P02 CSP and KRR

学号	姓名	专业(方向)
18340229 18308045	周泰来 谷正阳	计算机类（超算） 大数据

1.task

- Describe with sentences the main ideas of the GAC algorithm and the main differences between the GAC and the forward checking (FC) algorithm.
    - GAC算法维护一个弧（约束）相容队列。首先队列中包含CSP中的所有弧（约束）。GAC从队列中弹出弧（X_i ,X_j）,首先使X_j相对X_i弧相容。如果D_i（即X_i的可能值域的集合）没有变化，则处理下一条弧。若发生变化，那么每个关于X_i的约束（X_k,X_i）重新插入约束队列中进行检测。
    - 一个约束满足问题有若干个变量 V 和约束条件 C。若对于某个变量 Vi 的值域中的任一值，都能在相关变量的值域中找到一组值使这组取值满足约束条件 Ci，则称这一变量 Vi 相对于约束条件 Ci 是 GAC（一般边一致）的。若约束条件 Ci 相关的任一变量相对于 Ci 是 GAC 的，则约束条件 Ci 是 GAC 的。
    - 若一个约束满足问题的任一约束条件 Ci 是 GAC 的，则这个约束满足问题是 GAC 的。 GAC 算法的主要思想是在开始搜索前和对一个未赋值的变量赋值后，就对整个约束满足问题进 行一次 GAC 检测。具体的做法是建立一个检测队列 (先进先出)，将与这次赋值的变量有关的约束条件加入检测队列中（在进行初始 GAC 检测时，将所有约束条件加入检测队列中）。重复以下动作 直到检测队列为空：从队列中取出一个约束条件 C，对于约束条件 C 的每个变量的值域中的一个 值，如果不能在其余变量的值域中找到一组值使之满足约束条件 C，则将这个值从该变量的值域中 删去，然后将所有与该变量相关且不在检测队列中的约束条件添加到检测队列中。在进行一轮完整 的 GAC 检测后，如果有尚未赋值的变量值域变为空，则说明这次的赋值不可取，出现 DWO，将 GAC 检测改变的值域还原，进行回溯。
    - GAC 算法与 FC 算法的不同之处就在于在每一步做的缩小尚未赋值的变量的值域的检测不同。GAC 算法在赋值后以赋值相关的约束为起点对约束满足问题进行一次 GAC 检测，删去不满足GAC 的值域里的值，会在修改与被赋值变量相关的约束涉及到的变量的值域之后，再修改这些被修改了的变量相关的约束涉及到的变量的值域，直到没有变量的值域被修改为止，缩小值域的力度非常大，而 FC 算法只尽行一次 FC 检测，删去那些与这一步赋值的变量相关的变量的值域中与这一步的赋值不相容的值，仅修改与被赋值变量相关的约束涉及到的变量的值域。
- 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?
    - 进行这一步的原因是当一个变量的值域被改变时，可能会导致原先满足 GAC 的约束条件不再满足GAC。因为当前约束条件加入导致的D_i（即X_i的值域取值的集合）的改变有可能引起D_k（即任意X_k的取值值域的集合）的缩小，从而导致结果不正确
    - 可以被改进，在初始化时会存储所有限制条件，在移除值后，不用进行限制条件检测，只需遍历值域列表和更新限制计数器，以避免在删除传播期间重复几次相同的约束检查，在不牵扯约束具体内容的情况下，已经具有最优的最差时间复杂度。
    - 另外，只使用队列数据结构，检测约束条件是否在队 列中是 O(n) 的，可以将队列与数组或哈希表结合，使检测约束条件是否在检测队列中这一动作变 为 O(1) 的。在初始化时，对于每个值都要找出所有与其匹配的值，以证明该值的有效性。而如果只寻找一个匹配值，如果这个值从其值域中删除了，再寻找下一个匹配值
- Explain any ideas you use to speed up the implementation.

