华中科技大学

课程设计报告

题目：基于 SAT 的蜂窝数独游戏求解程序

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 2205 班

学　号：U202215510

姓　名：徐新陬

指导教师：李丹

报告日期：2023/9/23

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：$[(t-t_o)/t]*100\%$,其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 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%)

口 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

目录

1 引言

1.1 课题背景与意义

基于 SAT（可满足性问题，Satisfiability Problem）的蜂窝数独游戏求解程序是一个有趣且具有挑战性的课题，它结合了数学、计算机科学和逻辑推理的元素，因此，对于该课题的研究具有悠久的背景和重要的意义。

1.1.1 课题背景

SAT 问题是 NP 完全问题的代表之一，NP 完全问题是一类计算问题，解决 SAT 问题涉及到布尔逻辑的应用，有助于深入研究和理解逻辑系统的性质。在证明理论中，SAT 问题也具有重要的地位，它可以用于验证逻辑命题的可满足性，这对于证明数学定理具有十分重要的意义。同时，SAT 求解算法在自动化和人工智能领域中具有广泛的应用，可以用于自动规划、模型检测、硬件和软件验证、人工智能问题的求解等。在实际应用中，SAT 求解器可以用于解决复杂的调度问题、电路设计、自动规划问题等，在世界范围内，对于 SAT 问题的求解和研究已经具有了广泛的基础。

数独，是一种通过填数满足约束条件的数学谜题，在全球范围内已经广泛传播和流行多年，它有助于提高数学和逻辑推理技能，是一种老少皆宜的游戏，受到了广泛的欢迎，其在世界范围内具有很高的普及度。蜂窝数独是数独游戏的一种扩展版本，它增加了复杂性和游戏性，通过使用蜂窝网格替代标准九宫格，引入了新的规则和约束。因此，开发一个基于 SAT 的蜂窝数独求解程序可以更好地实现 SAT 问题的创新和应用，同时促使人们对于该项研究继续展开大量的研究。

1.1.2 课题意义

SAT 问题作为 NP 完全问题的代表，其对计算复杂性理论产生了深远的影响，通过研究 SAT 问题，人们可以更好地理解计算问题的可解性和困难性，与此同时，SAT 问题的 NP 完全性意味着如果能够有效地解决 SAT 问题，那么也将能够解决许多其他 NP 完全问题。许多组合优化问题可以转化为 SAT 问题的变种，通过将这些问题转化为 SAT 问题，可以应用 SAT 求解算法来寻找最优解，进而可以解决许多实际中的优化问题，这对于社会生产力的提高具有重要

意义。SAT 问题不仅在理论研究中具有深刻的意义，还在各个应用领域中发挥了关键作用。它对计算机科学和数学领域的发展产生了深远的影响，同时也在实际问题的求解和系统验证中提供了有力的工具。因此，SAT 问题一直是计算机科学领域的一个重要课题，其对计算机科学，人工智能领域的发展和社会经济的发展具有十分重要的意义。

1.2 国内外研究现状

关于 SAT 问题的研究，国内外都有广泛的基础，许多研究者和团队致力于改进 SAT 求解算法、应用 SAT 问题解决实际问题以及研究相关领域的理论性问题。

国际上的研究者一直在致力于改进 SAT 求解算法的效率和性能，包括开发新的启发式算法、子句学习策略、分支策略和剪枝技术，基于新的 SAT 求解算法的进步，一些著名的 SAT 求解器包括 MiniSAT、Glucose、CryptoMiniSat 等不断更新迭代，它们在国际 SAT 竞赛中表现出色。

SAT 问题在国际上被广泛应用于各种领域，包括自动规划、硬件和软件验证、模型检测、人工智能、网络设计等，这些应用推动了 SAT 求解算法的发展，并在实际中解决了众多复杂问题。一个典型的样例是组合优化领域，国际上的研究者关注将 SAT 问题与组合优化问题相结合，通过将组合优化问题转化为 SAT 问题的形式，可以应用现有的 SAT 求解器来解决这些问题，从而推动了组合优化领域的发展。

国内的研究者也在 SAT 求解算法的研究方面取得了丰硕的成果，包括改进算法的性能、开发自适应算法、并发求解算法等，中国的研究团队积极参与国际 SAT 竞赛，展示了他们的算法在全球范围内的竞争力，同时在人工智能、自动规划、电路设计和网络优化等领域的研究中也应用了 SAT 求解算法，这些研究推动了中国在这些领域的发展。在教育意义上，SAT 问题在国内的一些高校和研究机构被用作教学和培训工具，帮助培养学生的逻辑思维、问题建模和实践解决能力。

SAT 问题在国内外都是一个活跃的研究领域，研究者不断努力改进算法、应用于实际问题，并推动了相关领域的发展，使其在实际中产生了广泛的应用，对社会科技和经济的发展起到了十分关键的作用。

1.3 课程设计的主要研究工作

基于 DPLL（Davis-Putnam-Logemann-Loveland）算法的 SAT 求解程序是一个重要的课程设计主题，其在计算机科学、人工智能和形式化方法的领域具有十分重要的意义，该课程设计通过研究基于 DPLL 算法求解的 SAT 问题，以及通过 SAT-DPLL 算法求解蜂窝数独等应用，深入理解了 SAT 问题和 DPLL 算法的本质以及有效应用。

该课程设计深入理解了 SAT 问题的本质，即 SAT 问题是一个经典的布尔逻辑问题，其涉及到在给定的布尔表达式中找到一组变量的赋值，使得该表达式为真，同时在求解 SAT 问题的过程中也学习并理解了 CNF（Conjunctive Normal Form）表示和布尔代数的有关知识。

该课程设计学习了 DPLL 算法的工作原理，包括单位子句传播、纯文字传播、分支策略等，了解了如何使用这些技巧来逐步缩小搜索空间并找到解。研究如何改进 DPLL 算法的性能，包括改进分支策略、使用启发式方法、剪枝技巧等，加速程序求解大规模 SAT 实例的能力，对性能的优化也是该课程设计的一个非常重要的研究方向，同时进一步研究了解决 SAT 问题的 CDCL 算法（Conflict-Driven Clause Learning）以为提供更高效的性能。

该课程设计实现了一个基于 DPLL 算法的 SAT 求解程序。使用 C++来完成该程序的编写，其具有一个有效的 CNF 解析器，能够接受 CNF 格式的输入，将解析 CNF 文件格式转换为内部数据结构，并能够输出解（如果存在）或报告问题是不可满足的。

该程序可以用于求解一系列 SAT 问题实例，通过引入测试用 CNF 文件样例以评估该 SAT 求解程序，其中实例包括各种复杂性级别的问题，可以有效衡量程序的性能。在遇到数据量大、数据较难等难以解决的问题时，通过课程研究使 SAT 求解程序更具可扩展性和并行性，提高程序在处理大规模 SAT 问题时的效率。同时程序创建了一个用户友好的界面，使用户能够对求解器轻松输入各种指令，完成求解 SAT 问题，查看解决方案，存储.res 结果。

最后，该课程设计编写了详细的文档和报告，解释你的 SAT 求解程序的设计、实现和性能评估结果，以便帮助其他人理解和使用该 SAT 求解程序。

该课程设计通过完成这些任务，帮助设计者深入了解了 SAT 求解问题以及

DPLL 算法，并获得实际编程和算法优化的经验，帮助设计者提高计算机科学和人工智能技能，并为未来的研究或职业发展打下坚实的基础。

2 系统需求分析与总体设计

2.1 系统需求分析

该系统的用户需求分析如下:

1.用户可以输入符合 CNF 格式的 SAT 问题描述，以便程序能够进行求解，可通过输入文本文件或通过用户界面交互输入。同时，用户能够查看程序的输出结果，包括问题的解（如果存在的话），相关性能分析或报告问题是不可满足的。

2. 程序能够提供 SAT 问题求解的性能信息，例如所选择的分支策略、路径数、传播次数等，以帮助用户评估算法时间效率。同时程序能够处理用户输入的错误或无效数据，并提供有用的错误消息来指导用户改正输入。用户还可以根据需要调整程序的一些参数或设置，以适应不同类型的 SAT 问题和性能要求，充分发挥程序的可扩展性。

3. 程序能够将蜂窝数独游戏问题转化为 SAT 问题，并集成到上面的求解器，可以实现蜂窝数独游戏求解，游戏可玩，具有一定的交互性，用户可进行包括填数，删除，查看答案等游戏操作，或可以选择利用 SAT 求解器直接对蜂窝数独问题进行求解。

系统要实现的目标如下：

该程序需要提供一个用户界面，允许用户输入 SAT 问题，并显示解决方案或结果。其能够使用 DPLL 算法来求解 SAT 问题，当问题可满足时返回一个解，当问题不能满足时给出相应的报告，同时要求给出性能方面的相关数据。程序要能够解析 CNF 格式的输入，将其转换为内部数据结构以供 DPLL 算法使用，通过实现不同的分支策略，包括 VISDS，DLIS，MOMS，CDCL 等策略，以优化求解过程并提高程序性能，同时利用各种剪枝技巧来缩小搜索空间，提高工作效率。在求解完成后，记录并报告关于 SAT 问题的性能统计信息，便于用户了解程序的性能。

在进行蜂窝数独游戏的过程中，要求提供可视化展示数独谜题和解决方案的功能，使用户更容易理解和玩游戏。导入数独谜题时，如果用户输入无效的数独谜题或系统无法找到解决方案，系统提供有用的错误消息或提示，以帮助

用户解决问题。用户进行蜂窝数独游戏时，如果填入的数违背了蜂窝数独游戏的规则，系统将会给用户提供反馈，以便用户更好地重新填写。系统可以计时并记录用户完成数学题的时间，并保存最佳成绩，也可以由用户选择利用 SAT 求解器直接对蜂窝数独问题进行求解。

系统的事物处理流程如下：

进入交互界面，首先由用户选择进行蜂窝数独游戏还是进行 CNF 文件的 SAT 问题求解。进入 SAT 问题求解页面，用户可以输入所要求解的 CNF 文件名，并选择所要使用的求解策略，届时程序将会从 CNF 文件中读入数据并存储到本地的数据结构中，同时使用 DPLL 算法进行求解，求解完成后，程序会将结果返回与 CNF 文件同名的.res 文件，其中记录的内容包含三部分，分别为该 SAT 问题是否满足，该 SAT 问题各个变元的解，以及该 SAT 问题的求解时间，同时会和上次选择的策略进行比较，并得出相应的优化率。

进入蜂窝数独游戏界面，用户输入所要导入的蜂窝数独文件名，并且可以选择是进行蜂窝数独游戏还是直接对该蜂窝数独进行求解，通过函数将蜂窝数独初盘格局转化为 CNF 文件，并集成到上面的求解器进行蜂窝数独游戏求解。在进行蜂窝数独游戏的过程中，用户可以进行填数，删除，直接查看答案等操作，系统会记录用户的用时并且保留最好成绩。

2.2 系统总体设计

基于SAT（可满足性问题）的蜂窝数独游戏求解程序可以分为多个功能模块，以下是一个系统的总体设计，包括各个模块的功能描述和总的模块结构图。

用户交互选择部分(如图2-1所示)：

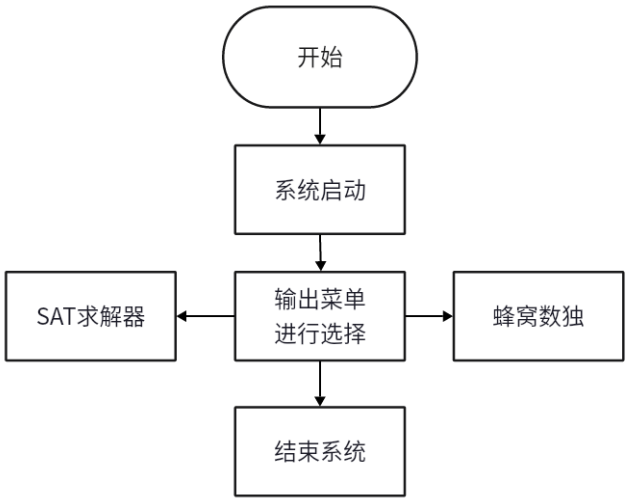

图 2-1 用户交互选择部分图

交互选择模块：用户通过系统所提供的提示来输入指令，使用系统所具有的不同功能。

SAT问题求解部分(如图2-2所示)：

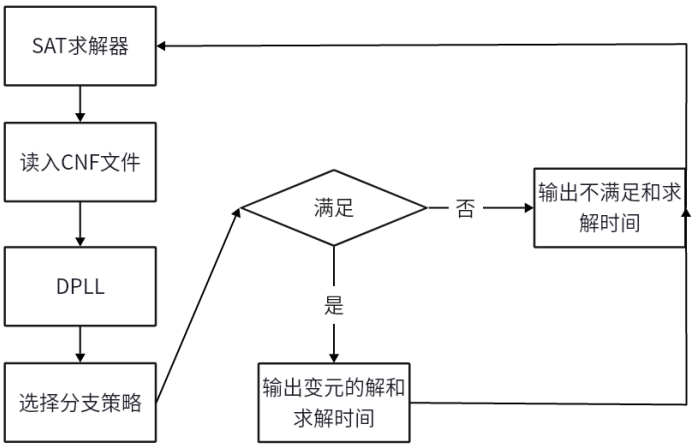

图 2-2 SAT 问题求解部分图

输入处理模块：接收用户输入的SAT问题或从本地导入CNF文件，即导入一系列布尔逻辑表达式，同时解析SAT问题输入，确保输入数据的有效性和合法性。将SAT问题交付给DPLL算法模块。

