数据结构与编程
大作业实验报告

姓名： 李森洋

学号： 201811051123

专业： 地理信息科学

年级： 2018

日期： 2020.07.09

目录

1 问题描述

实现一个大学生必修课成绩管理系统，包含如下功能：

（1）线性表：

用线性表实现学生成绩单的：读、增、删、改、查

基础数据：自己随机生成数据，姓名学号和成绩都随机生成。建议编写程序，随机生成成千上万人的资料以便后面测试算法的性能。

实现下列字段（1）学号（长整型）（2）姓名（字符串）（3）7 门课程成绩（0-100 之内的整型）。课程包括：高等数学，计算机导论，离散数学，程序设计，数据结构，计算机原理，数据库原理。这些都是必修课，每个学生都要选。

1. 用 C++语言实现线性表。

2. 读：将名单整理为文本文件，用 C++编写程序读取，并存入表中。

3. 增：将一名同学加入成绩单。

4. 删：将一名同学移出成绩单。

5. 改：对每一个字段实现单独修改。

6. 查：通过学号查询同学的姓名和所有课程的成绩。

7. 打印输出：将整个成绩单输出到屏幕。

（2）二叉排序树与平衡二叉树：

按照每一门课程的成绩，分别构造二叉排序树，平衡二叉树。 查找某课程得分为 XX 分的学生姓名和学号，测试二叉排序树和平衡二叉树的查找性能。

（3）排序：

对每一门课程的成绩进行排序。 实现希尔排序，快速排序，堆排序，测试三种排序算法的性能。

（4）AOV 网：

7 门必修课之间的关系如下：

编号 课程名称 先修课

C1 高等数学 无

C2 计算机导论 无

C3 离散数学 C1

C4 程序设计 C1, C2

C5 数据结构 C3，C4

C6 计算机原理 C2，C4

C7 数据库原理 C4，C5，C6

按照上述关系，建立 AOV 网络（参考 ppt 第六章第 145 页）。对这个 AOV 网进行拓扑排序，判断是否存在回路

2 解题思路

本次大作业无非就是各种问题的叠加，在解题时只要秉承各个击破的原理，就没有太大问题，因此我们将解题思路拆解如下：

2.1 基础数据生成

要求中需要自己随机生成数据，随机生成数据是我们的老朋友了，我专门编写了 randome()函数来实现在这个过程，输入参数是设定学生个数，生成学生信息结构体数组。因为建议随机生成千上万人的资料以便后面测试算法的性能，在申请数组时通通使用动态数组。

首先安排随机数种子保证不同时间生成不同的随机数。关于成绩的随机生成比较简单，使用 rand()函数生成 0-100 之间的数即可。生成学号的的话，为了不使学号重复，我决定采用按顺序生成数字并遍历将每一个值利用 swap（）与随机的值交换的方式生成无序不重复数组。生成姓名稍微麻烦一些，从网上搜寻百家姓相关信息，将各种姓氏粗略按其规模分级写入文本，同时搜集常用取名的字，写入文本。通过该程序读取，各自存入 string 数组，关于名的 string 数组另外增加 200 个存储“ ”(空格字符串)的空间，以便生成不同长度的名字，最后随机生成姓名，每一个姓名由三个部分组成，第一个是

随机生成的姓，第二个是随机生成的单字名，第三个是随机生成的单子名名或空字符串。

2.2 线性表

为了方便之后的排序和二叉树构造，我们使用顺序表存储学生信息，该顺序表的实现比较简单，为了满足各自要求，设计了不同的成员。

关于读入数据，直接在构造函数中打开生成的文本，将数据读入学生信息结构体中，该结构体包含每个学生的姓名·学号·各科成绩·总成绩。增加学生信息根据插入位置，输入对应信息，插入后数组依次后移，表长度加一。删除学生信息根据输入的学生姓名，将对应信息全部删除，后续数组依次前移，表长度减一。修改信息根据输入学号查找到地址，直接进行修改即可。查找信息也是根据输入学号进行循环遍历匹配，最后获得对应结果。打印结果根据学生信息结构体，依次循环打印输出各种信息到屏幕即可。

与此同时也声明了希尔排序·快速排序·堆排序及其相关函数，整个将在 2.4 进行说明。

另外因为之后构造二叉树需要利用线性表中存储的数据，所以我额外在线性表中申请了排序二叉树和平衡二叉树的友元类（friend class）

2.3 二叉树查找

2.3.1 **二叉排序树**

我们将二叉排序树申请为线性表的友元类。二叉排序树可分为两个部分构造和查找:

构造部分申请新的动态数组，按想要排序的科目将线性表数据导入数组，之后调用插入结点函数进行构造。插入结点函数本身利用了递归，主要思路就是判断将新数据与根结点比较，小于则递归到左子树，否则递归到右子树，直到这个节点为 NULL。这里尤其需要注意的是由于在构造函数中调用了插入结点函数，因此传递的是形式参数，在插入结点函数外是不会改变的，因此我们必须将输入的指向结点的指针改变为指向结点的指针的指针，即改变这个指针的地址。

查找部分相对构造比较简单，也是递归思路，输入数字，与根节点数据比较，小于则递归到左子树，否则则递归到右子树，若相等则需要继续追溯右子树直到不相等。这里要注意的是，由于是递归函数，但要用到私有成员根结点，因此需要在外边再包一层马甲用来调取根节点。

2.3.2 **平衡二叉树**

我们也将平衡二叉树申请为线性表的友元类。平衡二叉树可分为三个部分：构造，调整，查找:

构造部分申请新的动态数组，按想要排序的科目将线性表数据导入数组，之后也像排序二叉树调用插入结点函数进行构造。基本思路都是一

样，除了要注意将输入的指向结点的指针改变为指向结点的指针的指针外，为了调整方便，需要构造一个的父母节点关系（即->parent）。

平很二叉树的关键的是调整，每次插入一个结点就要执行调整步骤。首先是验证其是否为平衡二叉树，这里构造 balance()函数进行验证，思路是对于插入的最新节点 A 不停上溯，每上溯一个结点计算其左右子树的深度差的绝对值，若存在一个结点其左右深度差为 2，则导出该结点 B。接着对结点 A 与 B 的关系进行判断，如果 A 在 B 的左孩子的左子树上，则执行 LL 变幻；如果 A 在 B 的左孩子的右子树上，则执行 LR 变幻；如果 A 在 B 的右孩子的左子树上，则执行 RL 变幻；如果 A 在 B 的右孩子的右子树上，则执行 RR 变幻。关于变幻函数，因为也是调用函数，输入结点 B 的指针的地址，对于 LL 变幻，将 B 与其两亲关系转移到 B 的左孩子上，将 B 变为其左孩子的右孩子，将 B 的左孩子的右孩子（如果有的话）转为 B 的左孩子，注意操作顺序。RR 变幻与 LL 变幻类似。值得注意的是，每次变换后根结点可能会发生改变，因此需要验证到操作树顶部的最新结点是否有父母结点，若没有，则该结点就是最新根结点。至于 LR 变幻，是以输入 B 结点的左孩子地址为参数的 RR 变幻然后 B 地址的 LL 变幻的结合。RL 与 LR 变幻同理，此处不表。

查找部分与排序二叉树类似，但是对于多个相同值的查找，因为平衡二叉树进行调整过后的缘故，不是直接追溯右子树就可以完全获得的，因此我设计了两种查找模式，第一种是类似排序二叉树的查找模式，用来最大程度发挥平衡二叉树查找性能以便测试，第二种是可以查找到全部

相同值，但会牺牲部分速度，这部分具体在【讨论与结论】种进行补充说明。

2.4 排序算法

首先按常理，排序成绩一般为倒序，因此以下排序规则按倒序处理。在线性表中分别声明希尔排序·快速排序·堆排序函数及其调用函数，另外根据根据要排序的科目设置排序标准选择器函数，根据需要将对应科目数据存入新数组，排序函数直接利用该新数组信息排序即可，以下详细解释这三者排序的解题思路：

