P02 CSP and KRR

学号	姓名	专业(方向)
18340161	田蕊	计科（超算）
18340149	孙新梦	计科（超算）

1.task

Futoshiki

- **1）Describe with sentences the main ideas of the GAC algorithm and the main differences between the GAC and the forward checking (FC) algorithm.**
  - **GAC检测**：
    - 所谓GAC检测，就是Generalized Arc Consistency检测。它的主要思想是对于一个CSP问题的每一个约束相关的每一个变量，都检查它是不是存在一组赋值，使得这组赋值满足这个约束，对于那些不满足GAC的变量赋值，也就是我们检测出不能达到解，我们将其从论域中删去来达到剪枝的效果。
    - GAC检测在赋值之前就可以开始做，这样可以为我们的赋值带来更精确的方向，即使不会一下子找出解，也可大大缩减我们赋值的选项数目。
    - 当从值域移除一个值的时候，可能会触发更进一步的不一致性，所以我们每一轮对值域进行修改的时候，都需要再次把和该变量相关，但是又没有出现在GAC检测队列里的约束加入到队列中等待再次检测是否满足，直到队列为空。
    - GAC检测能够有效减少探索的节点数目。
  - **GAC检测和向前检测的区别**：
    - GAC检测：在确定下来一个变量A的取值之后，考虑其他所有未赋值的变量的取值是否存在一组取值使得约束满足，如果不能满足则要在论域中删掉这个取值。假定确定变量A的去之后，考虑与A存在约束的变量B的取值时，不仅考虑B，还要考虑所有和B相关的变量的论域取值。
    - 向前检测：在确定下来一个变量A的取值之后，根据包括变量A的约束集合S，将S涉及到的其他变量根据约束集合的约束进行论域的删减，只考虑取值是不是满足当前与A的这条约束，而不考虑其他未赋值的变量。
    - 因此GAC和FC的最主要区别就是考虑的约束范围，GAC赋值时同时考虑所有相关范围是否满足，但是FC只考虑做这个赋值是否能满足此条约束，不考虑其他的没有赋值的变量。

- **2）The GAC Enforce procedure from class acts as follows: when removing d from CurDom[V], push all constraints C' such that V ∈ scope(C') and C' ∉ GACQueue onto GACQueue. What's the reason behind this operation? Can it be improved and how?**

  因为d可能是所有满足`V ∈ scope(C') and C' ∉ GACQueue`的约束中某些变量满足GAC检测的一个有效支撑，当d从`CurDom[V]`中删除之后，这个支撑就有可能失效，那么就要重新判断那些以d为支撑的变量的论域是否都能满足GAC检测。

  改进：对于每一个约束，我们可以记录下来每一个赋值成立的时候，另一个相关集合的有效支撑（因为这里我们的约束都是二元约束，只有当作为有效支撑的变量值被删减的时候，我们才将涉及到这个变量的约束重新入队。

- 这里使用一个数据结构——evidence，是一个和约束等大小的向量，存放的时对于每一个对应位置的约束，不等号左边和右边取到digits数组每一个值的时候，对应的另外一方的支持是什么

```
vector<pair<vector<int>, vector<int>>> evidence;
```

- 每次判断一个位置的val赋值是否为可行，也就是在can_be_satisfied函数中，在约束队列中检查能否满足当前约束的时候，如果我们可以直接在evidence里面匹配到(x,y)对应数字和他们的支撑并且支撑还是valid的时候，就可以直接判断对应当前约束，这个赋值时可行的，跳过遍历另一个值域来寻找支持的步骤。
- 当值域更大的时候会取得更好的加速效果，因为这个时候能够节省遍历digits的时间，这就是GAC_Enforce的优化思路。

- **3）Use the GAC algorithm to implement a Futoshiki solver by C++ or Python.**
  - **经过优化之后算法思路**：在搜索过程中，我们首先判断此时位置是否满足目标状态，如果没有的话，当这个位置还没有赋值，就遍历数字找到valid的digit取值，保存此时状态并赋值，修改当前位置值域，再把相关的约束入队，做GAC_Enforce操作，用MRV来挑选下一个被赋值的位置，递归调用search函数，之后如果失败的话就需要回溯，恢复之前保存的状态再标记为没有赋值。
- **4）Explain any ideas you use to speed up the implementation**
  - 我们使用的加速方法为使用**MRV**——最小剩余值启发式寻找下一个被赋值的位置。这样可以寻找出更容易获得最后目标节点的下一个赋值位置，减少冗余搜索。
  - MRV算法的实现简要概述如下：
    - 使用数据结构二维数组来存放非法数字的个数，dom_num[i] [j]代表maps对应的第i行，第j列的位置的digits数组有多少个invalid取值，这个值越大表示值域越小，越应该优先赋值。

```
    //存放非法的数字个数，大的被优先赋值
    vector<vector<int>>dom_num;
```

    - 因此每次在GAC_Enforce等操作改变digits的valid与否之后，用函数来修改dom_num这个数组的值

```
countDom(maps[x][y], x, y);
```

    - 每次寻找优先赋值的位置的时候，就使用启发式寻找，找到值域最小的且没有赋值的位置作为下一个位置。

- **5）Run the following 5 test cases to verify your solver’s correctness. We also provide test file "datai.txt" for every test case i. Refer to the "readme.txt" for more details.**
  - 通过读取文件，运行程序测试不同样例，观察到结果满足约束要求，因此我们的程序正确找到解。
  - 具体的实验结果截图放在第三部分呈现。

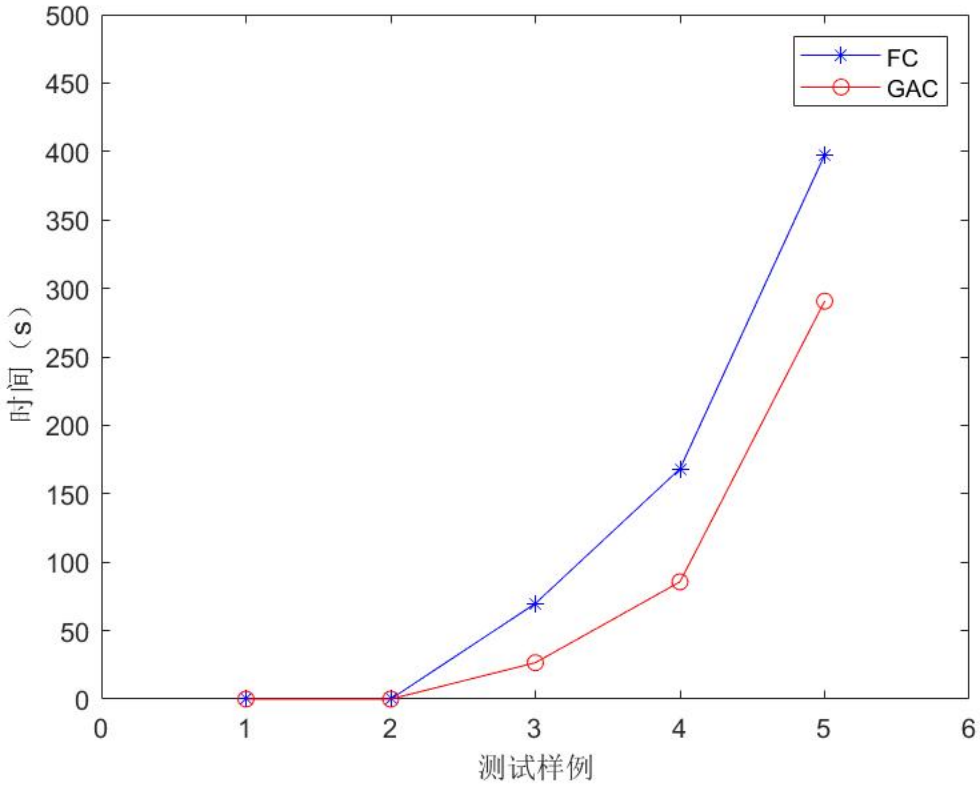
- **6）Run the FC algorithm you implemented in E04 and the GAC algorithm you implemented in Task 3 on the 5 test cases, and fill in the following table. In the table, "Total Time" means the total time the algorithm uses to solve the test case, "Number of Nodes Searched" means the total number of nodes traversed by the algorithm, and**

"Average Inference Time Per Node" means the average time for constraint propagation (inference) used in each node (note that this time is not equal to the total time divided by the number of nodes searched). Analyse the reasons behind the experimental results, and write them in your report

TestCase Number	Method	Time Used	Nodes Searched	Time Per Node
1	FC GAC	0.018 0.008	285 87	1.40351e-05 s 2.0202e-05 s
2	FC GAC	0.002 0.001	84 71	1.19048e-05 s 1.40845e-05 s
3	FC GAC	69.806 26.577	1449622 389308	1.59669e-05 s 1.3691e-05 s
4	FC GAC	168.252 85.662	3573328 1435736	1.49575e-05 s 1.32295e-05 s
5	FC GAC	374.989 290.648	8038421 4437434	1.39113e-05 s 1.30801e-05 s

