華中科技大學

课 程 实 验 报 告

题目：基于 SAT 的数独游戏求解程序

课程名称	程序设计综合课程设计
专业班级	CS2106
学　　号	U202115514
姓　　名	杨明欣
指导教师	纪俊文
助　　教	许海涛
报告日期	2022 年 10 月 2 日

计算机科学与技术学院

目　　录

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-to)/t]*100%, 其中 t 为未对 DPLL 优化时求解基准算例的执行时间，to 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)

（6）SAT 应用：将数双独游戏 [5] 问题转化为 SAT 问题 [6-8]，并集成到上面的求解器进行数独游戏求解，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 [6-8]。(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. Master thesis, 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 Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243

[5] 360 百科：数独游戏 https://baike.so.com/doc/3390505-3569059.html

Twodoku：https://en.grandgames.net/multisudoku/twodoku

[6] 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.

[7] Ins Lynce and Jol Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper_3485.pdf

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报 (自然科学版), 2015,32(2):187-191

1 引言

1.1 课题背景与意义

1.1.1 SAT 问题

SAT 问题即布尔可满足性问题，是确定是否存在满足给定布尔公式解释的问题。如果对于给定的布尔公式变量可以一致地用 TRUE 或者 FALSE 替换，那么该布尔公式可满足，相反则不满足。

可满足性问题（Satisfiability Problem,SAT）是人工智能领域中的一个重要研究方向，也是自动推理中的核心问题之一。SAT 问题广泛应用于工程技术、软件验证、数学定理证明、组合优化以及逻辑电路等价性验证等多个领域，因此研究 SAT 问题有助于拓展知识面以及今后的实际应用。

1.1.2 数独游戏

“数独 sudoku”是十八世纪瑞士数学家欧拉发明的。2004 年 11 月 12 日,“数独”游戏登上了《泰晤士报》的版面, 很快, 作为该报每日内容的“数独”游戏就风靡了英美! 规则很简单: 有 9×9 共 81 个方格 (即依次有 9 个九宫格, 详细图可参见第 4 节内容) , 在其中填入 1 到 9 的数字, 让每个数字在每一行、每一列及每一个九宫格里都只出现一次。谜题中会预先填入若干数字, 其它方格为空白, 玩家得依谜题中的数字分布状况, 逻辑推敲出剩下的空格里是什么数字。由于规则简单, 在推敲之中完全不必用到数学计算, 只需运用逻辑推理能力, 所以无论男女老幼, 人人都可以玩, 而且容易上手、容易入迷。世界各地有很多数独俱乐部, 还有的国家如法国等专门举行过数独比赛, 其风靡程度可见一斑。

国内外许多学者已对数独的求解算法做了深入研究，例如递归法、回溯候选数法、枚举算法等。

1.2 国内外研究现状

可满足性问题是计算机科学领域和人工智能等领域中的重要研究对象。但是其求解算法的时间开销和空间开销却异常惊人。对于一个含有 n 个命题逻辑变量的合取范式来说，如果使用穷举法来罗列所有真值指派进行求解，虽然在理

论上是可行的，但算法的时间复杂度却是指数级规模，为 $O(2^n)$，计算机如果采用这种方式进行求解将负担不起如此大的开销。搜索空间如此庞大，使得计算机在可等待的时间里不能计算出结果，进而产生组合爆炸问题，所花费的计算时间是人们不能容忍的。S.A.Cook 于 1971 年首次证明了布尔表达式的可满足性问题属于 NP 完全问题。NP 完全问题排在七大数学难题之首，在计算复杂性理论中具有非常重要的地位，一方面因为它有着极大的理论价值并且非常难解，另一方面是一旦被破解以后，在诸多的工程领域里还可以得到广泛的应用。但是在 $P \neq NP$ 的假设条件下，SAT 问题不可能有多项式时间的求解算法。SAT 问题已经成为了一类问题的难度标准，所以称其为 NP 完全问题的种子。

20 世纪 60 年代开始，SAT 问题倍受世人关注，不少学者在这方面做了大量工作，取得了很大突破。

1960 年，Martin Davis 和 Hilary Putnam 提出了首个求解 SAT 问题的完备算法，称为 DP 算法。

1962 年，George Logemann 和 Donald W. Loveland 等人在 DP 算法的基础上提出了 $DPLL$ 算法。

1971 年，Stephen Arthur. Cook 证明了 SAT 问题是 NP 完全问题。

1999 年，João 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。

2003 年，Bart Selman 和 Henry Kautz 在人工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。

2005 年，Niklas Eén 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。

2007 年，Bart Selman 和 Henry Kautz 对当前 SAT 问题的现状做了全面的整理和叙述。

每年此领域都会组织一次可满足性理论和应用方面的国际会议并组织 SAT 竞赛，以寻找高效的 SAT 求解器，并且详细展示出这些求解器各方面的性能，旨在进一步促进 SAT 问题算法的研究。

近年来，由于在计算机理论研究和实际应用中的重要作用，SAT 问题成为国内外研究的热点问题，许多研究人员在这方面做了大量研究，并在算法研究和技术实现上取得了较大的突破，这也间接推动了形式化验证和自动推理等领域的发展。

1.3 课程设计的主要研究工作

本设计基于 $DPLL$ 的算法与程序框架，实现一个完备 SAT 求解器，同时程序的改进，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，设计分支变元处理策略，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。

数独游戏可转化为 SAT 问题，用本系统实现的 SAT 求解器可以快捷地对数独问题转化的 CNF 文件进行求解，再以变元真值数据转化的数独盘格式输出求解答案。本系统具有一定的交互功能，用户可以利用本系统进行数独游戏，系统将自动判断解的正确性，并输出正确答案。

具体研究工程步骤如下：

1. 对于 SAT 求解问题的相关背景、基本原理以及传统的 $DPLL$ 框架进行深入了解，根据相关资料对于项目确定整体方向，设计项目整体流程。
2. 根据 SAT 求解器设计相应的数据结构和算法，进而实现基于 $DPLL$ 框架的 SAT 求解器，并使用相应的算例进行测试:
   (a) 数据结构主要使用十字链表的结构
   (b) CNF 文件的读取、解析和初始化
   (c) 基于 $DPLL$ 框架的 SAT 求解器（核心模块）
   (d) 双数独问题的生成和归约，转化成 SAT 问题并求解
3. 通过查阅文献和不断尝试，优化数据结构和算法，实现求解效率的提高。主要优化方向有：
   (a) 数据结构优化
   (b) 变元分支选取策略优化
   (c) 算法框架优化
4. 设计问题转化策略将数独问题归约为 SAT 问题并求解。
5. 在程序实现的过程中将程序各个结构模块化

2 系统需求分析与总体设计

2.1 系统需求分析

本设计基于 $DPLL$ 的算法与程序框架，实现一个完备 SAT 求解器，通过一定的策略在 $DPLL$ 框架的基础上实现优化。然后将数独问题规约为 SAT 问题，并基于完备 SAT 求解器完成数独游戏求解的设计。

2.2 系统总体设计

系统总体设计分为两个大的模块：基于 $DPLL$ 算法的 SAT 求解器和简易数独游戏，各自模块下面还有一些小的功能，大致介绍如下：

1. 基于 $DPLL$ 算法的 SAT 求解器，需要依次实现：
   (a) CNF 的读取解析，遍历输出，保存建立其对应的数据存储结构；
   (b) $DPLL$ 求解，计算求解时间并显示，将结果保存到同名 $.res$ 文件里；
   (c) 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。
2. 数独游戏，需要依次实现：
   (a) 随机数独游戏的创建，用户可选择挖洞数量进而自行调整难度；
   (b) 通过挖洞法生成数独具有唯一解，转化为 CNF 文件 $DPLL$ 进行求解，再可视化地将结果打印到屏幕上；
   (c) 实现数独界面，具有一定的交互性和可玩性，实现一定程度的可视化交互。

具体流程图如下：

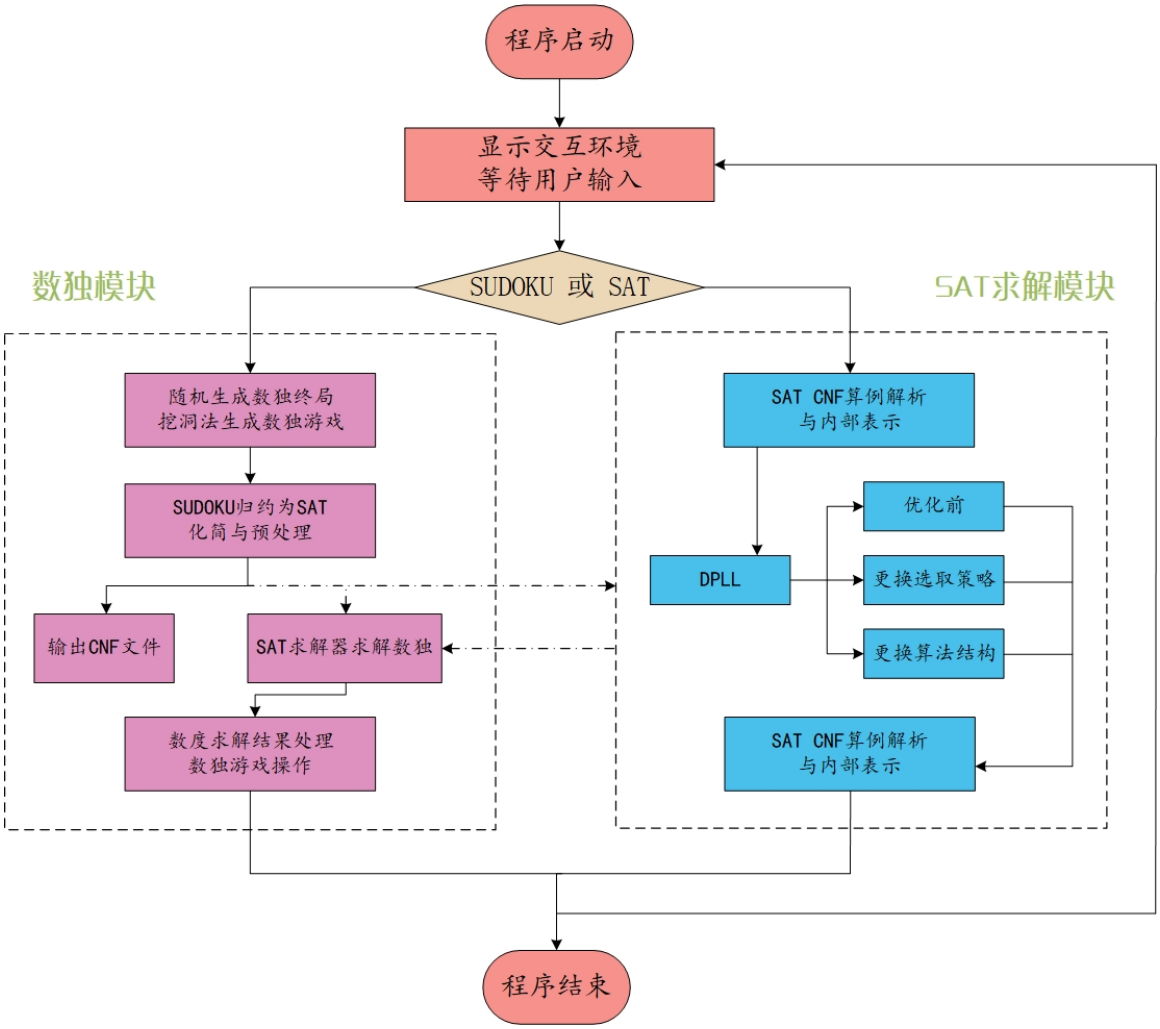
程序启动
显示交互环境
等待用户输入
SUDOKU 或 SAT
数独模块
SAT求解模块
随机生成数独终局
挖洞法生成数独游戏
SUDOKU归约为SAT
化简与预处理
输出CNF文件
SAT求解器求解数独
数度求解结果处理
数独游戏操作
SAT CNF算例解析
与内部表示
DPLL
优化前
更换选取策略
更换算法结构
SAT CNF算例解析
与内部表示
程序结束

图 2-1　系统整体设计图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 相关数据结构

本次实验对于 CNF 文件的解析结果整体采用十字链表的数据结构存储。首先对于文字建立一维链表，然后对于子句建立二维链表存储信息，建立 CNF 结构存储子句和相关信息，实现对于 CNF 文件信息的存储。具体数据结构及包含的数据项如表 3-1：

表 3-1 **数据结构相关定义**

需要处理的数据	所包含的数据项	数据类型
CNF 文件	变元数 (literal_num)	整型变量
	子句数 (clause_num)	整型变量
	首子句 (first_clause)	结构体指针类型变量
子句	文字数 (number)	整型变量
	首文字 (first_literal)	结构体类型变量
	被监视节点 (watch_literal[2])	整型数组
	下一子句 (next_clause)	结构体指针类型变量
文字	文字 (literal)	整型变量
	下一子句文字 (next_literal)	结构体指针类型变量
	下一相同文字 (next_same_literal)	结构体指针类型变量
	对应子句句首 (head)	结构体指针类型变量

对于变元建立结构体数组，存储处理变元所需要的相关信息，包括真值情况、存储处理时对应栈的深度、变量决策时对应的得分、文字总数、正文字总数和负文字总数等，具体的数据结构如表 3-2：

表 3-2 **数据结构相关定义**

需要处理的数据	所包含的数据项	数据类型
文字变元	真值情况 (is_value)	整型变量
	栈的深度 (dep)	整型变量
	变量得分 (score)	整型变量
	文字总数 (num)	整型变量
	正文字数目 (pos)	整型变量
	负文字数目 (nev)	整型变量
	首子句 (first)	结构体类型变量

3.1.2 数据结构关联

建立好的文字、子句和 CNF 结构等数据结构，具有一定的关联性，具体阐述如下：由文字结点链接成子句链表，同时子句结点链接又形成 CNF 链表，同时由相同文字建立好文字链表，形成一定的网状结构，便于搜索和修改。具体图示如图：

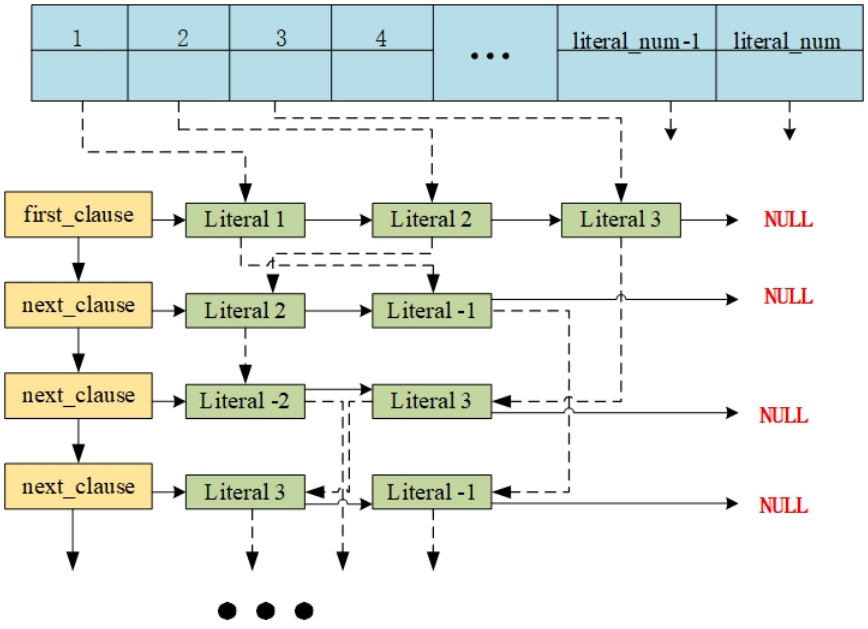

图 3-1 数据结构关联图

3.2 主要算法设计

对于 CNF 公式中为使所有的子句满足条件，需要对于所有文字进行合理的真值赋值，通过真值赋值所得出的搜索空间可以用一颗二叉树来表示。树的结点代表对应的文字变元，树的分支代表不同的赋值情况，从二叉树到叶子结点的一条包含所有变元的路径表示 CNF 公式对应的一组合理的真值赋值。基于 DPLL 的算法是通过对于这棵二叉树从根节点开始进行深度优先搜索寻找合适的通路，以得到问题的可满足解。

Algorithm 1 DPLL 框架

输入： S /*S 代表子句集 */

输出： $True\ or\ False$

```
function DPLL(S)
    while S 中存在单子句 do
        在 S 中选一个单子句 L;
        依据单子句规则，利用 L 化简 S;
        if S = Φ then return True
        else if S 中有空子句 then return False
        end if
    end while
    基于某种变元选取策略选取变元 v;
    if DPLL(S ∪ v) then return True
    end if
    if DPLL(S ∪ ¬v) then return True
    end if
end function
```

在搜索二叉树的过程中如果无法进行单子句传播，需要通过一定的策略进行变量决策；如果发生错误，则变量需要回溯到某一决策层。因此，DPLL 算法框架对应求解过程主要分为以下三个阶段：

1. 变量决策阶段：在搜索过程的每一分支阶段，选择未赋值的一个变量为其赋值为 0 或 1，该变量称为决策变量，决策变量在赋值时所处的二叉树中的高度称为它的决策层。

2. 推理阶段：每一次已选择变量赋值之后，识别该赋值所导致的必要的赋值或者根据已有的变量赋值对子句进行化简，即进行布尔约束传播过程。
3. 回溯阶段：推理过程中发生冲突时，实现算法的回溯，使搜索过程从较深的变量决策层返回至较浅的决策层。冲突即指 BCP 过程中，至少出现了一个子句不可满足的情况。

3.3 变量决策阶段

变量决策策略主要用在回溯算法的搜索过程中，对于未进行赋值的变量进行合理的选择，进而实现分裂搜索，以实现进一步传播。变量决策策略的选择可以影响决策层的数目和决策深度，对计算效率起到决定性作用。变元选择的基本准则是子句越短，子句所包含的变元越难满足，因此应该先进行选择；出现频率越高的变元，选择后满足的子句数更多，因此应该优先选择。

3.3.1 变量选取策略

本实验中在优化前的变量决策方法为在变元中顺次选择没有进行变元赋值的变量。为了对于变量决策阶段进行优化，主要采用以下方法：

1. 最大频率优先：在 CNF 文件中选择变元出现频率最高的变元。
2. 最短子句出现频率最大优先：在 CNF 文件中选择含变元数最少的子句，根据变元统计，选择所选子句中出现频率最高的变元，再根据变元正负文字比例中所占比例更大的文字作为所选文字。
3. 变元加权计算得分：在 CNF 文件中计算变元得分，在选择变元时优先选择得分较高的变元。由于变元出现频率与变元得分正相关，变元所在子句长度与变元得分负相关，因此通过多次尝试，最终确定计算公式如下：

$$score = \sum_{i=1}^{literal\_num} y_i \frac{clause\_num}{length^2} \tag{3.1}$$

其中 clause_num 表示子句数目，literal_num 表示变元数目，y_i 表示决策变量，若变元存在于子句中，$y_i = 1$，反之 $y_i = 0$，length 代表子句所含变元数。