2.4.1 希尔排序

希尔排序本质上是改造后的交换排序，只不过是从连续交换变为间隔交换，我们声明这个间隔的初始值是表长度的一半，之后进行循环，每循环一次间隔减半直到间隔为 0；在循环中内嵌新循环，新循环从间隔值加一位置的值开始，将其暂存，比较同一子序列的前一个记录，若大于，则交换，直到向前推到序列值小于 0；这样可以提前将较大值交换到前方，减少交换次数。但这里要注意，因为最终结果要打印的是排序后的整个学生信息，因此除了交换存入排序标准成绩的新数组，还要交换存放所有数据的源数组。

2.4.2 快速排序

快速排序本质上是改造后的冒泡排序，关键点在于对序列进行划分，减少交换次数。这里需要构造划分函数，该函数输入表的头和尾，分两部进行，第一部分为右侧扫描，比较尾指针指向值与头指针指向值的大

小，若头大于等于尾，无需交换，尾指针左移，否则交换，头指针向右移动，第二部分为左侧扫描，比较头指针指向值与尾指针指向值的大小，若头大于等于尾，无需交换，头指针右移，否则交换，尾指针向左移动头指针向右移动，这样扫描到头尾指针值相等，输出这个值 i，i 左侧的任一值大于等于右侧任一值。以该值 i 为划分值，分别用表头于 i-1 以及 i+1 和表尾为参数递归调用快速排序函数直到输入参数相等，最后实现排序。

2.4.3 堆排序

堆排序本质上是改造后的选择排序，需要利用堆结构，这里使用顺序表进行堆结构存储。这里涉及到无序序列建堆，移走堆顶，重建堆三个问题。重建堆可以通过构造堆调整函数进行，堆调整函数的设计用途是针对于一个已知堆遇到新值时的重建堆问题，首先是输入新值的序列号 i，有顺序表存储平衡二叉树的关系可知其左孩子序列号为 2i，当这个左孩子在顺序表存储范围内时，进行循环操作，获得左右孩子较小值，比较其与输入值，若输入值较小，则说明已经成堆，不用接着进行该函数了，否则需要将较小值交换上来，并在循环内继续比较被交换过后的新值与其新的孩子直到成堆或其本身就是叶子结点了。

从无序序列建堆可以利用该函数进行，令表长度的一半的值为 i，i 指向的值 a 按平衡二叉树来看是最后一个值的两亲节点值，将该值 a 输入到堆调整函数，就可以建立这个小堆，通过 i 进行循环，就依次建立了平衡二叉

树最后两层的分支堆-最后三层的分支堆-…-最后 n 层的堆，当 i=1 时建立起堆，最后 i=0，结束循环。最后构建小根堆。

具体排序过程就是通过循环，将堆顶的值与末尾值交换，这样最小值来到了末尾，将末尾值排除重建堆，重复该过程，整个序列就变为了倒序。

2.5 AOV 网

AOV 网我们决定使用邻接表构建，构造函数建立顶点表与边表，邻接表增添一个入度量来表示其前驱数目。将相关科目的名称和边的关系量化后导入邻接表即可。另外我构造了查询必要科目的函数，可以通过输入科目名称来获取修完这门课下一步才能修什么课的信息。

对于拓扑排序，构造函数的主要思路是先扫描顶点表，将入度为 0（即没有前驱）的结点存入栈，当栈不为空，释放该结点，链接其边表，将其边表指向结点的入度减一，当其入度归 0，则加入栈，循环操作输出序列，若 累加器累加的值到达边表存储值，则表示有拓扑回路，反之则没有。

2.6 用户交互

对于用户交互问题，我写了一个菜单函数 badboyschoice(list z)来执行。输入的是线性表（学生信息）对象。通过 switch 函数实现选择，根据提示进行不同的操作。涉及到线性表的选项直接进行操作。涉及到二叉树和邻接表的选项先声明该类的动态对象，调用数据进行操作，最后删除该对象。与此同时我们也通过 while 函数让用户可以进行不停的选择。

3 算法设计

3.1 线性表（顺序表）

✧ **ADT**

※数据结构

数据对象：学生信息结构体，表长度 ，暂时存放用作比较标准的成绩数据

数据关系：数据元素相邻，存储位置反映逻辑关系

※基本操作集

- list
  - 前置条件：表不存在
  - 输入：学生数目
  - 功能：表的初始化
  - 输出： 无
  - 后置条件：建一个顺序表
- ~list
  - 前置条件：表不存在
  - 输入：无
  - 功能：删除表

- 输出： 无
- 后置条件：表不存在

- Insert
  - 前置条件：表存在
  - 输入：无
  - 功能：在表的某处插入一个结构体数据
  - 输出： 无
  - 后置条件：表的长度加一，且新增一组数据
- Delete
  - 前置条件：表存在
  - 输入：删除结构体的姓名字符串
  - 功能：删除表中一项数据元素
  - 输出： 无
  - 后置条件：表的长度减一，减少一组数据
- Find
  - 前置条件：表存在
  - 输入：要找寻数据结构体中的学号数据（长整型）

- 功能：获取一个结构体内部某些数据的信息
- 输出： 无
- 后置条件：表依然不变

■ Change

- 前置条件：表存在
- 输入：要改变的结构体中的学号数据（长整型）
- 功能：改变数据元素中结构体的信息
- 输出： 无
- 后置条件：表中的某些数据发生改变

■ shiersort

- 前置条件：表存在，暂时存放用作比较标准的成绩数据有值
- 输入：无
- 功能：对表进行希尔排序
- 输出： 无
- 后置条件：表的顺序改变

■ kaisokusort

- 前置条件：表存在，暂时存放用作比较标准的成绩数据有值

- 输入：表头和表尾序号
- 功能：对表进行快速排序
- 输出： 无
- 后置条件：表的顺序改变

■ heapsort

- 前置条件：表存在，暂时存放用作比较标准的成绩数据有值
- 输入：无
- 功能：对表进行堆排序
- 输出： 无
- 后置条件：表的顺序改变

■ partition

- 前置条件：表存在，暂时存放用作比较标准的成绩数据有值
- 输入：需要划分的表头·表尾位置
- 功能：快速排序时进行一次划分
- 输出：划分轴位置
- 后置条件：表不变

■ sift

- 前置条件：表存在，暂时存放用作比较标准的成绩数据有值
- 输入：新加入值的序号，最后一个值序号
- 功能：堆排序时进行堆调整
- 输出：无
- 后置条件：表的顺序发生改变

✧ **存储结构**：顺序存储，数据元素相邻，占据提前分配好的内存

3.2 二叉排序树（二叉链表）

✧ **ADT**

※数据结构

数据对象：二叉树结点（结点中存放数值）

数据关系：一个数据元素包含数据域与指针域，指针域分为左孩指针·右孩指针，用指针来反映数据元素之间的逻辑关系。左孩指针指向左孩子结点，右孩指针指向右孩子结点。

※基本操作集

■ bisorttree

- 前置条件：二叉链表不存在
- 输入：线性表对象地址
- 功能：建立二叉链表

- 输出： 无
- 后置条件：建一个二叉链表

- searchbst
  - 前置条件：二叉链表存在
  - 输入：查找值,线性表对象地址
  - 功能：获得查找值的相关信息
  - 输出： 无
  - 后置条件：二叉链表不变
- insertbst
  - 前置条件：无
  - 输入：插入结点指针地址，插入结点值，插入结点序号
  - 功能：插入一个结点
  - 输出：无
  - 后置条件：二叉链表新增一结点或生成有一个结点的二叉链表
- forsearchbst
  - 前置条件：二叉链表存在
  - 输入：需要查找结点指针，查找值，线性表对象地址

