上 海 大 学

2021-2022 冬季学期

《数据结构（1）》（08305009）

课程考核报告

学 号：20121034

姓 名：胡才郁

课程考核评分表

序号	内容	分值	成绩
1	第一题	40	
2	第二题	40	
3	课程总结	10	
4	报告规范性	10	
考核成绩			
评分人			

计算机工程与科学学院

2022 年 4 月

一、考核题目

（一）病毒株种类

[问题描述]

2019 年末一种从未出现过的新型病毒开始在全球迅速蔓延，并且不断变异，人类社会面临了极大的威胁。全球科学家开始共同抗疫。科学家通过研究发现，两种病毒之间可能存同类关系和相克关系。而且有以下二点是肯定的。第一，病毒 A 的同类病毒的同类病毒也一定是病毒 A 的同类；第二，病毒 A 的相克病毒的相克病毒也一定是病毒 A 的同类病毒。两种病毒是一个病毒株的当且仅当它们是同类。现在给你一些关于病毒关系的信息，问你至多有多少种不同病毒株。

[输入数据]

输入的第一行为 N(2<=N<=1000)，表示病毒的种类数（从 1 编号到 N)。

第二行 M(1<=M<=100000)，表示病毒关系信息的条数。

以下 M 行，每行可能是 S p q 或是 H p q，分别表示 p 和 q 是同类病毒，或是相克病毒。 假设输入不会产生矛盾。

[输出数据]

输出只有一行，表示最大可能的病毒株数。

[输入样例]

6

4

S 2 4

H 1 3

H 3 5

S 1 6

[输出样例]

3

[样例说明]

该样例最多有 3 个病毒株，分别是 1、5、6 号病毒为一个病毒株，2、4 号病毒为一个病毒株，3 号病毒为一个病毒株。

[测试数据要求]

输入数据在 virus.in 文件中；输出数据在 virus.out 文件中。

（二）核酸检测系统

1. 系统功能

2022 年 3 月奥密克戎变异株迅速席卷东方帝国的魔都，攻破了魔都苦心构筑多年的精准防疫铜墙铁壁。魔都人民奋起抗击，开展了全民覆盖的核酸检测。为了更迅速开展检测，寻找确诊者、密接者和次密接者，需要开发一套核酸检测系统。系统需要有以下 6 个功能：

(1)排队：输入人员代码，并选择混合测试还是单人测试。将人员代码加入到相应队伍排队。人员代码是一个 8 位（xxxyyyyz）的数字，其中 xxx 表示楼栋号（000~999）；yyyy 表示房间号，例如：0801、1801 等；z 代表一个房间中人员序号（1~9）。

(2)检测：选择混合测试还是单人测试。相应队伍中最前面的人员进行测试，混合测试每 10 人一个测试管；单人测试一人一管。每个测试管对应一个管号，管号是一个 5 位（kbbbb）的数字，其中 k 为 0 表示混合测试管，k 为 1 表示单人测试管；bbbb 是一个流水号，从 0000 开始自动生成。

(3)查看排队情况：按排队先后顺序，分别显示混合测试和单人测试排队人员的代码。

(4)登记测试管信息：输入测试管编号和测试结果。混合测试结果分为阴性、阳性和可疑三种。对阴性测试管对应的人员标记阴性状态；对阳性单人管测试人员标记确诊状态；对阳性混合管和可疑管人员标记可疑状态。并且对于确诊人员，其同一栋楼人员以及测试时排在他前面的 10 人和后面的人设置为密接者；密接者的同一栋楼人员为次密接者。

(5)各类人员查询：可以分类显示阴性、确诊、可疑、密接、次密接、待上传结果、在排队 7 种状态的人员代码。

(6)个人查询：输入人员代码，显示其当前状态。当前状态包括：阴性、确诊、可疑、密接、次密接、等待上传结果、在排队和未检测 8 种状态（未检测状态表示他没有参加排队检测）。

2. 初始化数据文件

为了方便测试，系统启动时可以从文件读入初始排队信息和检测信息。初始排队信息文件第一行二个正整数 n 和 m（1<n,m<10000），分别混合测试和单管测试的排队人数。接下来 n 行是混合测试人员的编号，在混合测试人员编号后面是 m 行单管测试人员信息。初始测试信息文件中有一行二个正整数 x(0<x<=n)和 y(0<y<=m)，分别表示已经完成混合测试和单人管测试的人数。

二、考核要求

题目一：

完成算法的设计和实现，准备好 5 组测试用例。在报告中介绍主要算法的思想和用到的主要数据结构；每一组测试用例需要有输入数据和输出数据的文件；并提供 C++源程序和可执行代码。

题目二：

根据系统功能描述，采用模块化程序设计方法进行程序设计，要求程序结构清晰。上述各个功能模块要求分别用函数实现，在主函数中通过调用这些函数，完成系统功能的要求。代码书写要规范，有简要的注释，给出函数说明。

设计报告内容包括总体设计、详细设计、源代码和测试情况。

总体设计：对程序的整体设计思路进行描述，画出系统的总体功能模块图，说明系统使用的主要数据结构，列表给出需要用到的函数并描述其功能。

详细设计：画出函数调用关系图，分析并描述函数的功能。

测试情况：记录程序编写和调试过程中遇到的各种问题，以及解决这些问题的途径和方法。并提交适当运行的截屏。

课程总结：

在课程报告最后写一个课程总结。回顾整个《数据结构(1)》课程的学习过程，对课程学习进行总结，给出个人体会。

《数据结构（1）》（08305009）课程考核报告

题目一：

一、算法思想

此题根据题干描述，对于某一个病毒而言，另一个病毒可以是它的同类病毒，或者是他的相克病毒。重点在于处理题目要求中，相克病毒的相克病毒一定为同类病毒，因此可知，每一个病毒最多只会有一类相克的病毒（若有多个个病毒与该病毒为相克关系，则根据题意，可知这多个病毒组合成了同一个病毒类），这是此题解题的关键。因此，此题在维护一个并查集，存储每一类同类病毒之外，只需要维护一个相克病毒的数组，数组之中记录对应节点的一个相克病毒即可（可以根据此节点寻找到此病毒类的根节点）。

