

华中科技大学

课程设计报告

题目: 基于 SAT 的二进制数独游戏求解程序

课程名称: 程序设计综合课程设计

专业班级: _____

学 号: _____

姓 名: _____

指导教师: 

报告日期: 2020.03.30

计算机科学与技术学院

任 务 书

□ 设计内容

SAT 问题即命题逻辑公式的可满足性问题 (satisfiability problem)，是计算机科学与人工智能基本问题，是一个典型的 NP 完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于 DPLL 算法实现一个完备 SAT 求解器，对输入的 CNF 范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

□ 设计要求

要求具有如下功能：

- (1) **输入输出功能：**包括程序执行参数的输入，SAT 算例 cnf 文件的读取，执行结果的输出与文件保存等。(15%)
- (2) **公式解析与验证：**读取 cnf 算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
- (3) **DPLL 过程：**基于 DPLL 算法框架，实现 SAT 算例的求解。(35%)
- (4) **时间性能的测量：**基于相应的时间处理函数（参考 time.h），记录 DPLL 过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
- (5) **程序优化：**对基本 DPLL 的实现进行存储结构、分支变元选取策略^[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为： $[(t-t_0)/t]*100\%$ ，其中 t 为未对 DPLL 优化时求解基准算例的执行时间， t_0 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)
- (6) **SAT 应用：**将二进制数独游戏^[5] 问题转化为 SAT 问题^[6]，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献[3]与[6-9]。(15%)

□ 参考文献

- [1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社, 2000
- [2] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Masterthesis, Concordia University, Canada, 2009
- [3] 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 硕士学位论文, 电子科技大学, 2011
- [4] Carsten Sinz. Visualizing SAT Instances and Runsof the DPLL Algorithm. JAutom Reasoning (2007) 39:219–243
- [5] Binary Puzzle: <http://www.binarypuzzle.com/>
- [6] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Mathematics in Computer Science, (2017) 11:515–526
- [7] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.
- [8] Ins Lynce and Jo Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.
- [9] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57
- [10] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf

目 录

任务书	I
1 引言	5
1.1 课题背景与意义.....	5
1.1.1 课题背景.....	5
1.1.2 课题意义.....	5
1.2 国内外研究现状.....	6
1.3 课程设计的主要研究工作.....	7
2 系统需求分析与总体设计	8
2.1 系统需求分析.....	8
2.2 系统总体设计.....	8
3 系统详细设计	10
3.1 有关数据结构的定义.....	10
3.1.1 优化前的数据结构定义.....	10
3.1.2 优化后的数据结构定义.....	10
3.2 主要算法设计.....	11
3.2.1 DPLL 递归算法结构.....	11
3.2.2 变元选择策略优化（最短正子句策略和 MOM 策略）	13
3.2.3 DPLL 非递归算法结构（算法结构的优化）	13
3.2.4 二进制数独游戏规则转化（不添加附加变元）	14
3.2.5 二进制数独棋盘求解.....	16
3.2.6 具有唯一解的二进制数独棋盘的随机生成.....	16

4	系统实现与测试	19
4.1	系统实现	19
4.1.1	数据结构	19
4.1.2	函数声明	21
4.2	系统测试	27
4.2.1	SAT 求解器模块	27
4.2.2	二进制数独模块	35
4.2.3	用户交互模块	42
5	总结与展望	46
5.1	全文总结	46
5.2	工作展望	46
6	体会	48
	参考文献	50
	附录 源程序	51

1 引言

1.1 课题背景与意义

SAT 问题即命题逻辑公式的可满足性问题 (satisfiability problem)，指是否存在一组真值指派，使得布尔表达式成立的问题。SAT 问题被证明为 NP 完全问题。本文将介绍求解 SAT 的经典完备算法——DPLL 算法及其优化，基于此设计 SAT 求解器以求解 CNF 公式，并利用这个 SAT 求解器设计一个简单的二进制数独游戏——即具备随机生成一个二进制数独棋盘给用户填写和求解用户提供的二进制数独棋盘两个功能。

1.1.1 课题背景

SAT 问题在人工智能领域的重要地位，使得许多学者都在 SAT 问题求解领域做了大量的研究，可满足性问题进而也成为了国内外研究的热点问题，并在算法研究和技术实现上取得了较大的突破，这也推动了形式验证和人工智能等领域的发展。在 SAT 求解器被越来越多地应用到各种实际问题领域的今天，探寻解决 SAT 问题的高效算法仍然是一个吸引人并且极具挑战性的研究方向。

而二进制数独作为一个放置难题，是一个运用纸和笔进行演算和推理的逻辑游戏，具有开发智力和锻炼逻辑思维能力的优点，深受大家喜爱。

经过一年半的相关课程学习，我们具备进行程序设计、数据结构和算法设计的基本能力，可以进行初级 SAT 求解器的设计，并以此为基础设计一个简单的二进制数独游戏。

1.1.2 课题意义

CNF (Conjunctive Normal Form) 即合取范式，是 SAT 问题中重要的概念。由离散数学相关知识可知，任何命题逻辑公式都可以等价于某一 CNF 公式。因此，求解 CNF 的 SAT 问题对于数理逻辑、计算科学、硬件设计、人工智能以及安全协议验证等学科分支具有重要理论意义与应用价值^[1]。

除此以外，SAT 问题是一个典型的 NP 完全问题。由于 NP 完全问题之间可

在多项式时间内进行互相转化，如果可以找到解决 SAT 问题的高效算法，那么对于其他 NP 完全问题的研究也会有极大帮助，从而生产生活中的大量 NP 问题就可以得到解决。因此找到高效的 SAT 求解器意义重大。

1.2 国内外研究现状

自从 20 世纪 60 年代起，SAT 问题开始倍受学者关注，也有不少学者在这方面做了大量工作，也取得一些重大突破。

20 世纪 60 年代初期，Martin Davis 和 Hilary Putnam 提出了首个解决 SAT 问题的完备算法，称为 DP 算法。其后两年，George Logemann 和 Donald W. Loveland 等人在 DP 算法的基础上提出了 DPLL 算法。20 世纪 70 年代初期，Stephen Arhur. Cook 证明了 SAT 问题是 NP 完全问题。

20 世纪 90 年代末，Joao Marques Silva 等人在 DPLL 算法的基础上提出了 GRASP 算法，首次引入了冲突学习回溯策略。进入 21 世纪之后，Bart Selman 和 Henry Kautz 在仍智能国际合作会议上指出当前 SAT 问题面临的十大挑战性问题。随后，Niklas Een 等人提出的 SatEliteE 首次实现了预处理简化问题的规模和复杂性。之后，Bart Selman 和 Henry Kautz 对当前 SAT 问题的现状做了全面的整理和叙述。

目前，求解 SAT 问题的算法大致分为不完备算法和完备算法两大类。

不完备算法的主要基于局部搜索思想。对于一个特定的 SAT 问题，不完备算法先是给出一组真值指派，如果这组真值指派使得原问题得到满足，则直接得到该问题的一个满足解，同时搜索结束；如果这组真值指派不能使原问题得到满足，则反转其中某一个变量的赋值，或者反转次数达到阈值，这种情况下求解器无法判定原有问题的可满足性，同时搜索结束。

完备算法主要基于回溯搜索思想，对于可满足问题，完备算法一定能给出问题的可满足解，对于不可满足问题，可以给出不可满足的证明过程^[2]。DPLL 算法就是一个典型的完备算法，本文将基于 DPLL 算法及其优化来解决 SAT 问题并基于此设计二进制数独游戏。

1.3 课程设计的主要研究工作

本课程设计主要根据命题逻辑可满足性的相关理论知识, 基于 DPLL 的算法和基本框架, 设计一个 SAT 问题求解器, 并对基本 DPLL 的分支变元选取策略进行优化设计与实现, 也对数据结构和算法结构进行了优化。除此以外, 本课程设计还分析研究了与二进制数独游戏规则等价的命题公式, 并根据离散数学知识对命题公式进行等价推导, 从而在不引入附加变元的前提下得到与二进制数独游戏规则等价的一组 CNF 公式, 并由这一组 CNF 公式出发, 对二进制数独进行求解以及生成一个具有唯一解的二进制数独棋盘。

具体来说, 本文的主要研究内容如下:

1.学习了命题逻辑可满足性问题的基本理论知识, 对 DPLL 算法和基本框架进行了深入分析, 从逻辑上设计了针对 SAT 问题合理的数据结构。

2.对基于 DPLL 算法和基本框架的 SAT 求解器进行了实现, 并对系统的数据结构进行改进, 同时也针对 DPLL 的分支变元选取策略进行多种优化, 并给出数据结构的优化以及变元选择策略优化对求解 SAT 问题的优化效率。

3.通过对二进制数独游戏规则的研究以及离散数学知识的运用, 得到了一组与二进制游戏规则等价的 CNF 公式, 并通过编程将这一组 CNF 公式进行实现, 并在这一组 CNF 公式的基础上对二进制数独问题进行求解。

4.通过 SAT 求解器随机生成一个完整的二进制数独棋盘, 深入分析了如何通过挖洞法生成具有唯一解的二进制数独棋盘, 并对此进行实现。

2 系统需求分析与总体设计

2.1 系统需求分析

对于 SAT 问题，用户主要需求为求解一个特定的 CNF 公式，用户希望在输入这个扩展名为.cnf 的文件之后，得到该 CNF 公式的以下信息：可满足性、若可满足给出一组解。并将以上信息以特定的格式写入到一个与 cnf 文件同名的、扩展名为.res 的文件当中。

对于二进制数独游戏，用户有两大需求。其一，对于用户无法解决的二进制数独问题，系统予以解决；其二，用户可进行二进制数独游戏，即系统随机生成一个二进制数独棋盘，用户填写，系统判定正误，同时可显示答案。显然，用户希望该系统具有一定的交互性、易操作性以及稳定性。

2.2 系统总体设计

该系统具有两大主要板块，分别是基于 DPLL 基本框架的 SAT 求解器板块和二进制数独求解与交互板块。对于这两大板块，各自具有的模块如下。

1.基于 DPLL 基本框架的 SAT 求解器板块：CNF 文件的读取、CNF 文件输出校验、SAT 问题求解以及算例结果输出。

2.二进制数独求解与交互板块：求解用户提供的二进制数独棋盘、随机生成一个完整的二进制数独棋盘，挖洞法得到一个具有唯一解的二进制数独棋盘以及二进制数独游戏操作。

基于课程设计和实验要求设计得到的上述两大板块以及各自具有的二级模块，整个系统的程序流程图见下图 2-1。

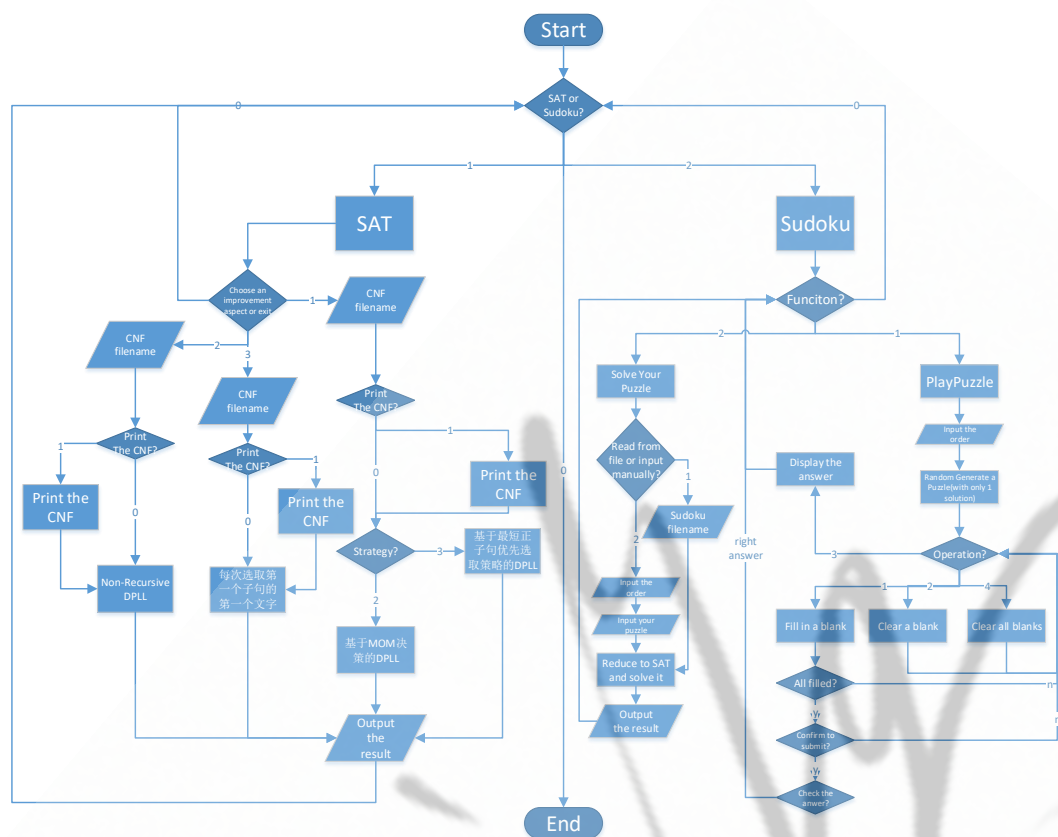


图 2-1 实验系统程序流程图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 优化前的数据结构定义

一个CNF公式由很多析取构成的子句(Clause)构成，构成子句的每个变元称为文字(Literal)。对于一个CNF公式的数据结构（如图3-1所示），子句结点由一个指向下一子句的指针和指向子句第一个文字结点的指针构成。文字结点由一个指向下一个文字结点的指针域和一个表示文字的数据域组成。指向第一个子句结点的根结点标记为cnf，构成一个CNF公式。

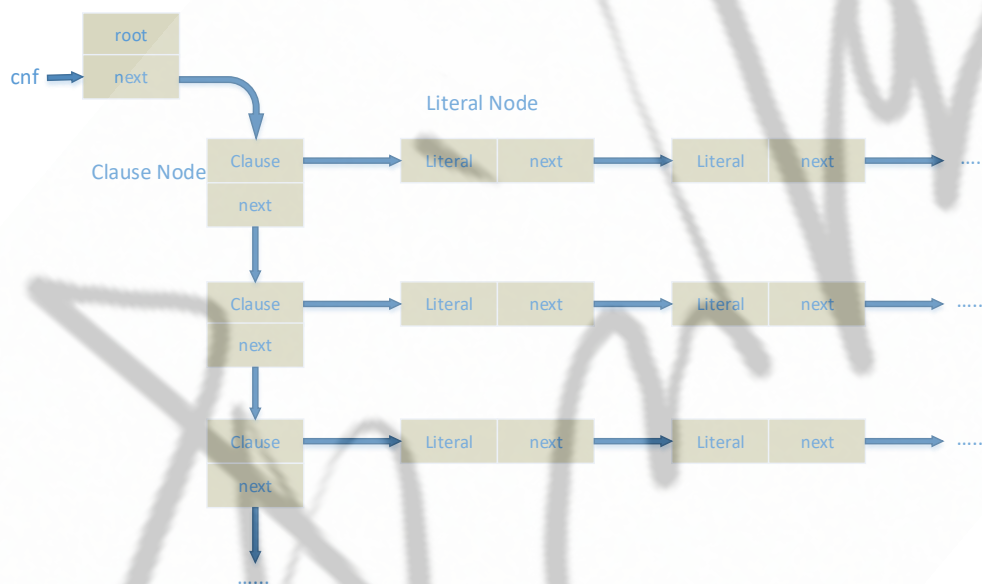


图 3-1 优化前的 CNF 公式数据结构图

3.1.2 优化后的数据结构定义

优化后的数据结构也是采用链表的形式，链表的结构和优化前链表的结构类似，但是吸收了“懒散”的思想，即用一个变量标记这个变元的状态，要么被满足、要么被删除、要么待定。通过这种优化，避免了布尔约束推导过程中繁琐的 CNF 公式复制过程，不仅降低了时间复杂度，更重要的是，显著降低了算法的空间复杂度，从根本上解决了内存溢出的问题。除此以外，优化后的数据结构增添了索引表，通过这个索引表可以直接定位到所有包含某个特定变元的子句，也可以降低布尔约束推导过程的复杂度。优化后的 CNF 公式数据结构图见图 3-2。

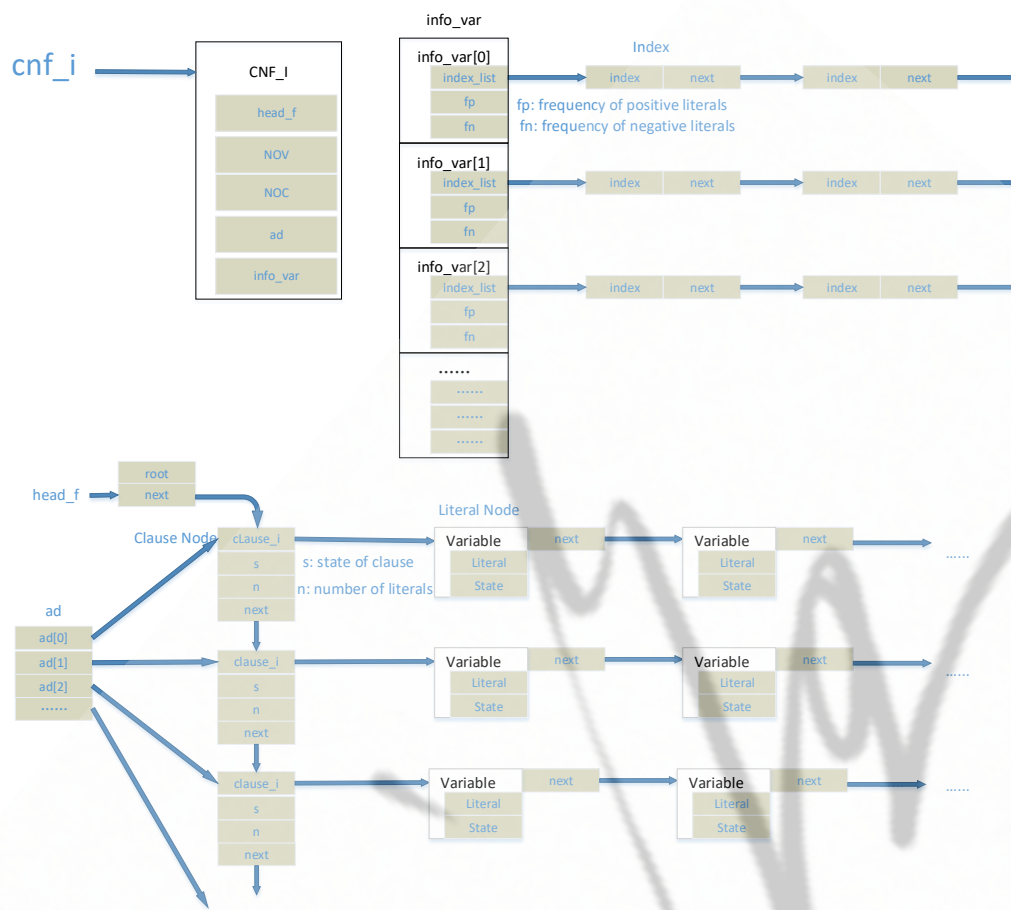


图 3-2 优化后的数据结构示意图

3.2 主要算法设计

3.2.1 DPLL 递归算法结构

DPLL 算法以回溯法为基本思想，以布尔约束推导(Boolean Constraint Propagation)为基本原理，即反复使用单子句规则和分裂策略。其算法描述见图 3-3。