- 功能：查找函数调用的核心函数
- 输出： 无
- 后置条件：二叉链表不变

✧ **存储结构**：链式存储，用一组任意的存储单元存放二叉链表的元素，有需要发生时申请内存（动态内存）

3.3 平衡二叉树（二叉链表）

✧ **ADT**

※数据结构（与 3.2 二叉排序树一致，此处不表）

※基本操作集

■ bibantree

- 前置条件：二叉链表不存在
- 输入：线性表对象地址
- 功能：建立二叉链表
- 输出： 无
- 后置条件：建一个二叉链表

■ searchbbt

- 前置条件：二叉链表存在
- 输入：查找值,线性表对象地址
- 功能：获得查找值的相关信息
- 输出： 无
- 后置条件：二叉链表不变

■ leverorder

- 前置条件：二叉链表存在
- 输入：线性表对象地址
- 功能：获得二叉排序树的层序
- 输出： 无
- 后置条件：二叉链表不变

■ insertbbt

- 前置条件：无
- 输入：插入结点指针地址，插入结点的父母节点指针地址，

  插入结点值，插入结点序号
- 功能：插入一个结点
- 输出：该插入结点指针

- 后置条件：二叉链表新增一结点或生成有一个结点的二叉链表

- forsearchbbt
  - 前置条件：二叉链表存在
  - 输入：需要查找结点指针，查找值，线性表对象地址
  - 功能：查找函数调用的核心函数
  - 输出： 无
  - 后置条件：二叉链表不变
- forsearchbbt2
  - 前置条件：二叉链表存在
  - 输入：需要查找结点指针，查找值，线性表对象地址
  - 功能：用另一种方式实现查找的核心功能
  - 输出： 无
  - 后置条件：二叉链表不变
- balance
  - 前置条件：存在二叉树
  - 输入：最新插入结点的指针
  - 功能：验证该二叉树是否平衡

- 输出：若平衡，输出 NULL，否则输出最小不平衡树顶点指针
- 后置条件：二叉树不变

■ depth

- 前置条件：存在二叉树
- 输入：该二叉树顶点结点指针
- 功能：获取二叉树深度，用于计算平衡因子
- 输出：深度值
- 后置条件：二叉树不变

■ ll/rr/lr/rl

- 前置条件：存在最小不平衡二叉树且为 LL/RR/LR/RL 型
- 输入：最小不平衡树顶点指针地址
- 功能：进行 LL/RR/LR/RL 变幻
- 输出：无
- 后置条件：二叉链表结构发生改变

✧ **存储结构（与 3.2 二叉排序树一致，此处不表）**

3.4 AOV 网（邻接表）

✧ **ADT**

※数据结构

数据对象：存放科目信息的顶点表，存放数据关系的边表，顶点个数，边数

数据关系：顶点表包含数据域与指针域，数据于存放科目信息，指针域反映与边表的逻辑关系。顶点表按顺序表示逻辑关系。边表包含数据域和指针域，数据域表示其对应的顶点表项，指针域表示其根源顶点表项与指向边项的对应顶点表项的关系。

※基本操作集

- algraph
  - 前置条件：邻接表不存在
  - 输入：顶点表数据项 1，顶点表数据项 2，边表数据项，表个数，边个数
  - 功能：建立邻接表
  - 输出：无
  - 后置条件：邻接表存在
- ~algraph
  - 前置条件：邻接表存在
  - 输入：无
  - 功能：销毁邻接表

- 输出： 无
- 后置条件：邻接表不存在

■ search

- 前置条件：邻接表存在
- 输入：无
- 功能：查询顶点表某一项的后继
- 输出：无
- 后置条件：邻接表不变

■ tuopu

- 前置条件：邻接表存在
- 输入：无
- 功能：进行拓扑排序，验证是否存在回路
- 输出：无
- 后置条件：邻接表不变

✧ **存储结构**：顶点表为顺序存储，边表为链式存储，顶点表通过指针域与边表关联，某个顶点表项的边表的各项与这个顶点表项存在边关系。

4 程序运行平台

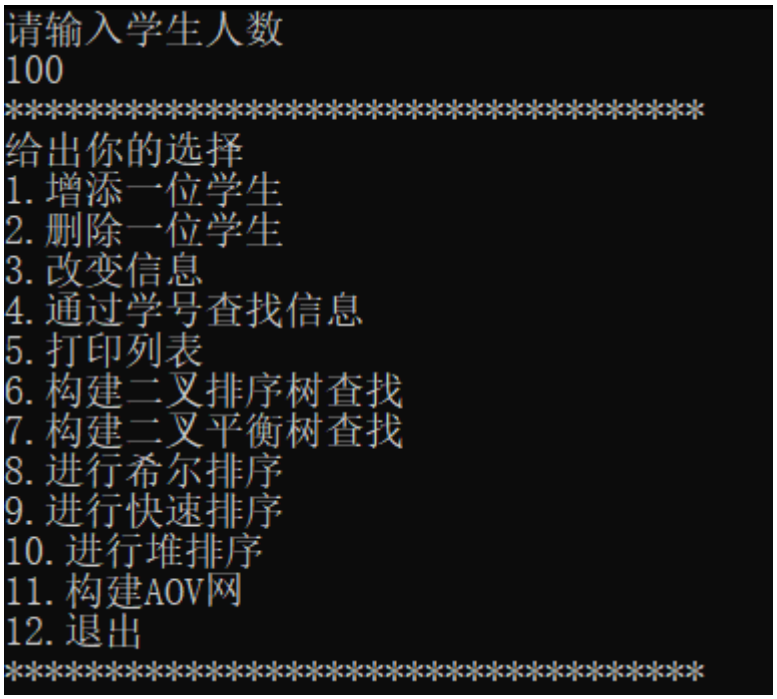
操作系统：WINDOWS 10

编译器：　GNU GCC Compiler

使用软件：CodeBlocks 17.12

5 程序运行结果

5.1 用户交互界面



以输入 100 人为例

5.2 线性表操作

```
**********
4
请输入学号
2018000041
姓名:李珊蔚 高等数学成绩:94 计算机导论成绩:71 离散数学成绩:46 程序设计成绩:45 数据结构成绩:35 计算机原理成绩:35 数据库原理成绩:30 总成绩356
**********
```

```
**********
1
请选择插入位置
50
请输入学号:
20202022
请输入姓名:
张三
请输入高等数学成绩:
100
请输入计算机导论成绩:
20
请输入离散数学成绩:
20
请输入程序设计成绩:
20
请输入数据结构成绩:
15
请输入计算机原理成绩:
20
请输入数据库原理成绩:
98
**********
```

```
**********
3
请输入学号
2018000043
请按下以下选项前的数字修改对应信息
1.学号 2.姓名 3. 高等数学成绩 4.计算机导论成绩 5.离散数学成绩
6.程序设计成绩 7.数据结构成绩 8.计算机原理成绩 9.数据库原理成绩 0.退出修改
6
输入新成绩
15
请按下以下选项前的数字修改对应信息
1.学号 2.姓名 3. 高等数学成绩 4.计算机导论成绩 5.离散数学成绩
6.程序设计成绩 7.数据结构成绩 8.计算机原理成绩 9.数据库原理成绩 0.退出修改
0
**********
```