DPLL算法模块：接收SAT问题并尝试找到满足所有约束条件的解，系统通过实现DPLL算法，解决SAT问题，如果找到解，将解交付给输出处理模块；否则，输出无解。

策略选择模块：用户通过在DPLL算法求解所需分支过程中，选择不同的变元选取策略，通过快速有效的方法解决不同的蜂窝数独谜题。

输出处理模块：解析DPLL算法模块返回的解，存储到.res文件中，并以可视化的方式显示出来，如果DPLL算法模块报告无解，通知用户，同时输出求解所用的时间以及和前一算法相比的优化率。

系统工作流程：

1. 用户导入一个CNF问题。
2. 输入处理模块验证并提交SAT问题给DPLL算法模块。
3. DPLL算法模块使用DPLL算法
4. 用户选择不同的分支策略解决SAT问题，直到找到解或者输出无解。
5. 如果找到解，解析并显示解决方案给用户；如果无解，通知用户。同时

输出求解所用的时间及算法优化率。

6. 用户可以继续输入新的SAT问题或退出程序。

这个系统的核心是DPLL算法和分支变元选择策略，这是是一种用于解决SAT问题的经典方法，输入处理和输出处理模块确保用户可以轻松地与程序交互并获得解决方案，通过在DPLL算法模块中添加优化和启发式算法，提高程序求解效率。

蜂窝数独部分(如图2-3所示)：

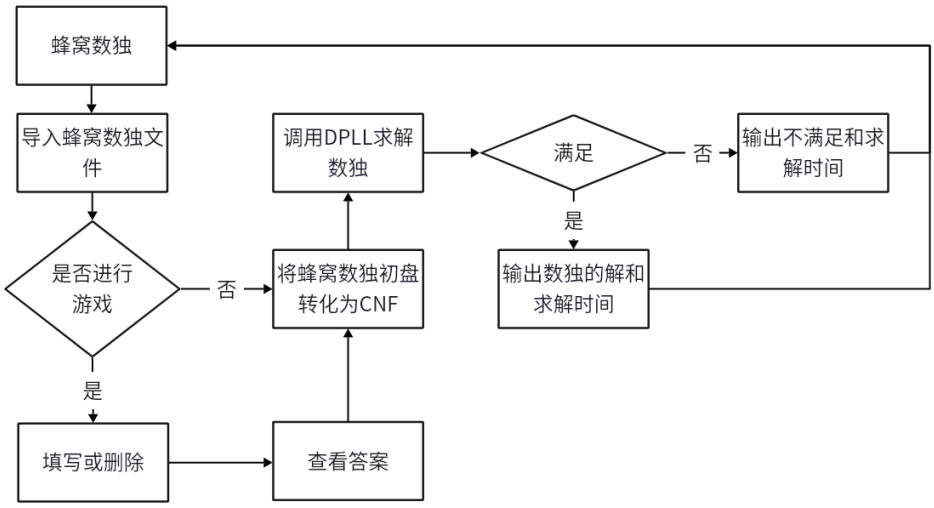

图 2-3 蜂窝数独部分流程图

输入处理模块：接收用户输入或由本地文件导入的数独谜题，通常是一个61格的蜂窝数独盘面，在生成数独初盘过程中，对用户输入进行验证，确保输入数据的有效性和合法性，并将用户输入的合法数独谜题初盘传递给SAT编码模块。

SAT编码模块：将数独谜题通过特定的公式转化为SAT问题，通过设计规则和约束条件，确保SAT问题能够准确表示数独谜题，其通常是一个布尔逻辑表达式，并将其存储在同名的CNF公式中进行编码，最后将编码后的SAT问题传递给SAT求解器模块。

SAT求解器模块：用户可以选择手动填写或使用SAT求解器来解决该蜂窝数独问题。对于手动填写，其包括填入，删除，直接查看答案等功能，用户如果填入的数违背了蜂窝数独游戏的规则，系统将会给用户提供反馈，以便用户更好地重新填写。也可直接使用SAT求解器，接收SAT编码模块传递的SAT问题，并尝试找到满足所有约束条件的解。如果找到解，将解传递给输出处理模块；

否则，报告无解。

输出处理模块：解析SAT求解器模块返回的解，将解转化为数独谜题的形式，重新可视化输出在蜂窝数独棋盘上，以可视化显示给用户，并记录下求解所用的时间，如果SAT求解器输入无解，也将通知用户。

系统工作流程：

1. 用户输入一个或通过本地文件导入一个蜂窝数独谜题。

2. 输入处理模块验证并传递蜂窝数独谜题给SAT编码模块。

3. SAT编码模块将蜂窝数独谜题转化为SAT问题。

4. SAT求解器模块解决SAT问题，找到解或报告无解。

5. 如果找到解，解析并显示数独谜题给用户；如果无解，通知用户。

6. 用户可以继续导入新的蜂窝数独谜题或退出程序。

该部分的核心是SAT编码（CNF约束）和SAT求解，它允许将复杂的数独问题转化为可用SAT求解器解决的形式，对用户友好的输入形式和输出处理确保了用户能够轻松地与程序交互并获得数独谜题的答案。

综合以上过程，总的系统可以如下表示(如图 2-4 所示)：

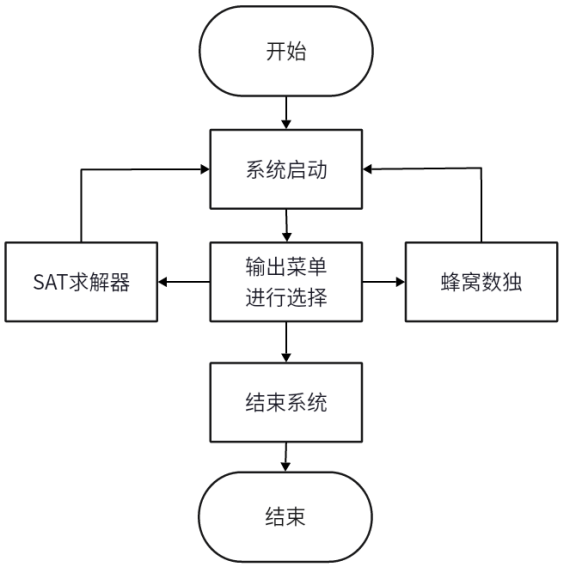

图 2-4 系统总体流程图

3 系统详细设计

3.1 有关数据结构的定义

系统首先要从CNF文件中读取文字个数和子句个数，接下来每行表示一个子句，同时每个子句以0做结尾（0不读入），采用临接链表来存储这样的数据结构，由头结点连接数据节点，表示一个子句的一串文字(如表3-1所示)。

表 3-1 有关数据结构的定义

待处理的数据	包含的数据项	数据类型
总变元个数	变元个数	int Varnum
总子句个数	子句个数	int Clausenum
单个子句	单个子句变元数	int HeadNode->num
变元	变元值	int DataNode->var

该系统的多种数据在数据结构中通过邻接链表关联，如下图所示。由头结点连接数据节点，表示一个子句的一串文字(如图 3-1 所示)。

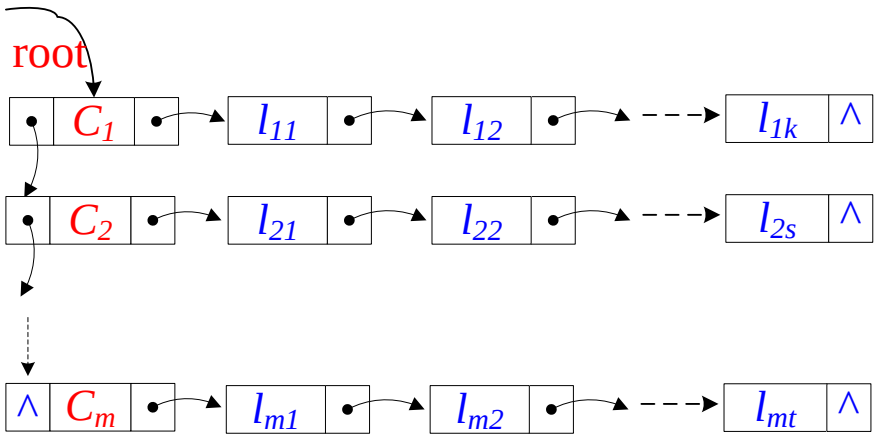

图 3-1 数据结构（临接链表）关联图

对于蜂窝数独游戏，该部分的数据在数据结构中采用二维数组的形式关联，通过二维数组的行列以表示蜂窝数独的坐标，通过二维数组的值以表示蜂窝数独棋盘中的值(如图 3-2 所示)。

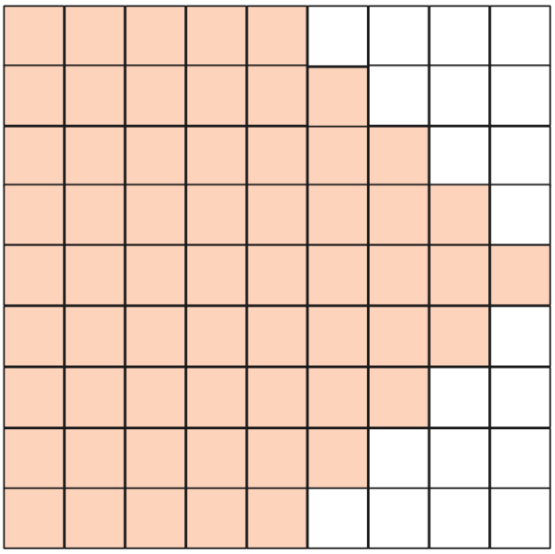

图 3-2 数据结构（蜂窝数独棋盘）图

3.2 主要算法设计

基于SAT的蜂窝数独游戏求解程序可以分为多个功能模块，以下为程序运行设计以及主要算法设计：

通过main进入系统，输入choice，采取如下while循环，保证系统可以进行多次操作处理（如图3-3所示）。

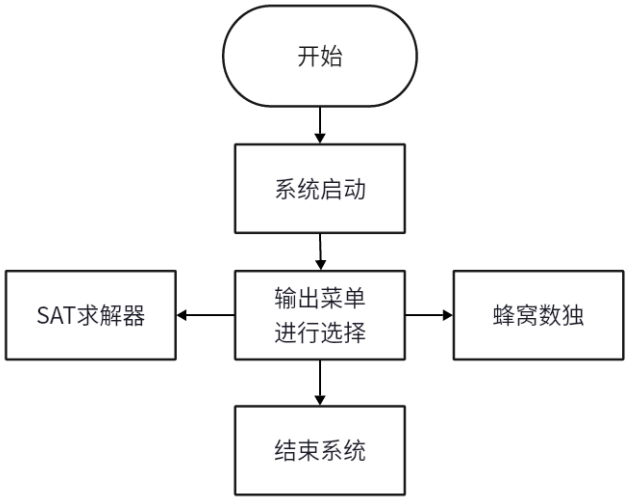

图 3-3 初始用户交互流程图

```
Start();//输出用户交互界面
cin>>choice;
while(choice) {
    if(choice==1) 进入 Hanidoku;
```

if(choice==2) 进入 CNF 文件 SAT 问题求解;

getchar();getchar();

system("cls");

Start();

cin>>choice;

}//while 循环

若输入choice为2，则会进入CNF文件SAT问题求解，首先进入CnfParser模块，从CNF文件中导入数据并存储到临接链表中（如图3-4所示），最后返回链表第一个头结点。

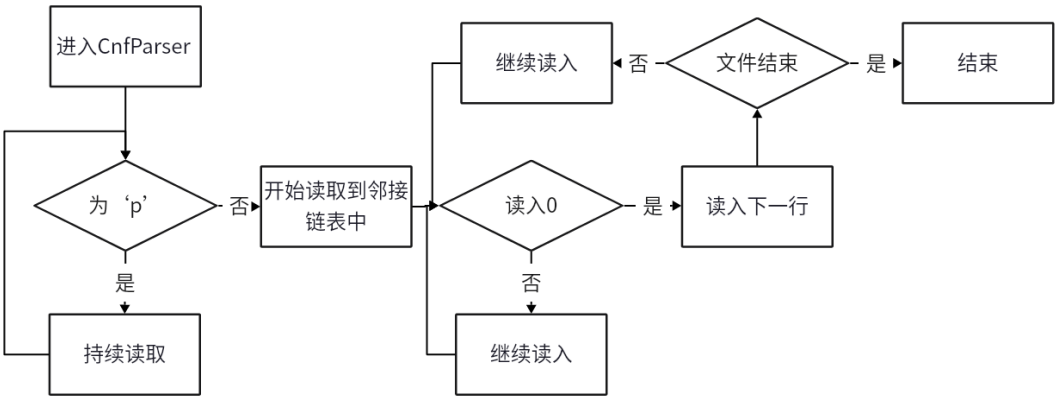

图 3-4 CnfParser 流程图

while (ch != 'p') {

fis.getline(buf, 100);

fis >> ch;

