

**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：**

**学 号：**

**姓 名：**

**指导教师： 许贵平**

**报告日期： 2023/10/6**

**计算机科学与技术学**

**任务书**

* **设计内容**

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

* **设计要求**

要求具有如下功能：

**（1）输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)

**（2）公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)

**（4）时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

**（5）程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)

**（6）SAT应用：**将数双独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

* **参考文献**

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

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

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

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

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

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

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

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

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

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

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

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

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

**目录**

**任务书** I

**参考文献** II

**1引言** 4

**2系统需求与分析设计** 6

**3系统详细设计** 7

**4** **系统实现与测试** 10

**5** **总结与感悟** 18

**附录□** 19

**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。**

**对于任一布尔变元x，x与其非“¬x”称为文字(literal)。对于多个布尔变元，若干个文字的或运算l1∨l2∨…∨lk称为子句(clause)。只含一个文字的子句称为单子句。不含任何文字的子句称为空子句，常用符号□表示。子句所含文字越多，越易满足，空子句不可满足。**

**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增大。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问题解决器，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问题解决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的蜂窝数独游戏求解程序”作为实验课题，实现SAT求解器和蜂窝数独游戏两个功能。**

**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。**

**2系统需求分析与总体设计**

**2.1系统需求分析**

**系统能对规模中等的CNF文件进行快速求解，以及能够根据已有文件生成蜂窝数独游戏初始格局，并予以求解，展示交互界面。**

**2.2系统总体设计**

**系统设计分为三部分，数据结构库，应用层调用，用户界面。数据结构库中实现了一个十字链表及其基本操作，并通过调用十字链表存储CNF文件的子句。应用层通过调用十字链表的接口实现SAT文件的求解及数独游戏格局的生成、求解等等操作。 用户界面仿照windows系统的shell编写，通过执行各种相应的指令，来实现应用层的各种功能操作。**

**下图为系统提供了一个程序处理流程图，红色部分为基于DPLL的SAT求解相关功能模块，蓝色部分是游戏生成、转化、求解等处理模块。**



--------章与章之间插入分页符----------

**3系统详细设计**

**3.1有关数据结构的定义**（

**十字链表结构体:**

**typedef struct SATNode {**

**int data; // 数据域**

**SATNode\* next;**

**} SATNode;**

**typedef struct SATList {**

**SATNode\* head; // 表头**

**SATList\* next;**

**} SATList;**

**说明：**

**该十字链表将子句表示为由文字构成的链表；整个公式则是由子句构成的链表，如下图所示：**



**3.2 主要算法设计**

**3.2.1 DPLL算法介绍**

**本实验主要采用DPLL算法，它是一种基于树/二叉树的回溯搜索算法，主要使用两种基本处理策略：**

**单子句规则。如果子句集S中有一个单子句L,那么L一定取真值，于是可以从S中删除所有包含L的子句（包括单子句本身），得到子句集S1，如果它是空集，则S可满足。否则对S1中的每个子句，如果它包含文字¬L,则从该子句中去掉这个文字，这样可得到子句集合S2。S可满足当且仅当S2可满足。单子句传播策略就是反复利用单子句规则化简S的过程。**

**分裂策略。按某种策略选取一个文字L.如果L取真值，则根据单子句传播策略，可将S化成S2；若L取假值（即¬L成立）时，S可化成S1。**

**交错使用上述两种策略可不断地对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树,如下图所示：**



**3.2.2 DPLL算法改进思路**

**我们首先假定，在初始状态下，不存在已经被赋值的变量，我们称未被赋值的变量为自由变量。主循环通过给一个自由变量赋值，从而进入一个分支，这种对变量进行赋值的操作被称为判定，这个被赋值的变量也就有了一个与之对应的判定级。判定级从1开始，随之后的分支变量出现而增长。**

**进入分支后，问题可以简化为由这一判定及其推论决定。完成一些必要的推理后，可以得出在当前的判定级下，为了使被判定实例满足而需要进行的变量赋值。在分支后由该分支判定的推论得到的被赋值变量，被定义为与该分支变量处于同一判定级。**

**推理结束后，如果所有变量已经被赋值且所有子句都被满足，则返回1；如果在推理后的赋值状态下，存在矛盾子句，则当前判定分支及其子分支不可能是能够使实例被满足的变量赋值，这时求解器会进行回退操作。回退到0判定级则表示即使未进入任何一个判定分支，该输入实例仍然是不可满足的，在这种情况下，求解器将给出该被求解实例不可满足的结论。其中回溯操作回退到上一判定级，如果选定的赋值变量的两个文字均已经推理完毕，则再次回退到上一判定级。**