查找/插入/修改

学号	姓名	成绩:	高等数学	计算机导论	离散数学	程序设计	数据结构	计算机原理	数据库原理	总成绩
2018000089	王羽绍		78	86	7	94	65	80	32	442
2018000003	郑诗梅		65	72	61	58	84	8	72	420
2018000090	王璇彪		49	49	59	71	52	34	22	336
2018000096	何臣		33	80	39	74	9	28	97	360
2018000088	徐君		25	4	66	79	81	98	21	374
2018000072	李臣睿		4	59	100	34	1	51	80	329
2018000080	李健		39	38	97	51	34	35	19	313
2018000038	张娟亦		9	90	31	82	11	51	84	358
2018000083	田景睿		42	100	88	53	80	57	62	482
2018000022	刘同		63	92	46	4	61	31	98	395
2018000017	张爱会		20	68	41	48	79	97	98	451
2018000040	隋红润		3	63	100	87	87	41	79	460
2018000025	刘滨伯		1	21	72	24	9	75	51	253
2018000000	苏荣世		0	52	30	96	93	32	89	392
2018000023	司阳小		71	79	40	10	64	80	30	374
2018000053	孙山		98	42	8	32	57	27	22	286
2018000019	冯石进		52	74	43	8	2	65	82	326
2018000063	张鸿诚		43	22	95	16	48	25	6	255
2018000030	赵贵声		3	85	43	69	93	4	61	358
2018000002	姚廷		84	20	15	34	22	35	26	236
2018000035	邹基		19	79	19	45	8	13	51	234
2018000086	宋妮群		18	47	82	3	16	80	0	246
2018000077	宋民来		5	26	65	70	21	92	66	345
2018000092	柴怀传		46	46	21	32	80	35	86	346
2018000098	姚芳良		42	71	14	77	55	3	1	263
2018000060	边朝妮		82	41	65	12	5	77	3	285
2018000021	邹子		40	81	48	63	63	25	45	365
2018000032	祁荣翰		26	96	18	99	45	56	31	371
2018000046	凌林胜		80	1	7	81	18	1	90	278
2018000045	涂来风		69	44	18	31	60	16	93	331
2018000028	萧松启		97	98	51	46	42	22	47	403
2018000013	吴珠善		52	55	59	25	100	28	5	324
2018000014	潘敬青		41	97	61	32	20	0	2	253
2018000042	袁旻		77	35	22	98	78	92	68	470
2018000061	王元柏		28	16	5	9	21	13	26	118
2018000064	董瑞		10	42	4	13	80	34	42	225
2018000006	田贤		70	15	32	8	83	10	23	241
2018000027	李静忠		7	21	10	52	14	82	28	214
2018000033	杜龙恒		59	4	17	73	53	85	31	322
2018000058	谷剑道		12	72	38	34	14	22	53	245
2018000099	何萍铁		3	52	79	41	36	81	25	317
2018000069	莫勇柏		95	44	7	96	77	90	48	457
2018000093	郑明胜		93	95	38	71	4	83	79	463
2018000056	罗贤坚		76	81	34	66	1	13	58	329
2018000062	翟昊		24	17	6	65	13	13	76	214
2018000012	张秋		36	12	60	37	42	53	87	327
2018000005	曹超同		25	47	41	33	71	69	94	380

打印列表（节选）（以 100 人为例）

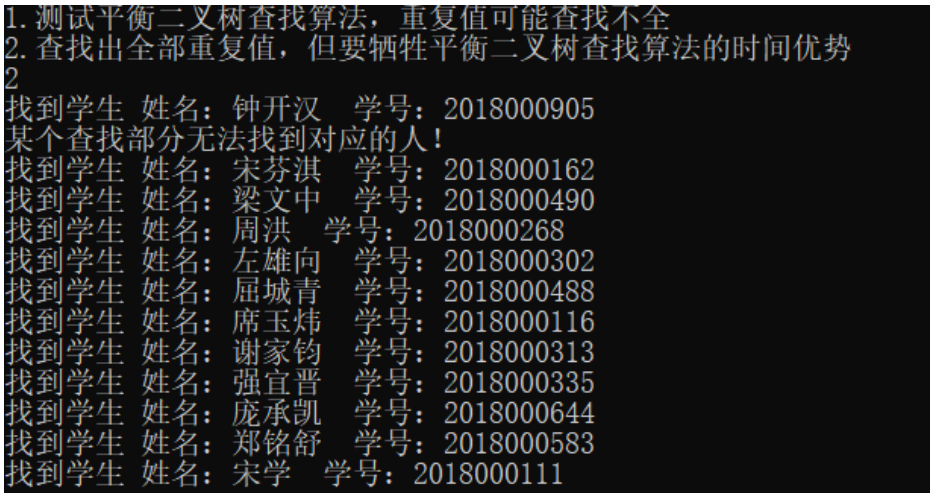
5.3 构造二叉树查找

```
**************************************
6
请按下数字选择您通过二叉排序树要排序的科目
1. 高等数学成绩  2. 计算机导论成绩  3. 离散数学成绩
4. 程序设计成绩  5. 数据结构成绩  6. 计算机原理成绩
7. 数据库原理成绩  !其它数字按键视为排序总成绩
1
输入您要查找的分数
50
找到学生  姓名：秦麟艳    学号：2018000050
找到学生  姓名：赵雪    学号：2018000974
找到学生  姓名：徐婷年    学号：2018000651
找到学生  姓名：左健    学号：2018000310
找到学生  姓名：谢道仁    学号：2018000316
找到学生  姓名：罗阳龙    学号：2018000434
**************************************
```

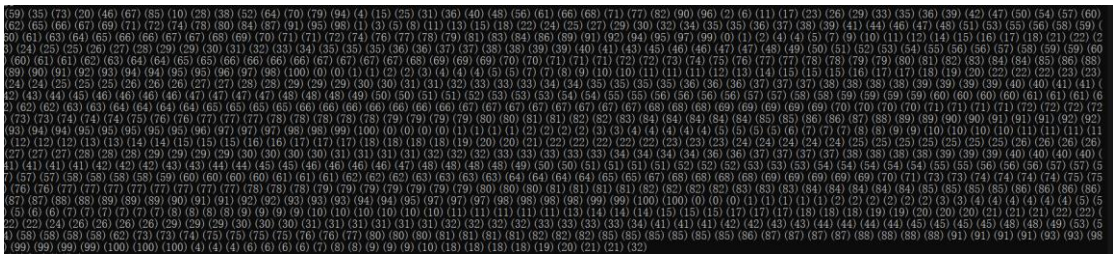
构造二叉排序树查找（以 1000 人为例，查找高等数学成绩为 50 的人）

```
**************************************
7
请按下数字选择您通过平衡二叉树要排序的科目
1. 高等数学成绩  2. 计算机导论成绩  3. 离散数学成绩
4. 程序设计成绩  5. 数据结构成绩  6. 计算机原理成绩
7. 数据库原理成绩  !其它数字按键视为排序总成绩
1
输入您要查找的分数
60
```

