

华中科技大学

课程设计报告

题目: 基于 SAT 的二进制数独游戏求解程序

课程名称: 程序设计综合课程设计

专业班级: CS1906

学 号: U201915115

姓 名: 郑舟

指导教师: 纪俊文

报告日期: 2021.3.14

计算机科学与技术学院

任务书

设计内容

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

设计要求

要求具有如下功能：

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

(6) **SAT 应用:** 将二进制数独游戏^[5, 6]问题转化为 SAT 问题^[6], 并集成到上面的求解器进行问题求解, 游戏可玩, 具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献[3]与[6-9]。(15%)

参考文献

- [1] 张健著. 逻辑公式的可满足性判定一方法、工具及应用. 科学出版社, 2000
- [2] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Masterthesis, Concordia University, Canada, 2009
- [3] 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 硕士学位论文, 电子科技大学, 2011
- [4] Carsten Sinz. Visualizing SAT Instances and Runsof the DPLL Algorithm. JAutom Reasoning (2007) 39:219–243
- [5] Binary Puzzle: <http://www.binarypuzzle.com/>
- [6] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Mathematics in Computer Science, (2017) 11:515–526
- [7] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.
- [8] Ins Lynce and 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.
- [9] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57
- [10] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf

目录

1 引言	1
1.1 课题背景与意义	1
1.1.1 SAT 问题简介	1
1.1.2 二进制数独游戏简介	1
1.2 国内外研究现状	2
1.3 课程设计的主要研究工作	3
2 系统需求分析与总体设计	4
2.1 系统需求分析	4
2.2 系统总体设计	4
3 系统详细设计	6
3.1 有关数据结构的定义	6
3.2 主要算法设计	7
4 系统实现与测试	10
4.1 系统实现	10
4.2 系统测试	16
4.2.1 SAT 求解模块测试	16
4.2.1 数独模块测试	21
5 总结与展望	25
5.1 全文总结	25
5.1 工作展望	25
6 体会	27
参考文献	28
附录	错误!未定义书签。

1 引言

1.1 课题背景与意义

SAT 问题即命题逻辑公式的可满足性问题 (satisfiability problem)，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。SAT 问题也是程序设计与竞赛的经典问题。

1.1.1 SAT 问题简介

SAT 的问题被证明是 NP 难解的问题。已知的 NP-complete 问题多达几百个，但作为这些问题的“祖先”，历史上第一个被证明的 NP-complete 问题是来自于布尔逻辑的可满足性问题 (SATISFIABILITY problem)，简称为 SAT。

SAT 问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可转化成 SAT 问题。因此致力于寻找求解 SAT 问题的快速而有效的算法，不仅在理论研究上而且在许多应用领域都具有极其重要的意义。

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

1.1.2 二进制数独游戏简介

普通数独游戏是一种运用纸、笔进行演算的逻辑游戏。玩家需要根据 9×9 盘面上的已知数字，推理出所有剩余空格的数字，并满足每一行，每一列，每一粗线格内的数字均含 1-9，且不能重复。每一道合格的数独谜题都有且仅有唯一的答案，推理方法也以此为基础，任何无解或多解的题目都不合格。

与普通数独游戏(Sudoku)相似, n 阶(偶数 $n=2m \geq 4$)二进制数独游戏(Binary Puzzle)要求在 $n \times n$ 的网格中每个单元(cell)填入一个数字 1 或 0, 必须满足约束: (1) 在每一行、每一列中不允许有连续的 3 个 1 或 3 个 0 出现; (2) 在每一行、每一列中 1 与 0 的个数相同; (3) 不存在重复的行与重复的列。

1.2 国内外研究现状

Bart Selman 和 Henry Kautz 分别于 1997 年和 2003 年在人工智能第五届国际合作会议上提出了 SAT 问题面临的十大挑战性问题, 并在 2001 年和 2007 年先后对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。SAT 解决器的实现是我们关心的主要问题, 目前的 SAT 算法大致可以归结为两大类: 完备算法(也称回溯搜索算法)和局部搜索算法。其中, 完备算法大都是基于回溯搜索的, 局部搜索算法是基于局部随机搜索的。完备算法基于穷举法思想, 它的优点是能保证找到对应 SAT 问题的解或证明公式不可满足, 但是效率极低, 它的平均时间复杂度虽是多项式级的, 但是最坏情况下的时间复杂度却是指数级的。一些完备算法采用了精巧的技术来减小搜索空间和问题规模, 提高了算法的时间效率, 相对于完备算法而言, 由于采用了启发式策略来指导搜索, 使得求解速度相对较快, 但是在某些实例上可能得不到解, 它不保证一定能够找到对应 SAT 问题的解, 即它不能证明 SAT 问题的不可满足性。局部搜索算法的研究热潮是在最近几年才兴起的。

最经典的求解 SAT 问题的完备算法是 DPLL 算法, 它是由 Davis 和 Putnam 等人在 1960 年提出, 其它的完备算法大都是在 DPLL 算法的基础上衍生出来的, 对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂度是指数级别, 最初这使得许多的研究者望而却步。而后, S.A.Cook 在 1971 年证明了 SAT 问题是 NP 完全问题, 这更加削弱了许多学者研究 SAT 问题的兴趣, 从而导致了 SAT 问题在很长的一段时间里都没有得到较好的重视, 发展非常缓慢, 研究成果较少。但是 1996 年以后, 很多国家都相继举办了一些 SAT 竞赛和研讨会, 这使得越来越多的人开始关注并研究 SAT 问题, 所以这段时间也涌现出了众多新的高效的 SAT 算法如 MINISAT、SATO、CHAFF、POSIT 和

GRASP 等, SAT 算法的研究成果显著, 求解算法也越来越多地应用到了实际问题领域。这些新兴的算法大都是基于 DPLL 算法的改进算法, 改进的方面包括: 采用新的数据结构、新的变量决策策略或者新的快速的算法实现方案。国内也涌现出了许多高效的求解算法, 如 1998 年作者梁东敏提出了改进的子句加权 WSAT 算法, 2000 年金人超和黄文奇提出的并行 Solar 算法, 2002 年作者张德富在文献中, 提出模拟退火算法。

1.3 课程设计的主要研究工作

本次课程设计要求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, 基于 DPLL 过程实现一个高效 SAT 求解器, 对于给定的中小规模算例进行求解, 输出求解结果, 统计求解时间。在此基础上, 通过改进算法、使用不同的选择策略或物理存储结构来优化求解过程, 使求解器能更快地求解。完成基于此算法的二进制数独游戏, 通过将二进制数独归约成 SAT 问题进行求解。

2 系统需求分析与总体设计

2.1 系统需求分析

本设计要求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，基于 DPLL 过程实现一个高效 SAT 求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。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。

(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。

(3)DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。

(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)程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现，提供明确的性能优化率结果。优化率的计算公式为： $[(t-t_0)/t]*100\%$ ，其中 t 为未对 DPLL 优化时求解基准算例的执行时间， t_0 则为优化 DPLL 实现时求解同一算例的执行时间。

(6) SAT 应用：将二进制数独游戏问题转化为 SAT 问题，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的交互性。

2.2 系统总体设计

系统主要由两个模块组成：SAT 问题的求解和数独模块

(1)SAT 求解模块

包括从 CNF 表读取 CNF 范式并用邻接表存储，查看读入的 CNF 表是否正确，根据未优化变元选择策略的 DPLL 算法或者优化了变元选择策略的 DPLL 算法求解 SAT 问题，计算优化率，以及输出解文件等功能。

