

华中科技大学

课程设计报告

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： 计算机科学与技术 2008 班

学 号： U202015533

姓 名： 徐瑞达

指导教师： 袁凌

报告日期： 2021 年 10 月 7 日

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，统计求解时间与优化效率。本设计要求实现基于 DPLL 的算法与程序框架，包括程序的改进也必须在此算法的基础上进行。

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：**将数独游戏^[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一方法、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>
- [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] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

目 录

任 务 书	I
设计内容	I
设计要求	I
参考文献	I
1 引言	1
1.1 课题背景与意义	1
1.2 国内外研究现状	1
1.3 课程设计的主要研究工作	2
2 系统需求分析与总体设计	3
2.1 系统需求分析	3
2.1.1 SAT 求解器	3
2.1.2 数独游戏	3
2.2 系统总体设计	3
2.2.1 SAT 求解器	3
2.2.2 数独游戏	4
3 系统详细设计	5
3.1 有关数据结构的定义	5
3.1.1 数据结构定义	5
3.1.2 数据结构逻辑关系	7
3.1.3 相关常量与预定义声明	7
3.2 主要算法设计	8
3.2.1 CNF 范式解析	8
3.2.2 DPLL 算法	14
3.2.3 数独生成与归约	15
3.3 算法优化设计	18
3.3.1 数据结构优化	18
3.3.2 变元决策策略的优化	19
4 系统实现与测试	20

4.1 系统实现	20
4.1.1 文件读取 (fin.h)	20
4.1.2 SAT 求解器 (dp11.h)	20
4.1.3 数独游戏 (sudoku.h)	21
4.2 系统测试	21
4.2.1 SAT 求解器模块测试	21
4.2.2 数独模块测试	25
4.3.3 性能测试表	30
5 总结与展望	32
5.1 全文总结	32
5.2 工作展望	32
6 体会	33
参考文献	35
附录	36

1 引言

1.1 课题背景与意义

对于计算机科学与技术学院的大二学生，在大一学年已经学习了 C 语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及 C 语言程序设计实验、数据结构实验两门编程实践课程，不仅具有较为系统性的 C 语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“综合程序设计”课程设计正是使大家从简单验证到综合应用，甚至在编程中实现智慧与风格升华的重要实践环节，为后续学习与进行计算机系统编程打下坚实的基础，让综合编程技能成为大家的固有能力和通向未来专业之门的钥匙。

1.2 国内外研究现状

近十多年来，可满足性问题研究逐渐升温，已成为了国际国内的研究热点，取得了一批相当重要的理论和实践成果，应该说当前的 SAT 问题研究比十多年前已取得了很大的突破，并直接或间接地推动了其他相关领域（比如形式验证，人工智能等领域）的发展。

国际上已提出了各种不同的局部搜索算法和回溯搜索算法，使得 SAT 解决器解决不同领域中的 SAT 问题的能力不断增强，能解决的问题的规模不断扩大。其中局部搜索算法显示出对于随机的 SAT 问题特别有用，而回溯搜索算法则被用来解决大规模实际应用领域中的 SAT 问题。事实上，国际上已提出了一大批采用回溯搜索算法的高效的 SAT 问题求解器，其中绝大多数提出来的回溯搜索算法是对原始的 DPLL 回溯搜索算法的改进算法。这些改进措施包括：新的变量决策策略，新的搜索空间剪除技术，新的推理和回溯技术以及新的更快的算法实现方案和数据结构等。当前水平的 SAT 问题求解器已能够轻松解决以前传统 SAT 问题求解器完全无法解决的可满足性问题。

尽管当前的 SAT 问题解决器已取得了相当重要的进步，但是研究的脚步不会停止，我们还可以提出一些值得研究的问题。比如，是否存在新的更高效的 SAT 问题处理技术可以集成到 DPLL 算法框架内；是否可以找到除局部搜索，回溯搜索之外的其他 SAT 算法来更有效地解决 SAT 问题；是否能提出更好的 SAT 改进算法和实现方案。

1.3 课程设计的主要研究工作

本次实验中，实验者选择了“基于 SAT 问题的数独游戏求解程序”作为实验课题，实现 SAT 求解器和数独游戏两个功能。

SAT 求解器基于 DPLL 的完备算法，对 CNF 范式算例文件进行求解，输出答案，并可选择遍历验证答案或将答案存入文件；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。

2 系统需求分析与总体设计

2.1 系统需求分析

本系统的主要目标是实现一个基于 DPLL 的 SAT 求解器，并应用该求解器实现具有一定可玩性的数独游戏。

2.1.1 SAT 求解器

SAT 求解器可对 CNF 范式算例文件进行解析，并存入定义的数据结构中，然后调用 DPLL 算法对其进行求解，求解完成后输出求解答案及求解所用时间，同时将求解结果存入同名文件（文件扩展名为.res），并可通过逐行输出 CNF 范式中子句及其真值来验证求解结果的正确性。

2.1.2 数独游戏

数独部分包括生成具有一定可玩性的数独游戏及求解数独 CNF 范式算例文件两部分。数独游戏模块将输出有着不同难度等级的含有未填数字（打印为空格）的数独游戏初盘，并给予游戏操作提示，用户可输入自己求解的答案，系统将检验其答案的正确性，答案错误时可选择直接查看正确答案或继续解题。数独文件模块将根据难度等级生成数独并将其归约为 CNF 文件调用 SAT 求解器求解，最后终端将以数独终盘的形式打印求解结果。

系统通过提示不同级菜单的层级和操作类别提示用户，用户可自行选择要进行的操作。

2.2 系统总体设计

系统有SAT求解器和数独功能两个功能模块：

2.2.1 SAT 求解器

SAT求解器求解CNF范式算例文件，实现该功能包括4个部分：

①CNF范式算例文件解析功能模块，建立其对应的数据存储结构；

该部分模块需实现对CNF算例文件的解析和存储，并能够验证并保存解析结果，销毁已生成的CNF范式链表等操作；

②基于DPLL算法的SAT求解模块，对文件算例进行求解：

该部分模块需实现进行对CNF范式的删除文字或子句，判断单子句，回溯邻接表等操作；

③结果正确性的验证功能模块；

该部分模块需实现遍历整个CNF范式，通过验证每个子句的真值情况，验证求解结果正确性的操作；

④保存结果模块，将求得的结果与用时保存至用户输入的CNF算例同名文件（文件扩展名为.res）。

2.2.2 数独游戏

数独游戏模块包括2个功能部分：

①生成数独游戏模块：

该模块需实现输出游戏操作提示，自定义难度等级，体现数独初盘和用户输入结果对比性，中途提示冲突，实现良好的游戏界面与用户体验等操作；

②求解数独CNF范式算例文件：

该模块需实现自定义难度等级，生成数独，读取数独文件，将数独归约为CNF范例文件，求解数独等操作。

系统模块结构图如图 2-1 所示：

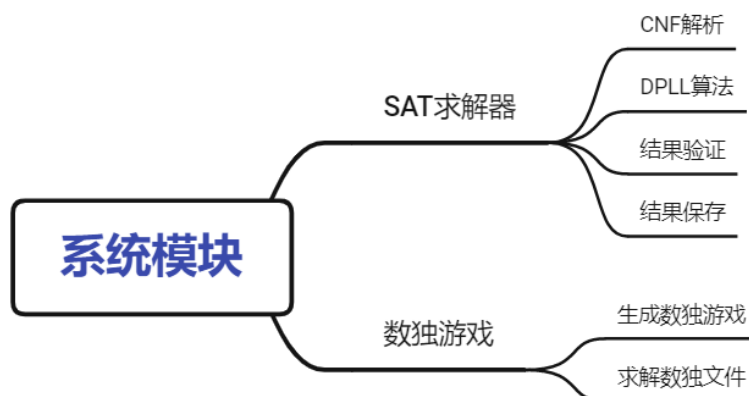


图 2-1 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 数据结构定义

本次实验主要运用了邻接链表，顺序存储线性表，二维数组三种数据结构来存储信息。其中，邻接链表用于存储CNF范式的子句与文字信息，顺序存储线性表用于存储与各变元有关的信息，二维数组则用于存储数独初盘，终盘，SAT求解器求解后的数独格局，便于数独的行列操作及输出。另外，其他还用到一些一维和二维数组作为辅助数组。

(1) 存储CNF范式的邻接链表的定义和声明如下：

① 子句链表文字结点数据结构类型定义：

```
typedef struct ClauseNode
{
    int flag;//标记该文字是否被删除
    int literal;//记录该结点的文字
    struct ClauseNode *next;//指向该子句中下一个文字结点
} ClauseNode;
```

② CNF范式链表结点（即子句链表头结点）数据结构类型定义：

```
typedef struct HeadNode
{
    int flag; //标记该子句是否被删除
    int count;//该子句中包含的文字结点个数
    struct HeadNode *next;//指向下一个子句链表头结点
    struct ClauseNode *horizon;//指向该子句链表中的第一个文字结点
} HeadNode;
```

③ CNF范式数据结构类型定义：

```
typedef struct CNF
```

```
{
    char type[20]; //CNF文件信息
    HeadNode *head; //CNF首个子句链表头结点
    int clause_num; //包含的子句数
    int literal_num; //包含的变元数
} CNF;
```

(2) 存储变元信息的顺序存储线性表定义如下：

① 变元线性表结点结构类型定义：

```
typedef struct VarValue
{
    int pos; //正变元数目
    int neg; //负变元数目
    int flag; //变元真值结果
    int times; //变元总出现次数
    int IsInit; //是否已经被赋真值
    HeadList *positive; //正变元信息链表
    HeadList *negative; //负变元信息链表
} VarValue;
```

② 存储变元信息链表的数据结构定义：

```
typedef struct HeadList
{
    HeadNode *phead; //该文字所在子句的链表头结点
    ClauseNode *Var; //该文字结点的指针
    struct HeadList *next; //指向下一个信息链表结点
} HeadList;
```