}//直到读到p才开始导入

for (int i = 0; i < ClauseNum; i++){

读入HeadNode

while (temp != 0) {//读入到0表示换行

读入DataNode

}

}//采取二重循环完成读入

成功进行读入以后，则会进入DPLL求解模块，这部分的伪代码如下所示，其核心在于分支变元的选择策略，这里以MOMS选择策略为例，其核心思想是选择变元个数最少得子句中第一个变元进行分支，保证系统的快速求解（如图

3-5所示）。

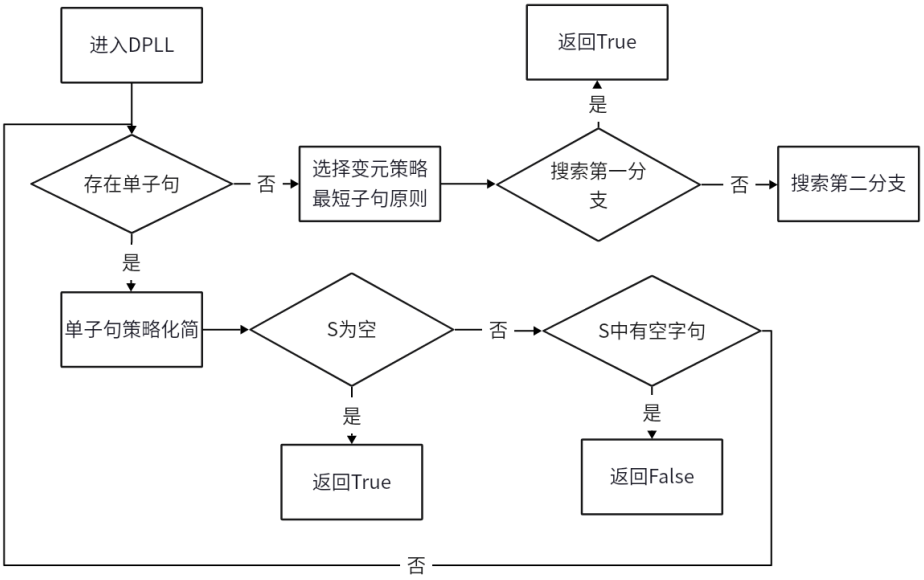

图 3-5 DPLLSolver 流程图

DPLL(*S*) :

/* *S* 为公式对应的子句集。若其满足，返回 TURE；否则返回 FALSE. */

{

while(*S* 中存在单子句) {//单子句传播

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 return(TRUE);

else if (*S* 中有空子句) return（FALSE）;

}//while

基于某种策略选取变元（MOMS）*v*;　　//策略对 DPLL 性能影响很大

if DPLL（*S* ∪*v* ）return(TURE); //在第一分支中搜索

return DPLL(*S* ∪¬*v*);//回溯到对 ***v*** 执行分支策略的初态进入另一分支

}

//MOMS策略

chooseMOMSVariable(HeadNode* LIST) {

将每个子句中未标记变元数初始化为一个大值;

```
    初始化未标记变元;
    for (循环遍历所有子句) {
        记录变元数;
        if (当前子句变元数小于原先最小子句变元数) {
            更新变元数与所选变元;
        }
    }
    return chosen_var;
}
```

若输入choice=1，则会进入Hanidoku模块，系统将会给出提示使你导入想要求解的蜂窝数独文件，在成功导入文件之后，会进入交互界面，用户可以选择进行蜂窝数独游戏或者直接使用SAT求解（如图3-6所示）。

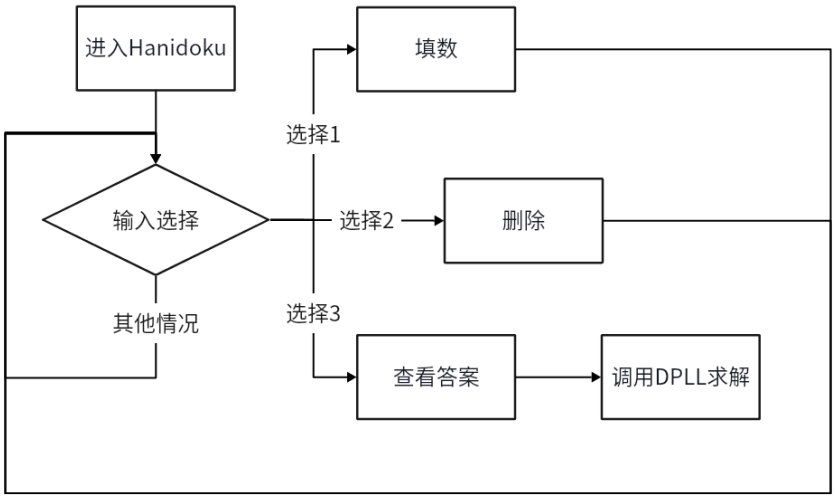

图3-6 Hanidoku选择流程图

```
cin>>choice;
while(choice) {
    输出用户数独交互页面;
    if(choice==1) 选择一个位置进行填数;
    if(choice==2) 将一个位置已经存在的数删除;
    if(choice==3) 直接查看答案
    getchar();getchar();
    system("cls");
    输出用户数独交互页面;
```

cin>>choice;

}//while 循环

若用户直接查看答案，则会调用ToCnf模块以及DPLL求解模块，其中DPLL求解模块与上文相同，而ToCnf模块则是将数独转化为CNF文件，这是求解蜂窝数独问题的关键（如图3-7所示）。

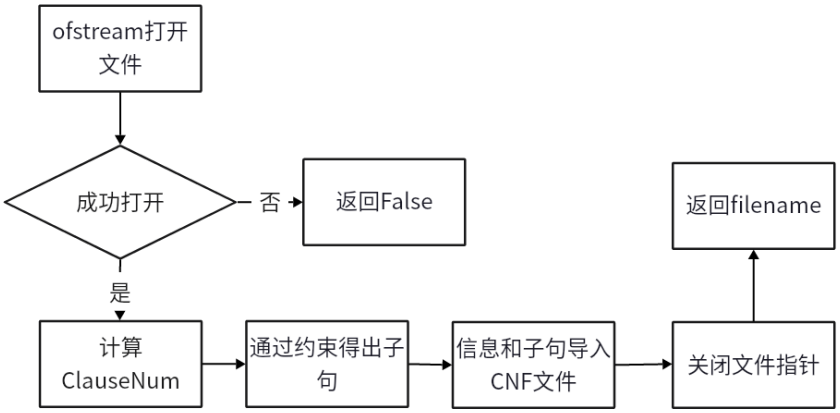

图3-7 ToCnf流程图

ofstream打开输出文件;

if（文件成功打开）{

计算ClauseNum;

引入初盘已经存在的数 61-holes的约束；

每个格子必须要从1~9中填一个数 61条子句；

每个格子只能从1-9中填一个数 61*(8+7+6+5+...+1)=61*36条子句；

行,主副对角线的元素不能重复且同一组中的数需要连续;

//提前将元素归组，便于满足连续的条件

将以上基本信息和子句导入CNF文件；

关闭文件指针；

return filename；

}

以上便为基于SAT的蜂窝数独游戏求解程序的主要算法设计，应兼顾效率与可读性，在用户合法输入的情况下，可实现程序的高效性和健壮性。

4 系统实现与测试

4.1 系统实现

在基于 SAT 的蜂窝数独求解程序的系统实现中，系统使用 C/C++来定义各种数据类型和完成实现各个功能模块。以下是关于程序设计系统实现的一些细节，包括系统的软硬件环境、数据类型定义以及函数说明。

软硬件环境：

编程语言：C/C++。

开发环境：通常的 C/C++编程环境，使用 Dev-cpp 和 Visual Studio 等集成开发工具进行开发。

平台：可以在 Windows 操作系统上运行。

CPU：AMD Ryzen 7 5800H。

数据类型定义：

在系统求解过程中，定义了以下一些主要的数据类型来表示 SAT 问题和蜂窝数独谜题。

1.HeadNode 类型：用于存储 CNF 文件的每个子句的相关信息，采用单项链表的形式

2.DataNode 类型：用于存储 CNF 文件的每个文字的相关信息，采用单项链表的形式，与 HeadNode 类型共同组成解析 CNF 文件的临接链表。

3.SudokuBoard 类型：表示整个数独谜题，是一个二维数组，每个元素是一个 int 类型。

函数说明：

//start模块

void Start();

Start函数输出程序初始的展示页面。

//cnfparser模块

HeadNode* CreateClause(int&, string&);

该函数将CNF文件解析读取并存储为临接链表的形式

//DPLLSolver模块

status DPLL(HeadNode*, consequence*);

主要的DPLL求解算法

HeadNode* IsSingleClause(HeadNode*);

判断该头结点连接的子句是否为单子句。

status IsEmptyClause(HeadNode*);

判断该头结点连接的子句是否为空子句。

HeadNode* ADDSingleClause(HeadNode*, int);

添加新的单子句。

HeadNode* Duplication(HeadNode*);

获取当前的临接链表的副本。

void DeleteHeadNode(HeadNode*, HeadNode*&);

删除该头结点连接的子句并释放空间。

void DeleteDataNode(int, HeadNode*&);

删除数据结点并释放空间。

int chooseDLISVariable(HeadNode*);

DLIS变元选择策略。

int calculateDLISScore(HeadNode*, int);

计算不同变元的DLIS分值。

int isClauseSatisfied(HeadNode*, consequence*);

判断子句集是否满足条件。

int calculateParticipation(HeadNode*, int, consequence*);

计算VSIDS策略中不同变元的参与度分值。

void increaseVariableScore(int);

提高VSIDS策略中某变元的总分值。

void decreaseVariableScore(int);

减少VSIDS策略中某变元的总分值。

void updateLastAssignedTime(int);

更新VSIDS策略中某变元的引用时间。

int calculateVariableScore(int);

计算VSIDS策略中某变元的总分值。

int calculateVSIDSScore(HeadNode*, int, consequence*);

计算VSIDS策略中所有变元的总分值。

int chooseVSIDSVariable(HeadNode*, consequence*);

依据VSIDS策略选择变元。

int chooseMOMSVariable(HeadNode*);

依据MOMS策略选择变元。

//Hanidoku模块

void randomFirstRow(int a0[], int n);

随机生成数独第一行。

void createSudoku(int a[][COL]);

通过挖洞法创建数独棋盘。

void print(int arr[][9]);

打印数独棋盘。

string ToCnf(int a[][COL], int holes);

将已有的数独棋盘通过CNF约束准则转化为CNF文件。

string createSudokuToFile();

将转化后的CNF文件存储。

status SudoDPLL(HeadNode* LIST, conse* result, int VARNUM);

通过DPLL求解速度。

void SudokuShow(conse* result, int VARNUM);

将数独求解结果存储到相应的.res文件中。

int getarraynumlength(int a[9]);

获得数组有效长度。

int getminposi(int a[9]);

获得数组最小正整数。

int getmaxposi(int a[9]);

获得数组最大正整数。

status SuDoKu_Judge(int arr[][9], int row, int col, int n);

用户进行数独游戏时，判断填入的数据是否合法。

bool generateSudokuBoard(int board[9][9]);

挖洞法生成数独棋盘

bool solveSudoku(int board[9][9], int* solutionCount);

确保挖洞法生成的数独棋盘具有唯一的解。

int getimpossiblevalue(int x, int len);

通过约束，获取某组中的不可能取值

void SudokuParser(string filename, int a[][9]);

从数独文件中读取数独

//CDCL模块

int CDCL2solve(string);

使用 CDCL 算法求解数独

函数调用关系：

系统首先调用 Start 模块，生成可供用户进行选择界面，根据用户输入的选择，系统再调用 DPLLSolver 模块或者 Hanidoku 模块。根据在 DPLLSolver 模块中选择不同的策略，调用不同的函数，若选择 CDCL 策略，则会调用 CDCL 模块。