Algorithm 1 DPLL

Require: A Conjunctive Normal Formula S

Ensure: Satisfiability of S

```

function DPLL( $S$ )
  while Unit Clause Exist in  $S$  do
    choose a unit clause in  $S$ 
    simplify  $S$  through Boolean Constraint Propagation
    if  $S = \emptyset$  then
      return TRUE
    else if Null Clause Exist in  $S$  then
      return FALSE
    end if
  end while
  choose a variable  $v$  through a certain strategy
  if DPLL( $S \cup v$ ) then
    return TRUE
  end if
  return DPLL( $S \cup \neg v$ )
end function
    
```

图 3-3 DPLL 算法描述

DPLL 算法的流程图见图 3-4。

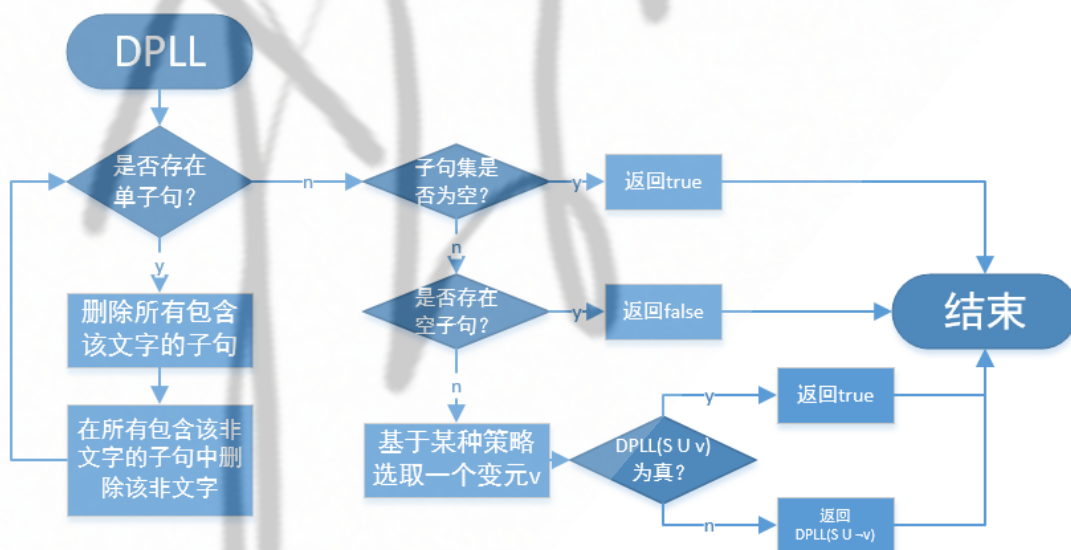


图 3-4 DPLL 算法流程图

3.2.2 变元选择策略优化（最短正子句策略和 MOM 策略）

优化前的变元选取策略采取随机思想，即选取第一个子句的第一个文字，但这种随机策略没有利用到 CNF 公式的特点，即越短的子句越难以满足，应使其优先满足。基于这一特点，可以使用最短正子句优先策略以及 MOM 策略（JW 算法）进行优化。

最短正子句优先策略指，在所有全部为正文字的子句中选取最短的那一个子句的第一个文字；MOM 策略则更为精确，利用公式

$$JW(v) = \sum 2^{-length(C_i)} \quad (for \ any \ C_i: v \in C_i) \quad (3-1)$$

计算出每一个变元的 JW 函数值，则选取 JW 函数值最大的那一个文字。

3.2.3 DPLL 非递归算法结构（算法结构的优化）

由于函数调用过程中需要保存寄存器的状态，因而在递归算法中存在大量的出栈入栈操作，因此考虑算法结构的优化，即递归结构向非递归结构的转化。非递归结构的 DPLL 算法的流程图见图 3-5。

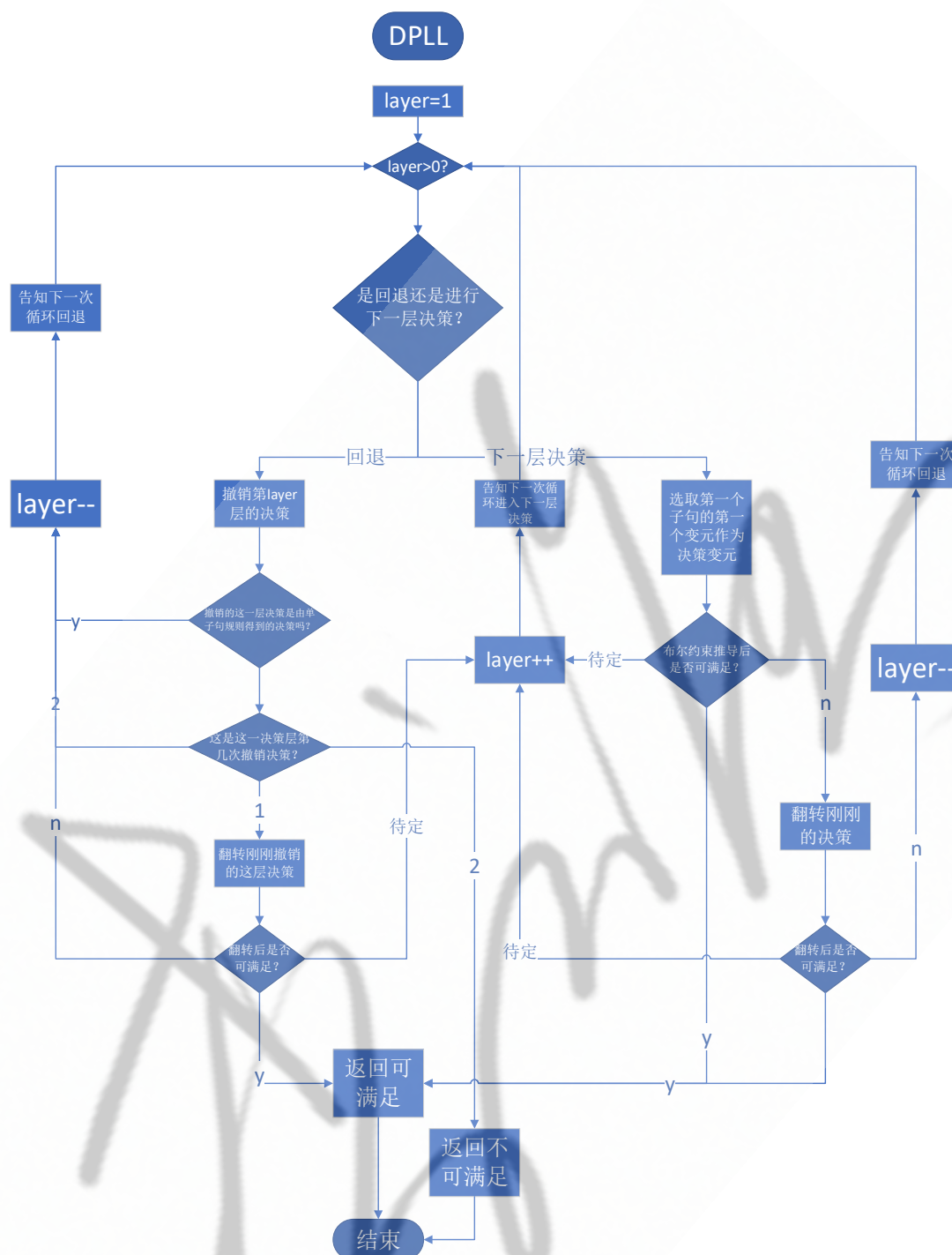


图 3-5 非递归结构的 DPLL 算法流程图

3.2.4 二进制数独游戏规则转化（不添加附加变元）

对于二进制数独游戏，一共有三条规则，这三条规则可以在不添加附加变元的前提下转化为一个CNF公式，且这个CNF公式具有普适性，即任意一个二进制棋盘都必定满足这个CNF公式。下面以6阶二进制数独游戏为例简要说明将规则

转化为CNF公式的方法。

该游戏共有36个单元，每个单元对应一个布尔变元，并用该单元的行号与列号两位数进行表示。如24表示第2行第4列单元对应的布尔变元，取真值时表示该单元填数字1；取假值时表示该单元填数字0。

规则一，不得有3个连续的0或者3个连续的1。则对于任意3个相邻的单元所对应的3个布尔变元 v_1 、 v_2 和 v_3 都必须满足

$$v_1 \vee v_2 \vee v_3 \quad (3-2)$$

$$\neg v_1 \vee \neg v_2 \vee \neg v_3 \quad (3-3)$$

对于一个6阶的二进制数独棋盘，这条规则共产生 $2 * 4 * 2C_6^1 = 96$ 条子句。

规则二，每一行和每一列中所填入的0和1的个数相等。根据鸽巢原理，对于一个 $2n$ 阶的二进制数独棋盘而言，就是说在任意一行或者任意一列中任取 $n+1$ 个单元，其中必然既有0，又有1。则在一个6阶二进制数独棋盘中的任意一行（列）中任取4个单元，其所对应的四个布尔变元都必然满足

$$v_1 \vee v_2 \vee v_3 \vee v_4 \quad (3-4)$$

$$\neg v_1 \vee \neg v_2 \vee \neg v_3 \vee \neg v_4 \quad (3-5)$$

对于一个6阶的二进制数独棋盘，这条规则共产生 $2C_6^4 * 2C_6^1 = 360$ 条子句。

规则三，没有完全相同的两行，也没有完全相同的两列。以第5行和第6行为例，必须满足：

$$\neg\{[(51 \wedge 61) \vee (\neg 51 \wedge \neg 61)] \wedge [(52 \wedge 62) \vee (\neg 52 \wedge \neg 62)] \wedge \dots \wedge [(56 \wedge 66) \vee (\neg 56 \wedge \neg 66)]\} \quad (3-6)$$

根据离散数学的知识，上面的公式如果转换成一个CNF公式，则包含以下子句：

$$51 \vee 61 \vee 52 \vee 62 \vee \dots \vee 56 \vee 66 \quad (3-7)$$

$$51 \vee 61 \vee 52 \vee 62 \vee \dots \vee \neg 56 \vee \neg 66 \quad (3-8)$$

.....

.....

$$51 \vee 61 \vee \neg 52 \vee \neg 62 \vee \dots \vee 56 \vee 66 \quad (3-9)$$

$$51 \vee 61 \vee \neg 52 \vee \neg 62 \vee \dots \vee \neg 56 \vee \neg 66 \quad (3-10)$$

.....

.....

.....

$$\neg 51 \vee \neg 61 \vee \neg 52 \vee \neg 62 \vee \dots \vee 56 \vee 66 \quad (3-11)$$

$$\neg 51 \vee \neg 61 \vee \neg 52 \vee \neg 62 \vee \dots \vee \neg 56 \vee \neg 66 \quad (3-12)$$

对于一个二进制数独棋盘，这条规则共产生 $2C_6^2 * 2^6 = 1920$ 条子句。

综上所述，一个6阶二进制数独棋盘的游戏规则若在不添加附加变元的前提下转化成一个CNF公式，则这个CNF公式共含有 $96 + 360 + 1920 = 2376$ 条子句。依次类推，一个 $2n$ 阶二进制数独棋盘的游戏规则在不添加附加变元的前提下转化成的CNF公式共含有 $8nC_{2n}^{n+1} + (2n^2 - 1)2^{2n+1} + 16n^2 - 16n$ 条子句。

3.2.5 二进制数独棋盘求解

在将二进制数独游戏规则转化成一个CNF公式之后，只需要将棋盘中的已给出的数字以单变元子句的形式添加到上述转化出来的CNF公式中，就得到了和这个棋盘完全等价的一个CNF公式，通过以上描述的DPLL算法对这个CNF公式进行求解，就得到了这个二进制数独棋盘的解。

3.2.6 具有唯一解的二进制数独棋盘的随机生成

对于一个空白的 $2n$ 阶二进制数独游戏棋盘，随机选取 $2n+2$ 个单元随机填入1或0，再通过上述二进制数独棋盘求解过程判断这个棋盘是否可满足，若不可满足，则重复上述过程，直至可满足为止；若可满足，则直接得到一个随机生成的二进制数独终盘，然后通过两个步骤完成挖洞。

第一步，根据规则二（不能出现3个连续的0或1），若出现‘101’或‘010’序列，则挖掉中间的数字；若出现‘1001’或‘0110’则挖掉两边的数字，根据规则二能保证挖洞后的棋盘仍具有唯一解。

第二步，由从大到小的顺序生成一个1至 $4n^2$ 的序列，同时去掉第一步中已经挖掉的空的编号，然后按照这个序列的顺序用以下方法判断方法是否可挖洞，若可挖洞，则挖去。判断方法如下，对于棋盘上的一个已经已有数字的单元格，将其反转，若反转后棋盘仍然可满足，则这个地方不能挖，反之，若反转后不可满足，则这个地方可挖。按照序列顺序操作完成后就得到了一个具有唯一解的二进制数独棋盘。这个随机生成的过程的流程图见图 3-6。

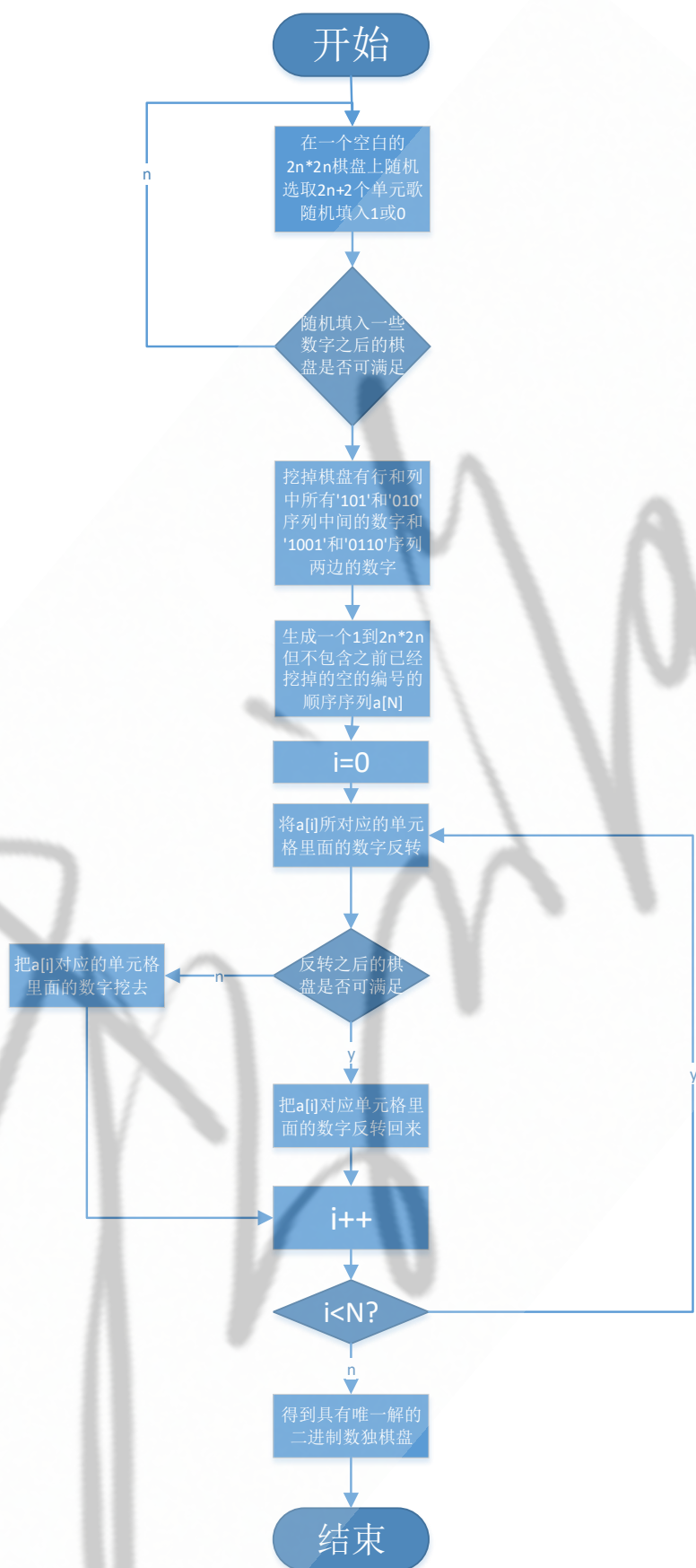


图 3-6 二进制数独棋盘随机生成过程流程图

说明：由于12阶二进制数独求解相比于10阶及以下耗时相对不可忽略，在随机生成12阶二进制数独时只进行第一步挖洞操作，不进行第二步操作。

Handwritten signature or mark in the background.