(3) 二维数组定义如下：

① 存储数独终盘的二维数组定义：

```
int final[9][9];
```

② 存储含空格数独问题的二维数组定义：

```
int Sudoku[9][9];
```

③ 存储SAT求解器得到的数独结果的二维数组定义：

```
int res[9][9];
```

(4) 系统中其他一些较重要的一维或二维数组的数据类型定义如下：

① 记录变元出现的不同次数等用于决策变量过程的线性表定义如下：

```
int *timesort, count, amount;
```

② 记录DPLL中用于回溯邻接链表的栈结构定义如下：

```
int *stack, StackSize;
```

③ 记录用户在数独游戏过程中出现的冲突的数组定义如下：

```
int conflict[9][9];
```

3.1.2 数据结构逻辑关系

线性变元表中各变元结点的正负文字信息链表依次指向CNF邻接链表中含有该文字的子句链表头结点和文字结点，如图3-1所示：

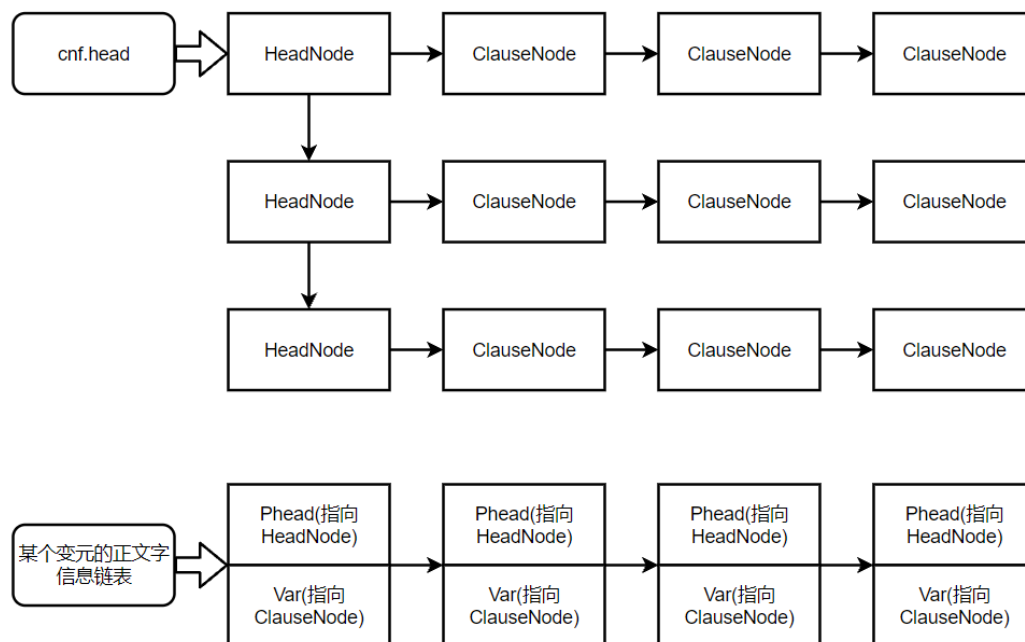


图 3-1 数据结构逻辑关系图

3.1.3 相关常量与预定义声明

```
#define OK 1

#define ERROR 0

#define TIMEOUT -3

#define overflow -2

#define INFEASIBLE -4

#define TIMELIMIT 180000

#define max(x, y) ((x) > (y) ? (x) : (y))
```

3.2 主要算法设计

本系统的主要算法包括对CNF范式链表的操作、DPLL算法和数独生成与归约算法三部分。

3.2.1 CNF 范式解析

对CNF范式数据结构的操作（如生成结构，删除子句，删除文字，判断单子句等）是DPLL函数的基础。

由于DPLL算法基于递归操作，需要在回溯时将子句链表复原，而采用复制链表的操作时间复杂度不理想，因此采用标记法记录文字和子句是否被删除，利用栈来记录单子句原则中选取的变元，从而易于回溯，同时这也大大方便了删除与复原操作。

该部分涉及的主要算法算法

（1）创建CNF范式链表：

创建 CNF 范式链表模块功能的实现为 `int LoadFile(const char *FileName, CNF *cnf)` 函数。函数流程图如图 3-2 所示。

该算法中，首先读取文件中的注释，然后读入 `cnf` 范式中的子句数和变元数。接着读取每一行子句，将每一行子句存储到一个链表中（关键为添加结点部分），并将这些链表的头结点串联形成纵向的头结点链表（关键语句为 `q->next = r;`）。

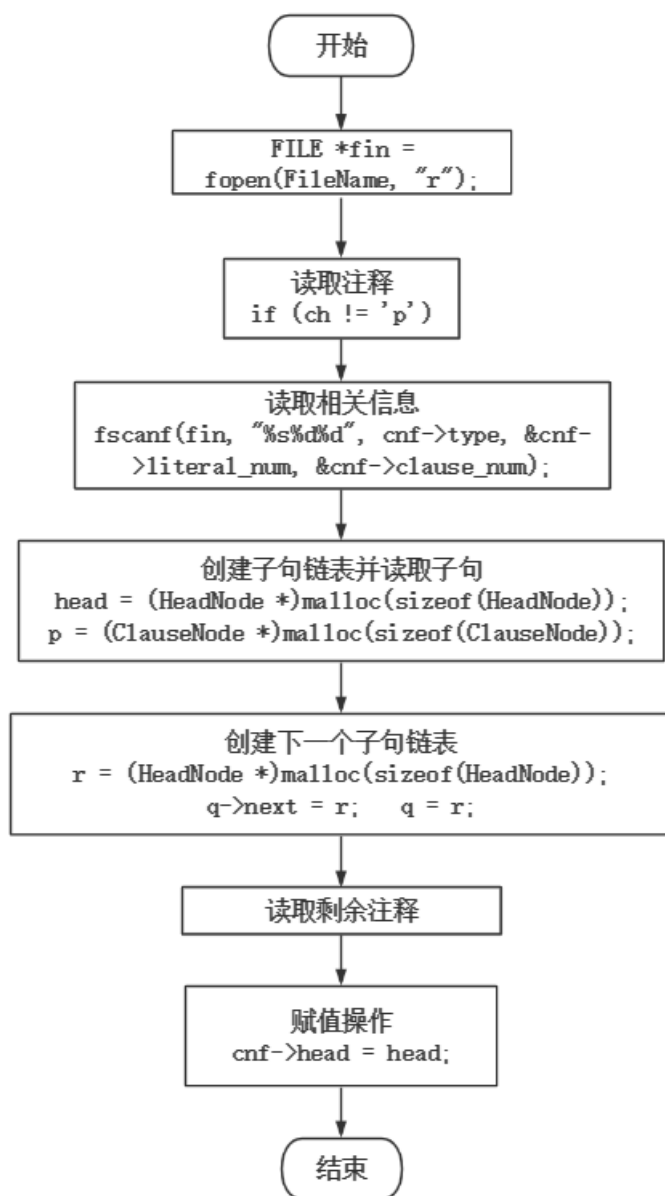


图 3-2 LoadFile 函数流程图

(2) 创建文字信息链表:

创建文字信息链表操作也即为每个文字的正负文字创建索引,便于进行删除与回溯,该功能的实现为void PreRecord(CNF *cnf)函数。函数流程图如图3-3所示。

该算法中,首先初始化ValueLisit链表,然后遍历CNF链表,对于每一个文字结点,更新其对应变元的出现次数,正负出现次数,并将其所在子句的链表头结点和该文字结点的指针存入对应变元的正文字信息表或负文字信息链表中。

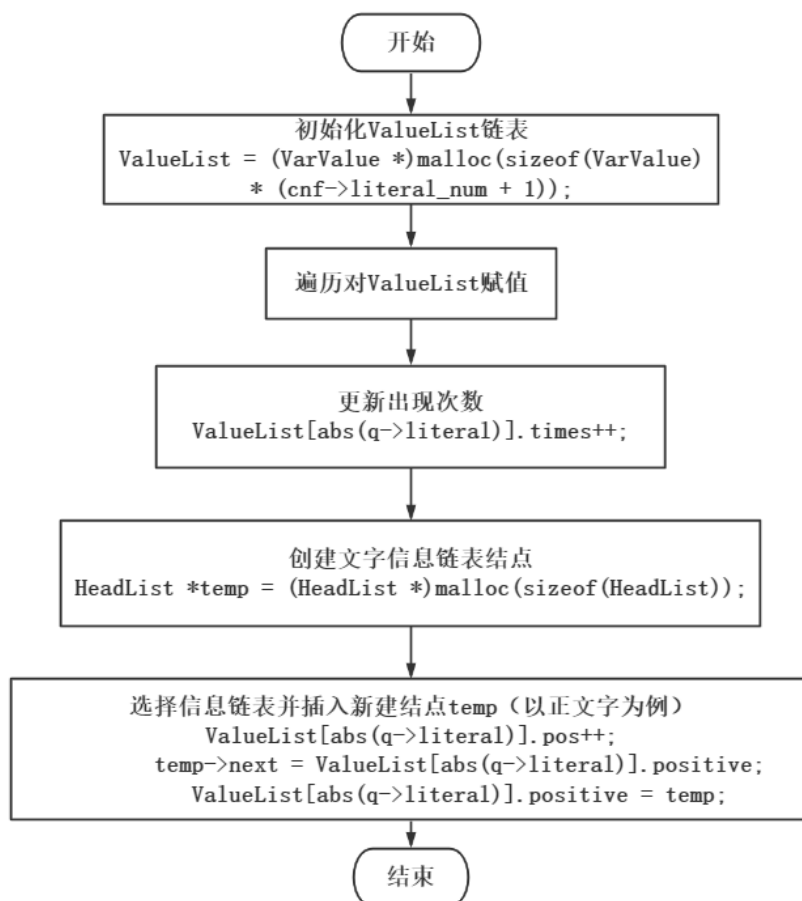


图 3-3 PreRecord 函数流程图

(3) 删除子句或文字：

删除与某文字相关的子句或文字操作是 DPLL 函数中 CNF 范式化简过程的基本操作之一，其实现为 `HeadNode *DeleteClause(HeadNode *head, int val)` 函数。函数流程图如图 3-4 所示。