(2)数独模块

包括生成数独终局，挖洞法生成数独题面，用户填写数独答案，验证用户答案是否正确，以及输出参考答案等功能。

系统模块结构图如图 2-1。

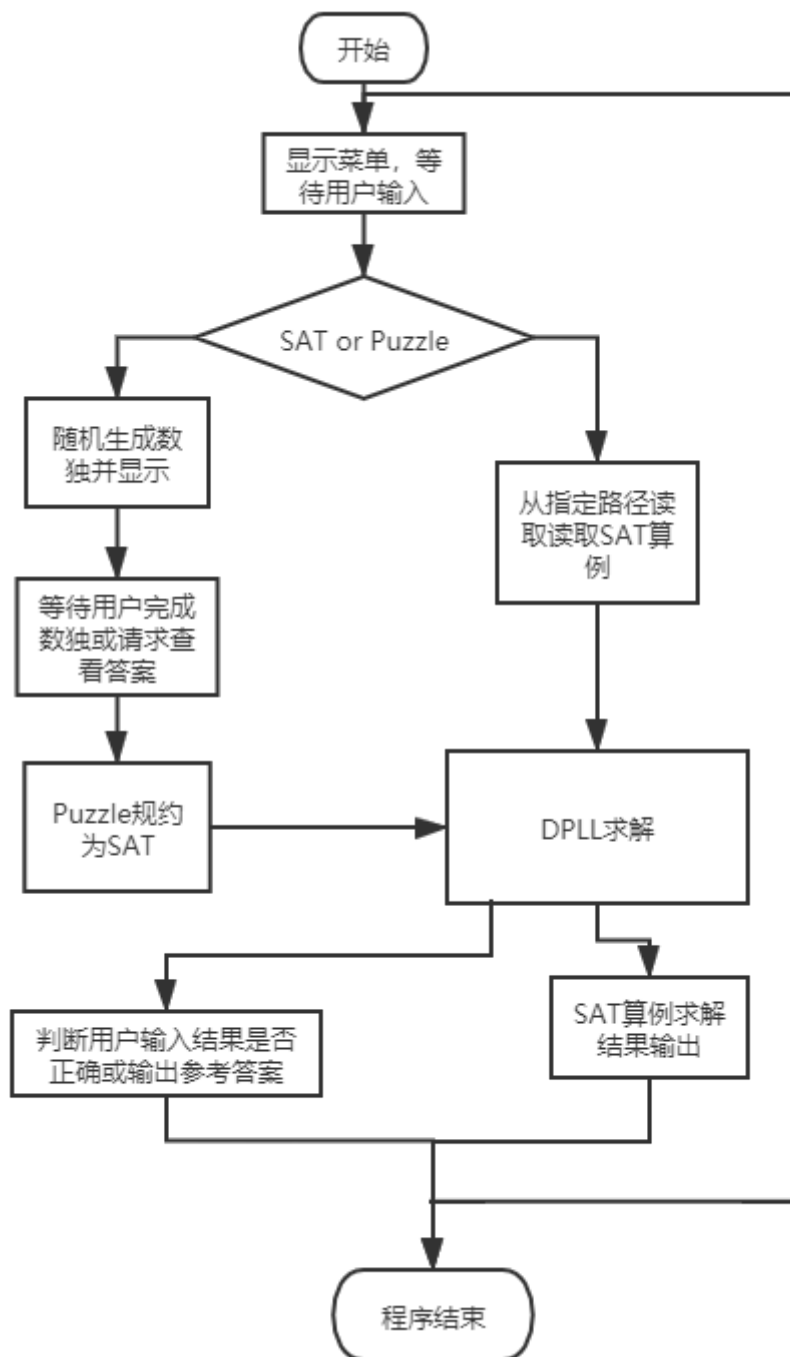


图 2-1：系统模块结构

3 系统详细设计

3.1 有关数据结构的定义

本实验要处理的数据主要为 CNF 公式，包括变元数量，子句数量，每个子句以及中的文字。为了方便操作，还记录了 CNF 公式中单子句的数量以及每一个子句中变元的数量。在优化变元选取策略时，我们还需要一个数据结构来储存每个文字的权重。综上，定义了以下数据类型与全局变量：

(1)记录文字的结构体 `node`，成员包括文字的值 `id` 以及指向下一个结构体的指针 `next`。

(2)记录子句的结构体 `clause`，成员包括指向存有该子句所有文字的链表的表头指针 `clausehead`，指向下一个子句的指针 `next`。

(3)记录 CNF 公式中变元总数与子句总数的全局变量 `argumentsCounts` 以及 `clausesCounts`。

(5)记录答案用的数组 `ans`，其下标代表变元序号，值为 1 或 0，为 1 时该变元取值为真，为 0 时为假。

(6)记录文字权重的数组 `ranks`，其下标小于等于 `argumentsCounts` 的部分记录正文字权重，大于 `argumentsCounts` 小于等于两倍 `argumentsCounts` 的部分记录负文字的权重。负文字所对应的带符号变元序号等于 `argumentsCounts` 减去其下标。

CNF 公式的存储结构如图 3-1。其中第一个子句节点是子句链表表头节点，其只有一个文字节点，该文字节点中的 `id` 存放的是整个 CNF 公式所含的单子句的数目，`next` 指向 NULL；其他的子句节点则代表一个子句，每个子句节点中 `clausehead` 向存有该子句所有文字的链表的表头。文字链表的表头节点的 `id` 存放的是该子句所含文字数目，`next` 指向该子句第一个变元所在的节点，其他节点的 `id` 则存放文字，`next` 指向下一个变元所在的节点。

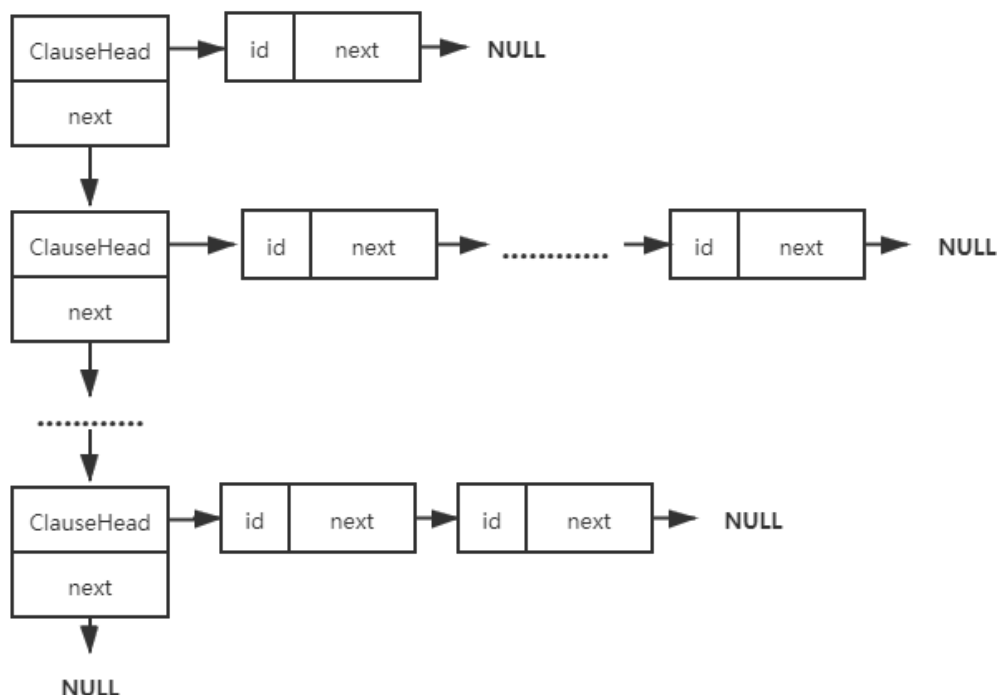


图 3-1：CNF 公式存储结构

3.2 主要算法设计

(1) SAT 求解

SAT 求解部分的核心是 DPLL 算法的实现，DPLL 算法的大致步骤为：先寻找单子句，记单子句中文字为 f 为真，然后删除所有包含文字 f 的子句，删除所有包含 $\neg f$ 的子句中的 $\neg f$ 项，操作完成后若 CNF 公式不含任何子句，则说明公式可满足，若存在空子句则说明公式不可满足，若产生了新的单子句则重复上述过程，若没有单子句，则选择一个变元 v ，令其为真，将其作为单子句插入 CNF 公式中，递归调用 DPLL 算法，若在此情况下公式能满足则说明在 v 为真时公式能满足，否则令 v 为假，将其作为单子句插入 CNF 公式，递归调用 DPLL 算法，若在此情况下能满足则说明 v 为假时公式可满足，否则说明公式不可满足。流程图如图 3-2。

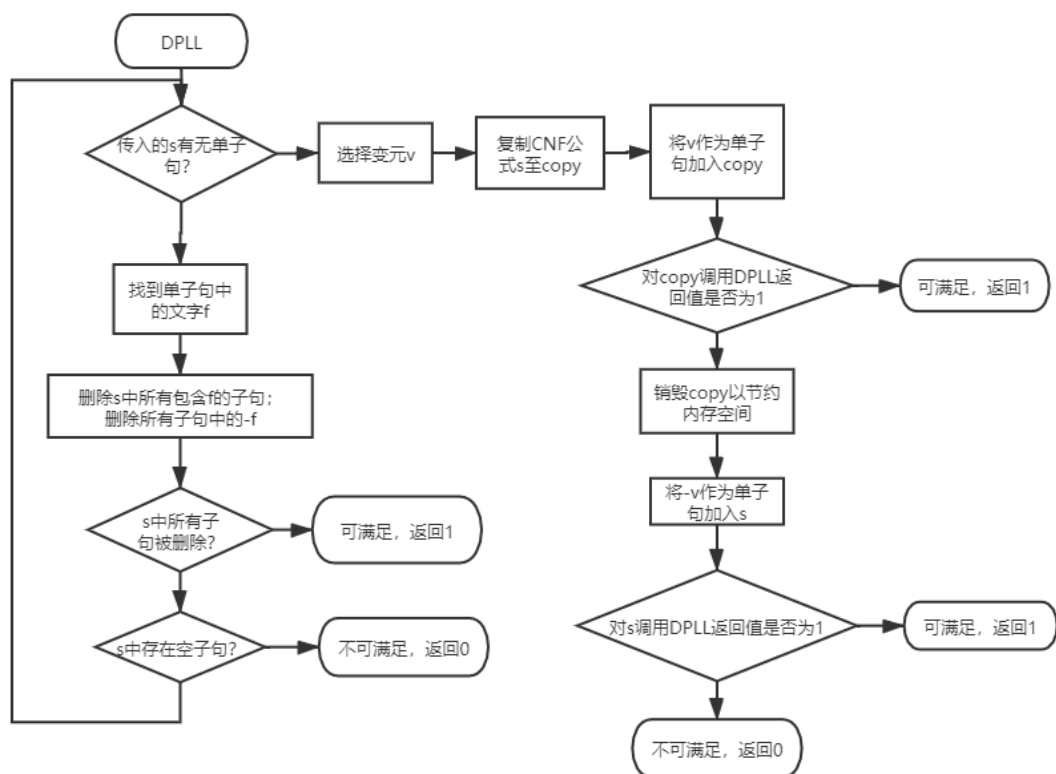


图 3-2: DPLL 算法

(2)数独生成

本实验只实现了六阶数独，生成数独时首先随机将两个不相同的格子填为 1，然后根据数独的规则生成一个合法的数独终盘，该数独终盘将作为参考答案，然后随机挖掉 15 至 24 个格子，生成题面，之后玩家便可以开始解题，玩家在解题过程中可以随时查看参考答案，查看参考答案之后便不可再玩，玩家填满所有格子后会根据规则检验玩家答案是否满足，满足则输出提示信息后游戏结束，否则提示答案有误还可以继续玩。

对于数独的三个规则，对于第一个规则，没有三个连续的 0 或 1，可以直接穷举得到 96 个子句。

对于第二个规则，每一行，每一列的 0 与 1 个数相等。考虑某一行或列数目不等时，将会出现四个及以上的 0 或 1，所以该条件等价于每一行，每一列任意 4 个变元不能全部为 0 或 1，于是可以穷举得到子句 360 个。