4 系统实现与测试

4.1 系统实现

系统实现的开发工具是 DEV C++ 5.11, 系统为 Windows 10, CPU 为 i7-8550U, 内存 8G, 其后的测试也均在此硬件环境下。

4.1.1 数据结构

4.1.1.1 优化前的数据结构

1. 存储一个文字的结点

```
struct node;
typedef struct node *Clause;
typedef struct node *Literal;
```

```
struct node{
    int ele;
    Literal next;
};
```

2. 存储链表头的结点

```
struct clause;
typedef struct clause *cnf;
typedef struct clause *conjunct;
```

```
struct clause{
    Clause C;
    conjunct next;
};
```

4.1.1.2 优化后的数据结构

1. 存储一个文字的结点

```
typedef struct Variable{
    int v;                //variable v
    int s;                //This variable is to signal wheather it is satisfied or not
```



```
}var;
```

```
typedef struct node_i{
    var L;                //Literal
    struct node_i *next;
}node_i;
```

```
typedef node_i *head_c;    //head pointer of a clause
```

```
typedef node_i *literal;
```

2.存储文字索引表的链表数组

```
typedef struct index{
    int k;                //pointer
    int pmc;              //plus or minus characteristic
    struct index *next;
}index;
```

```
typedef index *index_list;
```

```
typedef index *index_number;
```

```
typedef struct info_var{
    index_list in;
    int fp;                //the frequency of positive literals
    int fn;                //the frequency of negative literals
}info_var;
```

3.存储 CNF 公式的链表

```
typedef struct formula{
    head_c hc;
    int s;                //To signal whether this clause is removed or not
    int n;                //To denote how many variables are included in this clause
    struct formula *next;
}formula;
```

```
typedef formula *head_f;    //head pointer of a CNF formula
```

```
typedef formula *clause_i;
```

```
typedef struct CNF_I{
```

```
    int NOV;                //Number of Variables
    int NOC;                //Number of Conjuncts
    head_f orinal;          //original clauses
    clause_i *ad;           //address of each clause_i
    info_var *p;
```

```
}CNF_I;
```

```
typedef CNF_I *cnf_i;
```

4.1.2 函数声明

1. <Clause.h>中的函数声明（针对一个子句上的操作）

(1). Clause createClause(FILE *in);

从一个文件指针指向的位置开始读取整数，保存在一个链表中，遇 0 停止。

(2). void destroyClause(Clause C);

销毁以链表形式保存的一个子句。

(3). int ExistLiteral(Clause C,int k);

判断子句 C 中是否存在文字 k。

(4). void removeLiteral(Clause C,int k);

从子句 C 中去除文字 k。

(5). Clause copyClause(Clause C);

把一个以链表形式存储的子句 C 做一个备份。

(6). int isUnitClause(Clause C);

判断子句 C 是否为单子句。

(7). int checkClause(Clause C,int *assignment);

检查在真值指派 assignment 下子句 C 是否满足。

(8). void printClause(Clause C);

打印出子句 C 里面的内容。

2. <CNF.h>中的函数声明（针对一个 CNF 公式上的操作）

(1). cnf createCNF(FILE *in,int *_NOV);

从文件指针指向的文件中读取一个扩展名为.cnf 的文件，并把这个 CNF 文件的内容保存在一个以子句链表根节点为数据域的链表中，同时修改指针_NOV 所指向的内存，更改为这个 CNF 公式的变元数。

(2). cnf addUnitClause(cnf p,int k);

将指针 p 所指向的一个 CNF 公式做一个复制 p'，在 p'所指向的 CNF 公式中增加一个包含文字 k 的单子句，返回 p'。

(3). void removeClause(cnf p,conjunct c);

去除 c 后的下一个子句。

(4). int ExistNullClause(cnf p);

判断一个 CNF 公式中是否存在空子句。

(5). int ExistUnitClause(cnf p);

判断一个 CNF 公式中是否存在单子句。

(6). void clearLiteral(cnf p,int k);

从一个 CNF 公式中去除所有包含文字 k 的子句，再去掉所有 k 的否定变元。

(7). void printCNF(cnf p);

打印一个 CNF 公式。

(8). void destroyCNF(cnf p);

销毁一个 CNF 公式。

3. <DPLL.h>中的函数声明（针对 DPLL 算法和基本框架中的一些操作）

(1). int FindMax(int *a,int n);

找到数组 a 中的最大元素的下标。

(2). int DPLL(cnf p,int *assignment,int NOV,int STRATEGY,int *_times);

通过 STRATEGY 所决定的变元选取策略，判断 p 所指向的 CNF 公式是否可满足。若满足则通过数组 assignment 返回一组使其满足的真值指派，通过指针 _times 返回 DPLL 函数的调用次数。

(3). int checkAssignment(cnf p,int *assignment);

检查数组 assignment 所指向的一组真值指派是否能使 p 所指向的 CNF 公式满足。

(4). void output_file(char *filename,int r,int *assignment,int NOV,double w_time,int

times);

4. <MP.h>中包含的函数声明（针对最短正子句选取策略的一些操作）

(1). int allPositive(Clause C);

判断子句 C 中是否都是正文字。

(2). int MPstrategy(cnf p,int NOV);

从 p 所指向的 CNF 公式中只包含正文字的那些子句中选出最短的一个，选它的第一个文字。

5. <JW.h>中包含的函数声明（针对 MOM 策略/JW 算法的一些操作）

(1). float float_abs(float a);

返回一个浮点数的绝对值。

(2). int JW_length(Clause C,int k);

若子句 C 中不含变元 k 及其否定，返回 0；若子句 C 中包含变元 k，返回 C 的长度；若子句 C 中包含变元 k 的否定，返回 C 的长度的相反数。

(3). int FindJWMax(float *a,int n);

返回浮点数数组 a 中绝对值最大的那个元素的下标。

(4). float JWvalue(cnf p,int v);

返回变元 v 及其否定的 JW 函数值中较大的那一个，若是正文字，以正浮点数形式返回；若是负文字，以负浮点数返回。

(5). int JWstrategy(cnf p,int NOV);

计算一个 CNF 公式中所有变元的 JW 函数值，并返回 JW 函数值最大的那一个变元。

6. <CNF_ImprovedDataStructure.h>中的函数声明（针对优化后数据结构操作）

(1). cnf_i parseCNF(FILE *in);

在优化后的数据结构下读取 CNF 文件并进行逻辑结构解析。

(2). void outputCNF(cnf_i c);

在优化后的数据结构下输出一个 CNF 公式。

(3). void clearLiteral_2(cnf_i c,int layer,int k);

第 k 层决策在“懒散”数据结构下通过标记实现单子句传播。

(4). void retract(cnf_i c,int layer,int k);

撤销第 k 层决策并撤销由此产生的单子句传播。

(5). int FindUnitClause_2(cnf_i c);

判断 CNF 公式中是否有单子句，若有这个子句中的文字，否则返回 0。

(6). `int RandomStrategy(cnf_i c);`

找到第一个未被满足的子句的第一个文字作为这一次决策。

(7). `int checkCNF(cnf_i c);`

检查此时的 CNF 公式是否被满足，或者已经不可满足。

(8). `int checkAssignment_2(cnf_i c,int *assignment);`

在优化后的数据结构下检查这一组真值指派是否可使这一组 CNF 公式满足。

7. <DPLL_ImprovedDataStructure_nonRecursive.c>中的函数声明（非递归函数）

(1). `int DPLL_nonRecursive(cnf_i c,int *assignment);`

利用优化后的数据结构和非递归结构的 DPLL 实现。

8. <SudokuCNF.h>中的函数声明(将二进制数独游戏规则化成 CNF 公式的操作)

(1). `int order(int degree,int row,int column);`

将 degree 阶棋盘中的第 row 行第 column 列对应的单元格转化成变元序号。

(2). `void addrestriction(cnf p,int k);`

直接在 p 中增加一个单子句 k。

(3). `void addClause(cnf p,int *a,int n);`

在 p 中增加一个子句，增加的这个子句包含数组 a 中的所有元素。

(4). `void addClauses_rule1(cnf p,int degree);`

将规则 1 转化而来的子句(阶数为 degree)添加到 p 所指向的 CNF 公式中。

(5). `void addClauses_rule2(cnf p,int degree);`

将规则 2 转化而来的子句(阶数为 degree)添加到 p 所指向的 CNF 公式中。

(6). `void addClauses_rule3(cnf p,int degree);`

将规则 3 转化而来的子句(阶数为 degree)添加到 p 所指向的 CNF 公式中。

(7). `cnf createSudokuCNF(int degree);`

创建一个 CNF 公式，这个公式包含 degree 阶二进制数独游戏规则所等价的 CNF 公式中的所有子句。

9. <BinaryPuzzle.h>中的函数声明（针对随机生成具有唯一解的数独棋盘的操作）

(1). `int OnlySolution(cnf p,int NOV,int k);`

判断 p 所指向的 CNF 公式是否存在唯一解。

(2). `cnf RandomFilled(int degree,int *assignment);`

随机生成一个 degree 阶的二进制数独游戏终盘。

(3). void complement(int degree,cnf p,int *assignment);

生成一个从 1 到 degree*degree 的随机序列，将真值指派 assignment（终盘）对应的变元按照随机序列的顺序依次以单子句的形式添加到 p 所指向的 CNF 公式中。比如：一个 4 阶棋盘中，若第 2 排第 1 列填的是 0，就把单子句-5 添加到 CNF 公式中。

(4). void digHole(int degree,cnf p,int *puzzle);

puzzle 对应了一个完整的数独终盘，从这个终盘中挖去所有不增加可行性解个数的位置。

(5). cnf PuzzleWithOnlySolution(int degree,int *assignment,int *puzzle);

随机生成一个 degree 阶的具有唯一解的二进制数独棋盘。

10. <display.h>中的函数声明（针对交互性的操作）

(1). void displaySudoku(int *assignment,int degree);

打印 assignment 对应的 degree 阶的二进制数独棋盘。

(2). void SolveSudoku(void);

用户提供数独棋盘，系统求解。

(3). void PlayBinaryPuzzle(void);

系统随机生成一个二进制数独棋盘，用户求解，可进行简单交互，游戏可玩。

(4). void printMainMenu(void);

打印主菜单。

(5). void printBinaryPuzzleMenu(void);

打印数独系统菜单

(6). void printPuzzleOperation(void);

打印二进制数独游戏的可选操作菜单（填、删、清空）。

(7). void Puzzle(void);

进入数独系统。

(8). void printSATStrategyMenu(void);

打印 SAT 变元选择策略菜单。

(9). void SAT_Recursive(int k);

进入利用优化前数据结构和递归函数解决 SAT 问题的程序。

(10). void SAT_nonRecursive(void);

进入利用优化后数据结构和非递归函数解决 SAT 问题的程序。

(11). void SAT(void);

进入 SAT 问题求解系统。

4.1.3 函数调用关系

SAT 模块函数调用关系见图 4-1。

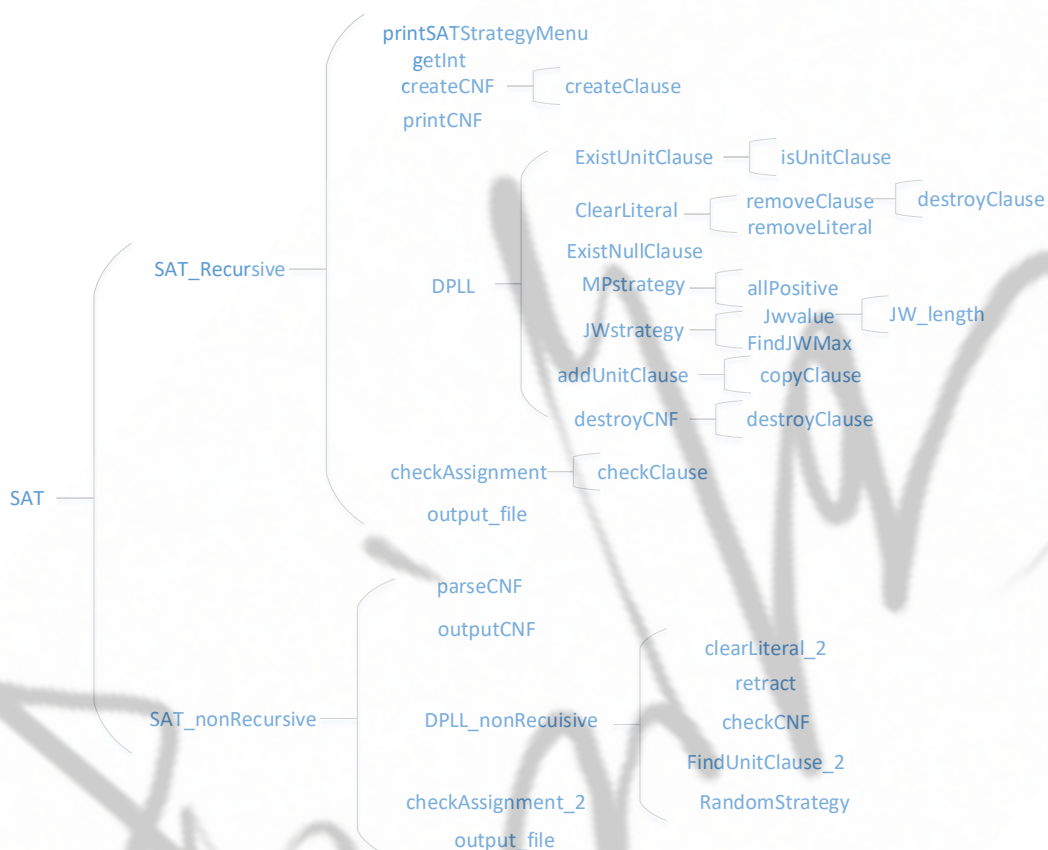


图 4-1 SAT 模块函数调用关系

SolveSudoku 模块函数调用关系见图 4-2。



图 4-2 SolveSudoku 模块函数调用关系

PlayBinaryPuzzle 模块函数调用关系见图 4-3。



图 4-3 PlayBinaryPuzzle 模块函数调用关系

4.2 系统测试

4.2.1 SAT 求解器模块

测试目标：这一模块应该能首先做到判断给定的CNF公式是否可满足，同时输出求解时间和递归算法下DPLL函数调用次数。对于可满足的CNF公式，应该能给出一组使其满足的真值指派。这一模块的测试目的主要在于检验SAT问题求

解器的正确性以及从求解所耗费的时间以及空间上分析数据结构和函数结构的优化对程序的改进。

测试大纲：测试数据的选取原则为尽量覆盖到范围比较广的CNF算例，这一点从变元数、子句数、变元与子句个数比等方面可以体现。基于以上原则，本次测试一共选取29个CNF算例。其中不可满足算例8个，这8个不可满足算例覆盖变元数从30至10410，覆盖子句数从100到43636，覆盖子句与变元个数比从2.000到43.636。可满足算例21个，其中小型和中型算例各8个，大型算例5个，这21个可满足算例覆盖变元数从100到1501，覆盖子句数从340到22060，覆盖子句与变元个数比从3.400到30.620。系统测试的主要对象为算例求解的正确性以及求解时间，通过测试结果从运行时间的维度计算时间优化率，并初步分析影响优化率的相关参数。除此以外，通过操作系统的任务管理器可以查看程序运行过程中所占用的内存，测试中也会基于此来变元选择策略的优化、测试数据结构的优化以及算法结构从递归到非递归的改变对算法的空间复杂度的优化性能，也会比较两种优化方法的性能。

4.2.1.1 SAT 求解器的正确性验证以及时间复杂度的优化

由于本次测试的数据较多，将用以下算例的运行结果截图作为本次测试的所有测试结果的参考截图。

```
#####
# Welcome to My SAT Solver #
#-----Choose an Improvement Aspect-----#
# 1.Variable Selection Strategy Improvement #
# 2.Data Structure && Algorithm Structure Improvement #
# 3.No Improvement #
# 0.Exit SAT Solver Module #
#####
Please Make Your choice:1

*****
*-Strategy for DPLL BCP-*
* 1.MP *
* 2.W *
*****

Please choose a strategy.
Your choice:1
Please input your filename:
sample\test\flat150-1.cnf

You want to see the Conjunctive Normal Formula?
0.no 1.yes
Your choice:0

This Conjunctive Normal Formula is SATISFIABLE!
The Truth Assignments are as followed
1 -2 -3 -4 5 -6 -7 8 -9 -10 11 -12 -13
14 15 -16 -17 18 -19 20 -21 -22 -23 24 -25 -26 27 -28 -29 30 31 -32 -33 -34 -35 36 37 -38 -39 -40 41 -42 43 -44 -45 -46 47
48 -49 50 -51 -52 -53 54 -55 -56 57 -58 59 -60 -61 -62 63 -64 -65 66 -67 -68 69 70 -71 -72 73 -74 -75 -76 -77 78 -79 -80 81 -82 83 -84 -85 86 -87 88 -89 -90 -91 9
2 -93 -94 -95 96 -97 -98 99 -100 101 -102 103 -104 -105 106 -107 -108 -109 -110 111 112 -113 -114 115 -116 -117 -118 -119 120 -121 -122 123 -124 125 -126 127 -128
-129 130 -131 -132 -133 -134 135 -136 137 -138 -139 140 -141 -142 -143 144 145 -146 -147 -148 -149 150 -151 152 -153 -154 155 -156 -157 158 -159 160 -161 -162 -163
164 -165 166 -167 -168 -169 170 -171 -172 173 -174 175 -176 -177 178 -179 -180 181 -182 -183 -184 185 -186 187 -188 -189 -190 191 -192 -193 194 -195 196 -197 -198
-199 200 -201 202 -203 -204 205 -206 -207 -208 -209 210 -211 -212 213 -214 215 -216 217 -218 -219 -220 -221 222 -223 224 -225 226 -227 -228 -229 -230 231 -232 233
-234 -235 236 -237 238 -239 -240 241 -242 -243 -244 245 -246 247 -248 -249 250 -251 -252 -253 254 -255 -256 -257 258 -259 -260 261 -262 263 -264 -265 -266 267 -26
8 269 -270 271 -272 -273 -274 -275 276 -277 278 -279 -280 281 -282 283 -284 -285 -286 -287 288 -289 290 -291 292 -293 -294 295 -296 -297 -298 299 -300 -301 302 -30
3 304 -305 -306 -307 -308 309 310 -311 -312 -313 -314 315 316 -317 -318 319 -320 -321 -322 -323 324 -325 -326 327 328 -329 -330 -331 -332 333 -334 335 -336 -337 -3
38 339 -340 341 -342 -343 344 -345 -346 347 -348 349 -350 -351 -352 353 -354 355 -356 -357 -358 -359 360 -361 362 -363 -364 365 -366 367 -368 -369 370 -371 -372 -3
73 -374 375 376 -377 -378 -379 380 -381 -382 -383 384 -385 -386 387 -388 389 -390 -391 392 -393 -394 395 -396 397 -398 -399 400 -401 -402 403 -404 -405 406 -407 -4
08 -409 410 -411 412 -413 -414 -415 416 -417 418 -419 -420 421 -422 -423 -424 -425 426 427 -428 -429 430 -431 -432 -433 -434 435 436 -437 -438 -439 440 -441 -442 4
43 -444 445 -446 -447 448 -449 -450
It has been checked that the truth assignments above CAN satisfy the CNF!

DPLL Core Module Running Time: 3742.000000ms
DPLL was called for 5031 times!
```

图4-4 以flat150-1.cnf为代表的可满足算例变元选择策略优化后运行结果截图

```
#####
#           Welcome to My SAT Solver           #
#-----Choose an Improvement Aspect-----#
#           1.Variable Selection Strategy Improvement           #
#           2.Data Structure && Algorithm Structure Improvement #
#           3.No Improvement                               #
#           0.Exit SAT Solver Module                     #
#####

Please Make Your choice:2

Please input your filename:
sample\test\S\CBS_k3_n100_m403_b10_0.cnf

You want to see the Conjunctive Normal Formula?
0.no 1.yes
Your choice:0

This Conjunctive Normal Formula is SATISFIABLE!
The Truth Assignments are as followed
-1 -2 -3 -4 5 6 -7 8 -9 10 11 12 -13 14 -15 16 17 -18 19 -20 21 -22 23 24 25 26
-27 28 -29 30 31 32 -33 -34 -35 -36 37 -38 39 -40 41 42 43 -44 -45 -46 -47 -48 4
9 50 51 0 53 54 55 -56 -57 -58 0 60 0 62 63 -64 0 -66 -67 68 69 0 -71 72 73 74 7
5 0 0 78 -79 -80 -81 -82 -83 -84 0 -86 -87 88 89 -90 0 -92 -93 94 -95 96 97 -98
-99 -100
It has been checked that the truth assignments above CAN satisfy the CNF!

DPLL Core Module Running Time: 734.000000ms
```

图4-5 以CBS_k3_n100_m403_b10_0.cnf为代表的可满足算例数据结构优化后程序运行结果截图

```
#####
#           Welcome to My SAT Solver           #
#-----Choose an Improvement Aspect-----#
#           1.Variable Selection Strategy Improvement           #
#           2.Data Structure && Algorithm Structure Improvement #
#           3.No Improvement                               #
#           0.Exit SAT Solver Module                     #
#####

Please Make Your choice:3

Please input your filename:
sample\S\S\problem3-100.cnf

You want to see the Conjunctive Normal Formula?
0.no 1.yes
Your choice:0

This Conjunctive Normal Formula is SATISFIABLE!
The Truth Assignments are as followed
-1 2 -3 -4 5 -6 -7 -8 9 10 -11 12 -13 14 -15 16 17 -18 -19 -20 -21 -22 23 24 -25
-26 -27 -28 29 30 -31 32 -33 34 35 -36 -37 -38 39 40 41 42 -43 44 -45 46 -47 48
49 50 51 52 -53 54 -55 56 57 -58 59 60 61 -62 -63 64 65 66 -67 68 -69 70 71 72
-73 74 75 76 77 -78 79 -80 -81 82 83 -84 -85 -86 87 -88 89 90 91 -92 -93 94 -95
-96 -97 98 99 100
It has been checked that the truth assignments above CAN satisfy the CNF!

DPLL Core Module Running Time: 31.000000ms
DPLL was called for 240 times!
```