该算法中，首先删除存在 `val` 的子句链表——选取对应的文字信息链表，将该链表中的每个头结点对应的子句标记为删除，并将被删除子句中的文字都标记为删除，同时要更新与之相关的 `ValueList` 中的信息；然后删除 `-val` 文字结点——选取对应的文字信息链表（应与上边的文字信息链表相反），将该链表中的每个文字结点标记为删除，同时更新其所在子句的信息及与之相关的 `ValueList` 中的信息。

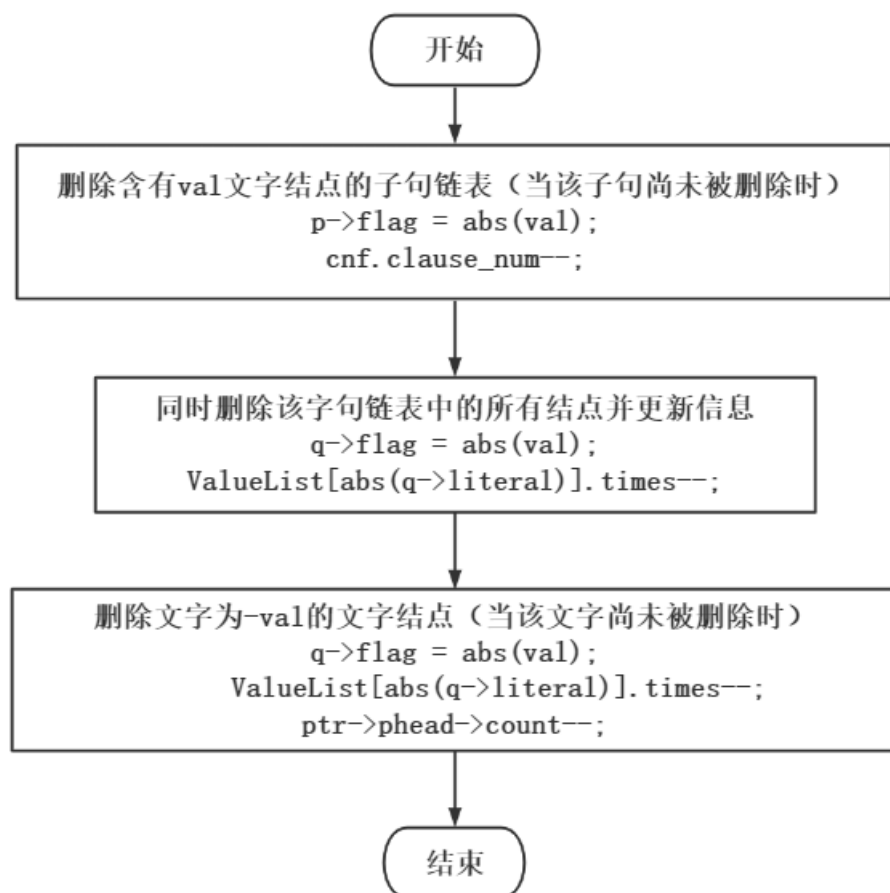


图 3-4 DeleteClause 函数流程图

（4）回溯子句或文字：

在 DPLL 回溯过程中，复原与某文字相关的子句或文字操作是必要操作，其实现为 `HeadNode *RecoverClause(HeadNode *head, int val)` 函数。函数流程图如图 3-5 所示。

该算法与删除子句或文字过程相逆，首先回溯存在 `val` 的子句链表——选取对应的文字信息链表，将该链表中的每个头结点对应的子句（注意被标记状态要与 `val` 相符）标记为未删除，并将被回溯子句中的文字（注意被标记状态要与 `val` 相符）都标记为未删除，同时要更新与之相关的 `ValueList` 中的信息；然后回溯 `-val` 文字结点——选取对应的文字信息链表（应与上边的文字信息链表相反），将该链表中的每个文字结点（注意被标记状态要与 `val` 相符）标记为未删除，同时更新其所在子句的信息及与之相关的 `ValueList` 中的信息。

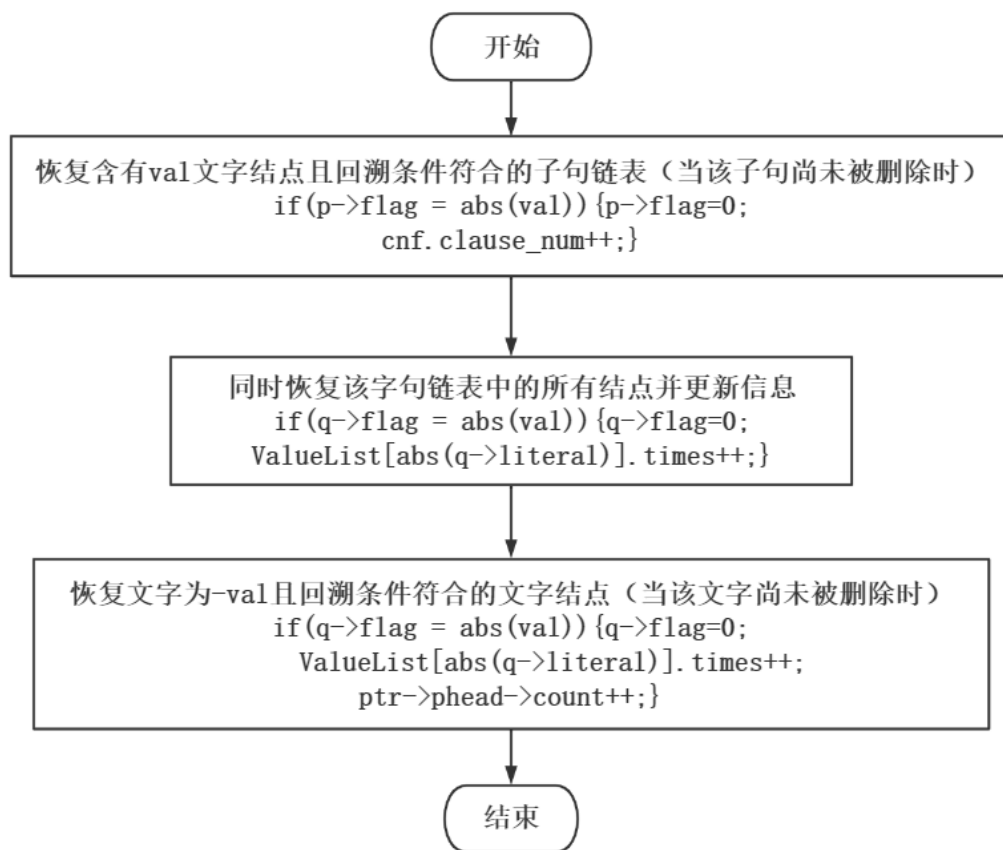


图 3-5 RecoverClause 函数流程图

(5) 寻找单子句：

在DPLL算法中查找单子句以进行删除等其他操作的功能的实现为函数int FindSingleClause(HeadNode *p)。函数流程图如图3-6所示。

该算法中，遍历子句头结点链表，如果该子句文字数为1而且该子句未被删除（ $p \rightarrow \text{flag} = 0$ ），则查找该子句中未被删除的文字结点，返回该文字。

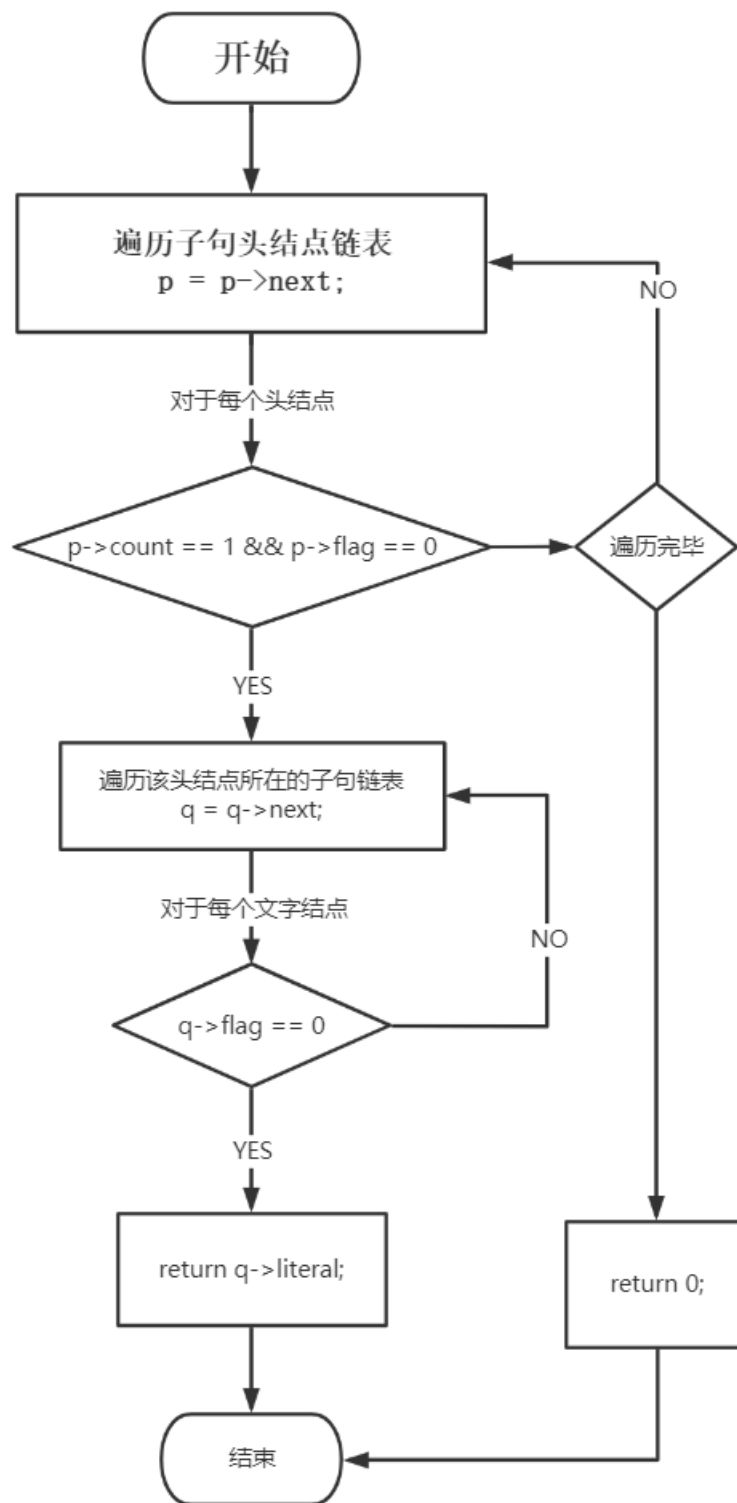


图 3-6 FindSingleClause 函数流程图

(6) 查找空子句:

空子句时退出DPLL算法的主要依据之一，查找空子句的功能实现为int EmptyClause(HeadNode *head)函数。函数流程图如图3-7所示:

该算法中，遍历子句头结点链表，如果该子句文字数为0而且该子句未被删除

($p \rightarrow \text{flag} == 0$)，则返回true。

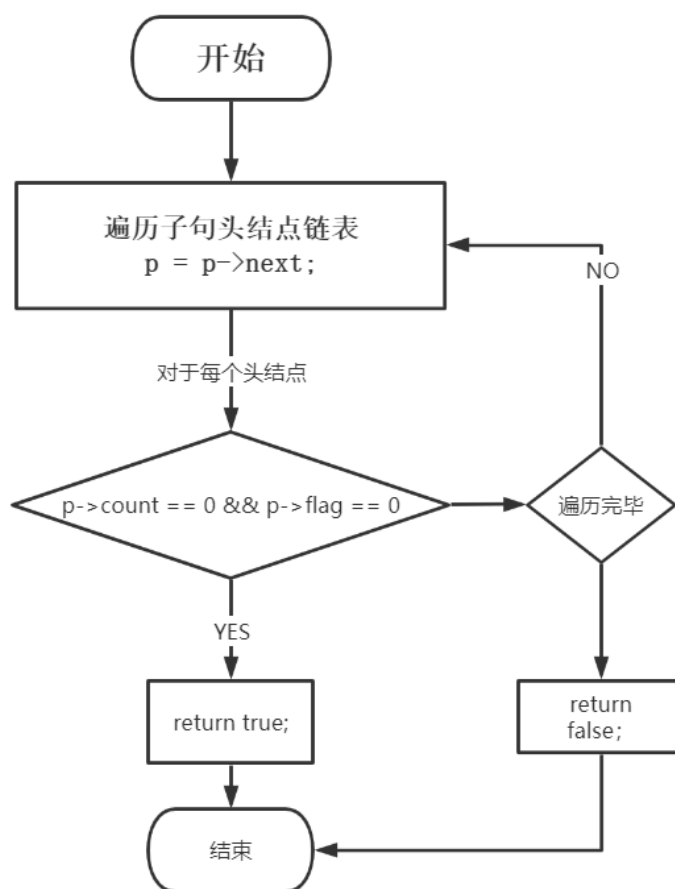


图 3-7 EmptyClause 函数流程图

3.2.2 DPLL 算法

DPLL 算法是 SAT 求解器的核心算法，利用递归工具，和单子句策略、分裂策略两个策略将 CNF 公式逐步化简。单子句策略是指找出 CNF 范式中的所有单子句，要使 CNF 范式有解其文字真值必为 1，则得出公式中所有含有该文字的子句真值均为 1，可删除这些子句；同时公式中所有该文字的负文字真值皆为 0，可以删去链表中该文字的负文字；分裂策略是指按某种策略选出一个未赋真值的变元的文字，将其作为单子句加到子句链表中（优化策略中设其真值为 1，无需将其加入链表中），然后递归再次执行单子句策略和分裂策略。如若过程中子句链表中出现空子句（该子句未被删除，而是所含结点数为 0），说明此次探测失败，则需回溯至前一次分裂策略，将该文字的负文字作为单子句加到链表中（优化策略中使该选择文字的真值为 0），继续进行 DPLL 递归算法。函数流程图如图 3-8 所示：

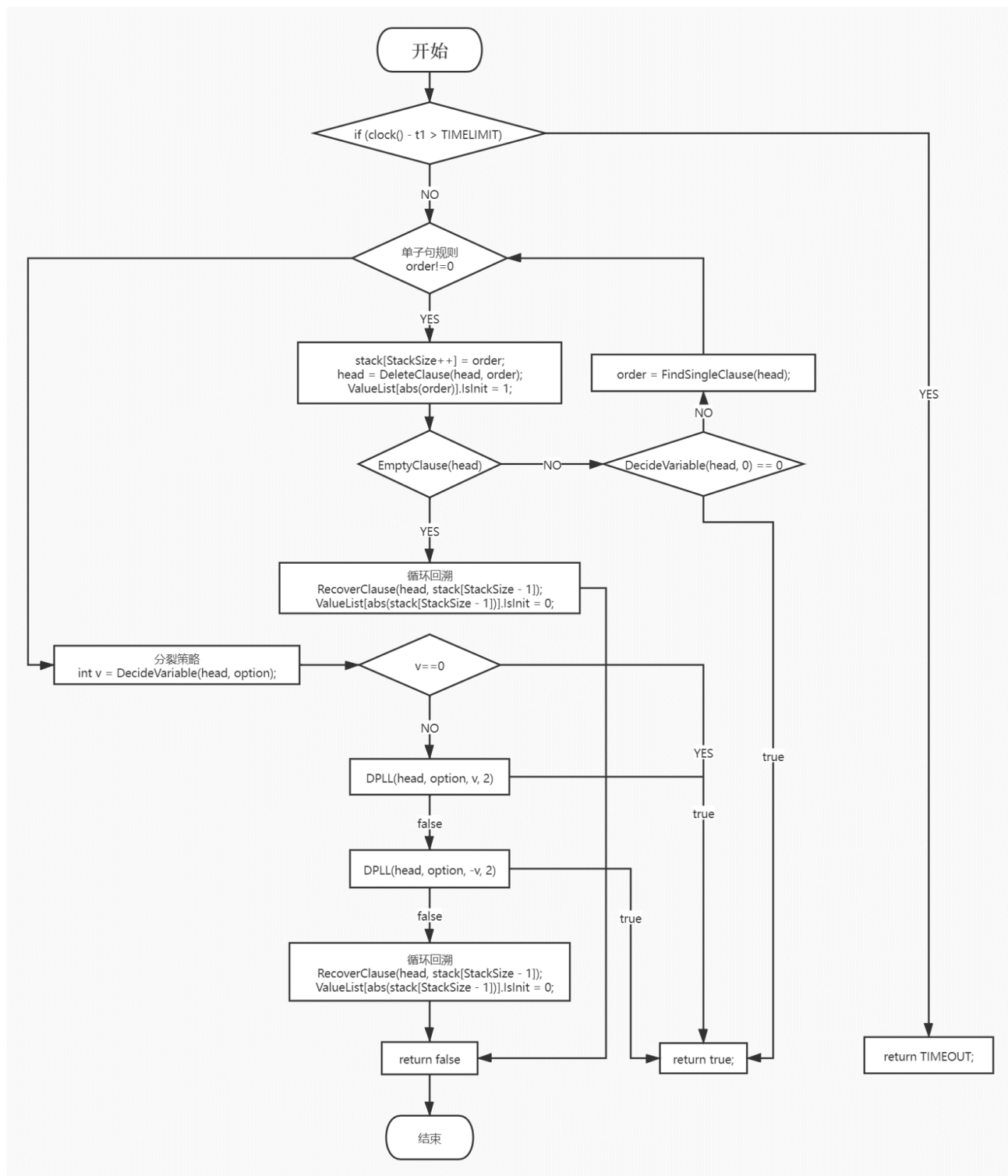


图 3-8 DPLL 函数流程图

3.2.3 数独生成与归约

数独部分包括三个核心算法：生成数独终盘、挖洞算法获取只有唯一解的数独初盘，数独的归约与求解。

(1) 生成数独终盘：

生成数独终盘算法由 LasVegas 算法，DFS 暴力求解算法和 check 辅助函数共同实现。生成数独终盘算法流程图如图 3-9 所示。

利用 LasVegas 算法，首先在空盘中先随机选取 11 个位置放入 1-9 数字（且不违反规则），然后调用 DFS 暴力搜索填满数独，如果未能成功，则继续进行 LasVegas 算法，直至生成一个数独终盘。

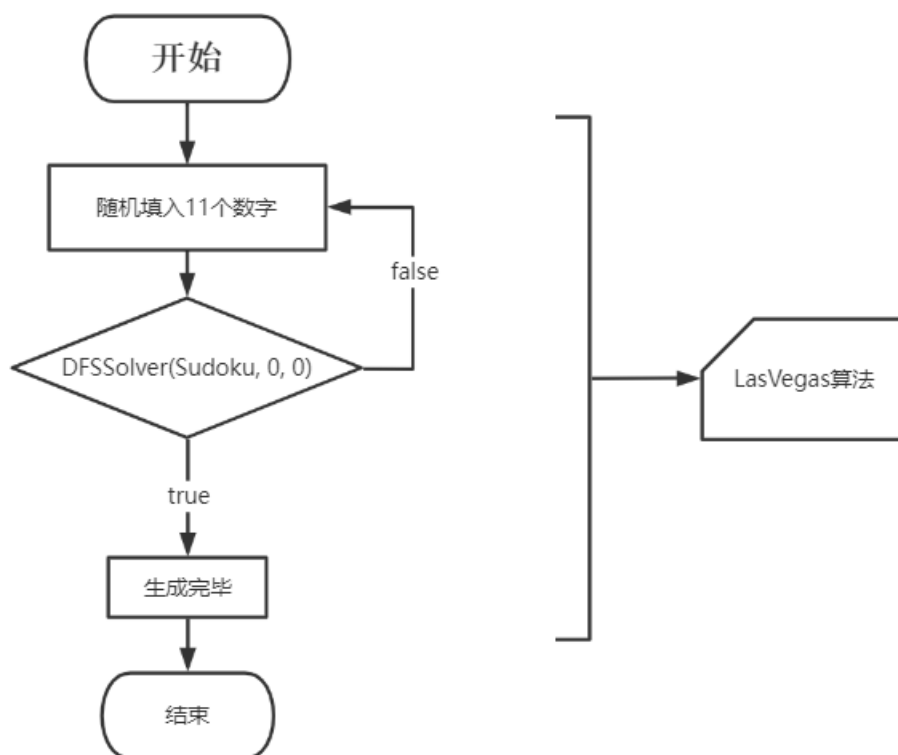


图 3-9 生成数独终盘算法流程图

(2) 挖洞法得到数独初盘：

挖洞法得到数独初盘算法由 DigHoles 函数和 check2，DigHoles 等辅助函数共同实现。挖洞法得到数独初盘算法流程图如图 3-10 所示。

挖洞法算法中，根据难度等级 1-3，对数独施以限制条件。第一个条件是挖洞数目限制，根据难度等级随机生成符合该难度等级的挖洞数目；第二个条件是每行每列最少剩余未挖洞数目，根据难度等级选取，难度越高剩余数目下限越少。同时已尝试过的位置不再尝试，利用剪枝算法优化挖洞过程。在每次尝试挖洞时，将该处数字替换为 1-9 的其他数字，调用 DFS 检查是否有解，如

