     int a, b, x;
66     while (ifs >> a >> b >> x) {
67         d[a][b] = d[b][a] = min(d[a][b], x); //判断重边
68     }
69     // ifs.close();
70
71     floyd();
72
73     cout << "\n请输入您必须要经过的固定建筑物编号，\
74     \n例如 4 1 2 3 4，第一个 4 代表个数，\
75     表示必须经过 1,2,3,4 四个点：";
76     vector<int> node;
77     int cnt;
78     cin >> cnt;
79     for (int i = 0; i < cnt; i++) {
80         int x;
81         cin >> x;
82         node.push_back(x);
83     }
84     sort(node.begin(), node.end());
85
86     int res = INF;
87     vector<int> best_path;
88     do {
89         int dist = d[startp][node[0]];
90         for (int i = 0; i < cnt - 1; i++) dist += d[node[i]][node[i + 1]];
91         dist += d[node[cnt - 1]][endp];
92         if (dist < res) {
93             res = dist;
94             best_path = node;
95         }
96     } while (next_permutation(node.begin(), node.end()));
97
98     //目前只是得到最短路d[][]的表示路径
99     //对于每一步最短路，都输出详细的单步路径
100     vector<int> final_path;
101     final_path.push_back(startp);
102     for (auto t : best_path) final_path.push_back(t);

```

```

102     final_path.push_back(endp);
103
104     cout << "\n已为您规划" << campus_now << "校区最短路径如下: \n";
105     cout << "-----\n";
106
107     for (int i = 0; i < (int)final_path.size() - 1; i++) {
108         cout << final_path[i] << "号建筑 -> ";
109         output_path(final_path[i], final_path[i + 1]);
110     }
111     cout << final_path.back() << " 号建筑" << endl; //终点
112     cout << "-----
\n";
113     cout << "最短路径距离总和为: " << res << " 米" << endl;
114     cout << "-----
\n";
115
116     system("pause");
117     system("cls");
118     return;
119 }

```

以上就是整个导航模块所有功能实现的详细过程讲述，至此所有项目模块的文档说明完毕。

## 4. 总结和心得







本次课设代码量和工程量较大，我们投入了很多精力和时间，也有很大的收获。

团队协作方面，使用 Github 协同开发，很大提升了协作效率。我们通过线上开会这一方式确定计划，在项目推进过程中始终保持明确的分工和清晰的思路，达到  $1 + 1 + 1 > 3$  的效果。

软件工程方面，我们进一步理解了项目从启动到验收的全过程，并且在编写大量代码的过程中培养了思维和代码能力。

数据结构方面，我们写了二叉搜索树，Floyd 算法，哈夫曼树，动态规划，AC自动机等较为复杂的算法和数据结构，加深了我们对其的理解。

## 5. 参考文献

1. [Binary search tree](#) 
2. [Aho-Corasick algorithm](#) 
3. [OI-Wiki AC 自动机](#) 
4. [Huffman coding](#) 
5. [Floyd-Warshall algorithm](#) 
6. [Find the shortest path in a graph which visits certain nodes](#) 
7. [Dynamic programming](#) 