西南石油大学课程设计报告

课程	姓名	学号
算法分析与设计		
专业班级	任课老师	成绩
软件工程	王欣	

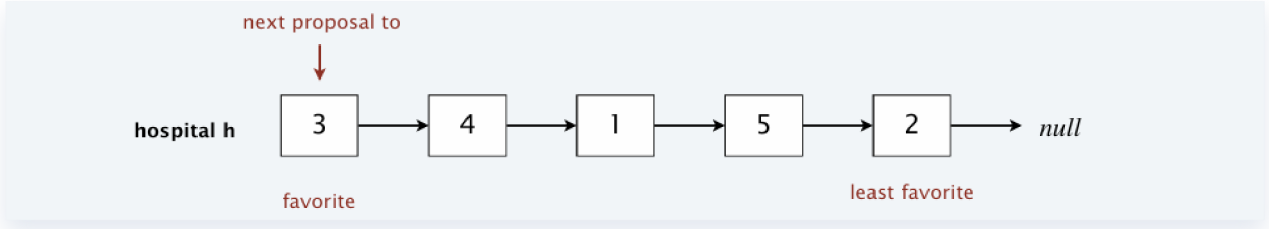
一、设计目的

给定n家医院和n名学生，以及他们的偏好列表，如果存在稳定的匹配，则找到这样的匹配，即为了解决在两个不同群体之间进行配对的问题，使得配对结果既满足个体的偏好，同时又保证配对的稳定性，并掌握稳定匹配问题中$Gale–Shapley$算法的C++实现方法。

二、关键操作

数据表示：医院向学生发送offer

- 关键操作：找到医院下一个最喜欢的学生
- 对于每个医院：维护一个针对学生的偏好列表
- 对于每个医院：维护一个指向学生的**指针**

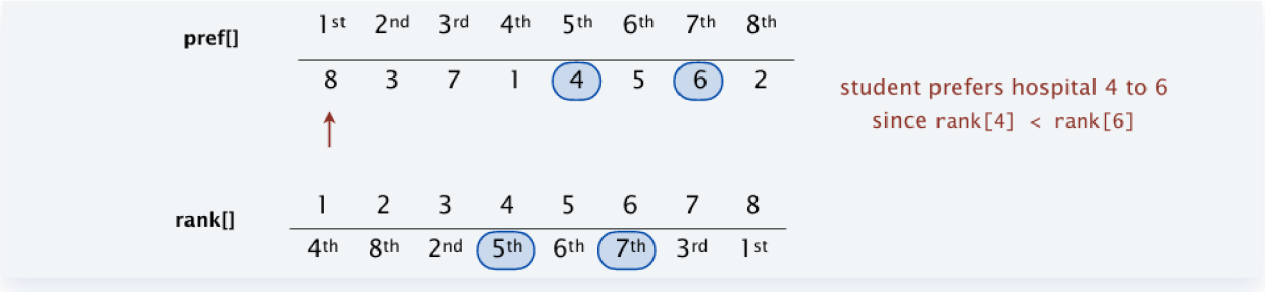


指针示意图

重点：发送一次offer只需要$O(1)$的时间

数据表示：学生接受/拒绝offer

- 比起医院h，如果学生s更喜欢h'怎么办?
- 针对每个学生，创建一个针对医院关于偏好的**倒排索引**



倒排索引

重点：在$O(n^2)$的预处理时间（创建n个排名数组）后，只需要花费$O(1)$的时间去接受/拒绝一个offer

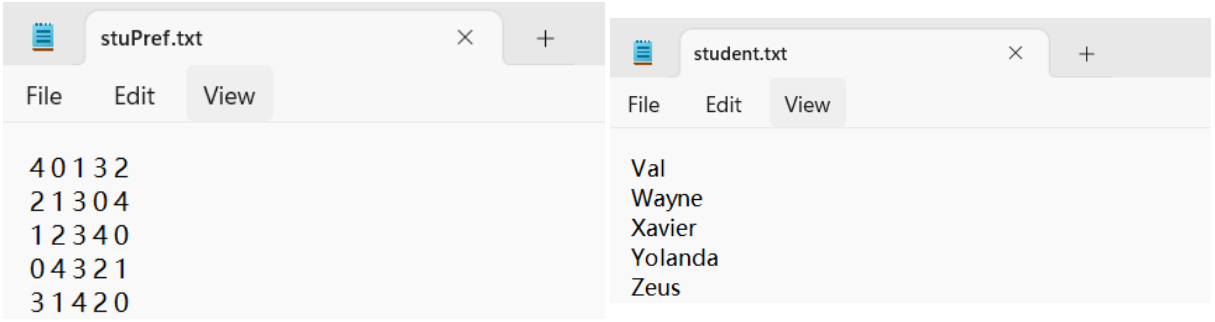
数据表示：未匹配的医院

- 维护一个存储未匹配的医院的队列或栈
- 在$O(1)$时间内能找到

三、步骤

1.数据预处理

将医院和学生的基本信息、偏好列表存储到4个 `txt` 文件中，如图：



偏好列表和基本信息

首先定义一个 `DataProcessor` 的类，用于对测试的数据进行预处理。它有5个成员变量，如下图所示：

```
// 建立索引到医院/学生的映射，便于后续输出
unordered_map<int, string> mapToHospital;
unordered_map<int, string> mapToStudent;

// 学生对医院/医院对学生的偏好列表
vector<vector<int>> stuPrefList;
vector<vector<int>> hosPrefList;

// 数据文件的根目录
const string ROOT=R"(D:\COURSE\assignment\01-Gale_Shapley\data\)";
```