果有解，说明挖去该洞后将导致多解，因此该洞不能挖去，标记该处并退出挖洞，如果 DFS 检查无多解现象（同时也满足挖洞限制条件），则挖洞成功。

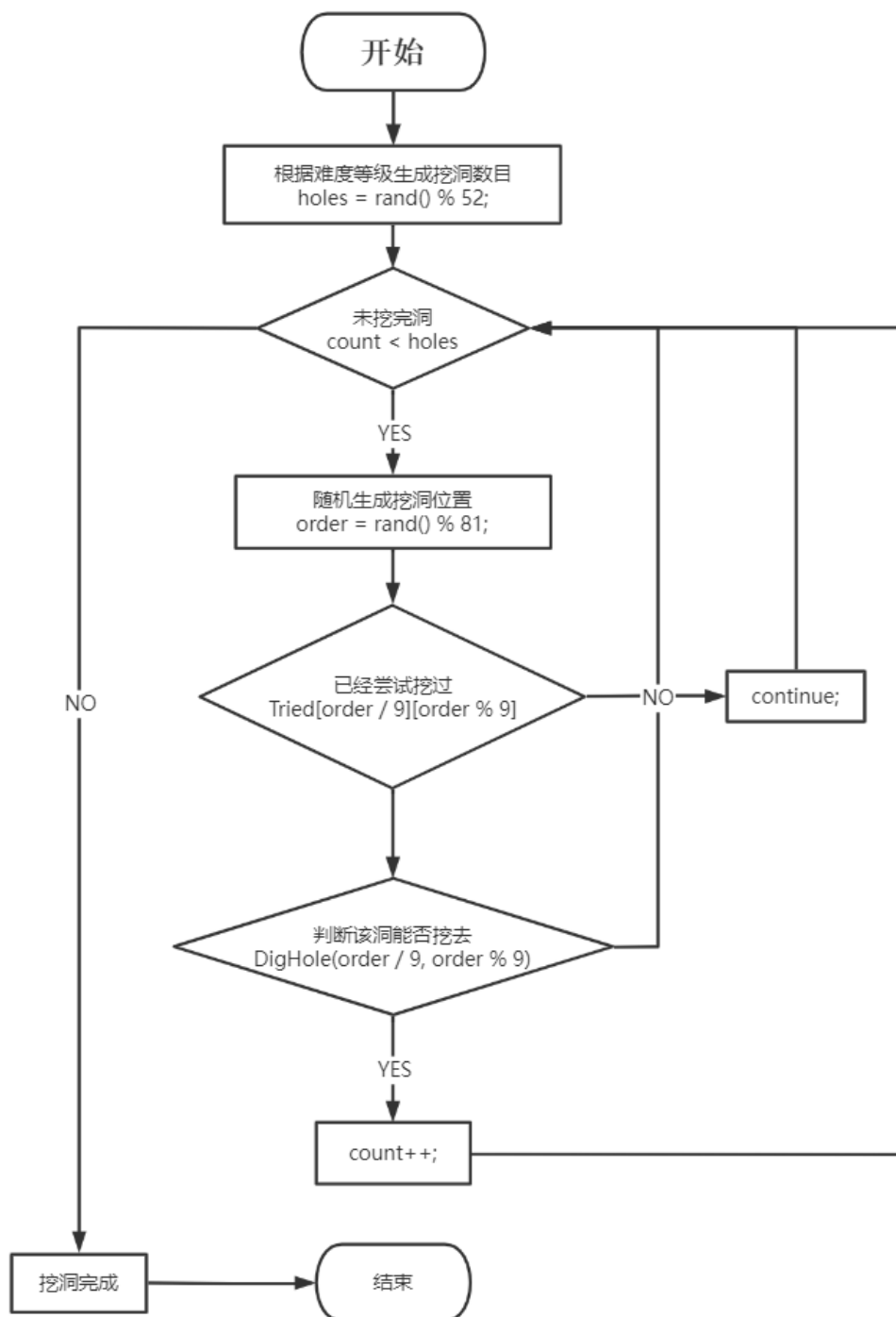


图 3-10 挖洞法得到数独初盘算法流程图

(3) 数独的归约与求解：

数独的归约与求解算法由 SaveSudoku 函数和 SolveSudoku 函数共同实现。数独的归约与求解算法流程图如图 3-11 所示。

数独归约为 CNF 范式主要分为两部分。第一部分是满足数独的基本条件，第二部分是数独初盘中的初始数字。对于第一部分，需要满足行约束，列约束和 3*3 九宫格约束；对于第二部分，仅需添加初始数字对应位置编码的单子句即可。同时，在输出文件时，将数独初盘保存在 CNF 范式前注释中，数独终盘保存在 CNF 范式后注释中。

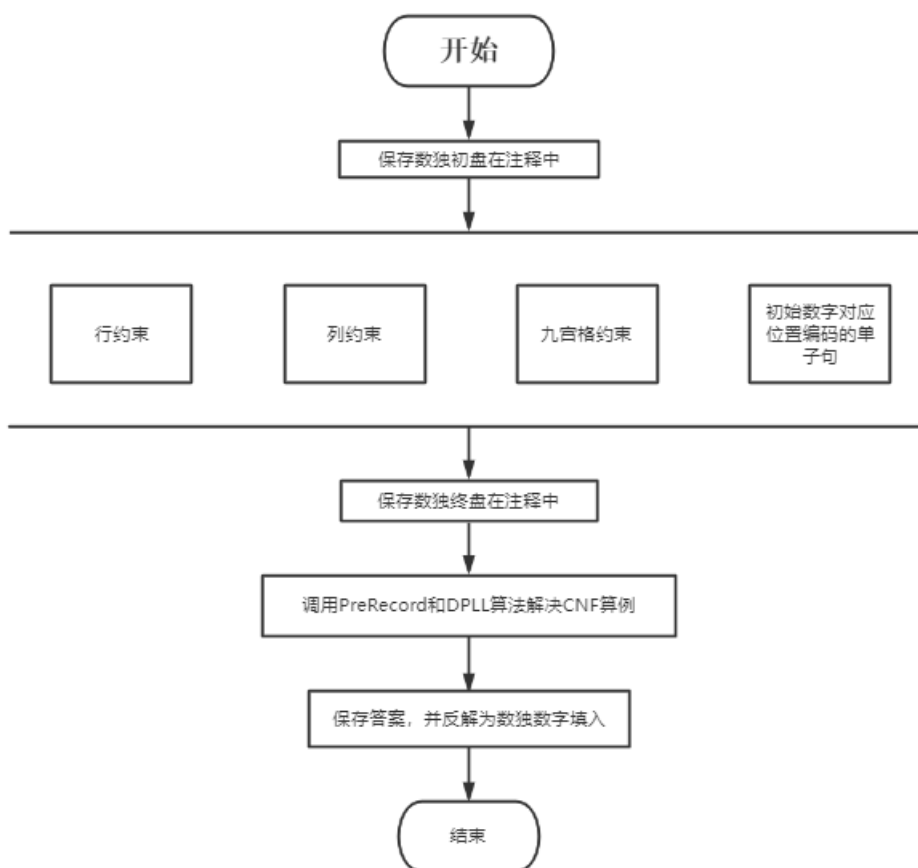


图 3-11 数独的归约与求解算法流程图

3.3 算法优化设计

算法优化设计主要在于优化 SAT 求解器的时间效率，本程序从两个方面对程序进行了优化，优化方案如下。

3.3.1 数据结构优化

数据结构的优化方面，在基础的 DPLL 算法中，所用的数据结构是最基本的 CNF 邻接链表，在 DPLL 过程中，通过删除结点和链表的方式来删除文字和子句，分裂策略中也是将单子句加到链表中，这样每次回溯时难度很大，如果选

取复制链表的方式则时间复杂度较高，整体效率较低；而在对每个子句链表及文字结点添加 flag 表示删除状态后，在 DPLL 过程中的删除和回溯操作可以转变为标记操作，同时利用栈存储每次选取的变元，利于回溯过程的进行，在添加了每个变元对应的正负文字信息链表后，每次进行删除和回溯操作时无需遍历整个链表，而只需针对某个文字遍历其信息链表，节省了时间开销，算法效率得到有效改善。

3.3.2 变元决策策略的优化

变元决策策略的优化方面，有多种策略，如：每次 DPLL 算法后子句中出现频率最高的文字、CNF 邻接链表中第一个未被删除的文字、初始链表中出现次数最多的文字、第一个未被赋真值的变元等。而在未优化前，选取的是 CNF 邻接链表中第一个未被删除的文字，对于大部分 CNF 样例的处理效率不如人意。