结果分析：

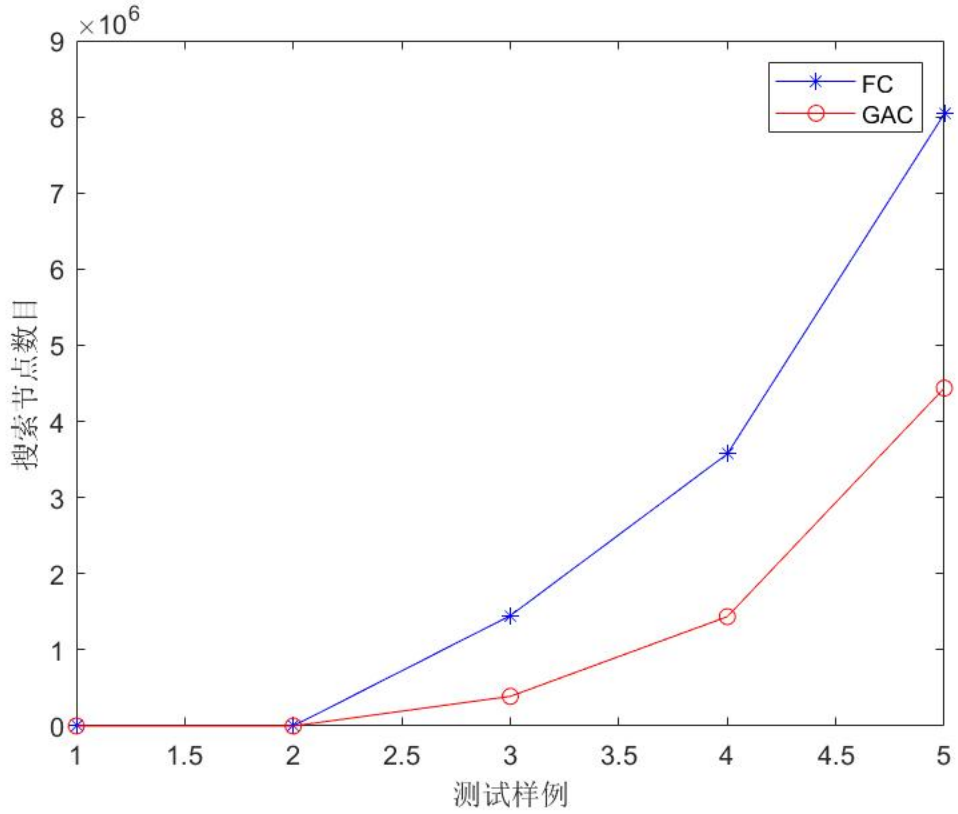
- **1.总时间**



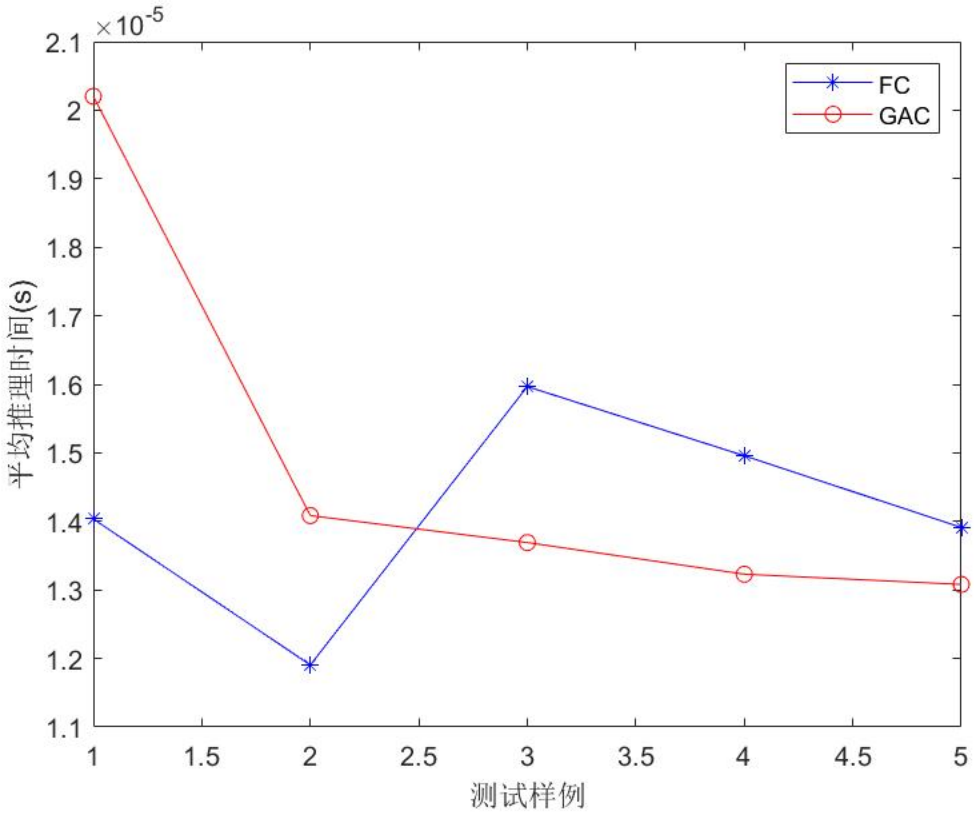
  - 对比不同的测试样例，我们发现随着问题复杂度的提升，不管是FC还是GAC的消耗时间都会增加，扩展的节点也会变多
    - 原因：随着约束增加以及问题规模的增加，不管是推理还是进行搜索的次数都是增加的
  - 在小规模问题上，GAC比FC消耗的时间更少，但是当问题规模增加的时候，使用的时间GAC多于FC检测
    - 原因：由于FC只考虑当前的变量，但是GAC是综合所有的约束在一起考虑是否这个赋值会违反约束，推理的时间开销很大，并且GAC的MRV实现思路会有很多额外的时间开

销，在每一轮search判断是否到达目标，会遍历maps进行判断，以及更新值域的时候都会比FC进行更多的操作，问题规模大的时候更容易产生更大的开销。

- **2.搜索节点数**



  - GAC算法比起FC算法，扩展的节点数目能够达到很好的减少，随着问题规模的增加，GAC对于搜索节点的减少作用越来越明显
    - 原因：当问题的难度增加的时候，也就是规模增加以及约束加多，GAC会在一次次搜索过程中削减值域，因此能够使得搜索的节点大大减少。
- **3.平均节点推理时间**



- 对于GAC算法和FC算法来说，随着问题复杂度的提升，平均每个节点的推理时间减少。
  - 原因：因为FC和GAC都随着搜索过程的进行将值域在一步步缩减，越是复杂的问题到后面推理的时间越少，因此当问题规模增大的时候，平均在每个节点上的推理时间会减少。
- 当问题规模小的时候，GAC算法的平均节点推理时间多于FC算法，但是当问题规模上升到一定规模，GAC算法的平均节点推理时间少于FC算法。
  - 原因：由于GAC_Enforce会将每次的相关约束入队，在值域修改的时候还会再次入队直到所有约束完美满足才会决定赋值进行下一轮搜索，搜索过程一步步缩减值域，能够带来值域的缩减比起FC是会更加前瞻性的，因此虽然问题规模小的时候，额外操作带来的时间开销使得平均推理时间长于FC，问题规模增大的时候GAC的额外开销带来的补偿是非常可观的，因此平均推理时间短于FC。

KRR

- **1).Implement the MGU algorithm.**

  > 源码在Codes部分给出。上交的源文件同样包含MGU部分代码。

  - 寻找MGU，我们寻找一个替换σ能够使得两个子句变为一样的，同时要求其他的替换都可以由σ和其他替换组合达到，也就是保证这个替换是最一般的。
  - 首先判断两个子句能否合一，如果不能直接返回
  - 之后只要两个子句还不是完全相等，就找到第一个不相等的位置，
    - 如果不一样的两项都是term（首字母都是小写），那么不能做合一
    - 如果一个是变量，另一个子句是含有这个变量的项term，那么也不能合一
    - 只有当至少一个是变量并且不会互相包含的时候，才可以合一
      - 对变量的那一方做替换为项，需要记录下做的替换进入σ
  - 寻找MGU的过程是为了归结子句打下基础

- **2).Using the MGU algorithm, implement a system to decide via resolution if a set of first-order clauses is satisfiable. The input of your system is a file containing a set of first-order clauses. In case of unsatisfiability, the output of your system is a derivation of the empty clause where each line is in the form of "R[8a,12c]clause". Only include those clauses that are useful in the derivation.**

  > 源码在Codes部分给出。上交的源文件同样包含Resolution部分代码。

  - 归结一阶子句，通过MGU算法
  - 我们的思路是将最后的结果取非作为输入进去，之后看能否归结出空子句。
  - 每次优先从子句列表的最后一个开始归结，从后往前依次两两归结，新产生的子句放在列表的末尾，也就是优先进行下一轮的归结。
  - 直到只剩空子句，归结过程结束。
  - 接下来进行格式化输出，由于我们在归结的过程中记录下来使用到子句的序号，所以从空子句那里开始依次向前把使用过的子句序号加入到clause_used列表里面，再按照格式输出就好。

- **3).Explain any ideas you use to improve the search efficiency.**

  在我们的算法中采用了两种方式来对搜索进行加速，如下所示

  - 优先选择与结论相关的子句进行匹配

因为为了推导出我们的结论，知识库中的知识我们是不必要全部使用的，如果一直对不需要使用到的知识进行消解那么就会产生很多冗余的操作浪费CPU资源，消耗搜索时间。所以这里我们采取优先搜索和结论相关的子句进行消解，以减少冗余的操作。