4. 独立变量衰减和策略修改版：首先通过变量加权计算得分。为了减少决策时间，可以在遇到冲突时将冲突的变元放入冲突栈中，在选择变元时，只更

新与冲突相关的变元得分（冲突变元所在子句的变元），进而选取得分高的变元。

说明：因为在遍历 CNF 文件，对变量计算得分的过程中，分数不断累加，有可能导致分数差距过大，因此设定某一阈值，存在变量得分达到或者超过阈值的时候，对于全体变元的得分进行等比例降低，有效避免某一变元分数过高，与其他变元差距较大的情况。

3.3.2 冲突分析

当 BCP 过程中遇到冲突时，通过对于冲突原因进行有效的分析可以对于变量决策有积极影响。本实验在面对冲突时采取的策略时对于冲突相关变量进行双倍计算得分，即对于所有冲突变元所在子句的所有变元重复一次得分的计算，这样可以在变量选择的过程中更容易选到这些变元，让不容易满足的子句所在变元被优先选择，可以更好地避免重复的冲突再次发生。

3.4 推理阶段

SAT 求解器的 80%～90% 时间都用来执行 BCP 过程，所以 BCP 的执行效率很重要。为此本文主要做了以下方面的努力：

3.4.1 数据结构优化

通过对于数据结构的优化，有效减少 BCP 执行的事件。

首先，增加邻接表数据结构。为每一个变元增加邻接表，在单子句传播过程中往往需要遍历整个 CNF 文件，然而这样极大地降低了 BCP 过程所用的时间。通过邻接表的数据结构，能够有效的遍历含有某一变元的所有子句，便于更方便定位每一个变元对应 CNF 文件中变元的位置，能够有效加快 BCP 执行的时间。

其次，对于 CNF 文件中的每一个变元，都增加一个指针指向子句的头指针，以便在更新变元的过程中，更加方便地获取加权公式中的参数值。

最终对应的文字结点对应参数和参照下图：

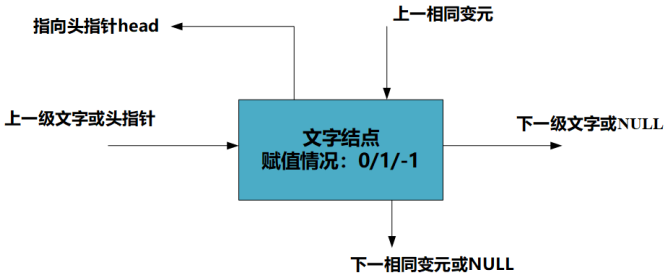

图 3-2　文字结点示意图

3.4.2　传播方式优化

采用双字面量监督算法通过减少单子句传播后回溯时操作次数，加速 BCP 过程。

依据性质一个子句只要有两个不同的字面量都不为 0（即都不判定为假），该子句就不会成为单位子句或冲突。因此，我们主要通过监视两个不同的字面量 watch_literals，来判断每一个子句的具体情况。两个字面量可以被初始化为子句中任意的一对位置不同的自由字面量。

当进行赋值的时候，对于每一个包含赋值变元 p 的子句进行检查，如果该变元并不是被监视的变元，则跳过该子句进行下一子句的检查；如果该变元是被监视的变元，则对于以下情况进行处理：

1. 在子句中如果能够找到与被监视的字面量不同的非零字面量，则更改监视变量 p 为此变量；
2. 如果唯一的非零字面量为另一监视变量 q，则需要分成两种情况进行讨论。如果 q 的字面量为 1，则不需要进行任何操作；如果 q 的字面量为 0，则需要对此变量进行赋值为真的操作，同时进行 BCP 过程。
3. 如果所有的字面量的值均为 0，则该子句发生冲突，需要进行回溯。

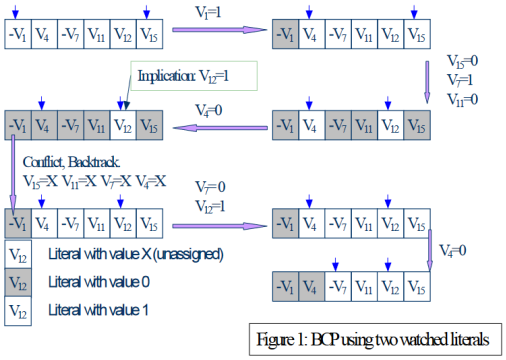

图 3-3　双字面量监督算法结构图

参考前人的算法结构图如上。因为在回溯过程中，被监督变量依然有效，因此只需调整变量的赋值，就可以重新进行变量选择，会节省大量时间。

3.5　回溯阶段

回溯是指在 BCP 过程中发生冲突，因此需要回溯到某一决策层进行重新赋值，进而重新进行 BCP 过程。

3.5.1　智能回溯

在 BCP 过程中，将单子句传播的变元放入回溯栈中，同时标注决策层的深度，便于在回溯的过程中进行同一决策层的回溯。

在回溯过程中，通过将同一决策层的变元出栈，进行变元赋值的取消，即将变元转变成未赋值的状态，实现回溯到上一决策层的目的，可以进一步倒转决策变量的赋值，继续进行 BCP 过程。

3.5.2　周期性重启策略

在 SAT 问题的求解过程中，由于最初的变量决策顺序可能不是最优的，这就会导致搜索陷入某些子空间中而白白耗费时间。重新启动机制，就是清除现在所有变量的赋值状态，重新选择一组决策变量进行赋值，然后进行正常的搜索过程。

周期性重启策略主要是为了对抗 CNF 文件的随机性，由于最初的变量赋值可能会导致 CNF 文件的求解陷入某一困境当中，重新根据当前得分进行赋值变

量选取，有利于打破僵局，更好地找到答案。该周期性重启策略，对于可满足的 CNF 文件求解效率提高具有显著作用。

3.6 数独问题的规约和求解

SAT 问题是理论计算机科学要研究的重要问题，在许多的实际领域都有应用，比如图论、数理逻辑、模型检测、EDA 领域等。这些问题都可以通过一些转换表示为 SAT 问题的形式进而运用 SAT 算法来进行求解。本文主要将其应用在求解数独问题中，通过求解器生成并求解数独。

3.6.1 数独终局的生成

数独的要求是在 9×9 的格子内，每一行和每一列和每一宫内都包含不重复的九个元素。双数独则要求左上数独的最后一宫和右下数独的第一宫完全相同。

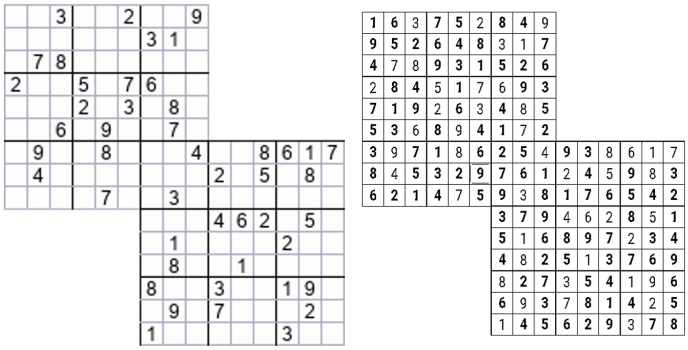

图 3-4　双数独样式及终局图

通过排列法生成终局是一种简单而有效的办法，通过随机数生成符合数独要求的第一行，通过行平移可以发现，在接下来的 8 行中分别对第一行移动 3、6、1、4、7、2、5、8 格时，就能生成一个符合要求的数独。我们还可以发现，对于任何一个数独终局的 1-3 行，我们任意交换这三行的顺序，得到的仍然是一个合法的终局，4-6 行和 7-9 行同理，列也同理。这样我们又可以在刚刚的基础上，按照这种方法扩展出很多种终局。

按照最后一宫将数独对称就可以得到双数独的终局，同时交换数字也可以增加终局生成的可能性。

3.6.2 数独问题规约

变元可按语义编码为 1-9 之间数字构成的四位整数 pijk，i, j, k∈{1,2,⋯,9}，p∈{0,1}, 其中 i 表示单元格的行号，j 表示单元格的列号，k 表示单元格 <i, j> 填入的数字为 k，p 代表数独盘的序号。如 0163 变元表示在左上数独盘第 1 行 6 列填入 3；负文字-1452 表示在右下数独盘第 4 行 5 列不填入 2。这样编码共有 1458 个变元。

约束条件按照 CNF 文件的格式要求写入 CNF 文件中，具体约束条件包括：

1. 每个格内只能含有 1-9 中的一个
2. 每一行 1-9 只出现一次
3. 每一列 1-9 只出现一次
4. 每一宫 1-9 只出现一次
5. 每一格一定要有 1-9 中的一个数字
6. 双数独重叠部分等价关系

最终空数独的 CNF 文件中包含的变元数为 1458，子句数为 20736。

3.6.3 挖洞法生成数独终局

通过挖洞法生成数独的算法效率较高，而且可以通过挖洞的顺序、数量等控制生成数独的难度，因此本实验主要通过挖洞法生成数独，同时通过控制挖洞数量进而控制生成数独的难度。

挖洞法的思想主要是通过挖洞的办法，不断生成含有空格的数独，同时在挖洞的过程中通过 SAT 求解器确保数独解的唯一性和确定性。

本实验主要采用挖洞法的剪枝优化策略，即通过由左至右，由上至下的顺序依次挖洞，对每一个格子只进行一次挖洞判断操作。例如对第一个格子进行挖洞后，通过填入其他数字，调用 SAT 求解器，判断数独问题是否有解，如果该数独填入其他数字后仍然有解，则说明该格子的挖去使得数独的为一街被破坏，因此应该将此格子恢复挖洞前的状态，同时标记此格不能被挖洞；如果没有其他解，则说明此洞可以被挖。

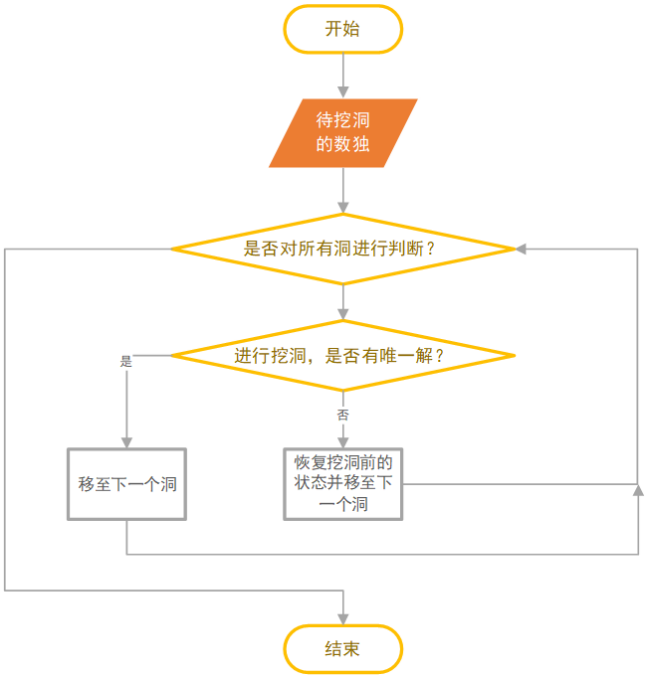

图 3-5　挖洞法算法流程图

3.6.4　数独问题求解

将已知的单子句写入单子句栈中，调用 SAT 求解器进行求解，对对应的语义编码转换为自然顺序编码，公式为：pijn → p*729+(i-1)*81+(j-1)*9+n，因此可以将此进行逆变换，通过分别对 9，81，729 进行取模操作，最终可以得到数独问题的语义编码。

3.6.5　数独问题的交互和可视化

为了提高游戏的可玩性和交互性，本实验设计将数独问题进行可视化交互，主要通过 $EasyX$ 进行图形化绘制。

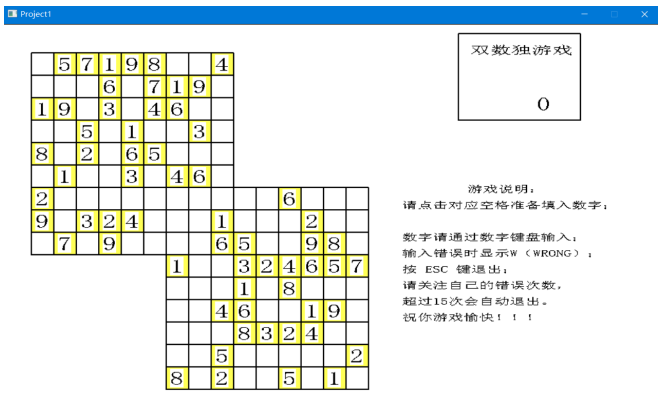

图 3-6　数独界面

数独问题的交互逻辑主要包括进行难度选择、绘制数独盘、进行方格选择、填充数字、判断正误、计算输入错误次数等，绘制好的数独界面如上图。

4 系统实现与测试

4.1 系统实现

4.1.1 软硬件环境

1. 硬件环境
   (a) 处理器：Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
   (b) 机带：16.0 GB (15.8 GB 可用)
   (c) 系统类型：64 位操作系统, 基于 x64 的处理器
2. 软件环境
   (a) Windows 规格：Windows 10 专业版
   (b) 编译器：Clion 2021.3.1 / Visual Studio 2022

4.1.2 各模块主要函数

4.1.3 CNF 文件的读取和解析

对于 CNF 文件的读取需要建立相应的抽象数据结构，包括创建 CNF 结构 CreateCNF、创建子句 CreateClause、销毁 CNF 结构 DestoryCNF、销毁子句 DestroyClause、销毁文字 DestoryLiteral、删除 DeleteLiteral、判断是否有单子句 HasUnitClause 等运算。具体函数及实现如下：

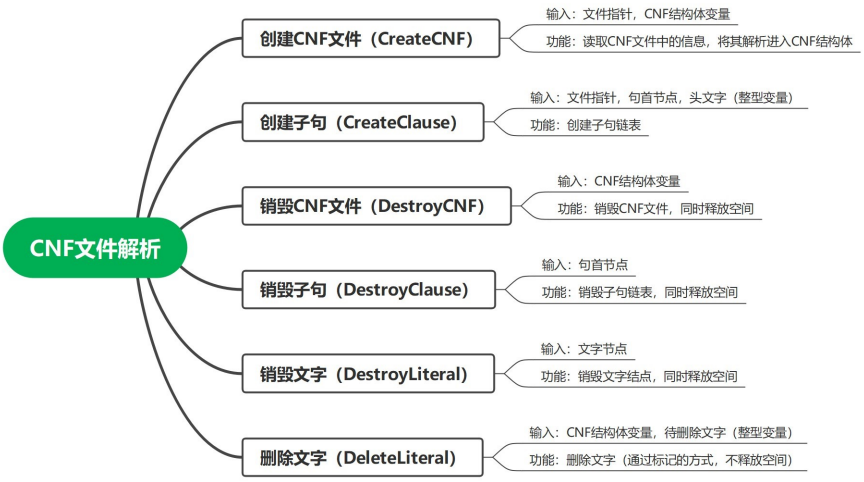

图 4-1　CNF 文件解析函数组成图

4.1.4　DPLL 算法

实现 DPLL 算法需要建立相应的抽象数据结构，同时由于不同的优化策略或者算法，抽象数据结构会有相应的修改，主要包括恢复文字结点 Recover-Literal，更新变元信息 update_storevalue，算法主体 $DPLL$，变元选取策略 Select_Literal_DPLL，变元分支处理 DPLL1_Partition，结点恢复 DPLL1_Recover 等运算。具体函数及实现如下：

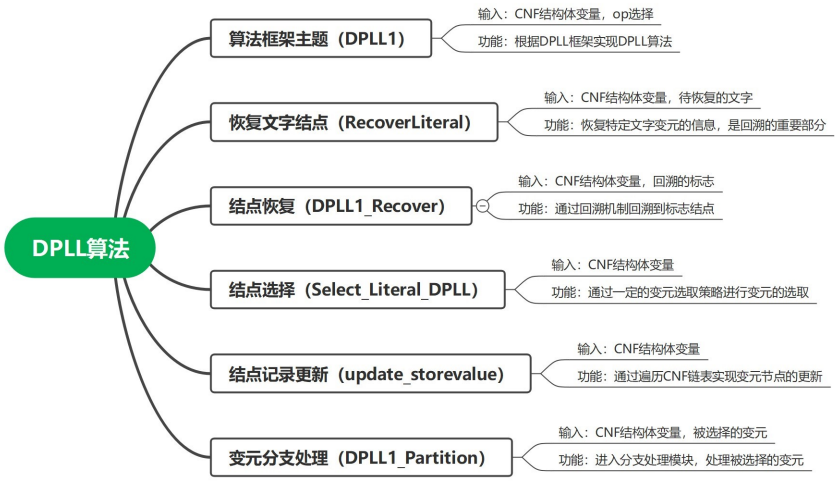

图 4-2　DPLL 算法框架函数组成图

4.1.5 数独问题的生成和规约

数独问题的生成和规约需要建立相应的抽象数据结构，主要包括创造数独终局 CreateSudoku，数独解析为 CNF 文件 CreatePreSudokuFile，通过挖洞法生成数独 Dig_Hole_Easy，求解挖洞数独并记录 sudokusat，调用 SAT 求解器求解数独 DPLL2_SUDOKU 等运算。具体函数及实现如下：

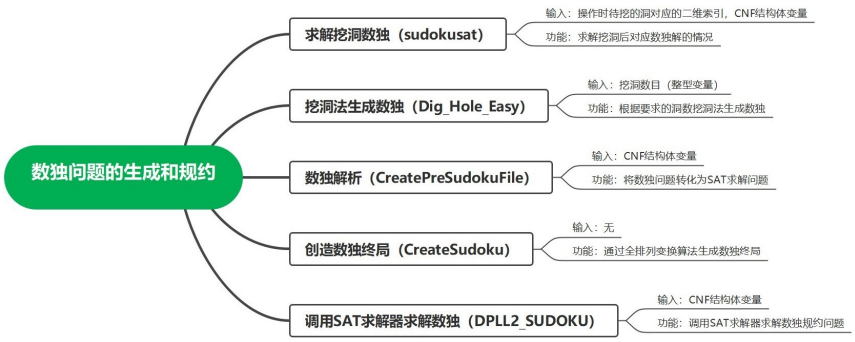

图 4-3　数独生成和规约函数组成图

4.1.6 程序输入输出

程序的输入输出需要建立相应的抽象数据结构，主要包括存储 CNF 求解文件 store_document，遍历 CNF 文件并输出 traveser_cnf，验证 CNF 解的正确性 prove_cnf，展示 CNF 解 show_cnf，输出数独终局 SudokuFinalPrint，输出数独求解时间 print_time 等运算。具体函数及实现如下：