对于第三个规则，以第五行与第六行为例，即要求满足 $\neg\{[(51 \wedge 61) \vee (\neg 51 \wedge \neg 61)] \wedge [(52 \wedge 62) \vee (\neg 52 \wedge \neg 62)] \wedge \dots \wedge [(55 \wedge 65) \vee (\neg 56 \wedge \neg 66)]\}$ ，将其展开转化为 CNF，可转化为一个每个子句包含 12 个变元，包含 $2^6=64$ 个子

句的 CNF 公式。根据这个思想,为确保每行每列都不相同可以写入 $2 \times C_6^2 \times 64 = 1920$ 个子句。

综上,一个六阶二进制数独转化为 CNF 个范式时,变元数即为 36 个,子句数为初始条件构成的单子句数加上 $96+360+1920=2376$ 。数独生成以及与用户交互的流程如图 3-3。

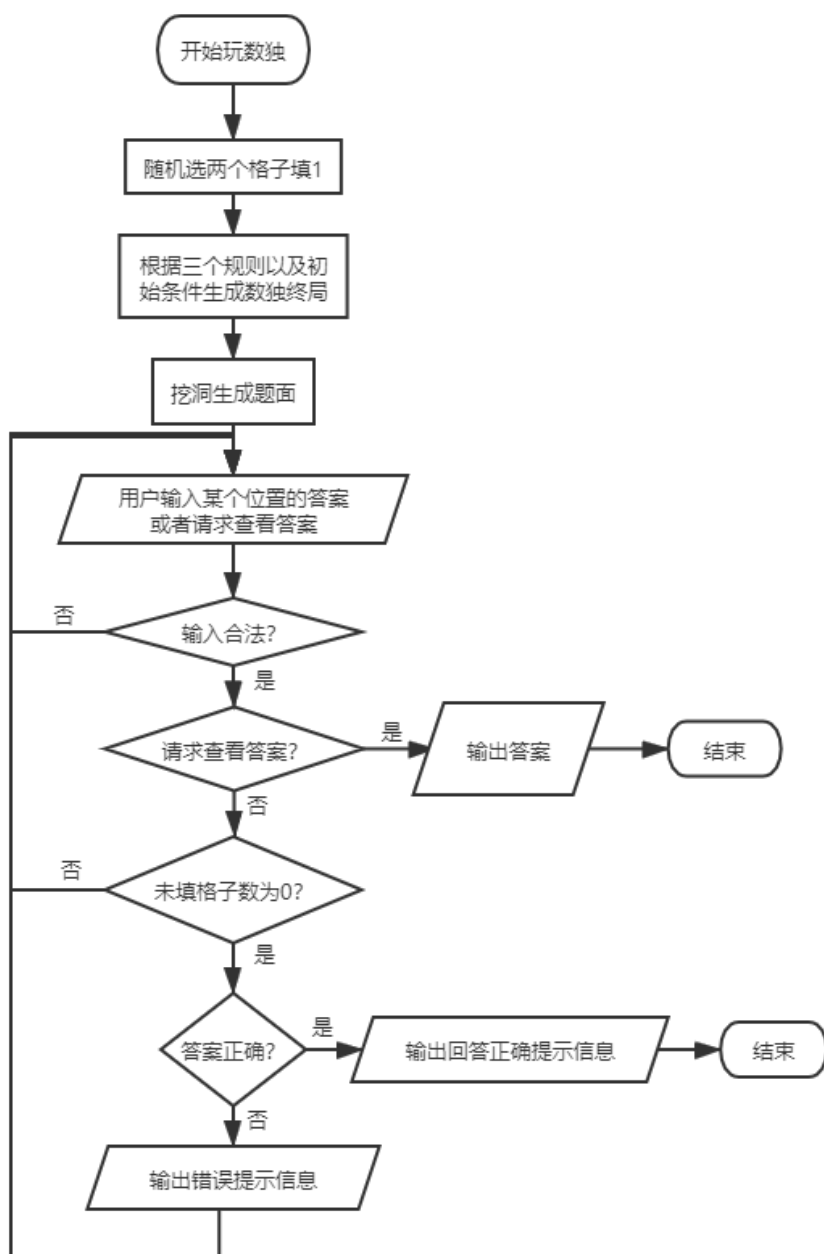


图 3-3: 交互玩数独

4 系统实现与测试

4.1 系统实现

1. 系统环境:

CPU: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz 内存: 16GB 编译器版本: gcc version 8.1.0

2. 自定义数据类型及全局变量

文字节点定义:

```
struct node
{
    int id;
    struct node *next;
};
```

子句节点定义:

```
struct clause
{
    struct node *clauseHead;
    struct clause *next;
};
```

用于存储答案的数组: int ans[5000];

用于存储文字权重的数组: int ans[5000];

用于存储变元数和子句数的数组: int argumentsCounts, clausesCounts;

3. 函数说明

(1) CNF 处理模块

int creatClause(char *filePath, struct clause **head): 该函数用于通过 filePath 指向的文件生成 CNF 公式并使 CNF 公式被 head 指向, 创建成功返回 1, 否则返回 0。函数首先尝试打开文件, 然后找到非注释部分, 读取变元数和子句数, 然后开始创建第一条子句。创建子句时每在子句中新增一个文字, 作为表头节点的

id 加一。当读取到 0 后表示该子句已读完，开始创建下一条子句，直到创建的子句数等于一开始读入的子句数。

void destroyClause(struct clause *t): 该函数用于销毁 t 指向的子句。函数先遍历并销毁所有文字节点，最后销毁 t 指向的子句节点。

int findUnitClause(struct clause *t): 该函数用于寻找 t 指向的 CNF 公式中的单子句，并返回单子句中的文字。

void addClause(struct clause *s, int x): 该函数用于向 s 指向的 CNF 公式中添加一个仅含文字 x 的单子句。

int copyCNF(struct clause *origin, struct clause **replication): 该函数用于将 origin 指向的 CNF 公式复制到 replication 指向的指针指向的 CNF 公式中以备回溯。特别的是，为了编写程序的方便，该函数在赋值时将原 CNF 公式中排在前面的子句放在了后面，将原子句中排在前面的文字放在了后面，所以复制得到的 CNF 公式与原 CNF 公式并不完全相同。

void destroyCNF(struct clause *s): 该函数用于销毁 s 指向的 CNF 公式，以在递归过程中节约内存空间。该函数主要调用了 destroyClause 函数，依次遍历并销毁每一条子句。

int dealWithClauseInOneLoop(struct clause *s, int x): 该函数用于在 DPLL 过程中处理子句，即删除与 x 有关的子句，删除与 -x 有关的子句中的 -x 文字，为节约运行时间，这两项操作将在一次对 CNF 公式中所有子句的所有文字的遍历中完成。在遍历过程中，若出现空子句则返回 1，说明此时 CNF 公式无法满足，否则返回 1。

int CalculateWeight(struct clause *s): 该函数用于根据最短子句出现频率最大优先规则计算 s 指向的 CNF 公式中每个文字的权重并记录在全局变量 ranks 中，假设共有 m 个子句， m 个子句中第 i 个子句的长度为 n_i ，则文字 l 的权重的规则为 $J(l) = \sum_{l \in C_i} 2^{-n_i}$ ，($i = 1, 2, 3 \dots m$)，即假设包含文字 1 的子句有三个，长度分别为 2, 3, 4，则文字 1 的权重为 $\frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} = 0.4375$ ，这样做是因为子句越短越难满足，子句越长越容易满足，在这些文字数较少的子句中挑出出现频率较大的文字使其优先满足，让那些不易满足的子句得到优先满足，使搜索过程可以尽快找到可满足解。存储权重的方式如 3.1 中所述。函数将返回权重最大的文字。

`void printAnswerToRes(const char filename[], int f, long t)`: 该函数用于输出解文件指定 res 文件中，具体在调用时 filename 传入的值为.cnf 文件的路径，f 表示是否有解，为 1 则有解，0 则无解，t 储存着运行时间。具体的解储存在全局变量 ans 中。函数将按照任务书的规则将 f, ans, t 写入与 filename 同名但后缀为 res 的文件中。

(2)DPLL 模块

`int DPLL(struct clause *s)`: 该函数用于对 s 指向的 CNF 公式进行 DPLL 求解。DPLL 算法流程图如图 3-2。该函数主要调用了 CNF 处理模块中的函数，调用情况如图 4-1 所示。