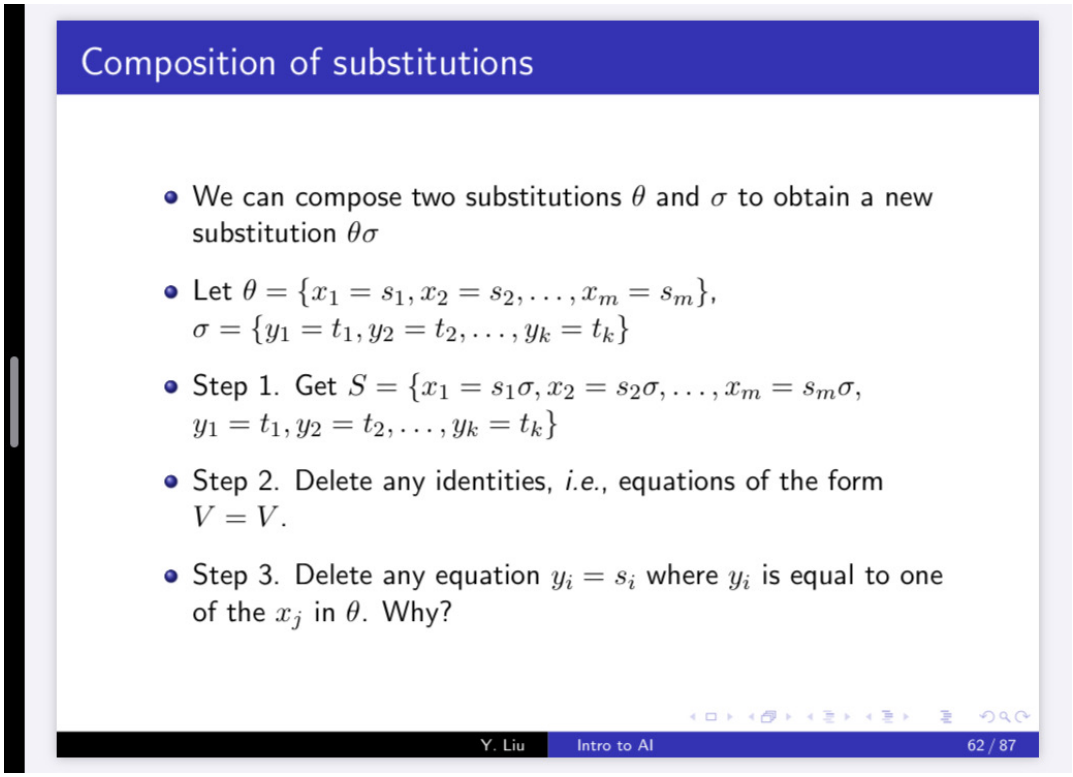
- 在初始化时会存储所有限制条件，在移除值后，不用进行限制条件检测，只需遍历值域列表和更新限制计数器，在找到一组解后在值域中将这组解标出来，被标记的值域中的值已经满足GAC，不需要进行重复的 GAC 检测
- 将每个变量有关的约束条件在初始化时储存在特殊的数据结构中，这样后面需要添加约束条件时就不需要遍历所有约束条件。
- 采用了针对不等式约束优化过的GAC_Enforce，避免了多层循环，提高效率
- ```
      pair<int, int> large_pos = constraint.inequality.second;
      pair<int, int> small_pos = constraint.inequality.first;
      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
              pushConstraintIntoQueue(large_pos, gac_queue);
                //行约束，以下方法已经过优化以提高效率
            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
                                    pushConstraintIntoQueue(make_pair(i,
  j1), gac_queue);
  ```
TestCase Number	Method	Time Used(μs)	Nodes Searched	Time Per Node
1	FC GAC	8783.2 6388.4	266 24	0.406 5.469
2	FC GAC	8345.6 7974.1	58 31	0.397 5.839
3	FC GAC	1104980 9912.5	433893 46	0.465 7.098
4	FC GAC	712309 14288.5	205516 58	0.601 8.019
5	FC GAC	∞ 30781.5	∞ 81	∞ 35.182
- 随着问题的规模增加，FC 解决问题的时间成倍增加，访问节点的个数呈指数型上升，每个节点进行 FC 检测的时间保持在较短的时间内。
  ```

- 使用 GAC 所耗的总时长比 FC 短，FC 访问的节点数比 GAC 多的多，进行一次 GAC 检测比进行一次 FC 检测的时间长的多。问题规模增加的时候，GAC 解决问题的速度相对FC问题显著变快，访问的节点个数很少，但是每个节点进行 GAC 检测的平均时间显著增加。
- 原因如下：进行一次 GAC 检测比进行一次 FC 检测复杂的多，耗费的时间当然更多。但是同时，GAC 缩减值域的效果也比 FC 好实在太多，所以 GAC 算法几乎不需要进行几次回溯，访问的节点数非常少，但 FC 算法会大量回溯，访问的节点数极多。而总体来看，GAC 算法的效果比 FC 算法好，在问题不复杂时尚不明显，在问题非常复杂时，两者的差距非常巨大，以至于第五个例 子 GAC 只需要 5 秒左右就能得到结果，而 FC 几个小时依然无法得到结果

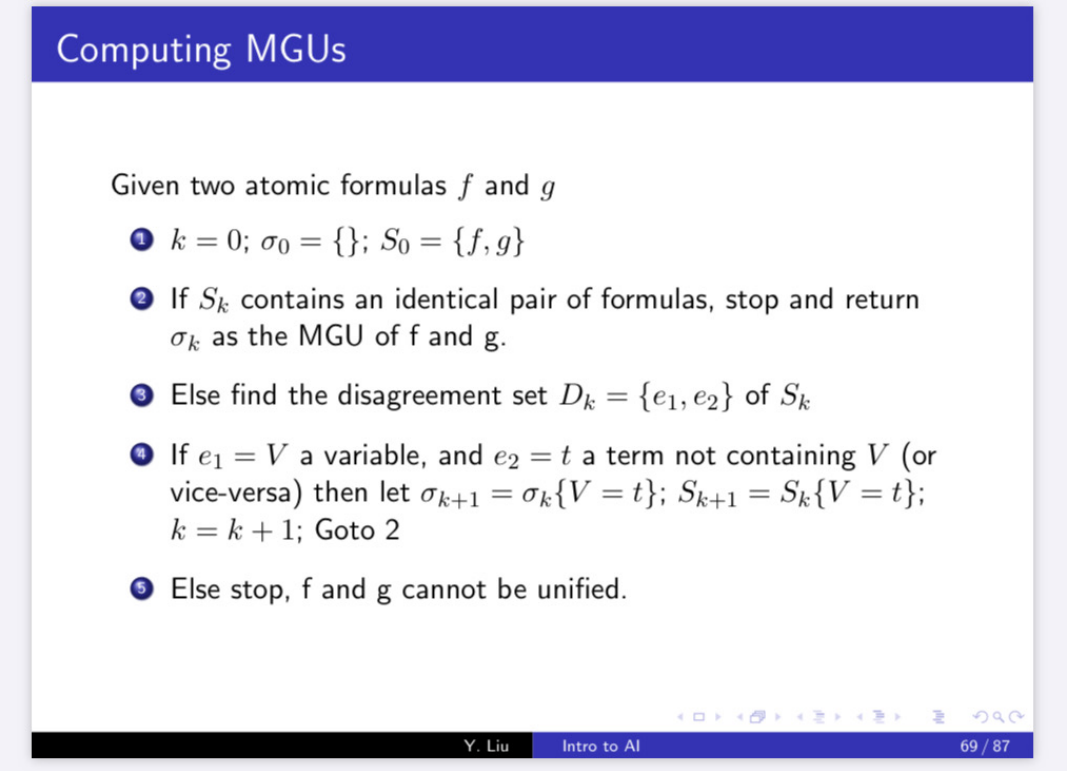
- Implement the MGU algorithm. (10 points)
  1. 运用了链表`Linked_list`来表示σ，链表使用游标实现，且实现了一些方法。

     ```
     # sigma: Linked_list(equation)
     # equation: [str, str]
     ```

  2. 置换组合算法参考课件代码：

  3. MGU算法参考课件代码：



- 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. (10 points)

    1. 术语解释：

        - `clauses`表示全部的子句，是`clause`的容器，逻辑上表示每个`clause`与。
        - `clause`表示单个子句，是`items`的容器，逻辑上表示每个`items`或。
        - `items`表示子句中的一项，是`item`的容器，在此做限定是作用于变量或常量上的谓词。
        - `item`表示谓词的每个部分，`item[0]`是谓词名，`item[1:]`是该谓词作用在的全部变量、常量。
        - 变量：单个字母，在此均设为小写。
        - 常量：至少两个字母，而且保证不包含变量，在此均设为大写。

        ```
        # 假设一个字母是变量，否则是常量，且变量名不包含常量。
        ```

        - σ：置换，见课件置换组合及MGU算法。
        - `equation`：置换中的每一条，见课件置换组合及MGU算法。

    2. 该部分自顶向下分别由`resolve`，`union`，`MGU`三个函数组成。

        - `resolve`用于在`clauses`中选择调用`union`进行合并得到新的`clause`，并判断是否是空，如果为空则返回`False`，结束归结，找到结果是最优解。
        - `union`判断两个`clause`能否合并，如果可以合并，找到相反的`items`，调用`MGU`，得到最一般和一，从而合并两个`clause`，如果无法合并则返回`None`。
        - `MGU`用来找两个`items`的最一般和一，详见课件算法。

- Explain any ideas you use to improve the search efficiency. (5 points)

    1. 数据结构选择：

- `clauses`、`clause`的数据结构最初使用的也是链表。
- 但是最终的`resolve`实现版本，`clauses`中的`clause`无需频繁增删，因而改用`Array`。
- 而`clause`中的`items`尽管需要增删但是频率不高（实际上只有`union`最终合并`clause`之后才会在新`clause`中删除相同的`items`），而且进入`union`的时候要先对`clause`进行拷贝，链表尽管复杂度不高但是拷贝效率低（对空`Linked_list`使用`extend`操作进行深拷贝），因而也改用`Array`。