因此，在具体实现时，对于同类病毒，直接使用 Union 操作合并，而对于相克病毒，找到此节点相克病毒的相克病毒，即为此节点的同类病毒，进行合并。算法流程图如下：

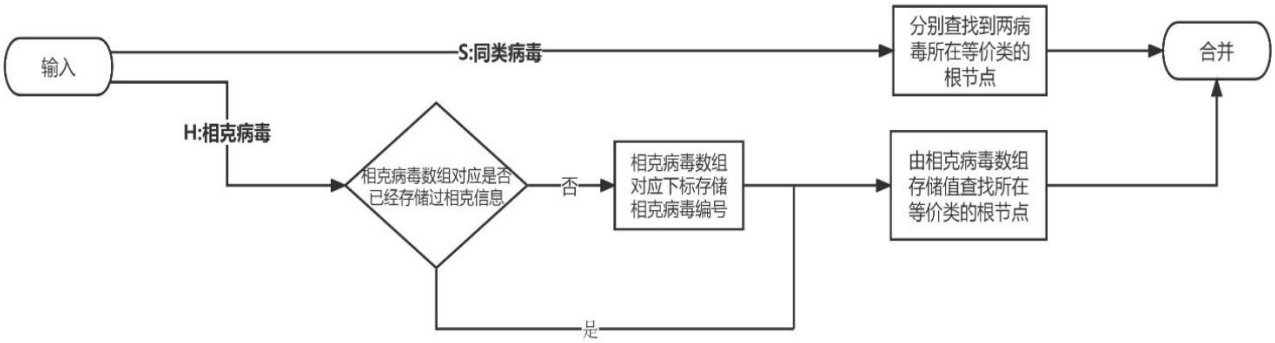

图 1. 算法流程图

二、主要数据结构

病毒存储在并查集之中，相克病毒的信息记录在数组之中。

1. 并查集

并查集是一种树形的数据结构，它用于处理一些无交集的合并及查询问题。它支持两种操作：

- 查找(Find):确定某个元素处于哪一个子集
- 合并(Union):将两个子集合合并成一个集合

1.1 路径压缩查找

本题目中，查找(Find)操作使用了带路径压缩功能的查找，一个同类病毒中，哪一个病毒作为这类病毒的根节点并不影响此类病毒与其它病毒的关系，这个病毒可以代表这一类病毒即可。将路径上的每一个节点都直接连在根上，即为路径压缩。

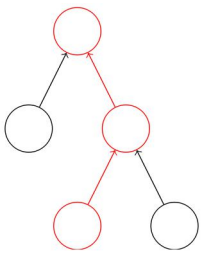

图 2. 查找示意图

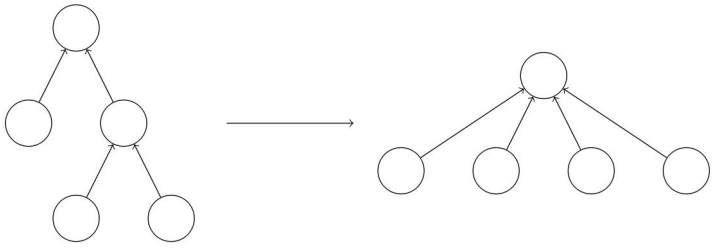

图 3. 路径压缩查找示意图

1.2 启发式合并(按秩合并)

对于并查集而言，支持操作只有集合的合并、查询操作，当我们需要将两个集合合二为一时，无论将哪一个集合连接到另一个集合的下面，都能得到正确的结果。但不同的连接方法存在时间复杂度的差异。具体来说，如果将一棵点数较小的集合树连接到一棵更大的集合树下，显然相比于另一种连接方案，接下来执行查找操作的用时更小（也会带来更优的最坏时间复杂度）。

1.3 数据成员与成员函数

在本题中 Node 之中的 data 域存储每一次输入所得的病毒编号值，查找操作使用带路径压缩功能的查找，合并操作采用启发式合并。经过路径压缩与启发式合并的算法优化之后，此算法的时间复杂度为$O(nlog(n))$，时间复杂度较低，能够较优的满足题目要求。

相克病毒的存储在数组之中，对于数组而言，数组下标为该病毒编号，此下标位置存储的值为该病毒的相克病毒编号。

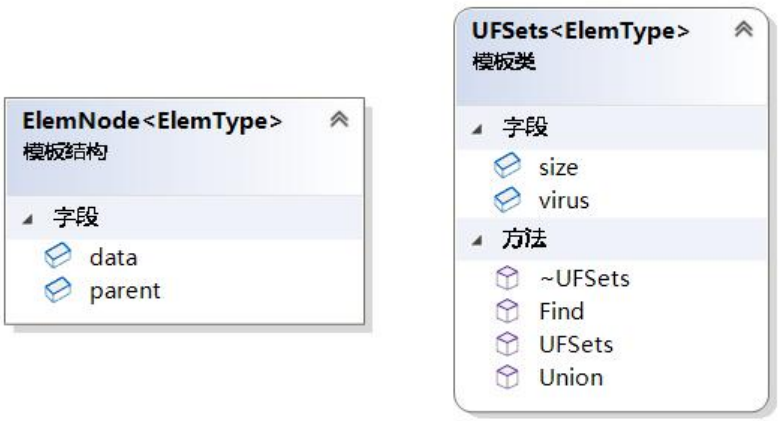

图 4. 并查集模板类与节点模板类 UML 图

表 1. UFSets 类模板中各函数的声明

函数原型	返回值类型	功能
UFSets();		空参构造函数，初始化并查集
~UFSets();		析构函数，清空并查集
int Find();	int	带有路径压缩功能，查找元素 e 所在树的根
void Union();	void	启发式合并(按秩合并)，合并两元素所在的集合