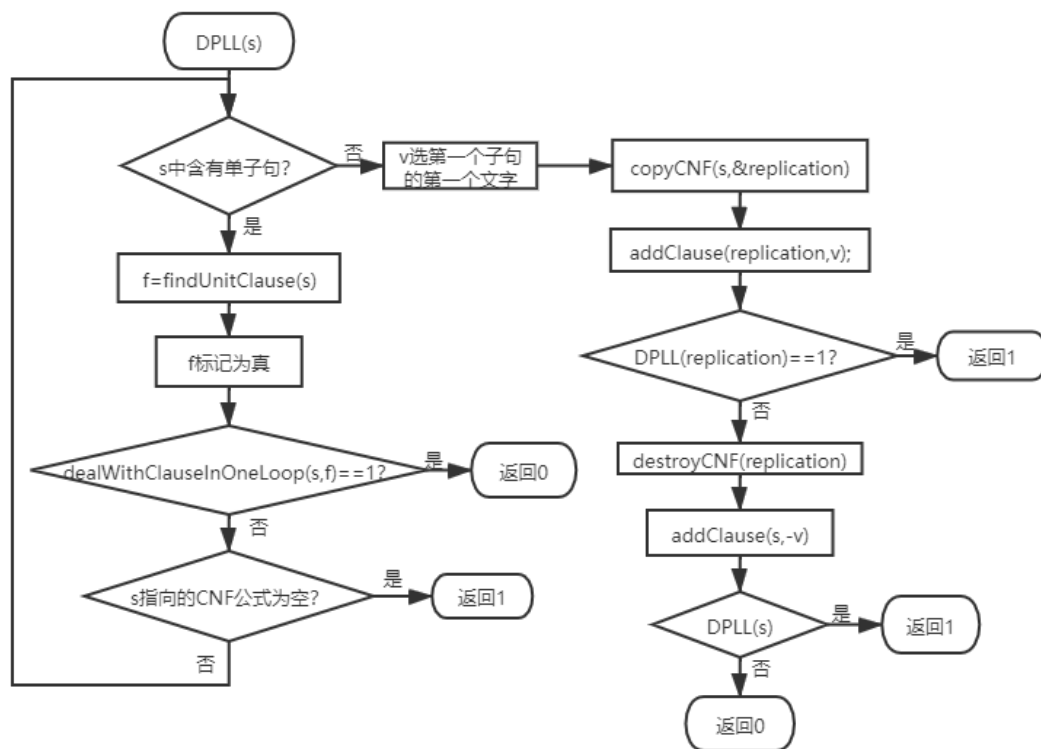


图 4-1：DPLL 对 CNF 处理模块的函数调用情况

`int betterDPLL(struct clause *s)`: 该函数为优化变元选取策略后的 DPLL 算法，流程与第一个函数基本一致，对函数的调用情况如图 4-2 所示。

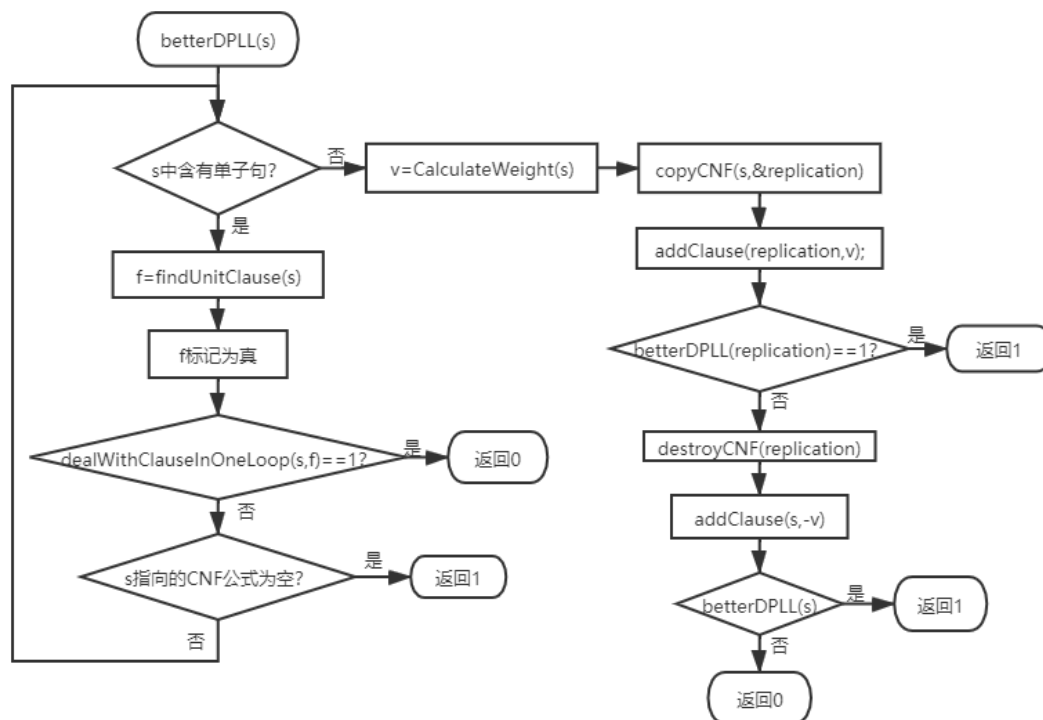


图 4-2: betterDPLL 对 CNF 处理模块的函数调用情况

(3)数独模块

`int rule_1(char filename[80])`: 该函数用于向 filename 指向的文件中写入二进制数独第一个规则，即每一行每一列没有连续的三个 0 或 1，枚举写入 96 个子句。

`int rule_2(char filename[80])`: 该函数用于向 filename 指向的文件中写入二进制数独的第二个规则，即每一行任意 4 个单元，不全为 0，不全为 1，枚举写入 360 个子句。

`int rule_3(char filename[80])`: 该函数用于向 filename 指向的文件中写入二进制数独的第三个规则，即没有任意两行或两列相同，穷举写入 1920 个子句。

`int Gotoxy(int x, int y)`: 该函数用于将控制台光标移至坐标 x, y，主要用在玩数独游戏时刷新界面。

`void printSudoku(int solution[])`: 该函数用于将 solution 中的数独打印到屏幕上，solution 中下标代表格子的序号，下标的元素为 1 时表示这一格填的是 1，为 0 时代表这一格填的是 0，为 -1 时表示这一格未填。输出时按照 6×6 矩阵输出各个格子的值，未填的格子输出下划线 ‘_’。

`int creatSudoku(const int *initial, int counts, char filename[], int blank, int sudoku[], int function)`: 该函数用于根据已填 `counts` 个格子的初始条件 `initial` 创建数独，并将数独转化为 CNF 公式写入 `filename` 指向的文件夹中，然后生成数独终盘，挖去 `blanks` 个空生成数独题面储存在 `sudoku` 中。该函数不仅用于生成题面，还可以用于检验 `initial` 是否合法，即可以用于检验用户完成的数独是否正确。由于函数执行过程中会输出一些提示信息，进行两种功能时输出的提示信息不同，所以通过参数 `function` 选择功能，`function` 为 1 时用于生成数独，为 0 时用于检验答案。该函数对之前的函数的调用情况如图 4-3。

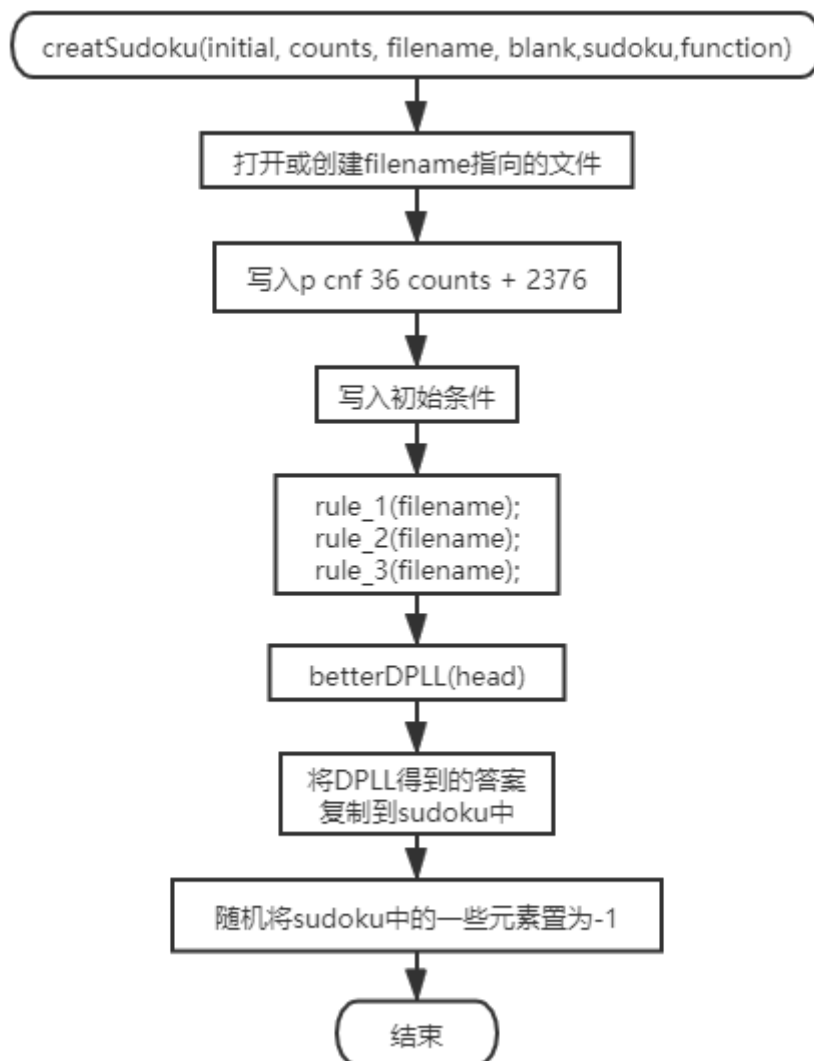


图 4-3: `creatSudoku` 函数调用情况

`void playSudoku(char filename[80])`: 函数用于交互玩数独，将数独生成的 CNF 公式存入 `filename` 指向的文件中。函数首先生成初始条件，即在第 1 至 36