经过对算例的求解和效率对比，我发现在不同的算例中，这几种策略的优劣各异，求解效率最高的策略也不尽相同，如 Decide_4 针对数独归约而来的 CNF 文件效率较高，因此要根据某种依据选取最佳的策略。而根据分析，这个依据便是文字的出现频率的高低（以及是否依赖文字出现频率）。而这种依赖性取决于范式中变元出现次数的多样性程度。当多样性程度过高或过低时，则说明变元分布较为均匀，利用文字频率高低来处理效果不佳，而使用随机选取的方式（如 CNF 邻接链表中第一个未被删除的文字）则能够避免这种问题。因此在进行 DPLL 前可利用 PreHandle 函数选取决策策略。

4 系统实现与测试

4.1 系统实现

系统在 windows11 21H2 版本系统下通过 VS Code 编程实现。

该项目中共由三个头文件（fin.h/dpll.h/sudoku.h）和源文件（main.c）组成，框架如图 4-1 所示。

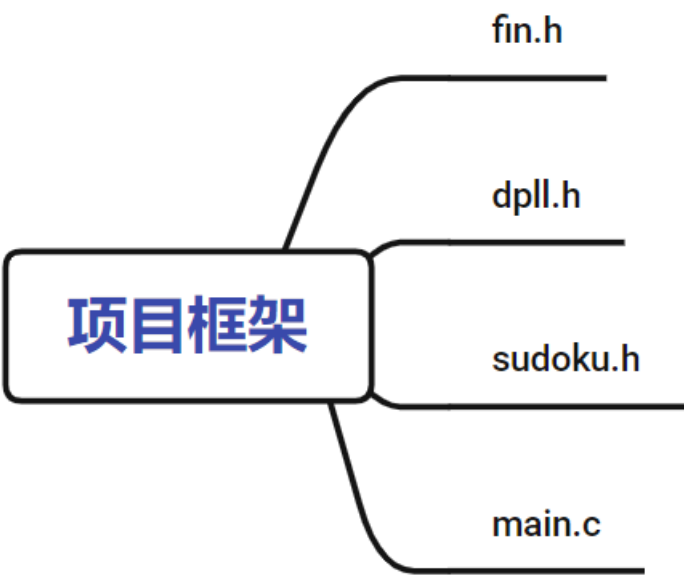


图 4-1 项目文件框架

程序以 main 函数为入口，输出主菜单，通过操作的选择进入 SAT 求解器和数独功能两个功能模块。

4.1.1 文件读取（fin.h）

函数声明	函数功能
int LoadFile(const char *FileName, CNF *cnf);	根据 FileName 文件解析 CNF 范式，存入 cnf 结构体
void DisplayCNF(CNF *cnf);	打印输出 cnf 链表
int SaveCNF(const char *FileName, CNF *cnf);	将 cnf 保存为文件 FileName
int DestoryCNF(CNF *cnf);	销毁 cnf 邻接链表

表 4-1 文件读取文件的函数列表

4.1.2 SAT 求解器（dpll.h）

函数声明	函数功能
void PreRecord(CNF *cnf);	为每个变元创建信息链表
int PreHandle(HeadNode *head, int num);	预处理选择变元决策策略
int FindSingleClause(HeadNode *p);	查找单子句并返回其文字
HeadNode *DeleteClause(HeadNode *head, int val);	删除与 val 相关的链表结点
HeadNode *RecoverClause(HeadNode *head, int val);	恢复与 val 相关的链表结点
int EmptyClause(HeadNode *head);	查找是否有空子句, 若有则返回 true
void Display(HeadNode *head);	打印输出邻接链表
int DPLL(HeadNode *head, int option, int val, int depth);	DPLL 主函数
int SaveResult(const char *FileName, int state, int time);	保存结果到文件中
int JudgeResult(char *FileName);	判断结果是否正确, 正确则返回 true
void SAT(void);	SAT 求解器主函数

表 4-2 SAT 求解器文件的函数列表

4.1.3 数独游戏 (sudoku.h)

函数声明	函数功能
void CreateSudoku(int level);	根据难度 level 生成数独
int LoadSudoku(char *FileName);	加载文件中的数独
void SaveSudoku(CNF *cnf, char *FileName);	将数独归约保存为 CNF 文件
void SolveSudoku(CNF *cnf, char *FileName);	调用 SAT 求解器解决 CNF 文件, 从而解决数独问题
bool DFSSolver(int (*Parameter)[9], int x, int y);	深度优先搜索求解器
bool LasVegas(int n);	LasVegas 算法生成终盘
void DisplaySudoku(int (*Parameter)[9]);	打印输出数独
bool check(int (*Parameter)[9], int x, int y, int num);	检查是否满足数独游戏规则
bool check2(int (*Parameter)[9], int x, int y, int limit);	检查是否满足挖洞规则
bool DigHole(int x, int y);	判断在 (x, y) 挖洞是否合法
void DigHoles(int level);	挖洞法主函数
int coding(int x, int y, int n);	将某位置编码为变元序号
void SuDoKu(void);	数独模块主函数

表 4-3 数独游戏文件的函数列表

4.2 系统测试

4.2.1 SAT 求解器模块测试

1. 满足算例 (sat-20.cnf) 的测试:

```

**          Menu for CNF problem
-----
1. LoadFile          2. DisplayCNF
3. SaveCNF            4. DestroyCNF
5. DPLL               6. ImprovedDPLL
7. JudgeResult        0. Exit
-----
Please Input your option[0-7]:1
Please Input The FilePath:sat-20.cnf
File Open Success!

```

图 4-2 输入文件进行解析界面

```

**          Menu for CNF problem
-----
1. LoadFile          2. DisplayCNF
3. SaveCNF            4. DestroyCNF
5. DPLL               6. ImprovedDPLL
7. JudgeResult        0. Exit
-----
Please Input your option[0-7]:2
The number of literals:20
The number of clauses:91
3:      4      -18      19
3:      3       18      -5
3:     -5       -8     -15

```

图 4-3 打印解析结果界面（未截取完整）

```

**          Menu for CNF problem
-----
1. LoadFile          2. DisplayCNF
3. SaveCNF            4. DestroyCNF
5. DPLL               6. ImprovedDPLL
7. JudgeResult        0. Exit
-----
Please Input your option[0-7]:5
Satisfied!
Time: 0 ms

The answer is set out below:
The value of the variable 1 is 0
The value of the variable 2 is 1
The value of the variable 3 is 1
The value of the variable 4 is 1
The value of the variable 5 is 0
The value of the variable 6 is 0
The value of the variable 7 is 0
The value of the variable 8 is 1
The value of the variable 9 is 1
The value of the variable 10 is 1
The value of the variable 11 is 1
The value of the variable 12 is 0
The value of the variable 13 is 0
The value of the variable 14 is 1
The value of the variable 15 is 1
The value of the variable 16 is 0
The value of the variable 17 is 1
The value of the variable 18 is 1
The value of the variable 19 is 1
The value of the variable 20 is 1

```

图 4-4 优化前 DPLL 算法界面

```

**          Menu for CNF problem
-----
1. LoadFile          2. DisplayCNF
3. SaveCNF            4. DestroyCNF
5. DPLL               6. ImprovedDPLL
7. JudgeResult        0. Exit
-----
Please Input your option[0-7]:6
The Decision:1
Satisfied!
Time before Optimized: 0 ms
Time after Optimized: 0 ms

Optimization failed!

The answer is set out below:
The value of the variable 1 is 0
The value of the variable 2 is 1
The value of the variable 3 is 1
The value of the variable 4 is 1
The value of the variable 5 is 0
The value of the variable 6 is 0
The value of the variable 7 is 0
The value of the variable 8 is 1
The value of the variable 9 is 1
The value of the variable 10 is 1
The value of the variable 11 is 1
The value of the variable 12 is 0
The value of the variable 13 is 0
The value of the variable 14 is 1
The value of the variable 15 is 1
The value of the variable 16 is 0
The value of the variable 17 is 1
The value of the variable 18 is 1
The value of the variable 19 is 1
The value of the variable 20 is 1
    
```

图 4-5 优化后 DPLL 算法界面

```

**          Menu for CNF problem
-----
1. LoadFile          2. DisplayCNF
3. SaveCNF            4. DestroyCNF
5. DPLL               6. ImprovedDPLL
7. JudgeResult        0. Exit
-----
Please Input your option[0-7]:7
The answer read from the file:
-1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20
 1 -1 1
 1 1 1
 1 -1 -1
-1 -1 1
 1 1 1
 1 -1 1
 1 1 -1
 1 1 1
 1 1 -1
    
```

图 4-6 验证求解正确性界面（最后输出 Right!）

2. 不满足算例（unsat-5cnf-30.cnf）的测试：

```

**                               Menu for CNF problem
-----
1. LoadFile                    2. DisplayCNF
3. SaveCNF                     4. DestroyCNF
5. DPLL                        6. ImprovedDPLL
7. JudgeResult                 0. Exit
-----
Please Input your option[0-7]:5
Unsatisfied!
Time: 67 ms

```

图 4-7 不满足算例未优化 DPLL 界面





```

**                               Menu for CNF problem
-----
1. LoadFile                    2. DisplayCNF
3. SaveCNF                     4. DestroyCNF
5. DPLL                        6. ImprovedDPLL
7. JudgeResult                 0. Exit
-----
Please Input your option[0-7]:6
The Decision:3
Unsatisfied!
Time before Optimized: 67 ms
Time after Optimized: 66 ms
Optimization rate: 0.00

```

图 4-8 不满足算例优化后 DPLL 界面

以上过程生成的文件截图如下：

 sat-20.cnf	2017/10/18 22:55	CNF 文件	2 KB
 sat-20.res	2021/10/13 20:41	RES 文件	1 KB
 unsat-5cnf-30.cnf	2011/3/24 3:05	CNF 文件	7 KB
 unsat-5cnf-30.res	2021/10/13 20:44	RES 文件	1 KB

```

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

unsat-5cnf-30.res - 记事本
文件(F)  编辑(E)  格式(O)  视图(V)  帮助(H)
$ 0
t 66

```

4.2.2 数独模块测试

1. 数独游戏界面的测试:

```
***                      Sudoku Game Tips
-----
1. 可选择难度等级(1-3难度递增, 0表示退出)
2. 可选择游戏过程中是否提示冲突(1/0)
3. 中途可通过输入“0 0 0”退出游戏并查看答案
4. 填写完成后如果错误可选择是否继续游戏(1/0)
5. 通过填写横纵坐标和数字来填数, 数字为0时表示清空, 为-1时表示提示数字
-----
Please Input The Difficulty Level:2
Please Choose Whether To Get Tips Before Success:1
```