在主程序中，针对两种不同情况的核心代码如下所示：

```
for (int i = 0; i < M; i++) {
    infile >> c >> a >> b; // 读入每一组查询
    if (c == 'S') { // 情况 1：同类病毒,直接合并
            sets.Union(a, b);
    } else if (c == 'H') { // 情况 2：相克病毒，找到此节点相克病毒的相克病毒
        if (!restriction_virus[a]) // 若没有存储过此病毒的相克病毒编号
            restriction_virus[a] = b;
        if (!restriction_virus[b])
            restriction_virus[b] = a;
        sets.Union(b, restriction virus[a]);
```

```
        sets.Union(a, restriction virus[b]);
    }
}
```

图 4. 核心处理代码

三、测试数据

选取了 5 组数据进行实验。除了题目样例的数据之外，为了验证程序的稳定性，选取部分特殊情况，如下表所示。

表 3. 测试用例说明

序号	文件编号	测试用例说明
1	Virus.in	题目测试标准用例(6 个病毒,4 组查询)
2	Virus1.in	病毒组数多(1000 个病毒,3 条查询),
3	Virus2.in	类似于题目标准测试用例(10 个病毒,5 条查询)
4	Virus3.in	病毒组数较多，种类间关系复杂(100 个病毒，100 条查询)
5	Virus4.in	类似于题目标准测试用例(6 个病毒,4 条查询)

由于篇幅有限，报告中截取了部分测试用例，从左至右的截图分别为 virus.in\virus.detail\virus.out 中的内容，其中并且为了使得程序内部原理更加清晰，将并查集内节点编号、节点数据、节点根节点下标输出到 virus.detail 文件之中，清晰的展示。题目要求答案为最右边的数字。(完整的 5 组测试样例在代码目录下)

```
6
4
S 2 4
H 1 3
H 3 5
S 1 6
```

```
节点下标: 0    节点数据: 1    节点所在集合根节点下标: 4
节点下标: 1    节点数据: 2    节点所在集合根节点下标: -2
节点下标: 2    节点数据: 3    节点所在集合根节点下标: -1
节点下标: 3    节点数据: 4    节点所在集合根节点下标: 1
节点下标: 4    节点数据: 5    节点所在集合根节点下标: -3
节点下标: 5    节点数据: 6    节点所在集合根节点下标: 4
```

```
3
```

图 5. 测试用例 1

```
100
100
H 76 20
S 74 36
H 95 70
S 35 28
```

```
节点下标: 0    节点数据: 1    节点所在集合根节点下标: -1
节点下标: 1    节点数据: 2    节点所在集合根节点下标: -1
节点下标: 2    节点数据: 3    节点所在集合根节点下标: 89
节点下标: 3    节点数据: 4    节点所在集合根节点下标: 87
节点下标: 4    节点数据: 5    节点所在集合根节点下标: 70
节点下标: 5    节点数据: 6    节点所在集合根节点下标: -1
```

```
47
```

图 5. 测试用例 2

```
1000
3
S 1 2
H 1 3
H 1 5
```

```
节点下标: 0    节点数据: 1    节点所在集合根节点下标: -2
节点下标: 1    节点数据: 2    节点所在集合根节点下标: 0
节点下标: 2    节点数据: 3    节点所在集合根节点下标: 4
节点下标: 3    节点数据: 4    节点所在集合根节点下标: -1
节点下标: 4    节点数据: 5    节点所在集合根节点下标: -2
节点下标: 5    节点数据: 6    节点所在集合根节点下标: -1
```

```
998
```

图 5. 测试用例 3

题目二：

一、总体设计

对于此核酸检测系统而言，所有的操作都围绕着两种不同的形式展开，分别是单人测试与混合测试两种。由于单人测试与混合测试的排队队列不同、检测管号不同、检测时单位试管包含人数不同，因此，有必要将单人测试，混合测试在进行系统设计时区分开。两种模式的数据结构有细微的差别，对于每一个功能而言，分单人测试、混合测试两种情况分别处理。

因此，在本报告中，设计思路以及实现部分都将分单人检测与混合检测两种类型进行介绍。

对于参与核酸检测的人员而言，他的当前状态会随着流程的推移进行改变。因此，如果只设置 1 个人员状态的字段，在每次更改状态时都改变此人的状态，操作繁琐。本系统在设计时采取了状态存储，展示时状态覆盖的设计思路，有以下优点：

1. 逻辑删除，并非物理删除，保留了人员中间状态信息。例如某人当前状态为阴性，但过去状态为阳性，使用逻辑删除状态并不会使他过去的阳性状态消失，仍可以得知此人曾经确诊过。数据不会消失，“被删除”的数据也非常有价值。

2. 操作简便，逻辑简单，在某人得到状态结果时，无需考虑多种逻辑情况，在最终查询状态时进行人员状态输出即可。

1．功能模块设计

为了便于系统功能的 1 管理，本系统设计了 managementSystem 类，用于完成每一个功能函数，同时设置 Person 类，存储人员代码、楼号代码以及此人的 8 种状态。具体如下图所示：