号空中随机选两个填为 1，然后调用 creatSudoku 生成数独，之后按照图 3-3 所示的流程进行交互玩数独。函数调用情况如图 4-4。

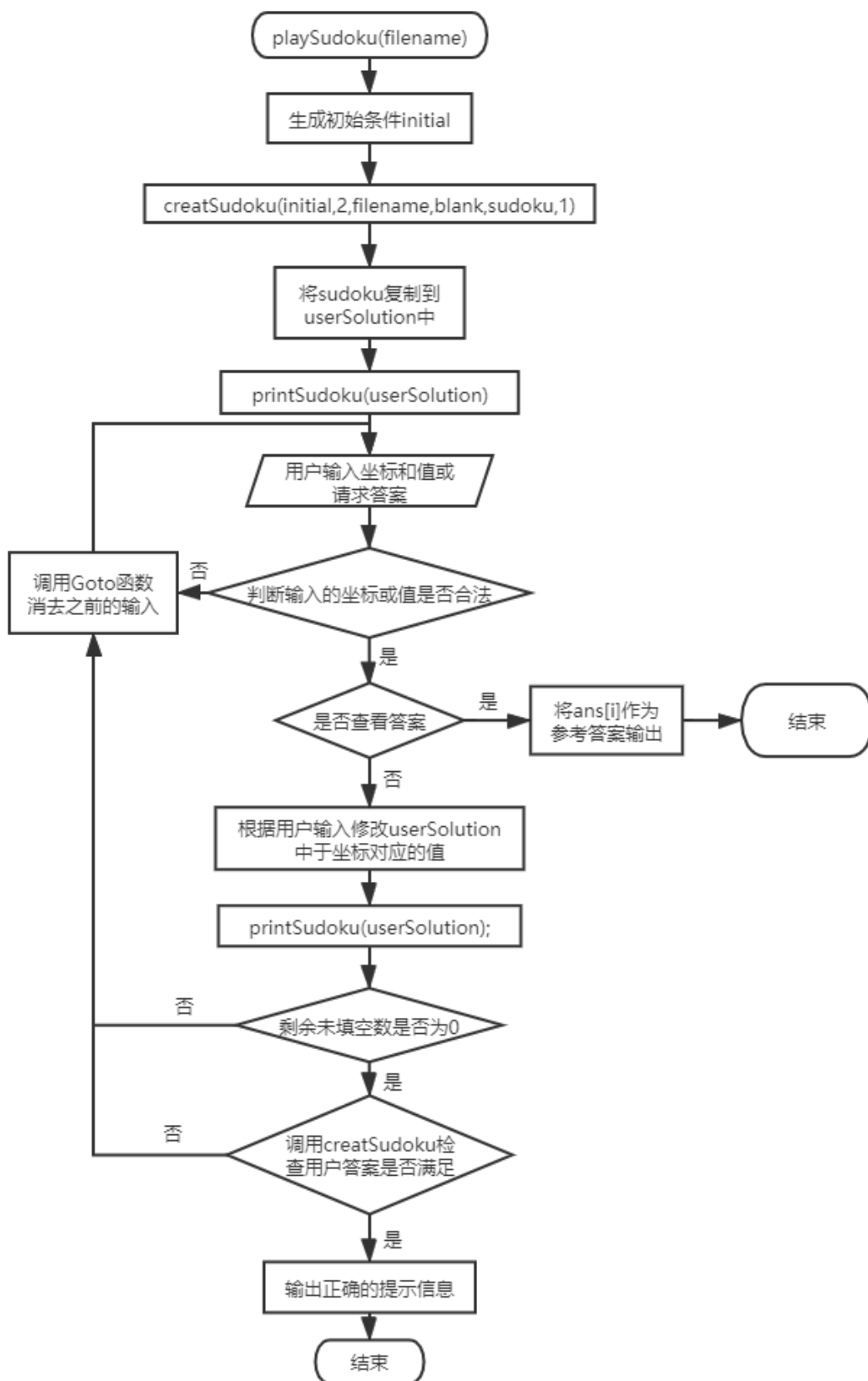


图 4-4: playSudoku 函数调用情况

4.2 系统测试

4.2.1 SAT 求解模块测试

(1)模块功能

该模块要实现的功能主要有：1.正确读取并解析 `cnf` 文件并能够逐行输出并显示每一个子句以验证解析正确性；2.基于 DPLL 正确求解 SAT 算例并测量程序执行时间；3.对同一算例执行优化后的 DPLL 算法并计算优化率；4.正确输出解文件。

(2)测试大纲

我们首先利用基准算例中功能测试用例 `sat-20.cnf` 和 `unsat-5cnf-30.cnf` 这两个较为简单的算例完整执行一遍所有功能，并利用 `verify.exe` 检验满足的算例得到的解是否正确，最后检验一些程序对非法输入的处理，以检验程序功能性是否正确；然后通过随机抽取的十个 `cnf` 文件对性能优化进行测试并记录。

(3)测试结果

检验 `sat-20.cnf` 是否正确存储的运行结果如图 4-5，4-6，4-7，4-8。

```

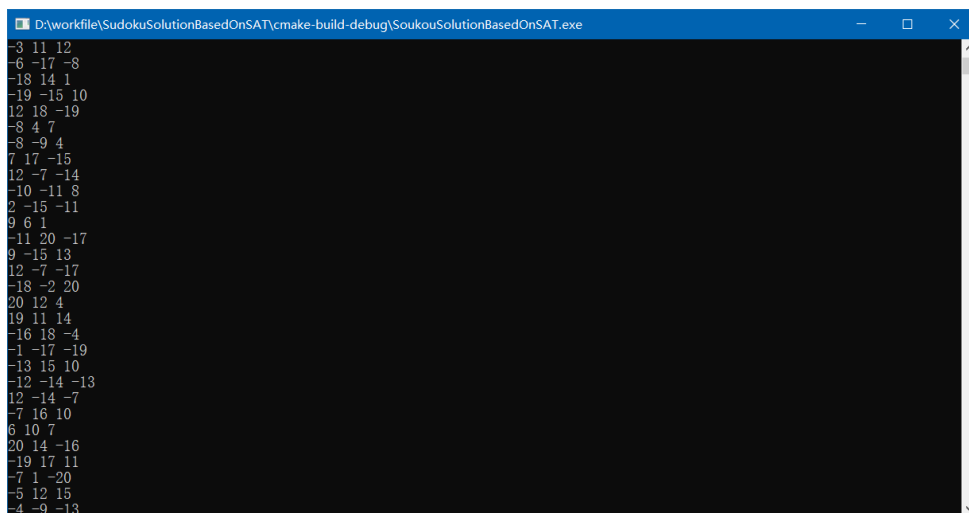
D:\workfile\SudokuSolutionBasedOnSAT\cmake-build-debug\SoukouSolutionBasedOnSAT.exe
请选择需要实现的功能
1. 求解SAT      2. 玩数独游戏
0. 退出

请选择你的操作[0~2]:1

请输入文件名: D:\FileRecv\程序设计综合课程设计任务及指导学生包\SAT测试备选算例\基准算例\功能测试\sat-20.cnf
共有20个变元, 91个子句
是否输出cnf文件
1. 是
2. 否
请选择:1

4 -18 19
3 18 -5
-5 -8 -15
-20 7 -16
10 -13 -7
-12 -9 17
17 19 5
-16 9 15
11 -5 -14
18 -10 13
-3 11 12
-6 -17 -8
-18 14 1
    
```

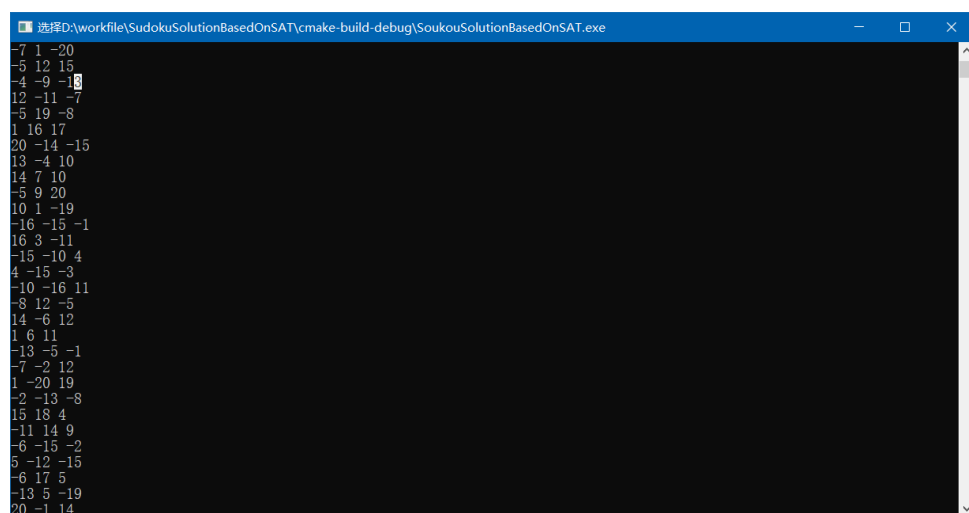
图 4-5：检验 `sat-20.cnf` 是否正确存储运行结果截图



```

-3 11 12
-6 -17 -8
-18 14 1
-19 -15 10
12 18 -19
-8 4 7
-8 -9 4
7 17 -15
12 -7 -14
-10 -11 8
2 -15 -11
9 6 1
-11 20 -17
9 -15 13
12 -7 -17
-18 -2 20
20 12 4
19 11 14
-16 18 -4
-1 -17 -19
-13 15 10
-12 -14 -13
12 -14 -7
-7 16 10
6 10 7
20 14 -16
-19 17 11
-7 1 -20
-5 12 15
-4 -9 -13
    
```