具体函数之间的调用关系将取决于程序的流程，数独模块中也可能在内部调用 DPLL 求解器的函数，以便解决 SAT 问题（如图 4-1 所示）。

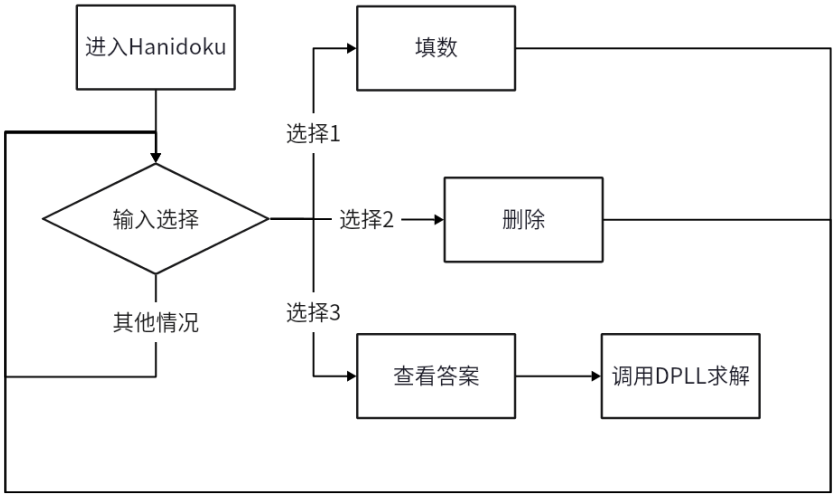

图4-1 Hanidoku模块流程图

4.2 系统测试

在设计基于SAT的蜂窝数独求解程序时，系统测试是确保程序正确性和稳定性的关键步骤，对于本系统，在设计时采用了以下常用的软件测试方法，以

确保系统的鲁棒性和健壮性。

单元测试：测试单个函数或模块的功能。通常通过提供各种输入来测试函数的各种情况。

集成测试：测试不同模块之间的交互和集成。确保模块协同工作正常。

功能测试：测试程序的功能是否符合规范，测试数独求解程序是否在解决蜂窝数独谜题时输出正确的解。

性能测试：测试程序的性能，包括响应时间、内存使用等，使用大型CNF文件来进行性能测试。

边界测试：测试在输入边界情况下程序的行为，可测试极端难度的SAT问题。

回归测试：在程序进行修改后，重新运行先前的测试用例，确保新更改未引入新问题。

在基于SAT的蜂窝数独求解程序中，在开发设计过程中进行了以下一些关键功能模块的测试。

功能模块：CnfParser

1.明确模块的功能和设计目标：

功能：加载CNF文件并存储到临接链表中。

设计目标：确保能够正确读取和解析CNF文件。

2.选取测试数据：

使用包含可满足SAT问题的测试文件和包含非法输入SAT问题的测试文件。对于可满足SAT问题的测试文件，加载并验证是否能够存储所有变元和参数；对于非法输入SAT问题的测试文件，验证加载过程是否能够识别非法输入并报告错误。

3.运行结果（如图4-2，4-3所示）：

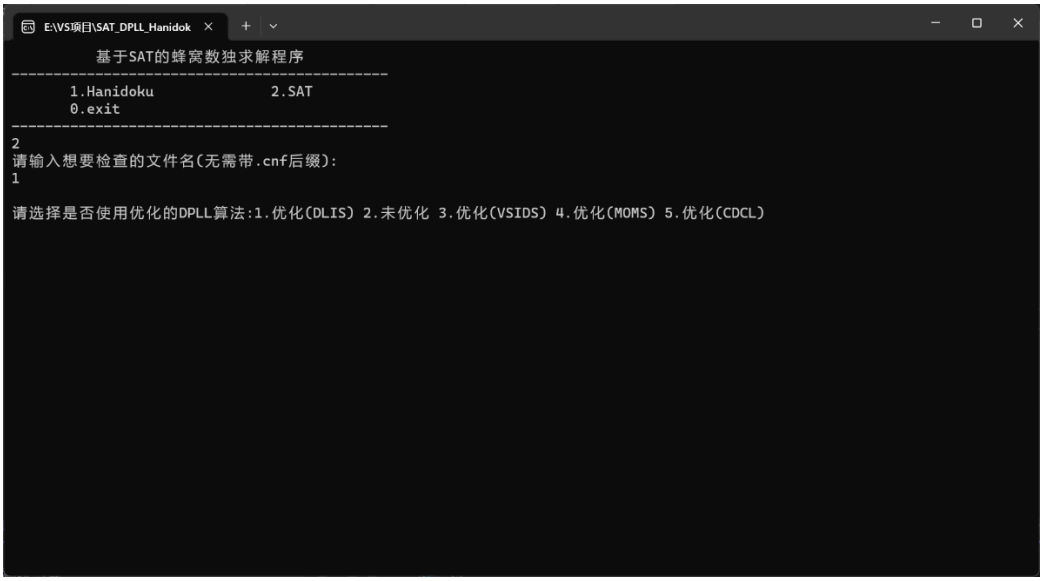

图4-2 输入合法CNF文件图

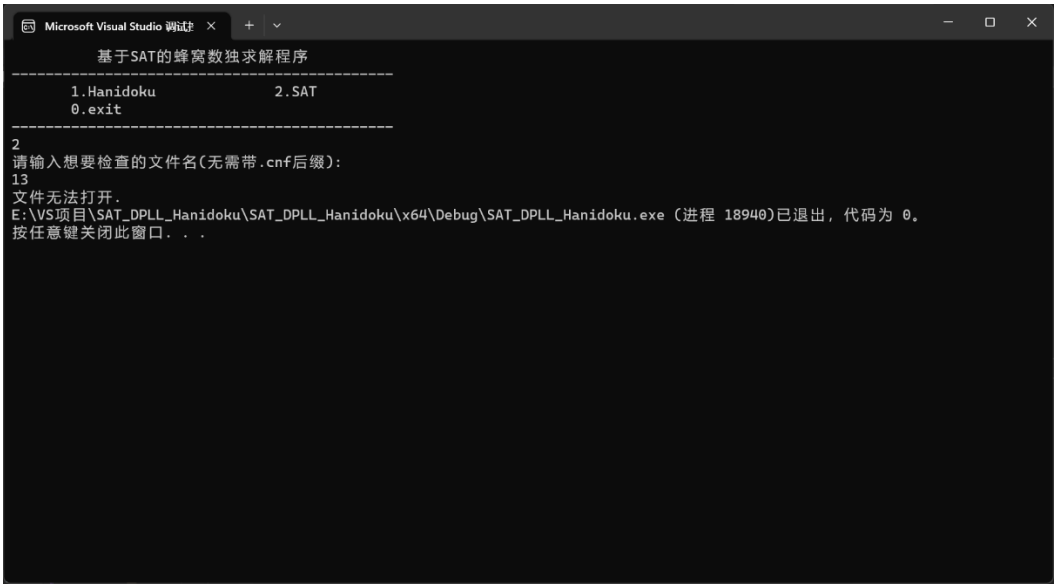

图4-3 输入非法CNF文件图

1. 分析运行结果：

通过分析运行结果可以发现临接链表数据与CNF文件中匹配，确认程序能够正确地检测和报告非法输入。

功能模块：DPLLSolver

1.明确模块的功能和设计目标：

功能：将临接链表存储的SAT问题进行求解。

设计目标：确保求解的正确性，并且能够正确地输出所有变元的解或者不满足信息和求解时间到.res文件中。

2.选取测试数据：

选择包含不同数据级别的SAT问题，包括小型、中型和大型，并要选择最终解为满足和不满足两种类型的SAT问题。对于包含不同数据级别的SAT问题，求解SAT问题并验证SAT问题解的正确性；对于不满足类型的SAT问题，验证DPLL求解器是否能够正确地输出不满足的结果。

3.运行结果（如图4-4，4-5，4-6所示）：

图4-4 DPLL求解图

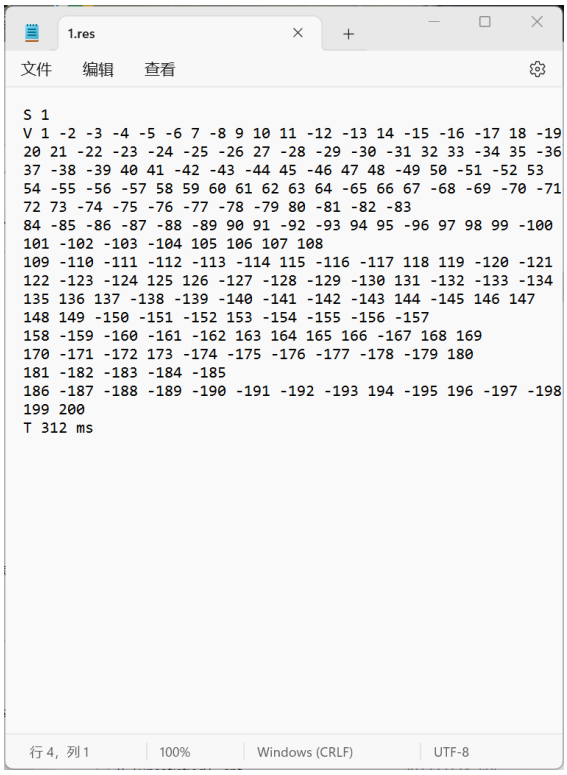

图4-5 满足时结果图

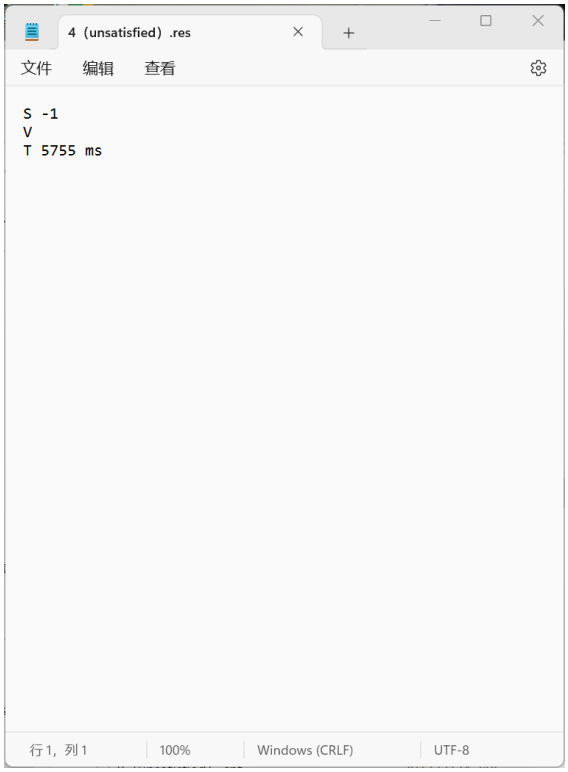

图4-6 不满足时结果图

4.分析运行结果：

可见，对于可满足的CNF文件，系统能够输出正确的解和求解所用时间，对于不满足的CNF文件，系统同样可以输出不满足的情况。

功能模块：chooseMOMSVariable

1.明确模块的功能和设计目标：

功能：依据MOMS策略选择变元进行分支。

设计目标：确保变元选择的正确性，并且能够正确地求解所有变元的分值并由分值高到低选择变元。

2.选取测试数据：

选择包含不同数据级别的SAT问题，包括小型、中型和大型，对于包含不同数据级别的SAT问题，求解SAT问题并验证变元选择的正确性和优越性；对于不满足类型的SAT问题，验证该分支策略是否能够正确地输出不满足的结果。

