P02 CSP&KRR

学号	姓名	专业(方向)
18340013,18340014	陈琮昊，陈嘉宁	计算机科学与技术（人工智能与大数据）

1.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`算法是检查每个变量是否满足边一致性。我们把`CSP`问题的每个约束看作一条边，边一致性就是说，对于V_i内的任意取值，都可以保证每个$V_j(j \neq i)$内至少有一个取值能够使得约束成立（即存在性问题）。如果当$V_i = d$时，其他变量无论怎样取值都不能保证满足约束，那么这个d是边不一致的，就要从V_i的论域内删除，因为这个值不可能帮助我们找到`CSP`问题的解。这就是`GAC`的主要思想。

   `FC`和`GAC`的区别在于：`GAC`是判断所有剩余的待赋值变量是否满足约束，而`FC`只判断下一个变量的可能取值集合是否为空，当下一个变量出现`DWO`时才会进行回溯。

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?

   在删除一个变量的取值之后，可能只是解决了眼前的不一致，但有可能会引发进一步的不一致，因此需要将与V相关且当前不在队列里约束条件加入队列进行检测。我觉得可以用空间来换时间：构建一个表和一个计数器，表内存放的是一个`tuple`：(V, ν, Z, z)，其含义为：当$V = \nu$时，满足约束C的另一个变量Z的可能取值z(当然$V, Z \in scope(C)$)；计数器内存放的是上面所说可能取值的个数，即$count = (V, \nu, Z)$。当V内的一个取值d被删除掉时，查询表找到与$\nu = d$时相关的`tuple`，对应的计数器就-1，直到出现某一个计数器为0则移除(因为此时意味着有一个变量出现了`DWO`)。这样就不用再进行一遍约束检测，但显然空间开销很大!

3. Use the `GAC` algorithm to implement a Futoshiki solver by **C++** or **Python**.

   C++实现`GAC`算法的代码见附录，也可见文件夹内的`Futoshiki_gac.cpp`文件。

4. Explain any ideas you use to speed up the implementation.

   依照`GAC`算法的思路，在每次赋值后都需要将与所赋值位置相关的所有约束（包括行约束，列约束和小于约束）作用于同行、同列的各点，从而更新各点的值域。但是更新值域后的下一次取值则是选取最近的未赋值位置。受到`FC`算法中`MRV`的启发，我认为可以在`GAC`算法的选取下一个赋值点的过程中加入一个`MRV`过程，实现方法如下：

```
else if (constraint.type > 0) {
            int i = constraint.type - 1;
            for (int j = 0; j < size; j++) {
                if (domains[i][j].size() == 1) {
                    int value = *domains[i][j].begin();
                    for (int j1 = 0; j1 < size; j1++) {
                        if (j1 != j) {
                            bool did_erased = domains[i][j1].erase(value);
                            if (did_erased) {
                                if (domains[i][j1].size() == 0) return
DOMAINS();  // DWO
```

```
                                push_cons(make_pair(i, j1), gac_queue);
                            }
                        }
                    }
                }
            }
        }
        // 列约束
        else {
            int j = -constraint.type - 1;
            for (int i = 0; i < size; i++) {
                if (domains[i][j].size() == 1) {
                    int value = *domains[i][j].begin();
                    for (int i1 = 0; i1 < size; i1++) {
                        if (i1 != i) {
                            bool did_erased = domains[i1][j].erase(value);
                            if (did_erased) {
                                if (domains[i1][j].size() == 0) return
DOMAINS();  // DWO
                                push_cons(make_pair(i1, j), gac_queue);
                            }
                        }
                    }
                }
            }
        }
```

另一方面，在`GACEnforce`的过程中需要判断所加入的限制是否与队列中原有限制重复。我们在实现这一过程是采用了find函数有效地避免了多重循环，提高了效率。

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.

   注：为了与代码保持一致，将`datai.txt`更改为两部分，分别为`constraintsi.txt`（约束文件）和`puzzlei.txt`（题目文件），运行测例时只需修改main函数里的文件名即可。

   `GAC`的运行结果截图如下：

   Test Case1:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_gac.cpp -o
 Futoshiki_gac } ; if ($?) { .\Futoshiki_gac }
-----Initialization-----
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
------Solution------
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
-----------------------Outcoming-------------------------
Run time is in GAC :1.961ms.
Number of nodes GAC : 25.
Average time for constraint propagation : 0.012968ms.
-----------------------------------------------------------
```

Test Case2:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_gac.cpp -o
 Futoshiki_gac } ; if ($?) { .\Futoshiki_gac }
-----Initialization-----
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
-------Solution------
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
----------------------Outcoming-------------------------
Run time is in GAC :3.1239ms.
Number of nodes GAC : 31.
Average time for constraint propagation : 0.0244516ms.
-------------------------------------------------------
```

Test Case3:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_gac.cpp -o
 Futoshiki_gac } ; if ($?) { .\Futoshiki_gac }
-----Initialization-----
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
-------Solution------
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
----------------------Outcoming-------------------------
Run time is in GAC :20.5248ms.
Number of nodes GAC : 112.
Average time for constraint propagation : 0.0240045ms.
-------------------------------------------------------
```

Test Case4:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_gac.cpp -o
 Futoshiki_gac } ; if ($?) { .\Futoshiki_gac }
-----Initialization-----
0 0 0 0 0 0 0 0
0 0 0 0 6 0 7 0
0 0 0 4 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 6
0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0
-------Solution------
8 7 6 5 1 2 3 4
4 8 2 3 6 5 7 1
7 2 3 4 8 1 6 5
6 3 1 2 4 7 5 8
5 4 8 1 7 3 2 6
2 6 5 8 3 4 1 7
1 5 7 6 2 8 4 3
3 1 4 7 5 6 8 2
----------------------Outcoming-------------------------
Run time is in GAC :117.456ms.
Number of nodes GAC : 412.
Average time for constraint propagation : 0.018918ms.
-------------------------------------------------------
```

Test Case5:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_gac.cpp -o
 Futoshiki_gac } ; if ($?) { .\Futoshiki_gac }
-----Initialization-----
0 0 6 0 0 0 0 4 0
0 0 0 6 0 4 0 0 7
0 0 0 0 0 0 3 0 0
0 0 0 0 0 8 0 0 0
0 0 0 0 0 0 8 0 0
0 0 0 0 0 0 0 0 5
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 8 0
0 0 0 0 0 0 0 0 0
-------Solution------
9 8 6 1 3 7 5 4 2
5 3 8 6 2 4 9 1 7
8 6 4 2 1 5 3 7 9
1 7 5 9 6 8 4 2 3
4 2 3 5 7 9 8 6 1
7 9 2 8 4 6 1 3 5
2 1 9 4 8 3 7 5 6
3 5 1 7 9 2 6 8 4
6 4 7 3 5 1 2 9 8
----------------------Outcoming-------------------------
Run time is in GAC :1598.76ms.
Number of nodes GAC : 5608.
Average time for constraint propagation : 0.0278508ms.
-------------------------------------------------------
```

`FC`的结果截图如下：

Test Case1:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_fc.cpp -o
Futoshiki_fc } ; if ($?) { .\Futoshiki_fc }
-----Initialization-----
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
------Solution------
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
----------------------Outcoming-------------------------
Run time is in FC :2.7607ms.
Number of nodes in FC : 84.
Average time for constraint propagation : 0.00171429ms.
-------------------------------------------------------------
```

Test Case2:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_fc.cpp -o
Futoshiki_fc } ; if ($?) { .\Futoshiki_fc }
-----Initialization-----
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
------Solution------
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
----------------------Outcoming-------------------------
Run time is in FC :4.6621ms.
Number of nodes in FC : 49.
Average time for constraint propagation : 0.00374898ms.
-------------------------------------------------------------
```

Test Case3:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_fc.cpp -o
Futoshiki_fc } ; if ($?) { .\Futoshiki_fc }
-----Initialization-----
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
------Solution------
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
----------------------Outcoming-------------------------
Run time is in FC :118.078ms.
Number of nodes in FC : 1609.
Average time for constraint propagation : 0.00310833ms.
-------------------------------------------------------------
```