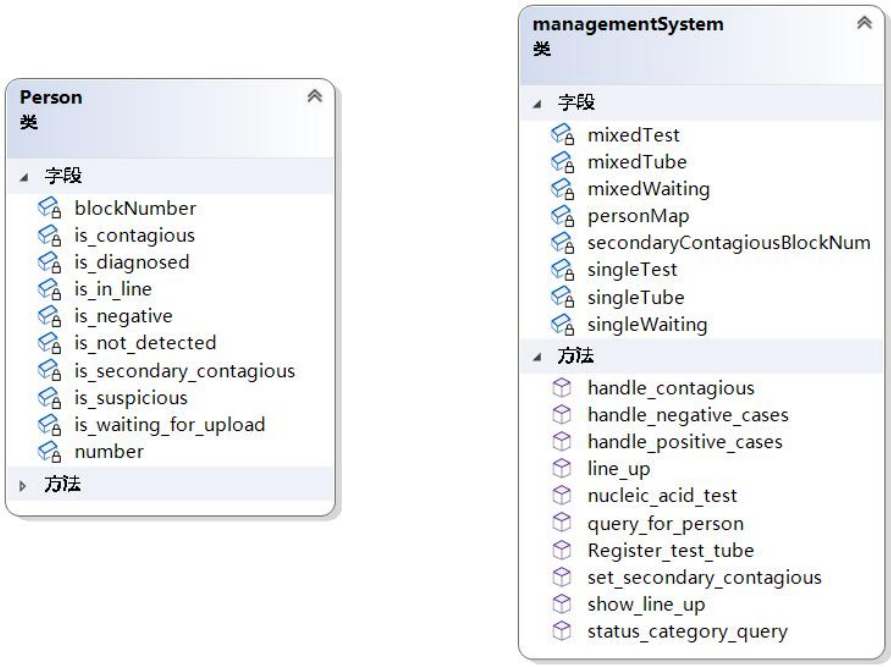

图 6. 核算检测系统类与人员类 UML 图

2．数据结构设计

数据结构针对排队、待上传、试管、次密接楼号、人员状态分别设计了对应的数据结构，并且同一种数据结构，针对单人检测与混合检测，分别实例化了一个 single 与 mixed 对象，分别用于存储在系统操作过程之中的数据。具体描述如下表：

表 6. 核酸检测系统类模板中数据结构的声明

序号	检测方式	函数原型	数据结构	描述
1	单人检测	LinkQueue<int> singleTest;	链队列	单人检测排队
2		vector<int> singleWaiting;	动态数组	单人检测待上传结果
3		unordered_map<int, int> singleTube;	哈希表	单人检测试管
4	混合检测	LinkQueue<int> mixedTest;	链队列	混合检测排队
5		vector<int> mixedWaiting;	动态数组	混合检测待上传结果
6		unordered_map<int, int *> mixedTube;	哈希表	混合检测试管
7	单人检测/混合检测	set<int> secondaryContagiousBlockNum;	集合	次密接楼号
8		unordered_map<int, Person> personMap;	哈希表	人员状态

2.1 排队：链队列

对于排队操作而言，常用到出队头，入队尾操作，因此使用队列这种数据结构。又由于排队人数不固定，根据题目要求，人员代码有 8 位数字（xxxyyyyz），则排队人数上下限浮动较大(1~99999999)。如果使用数组形式的队列，需开辟极大的空间来满足要求，但当排队人数较少时，则极易造成存储空间的浪费。因此，本系统的队列采用了链表进行实现。

manageSystem 类下两个成员变量 singleTest，mixedTest，分别存储单人检测队伍与混合检测队伍的人员代码。

2.2 核酸检测：动态数组

对于核酸检测操作而言，如果队伍之中出现了阳性，则需要将前 10 人以及此人之后的所有人都设置为阳性。此时，如果使用队列操作，通过遍历找到此确诊病例后，则难以将此人之前的 10 人全部进行标记。因此，当某人核酸检测完成之后，使他出排队队列，并将此人的人员代码存入动态数组，再次遍历寻找到此确诊病例之后，通过数组下标的偏移即可方便的寻找到排队时在他之前的 10 人。

manageSystem 类下两个成员变量 singleWaiting，mixedWaiting，分别存储单人检测待上传与混合检测待上传的人员代码。

2.3 登记测试管信息：哈希表

根据题意，每一个人员代码对应唯一的一个人，每一个人如果进行了核酸检测，那么他拥有唯一的试管编号。于是设计 manageSystem 类下两个成员变量 singleTube，mixedTube 为无序哈希表。对于 singleTube 而言，哈希表键(key)为试管编号，哈希表值(value)为该试管所对应的人员代码。而对于 mixedTube 而言，哈希表键(key)为试管编号，哈希表值(value)为数组指针，此数组指针指向存放着 10 位混检人员的人员代码的数组。并且对于哈希表的存储结构而言，查询与修改的效率为$O(1)$，很适合进行大规模多次的查询与修改。

2.4 次密接楼号：集合

再进行核酸检测登记时，如果队伍之中出现了阳性，则此确诊人员的前 10 人与之后排队的所有人都为密接，此时，有很大的概率此确诊人员之前、之后的人来自于同一栋宿舍楼，则他们的楼号相同。而如果将这些重复的楼号多次存储，则会造成存储空间，操作时间的很大浪费。因此，在存储楼号时，需要进行楼号的去重操作，所以使用集合存储次密接楼号。manageSystem 类下 secondaryContagiousBlockNum 集合中存储密接者所在的楼号，通过遍历集合即可设置相关人员为次密接状态。集合中的值不允许重复，如果插入的值在集合之中已经存在，则会自动完成去重操作，查找的效率为$O(1)$，能较好完成此任务。