3.运行结果（如图4-7，4-8所示）：

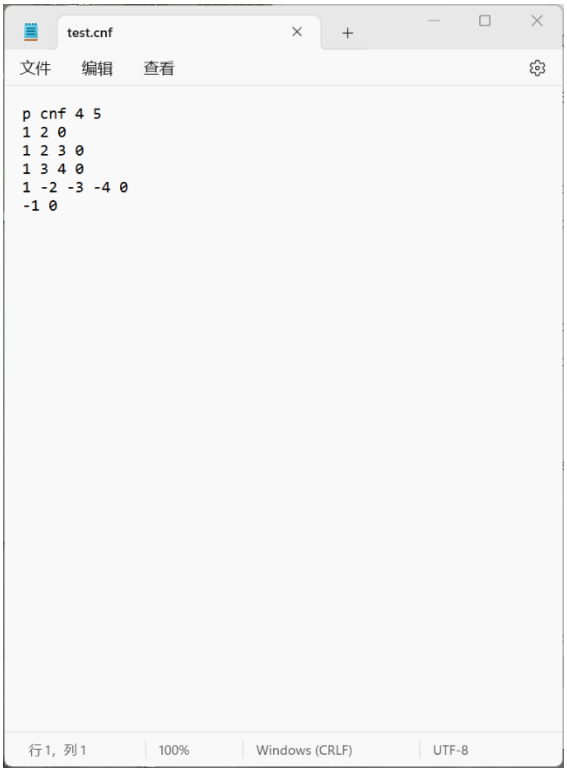

图4-7 测试用CNF文件图

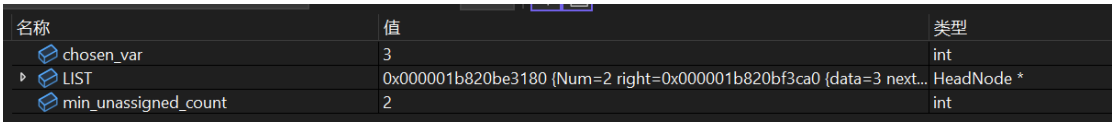

图4-8 变元选择图

4.分析运行结果：

该CNF文件通过单子句法则化简后，在运用MOMS变元选择策略应当选择变元3，经检测，其确实选择了变元3，说明MOMS法则的变元选择过程是正确的。

功能模块：ToCnf

1.明确模块的功能和设计目标：

功能：将当前的蜂窝数独棋盘依据相应约束转化为CNF文件有关的SAT问题进行求解。

设计目标：确保数独棋盘约束关系的正确性，并且能够正确地输出所有变元和子句以及总的变元数和子句数到CNF文件中。

2.选取测试数据：

选择包含不同完成度的蜂窝数独问题，包括导入不同文件数独初盘，以及用户进行输入删除操作过后的数独棋盘（如果填写正确的话）。对于包含不同

完成度的蜂窝数独问题，ToCNF模块应当能够输出不同的文件结果，并要将CNF文件传递给SAT求解系统，以使其求解得到正确的结果。

3.运行结果（如图4-9，4-10所示）：

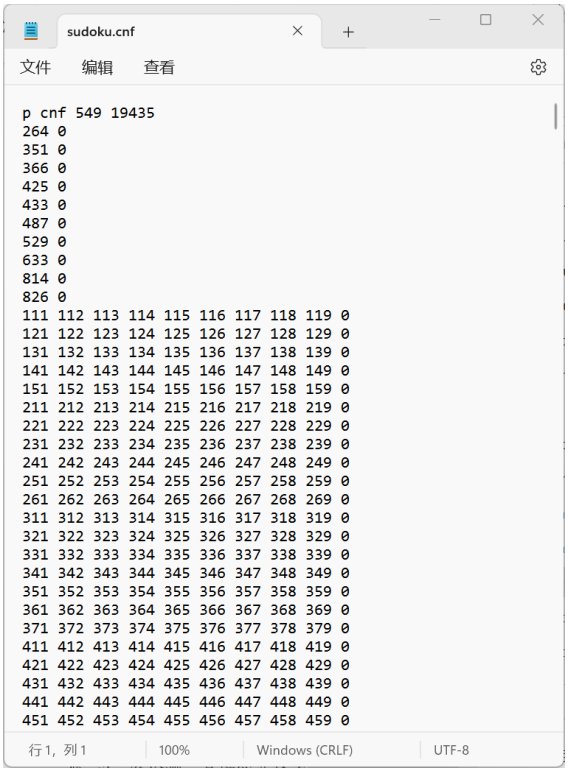

图4-9 蜂窝数独生成CNF文件图

图4-10 蜂窝数独终盘图

4.分析运行结果：

经验证，CNF文件能够正确生成，且在CNF文件正确生成的前提下，SAT

求解器能够求解得到正确的结果，输出数独终盘。

除了以上的模块测试，还应考虑各个模块之间的集成测试，以确保整个系统的协同工作正常，通过这些测试以提高程序的质量和可靠性，并确保其能够正确地解决各种SAT问题和蜂窝数独谜题。

5 总结与展望

5.1 全文总结

设计基于 SAT（可满足性问题）的蜂窝数独问题程序时，进行的主要工作如下：

（1）首先进行了问题建模，需要深入了解蜂窝数字游戏的规则和要求，然后将问题建模为一个布尔可满足性问题，将每个单元格、每个蜂窝以及每个数字都表示为布尔变量，并制定适当的规则和约束条件，以保证解满足数独游戏的规则，即数字每个在每行、主副对角线中都只出现一次且要保持连续。

（2）实现了系统的核心，即DPLL算法和分支变元选择策略，这是是一种用于解决SAT问题的经典方法，输入处理和输出处理模块确保用户可以轻松地与程序交互并获得解决方案，程序通过实现DPLL算法以及在DPLL算法模块中添加优化和启发式算法，提高程序求解效率。

（3）程序创建了一个用户界面，以便用户输入数据和谜题并查看解决方案。这包括图形用户界面和交互界面。最后，也为程序编写了详细的文档，总结了整个项目的工作流程、设计决策、代码结构以及性能评估，这将有助于其他人理解、使用和改进该基于 SAT（可满足性问题）的蜂窝数独问题程序。

5.2 工作展望

在今后的研究中，围绕基于SAT的蜂窝数独求解程序，我打算将在以下几个方面进行优化和改进：

1.进一步提高程序的求解性能，可以尝试学习不同的SAT求解器、启发式算法和优化技术，取长补短，以减少求解时间，特别是在处理更复杂的SAT问题时，能够更快求解且减少内存占用。

2.利用多核处理器或分布式计算资源，实现并行求解，同时开发一个自动难度评估系统，以确定SAT问题的难度级别，以便用户可以根据不同难度的SAT问题选择不同的分支策略，便于程序求解。

3.改进交互界面，使交互界面更加清晰、美观。

4.将机器学习技术与SAT求解器结合，进行机器学习集成，以提高解决复杂问题的能力，使用深度学习来预测最有可能的下一个数字，从而引导求解过程，

提高求解效率。

5.与其他研究者和爱好者交流经验，分享解决方案，推动该领域的研究和发展。

6.探索SAT求解在其他领域的应用，以展示SAT求解器的通用性和实用性。

在设计程序过程中，我了解到基于SAT的蜂窝数独求解程序有着广阔的研究和应用前景，我将对其不断完善改进优化，同时也将不断学习有关知识，将其打造成为一个强大且有重要实际应用作用的程序。

6 体会

对这次程序设计总结一下，我认为设计和实现基于 SAT 的蜂窝数独求解程序是一个复杂的任务，需要仔细考虑许多因素，也遇到了很多困难，但是在程序设计过程中，我也学到了很多相关知识，也积累了很多相关方面的经验。

首先便是需要合理的模块化设计，将整个程序划分为模块，每个模块负责不同的功能，这样可以提高代码的可维护性和可扩展性，以便在出现问题时，可以分模块对代码进行调试，防止对整个程序进行重新检查，极大地提高了程序编写的效率。

通过 CNF 约束将蜂窝数独 SAT 编码是数独项目的核心，必须确保将数独规则和约束条件正确转化为 SAT 问题，这需要深入理解 SAT 编码技巧和数独游戏规则。同时也要考虑于大规模数独问题，SAT 求解可能会非常耗时，应考虑使用启发式算法、剪枝策略或并行计算等技术来提高性能。

提供清晰良好的用户界面，以便用户能够轻松读懂如何操作系统，并验证输入的有效性，要给用户提供清晰的错误信息或明确的求解信息。通过一个直观的蜂窝数独棋盘画面确保解决方案的可读性和美观性。

处理程序可能遇到的各种异常情况，包括用户错误输入，数据结构越界，内存不足等情况，系统需要提供详细的错误信息，以帮助用户理解问题。在测试和调试阶段要进行广泛的测试，通过单元测试、集成测试和性能测试，使用不同难度级别的 CNF 文件进行测试，确保程序在各种情况下都能正常工作。

在程序设计过程中注意编写清晰的文档和注释，以便在开发，维护，调试等过程中快速理解代码的功能和结构，提升项目的可维护性。如果可能的话，获取其余同学的反馈，积极听取他人的意见，以改进程序的功能和用户体验。积极吸取 Github 上有关开源项目，集思广益，改进自己的程序，也可以考虑将个人项目开源，这样其他人也可以学习和改进本程序的代码。

最后，对于一个高效的 SAT 求解器来说，分支策略的选择是至关重要的，因为它直接影响整个 DPLL 算法的性能，进而会影响到程序的性能，无论是 CNF 文件求解还是蜂窝数独游戏的求解，其都与 SAT 求解器的性能直接挂钩，要积极吸取优秀的开源的 SAT 求解器的经验，例如 MiniSat 或 Glucose 等在 SAT 大赛上屡次获奖的求解器。

设计和实现基于 SAT 的蜂窝数独求解程序需要在坚持严谨性和创造性的同时，不断学习和改进，不断以满足用户的需求，不断提高代码的可读性，不断选择优秀的分支策略，优化 DPLL 算法，提高程序的性能。一言以蔽之，那就是：这是一个充满挑战但也非常有趣的项目！

参考文献

[1] 王洪琳. 基于 DPLL 的#SAT 子类算法的研究[D].东北师范大学,2013.

[2] 刘文秀. 可满足问题求解算法 DPLL 的优化技术研究[D].辽宁师范大学,2016.

[3] 李壮. 可满足性问题的相关问题研究[D].吉林大学,2020.

[4] 杨晗. 求解 SAT 问题中分支策略与删除策略的研究[D].西南交通大学,2019.

[5] 陈稳. 基于 DPLL 的 SAT 算法的研究及应用[D].电子科技大学,2011.

[6] 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

[7] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[8] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243

[9] 360 百科：数独游戏 https://baike.so.com/doc/3390505-3569059.html

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

[10] 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.

[11] 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.

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

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

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

[14] 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

[15] 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

附录

Global.h

//定义与引入

```
#ifndef WORKSHOP_GLOBAL_H
#define WORKSHOP_GLOBAL_H

#include <iostream>
#include <fstream>
#include <ctime>
#include <algorithm>
#include <cmath>

using namespace std;
#define TRUE 1
#define FALSE 0
#define ROW 9
#define COL 9
#define NoAnwser -1
#define MaxVarNum 50000
typedef int status;

typedef struct DataNode {
   int data = 0;
   DataNode* next{};
}DataNode;

typedef struct HeadNode {
   int Num = 0;
```

```
    DataNode* right{};
    HeadNode* down{};
}HeadNode;

struct consequence {
    int value = -1;//存真值 真时为true=1，假时为false=0,未标记时为-1
};

struct conse {
    int num = 0;
    int value = -1;//存真值 真时为true-1，假时为false-0
};

struct VariableInfo {
    int variable;
    int score;
    int lastAssignedTime;
};//结构体表示变量的信息分数

//start
void Start();
//cnfparser
HeadNode* CreateClause(int&, string&);
//DPLLSolver
status DPLL(HeadNode*, consequence*);
HeadNode* IsSingleClause(HeadNode*);
status IsEmptyClause(HeadNode*);
HeadNode* ADDSingleClause(HeadNode*, int);
HeadNode* Duplication(HeadNode*);
void DeleteHeadNode(HeadNode*, HeadNode*&);
```

```
void DeleteDataNode(int, HeadNode*&);
void show(struct consequence*);
int chooseDLISVariable(HeadNode* );
int calculateDLISScore(HeadNode*, int);
int isClauseSatisfied(HeadNode*, consequence*);
int calculateParticipation(HeadNode*, int, consequence*);
void increaseVariableScore(int);
void decreaseVariableScore(int);
void updateLastAssignedTime(int);
int calculateVariableScore(int);
int calculateVSIDSScore(HeadNode*, int, consequence*);
int chooseVSIDSVariable(HeadNode*, consequence*);
int chooseMOMSVariable(HeadNode*);
//Hanidoku
void randomFirstRow(int a0[], int n);
void createSudoku(int a[][COL]);
void print(int arr[][9]);
string ToCnf(int a[][COL], int holes);
string createSudokuToFile();
status SudoDPLL(HeadNode* LIST, conse* result, int VARNUM);
void SudokuShow(conse* result, int VARNUM);
int getarraynumlength(int a[9]);
int getminposi(int a[9]);
int getmaxposi(int a[9]);
status SuDoKu_Judge(int arr[][9], int row, int col, int n);
bool generateSudokuBoard(int board[9][9]);
bool solveSudoku(int board[9][9], int* solutionCount);
void generateSudokuPuzzle(int board[9][9], int holes);
int getimpossiblevalue(int x, int len);
void SudokuParser(string filename, int a[][9]);
```

```
//CDCL
int CDCL2solve(string);



#endif //WORKSHOP_GLOBAL_H
```

main.cpp

```
//基于SAT_DPLL的蜂窝数独求解程序
#include "Global.h"

int flag;
conse SudoResult[1000000];//记录最终的真假值(Hanidoku问题)
consequence result[MaxVarNum];//记录最终的真假值(SAT问题)

int main() {
    int t0=0,t1=0;
    Start();
    int choice = 0;
    cin >> choice;
    while (choice) {
        if (choice == 1) {
            cout << "请选择是1.导入数独文件  2.自动生成数独" << endl;
            cin >> flag;
            string filename = createSudokuToFile();
            int VARNUM;
            HeadNode* LIST = CreateClause(VARNUM, filename);
            clock_t StartTime, EndTime;
            StartTime = clock();
            int value = SudoDPLL(LIST, SudoResult, VARNUM);
```

```
        EndTime = clock();
        cout << "T " << (double)(EndTime - StartTime) / CLOCKS_PER_SEC *
1000.0 << " ms\n";
        if (value == 1)
            SudokuShow(SudoResult, VARNUM);
        else
            cout << "该数独无解!";
    }

    else if (choice == 2) {//SAT
        int VARNUM;
        string filename0;
        string filename;
        cout << "请输入想要检查的文件名(无需带.cnf后缀):" << endl;
        cin >> filename0;
        filename = filename0 + ".cnf";
        HeadNode* LIST = CreateClause(VARNUM, filename);
        //尝试采取对每个变量随机赋值
        srand((int)time(0));
        for (int i = 0;i < VARNUM;i++) {
            result[i].value = rand() % 2;
        }
        clock_t StartTime, EndTime;

        cout << "请选择是否使用优化的DPLL算法:1.优化(DLIS) 2.未优化 3.优化
(VSIDS) 4.优化(MOMS) 5.优化(CDCL)" << endl;
        cin >> flag;
        cout << "DPLL正在求解中....." << endl;
        int value = 0;
        StartTime = clock();
```

```
        if (flag == 5) CDCL2solve(filename0);
        else value = DPLL(LIST, result);
        EndTime = clock();
        t1 = t0;
        t0 = EndTime - StartTime;
        cout<<"本次用时"<< (double)(EndTime - StartTime) / CLOCKS_PER_SEC
* 1000.0 << " ms\n";
        if (flag == 5) goto end;
        if (t1 != 0 && t0 != 0)cout << "优化率为" << (double)(t1 - t0) / t0 * 100 <<
"%" << endl;
            //写到文件
        if (flag != 5) {
            string suffix = filename0 + ".res";
            string name = R"(E:\检查文件\)";
            string Outputfile = name + suffix;
            ofstream fos(Outputfile);
            if (!fos.is_open()) {
                cout << "无法打开新文件。\n";
                exit(0);
            }
            //结果
            if (value) {
                fos << "S " << TRUE << endl;
                fos << "V ";
                for (int i = 0; i < VARNUM; i++) {
                    if (result[i].value == TRUE)
                        fos << i + 1 << " ";
                    else if (result[i].value == FALSE)
                        fos << -(i + 1) << " ";
                    else
```

```
                fos << (i + 1) << " ";//剩下一堆可true可false，就索性输出true
            }
            fos << endl;
        }
        else {
            fos << "S " << NoAnwser << endl;
            fos << "V ";
            fos << endl;
        }
        //时间
        fos << "T " << (double)(EndTime - StartTime) / CLOCKS_PER_SEC *
1000.0 << " ms\n";
        fos.close();
      }
      end:
      cout << "求解完成，已经存放到相应.res文件中" << endl;
    }
    else { //其他值不理睬
      cout << "请输入正确的数字! " << endl;
      exit(0);
    }
    getchar();getchar();
    system("cls");
    Start();
    cin>>choice;
  }
}
```