Test Case4:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_fc.cpp -o
Futoshiki_fc } ; if ($?) { .\Futoshiki_fc }
-----Initialization-----
0 0 0 0 0 0 0 0
0 0 0 0 6 0 7 0
0 0 0 4 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 6
0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0
-------Solution------
8 7 6 5 1 2 3 4
4 8 2 3 6 5 7 1
7 2 3 4 8 1 6 5
6 3 1 2 4 7 5 8
5 4 8 1 7 3 2 6
2 6 5 8 3 4 1 7
1 5 7 6 2 8 4 3
3 1 4 7 5 6 8 2
----------------------Outcoming-------------------------
Run time is in FC :168.532ms.
Number of nodes in FC : 1849.
Average time for constraint propagation : 0.00291357ms.
-------------------------------------------------------------
```

Test Case5:

```
PS C:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014> cd "c:\Users\czh\Desktop\AI\P02_CSP&KRR\P02_18340013,18340014\" ; if ($?) { g++ Futoshiki_fc.cpp -o
Futoshiki_fc } ; if ($?) { .\Futoshiki_fc }
-----Initialization-----
0 0 6 0 0 0 0 4 0
0 0 0 6 0 4 0 0 7
0 0 0 0 0 0 3 0 0
0 0 0 0 0 8 0 0 0
0 0 0 0 0 0 8 0 0
0 0 0 0 0 0 0 0 5
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 8 0
0 0 0 0 0 0 0 0 0
-------Solution------
9 8 6 1 3 7 5 4 2
5 3 8 6 2 4 9 1 7
8 6 4 2 1 5 3 7 9
1 7 5 9 6 8 4 2 3
4 2 3 5 7 9 8 6 1
7 9 2 8 4 6 1 3 5
2 1 9 4 8 3 7 5 6
3 5 1 7 9 2 6 8 4
6 4 7 3 5 1 2 9 8
-----------------------Outcoming-------------------------
Run time is in FC :26795.6ms.
Number of nodes in FC : 272475.
Average time for constraint propagation : 0.00293915ms.
-----------------------------------------------------------
```

经检验，两种算法得到的结果与文档内所给结果完全一致。

6. Run the `FC` algorithm and the `GAC` algorithm 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.

   结果在前面的截图中已经显示，现填入表格内：

Test Case Number	Method	Time Used	Nodes Searched	Time Per Node
1	FC GAC	2.76ms 1.96ms	84 25	0.00171ms 0.01297ms
2	FC GAC	4.66ms 3.12ms	49 31	0.00375ms 0.02445ms
3	FC GAC	118.08ms 20.52ms	1609 112	0.00311ms 0.02400ms
4	FC GAC	168.53ms 117.46ms	1849 412	0.00291ms 0.01892ms
5	FC GAC	26795.60ms 1598.76ms	272475 5608	0.00294ms 0.02785ms

观察结果我们可以发现，`GAC`的性能与`FC`相比提升了许多，每一个测例下`GAC`算法都是运行时间较短的那个。尤其是随着规模越来越大，这种性能提升更加明显。从扩展节点数上也能看出来`GAC`明显要比`FC`少很多，到9×9时`FC`扩展的节点数几乎是`GAC`的48倍！这也更能体现出`GAC`的性能强大。不过观察“Average Inference Time Per Node”一栏可以发现`FC`要比`GAC`所用的时间要短。这也不难理解，因为`GAC`在对约束条件进行判断以及约束延伸这一步要比`FC`复杂，因此`GAC`要用的时间稍长，但是换来的是搜索节点数的大量减少，可以简单的说单步是用“时间换空间”。但最终从整个算法的运行时间来看还是`GAC`要快，毕竟搜索节点数的大幅减少带来的影响更大。

2.Resolution

1. Implement the `MGU` algorithm.

   `MGU`算法主要实现的是归结中的合一过程，即是将形如

$$A(tony)$$
$$\neg A(x), S(x), C(x)$$

的两个子句合一为形如$S(tony), C(tony)$的一个子句。由于在形式化存储子句时我们使用了`vector<vector<vector<string>>>`的结构，导致将子句作为参数传入`MGU`函数变得较为复杂，因此，在我们设计的`MGU`函数中，传入的参数为(int `x1`, int `y1`, int `x2`, int `y2`, int `size`)，其意义分别为： `x1` - 用于合一的第一个子句的索引 `y1` - 用于合一的第一个子句中待合一的谓词语句，在上述的例子中即是A(tony) `x2` - 用于合一的第二个子句的索引 `y2` - 用于合一的第二个子句中待合一的谓词语句，在上述的例子中即是~A(x) `size` - 待合一的谓词语句的参数个数+2（+2是形式化存储的格式所致） 在`MGU`算法内，首先需要遍历两个待合一的谓词语句的参数并一一对应地进行比较，根据结果来更新一个结构为`vector<pair<string, pair<int, string>>>`，该结构用于存储<待更新变量，所在位置，目标量>，在`MGU`算法的后面会根据这个结果来变换子句中的变量。 若两谓词语句中的参数均为变量（记为`x`，`y`），则将元组<`x`，参数位置索引，`y`>存入上述`vector`中；若两谓词语句中的参数一个为变量（记为`x`）另一个为常量（记为`A`），则存入<`x`，参数位置索引，`A`>，在上述例子中即为<`x`，2，`tony`>；若均为常量，相同则存入，不相同则清空该`vector`并返回空值。 接着从两个子句中删除待合一的两个谓词语句，并根据`vector<pair<string, pair<int, string>>>`来修改两个子句的剩余内容，过程如下： 对<`x`，`i`，`y`>，遍历两子句中的参数，若发现`x`则将其替换为`y`。所有元组处理完毕后，将两子句合并作为一个新子句。新子句需要进行内部检测，将字句内重复或可合一的部分删除。 若最终得到空子句，则归结完毕。

实现`MGU`函数的代码如下：

```
vector<pair<string, pair<int, string>>> MGU(int x1, int y1, int x2, int y2, int
size){
    vector<pair<string, pair<int, string>>> toBeUpdate;
    vector<pair<string, pair<int, string>>> nonUpdate;
    for(int i = 2; i < size; i ++){
        // 判断参数构成
        if(!(whether_const(clauses[x1][y1][i]) && whether_const(clauses[x2][y2]
[i]))){
            if(!whether_const(clauses[x1][y1][i])){
                toBeUpdate.push_back(make_pair(clauses[x1][y1][i], make_pair(i,
clauses[x2][y2][i])));
            }
            else{
                toBeUpdate.push_back(make_pair(clauses[x2][y2][i], make_pair(i,
clauses[x1][y1][i])));
            }
        }
        else{       // 都是常量
            if(clauses[x1][y1][i] == clauses[x2][y2][i]){
                toBeUpdate.push_back(make_pair(clauses[x1][y1][i], make_pair(i,
clauses[x2][y2][i])));
            }
            else{
                 return nonUpdate;  // 不相等，返回空
            }
        }
    }
    vector<vector<string>> ivtem1(clauses[x1]), ivtem2(clauses[x2]); // 两个待合一
子句的副本
    if(toBeUpdate.size()){
        ivtem1.erase(ivtem1.begin() + y1);  // 删除待合一的谓词语句
        ivtem2.erase(ivtem2.begin() + y2);
        for(int i = 0; i < toBeUpdate.size(); i ++){    // 根据toBeUpdate更新剩余部
分
            for(int v = 0; v < ivtem1.size(); v ++){
```