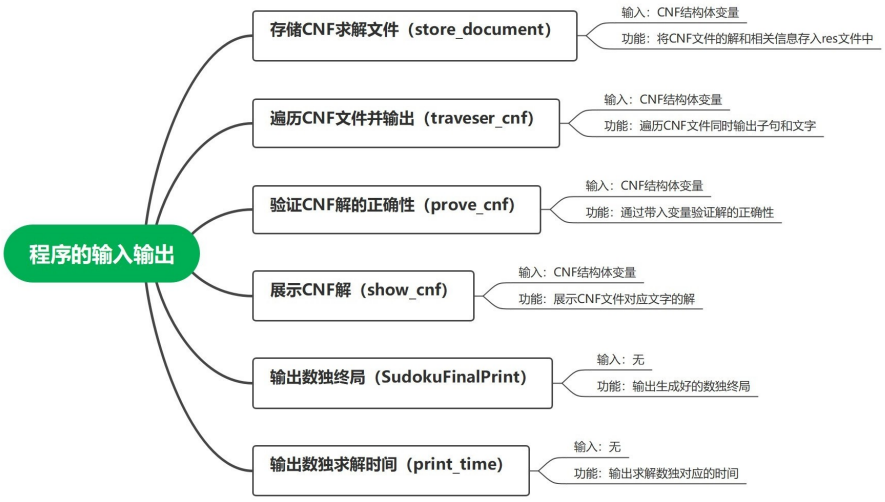

图 4-4　程序输入输出函数组成图

4.2　系统测试

4.2.1　交互页面展示

主要包括 *SAT* 求解过程中的一系列框型交互页面，以及数独问题的交互可视化页面。

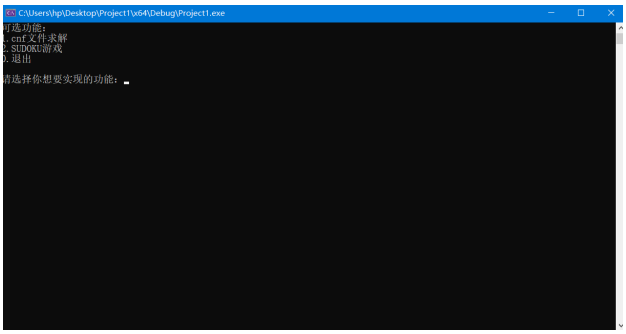

图 4-5　初始交互界面

刚开始运行 exe 文件可以看到如上初始交互界面，可以进行基本功能的选择，也可以选择退出。

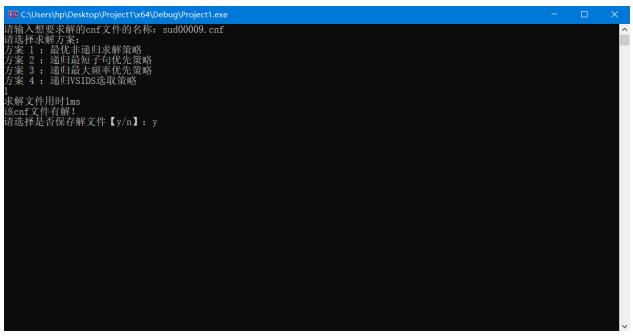

图 4-6　CNF 文件求解界面

上图为 CNF 求解界面，输入 CNF 文件名称后，选取方案策略进行求解，最后保存解文件可选。

图 4-7　数独游戏首页

上图为数独游戏的首页。

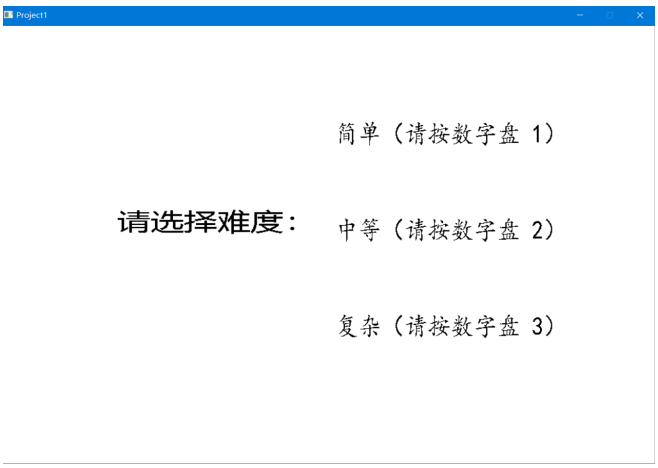

图 4-8　难度选择界面

上图为图形界面的难度选择界面，可以进行三个难度的选择。具体游戏界面已经在前文展示过，因此在此不再进行展示。

4.2.2　CNF 文件的读取和解析以及程序输入输出测试

首先简述一下过程：通过首页面选择 CNF 文件的读取和解析功能即可进入 CNF 文件读取解析模块，输入 CNF 文件名称后，可以通过选择求解方案对 CNF 文件进行求解，输出求解时间和求解判断，同时提供选项是否保存解文件，之后可以提供辅助功能进行 CNF 文件和解文件信息的查看。

该模块的测试主要为以下四个部分：

1. CNF 文件能否成功读取并解析
2. CNF 文件能否正确求解
3. 能否得出解的结构
4. 能否保存解的信息

测试算例 1：sud00009.cnf

首先选择读取 CNF 文件，然后通过方案 1 的 SAT 求解器对 CNF 文件进行求解并保存解文件，求解 CNF 文件的视图如下：

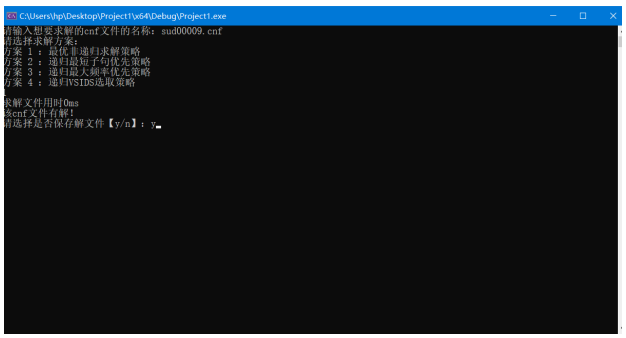

图 4-9　求解 CNF 文件的视图

然后通过一系列可以查看的信息对于 CNF 文件的读取和解析进行测试。

通过遍历 CNF 文件对 CNF 文件能否成功读取并解析进行测试，测试结果如下：

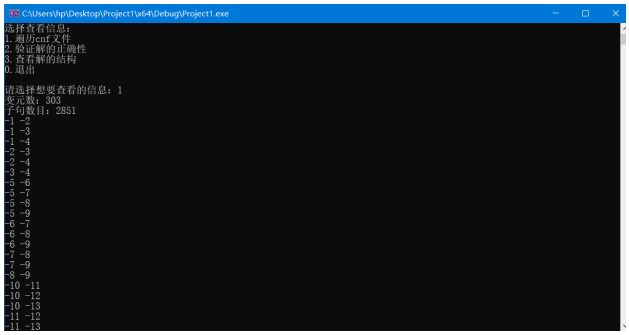

图 4-10　遍历 CNF 文件视图

通过验证解的正确性对 CNF 文件能否正确求解进行测试，测试结果如下：

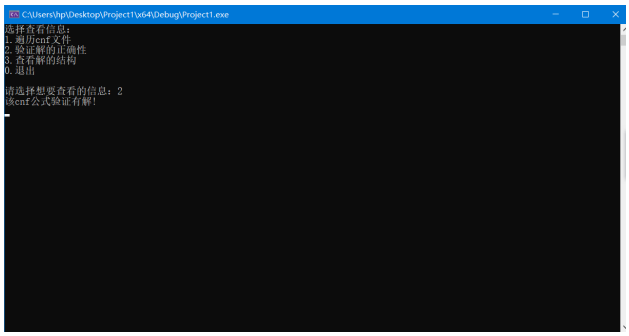

图 4-11　验证解的正确性视图

通过查看解的结构对 CNF 文件能否得出解的结构进行测试，测试结果如下：

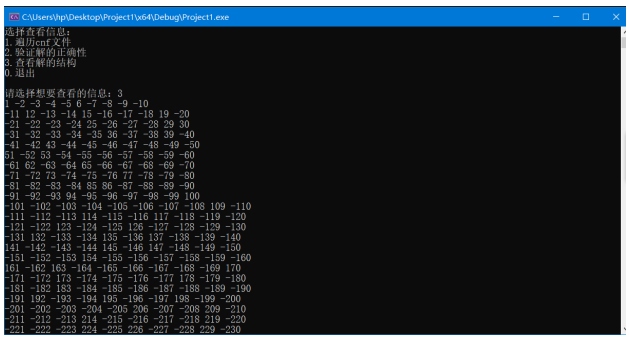

图 4-12　查看解的结构视图

通过查看 RES 文件对 CNF 文件能否保存解的信息进行测试，测试结果如下：

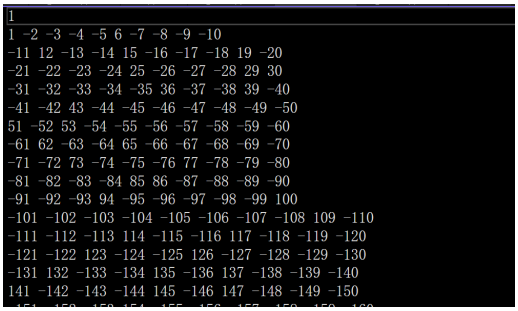

图 4-13　查看 RES 文件视图

测试算例 2：ais10.cnf

首先选择读取 CNF 文件，然后通过方案 1 的 SAT 求解器对 CNF 文件进行求解并保存解文件，求解 CNF 文件的视图如下：

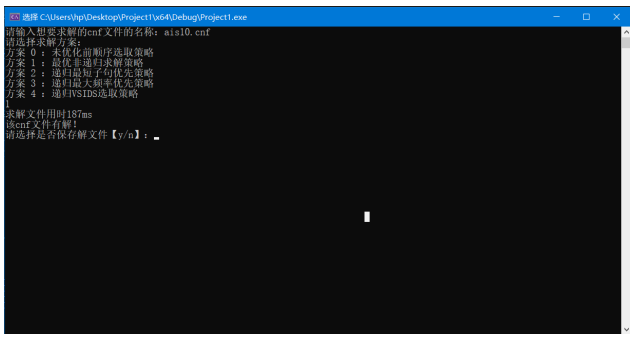

图 4-14　求解 CNF 文件的视图

然后通过一系列可以查看的信息对于 CNF 文件的读取和解析进行测试。

通过遍历 CNF 文件对 CNF 文件能否成功读取并解析进行测试，测试结果如下：

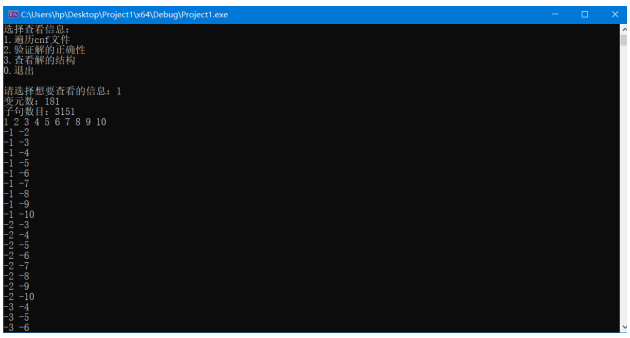

图 4-15　遍历 CNF 文件视图

通过验证解的正确性对 CNF 文件能否正确求解进行测试，测试结果如下：

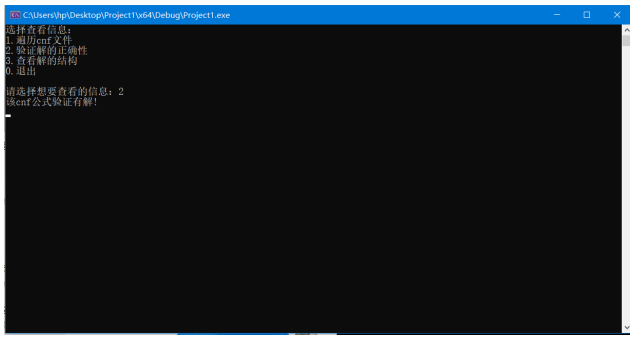

图 4-16　验证解的正确性视图

通过查看解的结构对 CNF 文件能否得出解的结构进行测试，测试结果如下：

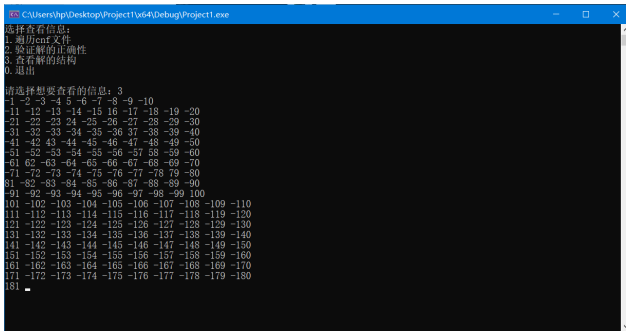

图 4-17　查看解的结构视图

通过查看 RES 文件对 CNF 文件能否保存解的信息进行测试，测试结果如下：

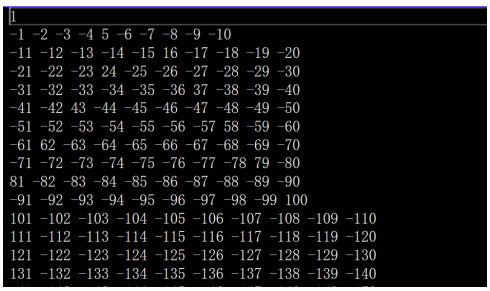

图 4-18　查看 RES 文件视图

4.2.3　DPLL 算法测试

说明：为了测试方便，在测试求解时间时，将这一模块抽取出来在 Clion 编译器中进行测试。

输入 CNF 文件后，通过不同的方案对 CNF 文件进行求解，同时比较求解的时间和优化率。

该模块的测试主要为获取求解时间进而计算优化率。计算优化率的公式如下：

$$\frac{t-t_0}{t}\times 100\% \tag{4.1}$$

测试算例：sud00009.cnf

对应五种策略的求解结果如下：

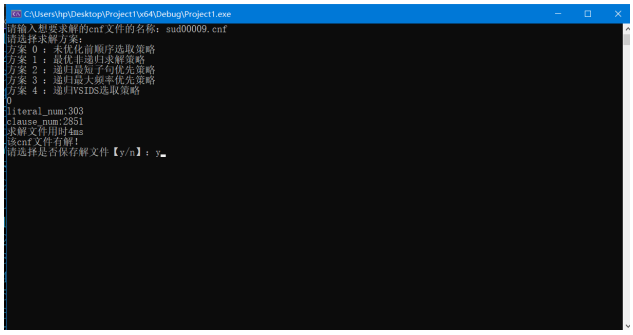

图 4-19　第一种策略求解视图

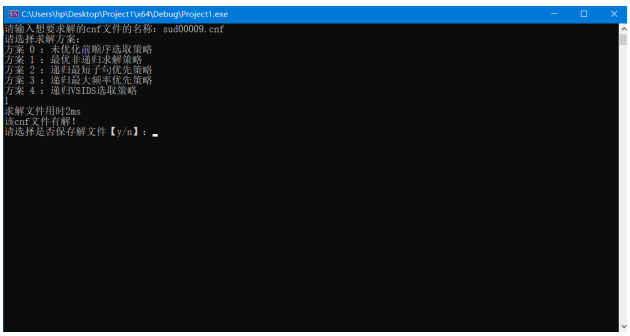

图 4-20　第二种策略求解视图

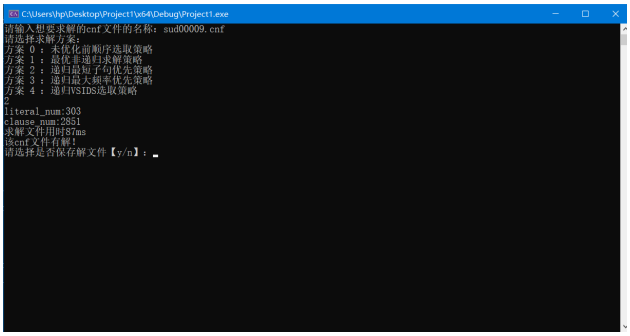

图 4-21　第三种策略求解视图

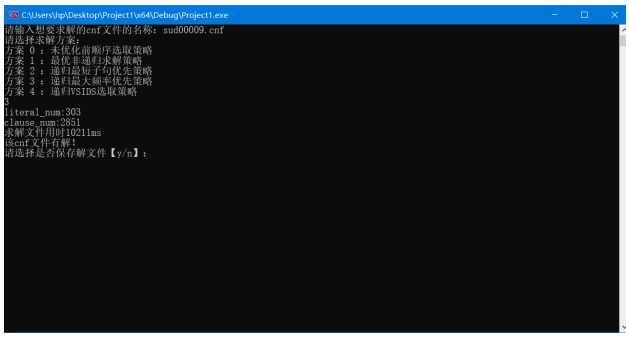

图 4-22　第四种策略求解视图

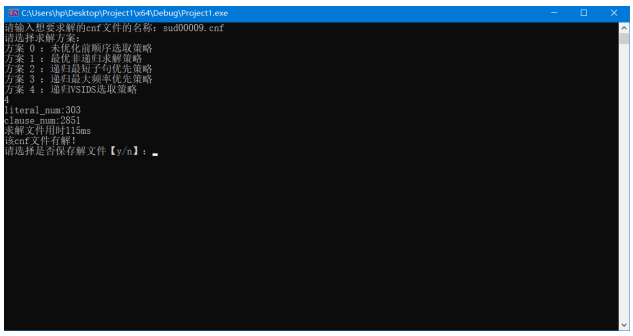

图 4-23　第五种策略求解视图

CNF 算例测试总结表格

不少于 18 个 SAT 算例，其中可满足的算例不少于 15 个，不满足的算例不少于 3 个，大中小算例各占三分之一。对算例规模的要求为：小型算例变元数为 100 个左右；中型算例变元数介于 200-500 个；大型算例变元数 600 个以上。

小型算例部分

表 4-1　算例性质

算例名称	变元数	子句数	子句数/变元数
problem2-50.cnf	50	80	1.6
problem3-100.cnf	100	340	3.4
problem11-100.cnf	100	600	6
problem8-50.cnf	50	300	6
7cnf20_90000_90000_7.shuffled-20.cnf	20	1532	76.6

表 4-2　算例求解结果

算例名称	优化前时间/ms	是否满足	优化后时间/ms（用 | 分隔）	最佳优化率/%
problem2-50.cnf	64	满足	1|0|7|0	100
problem3-100.cnf	754	满足	39|7|260|476	99.1
problem11-100.cnf	27	满足	1|5|14|6	96.3
problem8-50.cnf	1	满足	0|1|1|0	100
7cnf20_90000_90000_7.shuffled-20.cnf	49	满足	70|103|37|23	53.1

中型算例部分

表 4-3　算例性质

算例名称	变元数	子句数	子句数/变元数
bart17.shuffled-231.cnf	231	1166	5.0
problem12-200.cnf	200	1200	6
sud00021.cnf	308	2911	9.5
m-mod2c-rand3bip-sat-220-3.shuffled-as.sat05-2490-311.cnf	311	2192	7.0
sud00861.cnf	297	2721	9.2

表 4-4　算例求解结果

算例名称	优化前时间/ms	是否满足	优化后时间/ms（用 | 分隔）	最佳优化率/%
bart17.shuffled-231.cnf	>120000	满足	8|4|3|50	100
problem12-200.cnf	10292	满足	515|1|6854|8182	100
sud00021.cnf	52	满足	1|210|2784|251	99.9
m-mod2c-rand3bip-sat-220-3.shuffled-as.sat05-2490-311.cnf	>120000	满足	7983	100
sud00861.cnf	19	满足	2|25|2580|115	89.5

大型算例部分

表 4-5　算例性质

算例名称	变元数	子句数	子句数/变元数
ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf	1584	16587	10.5
eh-dp04s04.shuffled-1075.cnf	1075	3152	2.9
e-par32-3.shuffled-3176.cnf	3176	10297	3.2

表 4-6　算例求解结果

算例名称	优化前时间/ms	是否满足	优化后时间/ms（用 | 分隔）	最佳优化率/%
ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf	>120000	满足	659	100
eh-dp04s04.shuffled-1075.cnf	>120000	满足	21	100
e-par32-3.shuffled-3176.cnf	>120000	满足	1855	100

说明：由于处理器的限制，仅能跑出三个大型算例。

不满足算例部分

表 4-7　算例性质

算例名称	变元数	子句数	子句数/变元数
php-010-008.shuffled-as.sat05-1171.cnf	80	370	4.6
u-dp04u03.shuffled-825.cnf	825	2411	2.9
u-problem7-50.cnf	50	100	2
u-5cnf_3900_3900_060.shuffled-60.cnf	60	936	15.6
u-5cnf_3500_3500_30f1.shuffled-30.cnf	30	420	14

表 4-8　算例求解结果

算例名称	优化前时间/ms	是否满足	优化后时间/ms（用 | 分隔）	最佳优化率/%
php-010-008.shuffled-as.sat05-1171.cnf	2666	不满足	856|1613|1205|2522	71.8
u-dp04u03.shuffled-825.cnf	465475	不满足	70|14553|481	100
u-problem7-50.cnf	44	不满足	1|2|45|71	97.7
u-5cnf_3900_3900_060.shuffled-60.cnf	102739	不满足	19431|10602|42769	89.7
u-5cnf_3500_3500_30f1.shuffled-30.cnf	25	不满足	21|18|31|28	19.4

基准算例部分

表 4-9　算例性质

算例名称	变元数	子句数	子句数/变元数
ais10.cnf	181	3151	17.4
sud00009.cnf	303	2851	9.4

表 4-10　算例求解结果

算例名称	优化前时间/ms	是否满足	优化后时间/ms（用 | 分隔）	最佳优化率/%
ais10.cnf	1430	满足	20|1831|3090|2068	98.6
sud00009.cnf	10	满足	0|40|120|60	100

4.2.4　数独游戏求解及简易游玩模块测试

首先简介一下数独部分：基于挖洞法随机生成数独游戏盘进行交互游玩，并且有选择难度的棋盘生成，游玩过后可以查看答案.