2.5 人员状态查询：哈希表

由于每一个人员有唯一的人员代码，因此使用哈希表存储每一个人的状态。personMap 成员变量的键(key)为人员代码，值(value)为人员代码对应的 Person 对象，此 Person 对象的成员变量包含了此人的核

酸检测状态。使用哈希表存储，优点在于当根据人员代码查询相应的人员状态时，如果采用数组、链表等数据结构进行存储，则需要遍历完一遍数组或链表，时间复杂度为$O(n)$，效率较差。而使用哈希表存储时，根据此人员代码的哈希值即可判断此人是否在此哈希表是否存储过，时间复杂度为$O(1)$。

3．各个函数功能描述

本核酸检测系统采取了模块化设计的思想，将每一个要求的功能模块封装为对应的函数。这种设计思想便于在主函数菜单之中方便的调用，也更有利于开发时的维护，使得逻辑清晰、条理。

本系统将用到的函数分为以下两类：

1. 功能函数，即满足 6 个系统功能的对应函数
2. 辅助函数，即功能函数实现功能所依赖的函数，负责对数据结构与算法的实现。

两类函数声明如下表所示：

表 7. 核酸检测系统类模板中各功能模块对应功能函数的声明

功能	函数原型	描述
1 排队	void line_up();	分别对混合、单管进行排队操作
2 核酸检测	void nucleic_acid_test();	分别对混合、单管进行检测操作
3 查看排队情况	void show_line_up();	展示混合、单管目前的排队情况
4 登记测试管信息	void Register_test_tube();	录入检测信息，更新人员检测状态
5 各类人员查询	void status_category_query();	查询某检测状态下所有的人
6 个人查询	void query_for_person();	由人员编号，查询此人检测状态

表 8. 核酸检测系统类模板中辅助函数的声明

序号	函数原型	描述
1	void handle_negative_cases(int tubeNum, bool isSingle);	处理阴性检测结果
2	void handle_positive_cases(int tubeNum, bool isSingle);	处理阳性检测结果
3	void handle_contagious(int personNum);	处理密接检测结果
4	void set_secondary_contagious();	处理次密接检测结果

三、详细设计

1．各个函数的调用关系图

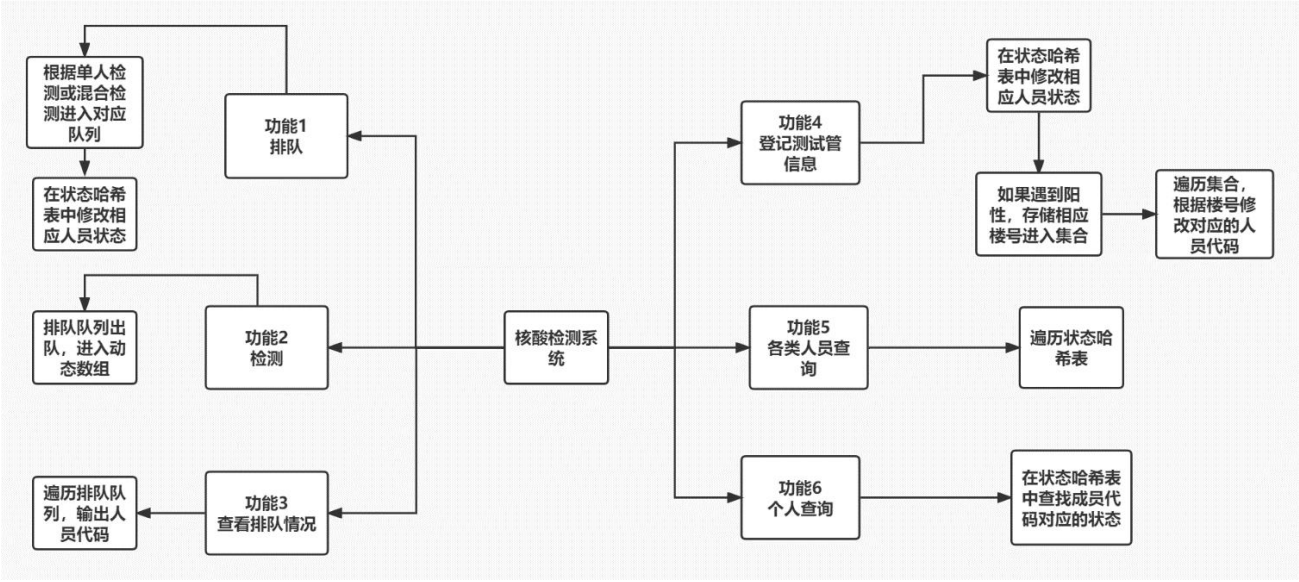

图 7. 函数调用关系图

2. 函数设计

2.1 排队：line_up 函数

从“line_up.in”排队信息文件之中读取到混合测试的排队人数与单人测试的排队人数，再依次将他们的人员代码排入链队列，此处将单人测试与混合测试两个队伍分开，分别存入对应的链队列之中。由于此处针对单人检测与混合检测的操作相同，因此只列出入队时部分核心代码，在此人入队后，将此人的状态在人的状态哈希表 personMap 中，更新此人状态为在排队，代码实现如下：

```
for (int i = 0; i < n; i++) {
    int num;
    lineupFile >> num;
    mixedTest.EnQueue(num);
    // 人员状态更新
    Person p = Person(num);
    p.setIsInLine(true);
    // 在状态哈希表中进行插入此人
    personMap[num] = p;
}
```