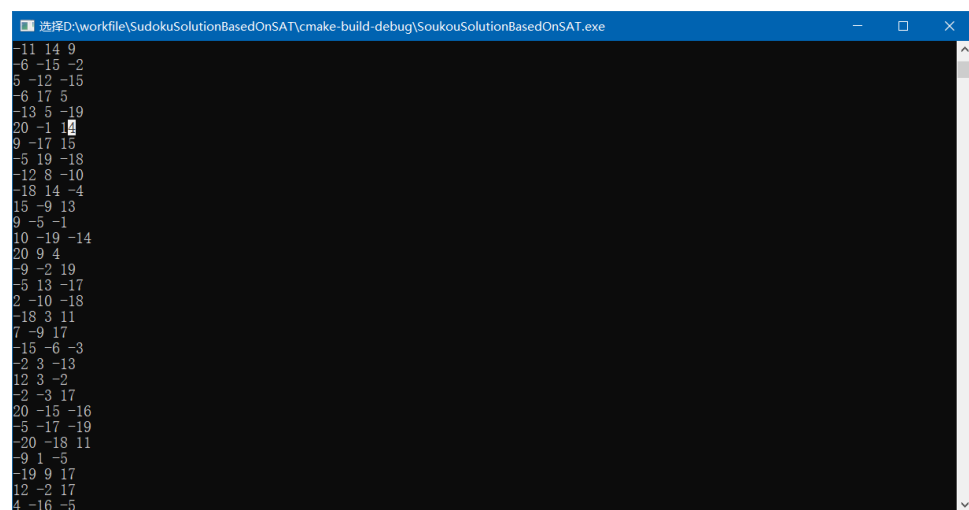
图 4-6：检验 sat-20. cnf 是否正确存储运行结果截图



```

-7 1 -20
-5 12 15
-4 -9 -18
12 -11 -7
-5 19 -8
1 16 17
20 -14 -15
13 -4 10
14 7 10
-5 9 20
10 1 -19
-16 -15 -1
16 3 -11
-15 -10 4
4 -15 -3
10 -16 11
-8 12 -5
14 -6 12
1 6 11
-13 -5 -1
-7 -2 12
1 -20 19
-2 -13 -8
15 18 4
-11 14 9
-6 -15 -2
5 -12 -15
-6 17 5
-13 5 -19
20 -1 14
    
```

图 4-7：检验 sat-20. cnf 是否正确存储运行结果截图



```

-11 14 9
-6 -15 -2
5 -12 -15
-6 17 5
-13 5 -19
20 -1 18
9 -17 15
-5 19 -18
-12 8 -10
-18 14 -4
15 -9 13
9 -5 -1
10 -19 -14
20 9 4
-9 -2 19
-5 13 -17
2 -10 -18
-18 3 11
7 -9 17
-15 -6 -3
-2 3 -13
12 3 -2
-2 -3 17
20 -15 -16
-5 -17 -19
-20 -18 11
-9 1 -5
-19 9 17
12 -2 17
4 -16 -5
    
```

图 4-8：检验 sat-20. cnf 是否正确存储运行结果截图

检验 sat-20.cnf 的求解，时间测量，优化率计算，解文件输出以及解的验证操作结果如图 4-7，4-8，4-9。

```

D:\workfile\SudokuSolutionBasedOnSAT\cmake-build-debug\SoukouSolutionBasedOnSAT.exe
-18 3 11
7 -9 17
-15 -6 -3
-2 3 -13
12 3 -2
-2 -3 17
20 -15 -16
-5 -17 -19
-20 -18 11
-9 1 -5
-19 9 17
12 -2 17
4 -16 -5
请选择使用何种算法
1. 未优化变元选择策略
2. 优化变元选择策略
您的选择是: 1
DPLL求解中.....
有解, 执行时间 0 ms
是否输出解文件
1. 是
2. 否
请选择: 1
已输出至cnf文件所在文件夹下同名res文件中
是否执行另一种算法
1. 是
2. 否
请选择: 1
执行时间0 ms, 无法计算优化率
    
```

图 4-7：对 sat-20. cnf 的求解的时间测量、优化率计算截图

```

sat-20.res - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
s 1
v -1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20
t 0ms
    
```

图 4-8：对 sat-20. cnf 的解文件输出结果截图

```

C:\WINDOWS\system32\cmd.exe
C:\Users\95498>D:\FileRecv\助教课演示检查要求\verify.exe D:\FileRecv\程序设计综合课程设计任务及指导\学生包\SAT测试设备\算例\基准算例\功能测试\sat-20.cnf D:\FileRecv\程序设计综合课程设计任务及指导\学生包\SAT测试设备\算例\基准算例\功能测试\sat-20.res
v value
1 0
2 1
3 1
4 1
5 0
6 0
7 0
8 1
9 1
10 1
11 1
12 0
13 0
14 1
15 1
16 0
17 1
18 1
19 1
20 1
Assignment is satisfying!
C:\Users\95498>
    
```