- 优先选择新生成的子句进行消解

  在优先选择与结论相关的子句进行匹配的基础上，当我们生成了一个新的子句的时候，那么这个经过消解新得到的子句大概率是更容易达到我们的目标的，所以优先选择新生成的子句进行消解，可以让我们更快地找问题的解。

- **4).Run your system on the examples of hardworker(sue), 3-blocks, Alpine Club. Include your input and output files in your report**

  运行结果和输入输出在结果展示部分给出。

- **5).What do you think are the main problems for using resolution to check for satisfiability for a set of first-order clauses? Explain**

  尽管我们在进行搜索归结出空子句的路径的时候采取了一些启发式的方法尽可能地是我们的搜索冗余减少，但是这还是远远不够的。相对来讲计算机对于子句消解的搜索过程还是十分盲目的，这些方法只能减少一些冗余，但是并不能在本质上使时间开销的数量级改变，所以搜索的时间复杂度其实是指数量级的，而指数级的时间复杂度开销往往是我们不愿意看到的也不能接受的。

2. Codes

1.GAC

```
#include <iostream>
#include <vector>
#include <map>
#include <ctime>
#include <fstream>
#include <string>
#include <queue>
//#include <set>
#define NUM_OF_TEST 2
using namespace std;


typedef pair<pair<int, int>, pair<int, int>> constrain;



class FutoshikiPuzzle {
public:
    //记录当前的
    class node {
    public:
        //当前位置的值
        int val = 0;

        //候选队列，false代表可以选择
        //开了十个空间但是可能只用前几个
        bool digits[10] = { false, false, false, false, false, false, false,
false, false, false };
    };

    struct file {
        int numConstrains;
```

```
        int numRow;
        string filename;
    };

    //地图
    vector<vector<node>> maps;
    //限制
    vector<pair<pair<int, int>, pair<int, int>>> less_constraints;//e.g.[< <1,1>
, <1,2> > ,< <2,2> , <1,2> >]

    //存放非法的数字个数，大的被优先赋值
    vector<vector<int>>dom_num;
    //暂时变量用于被赋值
    vector<vector<int>> copy_dom_num;

    vector<pair<vector<int>, vector<int>>> evidence;

    vector<file> allfile;
    queue<int> GAC_constrains;//存放GAC约束的队列

    int nRow, nColumn;
    int cnt = 0; //用来计数已经搜索的节点
    clock_t inf_start, inf_end;
    double total_inf_time = 0;

    void initial() {
        vector<vector<int>> copy_map = { {0, 0, 0, 7, 3, 8, 0, 5, 0},
                                         {0, 0, 7, 0, 0, 2, 0, 0, 0},
                                         {0, 0, 0, 0, 0, 9, 0, 0, 0},
                                         {0, 0, 0, 4, 0, 0, 0, 0, 0},
                                         {0, 0, 1, 0, 0, 0, 6, 4, 0},
                                         {0, 0, 0, 0, 0, 0, 2, 0, 0},
                                         {0, 0, 0, 0, 0, 0, 0, 0, 0},
                                         {0, 0, 0, 0, 0, 0, 0, 0, 0},
                                         {0, 0, 0, 0, 0, 0, 0, 0, 6} };

        nRow = copy_map.size();
        nColumn = copy_map[0].size();

        maps.resize(nRow);

        //复制到真实地图上
        for (int i = 0; i < nRow; i++) {
            maps[i].resize(nColumn);
            for (int j = 0; j < nColumn; j++) {
                maps[i][j].val = copy_map[i][j];
            }
        }

        //添加限制
        addConstraints(0, 0, 0, 1);
        addConstraints(0, 3, 0, 2);
        addConstraints(1, 3, 1, 4);
        addConstraints(1, 6, 1, 7);
        addConstraints(2, 6, 1, 6);
        addConstraints(2, 1, 2, 0);
        addConstraints(2, 2, 2, 3);
        addConstraints(2, 3, 3, 3);
```

```
        addConstraints(3, 3, 3, 2);
        addConstraints(3, 5, 3, 4);
        addConstraints(3, 5, 3, 6);
        addConstraints(3, 8, 3, 7);
        addConstraints(4, 1, 3, 1);
        addConstraints(4, 5, 3, 5);
        addConstraints(4, 0, 4, 1);
        addConstraints(5, 4, 4, 4);
        addConstraints(5, 8, 4, 8);
        addConstraints(5, 1, 5, 2);
        addConstraints(5, 4, 5, 5);
        addConstraints(5, 7, 5, 6);
        addConstraints(5, 1, 6, 1);
        addConstraints(6, 6, 5, 6);
        addConstraints(6, 8, 5, 8);
        addConstraints(6, 3, 6, 4);
        addConstraints(7, 7, 6, 7);
        addConstraints(7, 1, 8, 1);
        addConstraints(8, 2, 7, 2);
        addConstraints(7, 5, 8, 5);
        addConstraints(8, 8, 7, 8);
        addConstraints(8, 5, 8, 6);

        //根据已经填写的数字开始删除各个位置的候选值
        for (int i = 0; i < nRow; i++) {
            for (int j = 0; j < nColumn; j++) {
                //vector<pair<pair<int, int>, int>> catches = check(i, j);
                vector<pair<pair<int, int>, int>> catches;
                check(i, j, catches);
            }
        }
    }

    //filename带路径，否则就要在同一文件夹下
    void initial(int filenum) {
        allfile.resize(6);
        allfile[0].filename = "data1.txt";
        allfile[0].numRow = 5;
        allfile[0].numConstrains = 7;
        allfile[1].filename = "data2.txt";
        allfile[1].numRow = 6;
        allfile[1].numConstrains = 9;
        allfile[2].filename = "data3.txt";
        allfile[2].numRow = 7;
        allfile[2].numConstrains = 19;
        allfile[3].filename = "data4.txt";
        allfile[3].numRow = 8;
        allfile[3].numConstrains = 25;
        allfile[4].filename = "data5.txt";
        allfile[4].numRow = 9;
        allfile[4].numConstrains = 32;

        //这一部分是文件读取初始化
        fstream f;
        f.open(allfile[filenum].filename);
        vector<vector<int>> copy_map;
        copy_map.resize(allfile[filenum].numRow);
        string str;
```

```
        if (f.is_open()) {
            //获取地图
            for (int i = 0; i < allfile[filenum].numRow; i++) {
                copy_map[i].resize(allfile[filenum].numRow);
                getline(f, str);
                int k = 0;
                for (int j = 0; j < str.size(); j++) {
                    if (str.at(j) != ' ') {
                        copy_map[i][k] = str.at(j) - '0';
                        k++;
                        if (k == allfile[filenum].numRow) {
                            break;
                        }
                    }
                }
            }
            //获取约束
            for (int i = 0; i < allfile[filenum].numConstrains; i++) {
                getline(f, str);
                vector<int> temp;
                for (int i = 0; i < str.size(); i++) {
                    if (str.at(i) != ' ') {
                        temp.push_back(str.at(i) - '0');
                    }
                }
                addConstraints(temp[0], temp[1], temp[2], temp[3]);
            }

        }

        //到这里所有的初始化就结束了


        //复制到真实地图上
        nRow = copy_map.size();
        nColumn = copy_map[0].size();
        maps.resize(nRow);
        evidence.resize(allfile[filenum].numConstrains);
        for (int i = 0; i < less_constraints.size(); i++) {
            evidence[i].first.resize(10);
            evidence[i].second.resize(10);
        }

        for (int i = 0; i < nRow; i++) {
            maps[i].resize(nColumn);
            for (int j = 0; j < nColumn; j++) {
                maps[i][j].val = copy_map[i][j];
            }
        }

        //域里面的个数
        dom_num.resize(nRow);
        for (int i = 0; i < nRow; i++)
        {
            dom_num[i].resize(nColumn);
            for (int j = 0; j < nColumn; j++)
            {
                dom_num[i][j] = 0;//表示有多少非法的数字
```

```
            }
        }


        copy_dom_num.resize(nRow);
        for (int i = 0; i < nRow; i++)
        {
            copy_dom_num[i].resize(nColumn);
        }


        //进行初始的GAC检测
        //最外面两重循环是对于每一行每一列，也就是每一个元素
        for (int x = 0; x < nRow; x++)
        {
            for (int y = 0; y < nColumn; y++)
            {
                //这里是删除行列的候选值，没有使用GAC
                for (int i = 0; i < nRow; i++)
                {
                    //如果本来就是false，那就没必要加入缓存了
                    if (!maps[i][y].digits[maps[x][y].val])
                    {
                        maps[i][y].digits[maps[x][y].val] = true;
                        //意思是在[i,y]格子的候选者中删除了maps[x][y].val候选者
                        //catches.push_back({ {i, y}, maps[x][y].val });
                    }

                    if (!maps[x][i].digits[maps[x][y].val])
                    {
                        maps[x][i].digits[maps[x][y].val] = true;
                        //catches.push_back({ {x, i}, maps[x][y].val });
                    }
                }

                //删除关于限制的候选值
                //这里用的是GAC检测，就是对这个位置的论域中的每一个值都判断是不是有一个有效
支撑，因为约束只是简单的比较大小，所以其实另一方的所有不满足不等式的都不是有效支撑
                //将变量置为true的过程起始就是GAC检测删掉无效支撑的过程

                for (int i = 0; i < less_constraints.size(); i++)
                {
                    if (x == less_constraints[i].first.first && y ==
less_constraints[i].first.second)
                    {
                        maps[x][y].digits[nRow] = true;//先删去最大的候选
                        for (int j = maps[x][y].val; j >= 1; j--)
                        {
                            if (!maps[less_constraints[i].second.first]
[less_constraints[i].second.second].digits[j]) //若为false变为true
                            {
                                maps[less_constraints[i].second.first]
[less_constraints[i].second.second].digits[j] = true;

                                //catches.push_back({
{less_constraints[i].second.first, less_constraints[i].second.second}, j });
                            }
                        }
```

```
                    }

                    if (x == less_constraints[i].second.first && y ==
less_constraints[i].second.second) {
                        maps[x][y].digits[1] = true;
                        if (maps[x][y].val != 0) {
                            for (int j = maps[x][y].val; j <= nRow; j++) {
                                if (!maps[less_constraints[i].first.first]
[less_constraints[i].first.second].digits[j]) {
                                    maps[less_constraints[i].first.first]
[less_constraints[i].first.second].digits[j] = true;
                                    //catches.push_back({
{less_constraints[i].first.first, less_constraints[i].first.second}, j });
                                }
                            }
                        }


                    }
                }
            }
        }
    }

    void addConstraints(int x, int y, int x1, int y1) {
        less_constraints.push_back({ {x,  y},
                                    {x1, y1} });
    }

    //输出dom的不可取的数字个数，最大的最优先选择赋值
    void print_dom()
    {
        cout << "------------dom_num------------\n";
        //test to printout dom_num
        for (int i = 0; i < nRow; i++)
        {
            for (int j = 0; j < nColumn; j++)
            {
                cout << dom_num[i][j] << " ";
            }
            cout << endl;
        }
        cout << "-------------------------------\n";
    }

    void enqueueConstrains(int x, int y, queue<int>& relatedconstrains) {
        pair<int, int> temp = make_pair(x, y);
        for (int i = 0; i < less_constraints.size(); i++) {
            if (less_constraints[i].first == temp || less_constraints[i].second
== temp) {
                //GAC_constrains.push(less_constraints[i]);

                relatedconstrains.push(i);
            }
        }
    }

    //删除候选值,cache记录在哪个位置删除了哪个元素，方便回溯
```

```
    void check(int x, int y, vector<pair<pair<int, int>, int>>& catches) {
        //缓存，如果失败可能需要回溯
        //vector<pair<pair<int, int>, int>> catches;
        //纵向横向删除候选值：
        for (int i = 0; i < nRow; i++)
        {
            //如果本来就是false，那就没必要加入缓存了
            if (!maps[i][y].digits[maps[x][y].val])
            {
                maps[i][y].digits[maps[x][y].val] = true;
                //意思是在[i,y]格子的候选者中删除了maps[x][y].val候选者
                catches.push_back({ {i, y}, maps[x][y].val });
            }

            if (!maps[x][i].digits[maps[x][y].val])
            {
                maps[x][i].digits[maps[x][y].val] = true;
                catches.push_back({ {x, i}, maps[x][y].val });
            }
        }

    }

    //判断是否是终止状态
    bool Goal()
    {
        //this->show();
        for (int i = 0; i < nRow; i++)
        {
            for (int j = 0; j < nColumn; j++)
            {
                if (maps[i][j].val == 0)
                {
                    return false;
                }
            }
        }
        return true;
    }

    //MRV算法确定下一步的x和y
    void heuristicPick(int& x, int& y)
    {
        int maxi = 0, maxj = 0;
        //寻找最大dom_num的i,j
        for (int i = 0; i < nRow; i++)
        {
            for (int j = 0; j < nColumn; j++)
            {

                if (maps[i][j].val != 0)continue;//已经赋过值

                if (dom_num[i][j] > dom_num[maxi][maxj] || maps[maxi][maxj].val
!= 0)
                {
                    maxi = i;
                    maxj = j;
                    if (dom_num[maxi][maxj] == nRow - 1)//找到只有一个选项的
```

```
                {