图 4-6 以 problem3-100.cnf 为代表的可满足算例优化前程序运行结果截图

```
#####
#           Welcome to My SAT Solver           #
#-----Choose an Improvement Aspect-----#
#       1.Variable Selection Strategy Improvement       #
#       2.Data Structure && Algorithm Structure Improvement #
#               3.No Improvement               #
#               0.Exit SAT Solver Module               #
#####

Please Make Your choice:1

*****
*-Strategy for DPLL BCP--*
*       1.MP       *
*       2.JW       *
*****

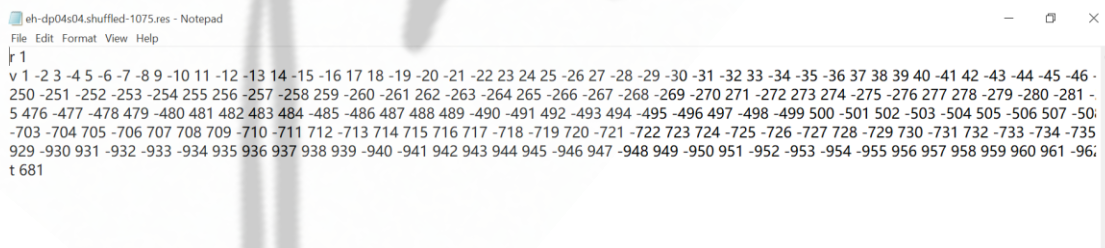
Please choose a strategy.
Your choice:1
Please input your filename:
sample\My\U\qg5-10.cnf

You want to see the Conjunctive Normal Formula?
0.no  1.yes
Your choice:0

This Conjunctive Normal Formula is UNSATISFIABLE

DPLL Core Module Running Time: 1744.000000ms
DPLL was called for 73 times!
```

图 4-7 以 qg5-10 为代表的不可满足算例变元选择策略优化后程序运行结果截图



```
eh-dp04s04.shuffled-1075.res - Notepad
File Edit Format View Help
1
v 1 -2 3 -4 5 -6 -7 -8 9 -10 11 -12 -13 14 -15 -16 17 18 -19 -20 -21 -22 23 24 25 -26 27 -28 -29 -30 -31 -32 33 -34 -35 -36 37 38 39 40 -41 42 -43 -44 -45 -46 -
250 -251 -252 -253 -254 255 256 -257 -258 259 -260 -261 262 -263 -264 265 -266 -267 -268 -269 -270 271 -272 273 274 -275 -276 277 278 -279 -280 -281 -
5 476 -477 -478 479 -480 481 482 483 484 -485 -486 487 488 489 -490 -491 492 -493 494 -495 -496 497 -498 -499 500 -501 502 -503 -504 505 -506 507 -508 -
703 -704 705 -706 707 708 709 -710 -711 712 -713 714 715 716 717 -718 -719 720 -721 -722 723 724 -725 -726 -727 728 -729 730 -731 732 -733 -734 -735
929 -930 931 -932 -933 -934 935 936 937 938 939 -940 -941 942 943 944 945 -946 947 -948 949 -950 951 -952 -953 -954 -955 956 957 958 959 960 961 -962
t 681
```

图4-8 以eh-dp04s04.shuffled-1075.cnf为例的程序运行结果产生的输出文件

eh-dp04s04.shuffled-1075.res的输出规范

具体的测试结果如下。

以下四个表格中优化方法一栏填写1表示变元选择策略优化（最短正子句策略，即程序中显示的MP策略），2表示数据结构优化。

1.小型可满足算例的测试结果（见表4-1）

表4-1 小型可满足算例的测试结果

算例名	变元数	子句数	子句与变元个数比	可满足性测试结果 Y/N	优化前运行时间/ms	优化后运行时间/ms	优化率	优化方法
problem3-100.cnf	100	340	3.400	Y	31	0	100%	2
problem11-100.cnf	100	600	6.000	Y	19	5	73.7%	2
CBS_k3_n100_m403_b10_0.cnf	100	403	4.030	Y	2291	734	68.0%	2
CBS_k3_n100_m403_b10_999.cnf	100	403	4.030	Y	4637	1937	58.2%	2
tst_v100_c425.cnf	100	425	4.250	Y	16	5	68.8%	2
RTI_k3_n100_m429_0.cnf	100	429	4.290	Y	3204	1373	57.1%	2
RTI_k3_n100_m429_111.cnf	100	429	4.29	Y	347	161	53.6%	2
flat50-1.cnf	150	545	3.633	Y	8	2	75.0%	2

2.中型可满足算例测试结果（见表4-2）

表4-2 中型可满足算例的测试结果

算例名	变元数	子句数	子句与变元个数比	可满足性测试	优化前运行时间/ms	优化后运行时间/ms	优化率	优化方法

				结 果 Y/N				
ais10.cnf (基准算例)	100	3151	17.409	Y	1806	45	97.5%	1
problem5-200.cnf	200	2851	14.255	Y	46	15	67.4%	2
bart17.shuffled-231.cnf	231	1166	5.408	Y	15	0	100%	2
sud00009.cnf (基准算例)	303	2851	9.409	Y	175	140	20%	1
flat125-1.cnf	375	1403	3.741	Y	107	96	10.3%	2
flat150-1.cnf	450	1680	3.733	Y	内存 溢出	3742		1
bw_large.a.cnf	459	4675	10.185	Y	1384	421	69.6%	1
sw100-70.cnf	500	3100	6.200	Y	62090	94	99.8%	1

3.大型可满足算例测试结果（见表4-3）

表4-3 大型可满足算例的测试结果

算例名	变 元 数	子 句 数	子 句 与 变 元 个 数 比	可 满 足 性 测 试 结 果 Y/ N	优 化 前 运 行 时 间/ms	优 化 后 运 行 时 间 /ms	优 化 率	优 化 方 法
qg6-09.cnf	729	1558 0	30.26 1	Y	347	23 6	32%	1
qg7-09.cnf	729	2206 0	30.62 0	Y	532	23 5	55.8 %	1
qg4-09.cnf	729	1558	21.37	Y	1872	47	97.5	1

		0	2		8	5	%	
eh-dp04s04.shuffled-1075.cnf	1075	3152	2.932	Y		474		1
ssa7552-038.cnf	1501	3575	2.382	Y	93	78	16.1%	2

4.不可满足算例测试结果（见表4-4）

表4-4 不可满足算例的测试结果

算例名	变元 数	子句 数	子句 与变元 个数比	可 满足 性 测 试 结 果 Y/ N	优 化 前 运 行 时 间/ms	优 化 后 运 行 时 间 /ms	优 化 率	优 化 方 法
unsat-5cnf-30.cnf	30	420	14.000	N	250	89	64.4%	1
u-problem7-50.cnf	50	100	2.000	N	23	1	95.7%	2
u-problem7-100.cnf	100	200	2.000	N	118	8	93.2%	2
uuf150-01.cnf	150	645	4.300	N	内存 溢出	28449		1
qg5-09.cnf	729	28540	39.150	N	3714	514	86.2%	1
u-dp04u03.shuffled-825.cnf	825	2411	2.922	N	81402	44795	45.0%	1
qg5-10.cnf	1000	4363	43.63	N	11149	1744	84.8	1

		6	6		3		%	
ssa6288-047.cnf	1041	3423	3.289	N	62	31	50.0	2
	0	8					%	

SAT模块的求解正确性以及优化效果分析如下。

(1) SAT模块能完全准确的完成SAT问题的求解，即一个CNF公式的可满足性的判断，对于可满足的CNF算例，能输出一组使其满足的真值指派，并且这组真值指派经过了程序的验证（见图4-1和图4-2），能够确保输出的真值指派确实能够满足这个CNF公式。

(2) 对于基准算例（sud00009.cnf和ais10.cnf），经过测试，使用最短正子句策略优化的效果更好，经过计算，ais10.cnf的优化率高达97.5%，sud00009.cnf的优化率为20%。

(3) 对于子句与变元个数比较小的算例，数据结构上的优化有很好的优化效果。经过计算，子句与变元个数比在6.000及以下的所有算例的平均优化率（平均优化率=（优化前运行总时间-优化后运行总时间）/优化前运行总时间）高达57%，对于部分算例优化率可以达到70%~80%的水平，对于个别算例,优化率甚至能达到100%。

(4) 而对于子句与变元个数比较大的算例以及部分规模较大的算例，最短正子句策略（程序中显示的MP策略）有很好的优化效果，极大地减少了函数递归次数，明显地缩短了求解时间，验证了该变元选择策略对于部分算例的优化效果。

4.2.1.2 空间复杂度的优化

对于优化前的程序，每递归调用一次DPLL函数就需要复制一次当前的CNF公式，会占用大量内存空间，随着问题规模的增加和程序的运行，占用的内存可能会越来越大，甚至系统内存的边界。算例uuf150-01.cnf在运行过程中就几乎达到了可调用内存的边界，见图4-9。

Name	Status	13% CPU	93% Memory	4% Disk	0% Network	3% GPU
Apps (4)						
> Task Manager		0.3%	18.1 MB	0 MB/s	0 Mbps	0%
> Windows Command Processor (...)		10.8%	5,393.1 MB	0.1 MB/s	0 Mbps	0%
> Windows Explorer		0.3%	29.5 MB	0.1 MB/s	0 Mbps	0%

图4-9 算例uuf150-01.cnf运行过程中任务管理器所显示的资源调用情况

而优化后的数据结构吸收了“懒散”数据结构的思想，通过变量来标记变元的状态，不需要通过复制CNF公式进行递归调用，使得整个程序的空间复杂度和这个CNF公式的总大小（子句数和子句中包含的变元数）相等，且不随程序的运行增加，从根本上避免了占用内存过多和内存溢出的问题，优化前程序和优化后程序的资源调用情况对比见图4-10。

Name	Status	21% CPU	93% Memory	1% Disk	0% Network	0% GPU	GPU engine	Power usage	Power usage t...
Apps (8)									
> Google Chrome (4)		0%	50.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Microsoft Excel		0%	31.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Snipping Tool		0%	3.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Task Manager		0.2%	24.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Windows Command Processor [...]		9.9%	4,628.2 MB	0 MB/s	0 Mbps	0%		High	High
> Windows Command Processor [...]		10.0%	17.8 MB	0 MB/s	0 Mbps	0%		High	High
> Windows Explorer		0.1%	70.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low

图4-10 优化前程序和优化程序同时运行过程中任务管理器显示的资源调用情况

第一个命令提示符代表优化前程序，第二个命令提示符代表数据结构优化后程序

4.2.2 二进制数独模块

4.2.2.1 二进制数独棋盘的求解

通过布尔代数的变换，在不添加附加变元的前提下将二进制数独规则转化为一个 CNF 公式（详细变换方法见 3.2.2 二进制数独游戏规则转化（不添加附加变元）），有效降低了常规转化方法中规模巨大的附加变元对求解时间和求解空间造成的巨大压力。

测试目标：本模块应该首先能正确判断一个数独棋盘是否有解，若有解，给出这个棋盘的满足二进制数独游戏规则的正确解，若无解，则提示用户输入的棋盘无解。本模块最优目标是能够较快地（在可等待时间内）求解任何一个有解的二进制数独棋盘。

测试大纲：将本课程设计所提供的所有二进制数独测试样例（6 阶至 14 阶）以特定的格式保存在以.txt 为后缀的文档中，测试时以文件读取方式逐一对每一算例进行测试，输出求解结果和求解时间，比对求解结果和样例所附的正确解是否完全相同，判断求解耗时是否在可等待时间范围内。

测试结果：本次测试中，二进制数独棋盘测试样例中的最高阶数为 14，对于这些样例，求解器全部能在较短时间内正确求解，在测试范围内，本模块实现了其功能——即能够较快地（在可等待时间内）求解 14 阶及以下的任何一个二进制数独棋盘（14 阶以上未考虑也未测试）。对于 8 阶及以下的二进制数独，求解时间基本可忽略不计；对于 10 阶的二进制数独，求解时间通常在 $10^{-1}s$ 量级；对于 12 阶的二进制数独，求解时间通常在 10^0s 量级；对于 14 阶的二进制数独，

求解时间通常在几十秒到两分钟之间，一般不超过两分钟。

测试样例截图：一个 10 阶数独棋盘见图 4-11，其求解结果见图 4-12，原棋盘的正确解见图 4-13；一个 12 阶数独棋盘见图 4-14，其求解结果见图 4-15；一个 14 阶数独棋盘见图 4-16，其求解结果见图 4-17，原棋盘的正确解见图 4-18。

	0								
1						1			
				0					
							1		1
0	0			1		1			
		1							
	0					0			
					1				
1	1				1				
									1

图 4-11 一个待求解的 10 阶二进制数独棋盘

Congratulations!
Your puzzle is solved!
The answer is as follows!

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	1	0	0	1	1	0	0
2	1	1	0	0	1	0	1	0	0	1
3	0	1	1	0	0	1	0	1	1	0
4	1	0	0	1	0	1	0	1	0	1
5	0	0	1	0	1	0	1	0	1	1
6	0	1	1	0	0	1	1	0	1	0
7	1	0	0	1	1	0	0	1	0	1
8	0	0	1	1	0	1	1	0	0	1
9	1	1	0	0	1	1	0	0	1	0
10	0	1	0	1	1	0	0	1	1	0

Solving time: 567.000000ms

图 4-12 图 4.7 中的棋盘的求解结果

1	0	1	1	0	0	1	1	0	0
1	1	0	0	1	0	1	0	0	1
0	1	1	0	0	1	0	1	1	0
1	0	0	1	0	1	0	1	0	1
0	0	1	0	1	0	1	0	1	1
0	1	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	0	1
0	0	1	1	0	1	1	0	0	1
1	1	0	0	1	1	0	0	1	0
0	1	0	1	1	0	0	1	1	0

图 4-13 图 4.7 中棋盘的正确解

						1		1			0
				0		1				1	
		1		0						0	0
0	0							1			
				0	0				0		
		0		0				1			
						0	0		0		0
	0	0			1						
		0					1				
0	0									0	
							0	0			
0	0				0						

图 4-14 一个待求解的 12 阶二进制数独棋盘

```

It may take a little while(a few seconds) to solve a Sudoku with a degree of 12.
Thank you for your patience...

Congratulations!
Your puzzle is solved!
The answer is as follows!

```

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	0	1	0	1	0	1	1	0	0
2	0	1	0	1	0	0	1	1	0	0	1	1
3	1	0	1	1	0	1	0	1	0	0	1	0
4	0	0	1	0	1	1	0	0	1	1	0	1
5	1	1	0	1	0	0	1	1	0	0	1	0
6	1	0	0	1	0	0	1	0	1	1	0	1
7	0	1	1	0	1	1	0	0	1	0	1	0
8	1	0	0	1	0	1	0	1	0	1	0	1
9	1	1	0	0	1	0	1	1	0	1	0	0
10	0	0	1	0	1	1	0	0	1	0	1	1
11	1	1	0	1	0	1	1	0	0	1	0	0
12	0	0	1	0	1	0	0	1	1	0	1	1

Solving time: 4949.000000ms

图 4-15 图 4-10 中棋盘的求解结果

						0						1	
		1	1						0			1	
	1										0		
									0	0		0	0
		0					1						0
							1			1	0		
		1		1									
					1								
				1					1		1		1
0									0		0		0
		0											
	1							1	1				1
			0	0		1		1					
											1		0

图 4-16 一个待求解的 14 阶二进制数独棋盘

```

It may take a little while(a minute or so, sometimes 2 minutes) to solve a Sudoku with a degree of 14.
Thank you for your patience...

Congratulations!
Your puzzle is solved!
The answer is as follows!

```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	1	0	0	1	1	0	0	1	1	0	1	1	0
2	0	0	1	1	0	0	1	0	1	0	1	0	1	1
3	1	1	0	0	1	0	0	1	0	1	1	0	0	1
4	1	0	1	1	0	1	1	0	1	0	0	1	0	0
5	0	1	0	1	1	0	1	1	0	1	0	0	1	0
6	1	0	1	0	0	1	0	1	0	0	1	0	1	1
7	0	0	1	0	1	0	1	0	1	1	0	1	0	1
8	1	1	0	1	0	1	0	1	0	0	1	0	1	0
9	1	0	0	1	0	1	0	0	1	1	0	1	0	1
10	0	1	1	0	1	0	1	0	0	1	0	1	0	1
11	1	0	0	1	1	0	1	1	0	0	1	0	1	0
12	0	1	1	0	0	1	0	1	1	0	1	0	1	0
13	0	0	1	0	0	1	1	0	1	1	0	1	0	1
14	1	1	0	1	1	0	0	1	0	0	1	1	0	0

Solving time: 77842.000000ms

图 4-17 图 4-12 中棋盘的求解结果

0	1	0	0	1	1	0	0	1	1	0	1	1	0
0	0	1	1	0	0	1	0	1	0	1	0	1	1
1	1	0	0	1	0	0	1	0	1	1	0	0	1
1	0	1	1	0	1	1	0	1	0	0	1	0	0
0	1	0	1	1	0	1	1	0	1	0	0	1	0
1	0	1	0	0	1	0	1	0	0	1	0	1	1
0	0	1	0	1	0	1	0	1	1	0	1	0	1
1	1	0	1	0	1	0	1	0	0	1	0	1	0
1	0	0	1	0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	0	1	0	0	1	0	1	0	1
1	0	0	1	1	0	1	1	0	0	1	0	1	0
0	1	1	0	0	1	0	1	1	0	1	0	1	0
0	0	1	0	0	1	1	0	1	1	0	1	0	1
1	1	0	1	1	0	0	1	0	0	1	1	0	0

图 4-18 图 4-12 中棋盘的正确解

测试结果分析：由于 12 阶算例未附有正确解，经过人工代入二进制数独游戏规则判定其求解结果正确，且经过比对求解结果和正确解可知 10 阶算例和 14 阶算例的求解结果也完全正确，验证了求解器的正确性，同时 10 阶算例求解时

间为 567ms，12 阶算例求解时间约为 5s，14 阶算例求解时间约为 1min15s，三个不同阶算例的求解时间均在可等待时间范围内，验证了该二进制数独求解器确实能较快地求解出一个 14 阶及以下的二进制数独棋盘，很好地完成了测试目标。

4.2.2.1 二进制数独棋盘的随机生成

测试目标：本模块应该能在较短时间内随机生成一个具有唯一解的二进制数独游戏棋盘。

测试大纲：分别测试 4 阶、6 阶、8 阶、10 阶、12 阶、14 阶具有唯一解的数独棋盘随机生成功能的正确性以及耗时是否在可等待时间范围内，并人工求解以判定是否具有唯一解。