图 4-9 数独游戏提示界面

```
      1  2  3  4  5  6  7  8  9
+---+---+---+---+---+---+---+---+
1 |   3  4 |   7   |   9   |
2 |   7   |   9   |  2  3 |
3 |  9  8   |   3   |  7   4 |
+---+---+---+---+---+---+---+
4 |   1  2 |   3  4 |   |
5 |  4  7  6 |  9   |   |
6 |  3   |   8   |  2  6  1 |
+---+---+---+---+---+---+
7 |  7   |   3  9 |   5  4 |
8 |   9  5 |   4  8 |  3  1 |
9 |  1   |   3  6  2 |   |
+---+---+---+---+---+---+
Please Input Your Answer:
```

图 4-10 数独游戏进行游戏界面

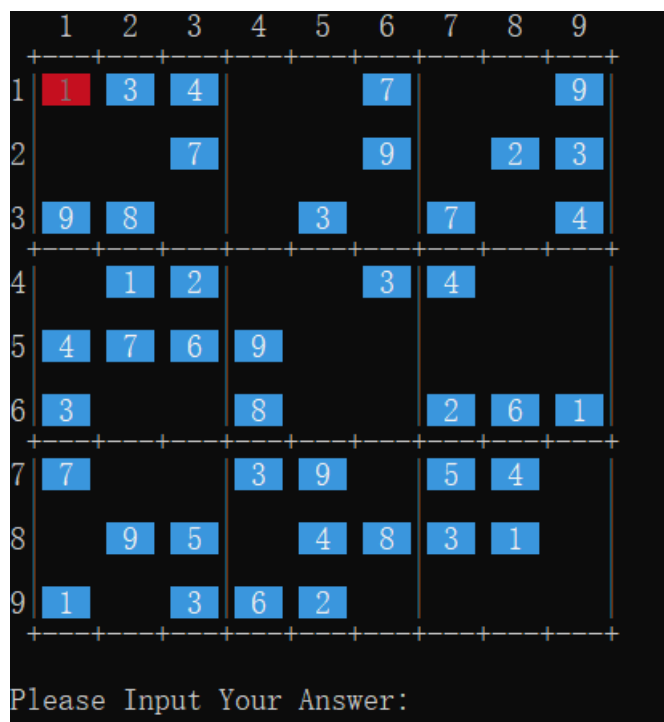


图 4-11 数独游戏填入有冲突的数字界面

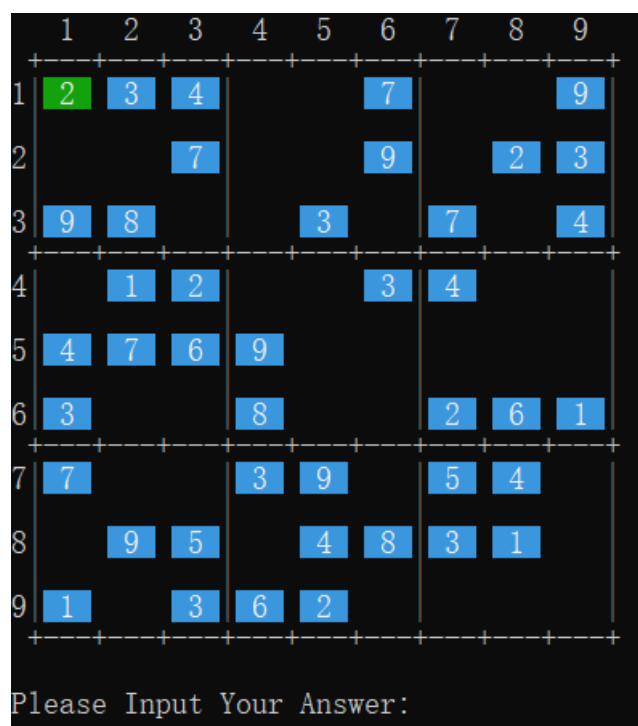


图 4-12 数独游戏填入无冲突数字界面

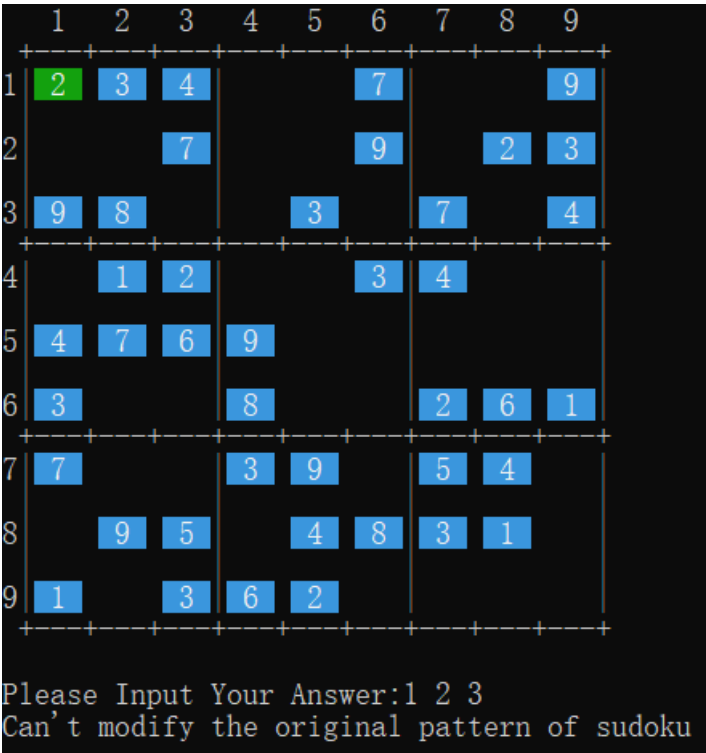


图 4-13 数独游戏填入与数独初盘冲突数字界面

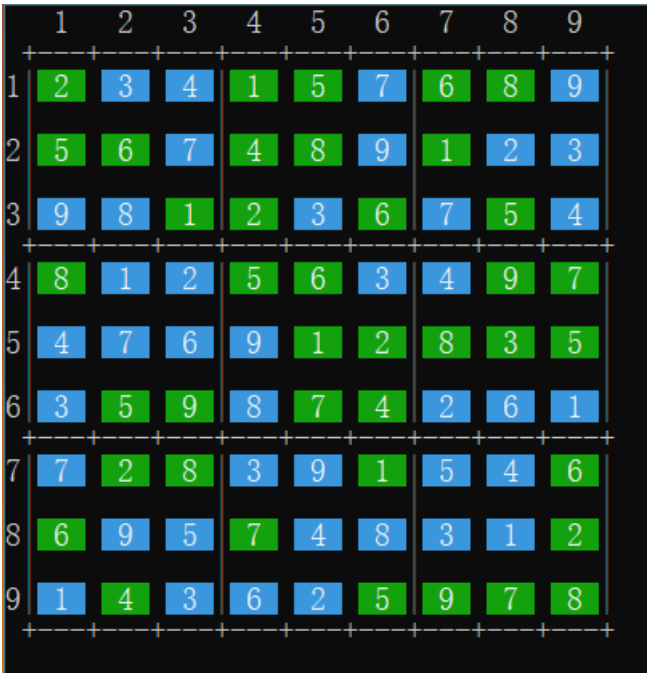


图 4-14 数独游戏结束游戏后界面

2. 数独文件的测试:


```

***      Menu for Sudoku problem
-----
1. CreateSudoku      2. LoadSudoku
3. SolveSudoku       0. Exit
-----
Please Input Your Option[0-3]:1
Please Input The Difficulty Level:2
  1  2  3  4  5  6  7  8  9
+---+---+---+---+---+---+---+---+
1 | 1 |   | 3 | 4 |   | 5 |   |   |
  |   | 5 | 6 |   | 9 |   |   | 2 | 7 |
2 |   |   |   |   |   |   |   |   |   |
  | 7 | 8 | 9 |   | 2 |   | 3 |   | 5 |
3 |   |   |   |   |   |   |   |   |   |
  |   | 4 | 1 |   | 6 |   | 8 |   |   |
4 |   |   |   |   |   |   |   |   |   |
  | 3 | 9 |   |   |   | 8 | 7 |   |   |
5 |   |   |   |   |   |   |   |   |   |
  |   | 6 |   | 3 |   |   |   | 1 | 2 |
6 |   |   |   |   |   |   |   |   |   |
  | 9 |   |   | 6 | 5 |   | 4 | 7 | 1 |
7 |   |   |   |   |   |   |   |   |   |
  |   | 1 |   |   | 8 | 4 | 9 | 3 |   |
8 |   |   |   |   |   |   |   |   |   |
  | 6 | 7 |   | 9 | 3 | 1 | 2 |   |   |
9 |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
The Number Of Digged Holes: 37
Time Used During Creating the Sudoku: 0 ms
Time Used During Saving the Sudoku: 0 ms

```

图 4-15 数独文件生成数独界面

```

***      Menu for Sudoku problem
-----
1. CreateSudoku      2. LoadSudoku
3. SolveSudoku       0. Exit
-----
Please Input Your Option[0-3]:3
  1  2  3  4  5  6  7  8  9
+---+---+---+---+---+---+---+---+
1 | 2 | 4 | 1 | 6 | 7 | 8 | 5 | 9 | 3 |
  | 3 | 6 | 7 | 1 | 5 | 9 | 4 | 8 | 2 |
2 | 5 | 8 | 9 | 2 | 4 | 3 | 6 | 7 | 1 |
  | 4 | 2 | 8 | 3 | 1 | 5 | 7 | 6 | 9 |
3 | 4 | 2 | 8 | 3 | 1 | 5 | 7 | 6 | 9 |
  | 6 | 1 | 5 | 7 | 9 | 2 | 3 | 4 | 8 |
4 | 7 | 9 | 3 | 4 | 8 | 6 | 1 | 2 | 5 |
  | 1 | 3 | 2 | 8 | 6 | 4 | 9 | 5 | 7 |
5 | 8 | 5 | 4 | 9 | 3 | 7 | 2 | 1 | 6 |
  | 9 | 7 | 6 | 5 | 2 | 1 | 8 | 3 | 4 |
6 |   |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
Time Used During Solving the Sudoku: 70 ms