                    //print_dom();
                    x = maxi;
                    y = maxj;
                    return;
                }
            }

        }
    }

    //print_dom();
    x = maxi;
    y = maxj;
    return;
}

//赋值临时dom_num，把第二个copy到第一个
void copy_dom(vector<vector<int>>& A, vector<vector<int>>& B)
{
    for (int i = 0; i < B.size(); i++)
    {
        for (int j = 0; j < B[0].size(); j++)
        {
            A[i][j] = B[i][j];
        }
    }
}

//计数一个节点的非法个数
void countDom(node n, int x, int y)
{
    dom_num[x][y] = 0;
    for (int i = 0; i < nRow; i++)
    {
        if (n.digits[i] == true)
        {
            dom_num[x][y]++;
        }
    }
}

/*
对在一个位置赋值，判断是否为可行的
*/
bool can_be_satisfied(int x, int y, int val)
{
    queue<int> related;

    enqueueConstrains(x, y, related);//向related入队所有相关约束

    bool can_satis = false;
    while (related.size()) //当related不为空继续检查
    {
        can_satis = false;
        int temp_i = related.front();
        constrain temp = less_constraints[temp_i];
        related.pop();
```

```
            //如果这个点是约束中较小的那一个
            if (x == temp.first.first && y == temp.first.second)
            {
                //那么我就对较大的那一个进行检查
                //如果较大的那一个还没有被赋值
                if (maps[temp.second.first][temp.second.second].val == 0)
                {
                    int support = evidence[temp_i].first[val];
                    if (maps[temp.second.first]
[temp.second.second].digits[support] == false) {
                        return true;
                    }
                    for (int i = nRow; i >= 1; i--)
                    {
                        if (!maps[temp.second.first]
[temp.second.second].digits[i]) //可以被赋值
                        {
                            //说明这个约束存在可满足状态
                            //TODO：可以记录下来这些可满足状态作为evidence
                            if (i > val)
                            {
                                can_satis = true;
                                evidence[temp_i].first[val] = i;
                                break;
                            }
                        }
                    }
                    if (!can_satis) return false;
                }
                else {//已经赋值直接检查
                    if (maps[temp.second.first][temp.second.second].val < val) {
                        return false;
                    }
                    else {
                        can_satis = true;
                    }
                }
            }
            else if (x == temp.second.first && y == temp.second.second)
            {//是约束中大的那一方
                if (maps[temp.first.first][temp.first.second].val == 0)
                {//较小的那一方还没有被赋值
                    int support = evidence[temp_i].second[val];
                    if (maps[temp.first.first]
[temp.first.second].digits[support] == false) {
                        can_satis = true;
                        continue;
                    }

                    for (int i = 1; i < nRow + 1; i++)
                    {
                        if (!maps[temp.first.first]
[temp.first.second].digits[i])
                        {
                            //说明这个约束存在可满足状态
                            if (i < val) {
                                can_satis = true;
```

```
                                evidence[temp_i].second[val] = i;
                                break;
                            }
                        }
                    }
                    if (!can_satis) return false;
                }
                else //已经被赋值，直接检查
                {
                    if (maps[temp.first.first][temp.first.second].val > val)
                    {
                        return false;
                    }
                    else {
                        can_satis = true;
                    }
                }
            }
        }
        return can_satis;
    }

    //行约束和列约束是我们已知的，如果一个值发生了变化那么一定要入队行列约束
    //所以可以不入队这两个约束，而是每次结束都检测
    void GACEnforce(vector<pair<pair<int, int>, int>>& catches) {
        //vector<pair<pair<int, int>, int>> catches;
        bool res = true;
        while (GAC_constrains.size())
        {
            int ci = GAC_constrains.front();
            constrain c = less_constraints[ci];//取一个约束

            GAC_constrains.pop();
            int cnt = catches.size();

            for (int i = 1; i < nRow + 1; i++)
            {
                if (!maps[c.first.first][c.first.second].digits[i])//小的那边没有
赋值
                {
                    res = can_be_satisfied(c.first.first, c.first.second, i);//
能否满足

                    if (!res) //不能满足
                    {
                        maps[c.first.first][c.first.second].digits[i] = true;//
标记为不许取i

                        //这一轮的GAC检测在这个位置上删掉了这个值
                        catches.push_back({ { c.first.first, c.first.second },i
});//catches加入
                    }
                }
            }
            //如果删掉了值，那么相关约束要重新入队
            if (catches.size() > cnt) enqueueConstrains(c.first.first,
c.first.second, GAC_constrains);
            cnt = catches.size();
```

```
            //大的那一方
            for (int i = 1; i < nRow + 1; i++)
            {
                if (!maps[c.second.first][c.second.second].digits[i]) {
                    bool res = can_be_satisfied(c.second.first, c.second.second,
i);
                    if (!res)
                    {
                        maps[c.second.first][c.second.second].digits[i] = true;
                        catches.push_back({ {c.second.first,c.second.second},i
});
                        enqueueConstrains(c.second.first, c.second.second,
GAC_constrains);
                    }
                }
            }
            if (catches.size() > cnt) enqueueConstrains(c.second.first,
c.second.second, GAC_constrains);
        }

        //return catches;
    }

    /*bool search(int x, int y)
    {
        //this->show();
        cnt++; //用来计数

        if (maps[x][y].val == 0) //当前位置还没有填
        {
            for (int i = 1; i < nRow + 1; i++)
            {
                //还在队列中没有被访问过
                if (!maps[x][y].digits[i])
                {
                    vector<pair<pair<int, int>, int>> catches;
                    maps[x][y].val = i;//赋值为i

                    //其他标记为不可赋值
                    for (int j = 0; j < nRow; j++)
                    {
                        if (j != i && !maps[x][y].digits[j])
                        {
                            maps[x][y].digits[j] = true;
                            catches.push_back({ {x,y},j });
                        }
                    }

                    //返回缓存，缓存存储了三个特征，横坐标，纵坐标，删除的值。恢复只需要把候
选队列中的值变为true即可。
                    //vector<pair<pair<int, int>, int>> catches = check(x, y);

                    check(x, y, catches);
                    enqueueConstrains(x, y, GAC_constrains);
                    GACEnforce(catches);

                    //已经赋值到了最后一个位置
```

```
                    if (x == nRow - 1 && y == nRow - 1)
                    {
                        return true;
                    }
                    //下一个位置
                    int next_x, next_y;
                    if (y != nRow - 1) {//右边
                        next_x = x;
                        next_y = y + 1;
                    }
                    else {//换行
                        next_x = x + 1;
                        next_y = 0;
                    }


                    if (search(next_x, next_y)) {//递归调用
                        return true;
                    }
                    //查找失败，需要回退到上一版本。这里回退比较简单，只需要设置true即可
                    pull_back(catches);
                    maps[x][y].val = 0;
                }
            }
        }
        else //当前位置已经填上，只需要跳过即可
        {
            if (x == nRow - 1 && y == nRow - 1) {
                return true;
            }
            int next_x, next_y;
            if (y != nRow - 1) {
                next_x = x;
                next_y = y + 1;
            }
            else {
                next_x = x + 1;
                next_y = 0;
            }


            if (search(next_x, next_y)) {
                return true;
            }
        }
        return false;
    }*/

    //搜索过程
    bool search(int x, int y)
    {
        cnt++;//search数目加一

        if (Goal())
        {
            cout << "-----Goal!!-----\n";
            return true;
        }
```

```
        else if (maps[x][y].val == 0)//没有赋值
        {


            for (int i = 1; i < nRow + 1; i++) //遍历数字
            {
                //还在队列中没有被访问过就赋值
                if (!maps[x][y].digits[i])
                {
                    vector<pair<pair<int, int>, int>> catches;
                    maps[x][y].val = i;

                    //cout << "assign " << x << " , " << y << " 为 " << i <<
endl;

                    //保存
                    copy_dom(copy_dom_num, dom_num);

                    for (int j = 0; j < nRow; j++)
                    {
                        if (j != i && !maps[x][y].digits[j])
                        {
                            maps[x][y].digits[j] = true;

                            catches.push_back({ {x,y},j });
                        }
                    }

                    //返回缓存，缓存存储了三个特征，横坐标，纵坐标，删除的值。恢复只需要把候
选队列中的值变为true即可。
                    //vector<pair<pair<int, int>, int>> catches = check(x, y);

                    //记录推理时间
                    inf_start = clock();

                    check(x, y, catches);
                    enqueueConstrains(x, y, GAC_constrains);

                    inf_end = clock();
                    total_inf_time += (double)(inf_end - inf_start) /
CLOCKS_PER_SEC;

                    GACEnforce(catches);

                    countDom(maps[x][y], x, y);
                    //print_dom();

                    //挑选下一个
                    int next_x, next_y;
                    heuristicPick(next_x, next_y);
                    //cout << "next_x= " << next_x << " next_y= " << next_y <<
endl;

                    if (search(next_x, next_y))
                    {
                        return true;
                    }
```

```
                    //查找失败，需要回退到上一版本。这里回退比较简单，只需要设置true即可
                    pull_back(catches);

                    //恢复
                    copy_dom(dom_num, copy_dom_num);


                    //cout << "恢复 " << x << " , " << y << endl;
                    maps[x][y].val = 0;//标记为没有赋值
                }
            }
        }
        else//已经赋值
        {
            int next_x, next_y;
            heuristicPick(next_x, next_y);


            if (search(next_x, next_y)) {
                return true;
            }
        }
        return false;
    }



    //倒退。根据缓存倒退上一版本
    void pull_back(vector<pair<pair<int, int>, int>> catches) {
        for (int i = 0; i < catches.size(); i++) {
            int x = catches[i].first.first;
            int y = catches[i].first.second;
            int digit = catches[i].second;
            maps[x][y].digits[digit] = false;
        }
    }

    void show() {
        for (int i = 0; i < nRow; i++) {
            for (int j = 0; j < nColumn; j++) {
                cout << maps[i][j].val << " ";
            }
            cout << endl;
        }
        cout << "======================" << endl;
    }

    void showconstrain() {
        for (int i = 0; i < less_constraints.size(); i++) {
            cout << less_constraints[i].first.first <<
less_constraints[i].first.second << less_constraints[i].second.first <<
less_constraints[i].second.second << endl;
        }
    }
};

int main(int argc, char* argv[]) {
    FutoshikiPuzzle* futoshikiPuzzle = new FutoshikiPuzzle();
```

```
    //初始化
    futoshikiPuzzle->initial(NUM_OF_TEST);
    //显示空表
    futoshikiPuzzle->show();

    //记录时间
    clock_t start, end;
    start = clock();
    //开始搜索
    futoshikiPuzzle->search(0, 0);
    end = clock();

    ////显示最终结果
    futoshikiPuzzle->show();

    cout << "-----test case " << NUM_OF_TEST << " ----- \n" << endl;
    cout << "GAC Time cost : " << (double)(end - start) / CLOCKS_PER_SEC << " s"
<< endl;
    cout << "Total nodes expanded: " << futoshikiPuzzle->cnt << endl;
    cout << "Total inference time: " << futoshikiPuzzle->total_inf_time <<" s "
<<endl;
    cout << "Average inference time: " << futoshikiPuzzle->total_inf_time/
futoshikiPuzzle->cnt << " s " << endl;
}
```

