

# CSP

## 变量集合、值域集合以及约束集合

### 变量集合：

每一天的每一班是一个变量，变量数量为 $D \times S$ ， $D$ 为天数， $S$ 为每天的班数

### 值域：

用“班次”代表每一天的每一班，则班次的值域为所有阿姨的编号，初始为-1表示未赋值，有 $N$ 个阿姨，则值域为 $\{0, 1, 2, \dots, N\}$ 。

### 约束集合

#### ■ 每天分为轮班次数个值班班次：

1. 在读入数据时，就初始化需要填充的排班表为一个 $D \times S$ 的矩阵， $D$ 为天数， $S$ 为轮班次数

#### ■ 不连续值班： $(i, j)$ 表示第 $i$ 天第 $j$ 班

1. 两天交界处： $(i, j)$ 为前一天最后一班
2. 两天交界处： $(i, j)$ 第二天第一班
3. 同一天连续两班：前一班和 $(i, j)$ 相同
4. 同一天连续两班：后一班和 $(i, j)$ 相同

#### ■ 公平性：

1. 每个宿管阿姨在整个排班周期中，应至少被分配到 $\lfloor \frac{D \cdot S}{N} \rfloor$  (向下取整) 次值班：对于得到的排班表，统计每个阿姨值班的次数，如果小于该阈值，则该值班方案不合法。

#### ■ 每一班都有阿姨值班：

1. 检查得到排班表：是否每个班次都有阿姨值班，即每个班次的阿姨编号都 $\geq 0$ ，如果没赋值则为-1

#### ■ 尽可能最大化满足的请求数：

1. 初始设置每个班次的剩余值为对应有请求的阿姨编号，如果剩余值为空，再去寻找是否还有存在请求但未满足的阿姨，虽然她没有请求当前班次，但当前班次没有阿姨值班，必须要选一个阿姨值班

# 算法的主要思路

## ■ 数据结构：为了更直观地操作变量和赋值，将阿姨和每天的排班分别设计为两个结构体

1. 阿姨：Aunt只有一个属性，request：阿姨的请求列表，是一个二元组的容器，存放对应阿姨的请求班次(i,j)：第i天第j班。这里阿姨不需要阿姨编号属性，因为后续将阿姨放入一个数组中，对应的下标就是阿姨编号。
2. 班次信息：DayShift有两个属性，分别是auntNo：值班阿姨编号，remainValue：剩余可选的阿姨，用一个数组表示。
3. 初始化：从文件中读取矩阵，根据N的值分割矩阵，得到每个阿姨的请求矩阵，再遍历每个阿姨的为1的坐标，存入阿姨的请求列表中，排班表根据阿姨的请求列表初始化，对应的阿姨编号初始化为-1。

```
//结构体：阿姨
struct Aunt {
    vector<pair<int,int>> request;//阿姨的请求列表
    Aunt() {
        request = vector<pair<int,int>>request();
    }
};

//结构体：某天的排班信息
struct DayShift{
    int auntNo; //值班的阿姨编号
    vector<int> remainValue; //剩余可选的阿姨
    DayShift(){
        auntNo = -1;
        remainValue = vector<int>();
    }
};
```

## ■ 算法：回溯搜索+启发式

1. 递归实现回溯搜索：传入的主要参数为排班表和阿姨的列表，在递归入口检查约束，如果满足则返回true，否则选择一种获取变量的方法，有枚举、minimum remain value、mrv+least constraint value、mrv + lcv +constraint propagation，返回值为一个班次(i,j)，在启发式方法中，还会修改对应班次的剩余值取值范围以及剩余值选择顺序等，选择好要赋值的变量后，就遍历该变量的剩余值，对每个取值检查是否与当前已选值相容，即阿姨不连续上班，如果满足的话，需要删去对应的剩余值取值，以及该取值(即阿姨)请求列表中的对应请求(i,j)，然后再进入下一个分支，即继续递归，前向检验的思想在回溯算法中体现，即不相容则当前分支提前失败，返回false。
2. 获取变量的方式分别为enumerate、mrv、mrv+lcv、mrv+lcv+cp，其中mrv选择最少阿姨请求的班次，lcv+mrv先将最少剩余的变量选出来再对变量的剩余值进行排序，mrv+lcv+cp则将变量的剩余值根据阿姨已经排班数从小到大排序。