图 8. 排队函数核心代码

2.2 核酸检测：nucleic_acid_test 函数

从“nucleic_acid_test.in”核酸检测信息文件之中读取到混合测试与单人测试的的已检测人数。将这些人从对应的队列之中出队，并将此人编号存储到待上传结果动态数组之中，并且设置他为待上传结果状态。

以下选取混合检测为例，由于每一个试管对应 10 个人员，因此，混合检测哈希表的键(key)存储生成的试管流水号，混合检测哈希表的值(value)存储指向大小为 10 个 int 的数组首地址，并且在堆空间之中开辟此数组，分别存储这 10 个人的编号。考虑到此次混检时，此试管可能对应人数少于 10 人，因此将多余的部分存储默认人员编号 0。考虑到了特殊情况.这样的处理使得系统可以处理未满 10 人的混合检测情况，使得程序更为健壮。

```
for (int i = 0; i < x; i++) { // x 为已经进行了混合检测的人数
        int temp;
        mixedTest.DelQueue(temp); // 混合队列出队
        // 设置为等待上传检测结果
        personMap[temp].setIsWaitingForUpload(true);
        personMap[temp].setIsInLine(false);
        // 每 10 人,在堆中重新申请一个数组
        if (i % 10 == 0) {
            // 初始化,同时赋初始值 0
            personNum = new int[10]();
            mixedTubeNum = i / 10;
            mixedTube[mixedTubeNum] = personNum; // 在混合试管哈希表中存储人员代码
        }
        personNum[i % 10] = temp;
        mixedWaiting.push_back(temp); // 在待上传结果动态数组中中存储人员代码
}
```

图 9. 核酸检测核心代码

2.3 查看排队情况：show_line_up 函数

即为遍历当前正在排队的单人检测与混合检测队列，将每个节点值进行输出,以下代码以混合检测为例，单人检测同理.

```
for (Node<int> *p = mixedTest.front->next; p != nullptr; p = p->next) {
    cout << "队内序号: " << count << " 人员编号: " << setw(8) << setfill('0') << p->data;}
```

图 10. 查看排队情况核心代码

2.4 登记测试管信息：Register_test_tube 函数

从文件“Registration.in”中读取每一根试管的状态，并且对阴性与阳性分开处理。本函数之中调用了两个辅助函数 handle_negative_cases 与 handle_positive_cases 分别处理阴性与阳性情况，如下：

```
for (int i = 0; i < m; i++) {
    int tubeNum, status;
    registrationFile >> tubeNum >> status; // 从文件中读取试管号与检测状态(阴性、阳性、混合)
        // 单人管 管号由 1 打头，判断是否为单人管
    bool isSingle = tubeNum >= 10000;
    switch (status) {
        case 1: // 处理阴性
            handle_negative_cases(tubeNum, isSingle);
            break;
        case 2: // 处理阳性
            handle_positive_cases(tubeNum, isSingle);
            break;
    }
}
set_secondary_contagious(); //全部处理好之后,统一处理次密接
singleWaiting.clear();// 处理完成,清空待上传 vector
mixedWaiting.clear();
```

图 11. 登记测试管信息核心代码

情况可分为 4 种:

1. 单人检测阴性
2. 单人检测阳性
3. 混合检测阴性
4. 混合检测阳性

其中，单人检测阴性与混合检测阴性、阳性思路相似，在试管 map 之中由试管编号寻找对应的人员代码，此处省略介绍。

对于单人检测阳性而言，情况复杂。在处理时分为以下步骤：

1. 阳性人员的宿舍楼中所有人设置为密接
2. 待上传结果动态数组中前十个人直至末尾设置为密接
3. 他们的宿舍楼中所有人设置为次密接
4. 正在排队的队列中所有人设置为密接
5. 他们的宿舍楼中所有人设置为次密接

这些操作中，在设置上传结果的动态数组以及排队队列时需要进行遍历操作，时间复杂度为$O(n)$，将密接的楼号存入楼号集合时，由于使用了集合的数据结构，利用了集合的去重操作，时间复杂度为$O(1)$。

```
void managementSystem::handle_positive_cases(int tubeNum, bool isSingle) {
    // 阳性
    int personNum;
    if (isSingle) {
        personNum = singleTube[tubeNum]; // 若为单管,在单管试管 map 中查找
        personMap[personNum].setIsDiagnosed(true); // 设置阳性人员状态为确诊
        personMap[personNum].setIsWaitingForUpload(false); // 设置为已上传检测结果
        int i;
        for (i = 0; singleWaiting[i] != personNum; i++); // 在 waiting 中找到阳性的那个人
        // 1. 处理在阳性人员的宿舍楼中所有人
        int blockNum = personMap[personNum].getBlockNumber(); // 取出楼号
        secondaryContagiousBlockNum.insert(blockNum); // 阳性人员所在楼号入集合
        for (auto &item: personMap)
            // 如果阳性的楼号与哈希表中的人楼号相同, 则修改其状态为密接
            if (blockNum == Utils::getBlockNum(item.first))
                item.second.setIsContagious(true);
        int j;
        // 2. 处理在单管队列中正在等待结果的所有人
        if (i <= 10) // 如果这名阳性是前 10 个人,则修改起始位置
            j = 0;
        else
            j = i - 10;
        for (; j < singleWaiting.size(); j++) {
            int frontNumbers = singleWaiting[j]; // 取得排在后面的人的代码
            handle_contagious(frontNumbers);
        }

        // 3. 处理在单管队列中正在排队的所有人
        for (Node<int> *p = singleTest.front->next; p != nullptr; p = p->next) {
            int subsequentNumbers = p->data; // 取得排在后面的人的代码
            handle_contagious(subsequentNumbers);
        }
    } else {
        // 若为混合检查,在混合检查试管 map 中查找
        for (int i = 0; i < 10; i++) {
            personNum = mixedTube[tubeNum][i];
            if (personNum != 0) {
                personMap[personNum].setIsSuspicious(true);
                personMap[personNum].setIsWaitingForUpload(false); // 设置已上传检测结果
            }
        }
    }
}
```