```
# clauses: Array(clause)
# clause: Array(item)
```

- `Array`包括一个用于存值的数组和一个表记录该数值是否存在，因而删除操作仅是将表中的该项改成`False`。
- `items`用`list`，因为无需增删所以不使用链表，而且需要改其中的`item`所以不用`tuple`而用可变对象`list`。
- `item`用`str`。

```
# f/g/items: list(item=str())
```

- `sigma`用链表`Linked_list`，考虑计算置换组合的三个步骤需要大量增删。

```
# sigma: Linked_list(equation)
```

- `equation`用`[左，右]`的`list`，考虑置换组合的三个步骤需要改变`equation`中第二项的值。

```
# equation: [str, str]
```

2. 由于`resolve`需要处理的数据最多（考虑`clauses`中无用的`clause`可能很多），所以要进行优化。
   - 由于要最优解，所以需要使用UCS算法。路径开销的定义见下。
   - `tree`用来记录路径和路径开销，`tree`的数据结构是`Tree`，是通过记录每个节点的两个孩子从而构建的二叉树。它有主要方法如下：
     1. `append`方法，接受两个孩子节点的下标`a`，`b`作为参数，来添加一个新的父节点。
     2. `is_sub`方法，接受两个节点的下标`a`，`b`，判断`a`树是不是`b`树的子树。
     3. `Size`方法，接受一个节点的下标`a`，返回`a`树的大小（树大小动态计算，存在`size`数组中，所以`Size`方法复杂度$O(1)$）。
   - `tree`存储所有`clause`的逻辑关系，如果`clause1`，`clause2`归结得到`clause3`，则`clause3`为`clause1`、`clause2`的父节点。
   - 得到`clause3`路径定义是以`clause3`为根的子树。
   - 得到`clause3`路径开销定义是$\text{Size}(clause1)+\text{Size}(clause2)$。考虑每两条边表示一次归结，一共有$\text{Size}(clause3)-1=\text{Size}(clause1)+\text{Size}(clause2)$条边，目标是归结次数最少，因而每次拓展$\text{Size}(clause1)+\text{Size}(clause2)$最小。
   - `is_sub`方法可以用于剪枝，考虑3 blocks问题，有`clauses`：

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

1和5归结出6，5和6还可以继续归结，会重复得到1。`is_sub`可以一定程度上避免循环论证的过程（条件推结论，再拿结论推条件）。

- `based_on`是存"得到一个`clause`所基于的全部初始`clause`的可重复集合"的`list`。
- "得到一个`clause`所基于的全部初始`clause`可重复集合"使用的`set_repeated`，是基于`dict`实现的可以重复的集合（`{key:counting}`），有并集操作（重载`+`），比较操作（重载`==`），长度操作（重载`len`）。其中长度是动态地记录，所以`len`的复杂度是$O(1)$。长度操作可以用来记录路径开销`tree`类似，复杂度类似，但是代码里没用到。
- `based_on`的另一个用途是用来剪枝，考虑如果`clause1`，`caluse2`归结得到的`clause3`与历史上得到的`clause0`使用的每个初始`clause`的数目均相同（在UCS算法中是可能的），则`clause3`是等于`clause0`的。这个可以提前通过`based_on[index_of_clause1] + based_on[index_of_clause2] in based_on`来判断，从而可能无需对`clause1`，`clause2`使用计算时间长的`union`。
- `resolve`虽然使用了USC算法，但是算法中寻找路径开销最小的部分，比较特别，这个问题是这样的：
    1. 由于USC每次都将新`clause`放在`clauses`末端，所以`clauses`是一个路径开销不严格递增的序列。
    2. 由新状态路径开销的定义，需要从`clauses`中找到路径开销和最小的两个`clause`。
    3. 由于一些剪枝的限制，真的路径开销和最小不一定取到，而是应取到满足约束的最小路径开销和。
- 因此这个问题就转化成了在一个非严格递增序列中，找到两个数的下标，使得这两个数的下标满足约束，而且这两个数的和最小。这里使用的算法是这样的：
    1. 用`left_bound`，`right_bound`规定我想找的更优的两个数下标所在范围，`left_bound`设为最小下标`1`，`right_bound`设为最大下标`len(clauses) - 1`（减1的原因在于python数组不是真的以1开头，这里1开头实际上是在0的位置加入了一个无效值）。
    2. `index`从`left_bound + 1`遍历到`right_bound`。
        - 如果中途找到`left_bound`和`index`是满足约束的，则停止遍历。设置$sum{max} = \max(sum{max}, cost[left_bound] + coct[index])$。设置`right_bound`为$index - 1$。
        - 如果遍历正常终止，则`left_bound`加1。
    3. 继续回到1，直到`left_bound == right_bound`。
- 这个算法基于的事实是，在非严格递增的数组中找到了符合约束的`a`和`b`，而且`a`之前都已经考虑过，如果想要找到更优的只能从`a`和`b`之间找两个数。
- 最后一步重要的优化是`min_spouse`，用来改进上述算法。在这个问题下，是每次更新`clauses`后都要跑一次上述算法，因而每次运行，都会重复判断一遍上次运行判断的节点，这是不必要的。因而用`min_spouse`记录每个`left_bound`这次还需判断的第一个`index`。每个`min_spouse[left_bound]`初始化为`left_bound + 1`，上述算法改成：

1. 用`left_bound`，`right_bound`规定我想找的更优的两个数下标所在范围，`left_bound`设为最小下标`1`，`right_bound`设为最大下标`len(clauses) - 1`（减1的原因在于python数组不是真的以1开头，这里1开头实际上是在0的位置加入了一个无效值）。
2. `index`从`min_spouse[left_bound]`遍历到`right_bound`。
    - 如果中途找到`left_bound`和`index`是满足约束的，则停止遍历。设置$sum{max} = \max(sum{max}, cost[left_bound] + coct[index])$。设置`right_bound`为$index - 1$。
    - 如果遍历正常终止，则`left_bound`加1。
3. 遍历终止后，设置`min_spouse[left_bound]`为`index`（考虑循环如果是找到满足约束终止，当前`index`是找到的满足约束的下标，由于不一定是最优，下一次运行还要继续考虑；如果是正常终止，当前`index`是`right_bound + 1`，表示`right_bound`及以前的都考虑过，下一次运行不用考虑。），继续回到1，直到`left_bound == right_bound`。

- Run your system on the examples of hardworker(sue), 3-blocks, Alpine Club. Include your input and output files in your report. (15 points)

  结果见后。

- What do you think are the main problems for using resolution to check for satisfiability for a set of first-order clauses? Explain. (10 points)

  1. 不满足问题可能开销很大，因为要归结至不再出现新的`clause`才会返回`True`。另外我当前算法没有设置环路检测，因为`clause`结构复杂，环路检测开销大。

2. Codes

1.GAC

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <list>
#include <set>
#include <string>
#include <vector>
using namespace std;

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

/* 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的绝对值为列约束。行列约束的索引均从1开始
    pair<pair<int, int>, pair<int, int>> inequality;  // 不等式约束，first>second，索引从0开始
    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();  // check whether the puzzle is solved
    void printPuzzle();