Start.cpp

```
//输出程序开始界面
#include "Global.h"

void Start() {
    printf("          基于SAT的蜂窝数独求解程序\n"
        "-------------------------------------------\n"
        "       1.Hanidoku            2.SAT\n"
        "       0.exit\n"
        "-------------------------------------------\n");
}
```

CnfParser.cpp

```
//cnf文件读取器
#include "Global.h"

int VarNum;
int ClauseNum;

//将cnf文件中的数据导入并存储到临接链表结构中
HeadNode* CreateClause(int& VARNUM, string& filename) {
    string HFilePath = R"(E:\检查文件\)";
    string path = HFilePath + filename;
    ifstream fis(path);
    if (!fis) {
        cout << "文件无法打开.";
        exit(0);
    }
    char ch;
    char buf[100];
    fis >> ch;
```

```
while (ch != 'p') {
    fis.getline(buf, 100);
    fis >> ch;
}
string cnf;

fis >> cnf >> VarNum >> ClauseNum;
fis.get();

//写入临接链表的数据结构中
HeadNode* HEAD = new HeadNode;
HeadNode* headRear = HEAD;
HeadNode* END = new HeadNode;
for (int i = 0; i < ClauseNum; i++) {
    //导入DATA LIST
    int temp;
    fis >> temp;
    //导入第一个DATANODE
    DataNode* front = new DataNode;
    front->data = temp;
    headRear->right = front;
    headRear->Num++;
    //导入后续的DATANODE
    fis >> temp;
    while (temp != 0) {
        DataNode* rear = new DataNode;
        front->next = rear;
        rear->data = temp;
        front = front->next;
        headRear->Num++;
```

```
            fis >> temp;
        }
        front->next = nullptr;
        fis.get();//换行符
        HeadNode* tp = new HeadNode;
        headRear->down = tp;
        END = headRear;
        headRear = headRear->down;
    }
    END->down = nullptr;

    //output link lists
    HeadNode* Phead = HEAD;
    DataNode* front;
    cout << "\n";
    while (Phead != nullptr) {
        front = Phead->right;
        while (front != nullptr) {
            //cout << front->data << " ";
            front = front->next;
        }
        //cout << endl;
        Phead = Phead->down;
    }

    VARNUM = VarNum;
    return HEAD;
}
```

DPLLSolver.cpp

```
//DPLL 求解器
//分别采用了 DLIS VSIDS MOMS 和暴力求解四种策略
//数据通过临接链表存储

#include "Global.h"

extern int VarNum;
extern int ClauseNum;
extern int flag;
struct VariableInfo variable_info[MaxVarNum]; // 存储变量信息的数组

//判断 LIST 中是否有空子句
status IsEmptyClause(HeadNode* LIST) {
   HeadNode* PHead = LIST;
   while (PHead != nullptr) {
      if (PHead->Num == 0)
         return TRUE;
      PHead = PHead->down;
   }
   return FALSE;
}
//寻找单子句,返回单子句的指针
HeadNode* IsSingleClause(HeadNode* Pfind) {
   while (Pfind != nullptr) {
      if (Pfind->Num == 1)
         return Pfind;
      Pfind = Pfind->down;
   }
   return nullptr;
}
```

```
//复制整个 LIST
HeadNode* Duplication(HeadNode* LIST) { //此处检验传参正常，开始检查复制有无逻辑错误
    HeadNode* SrcHead = LIST;
    HeadNode* ReHead = new HeadNode;//新链表的头节点
    ReHead->Num = SrcHead->Num;//复制第一个头节点
    HeadNode* Phead = ReHead;//Phead 创建头节点
    DataNode* ReData = new DataNode;//新链表的数据节点
    DataNode* FirstSrcData = SrcHead->right;//用于创建第一行的第一个数据节点
    ReData->data = FirstSrcData->data;//新链表的第一个数据节点的数值
    Phead->right = ReData;
    for (FirstSrcData = FirstSrcData->next;FirstSrcData != nullptr; FirstSrcData = FirstSrcData->next) {//第一行链表复制完成
        DataNode* NewDataNode = new DataNode;
        NewDataNode->data = FirstSrcData->data;
        ReData->next = NewDataNode;
        ReData = ReData->next;
    }
    //此下行数节点的复制 >=2th
    for (SrcHead = SrcHead->down; SrcHead != nullptr; SrcHead = SrcHead->down) {
        HeadNode* NewHead = new HeadNode;
        DataNode* NewData = new DataNode;
        NewHead->Num = SrcHead->Num;
        Phead->down = NewHead;
        Phead = Phead->down;
        DataNode* SrcData = SrcHead->right;
        NewData->data = SrcData->data;
        Phead->right = NewData;//第一个数据节点
        for (SrcData = SrcData->next;SrcData != nullptr; SrcData = SrcData->next) {//此行剩下的数据节点
            DataNode* node = new DataNode;
            node->data = SrcData->data;
```

```
            NewData->next = node;
            NewData = NewData->next;
        }
        NewData->next = nullptr;
    }
    Phead->down = nullptr;

    return ReHead;
}
//增加单子句
HeadNode* ADDSingleClause(HeadNode* LIST, int Var) { //所加的单子句位于链表的头
    HeadNode* AddHead = new HeadNode;
    DataNode* AddData = new DataNode;
    AddData->data = Var;
    AddData->next = nullptr;
    AddHead->right = AddData;
    AddHead->Num = 1;
    AddHead->down = LIST;
    LIST = AddHead;
    return LIST;
}
//删除数据结点
void DeleteDataNode(int temp, HeadNode*& LIST) {
    DataNode* p;
    HeadNode* pHeadNode0=LIST;
    for (HeadNode* pHeadNode = LIST; pHeadNode != nullptr; pHeadNode = pHeadNode0) {
        pHeadNode0 = pHeadNode->down;
        DataNode* rear = pHeadNode->right;
        while (rear != nullptr) {

            if (rear->data == temp) {//相等删除整行
```

```
            DeleteHeadNode(pHeadNode, LIST);
            break;
        }
        else if (abs(rear->data) == abs(temp)) { //仅仅是绝对值相等铲除该节点
            if (rear == pHeadNode->right) { //头节点删除
                pHeadNode->right = rear->next;
                pHeadNode->Num--;

                p = rear;
                rear = rear->next;
                delete p;
                continue;
            }
            else { //删除普通节点
                for (DataNode* front = pHeadNode->right; front != nullptr; front = front->next)
                    if (front->next == rear) {
                        front->next = rear->next;
                        pHeadNode->Num--;

                        p = rear;
                        rear = rear->next;
                        delete p;
                        continue;
                    }
            }
        }
        if(rear!=nullptr) rear = rear->next;
    }
  }
}
//删除头结点(即删除一行的数据节点)
void DeleteHeadNode(HeadNode* Clause, HeadNode*& LIST) {
```

```
    DataNode* p, * q;
    if (!Clause) return;
    if (Clause == LIST) {

        p = Clause->right;
        while (p != nullptr) {
            q = p;
            p = p->next;
            delete q;
        }

        LIST = Clause->down;

        delete Clause;
    }
    else {
        for (HeadNode* front = LIST; front != nullptr; front = front->down) {
            if (front->down == Clause) {

                p = Clause->right;
                while (p != nullptr) {
                    q = p;
                    p = p->next;
                    delete q;
                }


                front->down = Clause->down;

                delete Clause;
            }
        }
    }
}
```

```
//输出相应结果到屏幕上
void show(struct consequence* result) {
    cout << "V ";
    for (int i = 0; i < VarNum; i++) {
        if (result[i].value == TRUE)
            cout << i + 1 << " ";
        else if (result[i].value == FALSE)
            cout << -(i + 1) << " ";
        else
            cout << (i + 1) << " ";//输出相应结果到.res 文件中
    }
    cout << endl;
}


//DLIS 策略
//计算 DLIS 的分值
int calculateDLISScore(HeadNode* LIST, int var) {
    int score = 0;
    for (HeadNode* pHeadNode = LIST; pHeadNode != nullptr; pHeadNode =
pHeadNode->down) {
        for (DataNode* front = pHeadNode->right; front != nullptr; front = front->next)
{
            if (front->data == var) {
                score += pHeadNode->Num;
            }
        }
    }
    return score;
}
//依据 DLIS 策略选择变量
int chooseDLISVariable(HeadNode* LIST) {
    int maxScore = -1;
```

```
    int chosenVar = -1;

    for (int var = -VarNum; var <= VarNum; var++) {
        int score = calculateDLISScore(LIST, var);
        if (score > maxScore) {
            maxScore = score;
            chosenVar = var;
        }
    }

    return chosenVar;
}


//VSIDS 策略
//检查子句是否被满足
int isClauseSatisfied(HeadNode* Clause, consequence* result) {
    for (DataNode* front = Clause->right; front != nullptr; front = front->next) {
        int literal = front->data;
        int var = abs(literal);

        // 如果变元被分配并且满足子句条件，返回 1
        if (result[var-1].value == (literal > 0 ? 1 : 0)) {
            return 1;
        }
    }

    // 如果子句中没有变元被满足，返回 0
    return 0;
}
//计算参与度
int calculateParticipation(HeadNode* LIST, int var, consequence* result) {
    int participation = 0;
```

```
    for (HeadNode* pHeadNode = LIST; pHeadNode != nullptr; pHeadNode = pHeadNode->down) {
        if (isClauseSatisfied(pHeadNode, result)) {
            // 子句被满足，增加变元的参与解次数
            for (DataNode* front = pHeadNode->right; front != nullptr; front = front->next) {
                if (abs(front->data) == var) {
                    participation++;
                }
            }
        }
    }

    return participation;
}
// 函数用于增加变元的活跃度
void increaseVariableScore(int variable) {
    variable_info[variable].score++;
}
// 函数用于减小变元的活跃度
void decreaseVariableScore(int variable) {
    variable_info[variable].score--;
}
// 函数用于更新变元的上次分配时间
void updateLastAssignedTime(int variable) {
    variable_info[variable].lastAssignedTime = (int)time(NULL);
}
// 函数用于计算变元的得分，包括决策历史，活跃度和上次分配时间的影响
int calculateVariableScore(int variable) {
    int baseScore = variable_info[variable].score;

    // 根据上次分配时间，给予一定的奖励或惩罚
```

```
    time_t currentTime = time(NULL);
    int timeDifference = (int)difftime(currentTime, variable_info[variable].lastAssignedTime);
    int timeFactor = 1; // 根据需要调整奖励或惩罚的程度
    int timeScore = timeDifference * timeFactor;

    // 综合基础分数和时间分数
    int totalScore = baseScore + timeScore;

    return totalScore;
}
// 计算变元的 VSIDS 分值
int calculateVSIDSScore(HeadNode* LIST, int var, consequence* result) {
    // 变元的活跃度分值
    // 1. 变元的出现次数，可以用变元出现的绝对值次数表示
    int frequency = 0;
    for (HeadNode* pHeadNode = LIST; pHeadNode != nullptr; pHeadNode = pHeadNode->down) {
        for (DataNode* front = pHeadNode->right; front != nullptr; front = front->next) {
            if (front->data == var || front->data == -var) {
                frequency++;
            }
        }
    }

    // 2. 变元的参与解的次数，通常是指变元在子句中被满足的次数
    int participation = calculateParticipation(LIST, var, result);

    // 3. 其他因素，如决策历史等
        // 初始化变元的信息数组

    int historyscore = calculateVariableScore(var);
```

```
    // 4. 综合以上因素计算活跃度分值
    int score = (frequency + participation) * historyscore/10;

    return score;
}
//依据 VSIDS 策略选择变元
int chooseVSIDSVariable(HeadNode* LIST, consequence* result) {
    int maxScore = -1;
    int chosenVar = -1;

    for (int i = 1; i <= VarNum; i++) {
        int score = calculateVSIDSScore(LIST, i, result);
        if (score > maxScore) {
            maxScore = score;
            chosenVar = i;
        }
    }
    increaseVariableScore(chosenVar);
    updateLastAssignedTime(chosenVar);
    return chosenVar;
}


//MOMS 策略
int chooseMOMSVariable(HeadNode* LIST) {
    int min_unassigned_count = 99999; // 初始化为一个大值
    int chosen_var = -1;

    for (HeadNode* pHeadNode = LIST; pHeadNode != nullptr; pHeadNode =
pHeadNode->down) {
        int unassigned_count = pHeadNode->Num;
```

```
        if (unassigned_count < min_unassigned_count && unassigned_count > 0) {
            min_unassigned_count = unassigned_count;
            chosen_var = abs(pHeadNode->right->data);
        }
    }

    return chosen_var;
}


//DPLL 求解函数
status DPLL(HeadNode* LIST, consequence* result) {
    //单子句规则简化
    HeadNode* Pfind = LIST;
    HeadNode* SingleClause = IsSingleClause(Pfind);
    while (SingleClause != nullptr) {
        SingleClause->right->data > 0 ? result[abs(SingleClause->right->data) - 1].value
= TRUE : result[abs(SingleClause->right->data) - 1].value = FALSE;
        int temp = SingleClause->right->data;
        DeleteHeadNode(SingleClause, LIST);//删除单子句这一行
        DeleteDataNode(temp, LIST);//删除相等或相反数的节点
        if (!LIST) return TRUE;
        else if (IsEmptyClause(LIST)) return FALSE;
        Pfind = LIST;
        SingleClause = IsSingleClause(Pfind);//回到头节点继续进行检测是否有单子
句
    }
    //分裂策略
    int Var = 0;
    if (flag == 1) Var = chooseDLISVariable(LIST);//选取变元:采取 DLIS 策略，用
headnode 中的 num 代替该策略中的 weight
    if (flag == 2) Var = LIST->right->data;
```

```
    if (flag == 3) {
        for (int i = 1; i <= VarNum; i++) {
            variable_info[i].variable = i;
            variable_info[i].score = 0;
            variable_info[i].lastAssignedTime = (int)time(NULL)-10;
        }
        Var = chooseVSIDSVariable(LIST, result);
    }
    if (flag == 4) Var = chooseMOMSVariable(LIST);
    HeadNode* replica = Duplication(LIST);//存放 LIST 的副本 replica
    HeadNode* temp1 = ADDSingleClause(LIST, Var);//装载变元成为单子句
    if (DPLL(temp1, result)) return TRUE;
    else {
        HeadNode* temp2 = ADDSingleClause(replica, -Var);
        return DPLL(temp2, result);
    }
}
```