```
                for(int j = 2; j < size; j ++){
                    if(ivtem1[v][j] == toBeUpdate[i].first){
                        ivtem1[v][j] = toBeUpdate[i].second.second;
                    }
                }
            }
            for(int v = 0; v < ivtem2.size(); v ++){
                for(int j = 2; j < size; j ++){
                    if(ivtem2[v][j] == toBeUpdate[i].first){
                        ivtem2[v][j] = toBeUpdate[i].second.second;
                    }
                }
            }
        }
        vector<vector<string>> ivvtemp;     // 合并两子句剩余部分
        ivvtemp.insert(ivvtemp.end(), ivtem1.begin(), ivtem1.end());
        ivvtemp.insert(ivvtemp.end(), ivtem2.begin(), ivtem2.end());
        vector<vector<string>> ivvttemp = insideCheck(ivvtemp);     // 内部检测
        if(ivvttemp.size() == 0){
            vector<string> ivttempNull;
            ivttempNull.push_back(string("2"));     // 空子句的限制，否则club测例出
bug
            vector<vector<string>> ivvttempNull;
            ivvttempNull.push_back(ivttempNull);
            clauses.push_back(ivvttempNull);
            numClauses ++;
            if(x1 >= initNumClauses && x2 >= initNumClauses) found = 1;
        }
        else if(!check(ivvttemp)){
            clauses.push_back(ivvttemp);
            numClauses ++;
        }
    }
    return toBeUpdate;
}
```

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.

   实现的归结代码见附录，也可见文件夹的`resolution.cpp`文件内。

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

   在归结过程中，我们采用了类似于`BFS`加上剪枝的实现方式，能够有效地提高效率。`BFS`的过程可以用一棵树来近似，除根节点外，树的每个节点都是任意两个子句的组合，任意节点的"儿子"是该节点对应的两个子句合一得到的子句与其余所有子句的组合（也就是说，假设在进行节点`c`的合一时子句集中有n个子句，则节点`c`有n个"儿子"）。剪枝的过程则是删除子句集中重复的子句或是那些可以进行内部合一的子句。

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

   输入文件见文件夹内的`hardworker.txt`，`3blocks.txt`，`club.txt`。输出结果则未写入文件，直接在终端显示。这里给出运行结果截图：

   Test Case1 hardworker(sue):

```
PS C:\Users\czh\Desktop\AI> cd "c:\Users\czh\Desktop\AI\" ; if ($?) { g++ resolution.cpp -o resolution } ; if ($?) { .\resolution }
  1> GradStudent(sue)
  2> ~GradStudent(x), Student(x)
  3> ~Student(x), HardWorker(x)
  4> ~HardWorker(sue)
  5> R[1, 2a] {x=sue} : Student(sue)
  6> R[3b, 4] {x=sue} : ~Student(sue)
  7> R[5, 6] {} : ( )
```

Test Case2 3-blocks:

```
PS C:\Users\czh\Desktop\AI> cd "c:\Users\czh\Desktop\AI\" ; if ($?) { g++ resolution.cpp -o resolution } ; if ($?) { .\resolution }
  1> On(aa, bb)
  2> On(bb, cc)
  3> Green(aa)
  4> ~Green(cc)
  5> ~On(x, y), ~Green(x), Green(y)
  6> R[1, 5a] {x=aa,y=bb} : ~Green(aa),Green(bb)
  7> R[2, 5a] {x=bb,y=cc} : ~Green(bb),Green(cc)
  8> R[3, 6a] {} : Green(bb)
  9> R[4, 7b] {} : ~Green(bb)
 10> R[8, 9] {} : ( )
```

Test Case3 Alpine Club:

```
PS C:\Users\czh\Desktop\AI> cd "c:\Users\czh\Desktop\AI\" ; if ($?) { g++ resolution.cpp -o resolution } ; if ($?) { .\resolution }
  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[1, 6a] {x=tony} : S(tony),C(tony)
 13> R[2, 6a] {x=mike} : S(mike),C(mike)
 14> R[7a, 12b] {y=tony} : ~L(tony, rain),S(tony)
 15> R[7a, 13b] {y=mike} : ~L(mike, rain),S(mike)
 16> R[8a, 9a] {z=tony,u=snow} : ~S(tony),~L(mike, snow)
 17> R[8b, 15b] {z=mike} : L(mike, snow),~L(mike, rain)
 18> R[10b, 17b] {v=rain} : L(tony, rain),L(mike, snow)
 19> R[14a, 18a] {} : S(tony),L(mike, snow)
 20> R[16a, 19a] {} : ( )
```

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

   首先有一个问题就是在于子句简化上，不仅仅要去除冗余项，得到有用的项，在操作时还要保证不能改变原子句的可满足性或不可满足性；还有就是要实现正确的`MGU`，其中的替换要合法，不能出现类似两个常量替换等等。

3.Experimental experience

本次Project在第一个问题上相对而言花费的时间要比第二个问题小很多。第一个问题就是实现`FC`和`GAC`，这两个算法的思路很清晰，只需要按照流程一步一步来就好了，需要注意的问题就是最后一个task里所说的“Average Inference Time Per Node”代表什么。第二个问题则很麻烦，花费了相当长的时间。首先要处理输入的格式问题，然后判断变量和常量，以及如何选择子句进行归结。中间最离谱的就是在Alpine club的测例里直接将$\neg L(tony,u),\neg L(mike,u)$、$L(tony,v),L(mike,v)$两句归结得到()，显然这是错误的。为了解决这个问题又修改了一晚上的代码，最后总算是看到了正确的归结过程。当然本次实现的Resolution并不是完美的，对于比较简单友好的题目运行起来还是没有问题的，对于一些比较复杂的测例可能会出现问题，还有考虑和改进的地方。

4.Appendix

1. `GAC`解决Futoshiki问题的代码：

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <list>
#include <set>
#include <string>
#include <vector>
#include <ctime>
```

```cpp
#include <chrono>
using namespace std;
using namespace std::chrono;

typedef vector<vector<int>> PUZZLE;          // 棋盘每个格子的赋值
typedef vector<vector<set<int>>> DOMAINS;    // 棋盘上每个格子的取值域
long num_of_nodes = 0;                       // 用来计算拓展的节点数
double sum_time = 0.0;                       // 上述的和

/* 约束类，用于表示一个约束 */
struct Constraint {
    int type;      // type==0为不等式约束，>0为行约束，<0为列约束。行列约束的绝对值代表第几行或第几列
    pair<pair<int, int>, pair<int, int>> inequality;  // 不等式约束，first < second
    Constraint(int type, pair<pair<int, int>, pair<int, int>> inequality = make_pair(make_pair(-1, -1), make_pair(-1, -1)))
        : type(type), inequality(inequality) {}
    bool operator==(const Constraint& y) {  // 重载运算符，主要用于find函数
        return (type == y.type && inequality == y.inequality);
    }
};

class Futoshiki {
   public:
    Futoshiki(const string puz_filename, const string con_filename);
    bool isSolved();
    void Show();
    DOMAINS makeDomains(list<Constraint>& gac_queue);          // 初始化
    pair<int, int> getNextPos(const DOMAINS& domains);  // 找下一个未初始化的空
    void push_cons(pair<int, int> pos, list<Constraint>& gac_queue);  // 将约束加入队列
    PUZZLE gac(const DOMAINS& domains, list<Constraint>& gac_queue);  // 赋值和递归的主过程
    DOMAINS gacEnforce(DOMAINS domains, list<Constraint>& gac_queue); // 根据限制更新值域

   private:
    int size;     // 棋盘边长
    PUZZLE puzzle;
    vector<Constraint> constraints;  // 只储存不等式约束，行列约束是平凡的，无需储存
};

/*
    从文件“puzzlex.txt”和“constraintsx”中分别读取初始谜题和限制。
    限制中的坐标是从（0,0）开始的。
*/
Futoshiki::Futoshiki(const string puz_filename, const string con_filename)
    : size(0) {
    ifstream puz_file(puz_filename.c_str()), con_file(con_filename.c_str());
    if(!puz_file || !con_file) {
        cerr << "Failed to open file. Check again." << endl;
        exit(-1);
    }
    string temp;
    int con_num = 0;
    while (getline(puz_file, temp)) size ++;
    while (getline(con_file, temp)) con_num ++;
    puz_file.clear(); puz_file.seekg(0);
```

```
    con_file.clear(); con_file.seekg(0);
    puzzle.assign(size, vector<int>(size, 0));

    for (int i = 0; i < size; i ++) {
        for (int j = 0; j < size; j ++) {
            puz_file >> puzzle[i][j];
        }
    }
    for (int i = 0; i < con_num; i ++) {
        int x1, y1, x2, y2;
        con_file >> x1 >> y1 >> x2 >> y2;
        constraints.push_back(Constraint(0, make_pair(make_pair(x1, y1),
make_pair(x2, y2))));
    }
    puz_file.close();
    con_file.close();
}

/* 如果所有空都被填上则解题完成 */
bool Futoshiki::isSolved() {
    for (int i = 0; i < puzzle.size(); i ++) {
        for (int j = 0; j < puzzle[0].size(); j ++) {
            if (puzzle[i][j] == 0) {
                return false;
            }
        }
    }
    return true;
}

/* 展示矩阵内容 */
void Futoshiki::Show() {
    for (int i = 0; i < size; i ++) {
        for (int j = 0; j < size; j ++) {
            cout << puzzle[i][j] << " ";
        }
        cout << endl;
    }
}

/* 从上到下、从左到右选取第一个未赋值的位置 */
pair<int, int> Futoshiki::getNextPos(const DOMAINS& domains) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (puzzle[i][j] == 0) {
                return make_pair(i, j);
            }
        }
    }
    return pair<int, int>();
}

/* GAC ，赋值过程（也即拓展节点的过程）在这里 */
PUZZLE Futoshiki::gac(const DOMAINS& domains, list<Constraint>& gac_queue) {
    if (domains.size() == 0) return PUZZLE();
    if (isSolved()) {
        return puzzle;  // 找到解，返回
    }
```

```
    pair<int, int> pos = getNextPos(domains);
    for (auto pd = domains[pos.first][pos.second].begin(); pd !=
domains[pos.first][pos.second].end(); pd++) {
        puzzle[pos.first][pos.second] = *pd;  // 赋值
        num_of_nodes ++;
        auto temp_domains = domains;          // 在temp_domains上修改，否则难恢复
        temp_domains[pos.first][pos.second].clear();
        temp_domains[pos.first][pos.second].insert(*pd);
        push_cons(pos, gac_queue);

        temp_domains = gacEnforce(temp_domains, gac_queue);

        if (temp_domains.size() != 0) {  // not DWO
            PUZZLE ret = gac(temp_domains, gac_queue);
            if (ret.size() != 0) return ret;  // 已找到解，直接返回该解
        }
    }
    puzzle[pos.first][pos.second] = 0;  // 恢复未赋值状态
    return PUZZLE();                    // 无解，返回空矩阵，size为0
}

/* 把与变量pos相关的不等式约束不重复地加入队列 */
void Futoshiki::push_cons(pair<int, int> pos, list<Constraint>& gac_queue) {
    for (int i = 0; i < constraints.size(); i++) {
        if (constraints[i].type != 0) {
            continue;
        }  // 只考虑不等式约束。行列约束稍后单独考虑