2.KRR

```
import re
import copy
from queue import Queue,LifoQueue,PriorityQueue
#就是匹配一下这两个字符串
def isSamePredicate(a, b):
    if a==b :
        return True
    if a==b[1:] or a[1:]==b:
        return True
    return False

#f g是两个子句，查找两个子句中可以用于消解的谓词下标
def findPredicateToResolution(f, g):
    #对f中的每一个formula
    for i in range(len(f)):
        #在j中查找判断
        for j in range(len(g)):
            if (f[i][0] == g[j][0][1:] and g[j][0][0] == '¬') or (f[i][0][1:] ==
g[j][0] and f[i][0][0]=='¬'):
                return i,j
    return -1,-1

def findDisagreement(f, g):
    #从零开始比就可以，因为传进来就已经把为此的部分截掉了
    for i in range(len(f)):
        if f[i]!=g[i]:
            return i
    #
    return -1
```

```
#b里面包含a
def variableInterm(a, b):
    ret = []
    index1 = b.find('(')
    index2 = b.find(')')
    if(index1!=-1 and index2!=-1):
        #res现在是括号里的所有的
        res = b[index1+1:index2]
        it = re.finditer('[(, ,]'+a+'[), ,]',b)
        #注意这里得到的下标是带两边的符号的，记得-1和+1
        for match in it:
            ret.append(match.span()[0]+1)

    #没匹配上就返回空列表,匹配上了就返回起始下标
    return ret

#这里只是一个字符串的替换函数
def multi_sub(oristring, substring, ibegin, iend):
    new = list(substring)
    ori = list(oristring)
    ori[ibegin:iend] = iter(new)
    return ''.join(ori)

#有两个formula:f ，还有一个替换sigma,现在要把sigma作用在f上
def substitude(f ,sigma):
    #不要让sigma做外层循环，让每一个公式的项做外层循环，这样实现的才是同时替换
    #因为替换后下一次就不会在考虑他了
    for i in range(1,len(f)):
        for sub in sigma:
            #如果匹配可以替换
            if f[i]==sub[0]:
                f[i] = sub[1]
            else:
                #否则找里面的是不是能有可以替换的
                index = variableInterm(sub[0],f[i])
                for ind in index:
                    f[i] = multi_sub(f[i],sub[1], ind, ind + len(sub[0]))




#做一个简单的trick，和prolog保持一致，如果小写字母开头就是常量，如果大写字母开头就是变量
#f和g表示两个公式
def MGU(f, g):
    sigma = []
    # 如果不这样的话，在循环里面操作每次都要创建匿名变量我觉得效率低开销大
    f_list = f[1:]
    g_list = g[1:]
    k = 0
    if len(f) != len(g) or (not isSamePredicate(f[0],g[0])):
        #表示两个公式不能合一
        #sigma.append(-1)
        return None
    #只要两个公式还不相等
    while f_list!=g_list:
        #找到第一个不等的位置，接下来就要考虑做替换
        i = findDisagreement(f_list,g_list)
```

```
        #如果不一样的这一项的首字母都是小写，那么就是说这两个都是项，无论如何都不能做合一
        if f_list[i][0].islower() and g_list[i][0].islower():
            return None
        #如果f的这个是变量，但是g里面的项包含了它
        elif f_list[i][0].isupper() and variableInterm(f_list[i],g_list[i])!=[]:
            return None
        # 如果g的这个是变量，但是f里面的项包含了它
        elif g_list[i][0].isupper() and variableInterm(g_list[i],f_list[i]) !=
[]:
            return None
        #至少有一个是变量而且不相互包含，可以替换合一
        else:
            k = k+1
            tempf = f_list[i]
            tempg = g_list[i]
            #优先考虑f的是变量
            if f_list[i][0].isupper():
                #在找合一的时候就已经替换过了
                for sub in sigma:
                    if sub[1]== tempf:
                        sub[1] = tempg
                    else:
                        index = variableInterm(tempf,sub[1])
                        for ind in index:
                            sub[1] = multi_sub(sub[1],tempg,ind,ind +
len(tempf))
                sigma.append([f_list[i], g_list[i]])
                for sub in sigma:
                    if sub[0]==sub[1]:
                        sigma.remove(sub)


                for j in range(len(f_list)):
                    if f_list[j] == tempf:
                        f_list[j] = tempg
                    else:
                        index = variableInterm(tempf, f_list[j])
                        for ind in index:
                            f_list[j] = multi_sub(f_list[j],tempg,ind,ind +
len(tempf))
                for j in range(len(g_list)):
                    if g_list[j] == tempf:
                        g_list[j] = tempg
                    else:
                        index = variableInterm(tempf, g_list[j])
                        for ind in index:
                            g_list[j] = multi_sub(g_list[j], tempg, ind, ind +
len(tempf))
            else:
                for sub in sigma:
                    if sub[1]== tempg:
                        sub[1] = tempf
                    else:
                        index = variableInterm(tempg,sub[1])
                        for ind in index:
                            sub[1] = multi_sub(sub[1], tempf, ind, ind +
len(tempg))
                for sub in sigma:
```