测试结果：经过测试，除 14 阶的二进制数独棋盘随机生成过程不能在可等待时间范围内完成以外，其余阶数的二进制数独棋盘的随机生成都能在较短时间内完成。其中 4 阶、6 阶、8 阶二进制数独棋盘的随机生成过程耗时可忽略不计；10 阶二进制数独棋盘的随机生成需要耗费大约几秒钟的时间；12 阶二进制数独棋盘的随机生成需要耗费大约 1-2 分钟时间。经人工求解判定所有生成的随机棋盘均具有唯一解。

测试样例截图：图 4-19 是随机生成的一个 8 阶二进制数独棋盘(耗时 0.887s)，经过人工求解，它具有唯一解；图 4-20 是随机生成的一个 10 阶二进制数独棋盘（耗时 3.77s），经过人工求解，它具有唯一解；图 4-21 是随机生成的一个 12 阶二进制数独棋盘（耗时约 1 分 16 秒），经过人工求解，它具有唯一解。

```
Please Make Your Choice:1
Please input the order of the puzzle you wanna try(4 or 6 or 8 or 10 or 12):8

Generating time: 340.000000ms
```

	1	2	3	4	5	6	7	8
1	0	1		1	1		0	1
2						1		1
3		1	1		1	0		
4	1		1	1				0
5	0			0			1	1
6		0				1	1	
7			1		0	1		1
8	1		1		1		1	

HINT: ONLY SOLUTION GUARANTEED

图 4-19 随机生成的具有唯一解的 8 阶二进制数独棋盘

```

*****
*   Welcome to My Binary Puzzle Game   *
*--- ----FUNCTIONS-----*
*1.Play Binary Puzzle(with ONLY 1 solution)*
*   2.Solve the Puzzle You Provide     *
*   0.EXIT BINARY PUZZLE GAME          *
*****

Please Make Your Choice:1
Please input the order of the puzzle you wanna try(4 or 6 or 8 or 10 or 12):10

Generating a puzzle...
It may take a few seconds.
Thank you for your patience!

Generating time: 3770.000000ms

  | 1  2  3  4  5  6  7  8  9 10
--|-----
1 | 0  0      1      1  1      1  0
2 |      0  0      1  1      0
3 |      1      0  1      1  0
4 | 1  1      1  0      0      0
5 |      0      0      0  0
6 |      1  1      0
7 | 1  1      0      0      0  0
8 |      0      0  0      1
9 |      1  1      0      0  0      1
10| 0  0      0  1      1

HINT: ONLY SOLUTION GUARANTEED

```

图 4-20 随机生成的具有唯一解的 10 阶二进制数独棋盘

```

*-----FUNCTIONS-----*
*1.Play Binary Puzzle(with ONLY 1 solution)*
*   2.Solve the Puzzle You Provide   *
*   0.EXIT BINARY PUZZLE GAME       *
******
Please Make Your Choice:1
Please input the order of the puzzle you wanna try(4 or 6 or 8 or 10 or 12):12

Generating a puzzle...
It may take 1-2 minutes.
Thank you for your patience!

Generating time: 75943.000000ms

  | 1  2  3  4  5  6  7  8  9 10 11 12
  |-----
1 | 0  0      1  1      1  1      1  0
2 |      0  0      1      1  1      0
3 |      1      0      1  0
4 | 1  1      1      1  0      0
5 | 1      0      1  0
6 |      1      1      0  0
7 |      0  0      1  1
8 |      1  1      1      0  0
9 |      1  1      0      0  0      1
10| 0      0      0      1  1
11| 0  0      1  0      1  1
12|      1      0      0  0      1

HINT: ONLY SOLUTION GUARANTEED

```

图 4-21 随机生成的具有唯一解的 12 阶二进制数独棋盘

测试结果分析：本模块能在较短时间内（可等待范围内）随机生成一个 12 阶及以下的二进制数独棋盘，达到了本模块的预期效果，完成了测试目标。

4.2.3 用户交互模块

测试目标：本模块应该能够识别用户不按照提示或者规定而进行的错误的、不合理的输入，对于这种输入不进行下一步操作，友善提示用户重新输入，从而实现程序在任何输入下都不会出现异常终止的目标；除此以外，本模块应该能按照用户预期来处理正确的、合理的输入，尤其是在数独模块，应该能正确完成填

空、删除空、清空所有已填空等游戏交互功能。

测试大纲：

1. 对于程序中所有提示用户输入的地方进行测试，故意提供各种不合理的输入，测试程序是否会异常终止。
2. 对与程序中所有提示用户输入的地方进行测试，提供正确的、合理的输入，测试程序能否按照预期正确运行。
3. 在二进制数独游戏模块，除了尝试填空、删除空、清空所有已填空等常规操作外，还应该尝试删除、修改棋盘中给出的空，测试程序能否正确处理这种不合理的输入请求。

测试结果：测试过程中，对于各种不合理输入的测试（如输入提示以外的数字、输入字符串、输入不存在的文件名、输入数独棋盘的阶数为奇数或者不在指定范围中等），该模块能正确识别并提示用户重新输入；对于正确的、合理的输入，该模块均能正确处理。在二进制数独模块中，所有基本操作（填空、删除空、清空所有已填空）都能正确完成，对于各种非法操作（更改、删除棋盘提供的空，操作时输入超过棋盘阶数的行数或者列数等）均不进行并提示用户错误原因。可以看出，该模块很好地完成了用户交互功能，并且提高了系统的稳定性，避免系统因为错误的、不合理的输入而异常终止。

测试样例截图：图 4-22 是以主菜单为例展示的在功能选择时该模块对于错误输入做出的响应；图 4-23 是输入文件名不存在时该模块做出的响应；图 4-24 是删除棋盘给出的空时该模块做出的响应；图 4-25 是数独游戏操作过程中输入的行数超出阶数时该模块做出的响应。

```

-----|
Welcome to My SAT/Sudoku System
*****Menu For this system*****|
1.SAT      2.Sudoku      0.EXIT SYSTEM
-----|

Please choose the function you wanna try:a
INVALID INPUT!
Please input your choice once again:3
INVALID INPUT!
Please input your choice once again:-1
INVALID INPUT!
Please input your choice once again:]
INVALID INPUT!
Please input your choice once again:skdfjl32984
INVALID INPUT!
Please input your choice once again:1

#####
#           Welcome to My SAT Solver           #
#-----Choose an Improvement Aspect-----#
#           1.Variable Selection Strategy Improvement           #
#           2.Data Structure && Algorithm Structure Improvement  #
#           3.No Improvement                                     #
#           0.Exit SAT Solver Module                         #
#####
Please Make Your choice:
    
```

图 4-22 功能选择时应对错误、不合理输入


```

Please input your filename:
sample\S\S\problm1-20.cnf

Sorry, sample\S\S\problm1-20.cnf does NOT exist!
Please input your filename once again:
sample\S\S\problem1-20.cnf

You want to see the Conjunctive Normal Formula?
0.no 1.yes
Your choice:0

This Conjunctive Normal Formula is SATISFIABLE
The Truth Assignments are as followed
-1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20
It has been checked that the truth assignments above CAN satisfy the CNF!

DPLL Core Module Running Time: 0.000000ms
    
```

图 4-23 输入的文件名不存在

```

  | 1 2 3 4
--|-----
1 | 1 1
2 | 0 1
3 | 1 1 0
4 | 0

HINT: ONLY SOLUTION GUARANTEED

-----
| Available Operation
| 1.change/fill in a blank
| 2.clear a blank
| 3.quit & display the answer
| 4.clear all blanks
|-----
(Any other number would be considered to be 4)

choose a operation:1

Please fill in the blank as the following instructions says.
Input the row number :1

Input the column number :2

Input the value :0

Sorry, the number you chose is given by the puzzle.
You CANNOT change it!
    
```

图 4-24 删除棋盘给出的空

```

| 1 2 3 4 5 6
-----
1 | 1 1      0 0
2 | 1      0 0
3 |      1    1
4 |      1 1    1
5 |      0      1
6 | 0 0      1

HINT: ONLY SOLUTION GUARANTEED

-----
| Available Operation
| 1.change/fill in a blank
| 2.clear a blank
| 3.quit & display the answer
| 4.clear all blanks
|-----
(Any other number would be considered to be 4)

choose a operation:1

Please fill in the blank as the following instructions says.
Input the row number :7
INVALID INPUT!
Please try again:6

Input the column number :3

Input the value :1
    
```

图 4-25 输入的行数超过阶数

测试结果分析：通过各种错误、不合理的输入，以及数独游戏交互过程中的各种非法请求的测试，验证了该模块都能正确识别并提示用户重新输入，保障了系统的稳定性。说明该模块达到了预期效果，完成了测试目标。

5 总结与展望

5.1 全文总结

本次课程设计基于 SAT 问题的求解，完成了包括 SAT 问题国内外研究现状的总结和 SAT 求解器实现和二进制数独游戏的规约与实现等多个方面的研究工作。主要有

(1) 相关资料的查阅。通过查阅相关资料，了解和学习当前 SAT 问题的研究现状，掌握 SAT 求解器的完备算法 DPLL 算法的内核以及具体实现，了解了一些常见的变元选择策略及其实现，同时领悟了数独问题和 SAT 问题之间的规约，为课程设计的具体展开奠定了坚实的基础。

(2) 输入输出功能的实现。CNF 文件的读取与 CNF 公式的解析，算例求解结果的保存。

(3) 数据结构的设计以及数据结构的优化。基于 SAT 求解问题的特点，本课程设计完成了 CNF 公式的朴素链表实现，并在此基础上进行优化，即在朴素链表的基础上将其改为一种“懒散”的数据结构（通过标记实现删除文字，删除子句操作），同时添加了文字的索引邻接表。

(4) DPLL 算法基本框架研究、递归程序实现以及转化非递归结构的优化。

(5) 变元选择策略的实现与优化。了解并实现了最短正子句策略和 MOM 策略。

(6) 二进制数独游戏规则转化。根据布尔代数知识推导出在不添加附加变元的前提下将二进制数独游戏规则转化为 CNF 公式的方法。

(7) 二进制数独棋盘的求解以及随机生成。

(8) 输入模块稳定性的提升。通过输入有效性的检验实现了错误、不合理输入的判断，保证在系统在各种输入下均不会异常终止。

(9) 二进制数独游戏交互的实现。实现了二进制数独游戏的基本操作，并隔离了各种非法操作请求，保证了游戏的交互性和稳定性。

5.2 工作展望

本课程设计中所编写的 SAT 求解器已经可以求解小规模以及部分中大规模的 SAT 问题，并且可以将二进制数独问题转化为 SAT 问题，并通过 SAT 问题求解器进行求解和随机生成。但是，这次实现的 SAT 求解器仍有很多方面的不足，其中最明显的问题就是对于很多中大规模的 SAT 问题还不能在可等待时间

内完成，因此还需要进行更深一步的学习和研究。在今后的工作和研究中可以着重在以下几个方面进行突破。

(1) 对 CNF 公式进行预处理。目前关于 SAT 问题的预处理手段有很多，还需要查阅更多的文献来掌握更多的预处理方法，并通过实验来寻求各种情况下最适合的预处理手段。

(2) 增加冲突子句学习机制。目前所实现 SAT 求解器实现的 DPLL 算法遇到冲突时进行的是时序回溯，但有可能冲突的发生并不在当前的决策层，从而进行不必要的搜索，因而需要引入冲突子句学习机制，通过分析冲突产生原因进而实现非时序回溯，避免不必要的搜索，以减少求解时间。

(3) 继续进行更高效的数据结构优化。通过对 SAT 问题的深入了解，可以体会到 SAT 问题中存在大量的遍历、删除、增加操作，通过优化 CNF 公式的存储结构对于提升 SAT 求解器的性能至关重要。

(4) 寻求更优的变元选择策略。应该指出，变元选择策略对于 DPLL 算法的性能有极大影响，在今后的工作中可以查阅更多的文献，寻求更优的变元选择策略，以提高 SAT 求解器的性能。

6 体会

本次课程设计虽然过程一波三折，但收获可谓颇丰。刚开始拿到任务书的时候，非常迷茫，对 SAT 问题的理解不够深刻，阅读 DPLL 算法的内核也云里雾里，完全不知从哪上手。在这种情况下，我决定先搞清楚什么是 SAT 问题和 DPLL 算法是怎么算法。于是开始查阅相关文献，慢慢地，我逐渐理解了 SAT 问题的内核。在我看来，SAT 问题就是一个搜索问题，DPLL 算法就是利用逻辑的二元性来进行二叉搜索，从而判定 CNF 算例的可满足性。完事开头难，在想明白这一点之后，后面要做的就是思想转化为代码，但这个过程也不容易，毕竟有很多东西是做着做着才发现的，有一些细节不动手是搞不明白的，而且编程过程中也会出现很多漏洞，这都需要我们细心耐心地完成工作。就这样慢慢地，一点点地，按照自己的节奏，完成了课程设计。在整个过程中我的体会和主要想法有以下几点。

(1) 工作是一点点做的。刚开始的时候会觉得这个课程设计任务量好大，觉得难以完成，但通过自己一点点地努力，代码一行行地写，函数一个个地编，漏洞一个个地改，逐渐地觉得这个目标也不是那么的遥不可及。说明我们工作中不要想着一口吃个大胖子，要一点点来，心急也吃不了热豆腐，反而要设立一个又一个更小的目标，通过很多小目标的实现来完成一个大目标。

(2) 工作中要更加细致。在编写 CNF 公式的复制函数时由于用到了链表结构，而且我申请新的存储空间时忘记将 next 指针初始化为 NULL，导致了很多野指针的出现，也导致了程序的异常终止，最终耗费了将近一个下午才找到并修复了这个不起眼的漏洞。说明我们做任何事都要细心，编程更是如此，一个小小的疏忽就会带来很大的问题，有时候对程序来说甚至是致命的错误，而且需要耗费后期更多的时间来定位和修复漏洞，往往事倍功半，而且得不偿失。

(3) 动手之前要想清楚。在我们动手写代码之前一定要先想清楚整个算法的流程，不是说面面俱到，但起码也要有个整体的大概思路，这对我们后面的工作具有指导性的意义。如果我们不想清楚就开始动手，写出来的东西会像一团乱码，没有组织性，不清晰，不易读，也不易维护。

(4) 多查阅资料。在获取信息极度容易的今天，我们几乎可以轻而易举地得到我们想要的信息，我们要好好利用这个优势，多查阅资料，阅读文献，学习别人的思想，借鉴他人的解决方案，不仅能加深我们对问题的理解程度，也会对我们有新的启发。

(5) 机器永远是对的。在写程序过程中，有漏洞是再正常不过的事情，但

往往有些漏洞格外隐蔽，非常难看出来，这个时候如果我很长时间都没有看出问题，有时候会怀疑是机器出了错。但事实是，等我又过了几个小时终于找到错误的时候，又会觉得自己非常愚蠢。慢慢地，在写程序中我总是告诫自己，机器永远是对的。哪怕我觉得我的程序写得完全没有漏洞，只要跑起来不是那回事，那就一定是我错了，因为我的程序一定还有哪个地方存在疏漏。

(6) 事情没有我想的那么困难，也绝没有我想的那么简单。刚开始拿到题目的时候，看到数独两个字，就开始有点畏难了，我下意识地觉得这是一个非常宏伟的工程，觉得我没有能力做完这个课程设计，畏难的情绪逐渐将我淹没。但后来查阅了一些文献，慢慢理解了这个问题，又觉得这个问题好像很简单，就是一个类似二叉搜索的实现，但慢慢地在实现过程中又碰到了这样那样的问题，又会觉得这个东西太麻烦了。即使最后慢慢实现了求解器，会发现求解的复杂度太高，然后开始查阅文献、冥思苦想如何进行优化。再到后来实现将二进制数独游戏规则转化为 CNF 公式时看到如此巨大的附加变元的规模，觉得这种复杂度简直不切实际，于是我又开始用以前学的离散数学的知识钻研如何在不添加附加变元的前提下转化，后来做着做着也觉得没有想象得那么复杂，但写起程序来又觉得没我想得那么简单。其实工作就是这样，在做之前会觉得难或者简单，做起来又是另一幅模样，做完之后又有新的感受。我们在做完之前都不能想当然地去评判一项工作，只有彻底做完之后才有资格去总结，去分析，去吸取教训。

(7) 计算机没有人们想象的那么强大，也没有人们想象的那么弱小。在进入计算机专业之前，觉得计算机非常强大，似乎无所不能，能解决人们抛给他的任何问题。后来慢慢地了解到用计算机处理问题是有前提条件的，也了解到处理问题的复杂度也是有限制的。渐渐地我开始觉得计算机能做的事人都能做，只不过它做得比人快一些。而在这次课程设计过程中，刚实现 SAT 求解器的时候，跑个变元数在 200 左右的算例就得跑好久，有时候甚至会内存溢出然后黑屏，我觉得计算机就是废物，算个 200 的算例都算不出来。后来查阅了文献了解到一些求解器可以求解规模非常非常大规模的 CNF 算例的时候又觉得这玩意真神奇，奈何自己能力不够。后来用优化后的求解器去求解数独的时候，我又觉得它的能力简直超乎我想象，一两秒钟能解决我一个小时甚至两个小时解决不了的高阶数独。我对计算机的认识就这样在这次课程设计中有了很大的变化，我对计算机学科和计算机的认识也是在一次次动手实践的过程中越来越深刻。

参考文献

- [1] Cook, Stephen A . [ACM Press the third annual ACM symposium - Shaker Heights, Ohio, United States (1971.05.03-1971.05.05)] Proceedings of the third annual ACM symposium on Theory of computing, - STOC \71 - The complexity of theorem-proving procedures[C]// ACM, 1971:151-158.
- [2] 杜忠和.基于 CDCL 结构的 SAT 问题优化策略.成都:西南交通大学,2018.5

附录 源程序

1. Clause.c

```
#include "Clause.h"

#ifndef NULL
#define NULL 0
#endif/*NULL*/

#ifndef TRUE
#define TRUE 1
#endif/*TRUE*/

#ifndef FALSE
#define FALSE 0
#endif/*FALSE*/

struct node{
    int ele;
    Literal next;
};

Clause createClause(FILE *in){
    int tmp,i=0;
    Clause C=malloc(sizeof(struct node));
    Literal L=C;
    C->next=NULL;
    do{
        fscanf(in,"%d",&tmp);
        if(tmp!=0){
            L->next=malloc(sizeof(struct node));
            L=L->next;
            L->ele=tmp;
            L->next=NULL;
        }
    }while(tmp!=0);
    return C;
}
```



```
void destroyClause(Clause C){
    Literal L=C->next,tmp;
    while(L){
        tmp=L->next;
        free(L);
        L=tmp;
    }
    free(C);
}
```

```
int ExistLiteral(Clause C,int k){
    int r=FALSE;
    Literal L=C->next;
    int i=1;
    while(L){
        if(L->ele==k){
            r=TRUE;
            break;
        }
        L=L->next;
    }
    return r;
}
```

```
void removeLiteral(Clause C,int k){
    Literal L=C;
    while(L&&L->next){
        if(L->next->ele==k){
            Literal tmp=L->next;
            L->next=L->next->next;
            free(tmp);
        }
        L=L->next;
    }
}
```

```
Clause copyClause(Clause C){
    Clause _C=malloc(sizeof(struct node));
    _C->next=NULL;
```

```
Literal _L=_C,L=C->next;
while(L){
    _L->next=malloc(sizeof(struct node));
    _L=_L->next;
    _L->next=NULL;
    _L->ele=L->ele;
    L=L->next;
}
return _C;
}
```

```
int isUnitClause(Clause C){
    if(C->next) return C->next->next==NULL;
    else return FALSE;
}
```

```
int checkClause(Clause C,int *assignment){
    int r=FALSE;
    Literal L=C->next;
    int k;
    while(L){
        k=L->ele;
        if(k>0&&assignment[k-1]==1){
            r=TRUE;
            break;
        }
        else if(k<0 && assignment[-k-1]==0){
            r=TRUE;
            break;
        }
        L=L->next;
    }
    return r;
}
```

```
void printClause(Clause C){
    Literal L=C->next;
    while(L){
        printf("%4d ",L->ele);
        L=L->next;
    }
}
```

```
}
```

```
}
```