5个成员变量

前面两个哈希表用于建立由索引到医院/学生的映射，有利于后续主函数的输出，后面两个向量保存学生对医院/医院对学生的偏好列表，最后一个 `ROOT` 是数据文件的根目录，需要注意，这里面不应该对 `ROOT` 设置一个 `get` 的方法，应该保持该变量私有，因为没有理由将数据存储的地方暴露给主函数。

`DataProcessor` 还有两个成员函数，如下图所示：

```
// 给出文件名，将学生/医院的基本信息结果存储到哈希表中
void readInfo(string fileName);

// 给出文件名，将学生对医院/医院对学生的优先级列表存储到二维向量中
void readPrefList(string fileName);
```

2个成员函数

这两个函数的作用都是从文件中读取数据并将其存储到上述提到的成员变量中，但是因为基本信息和偏好列表存储的格式不同，因此分为两个函数。

2.计算结果

设计一个*GaleShapley*类，在其中实现 `GaleShapley` 函数。它将学生对医院和医院对学生的偏好列表作为输入，然后返回匹配的结果，所以调用稳定匹配函数 `GaleShapley` 对测试数据进行运算，并将得到的结果存储到一个二维向量 `result` 中。

3.输出结果

遍历 `result` ，并根据其中的索引值去哈希表中查找对应的医院和学生，最后输出结果。

四、代码

*GS*算法的实现如下：

```
vector<vector<int>> GaleShapley(vector<vector<int>> hosPrefList,
vector<vector<int>> stuPrefList)
{
    int hosSize = hosPrefList.size();
    int stuSize = stuPrefList.size();

    // 医生和学生数量不匹配，无法完美匹配
    if (hosSize != stuSize) return {};

    int N = hosSize;
```

```
    // 建立一个学生对医院偏好的倒排索引
    vector<vector<int>> invertedStuList;
    for (int i = 0; i < N; i++)
    {
        vector<int> rank(N, -1);
        for (int j = 0; j < N; j++)
            rank[stuPrefList[i][j]] = j;

        invertedStuList.push_back(rank);
    }

    // 匹配结果，初始化为空
    vector<vector<int>> S = {};

    // 维护一个未匹配的医院队列
    queue<int> unmatchedHos;
    for (int i = 0; i < N; i++)
        unmatchedHos.push(i);

    /*
     * HOSPITAL[s]，若学生s匹配到医院h，HOSPITAL[s]=h
     * 用-1表示医院和学生未匹配，初始化全设为-1
     */
    vector<int> HOSPITAL(N, -1);

    // 为每个医院维护一个指针（索引），记录喜爱列表中下一个学生的索引，初始化为0（数组索引从0开始）
    vector<int> stuPointer(N, 0);

    // 未匹配的医院队列非空
    while (!unmatchedHos.empty())
    {
        // 未匹配的医院队列出队一个
        int hos = unmatchedHos.front();

        // 从医院对学生的优先级列表中选择一个学生
        int stu = hosPrefList[hos][stuPointer[hos]];

        // 指针递增，指向下一个学生
        stuPointer[hos]++;

        // 该学生仍未匹配到任何一个医院
        if (HOSPITAL[stu] == -1)
        {
            S.push_back({hos, stu});
            unmatchedHos.pop();
```

```
            // 记录医院和学生匹配
            HOSPITAL[stu] = hos;
        }
        // 出现了更好的医院
        else if (invertedStuList[stu][HOSPITAL[stu]] > invertedStuList[stu]
[hos])
        {
            int index = 0;
            for (; index < S.size(); index++)
                if (S[index][1] == stu)
                    break;

            // 存储原匹配的医院
            int oldHos = S[index][0];

            // 从S中删除
            S.erase(S.begin() + index);

            // 在S中添加新匹配的hos
            S.push_back({hos, stu});

            // 将新匹配的医院出队
            unmatchedHos.pop();
            HOSPITAL[stu] = hos;

            // 原匹配的医院入队
            unmatchedHos.push(oldHos);
        }
        else
        {
            // 学生拒绝医院
        }
    }

    return S;
}
```

整体的代码结构：

```
│  DataProcessor.cpp
│  DataProcessor.h
│  GaleShapley.cpp
│  GaleShapley.h
│  main.cpp
│
└─data
        hospital.txt
        hosPref.txt
        student.txt
        stuPref.txt
```

5个代码文件，4个数据文件

五、测试

下面对实现的算法进行一个简单的测试，其中医院和学生的姓名，以及医院对学生的偏好列表和学生对医院的偏好列表如下所示：

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

测试数据

经过GS算法模拟计算后的结果如下图所示：

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

GS模拟后的结果

实际的GS算法程序跑出来的结果：

GS程序运行后的结果

可以看到两者的结果一致，测试完毕。

六、体会

使用C++语言编写了GS算法并给出了一个测试用例，体会到了算法实现、数据处理与主函数分离的好处，使用了C++内置的标准库，如队列、哈希表和向量，并融入了一点面向对象的编程技巧，重温了C++操作文件的知识，最后用命令行传递参数的形式来执行程序。遇到的困难是在进行文件读取的时候，使用绝对路径的时候基本没问题，但是相对路径就有一些莫名其妙问题：比如将 `ROOT` 定义为宏的时候运行无误，但是将它定义为类 `DataProcessor` 的一个私有变量的时候就报错"打不开文件"。还有一个问题就是在 `windwos` 平台上运行无误，但是在 `linux` 上就会有一些问题，可能是两个平台之间在文件路径上的表示和编码有所不同带来的结果。