构造平衡二叉树查找（以 1000 人为例，查找高等数学成绩 60 的人）



平衡二叉树查找结果



平衡二叉树层序遍历结果（1000 人，按高等数学成绩）

5.4 排序操作

以下为三种排序方法获得排序结果列表（500000 人，按总成绩倒序，节选）

学号	姓名	成绩:	高等数学	计算机导论	离散数学	程序设计	数据结构	计算机原理	数据库原理	总成绩
2018040326	邓升钰		89	98	94	97	97	70	100	645
2018491362	贾岳焕		99	98	90	95	93	93	74	642
2018180198	沈培森		83	98	85	99	94	89	92	640
2018197220	唐秋辰		97	84	81	92	96	92	94	636
2018047580	罗冰妮		72	99	94	88	98	100	85	636
2018171025	韦安		99	79	98	89	95	83	90	633
2018177521	吴芳天		97	64	100	90	97	90	93	631
2018449775	柴信凌		73	96	91	99	99	89	80	627
2018106153	王洁承		98	96	98	83	96	70	84	625
2018110018	郑鹤碧		94	96	85	66	92	94	98	625
2018034301	孔丽邦		88	100	94	98	71	92	82	625
2018215447	游艳昕		97	94	96	79	89	80	89	624
2018185902	郭臣		89	95	99	96	82	99	64	624
2018381645	华行雄		92	78	94	95	100	70	95	624
2018096393	魏玉思		65	96	97	91	98	85	91	623
2018082079	梁佩芝		98	92	63	92	99	80	99	623
2018405721	郑扬		67	98	91	83	87	98	99	623
2018307258	耿智仪		92	92	98	82	99	85	74	622
2018116953	范乃田		89	96	83	99	92	80	83	622
2018047130	江霖		87	67	95	84	93	100	96	622
2018411051	杨桐伟		98	91	97	88	78	93	77	622
2018413016	韩逸		84	88	97	61	96	96	100	622
2018047579	韦福依		99	80	87	98	91	74	92	621
2018356223	张铁		97	82	95	94	73	93	86	620
2018463332	万保友		95	89	86	73	81	98	97	619
2018041015	华滨宜		94	96	89	97	67	93	83	619
2018343403	徐礼		95	91	82	83	79	93	96	619
2018371797	任鲁诗		80	95	96	73	91	99	85	619
2018096038	施伯		84	98	88	99	87	76	87	619
2018116062	阳一雨		88	79	83	100	87	91	90	618
2018459220	杨恩凡		99	94	90	76	72	93	94	618
2018493257	黄铭		96	93	100	94	97	60	77	617
2018029129	涂灵正		97	83	84	90	83	96	83	616
2018337709	周根雯		86	87	99	100	74	71	98	615
2018097082	汪威旭		73	71	90	91	98	94	98	615
2018063253	李岚如		97	86	82	92	96	68	94	615
2018471460	徐锋勇		83	75	97	83	95	90	92	615
2018221518	薛荣有		99	77	98	88	90	83	80	615
2018350146	徐宜鹤		96	79	97	96	82	66	99	615
2018192504	徐标秀		94	80	92	54	100	100	94	614
2018079094	舒柏清		70	79	78	97	99	94	97	614
2018116437	谢亚波		96	78	93	86	86	93	81	613
2018262358	胡晓传		85	94	94	76	100	68	96	613
2018015647	涂炳博		99	100	91	37	90	100	96	613
2018380844	盛鹏满		97	88	98	96	98	38	97	612
2018103780	于庆睿		97	90	92	44	96	93	100	612
2018103259	华孝庆		50	88	100	91	92	99	92	612
2018057599	戴西辰		85	69	95	99	89	99	76	612

5.5 拓扑排序

```
****************************************
11
拓扑序列为: 计算机导论 高等数学 离散数学 程序设计 数据结构 计算机原理 数据库原理
拓扑验证成功！存在回路

要查询必要科目,请输入0,输入其它数字结束该进程(请不要输入数字以外的指示)
0
请按汉字输入课程名称
高等数学
修完该课程下一步才能修读以下课程:
程序设计
离散数学
****************************************
```

6 讨论与结论

6.1 算法的时间复杂度（粗略）

6.1.1 **线性表：**

操作函数	时间复杂度
构造	O(n)
析构	O(1)
查找	O(n)
更改	O(n)
插入	O(n)
删除	O(n)
打印	O(n)
获取表长	O(1)
排序标准选择器	O(n)
希尔排序	$O(n\log_2 n)$~O(n^2)
快速排序	$O(n\log_2 n)$~O(n^2)
快速排序调用一次划分函数	O(n)
堆排序	$O(n\log_2 n)$
堆排序调用堆调整函数	$O(\log_2 n)$

6.1.2 二叉排序树

操作函数	时间复杂度
构造	$O(nlog_2n)$~O(n^2)
查找	$O(log_2n)$~O(n)

6.1.3 平衡二叉树

操作函数	时间复杂度
构造	$O(m+nlog_2n)$
查找	$O(log_2n)$
LL/RR/LR/RL 变幻	O(1)
验证是否平衡	$O((log_2n) ^2)$
层序遍历	O(n)

6.1.4 AOV 网

操作函数	时间复杂度
构造	O(m+n)
析构	O(1)
查询必要科目	O(n*m)
拓扑排序	O(n+n+p*m)

6.2 算法的空间复杂度（粗略）

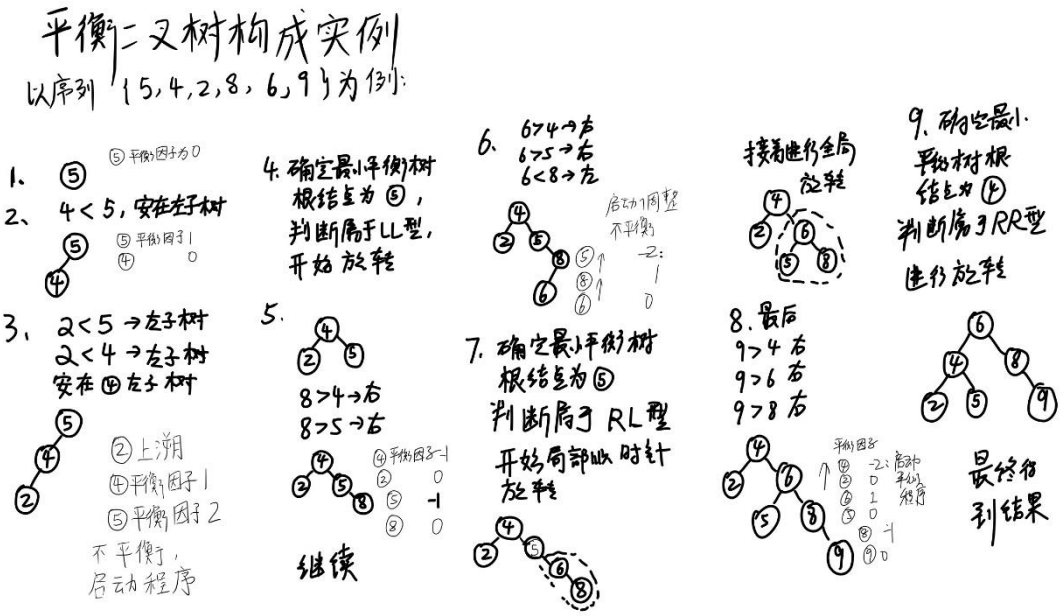
线性表：此处我们生成的时动态数组，因此数组长度根据我们定义的学生数目决定，可以合理利用空间，稍微将上限调高可以适应插入信息需要，空间利用率较高。对于排序的需求，需要额外构造数组存储用来排序的数据，对于三种排序算法，按理来说堆排序需要重新构造空间，但我们这里直接用已有线性表对应堆，合理节省空间。

二叉树：排序二叉树相对平衡二叉树而言，因为排序二叉树没有经过调整，所以其形态相对平衡二叉树可能更加倾向一方。虽然虽然结点数和总体空间利用率都是差不多的，但就单个结点的储存而言，平衡二叉树要么空间利用率偏高，要么偏低，而二叉排序树的单个结点空间利用率更均一。

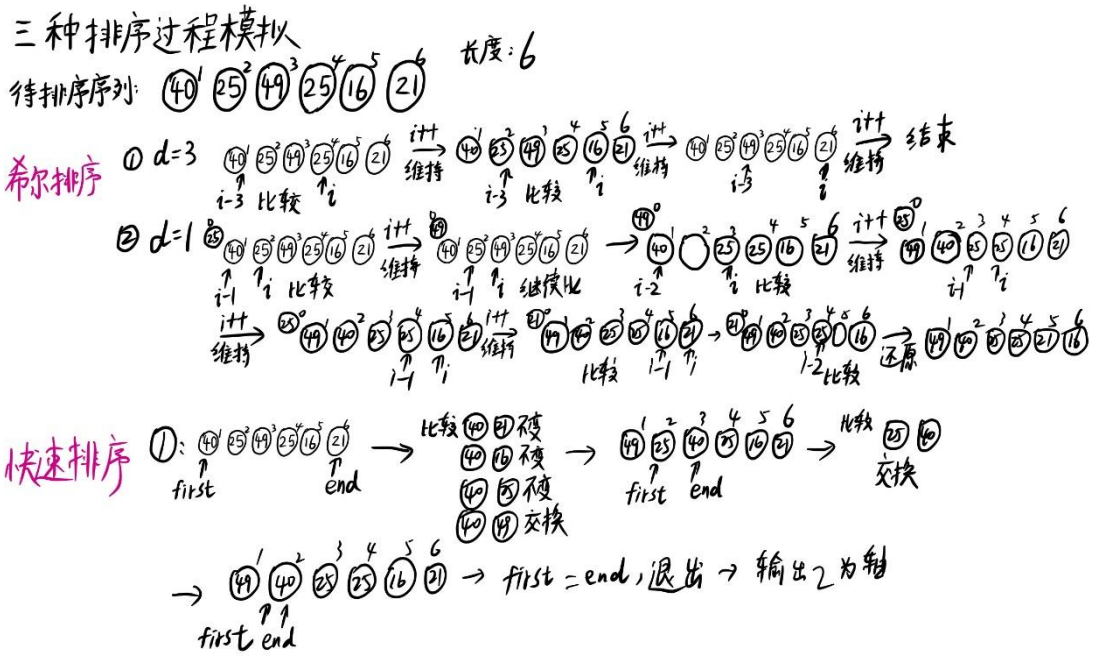
邻接表：顶点表包含数据域和指针域，丧失了顺序表的高结点数据密度，但可以存储边关系，边表使用数据域和指针域实现边的表示，总体结构来说虽然会有冗余，但很好地存储了有效信息，相比邻接矩阵对空间复杂度过分的要求，邻接表算是比较高效。