该模块的测试主要为以下四个部分：

1. 生成双数独游戏
2. 进行数字填入，包括正确填入和错误填入
3. 获取数独求解时间

生成双数独游戏

首先进行难度的选择。

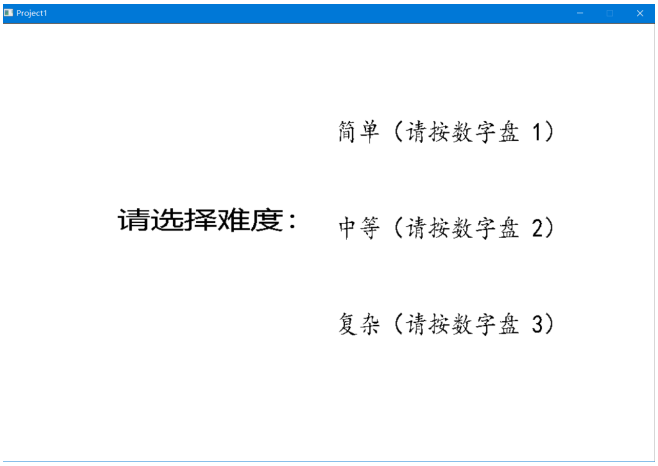

图 4-24　难度选择页面

通过正确输入难度序号可以成功进行选择，进入数独游戏界面。

进行数字填入，包括正确填入和错误填入

正确输入时可以看到，数字可以填入并且正确显示。

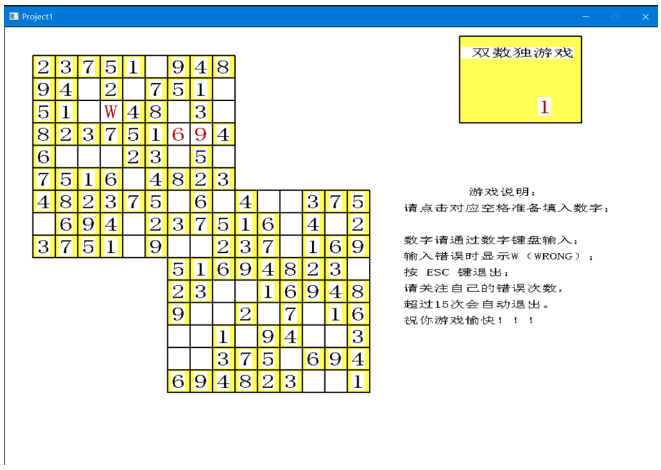

图 4-25　正确选择情形页面

错误输入时可以看到，数字错误输入时会显示 W，可以重新填入，但是错误次数累加。

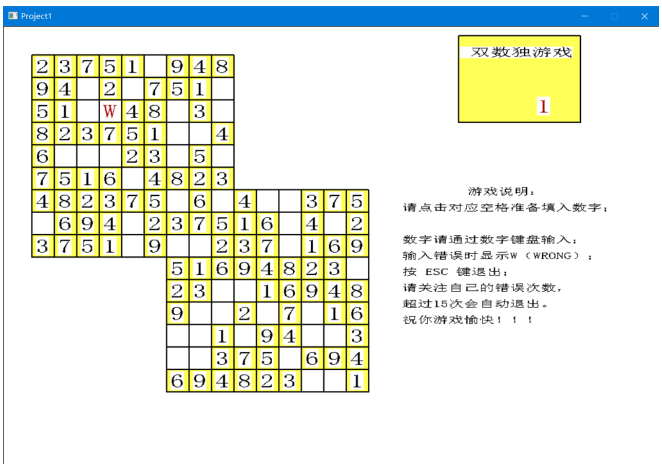

图 4-26　错误选择情形页面

获取数独求解时间

生成数独的求解时间如下：

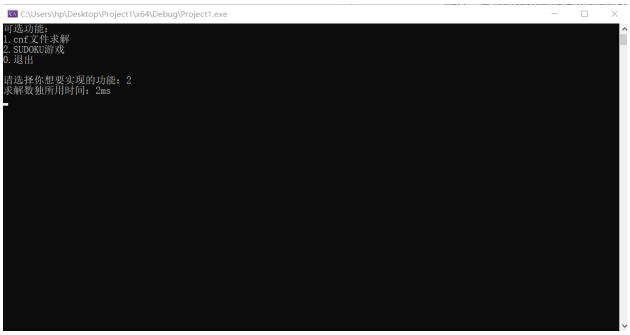

图 4-27　数独求解时间页面

由于 SAT 求解器对于数独规约后的 CNF 文件良好的求解能力，求解时间在 10ms 以内。

5 总结与展望

5.1 全文总结

对自己的工作做个总结，主要工作如下：

1. 实现了一个基于 DPLL 算法的 SAT 求解器；
2. 通识多篇论文，了解 SAT 求解器的发展历程；
3. 分别在变量决策阶段、单子句传播阶段和回溯阶段进行优化；
4. 实现了一定程度上论文方法的复现，包括重启策略，冲突分析等；
5. 完成了数独游戏的规约和求解，并通过 EasyX 实现了游戏的可交互性；
6. 设计了测试方案并完成了多个算例的测试；
7. 完善了各种部分的衔接，构建一个简易系统。

5.2 工作展望

在今后的研究中，围绕着如下几个方面开展工作：

（1）由于设备和算法原因，设计的 SAT 求解器较为低级。测试过程中，当变元数量较大时，对于有些大型算例可以求解但时间效率往往有点低下。在今后的工作中，希望可以得到更多的编程和项目经验；

（2）在 DPLL 算法实现中，最主要的优化方向是变元选取策略。因为变元选取策略没有最优化的结果，因此再下来还应该不断去寻找更好的选取策略。

（3）对于数据结构的选择有待完善，本次使用的链表结构存储各变元并进行相关操作。数据结构的选择的简单导致后续设计相应操作时较为复杂。接下来考虑使用懒惰数据结构进行更好地存储和操作。

（4）在数独游戏模块，主要采用了 EasyX 进行交互界面的设计，但仍然有很多的不足，接下来考虑使用 Qt 进行界面的设计。

6 体会

在本次课程设计的过程中我阅读了很多篇文献，提高了自己的编程能力，也认识到了自己的很多不足。以下是我按照完成整个课设的时间顺序写的一些体会：

1、由于是第一次完成项目，因此在最开始的时候会有些对自己的不自信，也查找了很多资料，阅读理解很多次任务书才能较为清晰的感受到大致的方向。然后就开始一个步骤一个步骤去完成，当不断上手过后，才开始有自信去不断重构，去完成这个项目。

2. 在实现基于 DPLL 的 SAT 求解器的过程中，由于自身代码能力和经验的限制，因此在实现 SAT 求解器的过程中遇到了很多 bug，开始时对于 bug 还是感到很无力的，但是在不断修改的过程中，我也开始不断提高自己修复 bug 的能力，最终面对错误也能够以正确的心态进行调整。

3. 在数独创建中，学习了挖洞法生成数独的算法，在学习过程中，开始了解如何真正学会一个算法，从理解到实现，最终应用到课程设计的实验中，我觉得自己对于算法的学习能力有了很大的成长。我相信在进一步的学习过程中，我一定能更快地掌握算法，同时也能更好地进行实现。

4. 最后就是整合为项目的问题，这个问题确实困扰了我很久。因为一开始使用的是 Clion 编译器，Clion 对于项目的组织结构要求很高，需要极大的时间配置 CMakeLists.txt，因此我在网上不断地进行学习；后来因为 EasyX 只兼容 Visual Studio，因此使用其作为最终的编译器来完成全部的项目。

总而言之，在这次课程设计中收获到了很多知识，在这个过程中，不仅要感谢纪老师的关心和解答，也要感谢许助教的帮助，同时也要感谢张晋铭同学对我思路的启发以及看文献的习惯。我相信我会在以后的学习中更加成长。

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] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm.[J]. J. Autom. Reasoning,2007,39(2).

[3] 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.

[4] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, ”Chaff: engineering an efficient SAT solver,” Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), 2001, pp. 530-535, doi: 10.1145/378239.379017.

[5] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

[6] 薛源海, 蒋彪彬, 李永卓, 闫桂峰, 孙华飞. 基于“挖洞”思想的数独游戏生成算法 [J]. 数学的实践与认识,2009,39(21):1-7.

附录

文件名：define.h

功能：项目头文件

```
//
// Created by hp on 2022-09-04.
//

#ifndef SUMIT_DEFINE_H
#define SUMIT_DEFINE_H

//引入头文件
#include <graphics.h>              // 引用图形库头文件
#include <conio.h>
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<list>
#include<cstring>
#include<sstream>
#include<queue>
#include<vector>
#include<map>
#include<fstream>
#include<set>
#include<bitset>
#include<cmath>
#include<unordered_map>
#include<random>
#include<ctime>
#include<sys/utime.h>

//有效定义
#define TRUE 1
#define FALSE 0
```

```
32 #define OK 1
33 #define ERROR 0
34 #define INFEASTABLE -1
35 #define INCREASEMENT 100
36 #define MAXN 0x7FFFFF
37 #define _CRT_SECURE_NO_WARNINGS
38
39 static struct timeval st;
40 static struct timeval ed;
41 static double time_total;
42
43
44 //结构体相互引用交叉定义
45 struct Clause_Node;
46 struct Literal_Node;
47 typedef struct Clause_Node Clause_Node, * Clause_List;
48 typedef struct Literal_Node Literal_Node, * Literal_List;
49 typedef int status;
50
51 /*定义子句链式结构节点*/
52 struct Clause_Node {
53     int number;//定义子句中的文字数，可判定是否为单子句
54     int tag;//定义字句是否被处理
55     Literal_List first_literal;//定义文字节点，指向第一文字
56     int watch_literal[2];//定义被监视节点
57     struct Clause_Node* next_clause;//定义字句节点，指向下一子句
58 };
59
60 /*定义文字链式结构节点*/
61 struct Literal_Node {
62     int literal;//定义整型文字
63     int tag;//定义文字是否被处理
64     struct Literal_Node* next_literal;//定义下一文字节点
65     struct Literal_Node* next_same_literal;//定义下一相同节点
66     struct Clause_Node* head;//定义该文字对应子句句首
```

```
67  };
68
69  /*定义CNF链式结构结点，存储CNF信息*/
70  typedef struct Conjunctive_Normal_Form {
71      int literal_num;//定义文字数目
72      int clause_num;//定义子句数目
73      Clause_List first_clause;//定义子句头节点，指向CNF的第一个子
        句
74  }Conjunctive_Normal_Form, * Conjunctive_Normal_Form_List;
75
76  typedef struct ArgueValue {
77      int tag;//是否被判断真假
78      int is_value;//判断真值情况
79      int dep;//栈的深度
80      int score;//该变量的得分
81      int num;//文字总数
82      int pos;//正文字数目
83      int nev;//负文字数目
84      Literal_List first;//首子句指针
85  }ArgueValue;
86
87  /*CNF文件读取处理函数*/
88  status DestroyClause(Clause_List& sentence);//定义DestroyClause函
    数，销毁子句结点
89  status DestroyLiteral(Literal_List& word);//定义DestroyLiteral函
    数，销毁文字结点
90  status DestroyCNF(Conjunctive_Normal_Form_List& cnf);//定义
    DestroyCNF函数，销毁CNF文件
91
92  //func_DPLL1所用函数
93  status CreateCNF(FILE* fp, Conjunctive_Normal_Form_List& cnf);//
    定义CreateCNF函数，创建CNF链表结果
94  status CreateClause(FILE* fp, Clause_List& sentence, int first);
    //定义CreateClause函数，创建子句链表
95  status DeleteLiteral(Conjunctive_Normal_Form_List& cnf, int
```

```
    literal);//定义DeleteLiteral函数，删除cnf结构中所有的literal文字
96  status RecoverLiteral(Conjunctive_Normal_Form_List& cnf, int
    literal);//定义RecoverLiteral函数，回复特定literal文字
97  status update_storevalue(Conjunctive_Normal_Form_List& cnf);//定
    义update_storevalue函数，更新存储文字
98  status update_storevalue2(Conjunctive_Normal_Form_List& cnf);//定
    义update_storevalue函数，更新存储文字
99  status DPLL1(Conjunctive_Normal_Form_List cnf, int op);//定义
    DPLL1函数，作为处理cnf文件的第一个DPLL文件
100 int Select_Literal_DPLL0(Conjunctive_Normal_Form_List& cnf);//定
    义Select_Literal函数，变量决策策略
101 int Select_Literal_DPLL1(Conjunctive_Normal_Form_List& cnf);//定
    义Select_Literal函数，变量决策策略
102 int Select_Literal_DPLL2(Conjunctive_Normal_Form_List& cnf);//定
    义Select_Literal函数，变量决策策略
103 int Select_Literal_DPLL3(Conjunctive_Normal_Form_List& cnf);//定
    义Select_Literal函数，变量决策策略
104 status DPLL1_Partition(Conjunctive_Normal_Form_List& cnf, int
    literal, int op);//定义DPLL1_Partition函数，变量分裂规则
105 status DPLL1_Recover(Conjunctive_Normal_Form_List& cnf, int
    literal);//定义DPLL1_Recover函数，变量回溯规则
106 Clause_List HasUnitClause(Conjunctive_Normal_Form_List& cnf);//定
    义HasUnitClause函数，评估CNF是否含有单子句
107 status print(Conjunctive_Normal_Form_List& cnf);//定义print函数，
    输出真值表，用于验证和调试
108
109
110 //func_DPLL2所用函数
111 status CreateCNF2(FILE* fp, Conjunctive_Normal_Form_List& cnf);//
    定义CreateCNF函数，创建CNF链表结果
112 status CreateClause2(FILE* fp, Clause_List& sentence, int first);
    //定义CreateClause函数，创建子句链表
113 status DestroyClause2(Clause_List& sentence);//定义DestroyClause
    函数，销毁子句结点
```

```
114 status DestroyLiteral2(Literal_List& word);//定义DestroyLiteral函数，销毁文字结点
115 status DestroyCNF2(Conjunctive_Normal_Form_List& cnf);//定义DestroyCNF函数，销毁CNF文件
116 status print2(Conjunctive_Normal_Form_List& cnf);//定义print函数，输出真值表，用于验证和调试
117 status DPLL2(Conjunctive_Normal_Form_List cnf);//定义DPLL1函数，作为处理cnf文件的第一个DPLL文件
118 status check_print2(Conjunctive_Normal_Form_List& cnf);//定义check_print2函数，进行检查
119
120 status CreateSudoku();//定义CreateSudoku函数，创建数独终盘
121 status Dig_Hole_Easy(int dig);//定义Dig_Hole_Easy函数，挖洞法生成数独
122 status sudokusat(int m, int n, Conjunctive_Normal_Form_List cnf);//定义sudokusat函数，求解数独问题
123 status SudokuFinalPrint(void);//定义SudokuFinalPrint函数，输出数独游戏
124 status DPLL2_SUDOKU(Conjunctive_Normal_Form_List cnf, int t[], int top);//定义DPLL2_SUDOKU函数，调用SAT求解器
125 int print_time(Conjunctive_Normal_Form_List cnf);//定义print_time函数，输出求解时间
126
127 void solve(int op);//定义solve函数，输出交互界面
128 void solve_cnf();//定义solve_cnf函数，输出cnf文件求解交互界面
129 void solve_sudoku();//定义solve_sudoku函数，输出数独求解交互界面
130 void store_document(Conjunctive_Normal_Form_List& cnf, char* filename, int d, int time0);//定义store_document函数，存储文件
131 void traveser_cnf(Conjunctive_Normal_Form_List cnf);//定义traveser_cnf函数，进行CNF文件遍历
132 void prove_cnf(Conjunctive_Normal_Form_List cnf);//定义prove_cnf函数，进行验证
133 void show_cnf(Conjunctive_Normal_Form_List cnf);//定义show_cnf函数，输出cnf文件求解结果
134 #endif //SUMIT_DEFINE_H
```