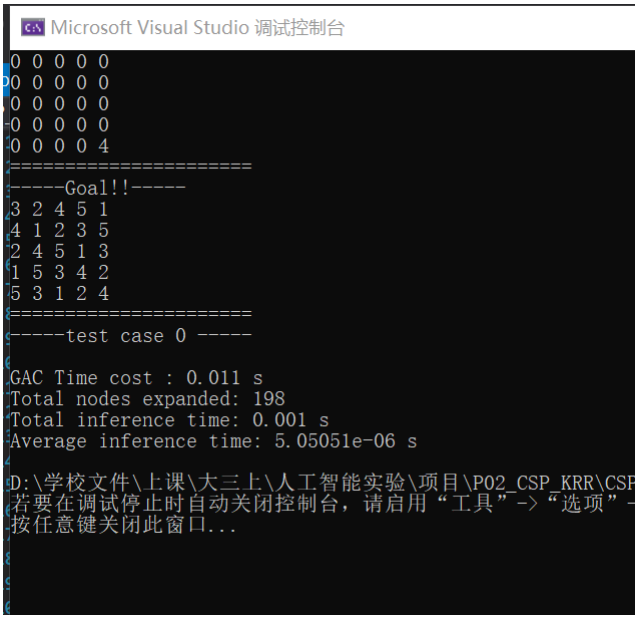
```
                    if sub[0]==sub[1]:
                        sigma.remove(sub)
                #含义就是用后面的替换前面的，就相当于省略了等号
                sigma.append([g_list[i],f_list[i]])
                tempf = f_list[i]
                tempg = g_list[i]
                for j in range(len(g_list)):
                    if g_list[j] == tempg:
                        g_list[j] = tempf
                    else:
                        index = variableInterm(tempg,g_list[j])
                        for ind in index:
                            g_list[j] = multi_sub(g_list[j], tempf, ind, ind +
len(tempg))

                for j in range(len(f_list)):
                    if f_list[j] == tempg:
                        f_list[j] = tempf
                    else:
                        index = variableInterm(tempg, f_list[j])
                        for ind in index:
                            f_list[j] = multi_sub(f_list[j], tempf, ind, ind +
len(tempg))
    return sigma



#消解两个子句,这里只是消解两个子句，后面再写一个函数消解所有的
#如果能消解，就把消解过程写进clauses,返回true；否则什么都不写，返回false
def resolution_two(f, g, clauses, ope):
    # Y.append(chr(97 + index))
    inf = clauses.index(f)
    ing = clauses.index(g)
    sigma = []
    tmpf = copy.deepcopy(f)
    tmpg = copy.deepcopy(g)
    hasre = False
    while(1):
        # 首先判断能不能消解，消解只有一种形式，同一谓词有非和无非
        indexf,indexg = findPredicateToResolution(tmpf,tmpg)
        if indexf==-1 and indexg==-1:
            break

        sigma = MGU(f[indexf],g[indexg])
        #说明存在合一
        if sigma != None:
            del(tmpf[indexf])
            del(tmpg[indexg])
            for i1 in range(len(tmpf)):
                substitude(tmpf[i1],sigma)
            for i1 in range(len(tmpg)):
                substitude(tmpg[i1],sigma)
            hasre = True
            newclause = tmpf + tmpg
            clauses.append(newclause)
            ope.append([inf + 1, chr(97 + indexf), ing + 1, chr(97 + indexg),
sigma, newclause])
        else:
```

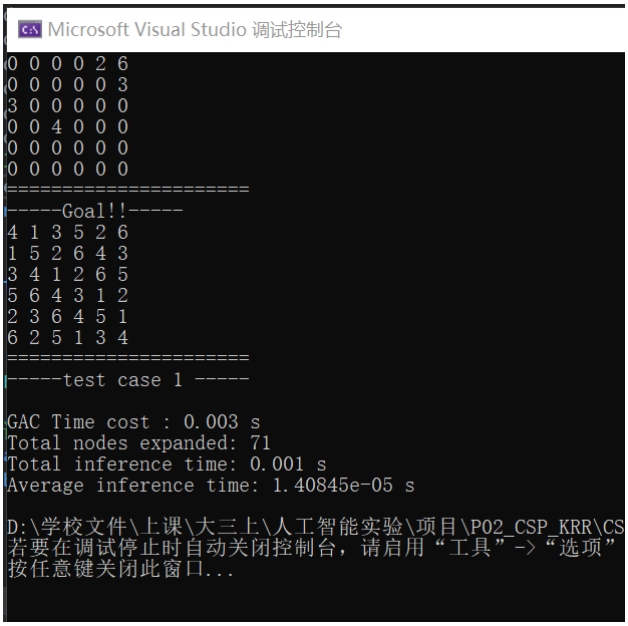
```
            break

def resolution_all(clauses,ope):
    #优先从最后一个开始归结
    find = False
    oneround = False
    index = len(clauses) - 2
    start = index + 1
    while(1):
        index = index + 1
        if index == start and oneround == True:
            index = last
        if index == len(clauses):
            oneround = True
            if last==index:
                find = False
                break
            last = index
            index = 0

        for i in range(index):
            resolution_two(clauses[index],clauses[i],clauses,ope)
            if clauses[len(clauses)-1] == []:
                find = True
                break
        if find==True:
            break
    return find,index

def findindex(clause_used,i,orinum):
    if i<=clause_used[0]:
        return i
    for j in range(len(clause_used)):
        if clause_used[j]==i:
            return j + orinum
    return i



num = 0
clauses = []
num=int(input())
for i in range(0, num):
    clause = []
    for item in re.findall(r'¬*[a-zA-Z]+\([a-zA-Z,\s]*\)', input()):
        items = re.findall(r'[¬a-zA-Z]+', item)
        clause.append(items)

    clauses.append(clause)
print(clauses)

# clauses = [[['On', 'aa', 'bb']], [['On', 'bb', 'cc']], [['Green', 'aa']],
[['¬Green', 'cc']], [['¬On', 'X', 'Y'], ['¬Green', 'X'], ['Green', 'Y']]]
# clauses = [[['GradStudent', 'sue']], [['¬GradStudent', 'X'], ['Student',
'X']], [['¬Student', 'X'], ['HardWorker', 'X']], [['¬HardWorker', 'sue']]]
```

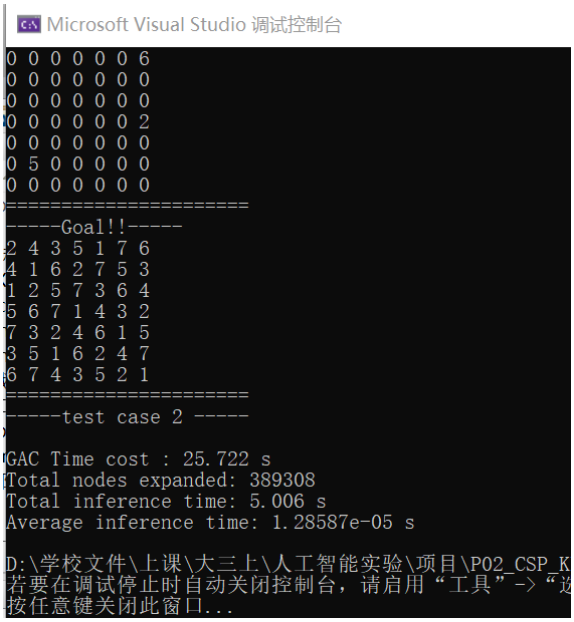
```
# clauses = [[['A', 'tony']], [['A', 'mike']], [['A', 'john']], [['L', 'tony',
'rain']], [['L', 'tony', 'snow']], [['¬A', 'X'], ['S', 'X'], ['C', 'X']],
[['¬C', 'Y'], ['¬L', 'Y', 'rain']], [['L', 'Z', 'snow'], ['¬S', 'Z']], [['¬L',
'tony', 'U'], ['¬L', 'mike', 'U']], [['L', 'tony', 'V'], ['L', 'mike', 'V']],
[['¬A', 'W'], ['¬C', 'W'], ['S', 'W']]]


ope = []
orinum = len(clauses)
find,index = resolution_all(clauses,ope)

#########################格式化输出###################################
clause_used = []
indexchain = Queue(maxsize=0)
index = len(ope)-1
clause_used.append(index + orinum)
indexchain.put(index)

while not indexchain.empty():
    index = indexchain.get()
    if ope[index][0] > orinum:
        clause_used.append(ope[index][0]-1)
        indexchain.put(ope[index][0] - orinum - 1)
    if ope[index][2] > orinum:
        clause_used.append(ope[index][2]-1)
        indexchain.put(ope[index][2] - orinum - 1)

clause_used = list(set(clause_used))
clause_used.sort()
for i in range(orinum):
    print(clauses[i])


for i in range(len(clause_used)):
    i1 = 1 + findindex(clause_used,ope[clause_used[i]-orinum][0]-1,orinum)
    i2 = 1 + findindex(clause_used,ope[clause_used[i]-orinum][2]-1,orinum)
    print("R[", i1, ope[clause_used[i] - orinum][1], ",", i2,
          ope[clause_used[i] - orinum][3], "]", ope[clause_used[i] - orinum][4],
ope[clause_used[i] - orinum][5])

print(find)
```