        pair<pair<int, int>, pair<int, int>> inequality =
constraints[i].inequality;
        if (pos == inequality.first || pos == inequality.second) {

            if (find(gac_queue.begin(), gac_queue.end(), constraints[i]) ==
gac_queue.end()) {  // 避免重复加入
                gac_queue.push_back(constraints[i]);
            }
        }
    }

    // 把变量的行列约束分别不重复地加入队列，特别注意type表示的行列索引是从1开始的
    Constraint row_constraint(Constraint(pos.first + 1));
    if (find(gac_queue.begin(), gac_queue.end(), row_constraint) ==
gac_queue.end()) {  // 避免重复加入
        gac_queue.push_back(row_constraint);
    }
    Constraint col_constraint(Constraint(-(pos.second + 1)));
    if (find(gac_queue.begin(), gac_queue.end(), col_constraint) ==
gac_queue.end()) {  // 避免重复加入
        gac_queue.push_back(col_constraint);
    }
}

DOMAINS Futoshiki::makeDomains(list<Constraint>& gac_queue) {
    // 初始化值域
    DOMAINS domains(size, vector<set<int>>(size, set<int>()));
    for (int i = 0; i < size; i ++) {
        for (int j = 0; j < size; j ++) {
            if (puzzle[i][j] == 0) {
                for (int k = 0; k < size; k ++) {
```

```
                    domains[i][j].insert(k + 1);
                }
            } else {
                domains[i][j].insert(puzzle[i][j]);
            }
        }
    }

    // 将所有约束（包括行列约束和不等式约束）加入gac_queue
    for (int i = 1; i <= size; i ++) {
        gac_queue.push_back(Constraint(i));   // 第i行约束
        gac_queue.push_back(Constraint(-i));  // 第i列约束，负数表示
    }
    for (int i = 0; i < constraints.size(); i++) {
        gac_queue.push_back(constraints[i]);  // 不等式约束
    }
    return gacEnforce(domains, gac_queue);  // 初始化后执行一次GAC enforce
}

/* GAC enforce ，也就是根据限制修改值域，根据这个时间来计算表格第五列 */
DOMAINS Futoshiki::gacEnforce(DOMAINS domains, list<Constraint>& gac_queue) {
    auto t1_per = steady_clock::now();
    while (!gac_queue.empty()) {
        Constraint constraint = gac_queue.front();
        gac_queue.pop_front();

        // 不等式约束
        if (constraint.type == 0) {
            // 以下采用了针对不等式约束优化过的GAC_Enforce，避免了多层循环，提高效率
            pair<int, int> small_pos = constraint.inequality.first;
            pair<int, int> large_pos = constraint.inequality.second;
            int minimum = *domains[small_pos.first][small_pos.second].begin();
            for (int k = 1; k <= minimum; k++) {
                bool did_erased = domains[large_pos.first]
[large_pos.second].erase(k);
                if (did_erased) {
                    if (domains[large_pos.first][large_pos.second].size() == 0)
return DOMAINS();  // DWO
                    push_cons(large_pos, gac_queue);
                }
            }
            int maximum = *domains[large_pos.first][large_pos.second].rbegin();
            for (int k = maximum; k <= size; k++) {
                bool did_erased = domains[small_pos.first]
[small_pos.second].erase(k);
                if (did_erased) {
                    if (domains[small_pos.first][small_pos.second].size() == 0)
return DOMAINS();  // DWO
                    push_cons(small_pos, gac_queue);
                }
            }
        }
        // 行约束
        else if (constraint.type > 0) {
            int i = constraint.type - 1;
            for (int j = 0; j < size; j++) {
                if (domains[i][j].size() == 1) {
                    int value = *domains[i][j].begin();
```

```
                    for (int j1 = 0; j1 < size; j1++) {
                        if (j1 != j) {
                            bool did_erased = domains[i][j1].erase(value);
                            if (did_erased) {
                                if (domains[i][j1].size() == 0) return
DOMAINS();  // DWO
                                push_cons(make_pair(i, j1), gac_queue);
                            }
                        }
                    }
                }
            }
        }
        // 列约束
        else {
            int j = -constraint.type - 1;
            for (int i = 0; i < size; i++) {
                if (domains[i][j].size() == 1) {
                    int value = *domains[i][j].begin();
                    for (int i1 = 0; i1 < size; i1++) {
                        if (i1 != i) {
                            bool did_erased = domains[i1][j].erase(value);
                            if (did_erased) {
                                if (domains[i1][j].size() == 0) return
DOMAINS();  // DWO
                                push_cons(make_pair(i1, j), gac_queue);
                            }
                        }
                    }
                }
            }
        }
    }
    auto t2_per = steady_clock::now();
    duration<double> time_span_per = duration_cast<duration<double>>(t2_per -
t1_per);
    sum_time += time_span_per.count();
    return domains;
}

int main() {
    Futoshiki game(string("puzzle5.txt"), string("constraints5.txt"));
    //puzzlei代表第i个题目文件，constraintsi代表第i个约束文件，修改i即可
    cout << "-----Initialization-----" << endl;
    game.Show();

    list<Constraint> gac_queue;
    auto domains = game.makeDomains(gac_queue);  // 初始化
    auto t1 = steady_clock::now();
    PUZZLE result = game.gac(domains, gac_queue);
    auto t2 = steady_clock::now();
    duration<double> time_span = duration_cast<duration<double>>(t2 - t1);

    if (result.size() != 0) {
        cout << "-------Solution------" << endl;
        game.Show();
        cout << "------------------------Outcoming--------------------------" <<
endl;
```

```
        cout << "Run time is in GAC :" << time_span.count()*1000 << "ms." <<
endl;
        cout << "Number of nodes GAC : " << num_of_nodes << "." << endl;
        cout << "Average time for constraint propagation : " << double(sum_time
/ num_of_nodes)*1000 << "ms." << endl;
        cout << "-------------------------------------------------------------" <<
endl;
    }
    else {
        cout << "No solution!" << endl;
    }
    return 0;
}
```