文件名：func_DPLL.cpp

功能：DPLL 框架 1 及多种变量选取策略的实现

```
1
2  #include "define.h"
3
4  /*定义有效的全局变量*/
5  ArgueValue* ValueList;//定义变元真值表
6  int backtracking_stack_DPLL1[MAXN], top1;//定义回溯栈
7  int conflict_stack[MAXN], ctop;
8  int ans1;
9
10 //定义DPLL1函数，作为处理cnf文件的第一个DPLL文件
11 status DPLL1(Conjunctive_Normal_Form_List cnf, int op) {
12     ans1 = 0;
13     top1 = 0;
14     ctop = 0;
15     Clause_List p;
16     Literal_List q;
17     while ((p = HasUnitClause(cnf)) != NULL) {
18         q = p->first_literal;
19         while (q->tag == 1) q = q->next_literal;
20         ValueList[abs(q->literal)].tag = 1;
21         if (q->literal > 0)
22             ValueList[abs(q->literal)].is_value = 1;
23         else
24             ValueList[abs(q->literal)].is_value = 0;
25         backtracking_stack_DPLL1[top1++] = q->literal;
26         if (!DeleteLiteral(cnf, q->literal))
27             return FALSE;
28         if (ans1 == cnf->clause_num) return TRUE;
29     }
30     int part_literal;
31     if (op == 0)
32         part_literal = Select_Literal_DPLL0(cnf);
33     else if (op == 1)
```

```
34          part_literal = Select_Literal_DPLL1(cnf);
35      else if (op == 2)
36          part_literal = Select_Literal_DPLL2(cnf);
37      else if (op == 3)
38          part_literal = Select_Literal_DPLL3(cnf);
39      if (DPLL1_Partition(cnf, part_literal, op))
40          return TRUE;
41      DPLL1_Recover(cnf, part_literal);
42      if (DPLL1_Partition(cnf, -part_literal, op))
43          return TRUE;
44      return FALSE;
45  }
46
47  //定义DPLL1_Partition函数, 变量分裂规则
48  status DPLL1_Partition(Conjunctive_Normal_Form_List& cnf, int
        literal, int op) {
49      ValueList[abs(literal)].tag = 1;
50      if (literal > 0)
51          ValueList[abs(literal)].is_value = 1;
52      else
53          ValueList[abs(literal)].is_value = 0;
54      backtracking_stack_DPLL1[top1++] = literal;
55      if (!DeleteLiteral(cnf, literal)) {
56          conflict_stack[ctop++] = literal;
57          return FALSE;
58      }
59      if (ans1 == cnf->clause_num) return TRUE;
60      Clause_List p;
61      Literal_List q;
62      while ((p = HasUnitClause(cnf)) != NULL) {
63          q = p->first_literal;
64          while (q && q->tag) q = q->next_literal;
65          ValueList[abs(q->literal)].tag = 1;
66          if (q->literal > 0)
67              ValueList[abs(q->literal)].is_value = 1;
```

```
68          else
69              ValueList[abs(q->literal)].is_value = 0;
70          backtracking_stack_DPLL1[top1++] = q->literal;
71          if (!DeleteLiteral(cnf, q->literal)) {
72              conflict_stack[ctop++] = q->literal;
73              return FALSE;
74          }
75          if (ans1 == cnf->clause_num) return TRUE;
76      }
77      int part_literal;
78      if (op == 0)
79          part_literal = Select_Literal_DPLL0(cnf);
80      else if (op == 1)
81          part_literal = Select_Literal_DPLL1(cnf);
82      else if (op == 2)
83          part_literal = Select_Literal_DPLL2(cnf);
84      else if (op == 3)
85          part_literal = Select_Literal_DPLL3(cnf);
86      if (DPLL1_Partition(cnf, part_literal, op))
87          return TRUE;
88      DPLL1_Recover(cnf, part_literal);
89      if (DPLL1_Partition(cnf, -part_literal, op))
90          return TRUE;
91      return FALSE;
92  }
93
94  //定义DPLL1_Recover函数, 变量回溯规则
95  status DPLL1_Recover(Conjunctive_Normal_Form_List& cnf, int
        literal) {
96      int tmp_literal;
97      while ((tmp_literal = backtracking_stack_DPLL1[--top1]) !=
            literal) {
98          ValueList[abs(tmp_literal)].tag = 0;
99          RecoverLiteral(cnf, tmp_literal);
100     }
```

```
101     RecoverLiteral(cnf, tmp_literal);
102     return OK;
103 }
104
105 //定义RecoverLiteral函数, 回复特定literal文字
106 status RecoverLiteral(Conjunctive_Normal_Form_List& cnf, int
        literal) {
107     if (!cnf) return ERROR;
108     Literal_List tmp = ValueList[abs(literal)].first;
109     while (tmp) {
110         if (literal == -tmp->literal) {
111             tmp->head->number++;
112             if (tmp->tag == -1) {
113                 tmp->tag = 0;
114                 break;
115             }
116             tmp->tag = 0;
117         }
118         else {
119             tmp->head->tag--;
120             if (!tmp->head->tag)
121                 ans1--;
122             tmp->head->number++;
123             if (tmp->tag == -1) {
124                 tmp->tag = 0;
125                 break;
126             }
127             tmp->tag = 0;
128         }
129         tmp = tmp->next_same_literal;
130     }
131     return OK;
132 }
133
134 //定义DeleteLiteral函数, 删除cnf结构中所有的literal文字
```

```
135 status DeleteLiteral(Conjunctive_Normal_Form_List& cnf, int
    literal) {
136     Literal_List tmp = ValueList[abs(literal)].first;
137     while (tmp) {
138         if (literal == -tmp->literal) {
139             tmp->tag = 1;
140             tmp->head->number--;
141             if (!tmp->head->number && !tmp->head->tag) {
142                 tmp->tag = -1;
143                 return FALSE;
144             }
145         }
146         else {
147             tmp->head->tag++;
148             if (tmp->head->tag == 1) {
149                 ans1++;
150             }
151             tmp->head->number--;
152             tmp->tag = 1;
153         }
154         tmp = tmp->next_same_literal;
155     }
156     return OK;
157 }
158
159 //定义Select_Literal_DPLL0函数, 变量决策策略
160 int Select_Literal_DPLL0(Conjunctive_Normal_Form_List& cnf) {
161     //     clock_t start = 0,finish = 0;//记录DPLL函数调用的起始和
        终止时间
162     //     int duration=0;//记录SAT求解时间
163     //     start = clock();
164     //     printf("1");
165     int i, num_literal;
166     for (i = 1; i <= cnf->literal_num; i++) {
167         if (!ValueList[i].tag) {
```

```
168                 return i;
169             }
170         }
171     }
172
173     //定义Select_Literal_DPLL1函数, 变量决策策略
174     int Select_Literal_DPLL1(Conjunctive_Normal_Form_List& cnf) {
175         //     printf("1");
176         if (!cnf) return 0;
177         Clause_List p = cnf->first_clause , q = NULL;
178         int num_literal = cnf->literal_num;
179         update_storevalue(cnf);
180
181         while (p) {
182             if (p->tag || !p->number) {
183                 p = p->next_clause;
184                 continue;
185             }
186             if (p->number < num_literal) {
187                 num_literal = p->number;
188                 q = p;
189             }
190             p = p->next_clause;
191         }
192         Literal_List m = q->first_literal , n = NULL;
193
194         int num_count = 0;
195         while (m) {
196             if (m->tag) {
197                 m = m->next_literal;
198                 continue;
199             }
200             if (ValueList[abs(m->literal)].num > num_count) {
201                 num_count = ValueList[abs(m->literal)].num;
202                 n = m;
```

```
203             }
204             m = m->next_literal;
205         }
206         if (ValueList[abs(n->literal)].pos >= ValueList[abs(n->
                literal)].nev)
207             return abs(n->literal);
208         else
209             return -abs(n->literal);
210     }
211
212     //定义Select_Literal_DPLL2函数, 变量决策策略
213     int Select_Literal_DPLL2(Conjunctive_Normal_Form_List& cnf) {
214         //     clock_t start = 0,finish = 0;//记录DPLL函数调用的起始和
                终止时间
215         //     int duration=0;//记录SAT求解时间
216         //     start = clock();
217         //     printf("1");
218         int num_literal = 0, num = 0, i;
219         update_storevalue(cnf);
220         for (i = 1; i <= cnf->literal_num; i++) {
221             if (!ValueList[i].tag && ValueList[i].num >= num) {
222                 num_literal = i;
223                 num = ValueList[i].num;
224             }
225         }
226         if (ValueList[num_literal].pos > ValueList[num_literal].nev)
227             return num_literal;
228         else
229             return -num_literal;
230     }
231
232     //定义Select_Literal_DPLL3函数, 变量决策策略
233     int Select_Literal_DPLL3(Conjunctive_Normal_Form_List& cnf) {
234         //     printf("1");
235         if (!cnf) return 0;
```

```
236     Clause_List p = cnf->first_clause, q = NULL;
237     int score = 0, i, literal;
238     update_storevalue2(cnf);
239     for (i = 1; i <= cnf->literal_num; i++) {
240         if (!ValueList[i].tag && ValueList[i].score >= score) {
241             score = ValueList[i].score;
242             literal = i;
243         }
244     }
245     if (ValueList[literal].pos > ValueList[literal].nev)
246         return literal;
247     else
248         return -literal;
249 }
250 
251 //定义update_storevalue2函数，更新存储文字
252 status update_storevalue2(Conjunctive_Normal_Form_List& cnf) {
253     int tmp0;
254     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
255         ValueList[tmp0].num = 0;
256         ValueList[tmp0].nev = 0;
257         ValueList[tmp0].pos = 0;
258         ValueList[tmp0].score = 0;
259     }
260     int literal;
261     Literal_List q, r;
262     while (ctop) {
263         literal = conflict_stack[--ctop];
264 
265         q = ValueList[abs(literal)].first;
266         while (q) {
267             if (q->head->tag) {
268                 q = q->next_same_literal;
269                 continue;
270             }
```

```
271             r = q->head->first_literal;
272             while (r) {
273                 if (r->tag) {
274                     r = r->next_literal;
275                     continue;
276                 }
277                 ValueList[abs(r->literal)].score += (cnf->
                        clause_num / (r->head->number * r->head->
                        number));
278                 r->literal > 0 ? ValueList[abs(r->literal)].pos++
                        : ValueList[abs(r->literal)].nev++;
279                 r = r->next_literal;
280             }
281             q = q->next_same_literal;
282         }
283     }
284     Clause_List p = cnf->first_clause;
285     while (p) {
286         if (p->tag) {
287             p = p->next_clause;
288             continue;
289         }
290         q = p->first_literal;
291         while (q) {
292             if (q->tag) {
293                 q = q->next_literal;
294                 continue;
295             }
296             ValueList[abs(q->literal)].score += (cnf->clause_num
                    / (p->number * p->number));
297             if (ValueList[abs(q->literal)].score > 100) {
298                 for (int i = 1; i <= cnf->literal_num; i++) {
299                     ValueList[i].score /= 10;
300                 }
301             }
```

```
302             q->literal > 0 ? ValueList[abs(q->literal)].pos++ :
                    ValueList[abs(q->literal)].nev++;
303             q = q->next_literal;
304         }
305         p = p->next_clause;
306     }
307     return OK;
308 }
309
310 //定义update_storevalue函数，更新存储文字
311 status update_storevalue(Conjunctive_Normal_Form_List& cnf) {
312     int tmp0;
313     for (tmp0 = 0; tmp0 <= cnf->literal_num; tmp0++) {
314         ValueList[tmp0].num = 0;
315         ValueList[tmp0].nev = 0;
316         ValueList[tmp0].pos = 0;
317     }
318     Clause_List p = cnf->first_clause;
319     Literal_List q;
320     while (p) {
321         if (p->tag) {
322             p = p->next_clause;
323             continue;
324         }
325         q = p->first_literal;
326         while (q) {
327             if (q->tag) {
328                 q = q->next_literal;
329                 continue;
330             }
331             ValueList[abs(q->literal)].num++;
332             if (ValueList[abs(q->literal)].num > 100) {
333                 for (int i = 1; i <= cnf->literal_num; i++) {
334                     ValueList[i].num /= 10;
335                 }
```

```
336                 }
337                 q->literal > 0 ? ValueList[abs(q->literal)].pos++ :
                        ValueList[abs(q->literal)].nev++;
338                 q = q->next_literal;
339             }
340             p = p->next_clause;
341         }
342         return OK;
343     }
344
345     //定义print函数, 输出真值表, 用于验证和调试
346     status print(Conjunctive_Normal_Form_List& cnf) {
347         for (int i = 1; i <= cnf->literal_num; i++) {
348             printf("{%d□%d}", ValueList[i].tag, ValueList[i].is_value
                    );
349         }
350         printf("\n");
351         return OK;
352     }
353
354     //定义CreateCNF函数, 创建CNF链表结果
355     status CreateCNF(FILE* fp, Conjunctive_Normal_Form_List& cnf) {
356         char readfile[20];//定义字符类型数组记录在文件中
357         int i, tmp;//整型变量定义
358         Clause_List clause_tmp1, clause_tmp2;//定义临时结点变量
359
360         //初始化CNF
361         cnf = (Conjunctive_Normal_Form*)malloc(sizeof(
                    Conjunctive_Normal_Form));
362         cnf->first_clause = NULL;
363
364         //CNF文件读取
365         while (fscanf(fp, "%s", readfile) != EOF) { //循环读文件
366             if (strcmp(readfile, "p") == 0) //持续读取文件直至开始标
                    志'p'
```

```
367                 break;
368         }
369         while (fscanf(fp, "%s", readfile) != EOF) {
370             if (strcmp(readfile, "cnf") == 0) {//从文件中读到字符串“
                    cnf”
371                 fscanf(fp, "%d", &cnf->literal_num);//读取CNF文件变元
                        数并存入literal_num
372                 fscanf(fp, "%d", &cnf->clause_num);//读取CNF文件子句
                        总数并存入clause_num
373                 break;
374             }
375         }
376
377         printf("literal_num:%d\nclause_num:%d\n", cnf->literal_num,
                cnf->clause_num);//输出基本信息
378
379         //定义并初始化变元真值表
380         ValueList = (ArgueValue*)malloc((cnf->literal_num + 2) *
                sizeof(ArgueValue));
381         if (!ValueList) return OVERFLOW;//没分配成功，返回OVERFLOW
382         for (i = 1; i <= cnf->literal_num; i++) {
383             ValueList[i].is_value = 0;
384             ValueList[i].tag = 0;
385             ValueList[i].num = 0;
386             ValueList[i].nev = 0;
387             ValueList[i].pos = 0;
388             ValueList[i].score = 0;
389             ValueList[i].first = NULL;
390         }
391
392         //创建CNF链式结构
393         if (cnf->clause_num) {
394             if (fscanf(fp, "%d", &tmp) != EOF && tmp != 0) {
395                 //创建CNF子句的头指针
396 //                  printf("%d ", tmp);
```

```
397             clause_tmp1 = (Clause_List)malloc(sizeof(Clause_Node)
                    );
398             if (!clause_tmp1) return OVERFLOW;
399             cnf->first_clause = clause_tmp1;//定义头节点
400             clause_tmp2 = clause_tmp1;
401             CreateClause(fp, clause_tmp1, tmp);//创建其对应子句链
                    表
402             clause_tmp1->next_clause = NULL;
403             clause_tmp1->tag = 0;
404             i = 2;
405             //创建完整的CNF子句链式结构
406             while (i++ <= cnf->clause_num) {
407                 fscanf(fp, "%d", &tmp);
408                 //                  printf("%d ", tmp);
409                 clause_tmp1 = (Clause_List)malloc(sizeof(
                        Clause_Node));
410                 if (!clause_tmp1) return OVERFLOW;
411                 CreateClause(fp, clause_tmp1, tmp);//创建其对应子
                        句链表
412                 clause_tmp2->next_clause = clause_tmp1;
413                 clause_tmp2 = clause_tmp1;
414                 clause_tmp2->tag = 0;
415                 clause_tmp1->next_clause = NULL;
416             }
417         }
418     }
419     return OK;
420 }
421
422 //定义CreateClause函数, 创建子句链表
423 //输入: 文件指针, 子句链式结点, 首值
424 //输出: 状态
425 status CreateClause(FILE* fp, Clause_List& sentence, int first) {
426     int num = 1, tmp;//定义文字数目, 临时变量
427     if (!first) return ERROR;
```

```
428     Literal_List literal_tmp1, literal_tmp2;//定义临时结点变量
429     literal_tmp1 = (Literal_List)malloc(sizeof(Literal_Node));
430     if (!literal_tmp1) return OVERFLOW;
431     //创建文字链式结构头节点
432     literal_tmp1->literal = first;
433     literal_tmp1->tag = 0;
434     ValueList[abs(first)].num++;
435     first > 0 ? ValueList[abs(first)].pos++ : ValueList[abs(first
            )].nev++;
436     if (!ValueList[abs(first)].first) {
437         literal_tmp1->head = sentence;
438         ValueList[abs(first)].first = literal_tmp1;
439         ValueList[abs(first)].first->next_same_literal = NULL;
440     }
441     else {
442         literal_tmp1->head = sentence;
443         literal_tmp1->next_same_literal = ValueList[abs(first)].
                first;
444         ValueList[abs(first)].first = literal_tmp1;
445     }
446     literal_tmp1->next_literal = NULL;
447     literal_tmp2 = literal_tmp1;
448     literal_tmp2->tag = 0;
449     sentence->first_literal = literal_tmp1;
450
451     // 创建文字链表结点
452     fscanf(fp, "%d", &tmp);
453     while (tmp != 0) {
454         ValueList[abs(tmp)].num++;
455         tmp > 0 ? ValueList[abs(tmp)].pos++ : ValueList[abs(tmp)
                ].nev++;
456         literal_tmp1 = (Literal_List)malloc(sizeof(Literal_Node))
                ;
457         if (!literal_tmp1) return OVERFLOW;
458         literal_tmp1->literal = tmp;
```

```
459             literal_tmp1->next_literal = NULL;
460             literal_tmp2->next_literal = literal_tmp1;
461             literal_tmp2 = literal_tmp1;
462             if (!ValueList[abs(tmp)].first) {
463                 literal_tmp1->head = sentence;
464                 ValueList[abs(tmp)].first = literal_tmp1;
465                 ValueList[abs(tmp)].first->next_same_literal = NULL;
466             }
467             else {
468                 literal_tmp1->head = sentence;
469                 literal_tmp1->next_same_literal = ValueList[abs(tmp)
                        ].first;
470                 ValueList[abs(tmp)].first = literal_tmp1;
471             }
472             literal_tmp2->tag = 0;
473             num++;
474             fscanf(fp, "%d", &tmp);//读取下一文字
475
476         }
477         sentence->number = num;
478         return OK;
479     }
480
481     //注意传入的是前一个结点
482     //定义DestroyClause函数, 销毁子句结点
483     //输入: 字句链式节点
484     //输出: 状态
485     status DestroyClause(Clause_List& sentence) {
486         Clause_List p;
487         p = sentence->next_clause;
488         if (!p) return ERROR;//判断合理性
489         sentence->next_clause = p->next_clause;
490         while (p->first_literal->next_literal) DestroyLiteral(p->
                first_literal);
491         free(p->first_literal);
```

```
492     p->first_literal = NULL;
493     free(p);
494     p = NULL;
495     return OK;
496 }
497
498 //注意传入的是前一个结点
499 //定义DestroyLiteral函数, 销毁文字结点
500 //输入: 文字链式结点
501 //输出: 状态
502 status DestroyLiteral(Literal_List& word) {
503     Literal_List p;
504     p = word->next_literal;
505     if (!p) return ERROR;//判断合理性
506     word->next_literal = p->next_literal;
507     free(p);
508     p = NULL;
509     return OK;
510 }
511
512 //定义DestroyCNF函数, 销毁CNF文件
513 //输入: cnf指针
514 //输出: 状态
515 status DestroyCNF(Conjunctive_Normal_Form_List& cnf) {
516     if (!cnf) return ERROR;//判断合理性
517     for (int i = 1; i <= cnf->literal_num; i++) {
518         ValueList[i].is_value = 0;
519         ValueList[i].tag = 0;
520         ValueList[i].num = 0;
521         ValueList[i].nev = 0;
522         ValueList[i].pos = 0;
523         ValueList[i].score = 0;
524         ValueList[i].first = NULL;
525     }
526     Clause_List clause_tmp1;
```

```
527     while (cnf->first_clause->next_clause) DestroyClause(cnf->
            first_clause);
528     clause_tmp1 = (Clause_List)malloc(sizeof(Clause_Node));
529     clause_tmp1->next_clause = cnf->first_clause;
530     DestroyClause(clause_tmp1);
531     free(clause_tmp1);
532     clause_tmp1 = NULL;
533     free(cnf);
534     cnf = NULL;
535     return OK;
536 }
537
538
539 //定义HasUnitClause函数, 评估CNF是否含有单子句
540 //输入: cnf指针
541 //输出: 文字链式结点
542 Clause_List HasUnitClause(Conjunctive_Normal_Form_List& cnf) {
543     if (!cnf) return NULL;
544     Clause_List p;
545     for (p = cnf->first_clause; p; p = p->next_clause) {
546         if (!p->tag && p->number == 1) {
547             return p;
548         }
549     }
550     return NULL;
551 }
552
553 status check_print2(Conjunctive_Normal_Form_List& cnf) {
554     Clause_List p = cnf->first_clause;
555     Literal_List q = NULL;
556     int flag;
557     while (p) {
558         q = p->first_literal;
559         flag = 0;
560         while (q) {
```

```
561             if (q->literal * ValueList[abs(q->literal)].is_value
                    > 0) {
562                 flag = 1;
563                 break;
564             }
565             q = q->next_literal;
566         }
567         printf("%d\n", flag);
568         p = p->next_clause;
569     }
570     return OK;
571 }
572
573 //定义store_document函数, 存储文件
574 void store_document(Conjunctive_Normal_Form_List& cnf, char*
    filename, int d, int time0) {
575     int i = 0;
576     while (1) {
577         if (filename[i] == 'c' && filename[i + 1] == 'n' &&
                filename[i + 2] == 'f')
578             break;
579         i++;
580     }
581     filename[i] = 'r', filename[i + 1] = 'e', filename[i + 2] = '
        s';
582     FILE* fp;
583     fp = fopen(filename, "w");
584     fprintf(fp, "%d\n", d);
585     for (int tmp0 = 1; tmp0 < cnf->literal_num; tmp0++) {
586         if (ValueList[tmp0].is_value == 1)
587             fprintf(fp, "%d□", tmp0);
588         else
589             fprintf(fp, "%d□", -tmp0);
590         if (tmp0 % 10 == 0)
591             fprintf(fp, "\n");
```

```
592     }
593     fprintf(fp, "%d", time0);
594     fclose(fp);
595     return;
596 }
597
598 void traveser_cnf(Conjunctive_Normal_Form_List cnf) {
599     Clause_List p = cnf->first_clause;
600     Literal_List q = NULL;
601     int flag;
602     printf("变元数：%d\n子句数目：%d\n", cnf->literal_num, cnf->
          clause_num);
603     while (p) {
604         q = p->first_literal;
605         while (q) {
606             printf("%d□", q->literal);
607             q = q->next_literal;
608         }
609         printf("\n");
610         p = p->next_clause;
611     }
612     return;
613 }
614
615 //定义prove_cnf函数，进行验证
616 void prove_cnf(Conjunctive_Normal_Form_List cnf) {
617     Clause_List p = cnf->first_clause;
618     Literal_List q = NULL;
619     int flag;
620     while (p) {
621         q = p->first_literal;
622         flag = 0;
623         while (q) {
624             if (q->literal * ValueList[abs(q->literal)].is_value
                  > 0) {
```

```
625                     flag = 1;
626                     break;
627                 }
628                 q = q->next_literal;
629             }
630             if (!flag) {
631                 printf("该cnf公式验证无解!\n");
632                 return;
633             }
634             p = p->next_clause;
635         }
636         printf("该cnf公式验证有解!\n");
637         return;
638     }
639
640     //定义show_cnf函数, 输出cnf文件求解结果
641     void show_cnf(Conjunctive_Normal_Form_List cnf) {
642         for (int i = 1; i <= cnf->literal_num; i++) {
643             if (ValueList[i].is_value == 1)
644                 printf("%d□", i);
645             else
646                 printf("%d□", -i);
647             if (!(i % 10))
648                 printf("\n");
649         }
650         return;
651     }
```