    DOMAINS makeDomains(list<Constraint>& gac_queue);          // 初始化每个变量的域
    pair<int, int> chooseUnassigned(const DOMAINS& domains);  // choose a
unassigned variable with minimum remaining values

    void pushConstraintIntoQueue(pair<int, int> pos, list<Constraint>&
gac_queue);  // 将与变量pos相关的所有约束加入队列
    PUZZLE gac(const DOMAINS& domains, list<Constraint>& gac_queue);
   // GAC外层
    DOMAINS gacEnforce(DOMAINS domains, list<Constraint>& gac_queue);
    // GAC enforce

   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
};

/*
    Read the puzzle and constraints from files to numpy matrices,
    and convert the coordinates into 0-indexed (coordinates in the
    file are 1-indexed).
*/
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 - 1, y1 - 1),
make_pair(x2 - 1, y2 - 1))));
    }
    puz_file.close();
    con_file.close();
}

/* Check whether all cells in the puzzle are filled. */
```

```cpp
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;
}

/* 打印整个棋盘 */
void Futoshiki::printPuzzle() {
    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变量 */
pair<int, int> Futoshiki::chooseUnassigned(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 */
PUZZLE Futoshiki::gac(const DOMAINS& domains, list<Constraint>& gac_queue) {
    if (domains.size() == 0) return PUZZLE();
    if (isSolved()) {
        return puzzle;  // 找到解
    }
    pair<int, int> pos = chooseUnassigned(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
        auto temp_domains = domains;          // 在temp_domains上修改。（注意，若在原
domains上修改会导致迭代器失效）
        temp_domains[pos.first][pos.second].clear();
        temp_domains[pos.first][pos.second].insert(*pd);
        pushConstraintIntoQueue(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();                    // 返回无解
```

```
}

/* 把与变量pos相关的不等式约束不重复地加入队列 */
void Futoshiki::pushConstraintIntoQueue(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) {
               // 与该变量相关的约束
            if (find(gac_queue.begin(), gac_queue.end(), constraints[i]) ==
gac_queue.end()) {  // 不重复加入
                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()) {  // 不重复加入
        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()) {  // 不重复加入
        gac_queue.push_back(col_constraint);
    }
}

DOMAINS Futoshiki::makeDomains(list<Constraint>& gac_queue) {
    // initialize
    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 ij = 1; ij <= size; ij++) {
        gac_queue.push_back(Constraint(ij));   // 第ij行
        gac_queue.push_back(Constraint(-ij));  // 第ij列，注意列用负数表示
    }
    for (int i = 0; i < constraints.size(); i++) {
        gac_queue.push_back(constraints[i]);
    }
    return gacEnforce(domains, gac_queue);  // 首次执行GAC enforce
}
```

```
/* GAC enforce */
DOMAINS Futoshiki::gacEnforce(DOMAINS domains, list<Constraint>& gac_queue) {
    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> large_pos = constraint.inequality.second;
            pair<int, int> small_pos = constraint.inequality.first;
            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
                    pushConstraintIntoQueue(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
                    pushConstraintIntoQueue(small_pos, gac_queue);
                }
            }
        }
        // 行约束，以下方法已经过优化以提高效率
        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
                                pushConstraintIntoQueue(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
                                pushConstraintIntoQueue(make_pair(i1, j),
gac_queue);
                            }
                        }
                    }
                }
            }
        }
    }
    return domains;
}

int main(int argc, char* argv[]) {
    if(argc < 2) {
        cerr << "[-] Usage: ./futoshiki_gac <test_id>" << endl;
        cerr << "test_id is a integer from 0 to 5." << endl;
        return -1;
    }
    Futoshiki game(string("tests/puzzle")+argv[1]+".txt",
string("tests/constraints")+argv[1]+".txt");
    list<Constraint> gac_queue;
    auto domains = game.makeDomains(gac_queue);  // 初始化各变量的取值域并首次执行
gacEnforce
    PUZZLE result = game.gac(domains, gac_queue);

    if (result.size() != 0) {
        cout << "Solution found:" << endl;
        game.printPuzzle();
    } else {
        cout << "[-] No solution!" << endl;
    }
    return 0;
}
```