# 优化方法

1. 为了能够最大化满足阿姨的请求数，选择对每个班次剩余值设置为对应请求该班次的阿姨编号，这样在使得变量有取值的同时能够优先满足请求。
2. 一开始将检查是否连续上班放在约束检查中，即得到完整方案才检查，发现可以通过约束传播，在每次选取变量赋值时就检查，这样能够更快发现失败的情况。
3. 检查值是否相容，提前发现失败的情况，如果当前位置取值会导致阿姨连续值班则直接返回false，而不用继续赋值。
4. 使用最少剩余值选择变量进行赋值，即优先选择阿姨请求最少的班次。

5. 最少约束值，班次有多个取值可选，此时就考虑最少排班的阿姨，这样就能够减少对其他阿姨选择的约束，因为每个阿姨都需要满足一定的阈值。

# 优化效果

## Enumerate

| 输入文件       | 运行结果              | 运行时间     | 遍历的方案数 | 满足请求数     |
|------------|-------------------|----------|--------|-----------|
| input0.txt | Find A Solution ! | 0.003s   | 102    | 20/21     |
| input1.txt | Run timed out !   | 22.745s  | 98414  | 28/60     |
| input2.txt | Find A Solution ! | 0.026s   | 98502  | 33/33     |
| input3.txt | Find A Solution ! | 10.299s  | 126162 | 114/114   |
| input4.txt | Find A Solution ! | 0.026s   | 126234 | 69/69     |
| input5.txt | Run timed out !   | 23.217s  | 127050 | 291/576   |
| input6.txt | Run timed out !   | 66.725s  | 128157 | 503/1008  |
| input7.txt | Run timed out !   | 18.301s  | 129351 | 191/378   |
| input8.txt | Run timed out !   | 495.891s | 131532 | 1068/2160 |
| input9.txt | Run timeout !     | 103.592s | 132288 | 352/720   |

## Minimun Remain Value

| 输入文件            | 运行结果              | 运行时间     | 遍历的方案数 | 满足请求数    |
|-----------------|-------------------|----------|--------|----------|
| 满足请求数input0.txt | Find A Solution ! | 0.002s   | 51     | 20/21    |
| 满足请求数input1.txt | Run timed out !   | 138.397s | 34499  | 28/60    |
| 满足请求数input2.txt | Find A Solution ! | 0.085s   | 34683  | 33/33    |
| 满足请求数input3.txt | Find A Solution ! | 0.436s   | 35361  | 114/114  |
| 满足请求数input4.txt | Find A Solution ! | 0.036s   | 35448  | 69/69    |
| 满足请求数input5.txt | Run timed out !   | 231.395s | 36213  | 291/576  |
| 满足请求数input6.txt | Run timed out !   | 163.298s | 37300  | 503/1008 |
| 满足请求数input7.txt | Run timed out !   | 142.87s  | 38180  | 191/378  |

| 输入文件            | 运行结果            | 运行时间     | 遍历的方案数 | 满足请求数     |
|-----------------|-----------------|----------|--------|-----------|
| 满足请求数input8.txt | Run timed out ! | 512.216s | 40359  | 1068/2160 |
| 满足请求数input9.txt | Run timeout !   | 103.977s | 41109  | 352/720   |

## MRV+Least Constraining Value

| 输入文件       | 运行结果              | 运行时间     | 遍历的方案数 | 满足请求数     |
|------------|-------------------|----------|--------|-----------|
| input0.txt | Find A Solution ! | 0.005s   | 140    | 20/21     |
| input1.txt | Find A Solution ! | 0.007s   | 200    | 60/60     |
| input2.txt | Find A Solution ! | 0.002s   | 233    | 33/33     |
| input3.txt | Find A Solution ! | 0.016s   | 347    | 114/114   |
| input4.txt | Find A Solution ! | 0.009s   | 416    | 69/69     |
| input5.txt | Run timed out !   | 130.652s | 1147   | 291/576   |
| input6.txt | Find A Solution ! | 3.425s   | 2155   | 1008/1008 |
| input7.txt | Find A Solution ! | 0.377s   | 2533   | 378/378   |
| input8.txt | Run timed out !   | 417.556s | 4705   | 1068/2160 |
| input9.txt | Find A Solution ! | 3.173s   | 5425   | 720/720   |