文件名：func_DPLL2.cpp

功能：DPLL2 框架的实现

```
1  //
2  // Created by hp on 2022-09-01.
3  //
4
5  #include "define.h"
```

```
6
7  /*定义有效的全局变量*/
8  extern ArgueValue* ValueList;//定义变元真值表
9  bool isConflict;
10 int prehandle_stack[MAXN], pre_top;
11 int decision_stack[MAXN], dec_top;
12 int decision_num[MAXN];
13 int conflict_stack_[MAXN], con_top;
14 double time_sum;
15
16 int cnt = 0, times = 1, cnt0 = 0;
17
18 //定义CreateCNF2函数, 创建CNF链表结果
19 status CreateCNF2(FILE* fp, Conjunctive_Normal_Form_List& cnf) {
20     isConflict = FALSE;
21     char readfile[20];//定义字符类型数组记录在文件中
22     int i, tmp;//整型变量定义
23     Clause_List clause_tmp1, clause_tmp2;//定义临时结点变量
24
25     //初始化CNF
26     cnf = (Conjunctive_Normal_Form*)malloc(sizeof(
           Conjunctive_Normal_Form));
27     cnf->first_clause = NULL;
28
29     //CNF文件读取
30     while (fscanf(fp, "%s", readfile) != EOF) { //循环读文件
31         if (strcmp(readfile, "p") == 0) //持续读取文件直至开始标
               志' p '
32             break;
33     }
34     while (fscanf(fp, "%s", readfile) != EOF) {
35         if (strcmp(readfile, "cnf") == 0) {//从文件中读到字符串"
               cnf"
36             fscanf(fp, "%d", &cnf->literal_num);//读取CNF文件变元
                   数并存入literal_num
```

```
37              fscanf(fp, "%d", &cnf->clause_num);//读取CNF文件子句
                    总数并存入clause_num
38              break;
39          }
40      }
41
42      //printf("literal_num:%d\nclause_num:%d\n", cnf->literal_num,
            cnf->clause_num);//输出基本信息
43
44      //定义并初始化变元真值表
45      ValueList = (ArgueValue*)malloc((cnf->literal_num + 2) *
            sizeof(ArgueValue));
46      if (!ValueList) return OVERFLOW;//没分配成功, 返回OVERFLOW
47      for (i = 1; i <= cnf->literal_num; i++) {
48          ValueList[i].is_value = 0;
49          ValueList[i].tag = 0;
50          ValueList[i].dep = 0;
51          ValueList[i].num = 0;
52          ValueList[i].nev = 0;
53          ValueList[i].pos = 0;
54          ValueList[i].score = 0;
55          ValueList[i].first = NULL;
56      }
57
58      //创建CNF链式结构
59      if (cnf->clause_num) {
60          if (fscanf(fp, "%d", &tmp) != EOF && tmp != 0) {
61              //创建CNF子句的头指针
62  //              printf("%d ", tmp);
63              clause_tmp1 = (Clause_List)malloc(sizeof(Clause_Node)
                    );
64              if (!clause_tmp1) return OVERFLOW;
65              cnf->first_clause = clause_tmp1;//定义头节点
66              clause_tmp2 = clause_tmp1;
67              CreateClause2(fp, clause_tmp1, tmp);//创建其对应子句
```

```
                链表
68              clause_tmp1->next_clause = NULL;
69              i = 2;
70              //创建完整的CNF子句链式结构
71              while (i++ <= cnf->clause_num) {
72                  fscanf(fp, "%d", &tmp);
73                  //                  printf("%d ", tmp);
74                  clause_tmp1 = (Clause_List)malloc(sizeof(
                        Clause_Node));
75                  if (!clause_tmp1) return OVERFLOW;
76                  CreateClause2(fp, clause_tmp1, tmp);//创建其对应
                        子句链表
77                  clause_tmp2->next_clause = clause_tmp1;
78                  clause_tmp2 = clause_tmp1;
79                  clause_tmp1->next_clause = NULL;
80              }
81          }
82      }
83      return OK;
84  }
85
86  //定义CreateClause2函数, 创建子句链表
87  //输入: 文件指针, 子句链式结点, 首值
88  //输出: 状态
89  status CreateClause2(FILE* fp, Clause_List& sentence, int first)
        {
90      int num = 1, tmp;//定义文字数目, 临时变量
91      //     printf("%d ", first);
92      if (!first) return ERROR;
93      Literal_List literal_tmp1, literal_tmp2;//定义临时结点变量
94      literal_tmp1 = (Literal_List)malloc(sizeof(Literal_Node));
95      if (!literal_tmp1) return OVERFLOW;
96      //创建文字链式结构头节点
97      literal_tmp1->literal = first;
98      ValueList[abs(first)].num++;
```

```
99      first > 0 ? ValueList[abs(first)].pos++ : ValueList[abs(first
            )].nev++;
100     if (!ValueList[abs(first)].first) {
101         literal_tmp1->head = sentence;
102         ValueList[abs(first)].first = literal_tmp1;
103         ValueList[abs(first)].first->next_same_literal = NULL;
104     }
105     else {
106         literal_tmp1->head = sentence;
107         literal_tmp1->next_same_literal = ValueList[abs(first)].
                first;
108         ValueList[abs(first)].first = literal_tmp1;
109     }
110     literal_tmp1->next_literal = NULL;
111     literal_tmp2 = literal_tmp1;
112     sentence->first_literal = literal_tmp1;
113     sentence->watch_literal[0] = literal_tmp1->literal;
114
115     // 创建文字链表结点
116     fscanf(fp, "%d", &tmp);
117     if (!tmp) {
118         prehandle_stack[pre_top++] = first;
119         int op = (first > 0 ? 1 : -1);
120         int oldliteral = ValueList[abs(first)].is_value;
121         if (oldliteral && oldliteral != op) {
122             isConflict = TRUE;
123         }
124         ValueList[abs(first)].is_value = (first > 0 ? 1 : -1);
125     }
126     while (tmp != 0) {
127         //          printf("%d ", tmp);
128         ValueList[abs(tmp)].num++;
129         tmp > 0 ? ValueList[abs(tmp)].pos++ : ValueList[abs(tmp)
                ].nev++;
130         literal_tmp1 = (Literal_List)malloc(sizeof(Literal_Node))
```

```
            ;
131         if (!literal_tmp1) return OVERFLOW;
132         literal_tmp1->literal = tmp;
133         literal_tmp1->next_literal = NULL;
134         literal_tmp2->next_literal = literal_tmp1;
135         literal_tmp2 = literal_tmp1;
136         if (!ValueList[abs(tmp)].first) {
137             literal_tmp1->head = sentence;
138             ValueList[abs(tmp)].first = literal_tmp1;
139             ValueList[abs(tmp)].first->next_same_literal = NULL;
140         }
141         else {
142             literal_tmp1->head = sentence;
143             literal_tmp1->next_same_literal = ValueList[abs(tmp)
                    ].first;
144             ValueList[abs(tmp)].first = literal_tmp1;
145         }
146         num++;
147         fscanf(fp, "%d", &tmp);//读取下一文字
148     }if (sentence->first_literal->next_literal)
149         sentence->watch_literal[1] = sentence->first_literal->
                next_literal->literal;
150     sentence->number = num;
151     //     printf("%d", num);
152     //     printf("\n");
153     return OK;
154 }
155
156 //注意传入的是前一个结点
157 //定义DestroyClause2函数, 销毁子句结点
158 //输入: 字句链式节点
159 //输出: 状态
160 status DestroyClause2(Clause_List& sentence) {
161     Clause_List p;
162     p = sentence->next_clause;
```

```
163     if (!p) return ERROR;//判断合理性
164     sentence->next_clause = p->next_clause;
165     while (p->first_literal->next_literal) DestroyLiteral2(p->
            first_literal);
166     free(p->first_literal);
167     p->first_literal = NULL;
168     free(p);
169     p = NULL;
170     return OK;
171 }
172 
173 //注意传入的是前一个结点
174 //定义DestroyLiteral函数, 销毁文字结点
175 //输入: 文字链式结点
176 //输出: 状态
177 status DestroyLiteral2(Literal_List& word) {
178     Literal_List p;
179     p = word->next_literal;
180     if (!p) return ERROR;//判断合理性
181     word->next_literal = p->next_literal;
182     free(p);
183     p = NULL;
184     return OK;
185 }
186 
187 //定义DestroyCNF2函数, 销毁CNF文件
188 //输入: cnf指针
189 //输出: 状态
190 status DestroyCNF2(Conjunctive_Normal_Form_List& cnf) {
191     if (!cnf) return ERROR;//判断合理性
192     Clause_List clause_tmp1;
193     while (cnf->first_clause->next_clause) DestroyClause2(cnf->
            first_clause);
194     clause_tmp1 = (Clause_List)malloc(sizeof(Clause_Node));
195     clause_tmp1->next_clause = cnf->first_clause;
```

```
196     DestroyClause2(clause_tmp1);
197     free(clause_tmp1);
198     clause_tmp1 = NULL;
199     free(cnf);
200     cnf = NULL;
201     return OK;
202 }
203
204 //定义Select_Literal3函数, 变量决策策略
205 int select_literal3(Conjunctive_Normal_Form_List cnf) {
206
207     //gettimeofday(&st, NULL);
208
209     int tmp0, flag = 0, tmp_literal, stp;
210     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
211         if (!ValueList[tmp0].is_value) {
212             flag = 1;
213         }
214     }
215     srand(int(time(0)));
216     int a;
217     if (!flag) return 0;
218     if (cnt0 > 5000 * times) {
219         times++;
220         for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
221             ValueList[tmp0].is_value = 0;
222             ValueList[tmp0].dep = 0;
223         }
224         //          printf("conflict:%d\n\n\n\n\n\n\n\n", times);
225     }
226
227     Clause_List p;
228     Literal_List q = NULL, r;
229     int literal;
230     while (con_top) {
```

```
231         literal = conflict_stack_[--con_top];
232         q = ValueList[abs(literal)].first;
233         while (q) {
234             if (q->head->tag) {
235                 q = q->next_same_literal;
236                 continue;
237             }
238             r = q->head->first_literal;
239             while (r) {
240                 if (r->tag) {
241                     r = r->next_literal;
242                     continue;
243                 }
244                 ValueList[abs(r->literal)].score += cnf->
                        literal_num / (r->head->number * r->head->
                        number);
245                 if (ValueList[abs(r->literal)].score > 5000) {
246                     //                      printf("score:%d\n",
                            ValueList[abs(r->literal)].score);
247                     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0
                            ++) {
248                         ValueList[tmp0].score /= 50;
249                     }
250                 }
251                 r->literal > 0 ? ValueList[abs(r->literal)].pos++
                        : ValueList[abs(r->literal)].nev++;
252                 r = r->next_literal;
253             }
254             q = q->next_same_literal;
255         }
256     }
257     stp = 0;
258     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
259         if (!ValueList[tmp0].is_value && ValueList[tmp0].score >=
                stp) {
```

```
260             stp = ValueList[tmp0].score;
261             literal = tmp0;
262         }
263     }
264     cnt++;
265     //gettimeofday(&ed, NULL);
266
267     //time_total = (ed.tv_sec - st.tv_sec) + (ed.tv_usec - st.
            tv_usec) / 1000000.0;
268     //time_sum += time_total;
269     //     printf("[time_total%d]:%lf S\n", cnt, time_total);
270     if (ValueList[abs(literal)].pos >= ValueList[abs(literal)].
            nev)
271         return abs(literal);
272     else
273         return -abs(literal);
274 }
275
276 //定义Select_Literal2函数, 变量决策策略
277 int select_literal2(Conjunctive_Normal_Form_List cnf) {
278     int tmp0, flag = 0;
279     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
280         ValueList[tmp0].num = 0;
281         ValueList[tmp0].pos = 0;
282         ValueList[tmp0].nev = 0;
283         ValueList[tmp0].score = 0;
284         if (!ValueList[tmp0].is_value)
285             flag = 1;
286     }
287     if (!flag) return 0;
288     Clause_List p;
289     Literal_List q = NULL, r;
290     int literal;
291     while (con_top) {
292         literal = conflict_stack_[--con_top];
```

```
293         q = ValueList[abs(literal)].first;
294         while (q) {
295             r = q->head->first_literal;
296             while (r) {
297                 if (ValueList[r->literal].is_value) {
298                     r = r->next_literal;
299                     continue;
300                 }
301                 ValueList[abs(r->literal)].score += (cnf->
                        literal_num / (r->head->number * r->head->
                        number));
302                 r->literal > 0 ? ValueList[abs(r->literal)].pos++
                         : ValueList[abs(r->literal)].nev++;
303                 r = r->next_literal;
304             }
305             q = q->next_same_literal;
306         }
307     }
308     Clause_List h = NULL;
309     for (p = cnf->first_clause; p; p = p->next_clause) {
310         flag = 0;
311         q = p->first_literal;
312         while (q) {
313             if (q->literal * ValueList[abs(q->literal)].is_value
                    > 0) {
314                 flag = 1;
315                 break;
316             }
317             q = q->next_literal;
318         }
319         if (flag) continue;
320         r = p->first_literal;
321         while (r) {
322             ValueList[abs(r->literal)].num++;
323             ValueList[abs(r->literal)].score += (cnf->literal_num
```

```
                / (r->head->number * r->head->number));
324             r->literal > 0 ? ValueList[abs(r->literal)].pos++ :
                    ValueList[abs(r->literal)].nev++;
325             if (ValueList[abs(r->literal)].score > 5000) {
326                 //                  printf("score:%d\n", ValueList[
                        abs(r->literal)].score);
327                 for (int i = 1; i <= cnf->literal_num; i++) {
328                     ValueList[i].score /= 50;
329                 }
330             }
331             r = r->next_literal;
332         }
333     }
334     int i, score = 0;
335     literal = 0;
336     for (i = 1; i <= cnf->literal_num; i++) {
337         if (!ValueList[i].is_value && ValueList[i].score >= score
                ) {
338             score = ValueList[i].score;
339             literal = i;
340         }
341     }
342     if (ValueList[literal].pos >= ValueList[literal].nev)
343         return abs(literal);
344     else
345         return -abs(literal);
346 }
347
348 //定义update_storevalue函数, 更新存储文字
349 int update_value(Conjunctive_Normal_Form_List cnf) {
350     int tmp0, tmp_literal;
351     int score;
352     Literal_List p = NULL;
353     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
354         p = ValueList[tmp0].first;
```

```
355             score = 0;
356             while (p) {
357                 score += (cnf->literal_num) / (p->head->number * p->
                        head->number);
358                 p = p->next_literal;
359             }
360             ValueList[tmp0].score = score;
361         }
362         return OK;
363     }
364
365     //定义Select_Literal函数, 变量决策策略
366     int select_literal(Conjunctive_Normal_Form_List cnf) {
367
368         //gettimeofday(&st, NULL);
369
370         int tmp0, flag = 0, tmp_literal, stp;
371         for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
372             if (!ValueList[tmp0].is_value) {
373                 flag = 1;
374             }
375         }
376         srand(int(time(0)));
377         int a;
378         if (!flag) return 0;
379         if (cnt0 > 100 * times) {
380             times++;
381             for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
382                 ValueList[tmp0].is_value = 0;
383             }
384             //          printf("conflict:%d\n\n\n\n\n\n\n", times);
385         }
386
387         Clause_List p;
388         Literal_List q = NULL, r;
```

```
389     int literal;
390     while (con_top) {
391         literal = conflict_stack_[--con_top];
392         q = ValueList[abs(literal)].first;
393         while (q) {
394             if (q->head->tag) {
395                 q = q->next_same_literal;
396                 continue;
397             }
398             r = q->head->first_literal;
399             while (r) {
400                 if (r->tag) {
401                     r = r->next_literal;
402                     continue;
403                 }
404                 ValueList[abs(r->literal)].score += cnf->
                    literal_num / (r->head->number * r->head->
                    number);
405                 if (ValueList[abs(r->literal)].score > 500) {
406                     //                      printf("score:%d\n",
                        ValueList[abs(r->literal)].score);
407                     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0
                        ++) {
408                         ValueList[tmp0].score /= 10;
409                     }
410                 }
411                 r->literal > 0 ? ValueList[abs(r->literal)].pos++
                        : ValueList[abs(r->literal)].nev++;
412                 r = r->next_literal;
413             }
414             q = q->next_same_literal;
415         }
416     }
417     stp = 0;
418     for (tmp0 = 1; tmp0 <= cnf->literal_num; tmp0++) {
```

```
419             if (!ValueList[tmp0].is_value && ValueList[tmp0].score >=
                    stp) {
420                 stp = ValueList[tmp0].score;
421                 literal = tmp0;
422             }
423         }
424         cnt++;
425         //gettimeofday(&ed, NULL);
426 
427         time_total = (ed.tv_sec - st.tv_sec) + (ed.tv_usec - st.
                tv_usec) / 1000000.0;
428         time_sum += time_total;
429         //     printf("[time_total%d]:%lf S\n", cnt, time_total);
430         if (ValueList[abs(literal)].pos >= ValueList[abs(literal)].
                nev)
431             return abs(literal);
432         else
433             return -abs(literal);
434 }
435 
436 //定义decide函数, 进行变量决策
437 status decide(Conjunctive_Normal_Form_List& cnf) {
438         if (pre_top)
439             return TRUE;
440         int part_literal = select_literal3(cnf);
441         if (!part_literal) {
442             return FALSE;
443         }
444         else {
445             decision_stack[dec_top] = part_literal;
446             decision_num[dec_top] = 1;
447             dec_top++;
448             return TRUE;
449         }
450 }
```

```
451
452 //定义bcp过程，实现单子句传播
453 status bcp(Conjunctive_Normal_Form_List& cnf) {
454     int literal;
455     if (dec_top) {
456         literal = decision_stack[dec_top - 1];
457         prehandle_stack[pre_top++] = literal;
458     }
459     Literal_List x = NULL;
460     Clause_List p = NULL;
461     while (pre_top) {
462         int pre_literal = prehandle_stack[--pre_top];
463         //          printf("(%d)", pre_literal);
464         ValueList[abs(pre_literal)].is_value = (pre_literal > 0 ?
                1 : -1);
465         ValueList[abs(pre_literal)].dep = dec_top;
466         for (x = ValueList[abs(pre_literal)].first; x; x = x->
               next_same_literal) {
467             p = x->head;
468             if (p->number == 1) {
469                 continue;
470             }
471             if ((abs(p->watch_literal[0]) == abs(pre_literal)) ||
                    (abs(p->watch_literal[1]) == abs(pre_literal))) {
472                 int up, st;
473                 if (abs(p->watch_literal[0]) == abs(pre_literal))
                        {
474                     up = 0;
475                     st = 1;
476                 }
477                 else {
478                     up = 1;
479                     st = 0;
480                 }
481                 if (p->watch_literal[up] == pre_literal) {
```

```
482                     continue;
483                 }
484                 Literal_List q = NULL;
485                 for (q = p->first_literal; q; q = q->next_literal
                        ) {
486                     if (q->literal == p->watch_literal[0] || q->
                            literal == p->watch_literal[1]) {
487                         continue;
488                     }
489                     if (q->literal * ValueList[abs(q->literal)].
                            is_value < 0) {
490                         continue;
491                     }
492                     break;
493                 }
494                 if (q == NULL) {
495                     if (!ValueList[abs(p->watch_literal[st])].
                            is_value) {
496                         prehandle_stack[pre_top++] = p->
                                watch_literal[st];
497                     }
498                     else if (p->watch_literal[st] * ValueList[abs
                            (p->watch_literal[st])].is_value < 0) {
499                         conflict_stack_[con_top++] = pre_literal;
500                         return FALSE;
501                     }
502                 }
503                 else {
504                     p->watch_literal[up] = q->literal;
505                 }
506             }
507         }
508     }
509     return TRUE;
510 }
```

```
511
512 //定义resolveConflict函数，解决冲突
513 status resolveConflict(Conjunctive_Normal_Form_List& cnf) {
514     //     printf("a");
515     int i;
516     while (dec_top > 0 && decision_num[dec_top - 1] == 2) {
517         decision_num[dec_top - 1] = 0;
518         dec_top--;
519     }
520     if (!dec_top) {
521         return FALSE;
522     }
523     decision_num[dec_top - 1]++;
524     decision_stack[dec_top - 1] = -decision_stack[dec_top - 1];
525     pre_top = 0;
526     for (i = 1; i <= cnf->literal_num; i++) {
527         if (ValueList[i].dep >= dec_top) {
528             ValueList[i].is_value = 0;
529             ValueList[i].dep = 0;
530         }
531     }
532     return TRUE;
533 }
534
535 //定义DPLL1函数，作为处理cnf文件的第一个DPLL文件
536 status DPLL2(Conjunctive_Normal_Form_List cnf) {
537     update_value(cnf);
538     if (isConflict) {
539         // conflict in single literal clauses
540         return FALSE;
541     }
542     while (TRUE) {
543         if (!decide(cnf)) {
544             //             printf("[total_time_sum]:%lf", time_sum
                    * 1000);
```

```
545                 return true;
546             }
547             while (!bcp(cnf)) {
548                 cnt0++;
549                 if (!resolveConflict(cnf)) {
550                     //                  check_print2(cnf);
551                     return false;
552                 }
553             }
554         }
555 }
```