2.resolution

```
class Array():
    def __init__(self):
        self.space = list()
        self.table = list()
        self.length = 0

    def append(self, value):
        self.space.append(value)
        self.table.append(True)
        self.length += 1

    def pop(self):
        self.table.pop()
```

```
        self.length -= 1
        return self.space.pop()

    def Length(self):
        return self.length

    def restore(self, index):
        if not self.table[index]:
            self.table[index] = True
            self.length += 1

    def __getitem__(self, index):
        if self.table[index] == False:
            return None
        else:
            return self.space[index]

    def __delitem__(self, index):
        if self.table[index]:
            self.table[index] = False
            self.length -= 1

    def __len__(self):
        return len(self.space)

    def __add__(self, array):
        new = Array()
        for i in range(len(self.space)):
            if self.table[i]:
                new.append(self.space[i])
        for i in range(len(array)):
            if array[i] is not None:
                new.append(array[i])
        return new


class Clauses(Array):
    pass


class Clause(Array):
    def __init__(self, index):
        super(Clause, self).__init__()
        self.index = index

    def Index(self):
        return self.index

    def deepcopy(self):
        new = Clause(self.index)
        for i in range(len(self.space)):
            if self.table[i]:
                new.append(copy.deepcopy(self.space[i]))
        return new

    def __add__(self, clause):
        global clause_index
        new = Clause(clause_index)
```