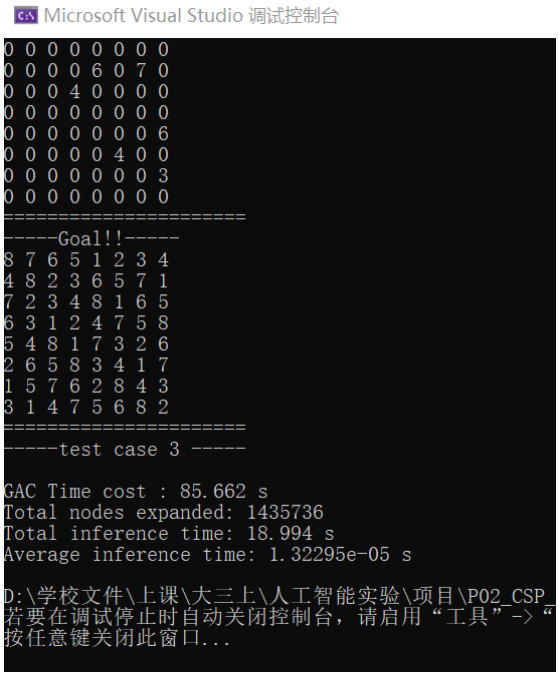
3.结果展示

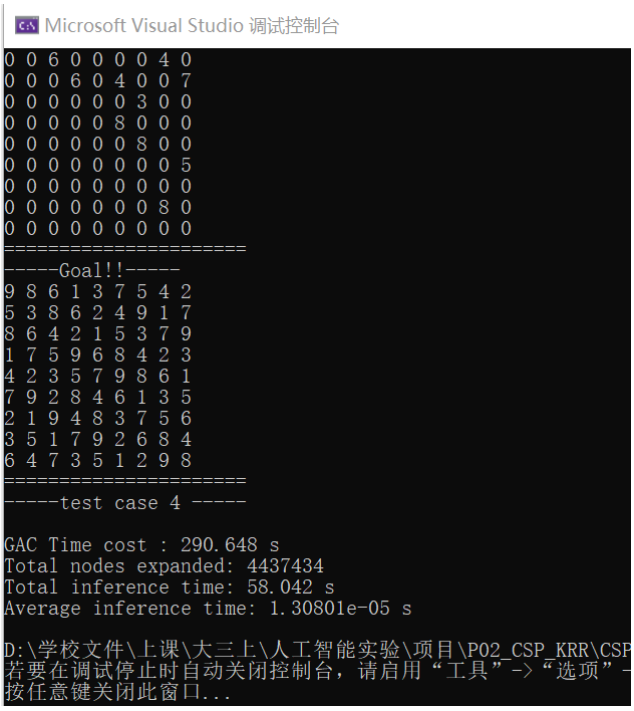
1.CSP


Microsoft Visual Studio 调试控制台
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 4
=====================
-----Goal!!-----
3 2 4 5 1
4 1 2 3 5
2 4 5 1 3
1 5 3 4 2
5 3 1 2 4
=====================
-----test case 0 -----
GAC Time cost : 0.011 s
Total nodes expanded: 198
Total inference time: 0.001 s
Average inference time: 5.05051e-06 s
D:\学校文件\上课\大三上\人工智能实验\项目\P02_CSP_KRR\CSP
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”-
按任意键关闭此窗口...


Microsoft Visual Studio 调试控制台
0 0 0 0 2 6
0 0 0 0 0 3
3 0 0 0 0 0
0 0 4 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
=====================
-----Goal!!-----
4 1 3 5 2 6
1 5 2 6 4 3
3 4 1 2 6 5
5 6 4 3 1 2
2 3 6 4 5 1
6 2 5 1 3 4
=====================
-----test case 1 -----
GAC Time cost : 0.003 s
Total nodes expanded: 71
Total inference time: 0.001 s
Average inference time: 1.40845e-05 s
D:\学校文件\上课\大三上\人工智能实验\项目\P02_CSP_KRR\CS
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”
按任意键关闭此窗口...


Microsoft Visual Studio 调试控制台
0 0 0 0 0 0 6
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 2
0 0 0 0 0 0 0
0 5 0 0 0 0 0
0 0 0 0 0 0 0
=====================
-----Goal!!-----
2 4 3 5 1 7 6
4 1 6 2 7 5 3
1 2 5 7 3 6 4
5 6 7 1 4 3 2
7 3 2 4 6 1 5
3 5 1 6 2 4 7
6 7 4 3 5 2 1
=====================
-----test case 2 -----
GAC Time cost : 25.722 s
Total nodes expanded: 389308
Total inference time: 5.006 s
Average inference time: 1.28587e-05 s
D:\学校文件\上课\大三上\人工智能实验\项目\P02_CSP_K
若要在调试停止时自动关闭控制台，请启用“工具”->“选
按任意键关闭此窗口...





2.KRR

test case1 : hardworker(sue)

输入文件

```
4
GradStudent(sue)
(¬GradStudent(X), Student(X))
(¬Student(X), HardWorker(X))
¬HardWorker(sue)
```

输出文件

```
[[['GradStudent', 'sue']], [['¬GradStudent', 'X'], ['Student', 'X']],
[['¬Student', 'X'], ['HardWorker', 'X']], [['¬HardWorker', 'sue']]]
1 [['GradStudent', 'sue']]
2 [['¬GradStudent', 'X'], ['Student', 'X']]
3 [['¬Student', 'X'], ['HardWorker', 'X']]
4 [['¬HardWorker', 'sue']]
5 R[ 4 a , 3 b ] [['X', 'sue']] [['¬Student', 'sue']]
6 R[ 5 a , 2 b ] [['X', 'sue']] [['¬GradStudent', 'sue']]
7 R[ 6 a , 1 a ] [] []
True
```

结果截图

```
E:\AIEXPYTHON\P01\P02\Scripts\python.exe E:/AIEXPYTHON/P02/resolution.py
4
GradStudent(sue)
(¬GradStudent(X), Student(X))
(¬Student(X), HardWorker(X))
¬HardWorker(sue)
[[['GradStudent', 'sue']], [['¬GradStudent', 'X'], ['Student', 'X']], [['¬Student', 'X'], ['HardWorker', 'X']], [['¬HardWorker', 'sue']]]
1 [['GradStudent', 'sue']]
2 [['¬GradStudent', 'X'], ['Student', 'X']]
3 [['¬Student', 'X'], ['HardWorker', 'X']]
4 [['¬HardWorker', 'sue']]
5 R[ 4 a , 3 b ] [['X', 'sue']] [['¬Student', 'sue']]
6 R[ 5 a , 2 b ] [['X', 'sue']] [['¬GradStudent', 'sue']]
7 R[ 6 a , 1 a ] [] []
True
```

test case2 : 3-blocks

输入文件

```
5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(X,Y), ¬Green(X), Green(Y))
```

输出文件

```
[[['On', 'aa', 'bb']], [['On', 'bb', 'cc']], [['Green', 'aa']], [['¬Green',
'cc']], [['¬On', 'X', 'Y'], ['¬Green', 'X'], ['Green', 'Y']]]
1 [['On', 'aa', 'bb']]
2 [['On', 'bb', 'cc']]
3 [['Green', 'aa']]
4 [['¬Green', 'cc']]
5 [['¬On', 'X', 'Y'], ['¬Green', 'X'], ['Green', 'Y']]
6 R[ 5 a , 1 a ] [['X', 'aa'], ['Y', 'bb']] [['¬Green', 'aa'], ['Green', 'bb']]
7 R[ 5 a , 2 a ] [['X', 'bb'], ['Y', 'cc']] [['¬Green', 'bb'], ['Green', 'cc']]
8 R[ 6 a , 3 a ] [] [['Green', 'bb']]
9 R[ 7 b , 4 a ] [] [['¬Green', 'bb']]
10 R[ 9 a , 8 a ] [] []
True
```