文件名：func_SUDOKU.cpp

功能：处理数独问题相关函数实现

```
1  //
2  // Created by hp on 2022-09-04.
3  //
4
5
6  #include "define.h"
7
8  int sudoku_table[18][9];//定义全局变量int类型二维数组存储双数独终
       盘
9  int users_sudoku[18][9];//存储输出的含空格数独问题格局
10 int shuffle_value[164];//存储被打乱的变元
11 extern ArgueValue* ValueList;//定义变元真值表
12 enum { easy = 1, medium, difficult };
13 int easynum = 50, midnum = 80, difnum = 110;
14
15 //定义CreatePreSudokuFile函数, 规约数独为CNF文件
16 FILE* CreatePreSudokuFile(Conjunctive_Normal_Form_List& cnf) {
17     int x, y, z, i, j, k, l;
18     FILE* fp;
19     fp = fopen("SudokuTableBase.cnf", "w");
20     if (fp == NULL) {
```

```
21          printf("File open error!\n");
22          exit(0);
23      }
24      fprintf(fp, "p cnf 1458 20736\n");//共有729个变元，9*9个数独空格每个格对应9个变元，填入1～9中某一值则对应变元为真，其他为假
25
26      //每个格内只能含有1-9中的一个
27      for (x = 0; x < 9; x++) {
28          for (y = 0; y < 9; y++)
29              for (z = 1; z < 9; z++)
30                  for (i = z + 1; i <= 9; i++)
31                      fprintf(fp, "%d %d 0\n", -(81 * x + 9 * y + z), -(81 * x + 9 * y + i));
32      }
33      for (x = 9; x < 18; x++) {
34          for (y = 0; y < 9; y++)
35              for (z = 1; z < 9; z++)
36                  for (i = z + 1; i <= 9; i++)
37                      fprintf(fp, "%d %d 0\n", -(81 * x + 9 * y + z), -(81 * x + 9 * y + i));
38      }
39      //每一行1-9只出现一次
40      for (x = 0; x < 9; x++) {
41          for (z = 1; z <= 9; z++)
42              for (y = 0; y < 8; y++)
43                  for (i = y + 1; i <= 8; i++)
44                      fprintf(fp, "%d %d 0\n", -(81 * x + 9 * y + z), -(81 * x + 9 * i + z));
45      }
46
47      for (x = 9; x < 18; x++) {
48          for (z = 1; z <= 9; z++)
49              for (y = 0; y < 8; y++)
50                  for (i = y + 1; i <= 8; i++)
```

```
51                      fprintf(fp, "%d□%d□0\n", -(81 * x + 9 * y + z
                            ), -(81 * x + 9 * i + z));
52          }
53          //每一列1-9只出现一次
54          for (y = 0; y < 9; y++) {
55              for (z = 1; z <= 9; z++)
56                  for (x = 0; x < 8; x++)
57                      for (i = x + 1; i <= 8; i++)
58                          fprintf(fp, "%d□%d□0\n", -(81 * x + 9 * y + z
                                ), -(81 * i + 9 * y + z));
59          }
60
61          for (y = 0; y < 9; y++) {
62              for (z = 1; z <= 9; z++)
63                  for (x = 9; x < 17; x++)
64                      for (i = x + 1; i <= 17; i++)
65                          fprintf(fp, "%d□%d□0\n", -(81 * x + 9 * y + z
                                ), -(81 * i + 9 * y + z));
66          }
67          //每一宫1-9只出现一次
68          for (i = 0; i < 3; i++) {
69              for (j = 0; j < 3; j++)
70                  for (z = 1; z <= 9; z++)
71                      for (x = 0; x < 3; x++)
72                          for (y = 0; y < 3; y++)
73                              for (k = x + 1; k < 3; k++)
74                                  for (l = 0; l < 3; l++)
75                                      if (y != l)
76                                          fprintf(fp, "%d□%d□0\n", -(81
                                                * (3 * i + x) + 9 * (3 *
                                                j + y) + z),
77                                                  -(81 * (3 * i + k) + 9 *
                                                      (3 * j + l) + z));
78          }
79
```

```
80      for (i = 0; i < 3; i++) {
81          for (j = 0; j < 3; j++)
82              for (z = 1; z <= 9; z++)
83                  for (x = 0; x < 3; x++)
84                      for (y = 0; y < 3; y++)
85                          for (k = x + 1; k < 3; k++)
86                              for (l = 0; l < 3; l++)
87                                  if (y != l)
88                                      fprintf(fp, "%d %d 0\n",
                                            -(729 + 81 * (3 * i + x) +
                                             9 * (3 * j + y) + z),
89                                            -(729 + 81 * (3 * i + k)
                                                + 9 * (3 * j + l) + z)
                                                );
90      }
91      //每一格一定要有1-9中的一个数字
92      for (x = 0; x < 9; x++) {
93          for (y = 0; y < 9; y++) {
94              for (z = 1; z <= 9; z++)
95                  fprintf(fp, "%d ", 81 * x + 9 * y + z);
96              fprintf(fp, "0\n");
97          }
98      }
99
100     for (x = 9; x < 18; x++) {
101         for (y = 0; y < 9; y++) {
102             for (z = 1; z <= 9; z++)
103                 fprintf(fp, "%d ", 81 * x + 9 * y + z);
104             fprintf(fp, "0\n");
105         }
106     }
107     //双数独重叠部分等价关系
108     for (x = 6; x < 9; x++) {
109         for (y = 6; y < 9; y++)
110             for (z = 1; z <= 9; z++) {
```

```
111                 fprintf(fp, "%d□%d□0\n%d□%d□0\n", -(81 * x + 9 *
                        y + z), (729 + 81 * (x - 6) + 9 * (y - 6) + z)
                        ,
112                      (81 * x + 9 * y + z), -(729 + 81 * (x - 6) +
                            9 * (y - 6) + z));
113             }
114     }
115     fclose(fp);
116     fp = fopen("SudokuTableBase.cnf", "r");
117     return fp;
118 }
119
120 void swap(int& x, int& y) {
121     int tmp = x;
122     x = y;
123     y = tmp;
124     return;
125 }
126
127 //定义CreateSudoku函数, 创建数独终盘
128 status CreateSudoku() {
129     srand(int(time(0)));
130     int i, j, k;
131     for (i = 0; i < 9; i++) {
132         sudoku_table[0][i] = i + 1;
133     }
134     for (i = 0; i < 9; i++) {
135         j = rand() % 9;
136         if (i != j) {
137             swap(sudoku_table[0][i], sudoku_table[0][j]);
138         }
139     }
140     int move[] = { 0, 3, 6, 1, 4, 7, 2, 5, 8 };
141     for (j = 1; j < 9; j++) {
142         int tmp[9], tag = 0;
```

```
143             for (i = 8 - move[j] + 1; i <= 8; i++) {
144                 tmp[tag++] = sudoku_table[0][i];
145             }
146             for (i = 0; i <= 8 - move[j]; i++) {
147                 sudoku_table[j][i + move[j]] = sudoku_table[0][i];
148             }
149             for (i = 0; i < tag; i++) {
150                 sudoku_table[j][i] = tmp[i];
151             }
152         }
153         sudoku_table[9][0] = sudoku_table[6][6], sudoku_table[9][1] =
                sudoku_table[6][7], sudoku_table[9][2] = sudoku_table
               [6][8];
154         sudoku_table[9][3] = sudoku_table[8][6], sudoku_table[9][4] =
                sudoku_table[8][7], sudoku_table[9][5] = sudoku_table
               [8][8];
155         sudoku_table[9][6] = sudoku_table[7][6], sudoku_table[9][7] =
                sudoku_table[7][7], sudoku_table[9][8] = sudoku_table
               [7][8];
156         for (j = 10; j < 18; j++) {
157             int tmp[9], tag = 0;
158             for (i = 8 - move[j - 9] + 1; i <= 8; i++) {
159                 tmp[tag++] = sudoku_table[9][i];
160             }
161             for (i = 0; i <= 8 - move[j - 9]; i++) {
162                 sudoku_table[j][i + move[j - 9]] = sudoku_table[9][i
                        ];
163             }
164             for (i = 0; i < tag; i++) {
165                 sudoku_table[j][i] = tmp[i];
166             }
167         }
168         return OK;
169     }
170
```

```
171 extern int prehandle_stack[MAXN];
172 extern int pre_top, dec_top;
173
174 //定义DPLL2_SUDOKU函数, 调用SAT求解器
175 status DPLL2_SUDOKU(Conjunctive_Normal_Form_List cnf, int t[],
        int top) {
176     //     printf("%d", top);
177     pre_top = 0;
178     dec_top = 0;
179     int i;
180     for (i = 0; i < top; i++) {
181         prehandle_stack[pre_top++] = t[i];
182     }
183     //     printf("pre_top%d", pre_top);
184     //     printf("lllll%dllll", cnf->first_clause->watch_literal
            [0]);
185     return DPLL2(cnf);
186 }
187
188 //定义sudokusat函数, 求解数独问题
189 status sudokusat(int m, int n, Conjunctive_Normal_Form_List cnf)
        {
190     int x, y, z, t, i;
191     int tmp[2000], tmp0, ans = 0;
192     for (t = 1; t <= 9; t++) {
193         tmp0 = 0;
194         pre_top = 0, dec_top = 0;
195         for (x = 0; x < 18; x++)
196             for (y = 0; y < 9; y++) {
197                 if (users_sudoku[x][y] != 1) {
198                     for (z = 1; z <= 9; z++) {
199                         if (z == sudoku_table[x][y]) {
200                             tmp[tmp0++] = 81 * x + 9 * y + z;
201                         }
202                         else {
```

```
                                tmp[tmp0++] = -(81 * x + 9 * y + z);
                            }
                        }
                    }
                }
            for (z = 1; z <= 9; z++) {
                if (z == t) {
                    tmp[tmp0++] = 81 * m + 9 * n + z;
                }
                else {
                    tmp[tmp0++] = -(81 * m + 9 * n + z);
                }
            }
            ans += DPLL2_SUDOKU(cnf, tmp, tmp0);
            for (i = 1; i <= cnf->literal_num; i++) {
                ValueList[i].is_value = 0;
                ValueList[i].tag = 0;
                ValueList[i].dep = 0;
                ValueList[i].num = 0;
                ValueList[i].nev = 0;
                ValueList[i].pos = 0;
                ValueList[i].score = 0;
            }
            //          printf("%d ", ans);
            if (ans > 1) return FALSE;
        }
        return true;
}

//定义Dig_Hole_Easy函数，挖洞法生成数独
status Dig_Hole_Easy(int dig) {
    int dig_num = dig * 45;
    srand(int(time(0)));
    int x, y, ans, randnum, d, cnt = 1, i, b, z;
    Conjunctive_Normal_Form_List cnf;
```

```
238     FILE* fp = CreatePreSudokuFile(cnf);
239     CreateCNF2(fp, cnf);
240     for (x = 0; x < 18; x++)
241         for (y = 0; y < 9; y++)
242             users_sudoku[x][y] = 0;
243     for (i = 1; i <= 162; i++)
244         shuffle_value[i] = i;
245     for (i = 162; i > 2; i--) {
246         randnum = rand() % i + 1;
247         if (i != randnum) {
248             d = shuffle_value[i];
249             shuffle_value[i] = shuffle_value[randnum];
250             shuffle_value[randnum] = d;
251         }
252     }
253     for (i = 1; i <= dig_num && cnt < 162; i++) {
254         x = (shuffle_value[cnt] - 1) / 9;
255         y = (shuffle_value[cnt] - 1) % 9;
256         //          x = rand() % 18;
257         //          y = rand() % 9;
258         cnt++;
259         if (users_sudoku[x][y] != -1 && users_sudoku[x][y] != 1)
                {
260             users_sudoku[x][y] = 1;
261             if (x >= 6 && x <= 8 && y >= 6 && y <= 8) {
262                 users_sudoku[x + 3][y - 6] = 1;
263             }
264             else if (x >= 9 && x <= 11 && y >= 0 && y <= 2) {
265                 users_sudoku[x - 3][y + 6] = 1;
266             }
267             ans = sudokusat(x, y, cnf);
268 
269             if (!ans) {
270                 users_sudoku[x][y] = -1;
271                 if (x >= 6 && x <= 8 && y >= 6 && y <= 8) {
```

```
272                     users_sudoku[x + 3][y - 6] = -1;
273                 }
274                 else if (x >= 9 && x <= 11 && y >= 0 && y <= 2) {
275                     users_sudoku[x - 3][y + 6] = -1;
276                 }
277                 i--;
278                 continue;
279             }
280             continue;
281         }
282         i--;
283     }
284     print_time(cnf);
285     return OK;
286 }
287
288 //定义print_time函数，输出求解时间
289 int print_time(Conjunctive_Normal_Form_List cnf) {
290     clock_t start, end;
291
292     int x, y, z, t, i;
293     int tmp[2000], tmp0, ans = 0;
294     tmp0 = 0;
295     pre_top = 0, dec_top = 0;
296     for (x = 0; x < 18; x++)
297         for (y = 0; y < 9; y++) {
298             if (users_sudoku[x][y] != 1) {
299                 for (z = 1; z <= 9; z++) {
300                     if (z == sudoku_table[x][y]) {
301                         tmp[tmp0++] = 81 * x + 9 * y + z;
302                     }
303                     else {
304                         tmp[tmp0++] = -(81 * x + 9 * y + z);
305                     }
306                 }
```

```
307                 }
308             }
309         start = clock();
310         ans += DPLL2_SUDOKU(cnf, tmp, tmp0);
311         end = clock();
312         for (i = 1; i <= cnf->literal_num; i++) {
313             ValueList[i].is_value = 0;
314             ValueList[i].tag = 0;
315             ValueList[i].dep = 0;
316             ValueList[i].num = 0;
317             ValueList[i].nev = 0;
318             ValueList[i].pos = 0;
319             ValueList[i].score = 0;
320         }
321         printf("求解数独所用时间: %dms\n", end - start);
322         return true;
323     }
324
325     //定义print_time函数, 输出求解时间
326     status SudokuFinalPrint(void) {
327         int x, y;
328         for (x = 0; x < 18; x++) {
329             printf("\n");
330             for (y = 0; y < 9; y++) {
331                 if (users_sudoku[x][y] == 1)
332                     printf("0□");
333                 else
334                     printf("%d□", sudoku_table[x][y]);
335             }
336         }
337         return 0;
338     }
```