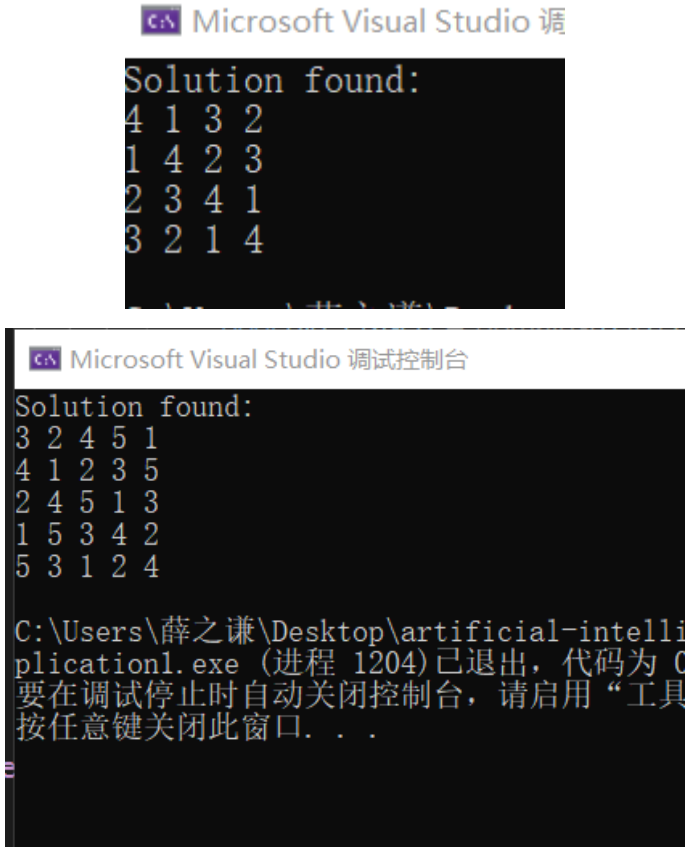
```
        for i in range(len(self.space)):
            if self.table[i]:
                new.append(self.space[i])
        for i in range(len(clause)):
            if clause[i] is not None:
                for j in range(len(self.space)):
                    if self.table[j]:
                        if self.space[j][0] == clause[i][0]:
                            for k in range(1, len(self.space[j])):
                                if self.space[j][k] != clause[i][k]\
                                    and not (len(self.space[j][k]) == 1
                                            and len(clause[i][k]) == 1):
                                    break
                            else:
                                break
                else:
                    new.append(clause[i])
        return new

    def Dump(self):
        l = list()
        for i in range(len(self.space)):
            if self.table[i]:
                l.append(self.space[i])
        return l


class Tree():
    def __init__(self):
        self.space = [()]
        self.value = [None]
        self.size = [0]

    def append(self, value, a, b):
        self.space.append((a, b))
        self.size.append(self.size[a] + self.size[b] + 1)
        self.value.append(value)

    def Size(self, a):
        return self.size[a]

    def Value(self, a):
        return self.value[a]

    def Children(self, a):
        return self.space[a]

    def is_sub(self, a, b):
        if a == b:
            return True
        if b == 0:
            return False
        return self.is_sub(a, self.space[b][0]) or self.is_sub(a, self.space[b]
[1])

    def __contains__(self, a):
        return a in self.space
```