图 4-9：对程序解得的 sat-20. res 的正确性进行测试截图

检验 unsat-5cnf-30.cnf 是否正确存储的运行结果如图 4-10，由于子句数较多故只给出部分截图，全部截图见压缩包测试结果目录。

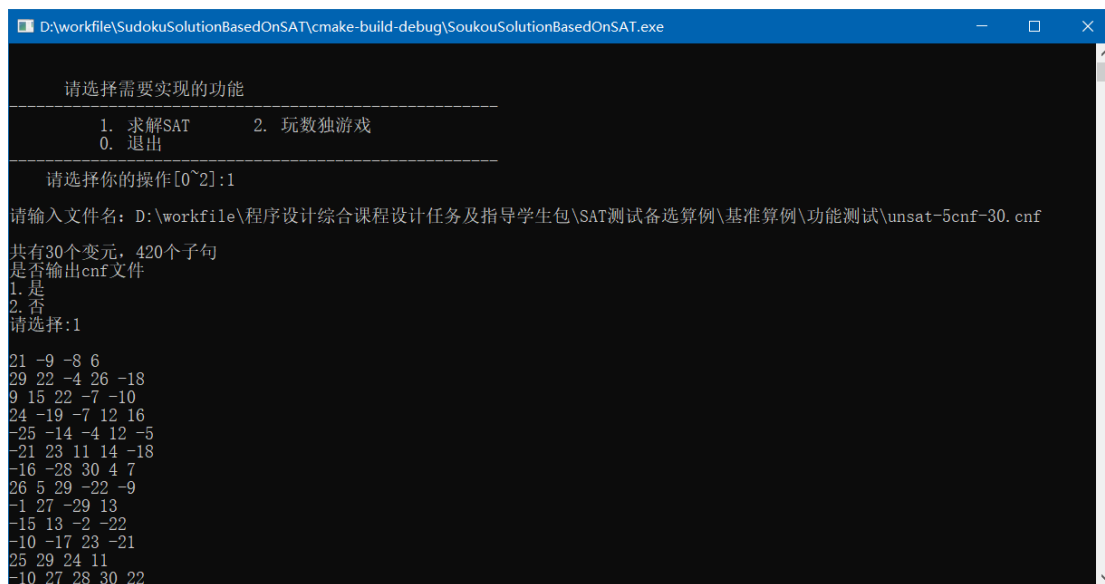


图 4-10：检验 unsat-5cnf-30.cnf 是否正确存储运行结果截图

检验 unsat-5cnf-30.cnf 的求解，时间测量，优化率计算以及解文件输出操作结果如图 4-11，4-12。

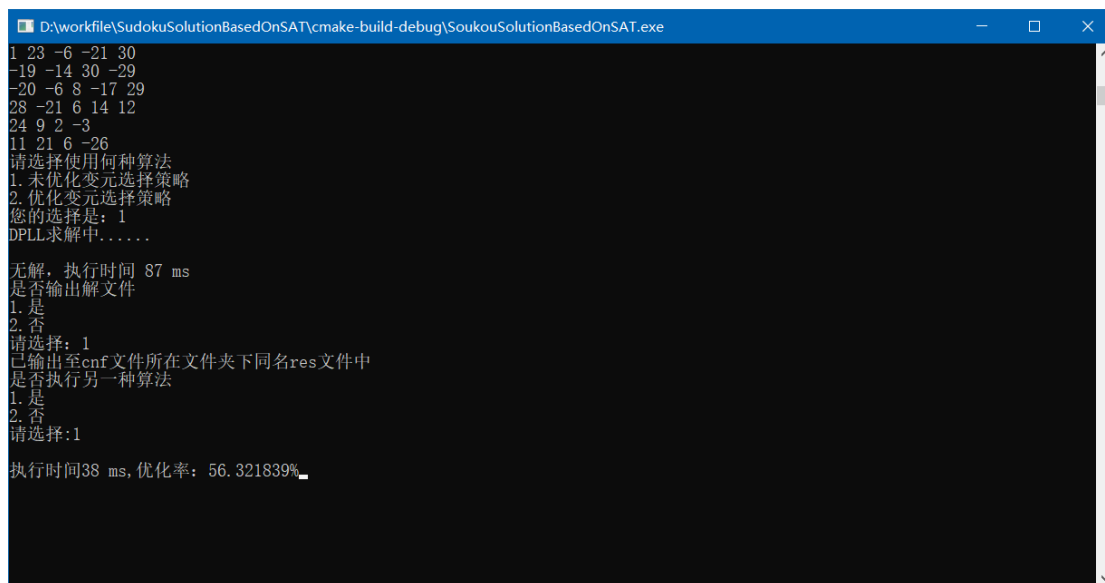


图 4-11：对 unsat-5cnf-30.cnf 的求解的时间测量、优化率计算截图

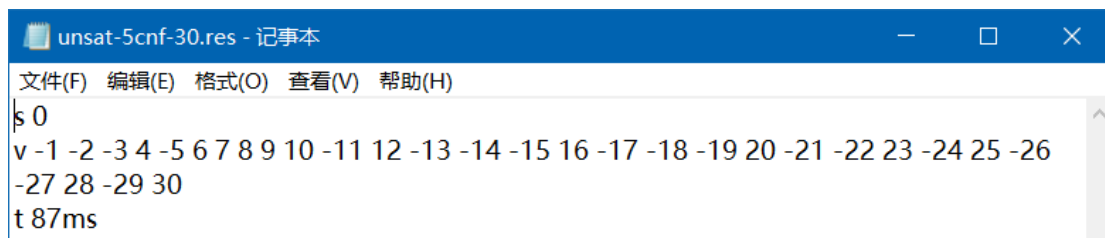


图 4-12：对 unsat-5cnf-30.cnf 的解文件输出结果截图

程序对输入非法功能序号或不存在的文件的处理截图如图 4-13，4-14。

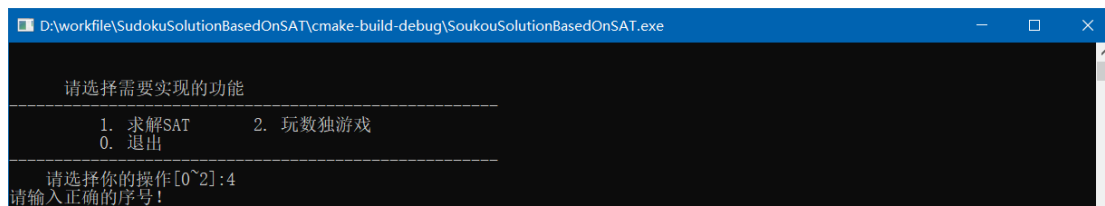


图 4-13：程序对输入非法功能序号的处理

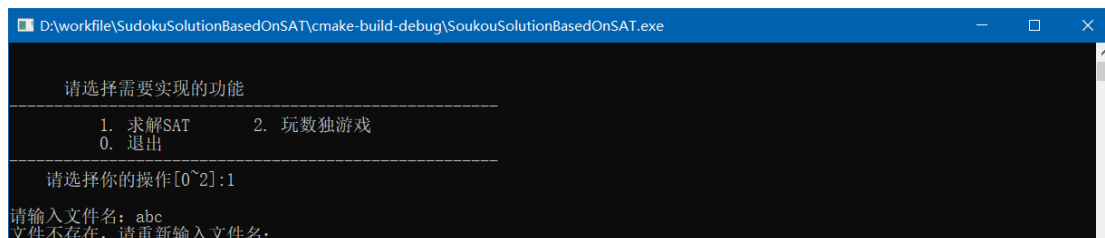


图 4-14：程序对输入不存在的文件名的处理

程序对随机选取的十个算例的性能优化结果如表 4-1，运行截图见压缩包测试结果目录。

表 4-1：优化测试结果

用例	文件名	变元数	子句数	文件大小(kb)	优化前时间(ms)	优化后时间(ms)	优化率(%)
1	php-010-008.shuffled-as.sat05-1171.cnf	80	370	否	6514	6092	6.48
2	u-5cnf_3900_3900_060.shuffled-60.cnf	60	936	否	317120	26545	91.63
3	u-problem7-50.cnf	50	100	否	148	109	26.35
4	ais10.cnf	181	3151	是	3415	1764	48.35
5	sud00009.cnf	303	2851	是	128	6502	<0
6	u-5cnf_3500_3500_30f1.shuffled-30.cnf	30	420	否	84	35	58.33
7	7cnf20_90000_90000_7.shuffled-20.cnf	20	1532	是	18	10	44.44
8	problem3-100.cnf	100	340	是	81	32	60.49
9	problem6-50.cnf	50	100	是	3	1	66.67
10	problem11-100.cnf	100	600	是	14	7	50.00
11	problem12-200.cnf	200	1200	是	389	320	17.74
12	sud00082.cnf	224	1762	是	35	1327	<0
13	ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf	1584	16587	是	溢出	19771	未定
14	eh-dp04s04.shuffled-1075.cnf	1075	3152	是	26602	6883	74.13

(4)测试结果分析

在功能测试中,程序能够正确存储 CNF 公式、求解规模不太大的 SAT 问题、测量运行时间以及输出解文件,并且对一些异常情况,如用户输入非法的功能序号、不存在的文件名或者计算优化率时 0 作为分母等情况做出相应处理,很好地满足了设计目标。

优化测试中,除了部分算例外,优化后的算法对大部分算例都能够减少运行时间。分析没有优化的算例,我们发现:首先这些算例基本属于中型算例,这就使优化算法中对变元选取的计算时间一方面比较长,另一方面相较于 DPLL 计算时间又不可忽略,只要是采取基于某种计算方法选取变元的策略都无法避免这一点;其次这些算例中大部分的子句的长度都是一样的,这就使得在文字数较少的子句中挑出出现频率较大的文字失去了意义,这属于该算法本身的缺陷。对于大部分的算例,程序能够满足设计目标。

4.2.1 数独模块测试

(1)模块功能

该模块要实现的功能主要有: 1.能够随机生成六阶二进制数独题面; 2.程序具有一定交互性,用户可以求解数独,最后程序能够判断用户的解是否正确 3.用户可以随时查看答案。

(2)测试大纲

进行两场数独游戏,一场直接选择查看答案,另一场完成数独,查看程序结果。在第一场游戏过程中模拟一些异常情况,如输入异常坐标,异常值等。

(3)测试结果

第一场游戏生成的题面如图 4-15,输入的横纵坐标大于 6 时的结果如图 4-16、4-17,输入的值不为 0 或 1 时的结果如图 4-18,输入的坐标为题面给出的格子坐标时的情况如图 4-19,查看答案时的结果如图 4-20。