2. FC解决Futoshiki问题的代码：

```
#include <fstream>
#include <iostream>
#include <map>
#include <set>
#include <string>
#include <vector>
#include <ctime>
#include <chrono>
using namespace std;
using namespace std::chrono;

long num_of_nodes = 0;                          // 用来计算拓展的节点数
double sum_time = 0.0;                          // 上述的和

class Futoshiki {
public:
    int size;
    int con_num;
    vector<vector<int>> puzzle;
    vector<pair<pair<int, int>, pair<int, int>>> constraints;

    Futoshiki(const char* puz_filename, const char* con_filename, int size, int
con_num);
    bool isSolved();
    vector<vector<set<int>>> makeDomains();
    vector<vector<set<int>>> updateDomains(vector<vector<set<int>>> domains,
const pair<int, int>& pos);
    pair<int, int> mrv(const vector<vector<set<int>>>& domains);
    vector<vector<int>> forwardChecking(const vector<vector<set<int>>>&
domains);

private:
    // 可取值的个数
    int domainCount(const vector<vector<set<int>>>& domains) {
        int count = 0;
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                count += domains[i][j].size();
            }
        }
        return count;
```

```
    }
};

/*
    从文件“puzzlex.txt”和“constraintsx.txt”中分别读取初始谜题和限制。
    限制中的坐标是从（0,0）开始的。
*/
Futoshiki::Futoshiki(const char* puz_filename, const char* con_filename, int
size, int con_num)
    : size(size), con_num(con_num), puzzle(size, vector<int>(size, 0)) {
    ifstream puz_file(puz_filename), con_file(con_filename);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            puz_file >> puzzle[i][j];
        }
    }

    for (int i = 0; i < con_num; i++) {
        int x1, y1, x2, y2;
        con_file >> x1 >> y1 >> x2 >> y2;
        // 这里不要 x-1, y-1什么的 ！！！！从0开始。
        constraints.push_back(make_pair(make_pair(x1, y1), make_pair(x2, y2)));
    }
    puz_file.close();
    con_file.close();
}

/* 如果所有空都被填上则解题完成 */
bool Futoshiki::isSolved() {
    for (int i = 0; i < puzzle.size(); i++) {
        for (int j = 0; j < puzzle[0].size(); j++) {
            if (puzzle[i][j] == 0) {
                return false;
            }
        }
    }
    return true;
}

vector<vector<set<int>>> Futoshiki::makeDomains() {
    // 初始化x*x*9的三维矩阵
    vector<vector<set<int>>> domains(size, vector<set<int>>(size, set<int>()));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (puzzle[i][j] == 0) {
                for (int k = 1; k <= size; k ++) {
                    domains[i][j].insert(k);
                }
            } else {
                domains[i][j].insert(puzzle[i][j]);
            }
        }
    }

    // 根据行列限制删除部分取值（同行同列数字不重复）
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (puzzle[i][j] != 0) {
```

```
                for (int i2 = 0; i2 < size; i2++) {
                    if (i2 != i) {
                        domains[i2][j].erase(puzzle[i][j]);
                    }
                }
                for (int j2 = 0; j2 < size; j2++) {
                    if (j2 != j) {
                        domains[i][j2].erase(puzzle[i][j]);
                    }
                }
            }
        }
    }

    // 根据小于限制删除部分取值
    int old_domain_count = 0, new_domain_count;
    // 外层while循环使得这个循环执行到任何值域都不能缩小为止
    do {
        for (int i = 0; i < con_num; i++) {
            pair<int, int> small_pos = constraints[i].first;
            pair<int, int> large_pos = constraints[i].second;
            if (puzzle[large_pos.first][large_pos.second] != 0) {  // large_pos
has been assigned
                for (int k = puzzle[large_pos.first][large_pos.second]; k <=
size; k++) {
                    domains[small_pos.first][small_pos.second].erase(k);
                }
            }
            else {  // large_pos has not been assigned
                int minimum = *domains[small_pos.first]
[small_pos.second].begin();
                domains[large_pos.first][large_pos.second].erase(minimum);
            }
            if (puzzle[small_pos.first][small_pos.second] != 0) {
                for (int k = 1; k <= puzzle[small_pos.first][small_pos.second];
k++) {
                    domains[large_pos.first][large_pos.second].erase(k);
                }
            }
            else {
                int minimum = *domains[large_pos.first]
[large_pos.second].rbegin();
                domains[small_pos.first][small_pos.second].erase(minimum);
            }
        }
        new_domain_count = domainCount(domains);
    } while(old_domain_count == new_domain_count);

    return domains;
}

/*
    在每次赋值（即拓展一个节点后）更新值域。
    主要还是根据行列限制和小于限制。
    根据这个时间来计算表格第五列.
*/
vector<vector<set<int>>> Futoshiki::updateDomains(vector<vector<set<int>>>
domains, const pair<int, int>& pos) {
```

```
    auto t1_per = steady_clock::now();
    // 检查同行是否有重复
    for (int i = 0; i < size; i++) {
        if (i == pos.first)
            continue;
        else if (puzzle[i][pos.second] == puzzle[pos.first][pos.second]) {
            return vector<vector<set<int>>>();  // DWO，其实这一段不太必要
        } else {
            domains[i][pos.second].erase(puzzle[pos.first][pos.second]);
            if (domains[i][pos.second].size() == 0) {
                return vector<vector<set<int>>>();  // DWO
            }
        }
    }

    // 检查同列是否有重复
    for (int j = 0; j < size; j++) {
        if (j == pos.second)
            continue;
        else if (puzzle[pos.first][j] == puzzle[pos.first][pos.second]) {
            return vector<vector<set<int>>>();  // DWO
        } else {
            domains[pos.first][j].erase(puzzle[pos.first][pos.second]);
            if (domains[pos.first][j].size() == 0) {
                return vector<vector<set<int>>>();  // DWO
            }
        }
    }

    // 小于限制
    for (int i = 0; i < con_num; i++) {
        pair<int, int> small_pos = constraints[i].first;
        pair<int, int> large_pos = constraints[i].second;
        if (pos == large_pos) {
            for (int k = puzzle[pos.first][pos.second]; k <= size; k++) {
                domains[small_pos.first][small_pos.second].erase(k);
                if (puzzle[small_pos.first][small_pos.second] == 0 &&
domains[small_pos.first][small_pos.second].size() == 0) {
                    return vector<vector<set<int>>>();  // DWO
                }
            }
        } else if (pos == small_pos) {
            for (int k = 1; k <= puzzle[pos.first][pos.second]; k++) {
                domains[large_pos.first][large_pos.second].erase(k);
                if (puzzle[large_pos.first][large_pos.second] == 0 &&
domains[large_pos.first][large_pos.second].size() == 0) {
                    return vector<vector<set<int>>>();  // DWO
                }
            }
        }
    }
    auto t2_per = steady_clock::now();
    duration<double> time_span_per = duration_cast<duration<double>>(t2_per -
t1_per);
    sum_time += time_span_per.count();
    return domains;
}
```

```cpp
/* 找到值域最小的点并返回以拓展之 */
pair<int, int> Futoshiki::mrv(const vector<vector<set<int>>>& domains) {
    int min_val = size * size;  // max size of domain
    pair<int, int> min_pos = make_pair(-1, -1);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (puzzle[i][j] == 0 && domains[i][j].size() < min_val) {
                min_val = domains[i][j].size();
                min_pos = make_pair(i, j);
            }
        }
    }
    return min_pos;
}

/*
    FC，赋值（即拓展节点）的过程在此
    根据MRV找到需要拓展的点，根据其值域遍历赋值
    然后递归
    若无解，最后要恢复
*/
vector<vector<int>> Futoshiki::forwardChecking(const vector<vector<set<int>>>&
domains) {
    if (isSolved()) {
        return puzzle;
    }

    pair<int, int> pos = mrv(domains);

    for (auto pd = domains[pos.first][pos.second].begin(); pd !=
domains[pos.first][pos.second].end(); pd++) {
        puzzle[pos.first][pos.second] = *pd;                // 赋值
        num_of_nodes ++;
        auto temp_domains = updateDomains(domains, pos);    // 创建一个副本用来迭代
        if (temp_domains.size() != 0) {  // 非 DWO
            vector<vector<int>> ret = forwardChecking(temp_domains);
            if (ret.size() != 0) return ret;
        }
    }

    puzzle[pos.first][pos.second] = 0;  // 该情况无解，恢复到赋值以前的状态
    return vector<vector<int>>();
}

int main() {
    //Futoshiki game("puzzle1.txt", "constraints1.txt", 5, 7);
    //Futoshiki game("puzzle2.txt", "constraints2.txt", 6, 9);
    //Futoshiki game("puzzle3.txt", "constraints3.txt", 7, 19);
    //Futoshiki game("puzzle4.txt", "constraints4.txt", 8, 25);
    Futoshiki game("puzzle5.txt", "constraints5.txt", 9, 32);
    //后面两个数字代表题目的size和约束条件的个数，即class内定义的size和con_num。

    cout << "-----Initialization-----" << endl;
    for (int i = 0; i < game.size; i++) {
        for (int j = 0; j < game.size; j++) {
            cout << game.puzzle[i][j] << " ";
        }
        cout << endl;
```

```
    }

    auto domains = game.makeDomains();

    auto t1 = steady_clock::now();
    vector<vector<int>> result = game.forwardChecking(domains);
    auto t2 = steady_clock::now();
    duration<double> time_span = duration_cast<duration<double>>(t2 - t1);

    if (result.size() != 0) {
        cout << "-------Solution------" << endl;
        for (int i = 0; i < game.size; i++) {
            for (int j = 0; j < game.size; j++) {
                cout << result[i][j] << " ";
            }
            cout << endl;
        }
        cout << "------------------------Outcoming--------------------------" <<
endl;
        cout << "Run time is in FC :" << time_span.count()*1000 << "ms." <<
endl;
        cout << "Number of nodes in FC : " << num_of_nodes << "." << endl;
        cout << "Average time for constraint propagation : " << double(sum_time
/ num_of_nodes)*1000 << "ms." << endl;
        cout << "-----------------------------------------------------------" <<
endl;
    } else {
        cout << "No solution!" << endl;
    }
    return 0;
}
```