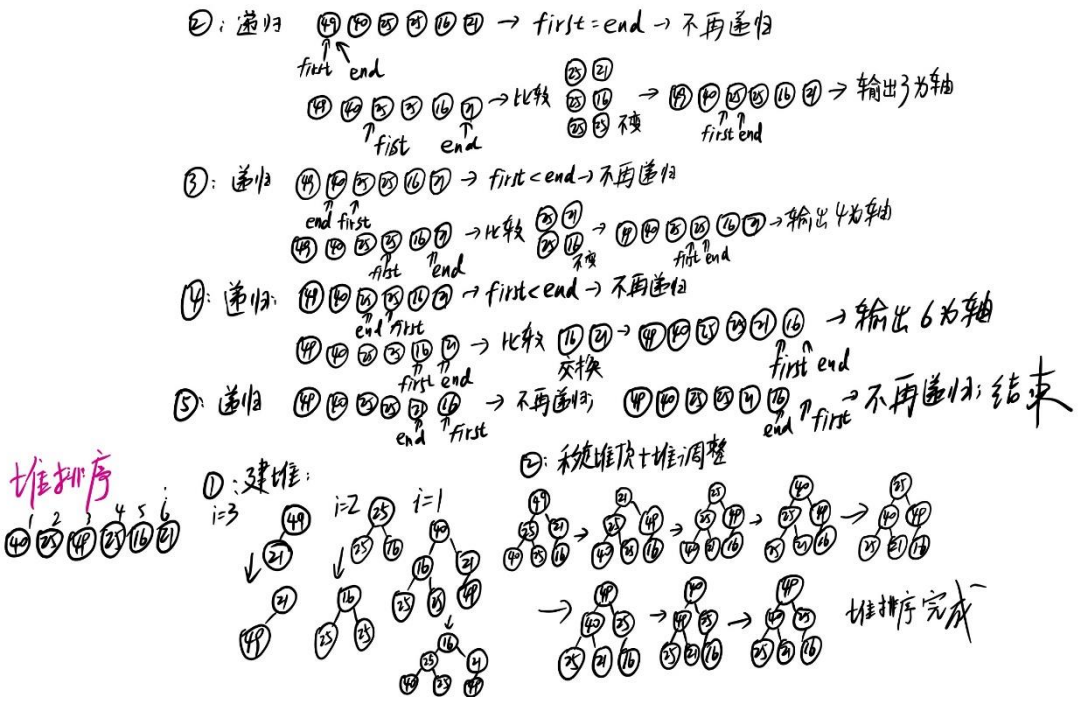
6.3 算法实现过程模拟

6.3.1 **平衡二叉树的构造过程模拟**

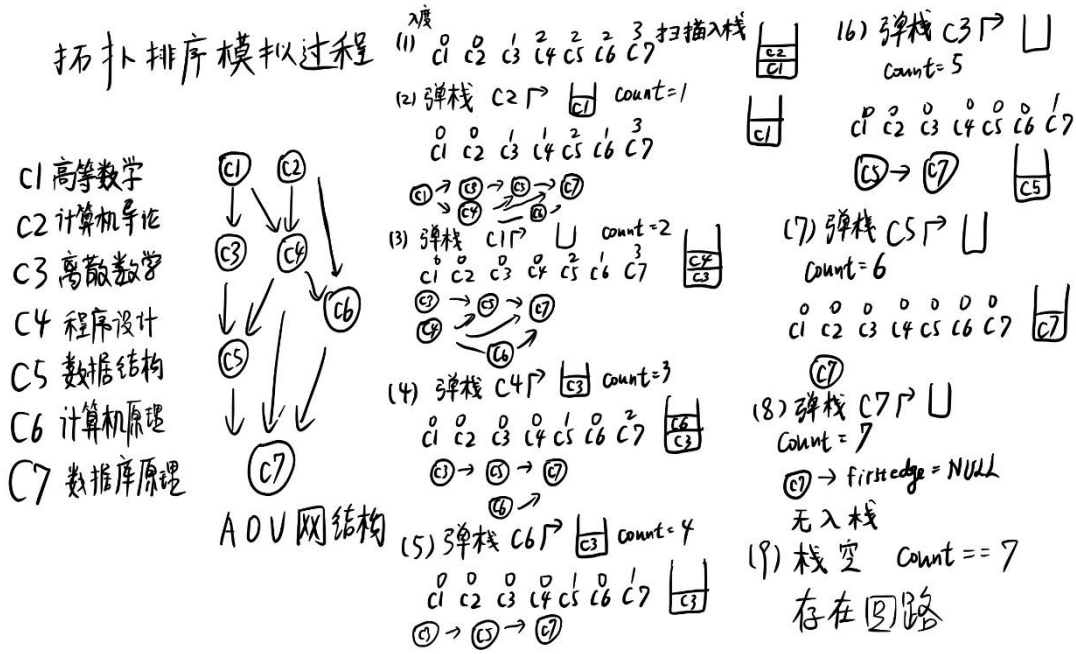


6.3.2 三种排序算法过程模拟




堆排序
①：建堆：
i=3
i=2
i=1
②：利用堆顶+堆调整
堆排序完成

6.3.3 拓扑排序模拟


拓扑排序模拟过程
C1 高等数学
C2 计算机导论
C3 离散数学
C4 程序设计
C5 数据结构
C6 计算机原理
C7 数据库原理
AOV网结构
(1) 扫描入栈
(2) 弹栈 C2
(3) 弹栈 C1
(4) 弹栈 C4
(5) 弹栈 C6
(6) 弹栈 C3
(7) 弹栈 C5
(8) 弹栈 C7
count=5
count=6
count=7
firstedge=NULL
无入栈
(9) 栈空 Count==7
存在回路

6.4 算法性能分析

6.4.1 二叉排序树和平衡二叉树的性能

我们利用随机生成的数据对二叉树查找算法性能进行测算，由于这类查找算法是构造和查找的共同作用结果，因此我们将构造和查找的时间分别统计，来比较两种算法的性能。时间计算利用 clock（）函数。