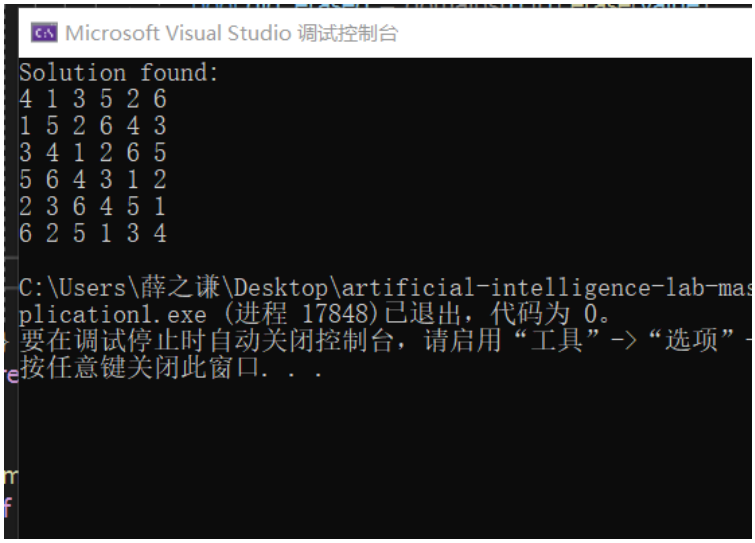
```
    def Back(self):
        return len(self.space) - 1


class set_repeated():
    def __init__(self, dict2):
        self.dict = dict(dict2)
        self.length = 0
        for key, value in dict2.items():
            self.length += value

    def Dict(self):
        return self.dict

    def __add__(self, set2):
        dict3 = copy.deepcopy(self.dict)
        for key, value in set2.Dict().items():
            if dict3.get(key) is None:
                dict3[key] = value
            else:
                dict3[key] += value
        return set_repeated(dict3)

    def __eq__(self, set2):
        return self.dict == set2.Dict()

    def __len__(self):
        return self.length()


def union(clause1_constant, clause2_constant):
    clause1 = clause1_constant.deepcopy()
    clause2 = clause2_constant.deepcopy()
    index1 = 0
    for i in range(len(clause1)):
        if clause1[i] is None:
            continue
        f = clause1[i]
        index2 = 0
        for j in range(len(clause2)):
            if clause2[j] is None:
                continue
            g = clause2[j]
            if f[0] == '¬' + g[0]\
                    or '¬' + f[0] == g[0]:
                sigma = MGU(f[1:], g[1:])
                if sigma is None:
                    return None, None, None
                pointer = sigma.Header()
                while pointer != sigma.Tail():
                    V, t, V_in = sigma[pointer]
                    if V_in == 'f':
                        for k in range(len(clause1)):
                            if clause1[k] is None:
                                continue
                            for index in range(len(clause1[k][1:])):
                                clause1[k][index +
```

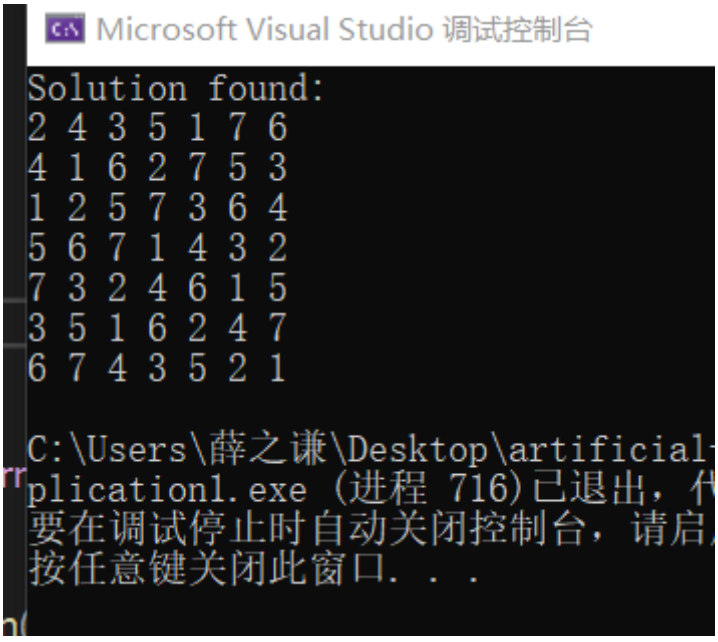
```
                                            1] = clause1[k][index + 1].replace(V,
t)
                    else:
                        for k in range(len(clause2)):
                            if clause2[k] is None:
                                continue
                            for index in range(len(clause2[k][1:])):
                                clause2[k][index +
                                           1] = clause2[k][index + 1].replace(V,
t)
                    pointer = sigma.Next(pointer)
                del clause1[i]
                del clause2[j]
                return clause1 + clause2, chr(ord('a') + index1), chr(ord('a') +
index2)
            index2 += 1
        index1 += 1
    return None, None, None


def resolve(clauses, tree, based_on, min_spouse):
    global clause_index
    while True:
        if clauses.Length() == 0:
            return False
        left_bound = 1
        right_bound = len(clauses) - 1
        min_sum = float('inf')
        min_clause1_index = None
        min_clause2_index = None
        min_clause3 = None
        min_items_index1 = None
        min_items_index2 = None
        while left_bound < right_bound:
            if clauses[left_bound] is None:
                left_bound += 1
                continue
            clause1 = clauses[left_bound]
            index = min_spouse[left_bound]
            while index <= right_bound:
                if clauses[index] is None:
                    index += 1
                    continue
                if based_on[left_bound] + based_on[index] in based_on:
                    index += 1
                    continue
                if tree.is_sub(left_bound, index):
                    index += 1
                    continue
                clause2 = clauses[index]
                clause3, items_index1, items_index2 = union(
                    clause1, clause2)
                if items_index1 is None:
                    index += 1
                    continue
                sum_base = tree.Size(left_bound) + tree.Size(index)
                if sum_base < min_sum:
                    min_sum = sum_base
```