Hanidoku.cpp

```
//求解 Hanidoku 有关的函数组
#include "Global.h"

#define CORRECT 0
#define WRONG -1
static int T = 0;
int cols[] = { 5,6,7,8,9,8,7,6,5 };
int maxdis = 9;
extern int flag;
int holes;

//初始化将坐标分组
int groupx[27][9] = {
```

```
    {1,1,1,1,1},
    {2,2,2,2,2,2},
    {3,3,3,3,3,3,3},
    {4,4,4,4,4,4,4,4},
    {5,5,5,5,5,5,5,5,5},
    {6,6,6,6,6,6,6,6},
    {7,7,7,7,7,7,7},
    {8,8,8,8,8,8},
    {9,9,9,9,9},
    {1,2,3,4,5},
    {1,2,3,4,5,6},
    {1,2,3,4,5,6,7},
    {1,2,3,4,5,6,7,8},
    {1,2,3,4,5,6,7,8,9},
    {2,3,4,5,6,7,8,9},
    {3,4,5,6,7,8,9},
    {4,5,6,7,8,9},
    {5,6,7,8,9},
    {5,6,7,8,9},
    {4,5,6,7,8,9},
    {3,4,5,6,7,8,9},
    {2,3,4,5,6,7,8,9},
    {1,2,3,4,5,6,7,8,9},
    {1,2,3,4,5,6,7,8},
    {1,2,3,4,5,6,7},
    {1,2,3,4,5,6},
    {1,2,3,4,5}
};
int groupy[27][9] = {
    {1,2,3,4,5},
    {1,2,3,4,5,6},
    {1,2,3,4,5,6,7},
    {1,2,3,4,5,6,7,8},
    {1,2,3,4,5,6,7,8,9},
```

```
    {1,2,3,4,5,6,7,8},
    {1,2,3,4,5,6,7},
    {1,2,3,4,5,6},
    {1,2,3,4,5},
    {1,1,1,1,1},
    {2,2,2,2,2,1},
    {3,3,3,3,3,2,1},
    {4,4,4,4,4,3,2,1},
    {5,5,5,5,5,4,3,2,1},
    {6,6,6,6,5,4,3,2},
    {7,7,7,6,5,4,3},
    {8,8,7,6,5,4},
    {9,8,7,6,5},
    {1,1,1,1,1},
    {1,2,2,2,2,2},
    {1,2,3,3,3,3,3},
    {1,2,3,4,4,4,4,4},
    {1,2,3,4,5,5,5,5,5},
    {2,3,4,5,6,6,6,6},
    {3,4,5,6,7,7,7},
    {4,5,6,7,8,8},
    {5,6,7,8,9}
};

int impossiblevalue[9];

//获取数组长度
int getarraynumlength(int a[9])
{
    int length = 0;
    for (int i = 0;i < 9;i++) {
        if (a[i] != 0) length++;
    }
```

```
    return length;
}
//获得数组中最小的正整数
int getminposi(int a[9])
{
    int minposi = 10;
    for (int i = 0;i < 9; i++) {
        if (a[i] != 0 && a[i] < minposi) {
            minposi = a[i];
        }
    }
    return minposi;
}
//获得数组中最大的正整数
int getmaxposi(int a[9])
{
    int maxposi = 0;
    for (int i = 0;i < 9; i++) {
        if (a[i] > maxposi) {
            maxposi = a[i];
        }
    }
    return maxposi;
}

//通过挖洞法生成蜂窝数独函数组
//数独之判断填入的数字是否正确
status SuDoKu_Judge(int arr[][9], int row, int col, int n)
{
    int mark_diag1[9], mark_diag2[9], mark_row[9];
    memset(mark_diag1, 0, sizeof(mark_diag1));
    memset(mark_diag2, 0, sizeof(mark_diag2));
    memset(mark_row, 0, sizeof(mark_row));
```

```
int i, j, maxnum = 0, minnum = 0, distan = 0;
int linelength = cols[row], diag1length = 0, diag2length = 0;

for (i = 0; i < cols[row]; i++) {// 判断行重复
   mark_row[i] = arr[row][i];
   if (arr[row][i] == n) {
      return FALSE;
   }
}

distan = getmaxposi(mark_row) - getminposi(mark_row);
if (distan + 1 > cols[row]) return FALSE;//判断行连续

// 判断主,副对角线重复
if (row <= 4) {
   for (j = 0; j < 9; j++) {
      if (j <= 4 && j >= 0) {

         mark_diag1[j] = arr[j][col];diag1length++;
         if (col + j - row >= 0) mark_diag2[j] = arr[j][col + j - row];diag2length++;

         if (arr[j][col] == n) return FALSE;//副
         if (col + j - row >= 0) {
            if (arr[j][col + j - row] == n) return FALSE;//主
         }
      }
      if (j > 4) {

         if (col - j + 4 >= 0) mark_diag1[j] = arr[j][col - j + 4];diag1length++;
         mark_diag2[j] = arr[j][col + 4 - row];diag2length++;


         if (col - j + 4 >= 0) {
```

```
            if (arr[j][col - j + 4] == n) return FALSE;//副
        }
        if (arr[j][col + 4 - row] == n) return FALSE;//主
    }
  }
}
else {
  for (j = 0;j < 9;j++) {
    if (j <= 4 && j >= 0) {

      mark_diag1[j] = arr[j][col + row - 4];diag1length++;
      if (col + j - 4 >= 0) mark_diag2[j] = arr[j][col - 4 + j];diag2length++;


      if (col + j - 4 >= 0) {
        if (arr[j][col + j - 4] == n) return FALSE;//主
      }
      if (arr[j][col - 4 + row] == n) return FALSE;//副
    }
    if (j > 4) {

      if (col - j + row >= 0) mark_diag1[j] = arr[j][col - j + row];diag1length++;
      mark_diag2[j] = arr[j][col];diag2length++;


      if (arr[j][col] == n) return FALSE;//主
      if (col - j + row >= 0) {
        if (arr[j][col - j + row] == n) return FALSE;//副
      }
    }
  }
}
```

```
    // 判断对角线连续
    distan = getmaxposi(mark_diag1) - getminposi(mark_diag1);
    if (distan + 1 > diag1length) return FALSE;

    distan = getmaxposi(mark_diag2) - getminposi(mark_diag2);
    if (distan + 1 > diag2length) return FALSE;



    return TRUE;
}
//随机生成第一行数
void randomFirstRow(int a[], int n) {
    srand((int)time(0));
    for (int i = 0; i < n; ++i) {
        a[i] = rand() % 9 + 1;
        int j = 0;
        while (j < i) {
            if (a[i] == a[j] || a[j] - a[i] >= n || a[j] - a[i] <= -n) {
                a[i] = rand() % 9 + 1; //generate nine random number
                j = 0;
            }
            else { j++; }
        }
    }
}
// 使用递归和回溯生成数独棋盘
bool generateSudokuBoard(int board[9][9]) {
    for (int row = 0; row < 9; row++) {
        for (int col = 0; col < cols[row]; col++) {
            if (board[row][col] == 0) {
                for (int num = 1; num <= 9; num++) {
                    if (SuDoKu_Judge(board, row, col, num) == TRUE) {
                        board[row][col] = num;
```

```
                if (generateSudokuBoard(board)) {
                    return true;
                }

                board[row][col] = 0; // 回溯
            }
        }
        return false; // 所有数字都尝试过，无解
    }
  }
  return true; // 所有格子都已填满，生成成功
}
//生成数独
void createSudoku(int a[][COL]) { //生成数独
  randomFirstRow(a[4], 9);//随机生成最长的一行
  generateSudokuBoard(a);//递归生成后 i 行
}
//枚举法求解数独
bool solveSudoku(int board[9][9], int* solutionCount) {
  for (int row = 0; row < 9; row++) {
    for (int col = 0; col < cols[row]; col++) {
      if (board[row][col] == 0) {
        for (int num = 1; num <= 9; num++) {
          if (SuDoKu_Judge(board, row, col, num)) {
            board[row][col] = num;

            if (solveSudoku(board, solutionCount)) {
              if (*solutionCount > 1) {
                return true; // 多个解，退出
              }
```

```
                }

                board[row][col] = 0; // 回溯
            }
        }
        return false; // 无解
    }
  }
  (*solutionCount)++;
  return true; // 解决成功
}
//挖洞生成数独
void generateSudokuPuzzle(int board[9][9], int holes) {
  srand((int)time(NULL));
  while (holes > 0) {
    int row = rand() % 9;
    int col = rand() % cols[row];
    if (board[row][col] != 0) {
      int temp = board[row][col];
      board[row][col] = 0;

      int copyBoard[9][9];
      for (int i = 0; i < 9; i++) {
        for (int j = 0; j < cols[i]; j++) {
          copyBoard[i][j] = board[i][j];
        }
      }

      int solutions = 0;
      if (!generateSudokuBoard(copyBoard)) {
        board[row][col] = temp; // 恢复原数字
      }
```

```
            else {
                solutions = 0;
                if (solveSudoku(copyBoard, &solutions)) {
                    if (solutions == 1) {
                        holes--;
                    }
                    else {
                        board[row][col] = temp; // 恢复原数字
                    }
                }
            }
        }
    }
}

//画出数独棋盘
void print(int arr[][9]) {
    int i, j, a, b;
    //31-1-代入 grid
    a = 0;
    b = 0;
    for (i = 0;i < 19;i++) {
        //边界
        if (i % 2 == 0) {
            if (i == 0 || i == 18) {
                printf("         ");
                if (i == 0) {
                    for (int j = 0;j < 5;j++) {
                        printf("/ \\ ");
                        if (j == 4) printf("\n");
                    }
                }
                else if (i == 18) {
```

```
        for (int j = 0;j < 5;j++) {
            printf("\\ / ");
        }
    }
}
else if (i == 2 || i == 16) {
    printf("       ");
    if (i == 2) {
        for (int j = 0;j < 6;j++) {
            printf("/ \\ ");
            if (j == 5) printf("\n");
        }
    }
    else if (i == 16) {
        for (int j = 0;j < 6;j++) {
            printf("\\ / ");
            if (j == 5) printf("\n");
        }
    }

}
else if (i == 4 || i == 14) {
    printf("     ");
    if (i == 4) {
        for (int j = 0;j < 7;j++) {
            printf("/ \\ ");
            if (j == 6) printf("\n");
        }
    }
    else if (i == 14) {
        for (int j = 0;j < 7;j++) {
            printf("\\ / ");
            if (j == 6) printf("\n");
        }
```

```
        }


    }
    else if (i == 6 || i == 12) {
        printf("    ");
        if (i == 6) {
            for (int j = 0;j < 8;j++) {
                printf("/ \\ ");
                if (j == 7) printf("\n");
            }
        }
        else if (i == 12) {
            for (int j = 0;j < 8;j++) {
                printf("\\ / ");
                if (j == 7) printf("\n");
            }
        }
    }
    else if (i == 8 || i == 10) {
        printf(" ");
        if (i == 8) {
            for (int j = 0;j < 9;j++) {
                printf("/ \\ ");
                if (j == 8) printf("\n");
            }
        }
        else if (i == 10) {
            for (int j = 0;j < 9;j++) {
                printf("\\ / ");
                if (j == 8) printf("\n");
            }
        }
    }
}
```

```
//空穴
else if (i % 2 != 0) {
    if (i == 1 || i == 17) {
        b = 0;
        printf("        ");
        for (j = 0;j < 11;j++) {
            if (j % 2 == 0) {
                printf("| ");
            }
            else {
                if (arr[a][b] != 0) printf("%d ", arr[a][b]);
                else printf("· ");
                b++;
            }
            if (j == 10) printf("\n");
        }
        a++;
    }
    else if (i == 3 || i == 15) {
        b = 0;
        printf("      ");
        for (j = 0;j < 13;j++) {