测试基准：（1）随机生成多组学生人数

（2）按照总成绩排序（理论上范围为 0-700）

（3）分别查找总分为 0，400，700 的学生并记录递归次数

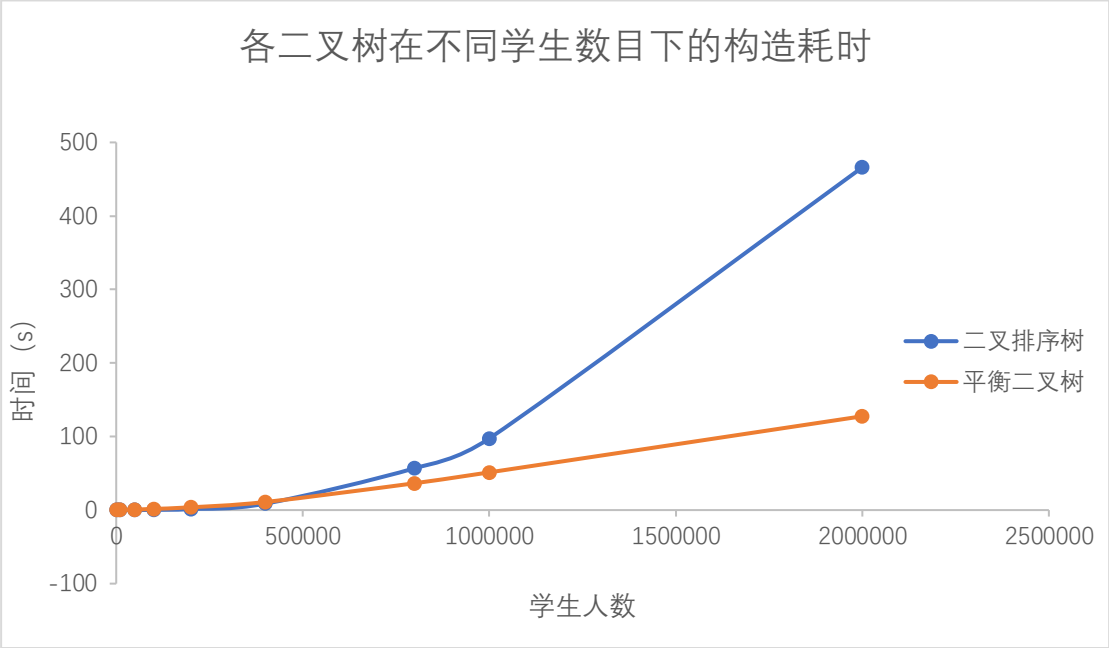
（经操作发现查找时间都接近 0，因此记录递归次数）

学生人数	构造 排序二叉树 时间（s）	构造 二叉平衡树 时间（s）	排序二叉树 查找递归次数			排序二叉树 查找递归次数		
			0	400	700	0	400	700
100	约为 0	约为 0	10	10	3	8	8	7
1000	约为 0	0.004	6	8	10	11	8	11
10000	0.003	0.098	14	15	16	14	7	14
50000	0.075	0.717	17	12	11	16	8	5
100000	0.290	1.638	18	10	14	18	6	16
200000	1.338	3.800	13	13	18	16	8	19
400000	8.748	11.002	18	7	18	19	4	18

800000	56.929	36.519	14	13	19	18	8	21
1000000	97.476	51.303	9	16	15	20	8	19
2000000	466.116	127.621	12	10	17	20	4	19

（人数大于 2000000 时构造过慢，因此就测试到 2000000 人为止）

观测结论：



（1）在测试范围内，排序二叉树的构造时间和平衡二叉树的构造时间都随人数增长而增长，但二叉排序树起初耗时短增长慢，而后期耗时边长，增长变快，平衡二叉树相比二叉排序树最先耗时多，但随人数增加时间耗费渐渐小于二叉排序树。结合这两种二叉树的构造原理可以推测，在人数较少时，二叉排序树树趋于规整，某个结点左孩子与右孩子深度差较小，而二叉平衡树需要验证是否平衡和调整，因此耗时较二叉排序树多。但随人数增加，二叉排序树趋于不规整的程度越来越深，构造时更可能在某条偏激的路径上走下去，二叉平衡树

虽然需要验证是否平衡和调整，但所有路径长度差不多，因此构造时更不可能花费较长时间，所以后期二叉平衡树构造时间相对排序二叉树越来越少。

（2）因为最后都构成了二叉树，所以查找时间相比构造时间可以忽略，为了测试查找性能，我们记录了查找函数递归的次数来表示。分别使用可以找到的值 400，不太可能找到的值 0，700 来进行测试。通过结果可以发现，在测试 400 这个可以找到的值时，二叉平衡树的递归次数明显少于二叉排序树，可以从这一点来理解平衡二叉树的稳定性，在同一深度平衡二叉树能比二叉树容纳更多数据，因此在可以找到值时，递归次数相对较少，查找效率更高。而对于 0，700 这类找不到的值而言，在递归次数的数据方面呈现的是随机性，平衡二叉树相比更稳定。因此，在追求稳定性和高效率的方面，平衡二叉树的查找是更好的选择。

（3）综合构造和查找来分析这两种二叉树来看，如果以查找耗时为基准来比较，由于查找耗时远低于构造耗时，因此主要考虑构造耗时，这样学生数目较小时二叉排序树更优，学生数目较大时平衡二叉树更优。而以查找次数为基准来比较，则需要综合进行考虑，整体来说平衡二叉树的查找能力与效果优于二叉排序树，再根据学生数目选择二叉树时，要额外考虑平衡二叉树的查找优势再进行决定。

（4）从观测结果来看，人数小于 100000 时耗时较少，构建二叉树查找的性能优势可以发挥到极致，但人数一旦超过，构造二叉树时间成本越来越高，就不宜再选用此类方式进行查找。

6.4.2 三种排序算法的性能

我们利用随机生成的数据对希尔排序·快速排序·堆排序进行测。时间计算利用clock（）函数，按以下标准测试：

测试基准：（1）随机生成多组学生人数

（2）按照总成绩排序（理论上范围为 0-700）

（3）每一组测试三个结果，计算平均值

A 希尔排序

学生数目	算法花费时间（s）			
	第 1 组	第 2 组	第 3 组	平均值
1000	0.001	0.001	0.001	0.001
5000	0.003	0.003	0.003	0.003
10000	0.007	0.007	0.007	0.007
50000	0.044	0.046	0.046	0.045
100000	0.094	0.092	0.092	0.092
250000	0.280	0.268	0.268	0.272
500000	0.564	0.575	0.563	0.567
1000000	1.278	1.227	1.255	1.257
2500000	3.347	3.174	3.128	3.210
5000000	6.687	6.831	6.786	6.768

B 快速排序

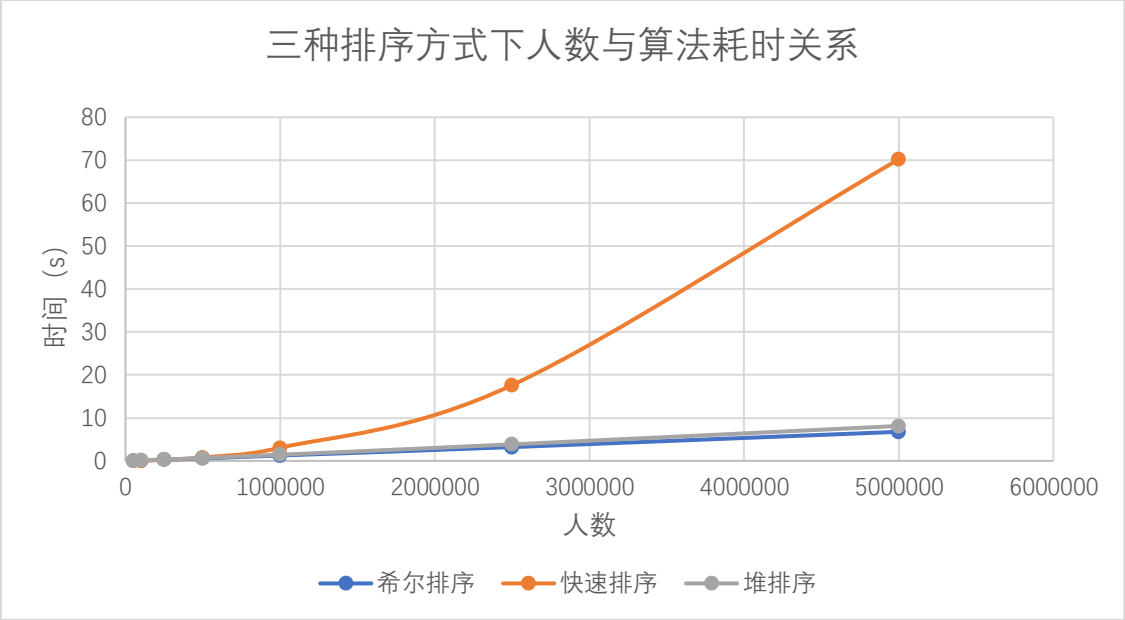
学生数目	算法花费时间（s）			
	第 1 组	第 2 组	第 3 组	平均值
1000	0.001	0.001	0.001	0.001
5000	0.002	0.002	0.002	0.002
10000	0.007	0.007	0.007	0.007
50000	0.020	0.020	0.021	0.020
100000	0.055	0.056	0.057	0.056
250000	0.238	0.246	0.251	0.245
500000	0.805	0.817	0.841	0.821
1000000	3.063	3.075	3.031	3.050
2500000	17.594	17.582	17.7311	17.636
5000000	70.358	70.178	70.082	70.206