2. CNF.c

```
#include "CNF.h"
```

```
struct clause{
    Clause C;
    conjunct next;
};
```

```
cnf createCNF(FILE *in,int *_NOV){
    char tmp[20];
    do{
        fscanf(in,"%s",tmp);
    } while(tmp[0]!='p' || tmp[1]!=0);
    int NOC; //Number of Conjuncts
    fscanf(in,"%s %d %d",tmp,_NOV,&NOC);
    int i;
    cnf p=malloc(sizeof(struct clause));
    p->next=NULL;
    conjunct c=p;
    for(i=0;i<NOC;i++){
        c->next=malloc(sizeof(struct clause));
        c=c->next;
        c->C=createClause(in);
        c->next=NULL;
    }
    return p;
}
```

```
cnf addUnitClause(cnf p,int k){
    cnf _p=malloc(sizeof(struct clause));
    _p->next=malloc(sizeof(struct clause));
    conjunct c=p->next,_c=_p->next;
    _c->C=malloc(sizeof(struct node));
    _c->C->next=malloc(sizeof(struct node));
    _c->C->next->ele=k;
    _c->C->next->next=NULL;
    while(c){
        _c->next=malloc(sizeof(struct clause));
```

```
    _c=_c->next;
    _c->C=copyClause(c->C);
    _c->next=NULL;
    c=c->next;
}
return _p;
}
```

```
void removeClause(cnf p,conjunct c){
    conjunct tmp=c->next;
    c->next=c->next->next;
    destroyClause(tmp->C);
    free(tmp);
}
```

```
int ExistNullClause(cnf p){
    int r=FALSE;
    conjunct c=p->next;
    while(c){
        if(!c->C->next){
            r=TRUE;
            break;
        }
        c=c->next;
    }
    return r;
}
```

```
int ExistUnitClause(cnf p){
    int r=0;
    conjunct c=p->next;
    while(c){
        if(isUnitClause(c->C)){
            r=c->C->next->ele;
            break;
        }
        c=c->next;
    }
    return r;
}
```



```

void clearLiteral(cnf p,int k){
    conjunct c=p;
    int i=1;
    while(c&& c->next){
        if(ExistLiteral(c->next->C,k)){
            removeClause(p,c);
        }
        else {
            removeLiteral(c->next->C,-k);
            c=c->next;
        }
    }
}

```

```

void printCNF(cnf p){
    conjunct c=p->next;
    int i=0;
    while(c){
        printClause(c->C);
        printf("    0\n");
        c=c->next;
        i++;
    }
}

```

```

void destroyCNF(cnf p){
    conjunct c=p->next;
    while(c){
        destroyClause(c->C);
        conjunct tmp=c->next;
        free(c);
        c=tmp;
    }
    free(p);
}

```

3. MP.c

```

#include "MP.h"

```

```

int allPositive(Clause C){
    int r=0;
    Literal L=C->next;
    while(L){
        if(L->ele>0)r++;
        else{
            r=0;
            break;
        }
        L=L->next;
    }
    return r;
}

int MPstrategy(cnf p,int NOV){
    int r=p->next->C->next->ele;
    int minlen=NOV;
    conjunct c=p->next;
    while(c){
        int len=allPositive(c->C);
        if(len && len<=minlen){
            minlen=len;
            r=c->C->next->ele;
        }
        c=c->next;
    }
    return r;
}

```

4. JW.c

```

#include "JW.h"

float float_abs(float a){
    if(a>0)return a;
    else return -a;
}

int JW_length(Clause C,int k){
    Literal L=C->next;
    int length=0,flag=0;
    while(L){

```

```

        if(L->ele==k)flag=1;
        else if(L->ele==-k)flag=-1;
        length++;
        L=L->next;
    }
    return flag*length;
}

int FindJWMax(float *a,int n){
    int i,r=1;
    float max=float_abs(a[0]);
    for(i=1;i<n;i++){
        if(float_abs(a[i])>max){
            if(a[i]>0) r=i+1;
            else r=-i-1;
            max=float_abs(a[i]);
        }
    }
    return r;
}

float JWvalue(cnf p,int v){
    conjunct c=p->next;
    float negative=.0,positive=.0;
    int len;
    while(c){
        len=JW_length(c->C,v);
        if(len>0) positive+=(float)pow(2,-len);
        else if(len<0) negative-=(float)pow(2,len);
        c=c->next;
    }
    if(positive+negative>0) return positive;
    else return negative;
}

int JWstrategy(cnf p,int NOV){
    int i=0;
    float JW[NOV];
    for(i=0;i<NOV;i++)
        JW[i]=JWvalue(p,i+1);
}

```

```

        return FindJWMax(JW,NOV);
    }

```

5. DPLL.c

```
#include "DPLL.h"
```

```

int FindMax(int *a,int n){
    int cursor=0,i;    //下标
    for(i=1;i<n;i++)
        if(a[i]>a[cursor])
            cursor=i;
    return cursor+1;
}

```

```

int DPLL(cnf p,int *assignment,int NOV,int STRATEGY,int *_times){
    int k;
    *_times=*_times+1;
    while(k=ExistUnitClause(p)){
        if(k>0)assignment[k-1]=1;
        else assignment[-k-1]=0;
        clearLiteral(p,k);
        if(!p->next) return TRUE;
        else if(ExistNullClause(p)) return FALSE;
    }
    if(STRATEGY==1)k=MPstrategy(p,NOV);
    else if(STRATEGY==2)k=JWstrategy(p,NOV);
    else k=p->next->C->next->ele;
    cnf p_p=addUnitClause(p,k);
    if(DPLL(p_p,assignment,NOV,STRATEGY,_times)){
        destroyCNF(p_p);
        return TRUE;
    }
    else{
        cnf n_p=addUnitClause(p,-k);
        assignment[abs(k)-1]=-1;
        if(DPLL(n_p,assignment,NOV,STRATEGY,_times)){
            destroyCNF(n_p);
            return TRUE;
        }
        else {
            assignment[abs(k)-1]=-1;

```



```

        destroyCNF(n_p);
        return FALSE;
    }
}
}

```

```

int checkAssignment(cnf p,int *assignment){
    int r=TRUE;
    conjunct c=p->next;
    while(c){
        if(!checkClause(c->C,assignment)){
            r=FALSE;
            break;
        }
        c=c->next;
    }
    return r;
}

```

```

void output_file(char *filename,int r,int *assignment,int NOV,double
w_time,int times){
    int len=strlen(filename);
    filename[len-3]='r';
    filename[len-2]='e';
    filename[len-1]='s';
    FILE *out=fopen(filename,"w");
    if(r){
        fprintf(out,"r 1\n");
        int i;
        fprintf(out,"v");
        for(i=0;i<NOV;i++)
            if(assignment[i])fprintf(out," %d",i+1);
            else if(!assignment[i])fprintf(out," %d",-i+1));
            else fprintf(out," 0");
        fprintf(out,"\nt %.0lfms %d(DPLL calls)",w_time,times);
    }
}

```

6. CNF_ImprovedDataStructure.c

```

#include "CNF_ImprovedDataStructure.h"

```

```

#ifndef NULL
#define NULL 0
#endif/*NULL*/

cnf_i parseCNF(FILE *in){
    char tmp[20];
    do{
        fscanf(in,"%s",tmp);
    }while(tmp[0]!='p' || tmp[1]!=0);
    cnf_i c=malloc(sizeof(CNF_I));
    fscanf(in,"%s %d %d",tmp,&c->NOV,&c->NOC);
    c->p=malloc(c->NOV*sizeof(info_var));
    c->ad=malloc(c->NOC*sizeof(clause_i));
    c->orinal=malloc(sizeof(formula));
    c->orinal->next=NULL;
    int i,L;
    for(i=0;i<c->NOV;i++){
        c->p[i].fp=0;
        c->p[i].fn=0;
        c->p[i].in=malloc(sizeof(index));
        c->p[i].in->next=NULL;
    }
    clause_i cl=c->orinal;
    for(i=0;i<c->NOC;i++){
        cl->next=malloc(sizeof(formula));
        c->ad[i]=cl->next;
        cl->next->s=UNCERTAIN;
        cl->next->n=0;
        cl->next->next=NULL;
        cl->next->hc=malloc(sizeof(node_i));
        literal lit=cl->next->hc;
        lit->next=NULL;
        while(1){
            fscanf(in,"%d",&L);
            if(!L)break;
            cl->next->n++;
            lit->next=malloc(sizeof(node_i));
            lit->next->L.v=L;
            lit->next->L.s=UNCERTAIN;
            lit->next->next=NULL;

```

```

        lit=lit->next;
        index_number tmp=c->p[abs(L)-1].in->next;
        c->p[abs(L)-1].in->next=malloc(sizeof(index));
        c->p[abs(L)-1].in->next->k=i;
        c->p[abs(L)-1].in->next->next=tmp;
        if(L>0){
            c->p[abs(L)-1].in->next->pmc=1;
            c->p[abs(L)-1].fp++;
        }
        else{
            c->p[abs(L)-1].in->next->pmc=-1;
            c->p[abs(L)-1].fn++;
        }
    }
    cl=cl->next;
}
return c;
}

```

```

void outputCNF(cnf_i c){
    int i;
    clause_i cl=c->orinal->next;
    while(cl){
        printf("      c");
        literal lit=cl->hc->next;
        while(lit){
            printf(" %5d",lit->L.v);
            lit=lit->next;
        }
        printf("%5d\n",0);
        cl=cl->next;
    }
}

```

```

void clearLiteral_2(cnf_i c,int layer,int k){
    int L=abs(k);
    index_number idx=c->p[L-1].in->next;
    literal lit;
    int isNull;
    while(idx){

```

```

        if(c->ad[idx->k]->s>=0){
            lit=c->ad[idx->k]->hc->next;
            isNull=1;
            while(lit){
                if(lit->L.s>=SATISFIED){
                    isNull=0;
                }
                else if(lit->L.s==UNCERTAIN && abs(lit->L.v)==L){
                    c->ad[idx->k]->n--;
                    if(lit->L.v==k){
                        lit->L.s=SATISFIED*layer;
                        c->ad[idx->k]->s=SATISFIED;
                        isNull=0;
                    }
                    else
                        lit->L.s=REMOVED*layer;
                }
                else if(lit->L.s==UNCERTAIN){
                    isNull=0;
                }
                lit=lit->next;
            }
            if(isNull)
                c->ad[idx->k]->s=UNSATISFIED;
        }
        idx=idx->next;
    }
}

```

```

void retract(cnf_i c,int layer,int k){
    int L=abs(k);
    index_number idx=c->p[L-1].in->next;
    literal lit;
    int isSatisfied;
    while(idx){
        lit=c->ad[idx->k]->hc->next;
        isSatisfied=0;
        while(lit){
            if(lit->L.s==SATISFIED*layer){
                lit->L.s=UNCERTAIN;
            }
        }
        idx=idx->next;
    }
}

```



```

        c->ad[idx->k]->n++;
    }
    else if(lit->L.s==UNSATISFIED*layer){
        lit->L.s=UNCERTAIN;
        c->ad[idx->k]->n++;
    }
    else if(lit->L.s>=SATISFIED)
        isSatisfied=1;
    lit=lit->next;
}
if(isSatisfied)
    c->ad[idx->k]->s=SATISFIED;
else
    c->ad[idx->k]->s=UNCERTAIN;
idx=idx->next;
}
}

```

```

int FindUnitClause_2(cnf_i c){
    int number=0,r;
    clause_i cl=c->orinal->next;
    literal lit;
    while(cl){
        number=0;
        if(cl->s==UNCERTAIN){
            lit=cl->hc->next;
            while(lit){
                if(lit->L.s==UNCERTAIN){
                    r=lit->L.v;
                    number++;
                }
                if(number>1){
                    r=0;
                    break;
                }
                lit=lit->next;
            }
        }
        if(number==1)
            break;
    }
}

```

```

        cl=cl->next;
    }
    return r;
}

int RandomStrategy(cnf_i c){
    clause_i cl=c->orinal->next;
    while(cl){
        if(cl->s==UNCERTAIN){
            literal lit=cl->hc->next;
            while(lit){
                if(lit->L.s==UNCERTAIN)
                    return lit->L.v;
                lit=lit->next;
            }
        }
        cl=cl->next;
    }
}

int checkCNF(cnf_i c){
    int r=SATISFIED;
    clause_i cl=c->orinal->next;
    while(cl){
        if(cl->s==UNSATISFIED)
            return UNSATISFIED;
        else if(cl->s==UNCERTAIN)
            r=UNCERTAIN;
        cl=cl->next;
    }
    return r;
}

int checkAssignment_2(cnf_i c,int *assignment){
    clause_i cl=c->orinal->next;
    literal lit;
    int r=0,k;
    while(cl){
        lit=cl->hc->next;
        r=0;

```

```

        while(lit){
            k=lit->L.v;
            if(assignment[abs(k)-1]==k){
                r=1;
                break;
            }
            lit=lit->next;
        }
        if(!r)
            return FALSE;
        cl=cl->next;
    }
    return TRUE;
}

```

7. DPLL_ImprovedDataStructure_nonRecursive.c

```

int DPLL_nonRecursive(cnf_i c,int *assignment){
    int layer=1;
    int k;
    int *a=malloc(c->NOV*sizeof(int));
    int *b=malloc(c->NOV*sizeof(int));
    int *s=malloc(c->NOV*sizeof(int));
    for(k=0;k<c->NOV;k++){
        a[k]=0;
        b[k]=0;
        s[k]=0;
    }
    int flag=1;
    int check,check_;
    while(layer>0 && layer<=c->NOV){
        if(!flag){
            retract(c,layer,a[layer-1]);
            if(s[layer-1]){
                assignment[abs(a[layer-1])-1]=0;
                s[layer-1]=0;
                a[layer-1]=0;
                b[layer-1]=0;
                layer--;
            }
            else if(b[layer-1]){
                assignment[abs(a[layer-1])-1]=0;

```

```

        if(check==UNSATISFIED){
            retract(c,layer,a[layer-1]);
            assignment[abs(a[layer-1])-1]=0;
            a[layer-1]=0;
            b[layer-1]=0;
            flag=0;
            layer--;
        }
        else if(check==SATISFIED)
            return TRUE;
        else{
            flag=1;
            layer++;
        }
    }
}
else{
    k=FindUnitClause_2(c);
    if(k)
        s[layer-1]=1;
    else
        k=RandomStrategy(c);
    assignment[abs(k)-1]=k;
    a[layer-1]=k;
    clearLiteral_2(c,layer,a[layer-1]);
    check=checkCNF(c);
    if(check==UNSATISFIED){
        retract(c,layer,a[layer-1]);
        assignment[abs(k)-1]=-k;
        a[layer-1]=-k;
    }
}

```



```

        b[layer-1]=1;
        clearLiteral_2(c,layer,a[layer-1]);
        check_=checkCNF(c);
        if(check_==UNSATISFIED){
            retract(c,layer,a[layer-1]);
            assignment[abs(k)-1]=0;
            a[layer-1]=0;
            b[layer-1]=0;
            flag=0;
            layer--;
        }
        else if(check_==SATISFIED)
            return TRUE;
        else
            layer++;
    }
    else if(check==SATISFIED)
        return TRUE;
    else
        layer++;
}
}
return FALSE;
}

```

8. SudokuCNF.c

```
#include "SudokuCNF.h"
```

```

int order(int degree,int row,int column){
    return (row-1)*degree+column;
}

```

```

void addrestriction(cnf p,int k){
    conjunct tmp=p->next;
    p->next=malloc(sizeof(struct clause));
    p->next->next=tmp;
    p->next->C=malloc(sizeof(struct node));
    p->next->C->next=NULL;
    Literal L=p->next->C;
    L->next=malloc(sizeof(struct node));
    L=L->next;
}

```

```

L->ele=k;
L->next=NULL;
}

```

```

void addClause(cnf p,int *a,int n){
    conjunct tmp=p->next;
    p->next=malloc(sizeof(struct clause));
    p->next->next=tmp;
    p->next->C=malloc(sizeof(struct node));
    p->next->C->next=NULL;
    Literal L=p->next->C;
    int i;
    for(i=0;i<n;i++){
        L->next=malloc(sizeof(struct node));
        L=L->next;
        L->ele=a[i];
        L->next=NULL;
    }
}

```

```

void addClauses_rule1(cnf p,int degree){
    int row,column,i;
    int a[3];
    for(row=1;row<=degree;row++){
        for(column=1;column<=degree-2;column++){
            for(i=0;i<3;i++){
                a[i]=order(degree,row,column+i);
                addClause(p,a,3);
            }
            for(i=0;i<3;i++){
                a[i]=-order(degree,row,column+i);
                addClause(p,a,3);
            }
        }
    }
    for(column=1;column<=degree;column++){
        for(row=1;row<=degree-2;row++){
            for(i=0;i<3;i++){
                a[i]=order(degree,row+i,column);
                addClause(p,a,3);
            }
            for(i=0;i<3;i++){
                a[i]=-order(degree,row+i,column);
            }
        }
    }
}

```

```

        addClause(p,a,3);
    }
}

void addClauses_rule2_4(cnf p){
    int column[3],row[3];
    int _row,_column;
    int a[3];
    for(_row=1;_row<=4;_row++){
        for(column[0]=1;column[0]<=2;column[0]++){
            for(column[1]=column[0]+1;column[1]<=3;column[1]++){
                for(column[2]=column[1]+1;column[2]<=4;column[2]++){
                    a[0]=order(4,_row,column[0]);
                    a[1]=order(4,_row,column[1]);
                    a[2]=order(4,_row,column[2]);
                    addClause(p,a,3);
                    a[0]=-order(4,_row,column[0]);
                    a[1]=-order(4,_row,column[1]);
                    a[2]=-order(4,_row,column[2]);
                    addClause(p,a,3);
                }
            }
        }
    }
    for(_column=1;_column<=4;_column++){
        for(row[0]=1;row[0]<=2;row[0]++){
            for(row[1]=row[0]+1;row[1]<=3;row[1]++){
                for(row[2]=row[1]+1;row[2]<=4;row[2]++){
                    a[0]=order(4,row[0],_column);
                    a[1]=order(4,row[1],_column);
                    a[2]=order(4,row[2],_column);
                    addClause(p,a,3);
                    a[0]=-order(4,row[0],_column);
                    a[1]=-order(4,row[1],_column);
                    a[2]=-order(4,row[2],_column);
                    addClause(p,a,3);
                }
            }
        }
    }
}