```

图 4-16 数独文件解决数独问题界面

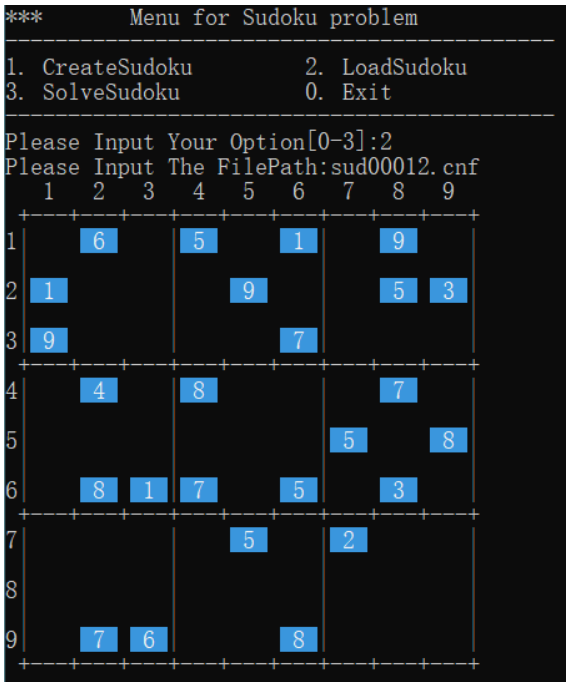



图 4-17 数独文件加载数独文件界面


以上过程生成的文件截图如下：

 Sudoku.cnf

2021/10/13 20:53

CNF 文件


121 KB

 sudoku.h

2021/10/11 22:18

C Header 源文件

18 KB

 Sudoku.res

2021/10/13 20:53

RES 文件

4 KB

Sudoku.cnf - 记事本

文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)

```
k
c Sudoku Puzzle
c
c 0 0 0 6 7 8 5 0 3
c 0 6 7 1 5 0 0 0 2
c 5 8 0 2 4 3 0 0 1
c 4 0 8 0 0 0 7 6 0
c 0 1 5 0 9 0 3 4 8
c 7 9 0 4 0 0 1 2 0
c 1 0 2 0 6 0 0 0 7
c 8 0 0 0 3 7 0 0 6
c 9 0 0 0 0 1 8 3 0
p cnf 729 9844
```

Sudoku.res - 记事本

文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)

```
1
v -1 2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 13 -14 -15 -16 -17 -18 19 -20 -21 -22 -23 -24 -25 -26 -27 -28 -29 -30 -3
232 -233 -234 235 -236 -237 -238 -239 -240 -241 -242 -243 -244 -245 -246 247 -248 -249 -250 -251 -252 -25
1 -442 -443 -444 -445 -446 -447 -448 449 -450 -451 -452 -453 -454 -455 456 -457 -458 -459 460 -461 -462 -4
-651 -652 -653 -654 -655 -656 657 -658 -659 -660 -661 -662 -663 664 -665 -666 -667 -668 -669 -670 -671 67
t 70
```

4.3.3 性能测试表

算例名	算例变元数	子句数与变元数比值	满足(1)/不满足(0)	优化前计算时间(ms)	优化后计算时间(ms)	优化率
ais10.cnf	181	17.4	1	3617	2776	23.30%
sat-20.cnf	20	4.51	1	0	0	\
ais8.cnf	113	13.45	1	97	31	68.00%
7cnf20_90000_90000_7.shuffle d-20.cnf	20	76.6	1	60	53	11.70%
problem1-20.cnf	20	4.55	1	0	0	\
problem11-100.cnf	100	6	1	0	0	\
problem2-50.cnf	50	1.3	1	0	0	\
problem3-100.cnf	100	3.4	1	0	1	\
problem6-50.cnf	50	2	1	0	0	\
problem8-50.cnf	50	6	1	0	0	\
problem9-100.cnf	100	2	1	8	0	→100%
tst_v25_c100.cnf	25	4	1	0	0	\
bart17.shuffled-231.cnf	231	5.05	1	2	2	0.00%
problem5-200.cnf	200	1.6	1	7	6	14.30%
problem12-200.cnf	200	6	1	8	9	— 12.50%
sud00001.cnf	301	9.24	1	20	20	0%
sud00009.cnf	303	9.41	1	91	10	89.00%
sud00012.cnf	232	8.193	1	19	22	— 15.80%
sud00021.cnf	308	9.45	1	440	129	70.70%
sud00079.cnf	301	9.34	1	62	9	85.60%

sud00082.cnf	224	7.87	1	0	7	\
sud00861.cnf	297	9.16	1	11	20	— 81.80%
tst_v200_c210.cnf	200	1.05	1	0	0	\
eh-dp04s04.shuffled-1075.cnf	1075	2.93	1	∞	643	\rightarrow 100%
tst_v10_c100.cnf	10	10	0	0	0	\
unsat-5cnf-30.cnf	30	14	0	59	82	— 39.00%
u- 5cnf_3500_3500_30f1.shuffled -30.cnf	30	14	0	57	83	— 45.60%
u-problem7-50.cnf	50	2	0	0	0	\
u-problem10-100.cnf	100	2	0	0	0	\

注：优化率为“\”的算例代表着系统运算速度快，时间间隔过短小于分度值无法度量。

表 4-4 SAT 求解器性能测试表

5 总结与展望

5.1 全文总结

在本次数据结构课程程序设计中，我实现了以下需求：

- ①解析 CNF 算例文件，并建立对应的数据结构；
- ②实现基于 DPLL 算法的 SAT 求解器，对上述数据结构进行求解；
- ③SAT 求解器结果的验证与保存；
- ④生成数独并将数独归约为 CNF 范式求解；
- ⑤实现具有一定可玩性的数独游戏。

实现过程中的主要工作如下：

- 1) 基础部分：建立合适的存储结构（邻接链表），执行 DPLL 算法；
- 2) 优化代码：优化数据结构，优化变元决策策略，提高运行效率；
- 3) 数独游戏：挖洞法生成数独，打印美观的数独界面，良好的用户体验。

5.2 工作展望

本项目基本达到课程设计目标，但仍存在不足：

- （1）求解器的计算效率仍待提高，求解大型 CNF 算例能力不足，求解挖洞数目较多（乃至挖洞数目最大值）的数独能力不足；
- （2）代码中存在些许逻辑上的混乱，不尽明晰，不易懂，尚需改进；
- （3）数独游戏模块的用户交互体验不佳，输入答案过程繁琐，可利用 Qt 工具实现图形化界面，简洁美观又易操作。

在今后的研究中，将围绕着以上三个方面开展工作，争取将本项目进一步加以完善。

6 体会

课程设计刚开始时，我没有考虑数据结构的优化对 DPLL 算法的改进方案，而只是利用邻接链表进行操作，最终在回溯操作时，对利用栈回溯一筹莫展的我只能选取拷贝的方法来进行递归算法。而这也为我之后优化 DPLL 算法时优化率总是不尽如人意埋下了伏笔。在屡次修改代码决定变元决策策略后，代码的优化效率并未得到很令人眼前一亮的结果。因此，我只得推翻一切，从头开始为每个子句和文字添加 flag 标记，同时构建了文字信息链表以方便进行删除和回溯操作，最后代码的效率得到了显著提高，但是在处理大型算力方面依旧不理想。

在写 DPLL 算法时，我遇到了虽然解出来了，但是答案却是错误的这样一个麻烦。经过一遍遍地调试和打印中间结果，终于发现是一个致命漏洞使得程序提前结束，从而导致结果总是错误的。这是因为我在进行删除标记操作时，将所有阶段被删除的文字和子句均标记为 1，从而使得在回溯时回溯操作不彻底或回溯地过多。在经过思考后，我发现应该将在以 val 为删除标准时，将该阶段所有被删除的结点的 flag 标记为 $\text{abs}(\text{val})$ ，这样在回溯时，只需比较 flag 是否为 $\text{abs}(\text{val})$ 来判定是否需要被标记为未删除状态，从而解决了问题。

在数独游戏方面，在了解了挖洞法生成数独后，我开始尝试写代码。但是在生成数独终盘写 DFS 算法时却遇到了问题，最终在同学的指导下才完成了这个算法。在归约数独时，我因为忽略了九宫格约束而导致归约结果不对，在查看了任务书的提示后方能成功。总之，完成数独的生成与归约很大程度上锻炼了我的编程能力，同时我也体会到了数学与编程之间的联系紧密，享受到了完成这样一个项目时的乐趣。

在数独游戏交互性界面方面，由于对于 Qt 工具等其他可用来写 UI 的工具未作提前了解，导致只能够在 exe 中运行游戏。在网上查阅后，我得知可以通过控制 printf 函数中的参数来实现打印不同颜色的内容，并将此应用到了我的数独界面中，用蓝色表示数独初盘数字，绿色代表无冲突数字，红色代表冲突数字，很大程度上提升了用户交互性体验，同时用户每输入一个数字，便刷新

数独格局，可玩性得以提高。但是这远不及 UI 界面上的简洁和易于操作，因此接下来我要了解 Qt 库等图形化工具，并应用到数独游戏中，早日完善项目。

总之，通过这次课程设计，我对数据结构的掌握能力得到了很大提升，收获颇丰，当然，针对一些不足之处如代码精简度，码风问题，我还需要进一步地学习和改进。

参考文献

- [1] 严蔚敏, 吴伟民. 数据结构 (C 语言版). 清华大学出版社
- [2] 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.
- [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] 熊伟. 可满足性 DPLL 算法研究. 硕士学位论文, 复旦大学: 20070522
- [5] 360 百科: 数独游戏 <https://baike.so.com/doc/3390505-3569059.html>
- [6] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf
- [7] 薛源海, 蒋彪彬, 李永卓. 基于“挖洞”思想的数独游戏生成算法 北京理工大学
- [8] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University, Canada, 2009