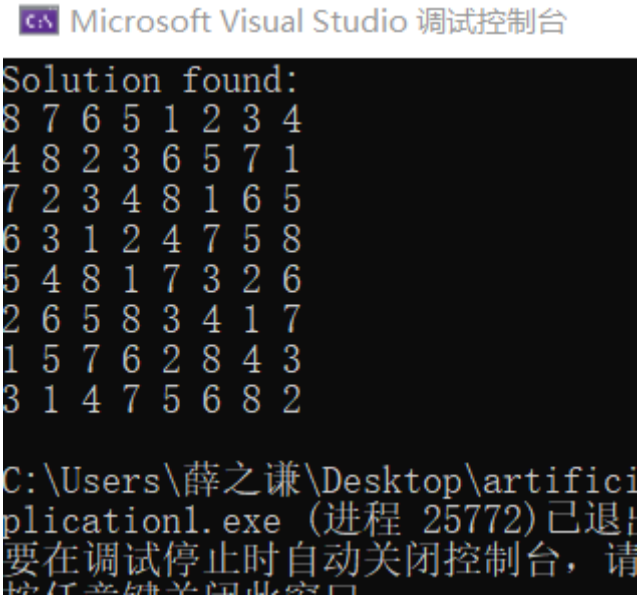
```
                min_clause1_index = left_bound
                min_clause2_index = index
                min_clause3 = clause3
                min_items_index1 = items_index1
                min_items_index2 = items_index2
            right_bound = index - 1
        min_spouse[left_bound] = index
        left_bound += 1
    if min_items_index1 is None:
        return True
    tree.append((min_items_index1, min_items_index2),
                min_clause1_index, min_clause2_index)
    clause_index += 1
    based_on.append(based_on[min_clause1_index] +
                    based_on[min_clause2_index])
    clauses.append(min_clause3)
    min_spouse.append(clause_index)
    if min_clause3.Length() == 0:
        return False
```

3.结果展示




Microsoft Visual Studio 调试控制台
Solution found:
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
C:\Users\薛之谦\Desktop\artificial-intelligence-lab-mas
plication1.exe (进程 17848)已退出，代码为 0。
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
Solution found:
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
C:\Users\薛之谦\Desktop\artificial-
plication1.exe (进程 716)已退出，代
要在调试停止时自动关闭控制台，请启
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
Solution found:
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
C:\Users\薛之谦\Desktop\artifici
plication1.exe (进程 25772)已退
要在调试停止时自动关闭控制台，请

```
❯ python resolution.py
1       [['GradStudent', 'SUE']]
2       [['¬GradStudent', 'x'], ['Student', 'x']]
3       [['¬Student', 'y'], ['HardWorker', 'y']]
4       [['¬HardWorker', 'SUE']]
5       R('3b', '4a')[['¬Student', 'SUE']]
6       R('2b', '5a')[['¬GradStudent', 'SUE']]
7       R('1a', '6a')[]
Time cost: 0.0 s
```

```
❯ python resolution.py
1       [['On', 'AA', 'BB']]
2       [['On', 'BB', 'CC']]
3       [['Green', 'AA']]
4       [['¬Green', 'CC']]
5       [['¬On', 'x', 'y'], ['¬Green', 'x'], ['Green', 'y']]
6       R('3a', '5b')[['¬On', 'AA', 'y'], ['Green', 'y']]
7       R('2a', '5a')[['¬Green', 'BB'], ['Green', 'CC']]
8       R('7a', '6b')[['Green', 'CC'], ['¬On', 'AA', 'BB']]
9       R('4a', '8a')[['¬On', 'AA', 'BB']]
10      R('1a', '9a')[]
Time cost: 0.00402379035949707 s
```

```
❯ python resolution.py
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('8b', '11c')[['L', 'w', 'SNOW'], ['¬A', 'w'], ['¬C', 'w']]
13      R('6b', '8b')[['¬A', 'z'], ['C', 'z'], ['L', 'z', 'SNOW']]
14      R('13b', '12c')[['¬A', 'w'], ['L', 'w', 'SNOW']]
15      R('5a', '9a')[['¬L', 'MIKE', 'SNOW']]
16      R('15a', '14b')[['¬A', 'MIKE']]
17      R('2a', '16a')[]
Time cost: 0.22440195083618164 s
```

4.Experimental experience

- 最开始做了很多尝试，包括不同数据结构，不同搜索算法（尝试过深度优先），不同剪枝策略。最后`min_spouse`的加入使得本来跑几分钟的代码加速到小于1秒。
- 尝试运用了多种加速策略，包括自定义了结合多种数据结构的类，在 GAC 剪枝时使用 depth 标记被减去的值域，在还原时将 depth 还原为 0等等