C 堆排序

学生数目	算法花费时间（s）			
	第 1 组	第 2 组	第 3 组	平均值
1000	0.001	0.001	0.001	0.001
5000	0.003	0.003	0.003	0.004
10000	0.008	0.008	0.008	0.008

50000	0.052	0.049	0.048	0.049
100000	0.104	0.103	0.104	0.104
250000	0.328	0.302	0.301	0.310
500000	0.673	0.684	0.692	0.683
1000000	1.412	1.465	1.436	1.438
2500000	3.874	3.827	3.824	3.842
5000000	8.154	8.184	8.031	8.123

从结果后我们可以了解，超过 5000000 人时间耗费更多，小于 50000 时间过小，因此我们从 50000-5000000 这个区间来分析:



（1）通过对图像和数据的观察总结，我们发现堆排序和希尔排序都是随着人数增加，时间成线性且稍微上扬的趋势增加，符合 $O(n\log_2 n)$的时间复杂度规律，堆排序出现这种规律很正常，希尔排序应该得益于增值的合理设置。而快速排

序呈现出一种类似于 O(n^2)的演进规律，这是接近所谓的“最坏情况”（即每次划分只得到一个比上一次划分少一个记录的子序列（另一个子序列为空））的可能结果。

（2）图中可知经过一定测试，堆排序和和希尔排序更有可能排除随机因素对算法的干扰，实现算法性能最佳化。这三种改进排序算法都有极强可用性，在人数小于 50000 基本秒解决，而随时间增长希尔排序和堆排序（尤其是堆排序）算法对时间的要求增长也不大，可以实现对千万级数据的排序。

（3）按理说堆排序非常稳定，时间耗费应该最少，但数据显示希尔排序时间略少于堆排序，初步分析是因为堆排序在开始需要从无序序列生成堆，这就耗费一定时间，使得在时间上稍微逊色于增值非常合理的希尔排序。

（4）以上生成数据是根据时间的种子进行的，不能做到完全随机，因此会影响各种排序的关键要素，每次测试也无法获得全部情况，因此 这是得到了各个算法的运行时间参考值，获得的仅仅是参考性结果。

6.5 讨论·总结·反思

6.5.1 **讨论**

（1）关于平衡二叉树的查找问题：排序二叉树在构造时小于结点值置于左孩，大于等于置于右孩，因此相同值都挂在右子树上，因此在排序二叉树查找时找到值只有向右下溯就可以找到所有相同值。但平衡二叉树与排序二叉树略有不

同，在构造时需要判断是否平衡并进行调整，这样相同值就可能挂在左子树。如果直接套用排序二叉树查找函数的话，就可能找不到所有相同值。因此需要对该函数进行一定改进，改造思路是找到值后对其左子树和右子树分别递归查找，这样可以保证找到所有值，并减少时间收到的影响。但这样终归无法体现平衡二叉树的真正优势，因此我也保留了第一种方法，以在测试平衡二叉树算法查找性能时使用。

（2）指针的形式参数问题：这次编程遇见的大问题就是在二叉树构造时由于调用了插入结点函数，传入指针后发现改变无效。后来才意识到指针也存在形式参数的问题，解决方法就是指针的指针，通过改变指针的地址实现在函数里对指针的改变，这是很有启发的一点。

（3）关于排序算法的讨论：希尔排序・快速排序・堆排序有一个共同点就是实行额外处理让关键码基本有序，减少不必要的交换。希尔排序建立在快速排序基础之上，通过引入增量构建子序列处理，实现基本有序，而且巧妙之处在于开始排序时增量大，子序列数目少，排序快，后来增量小，但已经基本有序，因此速度也比较快，让时间复杂度从 $O(n^2)$向 $O(nlog_2n)$减小；快速排序脱胎于冒泡排序，通过引入轴值和分割算法将较大值换到前面，较小值换到后面，递归处理，使得排序从每个值与每个值比较变为部分值与部分值比较，让时间复杂度从 $O(n^2)$向 $O(nlog_2n)$减小；堆排序改进自选择排序，改进点在某次比较后不仅获得最小记录，也获得较小记录，具体实现方式是通过堆来进行，时间复杂度降为 $O(nlog_2n)$，比前两种方式都更节省和稳定。但是需要借助堆结构，存在空间复杂度的考虑，这里可以直接使用原结构存储堆加以优化。

6.5.2 总结

本次编程大作业本质上就是各种数据结构算法设计的大杂烩。通过这次练习我获得了很多有启发性的经验。

首先是不同数据结构的组合，这次的数据结构类涉及线性表·二叉树·邻接表，在编写程序的同时通通可以作为类，并通过友元来实现数据共享。以后遇到此类程序编写问题时，可以提前厘清需要作为类使用的数据结构，设置友元连接，确保逻辑的清晰可观。

第二是用户交互设计，秉持主函数简洁的原则，可以设置菜单函数进行，通过将对象传入函数进行各种操作。同时可以根据不同要求创建不同类的不同对象，来实现差异化信息处理与表示。

第三是动态结构设计，以往的练习中常常构建静态对象，导致操作陷于死板，往往无法随心所欲进行处理。通过构建动态数组，既可以按找用户需要即时构建对应结构，也可使结构因存在堆上导致容纳数据量够大，以支撑我们接着进行的庞大的数据测试与处理。此外，通过申请动态的类的对象，我们可以及时删除该对象，之后进行重新申请以满足循环测试的需要。

最后是查找与排序的算法的经验，在需要处理的数据量较小时，利用要求中的新算法其实没有太大必要。对于较大数据量，我么可以先利用二叉排序树/平衡二叉树/排序算法排好序，排序算法可以使用优化后的成果，不考虑成本的话最

好使用平衡二叉树，通过这些预备工作，大量的查找工作的效率也得到了十足的提升。

6.5.3 反思

这次作业的完成过程中，仍然有许多不足之处，主要在于:

（1）随机生成信息时生成成绩是 0-100 分等概率生成的，不符合现实规律，生成姓名的时候姓和名囊括不全且比例与真实有出入，但这些不足的调整空间其实也并不大。

（2）编写程序的时候因为有各种数据类型的明确要求，因此没有编写模板类，很多函数都是针对题设具体问题而设计，因此整个程序如果套用到别的问题上，可能兼容性不是很强。

（3）在生成大量信息测试算法的时候，因为设定的成绩只有那么多，势必产生大量重复成绩，这对平衡二叉树和排序二叉树查找性能的分析构成了一定的影响，产生了一些误差。

（4）在测算三种排序算法的时候，由于随机生成的数据问题，无法完全体现各种算法的最好情况和最坏情况，囿于机能和时间限制，不能将所有结果体现出来，因此测得算法性能会随着随机值的影响而改变，对于不同算法的性能的认识也就会产生一定的偏差。

（5）AOV 网回路的信息由于针对题设，直接输入到了菜单函数里，没有设计一种更加自动化的算法来达到相同的效果。

（6）很多功能的可视化部分设计不足，名字为两个字的学生的可视化列表中会产生一定的后移动情况。同时当成绩为 0 或者 100 时，在可视化列表中显示的是一位数或者三位数，与一般的两位数无法对齐，最终使得该行出现一些位移，使得界面显得较为难看。同时对于查找和排序也都只展示了结果，没能较好地表示作用过程。