图 12. 阳性情况处理代码

2.5 各类人员查询：status_category_query 函数

对于阴性、确诊、可疑、密接、次密接这 5 种情况而言，操作类似，遍历存储人员状态的哈希表 personMap。对于待上传结果、在排队这 2 种情况而言，由于待上传结果数组与在排队队列在之前的操作之中一直在维护，因此，分别遍历数组、队列即可获得结果。需要注意的是，由于本系统设计时将单人检测与混合检测分别存储，输出时需要将两种检测方式都进行遍历。

针对阴性、确诊、可疑、密接、次密接这 5 种情况，以查询所有的次密接为例，需要注意条件，当此人并非确诊、密接、可疑状态时，输出此人的编号。

```
for (auto &item: personMap) {
    if (item.second.isSecondaryContagious() && !item.second.isContagious()){
        cout << '(' << i << ")\t" << " 人员代码:\t" << setw(8) << setfill('0') << item.first;
   }
}
```

图 13. 次密接状态查讯核心代码

2.6 各类人员查询：status_category_query 函数

在获取人员编号之后，在存储人员状态的哈希表中，根据人员代码查找到相应的 Person 人员对象。查找到此对象之后，进行如下图的状态判断，并将处理过的状态输出返回。

由于使用了哈希表进行存储，查询与修改的时间复杂度降低到$O(1)$。

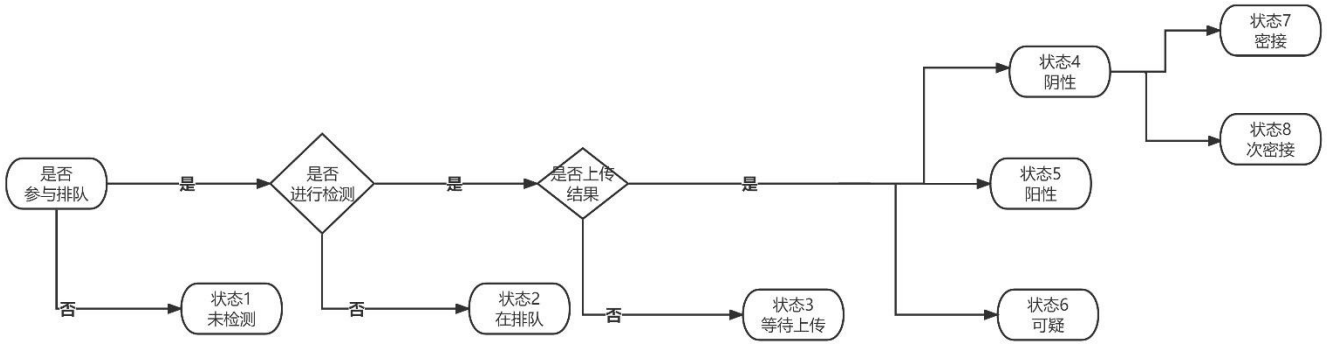

图 14. 各类人员查询流程

四、功能测试

4.1 各类人员查询测试

在进行排队操作、核酸检测操作、结果录入操作之后，调用各类人员查询功能。

本系统为了便于测试，将登记结果存入“Registration.in”文件之中。

4.1.1 混合检测部分

本测试样例对混检排队 12 人，已完成核酸 12 人。其中后两人为 1 个混检试管。其中，在检测结果登记时设置试管代码号为 00001 的试管为 2 即阳性，可得下图的结果。根据排队信息与核酸检测录入信息进行推理，结果应该为左图所示。右图为系统程序输出结果，结果相吻合，测试正确。