            if (j % 2 == 0) {
                printf("| ");
            }
            else {
                if (arr[a][b] != 0) printf("%d ", arr[a][b]);
                else printf("· ");
                b++;
            }
            if (j == 12) printf("\n");
        }
        a++;
```

```
}
else if (i == 5 || i == 13) {
    b = 0;
    printf("    ");
    for (j = 0;j < 15;j++) {
        if (j % 2 == 0) {
            printf("| ");
        }
        else {
            if (arr[a][b] != 0) printf("%d ", arr[a][b]);
            else printf("· ");
            b++;
        }
        if (j == 14) printf("\n");
    }
    a++;
}
else if (i == 7 || i == 11) {
    b = 0;
    printf("  ");
    for (j = 0;j < 17;j++) {
        if (j % 2 == 0) {
            printf("| ");
        }
        else {
            if (arr[a][b] != 0) printf("%d ", arr[a][b]);
            else printf("· ");
            b++;
        }
        if (j == 16) printf("\n");
    }
    a++;
}
else if (i == 9) {
```

```
            b = 0;
            for (j = 0;j < 19;j++) {
                if (j % 2 == 0) {
                    printf("| ");
                }
                else {
                    if (arr[a][b] != 0) printf("%d ", arr[a][b]);
                    else printf("· ");
                    b++;
                }
                if (j == 18) printf("\n");
            }
            a++;
        }
    }
}

//通过导入文件生成数独函数组
//得到每组不可能的取值
int getimpossiblevalue(int x, int len) {
    for (int i = 0;i < 9;i++) {
        impossiblevalue[i] = 0;//初始化 impossiblevalue 数组
    }
    int j = 0;
    for (int i = 1;i <= x - len;i++) {
        impossiblevalue[j] = i;
        j++;
    }
    for (int i = x + len;i <= 9;i++) {
        impossiblevalue[j] = i;
        j++;
    }
```

```
    impossiblevalue[j] = x;
    j++;//返回的 j 为无法取到的数字的个数
    return j;
}
//将数独转化为 cnf 文件
string ToCnf(int a[][COL], int holes) {
    ofstream in("E:\\检查文件\\sudoku.cnf");//定义输入文件
    if (!in.is_open())
        cout << "文件无法打开!\n";

    //尝试计算有多少条 Clause
    int Clausenum = 0;
    for (int x = 0; x < ROW; ++x) {
        for (int y = 0; y < cols[x]; ++y)
            if (a[x][y] != 0)
                Clausenum++;
    }
    for (int x = 1; x <= 9; ++x) {
        for (int y = 1; y <= cols[x - 1]; ++y) {
            for (int z = 1; z <= 9; ++z)
                Clausenum++;
        }
    }
    for (int i = 1;i <= 9;i++) {
        for (int j = 1;j <= cols[i - 1];j++) {
            for (int x = 1;x <= 8;x++) {
                for (int y = x + 1;y <= 9;y++) {
                    Clausenum++;
                }
            }
        }
    }
    for (int l = 0;l < 27;l++) {
```

```
    int len = getarraynumlength(groupx[l]);
    for (int temp1 = 0;temp1 < len;temp1++) {
        for (int temp2 = 0;temp2 < len;temp2++) {
            if (temp1 != temp2) {
                for (int x = 1;x <= 9;x++) {
                    int num = getimpossiblevalue(x, len);
                    for (int y = 0;y < num;y++) {
                        Clausenum++;
                    }
                }
            }
        }
    }
}


in << "p" << " " << "cnf" << " " << 549 << " " << Clausenum << " " << endl;
//一个映射将不同的坐标映射到 61 个空(111~959->1~549)
//single clause,即初盘已经存在的数 61-holes
for (int x = 0; x < ROW; ++x) {
    for (int y = 0; y < cols[x]; ++y)
        if (a[x][y] != 0)
            in << (x + 1) * 100 + (y + 1) * 10 + a[x][y] << " " << 0 << endl;
}
//entry,每个格子必须要从 1~9 中填一个数 61
for (int x = 1; x <= 9; ++x) {
    for (int y = 1; y <= cols[x-1]; ++y) {
        for (int z = 1; z <= 9; ++z)
            in << x * 100 + y * 10 + z << " ";
        in << 0;
        in << endl;
    }
}
```

```
    //每个格子只能从 1-9 中填一个数 61*(8+7+6+5+...+1)=61*36
    for (int i = 1;i <= 9;i++) {
        for (int j = 1;j <= cols[i - 1];j++) {
            for (int x = 1;x <= 8;x++) {
                for (int y = x + 1;y <= 9;y++) {
                    in << 0 - (i * 100 + j * 10 + x) << " "
                    << 0 - (i * 100 + j * 10 + y) << " " << 0 << endl;
                }
            }
        }
    }
    //行,主副对角线的元素不能重复且同一组中的数需要连续
    for (int l = 0;l < 27;l++) {
        int len = getarraynumlength(groupx[l]);
        for (int temp1 = 0;temp1 < len;temp1++) {
            for (int temp2 = 0;temp2 < len;temp2++) {
                if (temp1 != temp2) {
                    for (int x = 1;x <= 9;x++) {
                        int num = getimpossiblevalue(x, len);
                        for (int y = 0;y < num;y++) {
                            in << 0 - (groupx[l][temp1] * 100 + groupy[l][temp1] * 10 + x) << "
"
                                << 0 - (groupx[l][temp2] * 100 + groupy[l][temp2] * 10 +
impossiblevalue[y]) << " " << 0 << endl;
                        }
                    }
                }
            }
        }
    }


    in.close();
```

```
    return string("sudoku.cnf");
}
//导入本地的数独文件
void SudokuParser(string filename, int a[][9]) {
    int i, j;
    filename = R"(E:\蜂窝数独游戏格局文件（设计要求的应用任务）
\easy_hanidoku.txt)";
    ifstream fis(filename);
    if (!fis) {
        cout << "文件无法打开.";
        exit(0);
    }
    int line;
    cout << "请输入想要求解的蜂窝数独所在的行数" << endl;
    cin >> line;
    char buf[100] = { '0' };
    for (i = 1;i <= line;i++) {
        fis.getline(buf, 100);
    }
    for (i = 5, j = 0 ; j < 5 ; i++, j++) {
        a[0][j] = buf[i] - 48;
        if (a[0][j] == 0) holes++;
    }
    for (i , j = 0; j < 6; i++, j++) {
        a[1][j] = buf[i] - 48;
        if (a[1][j] == 0) holes++;
    }
    for (i, j = 0; j < 7; i++, j++) {
        a[2][j] = buf[i] - 48;
        if (a[2][j] == 0) holes++;
    }
    for (i, j = 0; j < 8; i++, j++) {
        a[3][j] = buf[i] - 48;
```

```
        if (a[3][j] == 0) holes++;
    }
    for (i, j = 0; j < 9; i++, j++) {
        a[4][j] = buf[i] - 48;
        if (a[4][j] == 0) holes++;
    }
    for (i, j = 0; j < 8; i++, j++) {
        a[5][j] = buf[i] - 48;
        if (a[5][j] == 0) holes++;
    }
    for (i, j = 0; j < 7; i++, j++) {
        a[6][j] = buf[i] - 48;
        if (a[6][j] == 0) holes++;
    }
    for (i, j = 0; j < 6; i++, j++) {
        a[7][j] = buf[i] - 48;
        if (a[7][j] == 0) holes++;
    }
    for (i, j = 0; j < 5; i++, j++) {
        a[8][j] = buf[i] - 48;
        if (a[8][j] == 0) holes++;
    }

}


void interact(int sudoku[][9]) {
    int choice = 0;
    int x, y, z;
    printf("\n            请开始你的数独游戏！      \n");
    print(sudoku);
    printf("\n    1:填写数字      2：删除数字     3.查看答案  \n");
    cin >> choice;
```

```
    while (choice) {
        if (choice == 1) {
            printf("请输入在第几行 第几列填写何数字(中间用空格分隔)\n");
            cin >> x >> y >> z;
            if (SuDoKu_Judge(sudoku, x - 1, y - 1, z) == FALSE) {
                sudoku[x - 1][y - 1] = 0;
                printf("填错了，再想想\n");
                goto end1;
            }
            else {
                sudoku[x - 1][y - 1] = z;
            }
        }
        if (choice == 2) {
            printf("请输入在第几行 第几列删除(中间用空格分隔)\n");
            cin >> x >> y;
            sudoku[x - 1][y - 1] = 0;
        }
        if (choice == 3) {
            return;
        }
    end1:
        getchar(); getchar();
        system("cls");
        printf("\n            请开始你的数独游戏！      \n");
        print(sudoku);
        printf("\n    1:填写数字      2：删除数字     3.查看答案  \n");
        cin >> choice;
    }
}

//创建求解数独的文件
string createSudokuToFile() {
```

```
    int sudoku[9][9] = { 0 };
    int copy[9][9] = { 0 };

    if (flag == 1) {
        string filename;
        cout << "请输入想要求解的文件名" << endl;
        cin >> filename;
        SudokuParser(filename, sudoku);
    }

    if (flag == 2) {

        cout << "请输入想要挖洞的个数:" << endl;
        cin >> holes;

        //holes= 5;//挖洞个数
        createSudoku(sudoku);//生成数独终盘
        //print(sudoku);
        generateSudokuPuzzle(sudoku, holes);

    }
    print(sudoku);//输出初盘
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            copy[i][j] = sudoku[i][j];
        }
    }
    interact(copy);
    //转化为 cnf 文件
    string filename = ToCnf(sudoku, holes);
    return filename;
}
```

```
//DPLL 求解数独
status SudoDPLL(HeadNode* LIST, conse* result, int VARNUM) {
   //单子句规则
   HeadNode* Pfind = LIST;
   HeadNode* SingleClause = IsSingleClause(Pfind);
   while (SingleClause != nullptr) {
      result[T].num = SingleClause->right->data;
      SingleClause->right->data > 0 ? result[T++].value = TRUE : result[T++].value =
FALSE;
      int temp = SingleClause->right->data;
      DeleteHeadNode(SingleClause, LIST);//删除单子句这一行
      DeleteDataNode(temp, LIST);//删除相等或相反数的节点
      if (!LIST) return TRUE;
      else if (IsEmptyClause(LIST)) return FALSE;
      Pfind = LIST;
      SingleClause = IsSingleClause(Pfind);//回到头节点继续进行检测是否有单子
句
   }
   //分裂策略
   int Var = chooseDLISVariable(LIST);
   //int Var = LIST->right->data;//选取变元
   HeadNode* replica = Duplication(LIST);//存放 LIST 的副本 replica
   HeadNode* temp1 = ADDSingleClause(LIST, Var);//装载变元成为单子句
   if (SudoDPLL(temp1, result, VARNUM)) return TRUE;
   else {
      HeadNode* temp2 = ADDSingleClause(replica, -Var);
      return SudoDPLL(temp2, result, VARNUM);
   }
}
//输出求解完成的数独棋盘
void SudokuShow(conse* result, int VARNUM) {
   int res[9][9] = { 0 };
```

```
    for (int i = 0; result[i].value!=-1; ++i) {
        if (result[i].value == TRUE) {
            int x = (int)(abs(result[i].num) / 100) - 1;
            int y = (int)((abs(result[i].num) - (x + 1) * 100) / 10) - 1;
            res[x][y] = abs(result[i].num) - (x + 1) * 100 - (y + 1) * 10;
        }
    }
    //输出 result 数组
    cout << "求解后的数独如下:" << endl;
    print(res);
}
```