```

```

    }
}

void addClauses_rule2_6(cnf p){
    int column[4],row[4];
    int _row,_column;
    int a[4];
    for(_row=1;_row<=6;_row++){
        for(column[0]=1;column[0]<=3;column[0]++){
            for(column[1]=column[0]+1;column[1]<=4;column[1]++){
                for(column[2]=column[1]+1;column[2]<=5;column[2]++){
                    for(column[3]=column[2]+1;column[3]<=6;column[3]++){
                        a[0]=order(6,_row,column[0]);
                        a[1]=order(6,_row,column[1]);
                        a[2]=order(6,_row,column[2]);
                        a[3]=order(6,_row,column[3]);
                        addClause(p,a,4);
                        a[0]=-order(6,_row,column[0]);
                        a[1]=-order(6,_row,column[1]);
                        a[2]=-order(6,_row,column[2]);
                        a[3]=-order(6,_row,column[3]);
                        addClause(p,a,4);
                    }
                }
            }
        }
    }
    for(_column=1;_column<=6;_column++){
        for(row[0]=1;row[0]<=3;row[0]++){
            for(row[1]=row[0]+1;row[1]<=4;row[1]++){
                for(row[2]=row[1]+1;row[2]<=5;row[2]++){
                    for(row[3]=row[2]+1;row[3]<=6;row[3]++){
                        a[0]=order(6,row[0],_column);
                        a[1]=order(6,row[1],_column);
                        a[2]=order(6,row[2],_column);
                        a[3]=order(6,row[3],_column);
                        addClause(p,a,4);
                        a[0]=-order(6,row[0],_column);
                        a[1]=-order(6,row[1],_column);

```



```

for(_column=1;_column<=8;_column++){
    for(row[0]=1;row[0]<=4;row[0]++){
        for(row[1]=row[0]+1;row[1]<=5;row[1]++){
            for(row[2]=row[1]+1;row[2]<=6;row[2]++){
                for(row[3]=row[2]+1;row[3]<=7;row[3]++){
                    for(row[4]=row[3]+1;row[4]<=8;row[4]++){
                        a[0]=order(8,row[0],_column);
                        a[1]=order(8,row[1],_column);
                        a[2]=order(8,row[2],_column);
                        a[3]=order(8,row[3],_column);
                        a[4]=order(8,row[4],_column);
                        addClause(p,a,5);
                        a[0]=-order(8,row[0],_column);
                        a[1]=-order(8,row[1],_column);
                        a[2]=-order(8,row[2],_column);
                        a[3]=-order(8,row[3],_column);
                        a[4]=-order(8,row[4],_column);
                        addClause(p,a,5);
                    }
                }
            }
        }
    }
}

void addClauses_rule2_10(cnf p){
    int column[6],row[6];
    int _row,_column;
    int a[6];
    for(_row=1;_row<=10;_row++){
        for(column[0]=1;column[0]<=5;column[0]++){
            for(column[1]=column[0]+1;column[1]<=6;column[1]++){
                for(column[2]=column[1]+1;column[2]<=7;column[2]++){

for(column[3]=column[2]+1;column[3]<=8;column[3]++){

for(column[4]=column[3]+1;column[4]<=9;column[4]++){

for(column[5]=column[4]+1;column[5]<=10;column[5]++){

```



```

    }
}
for(_column=1;_column<=10;_column++){
    for(row[0]=1;row[0]<=6;row[0]++){
        for(row[1]=row[0]+1;row[1]<=7;row[1]++){
            for(row[2]=row[1]+1;row[2]<=8;row[2]++){
                for(row[3]=row[2]+1;row[3]<=9;row[3]++){
                    for(row[4]=row[3]+1;row[4]<=10;row[4]++){
                        for(row[5]=row[4]+1;row[5]<=11;row[5]++){
                            for(row[6]=row[5]+1;row[6]<=12;row[6]++){
                                a[0]=order(12,row[0],_column);
                                a[1]=order(12,row[1],_column);
                                a[2]=order(12,row[2],_column);
                                a[3]=order(12,row[3],_column);
                                a[4]=order(12,row[4],_column);
                                a[5]=order(12,row[5],_column);
                                a[6]=order(12,row[6],_column);
                                addClause(p,a,7);
                                a[0]=-order(12,row[0],_column);
                                a[1]=-order(12,row[1],_column);
                                a[2]=-order(12,row[2],_column);
                                a[3]=-order(12,row[3],_column);
                                a[4]=-order(12,row[4],_column);
                                a[5]=-order(12,row[5],_column);
                                a[6]=-order(12,row[6],_column);
                                addClause(p,a,7);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
}
}
}

```

```

void addClauses_rule2_14(cnf p){
    int column[8],row[8];
    int _row,_column;
    int a[8];
    for(_row=1;_row<=14;_row++){
        for(column[0]=1;column[0]<=7;column[0]++){
            for(column[1]=column[0]+1;column[1]<=8;column[1]++){
                for(column[2]=column[1]+1;column[2]<=9;column[2]++){

for(column[3]=column[2]+1;column[3]<=10;column[3]++){

for(column[4]=column[3]+1;column[4]<=11;column[4]++){

for(column[5]=column[4]+1;column[5]<=12;column[5]++){

for(column[6]=column[5]+1;column[6]<=13;column[6]++){

for(column[7]=column[6]+1;column[7]<=14;column[7]++){
                    a[0]=order(14,_row,column[0]);
                    a[1]=order(14,_row,column[1]);
                    a[2]=order(14,_row,column[2]);
                    a[3]=order(14,_row,column[3]);
                    a[4]=order(14,_row,column[4]);
                    a[5]=order(14,_row,column[5]);
                    a[6]=order(14,_row,column[6]);
                    a[7]=order(14,_row,column[7]);
                    addClause(p,a,8);
                    a[0]=-order(14,_row,column[0]);
                    a[1]=-order(14,_row,column[1]);
                    a[2]=-order(14,_row,column[2]);
                    a[3]=-order(14,_row,column[3]);
                    a[4]=-order(14,_row,column[4]);
                    a[5]=-order(14,_row,column[5]);
                    a[6]=-order(14,_row,column[6]);

```



```
    }  
    }  
    }  
    }  
    }  
    }  
    }  
}
```

```
void addClauses_rule2(cnf p,int degree){  
    switch(degree){  
        case 4:  
            addClauses_rule2_4(p);  
            break;  
        case 6:  
            addClauses_rule2_6(p);  
            break;  
        case 8:  
            addClauses_rule2_8(p);  
            break;  
        case 10:  
            addClauses_rule2_10(p);  
            break;  
        case 12:  
            addClauses_rule2_12(p);  
            break;  
        case 14:  
            addClauses_rule2_14(p);  
            break;  
    }  
}
```

```
void addClauses_rule3_4(cnf p){  
    int flag[4],a[8];  
    int row1,row2,column1,column2,i;  
    for(row1=1;row1<=3;row1++){  
        for(row2=row1+1;row2<=4;row2++){  
            for(flag[0]=0;flag[0]<2;flag[0]++){
```



```

        for(flag[1]=0;flag[1]<2;flag[1]++){
            for(flag[2]=0;flag[2]<2;flag[2]++){
                for(flag[3]=0;flag[3]<2;flag[3]++){
                    for(i=0;i<4;i++){
                        if(flag[i]){
                            a[2*i]=order(4,row1,i+1);
                            a[2*i+1]=order(4,row2,i+1);
                        }
                        else{
                            a[2*i]=-order(4,row1,i+1);
                            a[2*i+1]=-order(4,row2,i+1);
                        }
                    }
                    addClause(p,a,8);
                }
            }
        }
    }
}

for(column1=1;column1<=3;column1++){
    for(column2=column1+1;column2<=4;column2++){
        for(flag[0]=0;flag[0]<2;flag[0]++){
            for(flag[1]=0;flag[1]<2;flag[1]++){
                for(flag[2]=0;flag[2]<2;flag[2]++){
                    for(flag[3]=0;flag[3]<2;flag[3]++){
                        for(i=0;i<4;i++){
                            if(flag[i]){
                                a[2*i]=order(4,i+1,column1);
                                a[2*i+1]=order(4,i+1,column2);
                            }
                            else{
                                a[2*i]=-order(4,i+1,column1);
                                a[2*i+1]=-order(4,i+1,column2);
                            }
                        }
                    }
                    addClause(p,a,8);
                }
            }
        }
    }
}

```

```

    }
  }
}

void addClauses_rule3_6(cnf p){
  int flag[6],a[12];
  int row1,row2,column1,column2,i;
  for(row1=1;row1<=5;row1++){
    for(row2=row1+1;row2<=6;row2++){
      for(flag[0]=0;flag[0]<2;flag[0]++){
        for(flag[1]=0;flag[1]<2;flag[1]++){
          for(flag[2]=0;flag[2]<2;flag[2]++){
            for(flag[3]=0;flag[3]<2;flag[3]++){
              for(flag[4]=0;flag[4]<2;flag[4]++){
                for(flag[5]=0;flag[5]<2;flag[5]++){
                  for(i=0;i<6;i++){
                    if(flag[i]){
                      a[2*i]=order(6,row1,i+1);
                      a[2*i+1]=order(6,row2,i+1);
                    }
                    else{
                      a[2*i]=-order(6,row1,i+1);
                      a[2*i+1]=-order(6,row2,i+1);
                    }
                  }
                  addClause(p,a,12);
                }
              }
            }
          }
        }
      }
    }
  }
  for(column1=1;column1<=5;column1++){
    for(column2=column1+1;column2<=6;column2++){
      for(flag[0]=0;flag[0]<2;flag[0]++){
        for(flag[1]=0;flag[1]<2;flag[1]++){
          for(flag[2]=0;flag[2]<2;flag[2]++){

```

```

        for(flag[3]=0;flag[3]<2;flag[3]++){
            for(flag[4]=0;flag[4]<2;flag[4]++){
                for(flag[5]=0;flag[5]<2;flag[5]++){
                    for(i=0;i<6;i++){
                        if(flag[i]){
                            a[2*i]=order(6,i+1,column1);

a[2*i+1]=order(6,i+1,column2);

                        }
                        else{

a[2*i]=-order(6,i+1,column1);

a[2*i+1]=-order(6,i+1,column2);

                        }
                    }
                }
            }
        }
    }
}

void addClauses_rule3_8(cnf p){
    int flag[8],a[16];
    int row1,row2,column1,column2,i;
    for(row1=1;row1<=7;row1++){
        for(row2=row1+1;row2<=8;row2++){
            for(flag[0]=0;flag[0]<2;flag[0]++){
                for(flag[1]=0;flag[1]<2;flag[1]++){
                    for(flag[2]=0;flag[2]<2;flag[2]++){
                        for(flag[3]=0;flag[3]<2;flag[3]++){
                            for(flag[4]=0;flag[4]<2;flag[4]++){
                                for(flag[5]=0;flag[5]<2;flag[5]++){

for(flag[6]=0;flag[6]<2;flag[6]++){

```



```

        if(flag[i]){

a[2*i]=order(8,i+1,column1);

a[2*i+1]=order(8,i+1,column2);

        }
        else{

a[2*i]=-order(8,i+1,column1);

a[2*i+1]=-order(8,i+1,column2);

        }
    }
    addClause(p,a,16);
}
}
}
}
}
}
}
}
}
}
}

void addClauses_rule3_10(cnf p){
    int flag[10],a[20];
    int row1,row2,column1,column2,i;
    for(row1=1;row1<=9;row1++){
        for(row2=row1+1;row2<=10;row2++){
            for(flag[0]=0;flag[0]<2;flag[0]++){
                for(flag[1]=0;flag[1]<2;flag[1]++){
                    for(flag[2]=0;flag[2]<2;flag[2]++){
                        for(flag[3]=0;flag[3]<2;flag[3]++){
                            for(flag[4]=0;flag[4]<2;flag[4]++){
                                for(flag[5]=0;flag[5]<2;flag[5]++){

for(flag[6]=0;flag[6]<2;flag[6]++){

```



```

int flag[12],a[24];
int row1,row2,column1,column2,i;
for(row1=1;row1<=11;row1++){
    for(row2=row1+1;row2<=12;row2++){
        for(flag[0]=0;flag[0]<2;flag[0]++){
            for(flag[1]=0;flag[1]<2;flag[1]++){
                for(flag[2]=0;flag[2]<2;flag[2]++){
                    for(flag[3]=0;flag[3]<2;flag[3]++){
                        for(flag[4]=0;flag[4]<2;flag[4]++){
                            for(flag[5]=0;flag[5]<2;flag[5]++){

for(flag[6]=0;flag[6]<2;flag[6]++){

for(flag[7]=0;flag[7]<2;flag[7]++){

    for(flag[8]=0;flag[8]<2;flag[8]++){

        for(flag[9]=0;flag[9]<2;flag[9]++){

            for(flag[10]=0;flag[10]<2;flag[10]++){

                for(flag[11]=0;flag[11]<2;flag[11]++){

for(i=0;i<12;i++){

    if(flag[i]){

a[2*i]=order(12,row1,i+1);

a[2*i+1]=order(12,row2,i+1);

}
else{

a[2*i]=-order(12,row1,i+1);

a[2*i+1]=-order(12,row2,i+1);

}

}

addClause(p,a,24);

```



```
for(flag[6]=0;flag[6]<2;flag[6]++){

for(flag[7]=0;flag[7]<2;flag[7]++){

for(flag[8]=0;flag[8]<2;flag[8]++){

for(flag[9]=0;flag[9]<2;flag[9]++){

for(flag[10]=0;flag[10]<2;flag[10]++){

for(flag[11]=0;flag[11]<2;flag[11]++){

for(flag[12]=0;flag[12]<2;flag[12]++){

for(flag[13]=0;flag[13]<2;flag[13]++){

for(i=0;i<12;i++){

if(flag[i]){

a[2*i]=order(14,row1,i+1);

a[2*i+1]=order(14,row2,i+1);

}

else{

a[2*i]=-order(14,row1,i+1);

a[2*i+1]=-order(14,row2,i+1);

}

}

addClause(p,a,28);

}

}

}

}
```

```
        }
    }
}

for(column1=1;column1<=13;column1++){
    for(column2=column1+1;column2<=14;column2++){
        for(flag[0]=0;flag[0]<2;flag[0]++){
            for(flag[1]=0;flag[1]<2;flag[1]++){
                for(flag[2]=0;flag[2]<2;flag[2]++){
                    for(flag[3]=0;flag[3]<2;flag[3]++){
                        for(flag[4]=0;flag[4]<2;flag[4]++){
                            for(flag[5]=0;flag[5]<2;flag[5]++){
                                for(flag[6]=0;flag[6]<2;flag[6]++){
                                    for(flag[7]=0;flag[7]<2;flag[7]++){
                                        for(flag[8]=0;flag[8]<2;flag[8]++){
                                            for(flag[9]=0;flag[9]<2;flag[9]++){
                                                for(flag[10]=0;flag[10]<2;flag[10]++){
                                                    for(flag[11]=0;flag[11]<2;flag[11]++){
                                                        for(flag[12]=0;flag[12]<2;flag[12]++){
                                                            for(flag[13]=0;flag[13]<2;flag[13]++){
                                                                for(i=0;i<12;i++){
                                                                    if(flag[i]){
```



```
        case 6:
            addClauses_rule3_6(p);
            break;
        case 8:
            addClauses_rule3_8(p);
            break;
        case 10:
            addClauses_rule3_10(p);
            break;
        case 12:
            addClauses_rule3_12(p);
            break;
        case 14:
            addClauses_rule3_14(p);
            break;
    }
}
```

```
cnf createSudokuCNF(int degree){
    cnf p=malloc(sizeof(struct clause));
    p->next=NULL;
    addClauses_rule1(p,degree);
    addClauses_rule2(p,degree);
    addClauses_rule3(p,degree);
    return p;
}
```

9. BinaryPuzzle.c

```
#include "BinaryPuzzle.h"

cnf RandomFilled(int degree,int *assignment){
    srand((unsigned)time(NULL));
    cnf p=createSudokuCNF(degree);
    int i,o,v,times;
    cnf t[degree+4];
    do{
        times=0;
        for(i=0;i<degree+4;i++){
            o=rand()%(degree*degree-1)+1;
            v=rand()%2;
            if(!v) o=-o;
        }
    } while(times==0);
}
```

```

        if(i==0)t[i]=addUnitClause(p,o);
        else t[i]=addUnitClause(t[i-1],o);
    }
    for(i=0;i<degree*degree;i++)
        assignment[i]=-1;
    if(DPLL(t[degree+3],assignment,degree*degree,3,&times)){
        for(i=0;i<degree*degree;i++)
            if(assignment[i]==-1) assignment[i]=rand()%2;
        for(i=0;i<degree+4;i++)
            destroyCNF(t[i]);
        break;
    }
    else{
        for(i=0;i<degree+4;i++)
            destroyCNF(t[i]);
    }
}while(1);
return p;
}

int complement(int degree,int mode,cnf p,int *assignment){
    int a[degree][degree],i,j,chosen[degree*degree],o;
    for(i=0;i<degree;i++)
        for(j=0;j<degree;j++)
            a[i][j]=assignment[i*degree+j];
    for(i=0;i<degree;i++)
        for(j=0;j<degree-1;j++)
            if(a[i][j]!=-1 && a[i][j]==a[i][j+1]){
                if(j-1>=0) a[i][j-1]=-1;
                if(j+2<degree) a[i][j+2]=-1;
            }
    for(i=0;i<degree-1;i++)
        for(j=0;j<degree;j++)
            if(a[i][j]!=-1 && a[i][j]==a[i+1][j]){
                if(i-1>=0) a[i-1][j]=-1;
                if(i+2<degree) a[i+2][j]=-1;
            }
    for(i=0;i<degree-2;i++)
        for(j=0;j<degree;j++)
            if(a[i][j]!=-1 && a[i][j]==a[i+2][j])

```

```

        a[i+1][j]=-1;
    for(i=0;i<degree;i++)
        for(j=0;j<degree-2;j++)
            if(a[i][j]!=-1 && a[i][j]==a[i][j+2])
                a[i][j+1]=-1;
    for(i=0;i<degree;i++)
        for(j=0;j<degree;j++){
            if(a[i][j]==0 || a[i][j]==1)
                chosen[i*degree+j]=0;
            else chosen[i*degree+j]=1;
            assignment[i*degree+j]=a[i][j];
        }
    int n_tochoose=0;
    if(mode==2){
        for(o=0;o<degree*degree;o++){
            if(!chosen[o]){
                n_tochoose++;
                if(!assignment[o])
                    addrestriction(p,-o-1);
                else
                    addrestriction(p,o+1);
            }
        }
    }
    return n_tochoose;
}

void digHole(int degree,cnf p,int *puzzle,int n){
    conjunct c=p;
    int i,occupation_times,tmp;
    int *occupation_assign=malloc(degree*degree*sizeof(int));
    for(i=0;i<1;i++){
        tmp=c->next->C->next->ele;
        removeClause(p,c);
        cnf p_=addUnitClause(p,-tmp);

        if(DPLL(p_,occupation_assign,degree*degree,3,&occupation_times)==0){
            puzzle[abs(tmp)-1]=-1;
        }
        else
    
```



```

        addrestriction(p,tmp);
        destroyCNF(p_);
    }
}