## MRV+Least Constraining Value+Constraint Propagation

| 输入文件       | 运行结果            | 运行时间   | 遍历的方案数 | 满足请求数   |
|------------|-----------------|--------|--------|---------|
| input0.txt | Find A Solution | 0s     | 21     | 20/21   |
| input1.txt | Find A Solution | 0.006s | 81     | 60/60   |
| input2.txt | Find A Solution | 0.002s | 114    | 33/33   |
| input3.txt | Find A Solution | 0.018s | 228    | 114/114 |
| input4.txt | Find A Solution | 0.007s | 297    | 69/69   |
| input5.txt | Find A Solution | 1.058s | 873    | 576/576 |

| 输入文件       | 运行结果            | 运行时间    | 遍历的方案数 | 满足请求数     |
|------------|-----------------|---------|--------|-----------|
| input6.txt | Find A Solution | 3.394s  | 1881   | 1008/1008 |
| input7.txt | Find A Solution | 0.345s  | 2259   | 378/378   |
| input8.txt | Find A Solution | 89.495s | 4508   | 2160/2160 |
| input9.txt | Find A Solution | 3.125s  | 5228   | 720/720   |

## input0.txt 的安排方式

在枚举选取变量进行回溯搜索的方式下，取得的安排方式为：

```
1,2,1
2,3,1
3,2,1
3,2,3
1,2,1
3,2,3
1,2,3
20
```

在后续几种方式中，取得的安排方式为：

```
1,2,3
1,3,2
3,2,1
3,1,2
1,2,1
3,2,3
2,1,3
20
```

## Debug

- 一开始发现算法连第一个都没法跑出来，通过debug打印过程中一些变量的信息，发现得到的排班方案总是有几个班次没有阿姨排班，检查程序发现，由于最初设置的每个班次剩余值为对该班次有请求的阿姨编号，自己的算法对于每个班次在选择剩余值的时候，如果该班次没有阿姨请求，那么就没有剩余值可选，就会返回false，并且在第一个样例中，恰好有一个班次所有阿姨都没有请求，所以在这种情况下，需要对剩余值为空的变量补充剩余值，具体方法是将还有请求的阿姨加入剩余值中。
- 调整完毕后，发现自己的算法除了第一个后续都是一直跑不出来，但自己的启发式方法并没有设计错误，也想不到有什么还能更优化的方法，所以一度陷入的瓶颈，感觉自己甚至可能没法做出这个实验了。后来通过调试发现，自己的算法一直无法找到满足每个阿姨最小值班阈值的约束，但是打印出不满足的阿姨的值班次数发现，很多阿姨的值班次数其实已经满足要求，但是被判定为不满足，于是发现自己将 $\lfloor \frac{D \cdot S}{N} \rfloor$ 在代码中写成了 $\lfloor \frac{D \cdot N}{S} \rfloor$ ，

- 这导致后续的N都比S更大，阈值也就大很多，并且在第一个样例中，S和N相等，所以只有第一个样例可以通过！这也是困扰了我最近的一个bug，往往越简单的bug要找最久，因为它的影响比较不明显！
3. 在回溯的递归中，最初的枚举不是通过函数来获取一个对应的(i, j)，而是直接在回溯算法的递归程序中，使用双重for循环，遍历D\*S个(i, j)取值，这导致搜索空间非常大并且存在非常多的重复状态！即使第一个很简单的样例也要运行很久！所以修改这个取值方式，算法效率得到很大改善。
  4. 对于vector的使用，要非常注意初始化的分配空间大小，并且有时候很容易在循环遍历vector时出现越界的错误。

## 总结与体会

1. 通过这次实验，熟悉了CSP问题的定义和结构，进一步理解了什么是约束满足问题，发现人工智能的本质原来就是for循环+if语句。
2. 发现原来用最朴素的搜索算法加上一些针对问题设计的启发式方法，也能够很快地解决一些看起来很复杂的问题，而不一定需要设计多么精巧的数据结构或者使用很多trick的算法。
3. 并且有时候问题看起来很复杂，实际上真正去解决时，很多复杂的情况其实并不会遇到，或者遇到的概率很小，这在很大程度上降低了问题的复杂程度，让我的算法不需要优化到多么高效的地步。
4. 使用的回溯算法让我进一步理解了递归，虽然感觉递归还是不容易理解，比较难在脑海中debug递归的运行过程，但编程的过程还是帮助我加深了对递归的理解。
5. 和A\*实验对比，这个实验我用的指针操作等比较少，出现的报错也更少一点，空指针和指针异常的报错很让人头疼。
6. 我个人认为我的代码结构比较容易理解，查看一些csp问题的代码示例，发现它们的结构大部分都是直接在一个类里面定义各个属性及方法，这样就可以统一在一个域下发挥作用，也比较容易管理，减少了各种传参以及引用之类的，减少了代码的复杂度和出错的可能性，有值得我学习的地方，还有就是虽然这个问题比较简单，但是第一次编程实现csp问题，感觉主要困难是在于如何理解csp问题，当理解之后，代码的逻辑实现其实比较简单，而我认为如果问题更复杂的话，那么编写代码需要用到的优化方法也会更复杂。