文件名：func_front.cpp

功能：程序前端交互

```
1  //
2  // Created by hp on 2022-09-09.
3  //
4
5  #include "define.h"
6  Conjunctive_Normal_Form_List cnf;//定义cnf
7  char filename[100];//定义文件名
8
9
10 extern int sudoku_table[18][9];//定义全局变量int类型二维数组存储
       双数独终盘
11 extern int users_sudoku[18][9];//存储输出的含空格数独问题格局
12 extern int shuffle_value[164];//存储被打乱的变元
13
14 extern ArgueValue* ValueList;//定义变元真值表
15
16 //定义solve函数，输出交互界面
17 void solve(int op) {
18     switch (op) {
19     case 1: solve_cnf(); break;
20     case 2: solve_sudoku(); break;
21     default:break;
22     }
23     return;
24 }
25
26 //定义solve_cnf函数，输出cnf文件求解交互界面
27 void solve_cnf() {
28     int create_flag = 0, d;
29     clock_t start = 0, finish = 0;//记录DPLL函数调用的起始和终止
           时间
30     int duration = 0;//记录SAT求解时间
31     system(("cls"));
32     printf("请输入想要求解的cnf文件的名称: ");
33     scanf("%s", filename);
```

```
34      FILE* fp;
35
36      if ((fp = fopen(filename, "r")) == NULL) {
37          printf("Fail to open file!\n");
38          exit(0);  //退出程序（结束程序）
39      }
40
41      int op1;
42
43      printf("请选择求解方案: \n");
44      printf("方案 0 : 未优化前顺序选取策略\n");
45      printf("方案 1 : 最优非递归求解策略\n");
46      printf("方案 2 : 递归最短子句优先策略\n");
47      printf("方案 3 : 递归最大频率优先策略\n");
48      printf("方案 4 : 递归VSIDS选取策略\n");
49
50      scanf("%d", &op1);
51
52      switch (op1) {
53      case 0:
54          create_flag = CreateCNF(fp, cnf);
55          if (!create_flag) {
56              printf("Fail to create cnf!");
57              exit(0);  //退出程序（结束程序）
58          }
59          start = clock();
60          d = DPLL1(cnf, 0);
61          //      check_print2(cnf);
62          //      print2(cnf);
63          finish = clock();
64          duration = (finish - start);
65
66          printf("求解文件用时%dms\n", duration);
67          if (d == 1) {
68              printf("该cnf文件有解! \n");
```

```
69          }
70          else
71              printf("该cnf文件无解！\n");
72          break;
73      case 1:
74          create_flag = CreateCNF2(fp, cnf);
75          if (!create_flag) {
76              printf("Fail to create cnf!");
77              exit(0);  //退出程序（结束程序）
78          }
79          start = clock();
80          d = DPLL2(cnf);
81          //     check_print2(cnf);
82          //     print2(cnf);
83          finish = clock();
84          duration = (finish - start);
85
86          printf("求解文件用时%dms\n", duration);
87          if (d == 1) {
88              printf("该cnf文件有解！\n");
89          }
90          else
91              printf("该cnf文件无解！\n");
92          break;
93      case 2:
94          create_flag = CreateCNF(fp, cnf);
95          if (!create_flag) {
96              printf("Fail to create cnf!");
97              exit(0);  //退出程序（结束程序）
98          }
99          start = clock();
100         d = DPLL1(cnf, 1);
101         //     check_print2(cnf);
102         //     print2(cnf);
103         finish = clock();
```

```
104         duration = (finish - start);
105
106         printf("求解文件用时%dms\n", duration);
107         if (d == 1) {
108             printf("该cnf文件有解！\n");
109         }
110         else
111             printf("该cnf文件无解！\n");
112         break;
113     case 3:
114         create_flag = CreateCNF(fp, cnf);
115         if (!create_flag) {
116             printf("Fail to create cnf!");
117             exit(0);  //退出程序（结束程序）
118         }
119         start = clock();
120         d = DPLL1(cnf, 2);
121         //     check_print2(cnf);
122         //     print2(cnf);
123         finish = clock();
124         duration = (finish - start);
125
126         printf("求解文件用时%dms\n", duration);
127         if (d == 1) {
128             printf("该cnf文件有解！\n");
129         }
130         else
131             printf("该cnf文件无解！\n");
132         break;
133     case 4:
134         create_flag = CreateCNF(fp, cnf);
135         if (!create_flag) {
136             printf("Fail to create cnf!");
137             exit(0);  //退出程序（结束程序）
138         }
```

```
139         start = clock();
140         d = DPLL1(cnf, 3);
141         //      check_print2(cnf);
142         //      print2(cnf);
143         finish = clock();
144         duration = (finish - start);
145 
146         printf("求解文件用时%dms\n", duration);
147         if (d == 1) {
148             printf("该cnf文件有解！\n");
149         }
150         else
151             printf("该cnf文件无解！\n");
152         break;
153     default:
154         printf("选取策略无效，请正确选取！\n");
155         break;
156     }
157     fclose(fp);
158     printf("请选择是否保存解文件【y/n】：");
159     char s[10];
160     scanf("%s", s);
161     if (s[0] == 'y') {
162         store_document(cnf, filename, d, duration);
163     }
164     int op2 = 1;
165     while (op2) {
166         system("cls");
167         printf("选择查看信息：\n");
168         printf("1.遍历cnf文件\n");
169         printf("2.验证解的正确性\n");
170         printf("3.查看解的结构\n");
171         printf("0.退出\n\n");
172 
173         printf("请选择想要查看的信息：");
```

```
174             scanf("%d", &op2);
175             switch (op2) {
176             case 1:traveser_cnf(cnf); getchar(); getchar(); break;
177             case 2:prove_cnf(cnf); getchar(); getchar(); break;
178             case 3:show_cnf(cnf); getchar(); getchar(); break;
179             case 0:break;
180             default:printf("请正确选择查看信息, 否则请按0退出");
                    getchar(); getchar(); break;
181             }
182         }
183         free(ValueList);
184         DestroyCNF(cnf);
185
186     }
187
188     //定义solve_sudoku函数, 输出数独求解交互界面
189     void solve_sudoku() {
190         char c, v;
191
192         initgraph(1024, 684);          // 创建绘图窗口, 大小为 640x480
                像素
193
194         settextstyle(16, 8, _T("Courier")); // 设置字体
195         // 设置颜色
196         settextcolor(BLACK);
197         setlinecolor(BLACK);
198         setlinestyle(PS_SOLID, 2);
199         fillrectangle(0, 0, 1024, 700);
200
201         setfillcolor(WHITE);
202         setlinecolor(BLACK);
203
204         setbkcolor(WHITE);
205
206         IMAGE picture;
```

```
207
208     loadimage(&picture, "c.jpg", 1024, 700);
209
210     putimage(0, 0, &picture);
211
212     loadimage(&picture, "d.jpg", 250, 254);
213
214     putimage(774, 446, &picture);
215     //欢迎进入数据结构的数独世界
216     settextstyle(45, 20, _T("微软雅黑"));
217     outtextxy(230, 300, _T("欢迎进入数据结构的数独世界□"));
218
219     wchar_t str0[] = L"□□请按ENTER键开始";
220     settextstyle(45, 20, _T("微软雅黑"));
221     outtextxy(330, 400, _T("□请按ENTER键开始"));
222
223     char s0;
224
225     while (TRUE) {
226         s0 = getch();
227         if (s0 == 13)
228             break;
229     }
230     fillrectangle(0, 0, 1024, 700);
231
232     settextstyle(50, 25, _T("微软雅黑"));
233     outtextxy(185, 280, _T("请选择难度: "));
234     settextstyle(40, 18, _T("楷体"));
235     outtextxy(525, 150, _T("简单（请按数字盘□1） "));
236     outtextxy(525, 300, _T("中等（请按数字盘□2） "));
237     outtextxy(525, 450, _T("复杂（请按数字盘□3） "));
238
239     int dig_slt = 0;
240
241     while (TRUE) {
```

```
242         s0 = getch();
243         if (s0 == 49) {
244             dig_slt = 1;
245             outtextxy(325, 550, _T("正在加载中……"));
246             break;
247         }
248         if (s0 == 50) {
249             dig_slt = 2;
250             outtextxy(325, 550, _T("正在加载中……"));
251             break;
252         }
253         if (s0 == 51) {
254             dig_slt = 3;
255             outtextxy(325, 550, _T("正在加载中……"));
256             break;
257         }
258     }
259
260     CreateSudoku();
261     Dig_Hole_Easy(dig_slt);
262
263     fillrectangle(0, 0, 1024, 700);
264     TCHAR o = _T('□');
265     for (int i = 1; i <= 9; i++)
266         for (int j = 1; j <= 9; j++) {
267             if (users_sudoku[i - 1][j - 1] != 1) {
268                 setfillcolor(YELLOW);
269                 fillrectangle(10 + 35 * j, 10 + 35 * i, 45 + 35 *
                        j, 45 + 35 * i);
270             }
271             else {
272                 setfillcolor(WHITE);
273                 fillrectangle(10 + 35 * j, 10 + 35 * i, 45 + 35 *
                        j, 45 + 35 * i);
274             }
```

```
275
276             RECT r = { 10 + 35 * j, 10 + 35 * i, 45 + 35 * j, 45
                    + 35 * i };
277             TCHAR s[5];
278         }
279     for (int i = 1; i <= 9; i++)
280         for (int j = 1; j <= 9; j++) {
281             if (users_sudoku[i + 8][j - 1] != 1) {
282                 setfillcolor(YELLOW);
283                 fillrectangle(10 + 35 * (6 + j), 10 + 35 * (6 + i
                        ), 45 + 35 * (6 + j), 45 + 35 * (6 + i));
284             }
285             else {
286                 setfillcolor(WHITE);
287                 fillrectangle(10 + 35 * (6 + j), 10 + 35 * (6 + i
                        ), 45 + 35 * (6 + j), 45 + 35 * (6 + i));
288             }
289             TCHAR s[5];
290
291             _stprintf(s, _T("%d"), sudoku_table[i + 8][j - 1]);
                            // 高版本 VC 推荐使用 _stprintf_s 函数
292             settextstyle(29, 21, s);
293             outtextxy(16 + 35 * (6 + i), 13 + 35 * (6 + j), s);
294             outtextxy(16 + 35 * (6 + j), 13 + 35 * (6 + i), s);
295         }
296
297     TCHAR s[50];
298     clock_t end;
299     clock_t start, finish;
300     int duration;
301     for (int i = 1; i <= 9; i++)
302         for (int j = 1; j <= 9; j++) {
303             _stprintf(s, _T("%d"), sudoku_table[i - 1][j - 1]);
                            // 高版本 VC 推荐使用 _stprintf_s 函数
304             settextstyle(29, 21, s);
```

```
305
306             if (users_sudoku[i - 1][j - 1] == 1)
307                 outtextxy(16 + 35 * j, 13 + 35 * i, o);
308             else {
309                 outtextxy(16 + 35 * j, 13 + 35 * i, s);
310             }
311
312         }
313     for (int i = 1; i <= 9; i++)
314         for (int j = 1; j <= 9; j++) {
315
316             _stprintf(s, _T("%d"), sudoku_table[i + 8][j - 1]);
                        // 高版本 VC 推荐使用 _stprintf_s 函数
317             settextstyle(29, 21, s);
318
319             if (users_sudoku[i + 8][j - 1] == 1)
320                 outtextxy(16 + 35 * (6 + j), 13 + 35 * (6 + i), o
                    );
321             else {
322                 outtextxy(16 + 35 * (6 + j), 13 + 35 * (6 + i), s
                    );
323             }
324         }
325     wchar_t str1[] = L"□□SUDOKU";
326     fillrectangle(710, 15, 900, 150);
327     settextstyle(18, 16, s);
328     outtextxy(712, 32, _T("□双数独游戏"));
329     int x, y, l, r, a = 0, b = 0;
330     ExMessage m, n;                 // 定义消息变量
331     settextcolor(BLACK);
332     start = clock();
333     end = clock();
334     _stprintf(s, _T("%d"), 0);              // 高版本 VC 推荐使用
        _stprintf_s 函数
335     settextstyle(29, 21, s);
```

```
336     outtextxy(832, 110, s);
337     wchar_t str2[] = L"□□错误次数:";
338     settextstyle(15, 12, s);
339     outtextxy(712, 250, _T("□游戏说明: "));
340     outtextxy(610, 275, _T("□请点击对应空格准备填入数字; "));
341     outtextxy(610, 325, _T("□数字请通过数字键盘输入; "));
342     outtextxy(610, 350, _T("□输入错误时显示W(WRONG); "));
343     outtextxy(610, 375, _T("□按□ESC□键退出; "));
344     outtextxy(610, 400, _T("□请关注自己的错误次数, "));
345     outtextxy(610, 425, _T("□超过15次会自动退出。"));
346     outtextxy(610, 450, _T("□祝你游戏愉快!!! "));
347     settextstyle(29, 21, s);
348     o = _T('W');
349     settextcolor(RED);
350     int wrong_times = 0, flag = 0;
351     while (true)
352     {
353         // 获取一条鼠标或按键消息
354         m = getmessage(EX_MOUSE | EX_KEY);
355         switch (m.message)
356         {
357         case WM_LBUTTONDOWN:
358             a = m.x, b = m.y;
359             if (a > 45 && a < 360 && b > 45 && b < 360) {
360                 x = 0, y = 0;
361                 l = 0, r = 0;
362                 while (x < a) {
363                     l++;
364                     x = 10 + 35 * l;
365                 }
366                 while (y < b) {
367                     r++;
368                     y = 10 + 35 * r;
369                 }
370                 if (users_sudoku[r - 2][l - 2] != 1)
```

```
371                     break;
372                 if (m.ctrl)
373                     // 画一个大方块
374                     rectangle(m.x - 5, m.y - 5, m.x + 5, m.y + 5)
                            ;
375                 else
376                     // 画一个小方块
377                     rectangle(m.x - 2, m.y - 2, m.x + 2, m.y + 2)
                            ;
378             }
379             if (a > 255 && a < 360 && b > 255 && b < 360) {
380                 break;
381             }
382             if (a > 255 && a < 570 && b > 255 && b < 570) {
383                 x = 0, y = 0;
384                 l = 6, r = 6;
385                 while (x < a) {
386                     l++;
387                     x = 10 + 35 * l;
388                 }
389                 while (y < b) {
390                     r++;
391                     y = 10 + 35 * r;
392                 }
393                 if (users_sudoku[r - 2 - 6 + 9][l - 2 - 6] != 1)
394                     break;
395                 if (m.ctrl)
396                     // 画一个大方块
397                     rectangle(m.x - 5, m.y - 5, m.x + 5, m.y + 5)
                            ;
398                 else
399                     // 画一个小方块
400                     rectangle(m.x - 2, m.y - 2, m.x + 2, m.y + 2)
                            ;
401             }
```

```
402             break;
403
404         case WM_KEYDOWN:
405             if (m.vkcode == VK_ESCAPE) {
406                 flag = 1;
407                 break;
408             }
409
410             if (a > 45 && a < 360 && b > 45 && b < 360) {
411                 x = 0, y = 0;
412                 l = 0, r = 0;
413                 while (x < a) {
414                     l++;
415                     x = 10 + 35 * l;
416                 }
417                 while (y < b) {
418                     r++;
419                     y = 10 + 35 * r;
420                 }
421                 c = sudoku_table[r - 2][l - 2] + 48;
422                 v = _getch();
423                 if (v == c) {
424                     outtextxy(16 + 35 * (l - 1), 13 + 35 * (r -
                            1), c);
425                 }
426                 else {
427                     outtextxy(16 + 35 * (l - 1), 13 + 35 * (r -
                            1), o);
428                     wrong_times++;
429                     _stprintf(s, _T("%d"), wrong_times);
                                    // 高版本 VC 推荐使用 _stprintf_s
                            函数
430                     outtextxy(832, 110, s);
431                 }
432             }
```

```
433             if (a > 255 && a < 360 && b > 255 && b < 360) {
434                 break;
435             }
436             if (a > 255 && a < 570 && b > 255 && b < 570) {
437                 x = 0, y = 0;
438                 l = 6, r = 6;
439                 while (x < a) {
440                     l++;
441                     x = 10 + 35 * l;
442                 }
443                 while (y < b) {
444                     r++;
445                     y = 10 + 35 * r;
446                 }
447                 c = sudoku_table[r - 2 - 6 + 9][l - 2 - 6] + 48;
448                 v = _getch();
449                 if (v == c) {
450                     outtextxy(16 + 35 * (l - 1), 13 + 35 * (r -
                            1), c);
451                 }
452                 else {
453                     outtextxy(16 + 35 * (l - 1), 13 + 35 * (r -
                            1), o);
454                     wrong_times++;
455                     _stprintf(s, _T("%d"), wrong_times);
                                    // 高版本 VC 推荐使用 _stprintf_s
                            函数
456                     outtextxy(832, 110, s);
457                 }
458             }
459             break;
460         }
461         if (flag) break;
462         if (wrong_times > 15) break;
463     }
```

```
464     _getch();                                   // 按任意键继续
465     closegraph();                               // 关闭绘图窗口
466 }
```

文件名：main.cpp

功能：主函数

```
1  #include "define.h"
2
3  /*定义有效的全局变量*/
4  //char filename[100];//定义文件名
5  //Conjunctive_Normal_Form_List cnf;//定义cnf
6  extern ArgueValue* ValueList;//定义变元真值表
7  int op;
8
9  extern int sudoku_table[18][9];//定义全局变量int类型二维数组存储
       双数独终盘
10 extern int users_sudoku[18][9];//存储输出的含空格数独问题格局
11 extern int shuffle_value[164];//存储被打乱的变元
12 using namespace std;
13
14 int main() {
15
16     while (true) {
17         system("cls");
18         printf("可选功能：\n");
19         printf("1.cnf文件求解\n");
20         printf("2.SUDOKU游戏\n");
21         printf("0.退出\n\n");
22         printf("请选择你想要实现的功能：");
23         scanf("%d", &op);
24         if (op == 0)
25             break;
26         solve(op);
27         getchar(); getchar();
28     }
```

```
29
30      return 0;
31  }
```