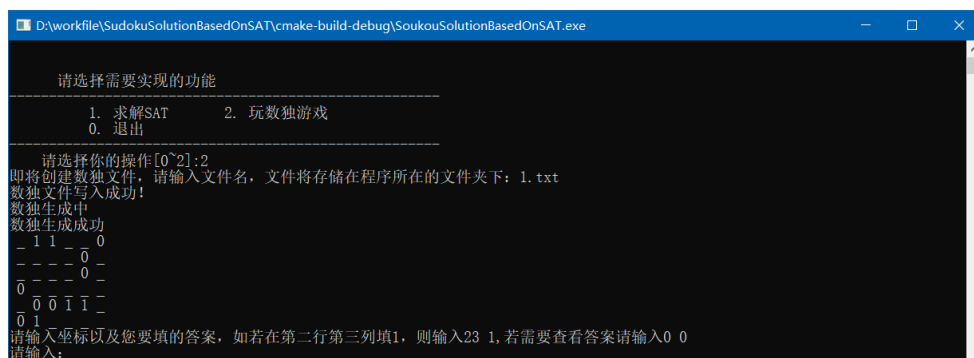


图 4-15：第一场游戏生成题面

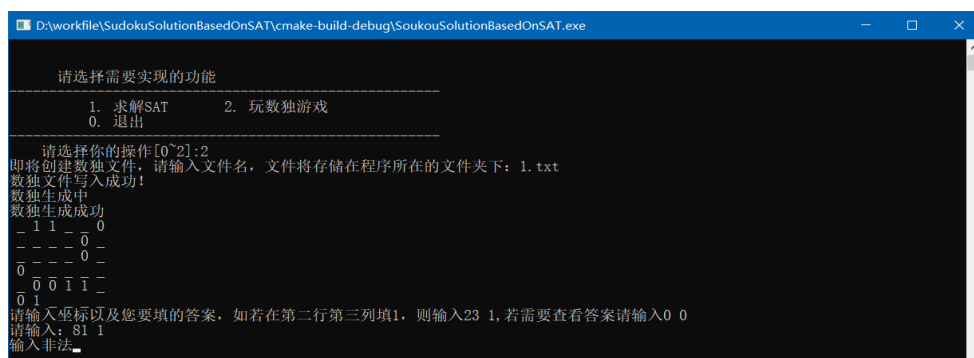


图 4-16：第一场游戏输入横坐标大于 6

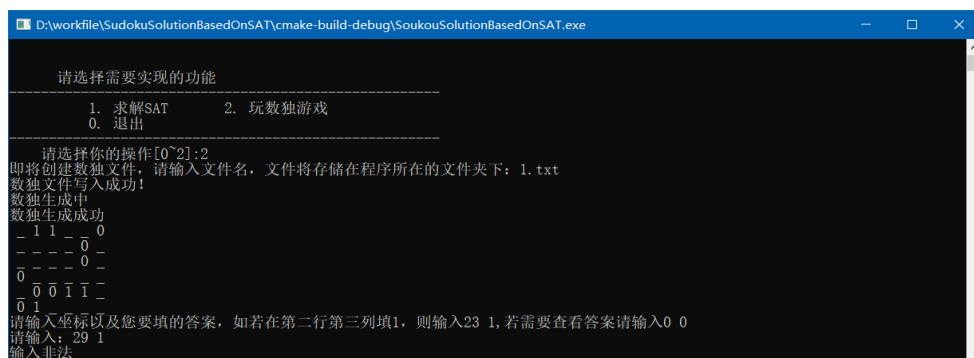


图 4-17：第一场游戏输入纵坐标大于 6

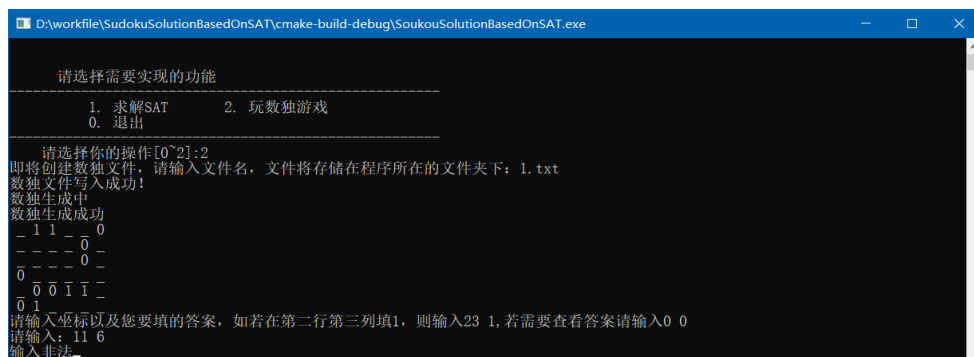


图 4-18：第一场游戏输入的值不为 0 或 1

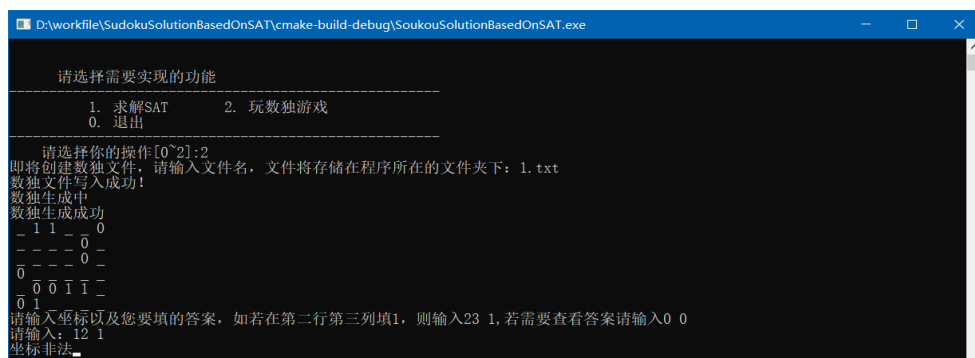


图 4-19：第一场游戏输入坐标为题面给出的格子坐标



图 4-20：第一场游戏输入参考答案时的结果

第二场游戏生成的题面如图 4-21，填第一个空时的效果如图 4-22，填错时的效果如图 4-23，填对时的效果如图 4-24。

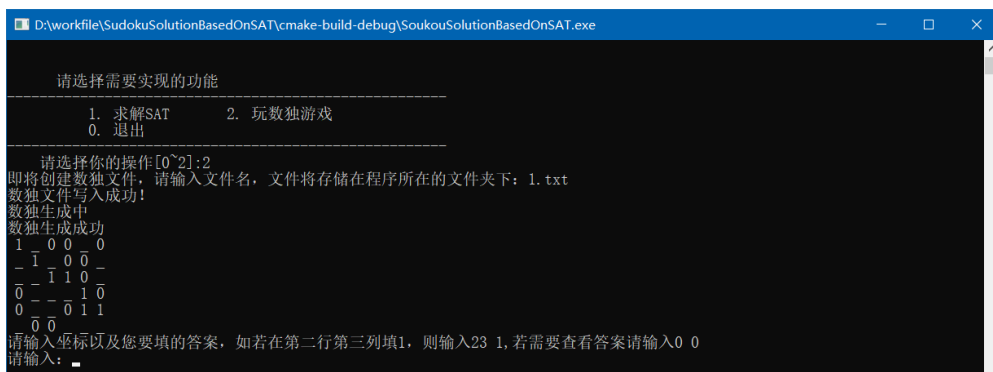


图 4-21：第二场游戏生成的题面

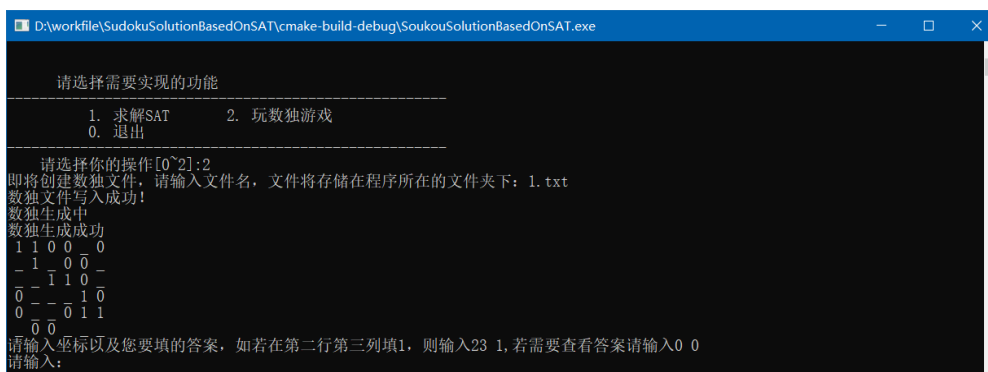


图 4-22：填完第一个空的效果

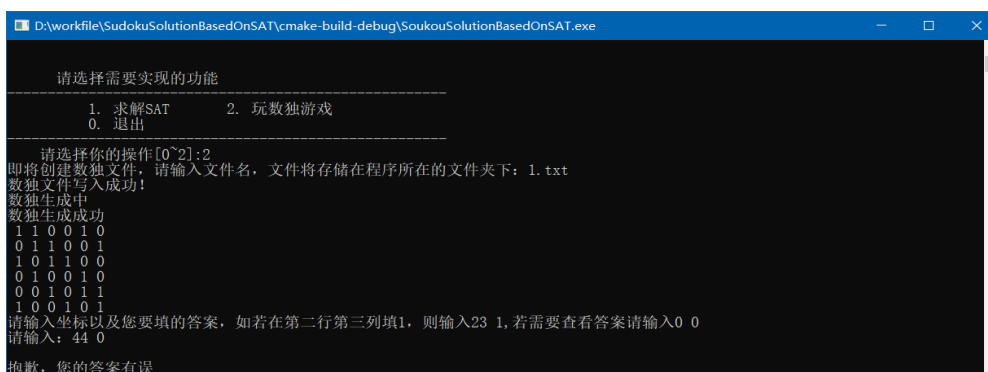


图 4-23：填完最后一个空后填错的效果

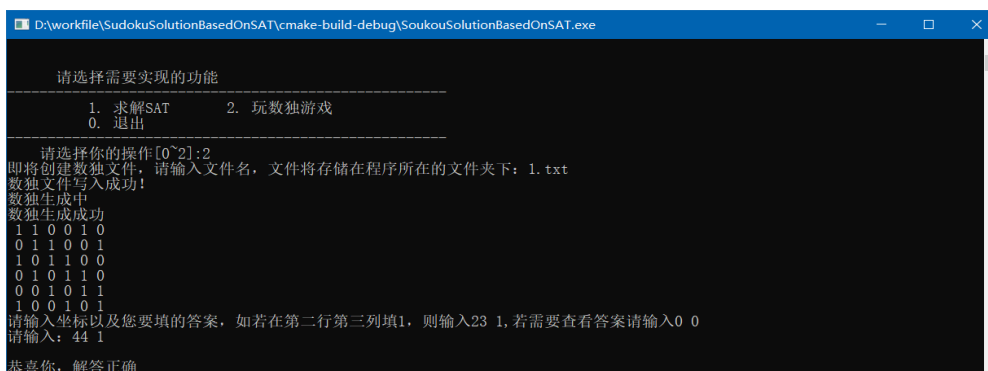


图 4-24：填完最后一个空后填对的效果

(4)测试结果分析

程序能够随机生成数独，能够进行交互解题，在用户填完所有的空后能够正确判断用户提供的答案是否正确，也能提供正确答案供用户参考，在用户正确解答后或请求查看答案后游戏结束，若用户未填完或解答错误时不会停止游戏，符合游戏设计逻辑，总体上能够满足设计目标。但是数独题面有多解的情况出现，虽然在用户提供的答案于参考答案不同但依然正确时程序能够正确判断，但原则上数独不应该有多解，这里有待改进。

5 总结与展望

5.1 全文总结

本次实验完成了基于 SAT 的二进制数独游戏求解程序的设计与实现，主要完成了以下方面的工作：

(1)查阅了相关文献资料，了解了国内外关于 SAT 问题的研究成果，学习了一些关于布尔逻辑的可满足性问题的基本概念。

(2)设计了有效的数据结构用以存储 CNF 公式

(3)查阅相关资料，了解了 DPLL 算法求解 SAT 问题的过程，并编写程序实现了 DPLL 算法，同时通过查阅文献学习了一些变元选取策略的优化，并实现了其中一种，即最短子句出现频率最大优先法用于优化 DPLL 算法。

(4)学习了二进制数独相关内容，利用数独的三个限制条件将数独的初始条件转化为 CNF 公式用于 DPLL 求解。

(5)设计了数独游戏的交互界面。

(6)实现了一个简单的具有交互性的菜单功能。

5.1 工作展望

针对本次实验中出现的不足，在今后的研究中，围绕着如下几个方面开展工作。

(1)本次实验采用的数据结构为最基本的将子句表示为由文字构成的链表，整个公式表示为子句构成的链表的结构，该结构最大的弊端在于查找的时间复杂度大，可以尝试改进为更加便于查找的结构。

(2)采用递归实现DPLL算法，递归的效率不高，而且一旦栈溢出程序便崩溃，可以尝试采用非递归实现，提升程序的性能。

(3)本次实验采用的优化变元选取策略在一些算例中暴露出了明显的弊端，这种策略的普适性显然不强，今后可以尝试寻找一些普适性较强的策略。

(4)数独的交互界面是基于控制台的，交互性不强，而且经常会出现一些字符重叠的现象，今后可以考虑实现一个 GUI。

(5)数独生成时没有一个有效的办法使生成的数独只有唯一解，以后可以去思考如何保证挖洞生成的数独具有唯一解。

6 体会

本次实验工作量较大，涉及的知识较多。为了完成这次实验我查阅了不少资料，花了很多的时间。对于这次实验，我有如下体会：

1. 在写比较大的项目的时候要提前做好前期准备工作，比如进行模块设计，画流程图，多查阅相关资料以及论文，能够大大减少走弯路的可能性。
2. 注意程序的模块化，尽量减少代码之间的耦合，降低调试难度
3. 设计程序时眼光不能只盯着某一个功能看，要有系统意识，不仅要能够写出实现某个功能的函数，更要考虑到函数之间的参数传递的关系。
4. 养成在关键的地方写注释的习惯。本次实验的代码量较大，在一开始我几乎没有写注释时想要寻找某一段可能有问题的代码时非常困难，在加了注释之后视觉上来说代码的分块会更明显，能够提高开发效率。
5. 对基础的数据结构的操作熟练的话能够大大提高开发效率。本次程序涉及了大量对链表的增删查改等操作，很难想象如果我对链表的操作水平若是还和大一时一样这个实验该怎样进行，本次实验让我更加深刻的意识到了“程序等于数据结构加算法”这句话的意义。

参考文献

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