```

```

cnf PuzzleWithOnlySolution(int degree,int mode,int *assignment,int
*puzzle){
    cnf p=RandomFilled(degree,assignment);
    int i;
    for(i=0;i<degree*degree;i++)
        puzzle[i]=assignment[i];
    int n=complement(degree,mode,p,puzzle);
    digHole(degree,p,puzzle,n);
    return p;
}

```

10. display.c

```

#include "display.h"

int getInt(int max,int min){
    char ch[50];
    int r;
    if(max>=10){
        scanf("%s",ch);
        while( (strlen(ch)!=1|| !(ch[0]>=min+'0'&&ch[0]<= '9')) &&
(strlen(ch)!=2||(ch[0]!='1' || !(ch[1]>='0'&&ch[1]<=max-10+'0')))){
            printf("INVALID INPUT!\nPlease try again:");
            scanf("%s",ch);
        }
        if(strlen(ch)==1) r=ch[0]-'0';
        else r=ch[1]-'0'+10;
    }
    else{
        scanf("%s",ch);
        while( (strlen(ch)!=1|| !(ch[0]>=min+'0'&&ch[0]<=max+'0'))){
            printf("INVALID INPUT!\nPlease try again:");
            scanf("%s",ch);
        }
        r=ch[0]-'0';
    }
    return r;
}

```

```

    }

    int getIntFromFile(FILE *in,int max,int min){
        char ch[50];
        int r;
        if(max>=10){
            fscanf(in,"%s",ch);
            while( (strlen(ch)!=1|| !(ch[0]>=min+'0'&&ch[0]<='9')) &&
                (strlen(ch)!=2||(ch[0]!='1' || !(ch[1]>=0'&&ch[1]<=max-10+'0')))){
                fscanf(in,"%s",ch);
            }
            if(strlen(ch)==1) r=ch[0]-'0';
            else r=ch[1]-'0'+10;
        }
        else{
            fscanf(in,"%s",ch);
            while( (strlen(ch)!=1|| !(ch[0]>=min+'0'&&ch[0]<=max+'0'))){
                fscanf(in,"%s",ch);
            }
            r=ch[0]-'0';
        }
        return r;
    }
}

```

```

void displaySudoku(int *assignment,int degree){
    int i,j;
    printf("  | ");
    for(j=0;j<degree;j++)
        printf("%2d  ",j+1);
    printf("\n");
    printf("---|");
    for(i=0;i<degree;i++){
        printf("----");
        printf("\n");
        for(i=0;i<degree;i++){
            printf("%2d | ",i+1);
            for(j=0;j<degree;j++){
                if(assignment[order(degree,i+1,j+1)-1]==-1)printf("    ");
                else printf("%2d  ",assignment[order(degree,i+1,j+1)-1]);
            }
        }
    }
}

```

```

        printf("\n");
        if(i!=degree-1)printf("    \n");
    }
}

void SolveSudoku(void){
    int degree;
    cnf p;
    printInputMethod();
    if(getInt(2,1)==1){
        printf("\nPlease input the size of your binary puzzle(4 or 6 or 8 or 10
or 12 or 14):");
        char ch[10];
        scanf("%s",ch);
        while( (strlen(ch)!=1||(ch[0]!='4'&&ch[0]!='6'&&ch[0]!='8')) &&
(strlen(ch)!=2||(ch[0]!='1'|(ch[1]!='0'&&ch[1]!='2'&&ch[1]!='4')) )){
            printf("INVALID INPUT!\nPlease input your choice again:");
            scanf("%s",ch);
        }
        if(strlen(ch)==1) degree=ch[0]-'0';
        else degree=ch[1]-'0'+10;
        if(degree==14)
            printf("\nPreparing! Please wait for a little while!\n");
        p=createSudokuCNF(degree);
        int row,column,value;
        printf("\n\nPLEASE INPUT YOUR PUZZLE!\n");
        printf("Please Input your puzzle as the following instructions
says.\nPlease start your input!\n");
        printf("\nInput the row number (0 means the end of your input):");
        while(row=getInt(degree,0)){
            printf("\nInput the column number :");
            column=getInt(degree,1);
            printf("\nInput the value :");
            value=getInt(1,0);
            if(value) addrestriction(p,order(degree,row,column));
            else addrestriction(p,-order(degree,row,column));
            printf("\nInput the row number (0 means the end of your
input):");
        }
    }
}

```

```

else{
    char filename[50];
    FILE *in=NULL;
    printf("Please input your filename:\n");
    do{
        scanf("%s",filename);
        while(access(filename,0)==-1){
            printf("\nSorry, %s does NOT exist!\nPlease input your
filename once again:\n",filename);
            scanf("%s",filename);
        }
        in=fopen(filename,"r");
        if(!in){
            printf("Sorry, %s CANNOT be opened!\nPlease
reassure %s is a CNF file!\n",filename,filename);
            printf("Please input your filename once again after
confirmation:\n");
        }
    }while(!in);
    in=fopen(filename,"r");
    char ch[10];
    fscanf(in,"%s",ch);
    while( (strlen(ch)!=1||(ch[0]!='4'&&ch[0]!='6'&&ch[0]!='8')) &&
(strlen(ch)!=2||(ch[0]!='1'|(ch[1]!='0'&&ch[1]!='2'&&ch[1]!='4')) ) ){
        fscanf(in,"%s",ch);
    }
    if(strlen(ch)==1) degree=ch[0]-'0';
    else degree=ch[1]-'0'+10;
    if(degree==14)
        printf("\nPreparing...\nPlease wait for a little while...\n");
    p=createSudokuCNF(degree);
    int row,column,value;
    while(row=getIntFromFile(in,degree,0)){
        column=getIntFromFile(in,degree,1);
        value=getIntFromFile(in,1,0);
        if(value) addrestriction(p,order(degree,row,column));
        else addrestriction(p,-order(degree,row,column));
    }
}
int *assignment=malloc(degree*degree*sizeof(int));

```



```

int i;
for(i=0;i<degree*degree;i++)
    assignment[i]=-1;
int times=0;
if(degree==12)
    printf("\nIt may take a little while(a few seconds) to solve a Sudoku
with a degree of 12.\nThank you for your patience...\n");
else if(degree==14)
    printf("\nIt may take a little while(a minute or so, sometimes 2
minutes) to solve a Sudoku with a degree of 14.\nThank you for your
patience...\n");
int r;
double start=clock();
r=DPLL(p,assignment,degree*degree,3,&times);
double end=clock();
if(r){
    printf("\n\nCongratulations!\nYour puzzle is solved!\nThe answer is
as follows!\n");
    displaySudoku(assignment,degree);
}
else printf("\nSorry, there is no such an answer that satisfies your
puzzle!\n");
printf("Solving time: %lfms\n",(end-start)/CLOCKS_PER_SEC*1000);
free(assignment);
}

void PlayBinaryPuzzle(void){
    printf("Please input the order of the puzzle you wanna try(4 or 6 or 8 or
10 or 12):");
    int degree,mode;
    char ch[10];
    scanf("%s",ch);
    while( (strlen(ch)!=1||(ch[0]!='4'&&ch[0]!='6'&&ch[0]!='8'))&&(strlen(
ch)!=2||(ch[0]!='1'&&ch[1]!='0'&&ch[1]!='2'))){
        printf("INVALID INPUT!\nPlease input your choice again:");
        scanf("%s",ch);
    }
    if(strlen(ch)==1) degree=ch[0]-'0';
    else degree=ch[1]-'0'+10;
    if(degree<=10){

```

```

        mode=2;
        if(degree==10)printf("\nGenerating a puzzle...\nIt may take a few
seconds.\nThank you for your patience!\n");
    }
    else{
        mode=1;
        printf("\nGenerating a puzzle...\nIt may take 1-2 minutes.\nThank
you for your patience!\n");
    }
    int *assignment=malloc(degree*degree*sizeof(int));
    int *puzzle=malloc(degree*degree*sizeof(int));
    double start=clock();
    cnf p=PuzzleWithOnlySolution(degree,mode,assignment,puzzle);
    double end=clock();
    int *users=malloc(degree*degree*sizeof(int));
    int i,blanks=0,filled=0,flag,r,c,v,confirmation=0;
    for(i=0;i<degree*degree;i++)
        if((users[i]=puzzle[i])== -1) blanks++;
    printf("\nGenerating
time: %lfms",(end-start)/CLOCKS_PER_SEC*1000);
    while(!confirmation){
        printf("\n\n");
        displaySudoku(users,degree);
        printPuzzleOperation();
        flag=getInt(4,1);
        switch(flag){
            case 1:
                printf("\nPlease fill in the blank as the following
instructions says.");
                printf("\nInput the row number :");
                r=getInt(degree,1);
                printf("\nInput the column number :");
                c=getInt(degree,1);
                printf("\nInput the value :");
                v=getInt(1,0);
                if(puzzle[order(degree,r,c)-1]==-1){
                    if(users[order(degree,r,c)-1]==-1) filled++;
                    users[order(degree,r,c)-1]=v;
                }
                else printf("\nSorry, the number you chose is given by the

```

```
puzzle.\nYou CANNOT change it!\n");
    break;
case 2:
    printf("\nPlease fill in the blank as the following
instructions says.");
    printf("\nInput the row number :");
    r=getInt(degree,1);
    printf("\nInput the column number :");
    c=getInt(degree,1);
    if(puzzle[order(degree,r,c-1)]==-1){
        if(users[order(degree,r,c)-1]==-1)printf("This blank has
NOT been filled yet!\n");
        else{
            users[order(degree,r,c)-1]=-1;
            filled--;
        }
    }
    else printf("\nSorry, the number you chose is given by the
puzzle.\nYou CANNOT clear it!\n");
    break;
case 3:
    printf("\nIt is a pity for you to quit.\nWish you more
patience next time!\n");
    break;
case 4:
    for(i=0;i<degree*degree;i++)
        users[i]=puzzle[i];
    filled=0;
    break;
}
if(filled==blanks){
    displaySudoku(users,degree);
    printf("|-----|\n");
    printf("| Confirm to Submit?  |\n");
    printf("|          1.yes          |\n");
    printf("|          0.no           |\n");
    printf("|-----|\n");
    confirmation=getInt(1,0);
}
else if(flag==3)
```

```

        break;
    }
    printf("Your answer is as follows.\n");
    displaySudoku(users,degree);
    if(checkAssignment(p,users))printf("CONGRATULATIONS!  \nYour
answer is exactly what it should be!\n");
    else if(flag==3){
        printf("The CORRECT answer is as follows.\n");
        displaySudoku(assignment,degree);
    }
    else{
        printf("Sorry, maybe there is something wrong with your
answer!\nThe CORRECT answer is as follows.\n");
        displaySudoku(assignment,degree);
    }
}
}

```

```

void printMainMenu(void){
    printf("|-----|\n");
    printf("|      Welcome to My SAT/Sudoku System      |\n");
    printf("|*****Menu For this system*****|\n");
    printf("| 1.SAT      2.Sudoku      0.EXIT SYSTEM |\n");
    printf("|-----|\n");
    printf("\nPlease choose the function you wanna try:");
}

```

```

void printBinaryPuzzleMenu(void){
    printf("*****\n");
    printf("*      Welcome to My Binary Puzzle Game      *\n");
    printf("*-----FUNCTIONS-----*\n");
    printf("*1.Play Binary Puzzle(with ONLY 1 solution)*\n");
    printf("*      2.Solve the Puzzle You Provide      *\n");
    printf("*      0.EXIT BINARY PUZZLE GAME      *\n");
    printf("*****\n");
    printf("\n\nPlease Make Your Choice:");
}

```

```

void printInputMethod(void){
    printf("|-----|\n");
    printf("|      How do you want to INPUT your puzzle?      |\n");
}

```



```

printf("|                                     |\n");
printf("|               1.input MANUALLY               |\n");
printf("|               2.READ from a FILE               |\n");
printf("|-----|\n");
printf("\nPlease choose a mode:");
}

void printMode(void){
printf("|-----|\n");
printf("|   MODE OPTION   |\n");
printf("|               |\n");
printf("|   1.EASY        |\n");
printf("|   2.HARD        |\n");
printf("|-----|\n");
printf("\nPlease choose a mode:");
}

void printPuzzleOperation(void){
printf("\nHINT: ONLY SOLUTION GUARANTEED\n\n");
printf("|-----|\n");
printf("|   Available Operation   |\n");
printf("|               |\n");
printf("| 1.change/fill in a blank |\n");
printf("| 2.clear a blank         |\n");
printf("| 3.quit & display the answer|\n");
printf("| 4.clear all blanks      |\n");
printf("|-----|\n");
printf("(Any other number would be considered to be 4)\n\nchoose a
operation:");
}

void Puzzle(void){
printf("\n\n");
printBinaryPuzzleMenu();
char c[10];
while(scanf("%s",c)){
while(strlen(c)!=1||((c[0]!='0'&&c[0]!='1'&&c[0]!='2'))){
printf("INVALID INPUT!\nPlease input your choice once
again:");
scanf("%s",c);
}
}

```

```

    }
    if(c[0]=='1')PlayBinaryPuzzle();
    else if(c[0]=='2')SolveSudoku();
    else break;
    printf("\n\n");
    printBinaryPuzzleMenu();
}
}

void printSATStrategyMenu(void){
    printf("\n");
    printf("*****\n");
    printf("*-Strategy for DPLL BCP--*\n");
    printf("*          1.MP          *\n");
    printf("*          2.JW          *\n");
    printf("*****\n");
    printf("\nPlease choose a strategy.\nYour choice:");
}

void SAT_Recursive(int k){
    printf("\n\n");
    int STRATEGY;
    if(!k){
        printSATStrategyMenu();
        STRATEGY=getInt(2,1);
    }
    else STRATEGY=3;
    char filename[50];
    FILE *in=NULL;
    printf("Please input your filename:\n");
    do{
        scanf("%s",filename);
        while(access(filename,0)==-1){
            printf("\nSorry, %s does NOT exist!\nPlease input your\nfilename once again:\n",filename);
            scanf("%s",filename);
        }
        in=fopen(filename,"r");
        if(!in){
            printf("Sorry, %s CANNOT be opened!\nPlease reassure %s is

```

```

a CNF file!\n",filename,filename);
        printf("Pease input your filename once again after
confirmation:\n");
    }
}while(!in);
int NOV;
cnf p=createCNF(in,&NOV);
fclose(in);
printf("\nYou want to see the Conjunctive Normal Formula?\n");
printf("0.no 1.yes\nYour choice:");
int *assignment=malloc(NOV*sizeof(int));
char c[10];
scanf("%s",c);
while(strlen(c)!=1||(c[0]!='0'&&c[0]!='1')){
    printf("INVALID INPUT!\nPlease input your choice again:");
    scanf("%s",c);
}
if(c[0]=='1')printCNF(p);
int i;
for(i=0;i<NOV;i++)
    assignment[i]=-1;
int times=0;
double start=clock();
int r=DPLL(p,assignment,NOV,STRATEGY,&times);
double end=clock();
if(r){
    printf("\nThis Conjunctive Normal Formula is SATISFIABLE! \n");
    printf("The Truth Assignments are as followed\n");
    for(i=0;i<NOV;i++)
        if(assignment[i])printf("%d ",i+1);
        else if(!assignment[i])printf("%d ",-(i+1));
        else printf("0 ");
    if(checkAssignment(p,assignment))printf("\nIt has been checked
that the truth assignments above CAN satisfy the CNF!\n");
    else printf("\n\nIt has been checked that the truth assignments above
CANNOT satisfy the CNF!\n");
}
else printf("\nThis Conjunctive Normal Formula is
UNSATISFIABLE\n");
printf("\nDPLL Core Module Running

```

```
Time: %lfms\n", (end-start)/CLOCKS_PER_SEC*1000);
    printf("DPLL was called for %d times!\n", times);
    //output_file(filename, r, assignment, NOV, (end-start)/CLOCKS_PER_SE
C*1000, times);
    //printf("\nThe Running Result is also saved in a file whose path
is %s\n", filename);
}
```

```
void SAT_nonRecursive(void){
    printf("\n\n");
    char filename[50];
    FILE *in=NULL;
    printf("Please input your filename:\n");
    do{
        scanf("%s", filename);
        while(access(filename, 0)==-1){
            printf("\nSorry, %s does NOT exist!\nPlease input your
filename once again:\n", filename);
            scanf("%s", filename);
        }
        in=fopen(filename, "r");
        if(!in){
            printf("Sorry, %s CANNOT be opened!\nPlease reassure %s is
a CNF file!\n", filename, filename);
            printf("Pease input your filename once again after
confirmation:\n");
        }
    }while(!in);
    in=fopen(filename, "r");
    if(!in)printf("Sorry, %s CANNOT be opened!\nDon't worry, let's try
again!\n\n", filename);
    cnf_i p=parseCNF(in);
    fclose(in);
    printf("\nYou want to see the Conjunctive Normal Formula?\n");
    printf("0.no 1.yes\nYour choice:");
    int *assignment=malloc(p->NOV*sizeof(int));
    char c[10];
    scanf("%s", c);
    while(strlen(c)!=1||(c[0]!='0'&& c[0]!='1')){
        printf("INVALID INPUT!\nPlease input your choice again:");
    }
```



```

        scanf("%s",c);
    }
    if(c[0]=='1')outputCNF(p);
    int i;
    for(i=0;i<p->NOV ;i++)
        assignment[i]=0;
    double start=clock();
    int r=DPLL_nonRecursive(p,assignment);
    double end=clock();
    if(r){
        printf("\nThis Conjunctive Normal Formula is SATISFIABLE! \n");
        printf("The Truth Assignments are as followed\n");
        for(i=0;i<p->NOV;i++)
            printf("%d ",assignment[i]);
        if(checkAssignment_2(p,assignment))printf("\nIt has been checked
that the truth assignments above CAN satisfy the CNF!\n");
        else printf("\n\nIt has been checked that the truth assignments above
CANNOT satisfy the CNF!\n");
    }
    else printf("\nThis Conjunctive Normal Formula is
UNSATISFIABLE\n");
    printf("\nDPLL Core Module Running
Time: %lfms\n",(end-start)/CLOCKS_PER_SEC*1000);
    //output_file(filename,r,assignment,NOV,(end-start)/CLOCKS_PER_SE
C*1000,times);
    //printf("\nThe Running Result is also saved in a file whose path
is %s\n",filename);
}

void SAT(void){
    printf("\n");
    printf("#####\n");
    printf("# Welcome to My SAT Solver
#\n");
    printf("#-----Choose an Improvement Aspect-----#\n");
    printf("# 1.Variable Selection Strategy Improvement
#\n");
    printf("# 2.Data Structure && Algorithm Structure Improvement #\n");
    printf("# 3.No Improvement

```

```
#\n");
    printf("#                                0.Exit SAT Solver Module
#\n");
    printf("#####\n");
    printf("\nPlease Make Your choice:");
    int s=getInt(3,0);
    if(s==1)
        SAT_Recursive(0);
    else if(s==2)
        SAT_nonRecursive();
    else if(s==3)
        SAT_Recursive(1);
}
```