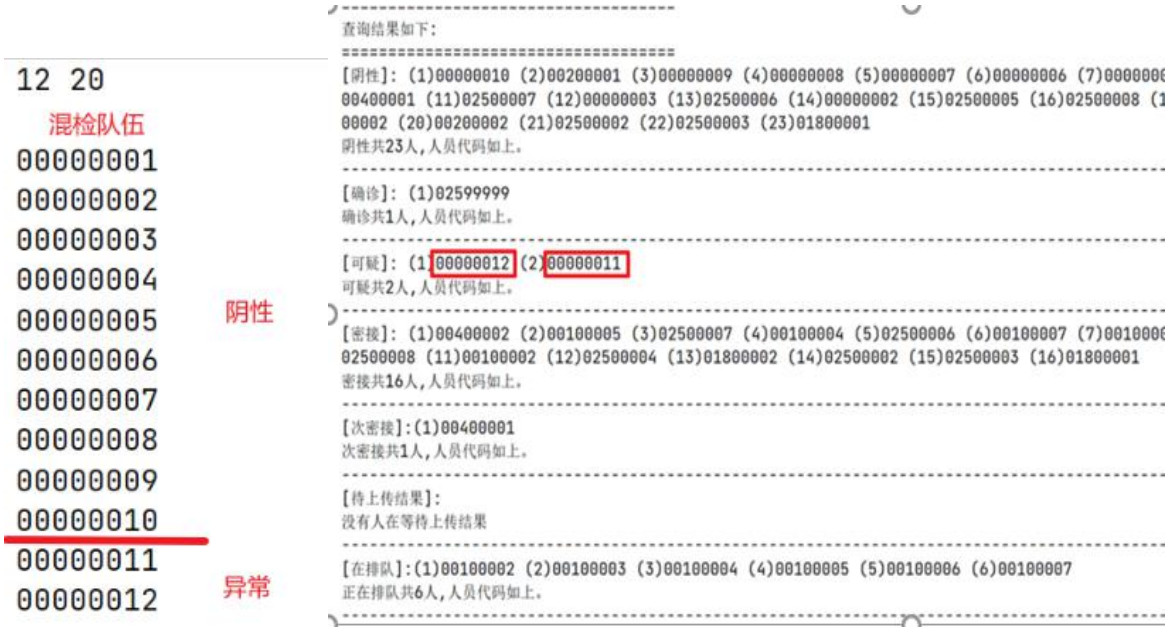

图 15. 混检队伍检测推理结果

图 16.系统功能输出结果

4.1.2 单人检测部分

本测试样例对单人排队 20 人，已完成核酸 14 人。其中有一人为阳性，即人员编号 02599999。下面对于此题样例的检测结果进行推理：根据题意，他之前的 10 人与之后的所有人均为密接。阳性的 02599999 之前 10 人的分界线为 00400002 号，因此，此人的状态为阴性、密接，而 00400001 是阳性的 02599999 之前的第 11 个人，他并不是密接，但由于他与密接 00400002 都来自于同一个宿舍楼 004，因此他为密接。

00200002、00200001 分别为阳性的 02599999 之前的第 12、13 人，他们均为阴性，且既不是密接，也不是次密接。

而 02599999 是本次完成核酸的最后一个人，在他之后的 0010002、0010003 等人均没有进行核酸，但由于他们处于阳性的 02599999 之后，因此，他们的状态为在排队、密接。

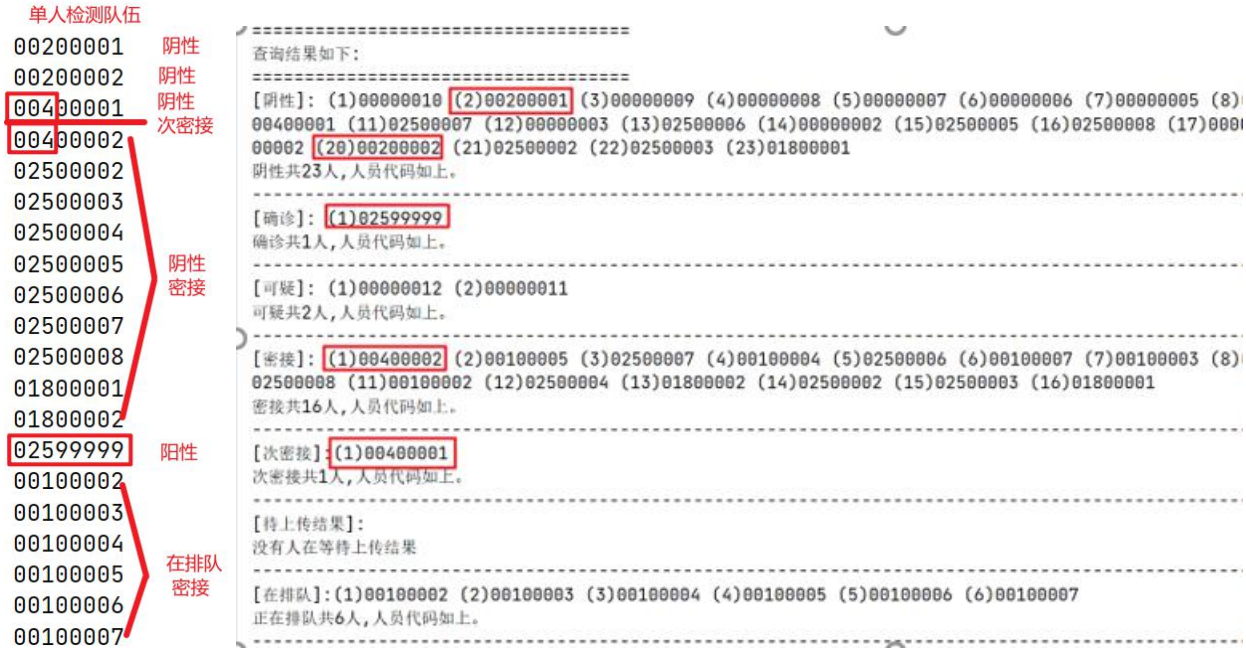

图 17. 单人检测推理结果

图 18. 系统功能输出结果

通过推理分析可知，系统展示结果与推理结果相同，此程序可以处理类似的复杂情况。

4.2 各类人员查询测试

本测试样例为在各类人员查询测试上，继续对于特殊的人员代码进行测试，分析思路同上，测试结果如下图所示。

图 19. 各类人员查询测试结果

课程总结：

在本学期《数据结构 1》的学期过程之中，我接触到了许多数据结构，链表、队列、数组、串等等。尝试从效率、时间、空间、稳定性等多个角度全面的思考等待解决的问题。我更理解到针对不同的问题，首先分析问题的特点，有针对性的选用数据结构，使用合适的算法。学习数据结构，培养了我一种严谨的思维习惯，正如同数据结构的设计一样，如果在做一件事情前没有提前对问题进行充分的思考与琢磨，不假思索便草草上手，那么在最后，即使能够完成问题，做这件事情的效率也不会高。也因此许多人把《数据结构》这门课程看作计算机专业最重要的一门课程。

最后，十分感谢在百忙之中抽出时间审阅本文的老师。也正有着老师的带领，为我打开了数据结构的大门。由于本人的学识和写作的水平有限，在本报告的写作中难免有僻陋，恳请老师多指教。