结果截图

```
E:\AIEXPYTHON\P01\P02\Scripts\python.exe E:/AIEXPYTHON/P02/resolution.py
5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(X,Y), ¬Green(X), Green(Y))
[[['On', 'aa', 'bb']], [['On', 'bb', 'cc']], [['Green', 'aa']], [['¬Green', 'cc']], [['¬On', 'X', 'Y'], ['¬Green', 'X'], ['Green', 'Y']]]
1 [['On', 'aa', 'bb']]
2 [['On', 'bb', 'cc']]
3 [['Green', 'aa']]
4 [['¬Green', 'cc']]
5 [['¬On', 'X', 'Y'], ['¬Green', 'X'], ['Green', 'Y']]
6 R[ 5 a , 1 a ] [['X', 'aa'], ['Y', 'bb']] [['¬Green', 'aa'], ['Green', 'bb']]
7 R[ 5 a , 2 a ] [['X', 'bb'], ['Y', 'cc']] [['¬Green', 'bb'], ['Green', 'cc']]
8 R[ 6 a , 3 a ] [] [['Green', 'bb']]
9 R[ 7 b , 4 a ] [] [['¬Green', 'bb']]
10 R[ 9 a , 8 a ] [] []
True
```

test case3 : Alpine Club

输入文件

```
11
A(tony)
A(mike)
A(john)
L(tony, rain)
L(tony, snow)
(¬A(X), S(X), C(X))
(¬C(Y), ¬L(Y, rain))
(L(Z, snow), ¬S(Z))
(¬L(tony, U), ¬L(mike, U))
(L(tony, V), L(mike, V))
(¬A(W), ¬C(W), S(W))
```

输出文件

```
1 [['A', 'tony']]
2 [['A', 'mike']]
3 [['A', 'john']]
4 [['L', 'tony', 'rain']]
5 [['L', 'tony', 'snow']]
6 [['¬A', 'X'], ['S', 'X'], ['C', 'X']]
7 [['¬C', 'Y'], ['¬L', 'Y', 'rain']]
8 [['L', 'Z', 'snow'], ['¬S', 'Z']]
9 [['¬L', 'tony', 'U'], ['¬L', 'mike', 'U']]
10 [['L', 'tony', 'V'], ['L', 'mike', 'V']]
11 [['¬A', 'W'], ['¬C', 'W'], ['S', 'W']]
12 R[ 11 a , 1 a ] [['W', 'tony']] [['¬C', 'tony'], ['S', 'tony']]
13 R[ 12 a , 6 c ] [['X', 'tony']] [['S', 'tony'], ['¬A', 'tony'], ['S',
'tony']]
14 R[ 13 b , 1 a ] [] [['S', 'tony'], ['S', 'tony']]
15 R[ 14 a , 8 b ] [['Z', 'tony']] [['S', 'tony'], ['L', 'tony', 'snow']]
16 R[ 15 a , 8 b ] [['Z', 'tony']] [['L', 'tony', 'snow'], ['L', 'tony',
'snow']]
17 R[ 16 a , 9 a ] [['U', 'snow']] []
True
```

结果截图

```
E:\AIEXPYTHON\P01\P02\Scripts\python.exe E:/AIEXPYTHON/P02/resolution.py
11
A(tony)
A(mike)
A(john)
L(tony, rain)
L(tony, snow)
(¬A(X), S(X), C(X))
(¬C(Y), ¬L(Y, rain))
(L(Z, snow), ¬S(Z))
(¬L(tony, U), ¬L(mike, U))
(L(tony, V), L(mike, V))
(¬A(W), ¬C(W), S(W))
[[['A', 'tony']], [['A', 'mike']], [['A', 'john']], [['L', 'tony', 'rain']], [['L', 'tony', 'snow']], [['¬A', 'X'], ['S', 'X'], ['C', 'X']], [['¬C',
1 [['A', 'tony']]
2 [['A', 'mike']]
3 [['A', 'john']]
4 [['L', 'tony', 'rain']]
5 [['L', 'tony', 'snow']]
6 [['¬A', 'X'], ['S', 'X'], ['C', 'X']]
7 [['¬C', 'Y'], ['¬L', 'Y', 'rain']]
8 [['L', 'Z', 'snow'], ['¬S', 'Z']]
9 [['¬L', 'tony', 'U'], ['¬L', 'mike', 'U']]
10 [['L', 'tony', 'V'], ['L', 'mike', 'V']]
11 [['¬A', 'W'], ['¬C', 'W'], ['S', 'W']]
12 R[ 11 a , 1 a ] [['W', 'tony']] [['¬C', 'tony'], ['S', 'tony']]
13 R[ 12 a , 6 c ] [['X', 'tony']] [['S', 'tony'], ['¬A', 'tony'], ['S', 'tony']]
14 R[ 13 b , 1 a ] [] [['S', 'tony'], ['S', 'tony']]
15 R[ 14 a , 8 b ] [['Z', 'tony']] [['S', 'tony'], ['L', 'tony', 'snow']]
16 R[ 15 a , 8 b ] [['Z', 'tony']] [['L', 'tony', 'snow'], ['L', 'tony', 'snow']]
17 R[ 16 a , 9 a ] [['U', 'snow']] []
True
```

4.Experimental experience

CSP

本次实验使用C++来完成，从最初的FC代码开始修改，经过原始的GAC，到加入支撑集，再到MRV算法寻找最小值域赋值的逐步优化，完成了本次的实验。

- 数据结构的构思问题：

  为了进行优化我们考虑了很多种数据结构，尤其是在存储evidence支撑集的部分，本来选用了vector< set < pair < int , int >>>保证每次存下来的支撑是不同的，但是set操作消耗时间太大，而且每次查找的时间也不足以用换来的节约的时间来弥补，因此我们选用了vector< pair < vector , vector>>来存储支撑集，这样在对应值域数字的位置就可以很方便的查找到他的支撑，节约了很多时间。

- 问题规模小的时候存放支撑集并没有怎么加速：

  这个测试的时候有些百思不得其解，以为有了支撑集就好像有了一个目录快查一样，但是后来思考讨论之后，明白了其实我们本身的数据（Futoshiki棋盘）并不是很大，遍历一遍来搜索支撑的时间也不算很长，也就是遍历几个位置而已，和我们额外的那个证据支撑集的查找O(1)时间差不了太多。因此只有在问题规模大的时候才有比较好的判断赋值是否合法的加速效果。

- 回溯时的问题：

  因为深度优先搜索在搜不下去的时候需要回溯，这个时候什么要恢复到以前的样子？

  1.其一是值域的修改，我们选择了catches这个向量来保存之前进行过什么值域的修改，当需要回溯的时候只需要把对应的digits修改回可赋值状态就可

  2.其二是对启发式选择下一个赋值位置的辅助数据结构——dom_num（存放每个maps位置非法的数字的个数）来说，在进行一次赋值之后应该把当前的dom_num拷贝下来，再进行下一轮的深度优先搜索，等到需要回溯的时候反过来拷贝回来。

  就可以保证在回溯的时候恢复到原来的样子。

在实验中我们遇到了很多问题也学会很多新的知识，有了对CSP问题解决的更深层的理解。

KRR

本次实验使用python完成，无论是在实验思路上和编程语言的运用上都遇到了一些问题，在解决问题的过程中也学到了很多的知识。下面我们总结一下遇到的问题。

- 首先在实现MGU的算法的部分，需要解决变量和项的问题。在理论推导的时候我们可以清晰地知到符号的含义，但是让计算机理解变量和项却并不轻松，这里我们沿用了`prolog`的方式让首字母大写的符号表示变量，否则表示项。
- 在编写MGU算法的时候，不仅仅要考虑变量本身，还要考虑这个项是否是一个函数的返回值，如果函数的参数包含变量，那么还要做额外的判断，所以在编写MGU部分的代码的时候一定要注意对函数参数部分是否包含目标变量的判断。
- 在编写归结部分的代码的时候，我们需要考虑同时替换和替换的复合。在我们寻找MGU的时候，每一次迭代的过程实际上是替换的复合，这个时候要求我们将第二个替换作用到第一个替换的每一个上面，也要讲新的替换作用到上一步的中间子句的结果上面。对于这种情况可以使用每一个替换项做外重循环，被替换的公式做内重循环。但是在找到MGU后对子句进行合一替换的时候，这个时候执行的是同时替换，需要注意此时要让被替换的公式的每个参数做外重循环，找到的MGU的每一个替换项做内重循环，只有这样才能实现同时替换，而不是替换的复合。
- 对于python的list赋值问题在本次实验中遇到了一些问题，下面重点讲解一些python的list赋值问题。python对list的赋值方式有很多，包括下面几种

  ```
  list1 = list2
  list1 = list2[:]
  list1 = list(list2)
  list1 = list2.copy()
  list1 = copy.deepcopy(list2)
  ```

  第一种赋值方式是最典型的浅拷贝，实际上我们并没有创建新的list，而只是定义了一个新的指针指向这个空间。第二，三，四种拷贝方式在一定意义上是深拷贝，因为输出`list1`和`list2`的地址确实是不同的，但是如果我们查看`list1[1]`和`list2[1]`的地址，会发现他们的数据还是一样的，也就是说在这种情况，仅仅是将头指针进行了深拷贝，而后面的元素没有变化，如果将`list1[1]`更改，那么`list2[1]`也会被更改。而最后一种方式才是真正意义上的深拷贝，对于每一个节点都开辟了一个新的地址空间并赋值内容，这样得到的`list1`和`list2`完全独立，对其中一个无论进行何种操作，都不会影响另一个。