3. 归结代码如下：

```
/*
输入实例：

###### input for AIpine Club ######
# 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)
####################################


#### input for hardworker(sue) ####
# GradStudent(sue)
# ~GradStudent(x), Student(x)
# ~Student(x), HardWorker(x)
# ~HardWorker(sue)
```

```
###################################

####### input for 3' blocks #######
# On(aa, bb)
# On(bb, cc)
# Green(aa)
# ~Green(cc)
# ~On(x, y), ~Green(x), Green(y)
###################################

*/

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

using namespace std;

/*
    用于存储子句的容器。
    每一层 vector<string> 存储一个谓词语句，结构为：
        | 是否为真 | 函数名 | 量名 | 量名 | 。。。|
    其中是否为真用字符“0”表示假，字符“1”表示真。
    每一层 vector<vector<string>> 表示一个子句，因为一个子句中常有多个谓词语句。
        | 是否为真 | 函数名 | 量名 | 量名 | 。。。|
        | 是否为真 | 函数名 | 量名 | 量名 | 。。。|
        | 是否为真 | 函数名 | 量名 | 量名 | 。。。|
        。。。。。。
    最外层则是所有子句的集合。
*/
vector<vector<vector<string>>> clauses;
vector<string> results;
int nowClauses = 0; // 当前子句位置
int numClauses = 0; // 字句数量
int found = 0;      // 没找到
// int lastFound = 0;
bool checkout = 0;  // 无重复
int initNumClauses = 0;  // 初始子句数量

/*
    用来判断是否为常量，因为题目友好，只要判断长度是否为1即可
    常量返回1（长度大于1），变量返回0
*/
bool whether_const(string &s1){
    return !(s1.size() == 1);
}

void show(){
    cout << "-----------------------------------------------------" << endl;
    for(int i = 0; i < clauses.size(); i ++){
        if(i < 10) cout << "  ";
        else if(i < 100) cout << " ";
        cout << i << "> ";
        for(int j = 0; j < clauses[i].size(); j ++){
            for(int k = 0; k < clauses[i][j].size(); k ++){
                cout << clauses[i][j][k] << " ";
            }
```

```
            cout << " | ";
        }
        cout << endl;
    }
    cout << "-------------------------------------------------"<< endl;
}

void removeUselessCons(){
    int toBeChanged[results.size()] = {0};
    for(int i = 0; i < initNumClauses; i ++) toBeChanged[i] = i + 1;
    int now_num = results.size() - 1;
    toBeChanged[now_num] = initNumClauses + 1;
    while(now_num >= initNumClauses){
        for(int j = 0; j < results[now_num].size(); j ++){
            if(results[now_num][j] >= '0' && results[now_num][j] <= '9'){
                int num_before_now = 0, a_pos = results[now_num][j] - '1';
                if(results[now_num][j + 1] >= '0' && results[now_num][j + 1] <=
'9'){
                    a_pos = (a_pos + 1) * 10 + results[now_num][j + 1] - '1';
                    j ++;
                    if(results[now_num][j + 1] >= '0' && results[now_num][j + 1]
<= '9'){
                        a_pos = (a_pos + 1) * 10 + results[now_num][j + 1] -
'1';
                        j ++;
                    }
                }
                if(toBeChanged[a_pos] == 0){
                    for(int k = 0; k < a_pos; k ++){
                        if(toBeChanged[k] != 0) num_before_now ++;
                    }
                    toBeChanged[a_pos] = num_before_now + 1;
                    for(int k = a_pos + 1; k < results.size(); k ++){
                        if(toBeChanged[k] != 0) toBeChanged[k] ++;
                    }
                }
            }
        }
        now_num --;
        while(toBeChanged[now_num] == 0) now_num --;
    }
    // for(int i = 0; i < results.size(); i ++){
    //     if(toBeChanged[i] != 0) cout << i << " ";
    // }
    // cout << endl;
    int num_results = results.size(), num_removed = 0;
    for(int i = 0; i < num_results; i ++){
        // cout << results[i - num_removed] << " " << endl ;
        if(toBeChanged[i] == 0){
            results.erase(results.begin() + i - num_removed);
            num_removed ++;
        }
        else{
            // cout << i << " " << i - num_removed << endl;
            for(int j = 0; j < results[i - num_removed].size(); j ++){
                if(results[i - num_removed][j] >= '0' && results[i -
num_removed][j] <='9'){
```

```
                    int a_pos = results[i - num_removed][j] - '1', moreThanTen =
0, moreThanHun = 0;
                    if(results[i - num_removed][j + 1] >= '0' && results[i -
num_removed][j + 1] <= '9'){
                        a_pos = (a_pos + 1) * 10 + results[i - num_removed][j +
1] - '1';
                        j ++;
                        moreThanTen = 1;
                        if(results[i - num_removed][j + 1] >= '0' && results[i -
num_removed][j + 1] <= '9'){
                            a_pos = (a_pos + 1) * 10 + results[i - num_removed]
[j + 1] - '1';
                            j ++;
                            moreThanHun = 1;
                        }
                    }
                    // cout << a_pos << " ";
                    if(moreThanHun && toBeChanged[a_pos] >= 100){
                        results[i - num_removed][j] = toBeChanged[a_pos] % 10 +
'0';
                        results[i - num_removed][j - 1] = toBeChanged[a_pos] /
10 % 10 + '0';
                        results[i - num_removed][j - 2] = toBeChanged[a_pos] /
100 + '0';
                    }
                    else if(moreThanHun && toBeChanged[a_pos] >= 10){
                        results[i - num_removed][j - 1] = toBeChanged[a_pos] %
10 + '0';
                        results[i - num_removed][j - 2] = toBeChanged[a_pos] /
10 + '0';
                        results[i - num_removed].erase(results[i -
num_removed].begin() + j);
                        j --;
                    }
                    else if(moreThanTen && toBeChanged[a_pos] >= 10){
                        results[i - num_removed][j] = toBeChanged[a_pos] % 10 +
'0';
                        results[i - num_removed][j - 1] = toBeChanged[a_pos] /
10 + '0';
                    }
                    else if(moreThanTen && toBeChanged[a_pos] < 10){
                        results[i - num_removed][j - 1] = toBeChanged[a_pos] +
'0';
                        results[i - num_removed].erase(results[i -
num_removed].begin() + j);
                        j --;
                    }
                    else{
                        results[i - num_removed][j] = toBeChanged[a_pos] + '0';
                    }
                }
            }
            // cout << endl;
        }
    }
}

/*
```

```
    每次都要检测当前子句集中是否有重复的项
    有重复返回1
*/
bool check(vector<vector<string>> &ivvarr){
    for(int i = 0; i < clauses.size(); i ++){
        if(ivvarr == clauses[i]){
            checkout = 1;
            return 1;
        }
    }
    return 0;
}

/*
    检测是否真的找到解
    因为中途遇到直接用条件里的两个子句归结得到空的情况
    （说的就是你！AIpine_Club的
        ~L(tony, u), ~L(mike, u)
        L(tony, v), L(mike, v)
    就离谱！）
*/
// bool checkFound(int x1, int x2){
//     if(x1 > initNumClauses - 1 && x2 > initNumClauses - 1){
//         int x1a_pos = -1, x1b_pos = -1, x2a_pos = -1, x2b_pos = -1;
//         for(int j = 0; j < results[x1].size(); j ++){
//             if(results[x1][j] >= '0' && results[x1][j] <= '9'){
//                 int pos = results[x1][j] - '1';
//                 if(results[x1][j + 1] >= '0' && results[x1][j + 1] <= '9'){
//                     pos = (pos + 1) * 10 + results[x1][j + 1] - '1';
//                     j ++;
//                 }
//                 if(x1a_pos < 0) x1a_pos = pos;
//                 else x1b_pos = pos;
//             }
//         }
//         for(int j = 0; j < results[x2].size(); j ++){
//             if(results[x2][j] >= '0' && results[x2][j] <= '9'){
//                 int pos = results[x2][j] - '1';
//                 if(results[x2][j + 1] >= '0' && results[x2][j + 1] <= '9'){
//                     pos = (pos + 1) * 10 + results[x2][j + 1] - '1';
//                     j ++;
//                 }
//                 if(x2a_pos < 0) x2a_pos = pos;
//                 else x2b_pos = pos;
//             }
//         }
//         cout << x1a_pos << x1b_pos << x2a_pos << x2b_pos << endl;
//         if(x1a_pos >= 0) return checkFound(x1a_pos, x1b_pos) ||
checkFound(x2a_pos, x2b_pos);
//         else return false;
//     }
//     else if(x1 == initNumClauses - 1 || x2 == initNumClauses - 1) return
true;
//     else if(x1 < initNumClauses - 1 && x2 < initNumClauses - 1) return false;
// }

/*
    每次检测字句内是否有可归结或重复的项
```

```
*/
vector<vector<string>> insideCheck(vector<vector<string>> &ivvarr){
    for(int i = 0; i < ivvarr.size(); i ++){
        for(int j = i + 1; j < ivvarr.size(); j ++){
            if(ivvarr[i][1] != ivvarr[j][1]){
                continue;
            }
            else{
                int f1 = 1;
                for(int k = 2; k < ivvarr[i].size(); k ++){
                    if(ivvarr[i][k] != ivvarr[j][k]){
                        f1 = 0;
                    }
                }
                if(f1){
                    if(ivvarr[i][0] != ivvarr[j][0]){
                        ivvarr.erase(ivvarr.begin() + j);
                        ivvarr.erase(ivvarr.begin() + i);
                    }
                    else{
                        ivvarr.erase(ivvarr.begin() + j);
                    }
                }
            }

        }
    }
    return ivvarr;
}

vector<pair<string, pair<int, string>>> MGU(int x1, int y1, int x2, int y2, int
size){
    vector<pair<string, pair<int, string>>> toBeUpdate;
    vector<pair<string, pair<int, string>>> nonUpdate;
    for(int i = 2; i < size; i ++){
        // cout << clauses[x1][y1][i] << " " << clauses[x2][y2][i] << endl;
        if(!(whether_const(clauses[x1][y1][i]) && whether_const(clauses[x2][y2]
[i]))){
            if(!whether_const(clauses[x1][y1][i])){
                toBeUpdate.push_back(make_pair(clauses[x1][y1][i], make_pair(i,
clauses[x2][y2][i])));
            }
            else{
                toBeUpdate.push_back(make_pair(clauses[x2][y2][i], make_pair(i,
clauses[x1][y1][i])));
            }
        }
        else{
            if(clauses[x1][y1][i] == clauses[x2][y2][i]){
                toBeUpdate.push_back(make_pair(clauses[x1][y1][i], make_pair(i,
clauses[x2][y2][i])));
            }
            else{
                 return nonUpdate;
            }
        }
    }
    vector<vector<string>> ivtem1(clauses[x1]), ivtem2(clauses[x2]);
```

```
    if(toBeUpdate.size()){
        ivtem1.erase(ivtem1.begin() + y1);
        ivtem2.erase(ivtem2.begin() + y2);
        for(int i = 0; i < toBeUpdate.size(); i ++){
            for(int v = 0; v < ivtem1.size(); v ++){
                for(int j = 2; j < size; j ++){
                    if(ivtem1[v][j] == toBeUpdate[i].first){
                        ivtem1[v][j] = toBeUpdate[i].second.second;
                    }
                }
            }
            for(int v = 0; v < ivtem2.size(); v ++){
                for(int j = 2; j < size; j ++){
                    if(ivtem2[v][j] == toBeUpdate[i].first){
                        ivtem2[v][j] = toBeUpdate[i].second.second;
                    }
                }
            }
        }
        vector<vector<string>> ivvtemp;
        ivvtemp.insert(ivvtemp.end(), ivtem1.begin(), ivtem1.end());
        ivvtemp.insert(ivvtemp.end(), ivtem2.begin(), ivtem2.end());
        vector<vector<string>> ivvttemp = insideCheck(ivvtemp);
        if(ivvttemp.size() == 0){
            vector<string> ivttempNull;
            ivttempNull.push_back(string("2"));
            vector<vector<string>> ivvttempNull;
            ivvttempNull.push_back(ivttempNull);
            clauses.push_back(ivvttempNull);
            numClauses ++;
            if(x1 >= initNumClauses && x2 >= initNumClauses) found = 1;
            // show();
            // if(checkFound(x1, x2)) found += 1;
            // else{
            //     // cout << "a" << endl;
            //     vector<string> ivttempNull;
            //     ivttempNull.push_back(string("2"));
            //     vector<vector<string>> ivvttempNull;
            //     ivvttempNull.push_back(ivttempNull);
            //     clauses.push_back(ivvttempNull);
            //     numClauses ++;
            //     //show();
            // }
        }
        else if(!check(ivvttemp)){
            clauses.push_back(ivvttemp);
            numClauses ++;
        }
        // show();
        // for(int j = 0; j < clauses.back().size(); j ++){
        //     cout << " | ";
        //     for(int k = 0; k < clauses.back()[j].size(); k ++){
        //         cout << clauses.back()[j][k] << " ";
        //     }
        // }
        // cout << endl;
    }
    return toBeUpdate;
```

```
}

void do_big_things(){
    if(clauses[nowClauses][0][0][0] == '2') return;
    int max = numClauses;
    for(int i = 0; i < clauses[nowClauses].size(); i ++){
        char needTruth = '1' + '0' - clauses[nowClauses][i][0][0];
        string needFuncName = clauses[nowClauses][i][1];
        // cout << needTruth << " " << needFuncName << endl;
        for(int j = 0; j < max; j ++){
            if(j == nowClauses) continue;
            for(int k = 0; k < clauses[j].size(); k ++){
                if(found == 1) break;
                if(clauses[j][k][0][0] == needTruth && clauses[j][k][1] ==
needFuncName){
                    // cout << nowClauses << " " << i << " " << j << " " << k <<
" " << endl;
                    vector<pair<string, pair<int, string>>> updated =
MGU(nowClauses, i, j, k, clauses[j][k].size());
                    // if(found == 1) cout << "b" << endl;
                    if(updated.size()){
                        string sstemp = "R[";
                        sstemp += to_string(nowClauses + 1);
                        if(clauses[nowClauses].size() > 1){
                            sstemp += string("a");
                            sstemp[sstemp.size() - 1] += i;
                        }
                        sstemp += string(", ");
                        sstemp += to_string(j + 1);
                        if(clauses[j].size() > 1){
                            sstemp += string("a");
                            sstemp[sstemp.size() - 1] += k;
                        }
                        sstemp += string("] {");
                        for(int v = 0; v < updated.size(); v ++){
                            if(!(updated[v].first == updated[v].second.second &&
(whether_const(updated[v].first) && whether_const(updated[v].second.second)))){
                                sstemp += updated[v].first;
                                sstemp += string("=");
                                sstemp += updated[v].second.second;
                                if(v != updated.size() - 1) sstemp +=
string(",");
                            }
                            // else{
                            //     sstemp += string(" ");
                            // }
                        }
                        if(sstemp.back() == ',') sstemp.erase(sstemp.end() - 1);
                        sstemp += string("} : ");
                        // if(clauses.back().size() > 1){
                        //     sstemp += string("(");
                        // }
                        if(!found){
                            // cout << "b" << endl;
                            for(int v = 0; v < clauses.back().size(); v ++){
                                // cout << "c" <<endl;
                                if(clauses.back()[v][0] == "0"){
                                    sstemp += string("~");
```

```
                                }
                                else if(clauses.back()[v][0] == "2"){
                                    sstemp += string("( )");
                                    break;
                                }
                                sstemp += clauses.back()[v][1];
                                sstemp += string("(");
                                for(int n = 2; n < clauses.back()[v].size(); n
++){
                                    sstemp += clauses.back()[v][n];
                                    if(n != clauses.back()[v].size() - 1) sstemp
+= string(", ");
                                }
                                sstemp += string(")");
                                if(v != clauses.back().size() - 1) sstemp +=
string(",");
                            }

                        }
                        else{
                            sstemp += string("( )");
                            // lastFound = found;
                        }
                        if(!checkout){
                            results.push_back(sstemp);
                        }
                        else{
                            checkout = 0;
                        }
                        // cout << "end in : " << results.size() << endl;
                    }
                }
            }
        }
    }
}

int main(){
    string filename = "hardworker.txt";        // 测例文件，修改文件名即可测试不同测例
    ifstream pfile;
    string buf;
    pfile.open(filename.c_str());

    /*
        以下是将输入转化为clauses指定格式的过程
        注意，题目友好（指一个谓词语句只有一对）时才成立
    */
    while(getline(pfile, buf)){
        numClauses ++;
        results.push_back(buf);
        vector<string> iv;
        vector<vector<string>> ivv;
        int newOne = 1;
        for(int i = 0; i < buf.size(); i ++){
            if(newOne == 1){
                if(buf[i] == '~'){
                    iv.push_back(string(1, '0'));
                    newOne = 0;
```

```
                continue;
            }
            else{
                iv.push_back(string(1, '1'));
                newOne = 0;
                int j = i;
                string funcName, vcName;
                while(buf[j] != '('){       // 读取函数名
                    funcName += buf[j];
                    j ++;
                }
                iv.push_back(funcName);
                j ++;
                while(buf[j] != ')'){       // 读取量名，变量常量都读进来
                    if(buf[j] == ','){
                        iv.push_back(vcName);
                        vcName.clear();
                        j += 2;
                    }
                    else{
                        vcName += buf[j];
                        j ++;
                    }
                }
                iv.push_back(vcName);
                i = j + 2;
                newOne = 1;
                ivv.push_back(iv);
                iv.clear();
            }
        }
        else{
            int j = i;
            string funcName, vcName;
            while(buf[j] != '('){       // 读取函数名
                funcName += buf[j];
                j ++;
            }
            iv.push_back(funcName);
            j ++;
            while(buf[j] != ')'){       // 读取量名，变量常量都读进来
                if(buf[j] == ','){
                    iv.push_back(vcName);
                    vcName.clear();
                    j += 2;
                }
                else{
                    vcName += buf[j];
                    j ++;
                }
            }
            iv.push_back(vcName);
            i = j + 2;
            newOne = 1;
            ivv.push_back(iv);
            iv.clear();
        }
    }
```

```
        // for(int i = 0; i < ivv.size(); i ++){
        //     for(int j = 0; j < ivv[i].size(); j ++){
        //         cout << ivv[i][j] << " ";
        //     }
        //     cout << endl;
        // }
        clauses.push_back(ivv);
    }
    initNumClauses = numClauses;
    //show();
    while(nowClauses < clauses.size()){
        do_big_things();
        nowClauses ++;
        if(found == 1) break;
    }

    /* test */
    // nowClauses = 3;
    // do_big_things();

    //cout << results.size() << endl;
    // for(int i = 0; i < results.size(); i ++){
    //     if(i + 1 < 10) cout << "  ";
    //     else if(i + 1 < 100) cout << " ";
    //     cout << i + 1 << "> " << results[i] << endl;
    // }
    // show();
    removeUselessCons();
    for(int i = 0; i < results.size(); i ++){
        if(i + 1 < 10) cout << "  ";
        else if(i + 1 < 100) cout << " ";
        cout << i + 1 << "> " << results[i] << endl;
    }
}
```