**在实际应用中，这种启发式算法，即通过计算同一变量文字的不同逻辑值的动态最大组合总和来选定分支变量的启发式算法，在标准测试数据上可以得到很好的结果。**

**4系统实现与测试**

**4.1系统实现**

**4.1.1 实验环境**

**实验环境为 Windows 11，代码采用编辑器 Visual Studio code 编写。**

**文件说明：**

**Sudoku.cpp：整个系统构建和用户操作实现**。

**4.1.1 SAT问题求解的主要函数及其功能与时空效率分析**

**本部分说明算法时直接用函数原型代替。**

1. **int ReadFile(SATList\*& cnf);**

**功能：用文件指针fp打开用户指定的文件，并读取文件内容保存到给定参数中，读取成功返回1，失败返回0。**

**算法实现：依次读取文件内容，读取到cnf时，记录后面的变元数和句子数。再依次开辟十字链表空间，记录每一条句子和文字。**

**时空效率分析：依次遍历每一条句子的每个文字，时间复杂度为O(n\*m)，需给十字链表开辟空间，空间复杂度为O(n\*m)，n和m分别为句子数量和每个句子所含文字数量。**

1. **void destroyClause(SATList\*& cnf);**

**功能：销毁链表。**

**算法实现：设置记录变量，遍历销毁十字链表的每一句子和每一个文字。**

**时空效率分析：需遍历每一个句子和每一个文字，时间复杂度为O(n\*m)，空间复杂度为O(1)，n和m分别为句子数量和每个句子所含文字数量。**

1. **int isUnitClause(SATNode\* cnf);**

**功能：判断是否为单子句，是返回1，不是返回0。**

**算法实现：判断传入参数句子是否符合“头指针不为空，下一指针为空”的条件，符合即为单子句，否则不是单子句。**

**时空效率分析：时间复杂度和空间复杂度均为O(1)。**

1. **int evaluateClause(SATNode\* cnf, int v[ ]);**

**功能：评估子句的真假状态，真返回1，假返回0。**

**算法实现：根据结果记录数组v里面的值，如果参数句子cnf里的某一个文字真值与v中的对应，则该句子为真，否则为假。**

**时空效率分析：需遍历该句子的所有文字，时间复杂度为O(m)，空间复杂度为O(1)，m是句子中的文字数量。**

1. **int removeClause(SATList\*& cnf, SATList\*& root);**

**功能：在已有的十字链表中删除指定的子句，删除成功返回1，失败返回0。**

**算法实现：若删除为头，则使头指针指向下一个结点，然后释放内存；若删除为其他位置，找到该结点的直接前驱，使其next指针指向要删除结点的下一结点，然后释放删除结点内存。**

**时空效率分析：需遍历查找指定句子，时间复杂度为O(n)，空间复杂度为O(1)，n是链表中句子的数量。**

1. **int removeNode(SATNode\*& cnf, SATNode\*& head);**

**功能：在指定的子句中删除指定的文字，删除成功返回1，失败返回0。**

**算法实现：若删除为头，则使头指针指向下一个结点，然后释放内存；若删除为其他位置，找到该结点的直接前驱，使其next指针指向要删除结点的下一结点，然后释放删除结点内存。**

**时空效率分析：需遍历查找指定文字，时间复杂度为O(m)，空间复杂度为O(1)，m是句子中文字的数量。**

1. **int addClause(SATList\* cnf, SATList\*& root);**

**功能：在已有的十字链表中添加指定的子句，添加成功返回1，失败返回0。**

**算法实现：直接添加在已有链表头部，使该结点next指针指向原链表头，然后原链表头赋值为该节点。**

**时空效率分析：时间复杂度和空间复杂度均为O(1)。**

1. **int emptyClause(SATList\* cnf);**

**功能：判断是否含有空子句，是返回1，不是返回0。**

**算法实现：遍历所有句子，若出现某一句子不含任何文字即head为空，则含有空子句，否则不含。**

**时空效率分析：需遍历链表中所有句子，时间复杂度为O(n)，空间复杂度为O(1)，n是子句的数量。**

1. **int DPLL(SATList\*& cnf, int value[ ]);**

**功能：求解SAT问题，给出满足条件时的一个式子,若有解则返回1，无解返回0。**

**算法实现：**

**1）查找链表中的单子句tp，使结果记录数组v中对应记录为真，即v[tp->data]=1或v[-tp->data]=0。**

**2）删除所有含文字tp->data的子句，以及删除所有负文字-tp->data。**

**3）删除完后，判断链表是否为空，若为空，则说明问题已解决，返回1；判断链表中是否含有空子句，若有，则说明问题无解，返回0；**

**4）重复1）、2）、3)直到链表中不出现单子句。**

**5）选取变元a策略：记录每一个变元出现的次数（正文字和负文字区别开来），若链表中有正文字存在，则在正文字中找到出现次数最多的变元，该变元为所选变元。若链表中只有负文字，则在负文字中找到出现次数最多的变元，该变元为所选变元。**

**6）在原有链表中添加只含文字a的单子句，递归DPLL函数，从步骤1）开始继续运算，若结果为1，则问题求解成功，返回1；若结果为0，则在原有链表中添加只含文字-a的单子句，再次递归DPLL函数，并直接返回求解结果。**

**时空效率分析：需遍历每个句子查找单子句，且要遍历每个文字找到出现次数最多的变元，即每一步的时间复杂度为O(n\*m)，最坏情况下，总共需走2^b步，即总时间复杂度为O(n\*m\*2^b)，每一步要记录先前的链表，空间复杂度为O(b)，n是链表子句数量，m是句子中文字的数量，b是总变元的数量。**

1. **void CopyClause(SATList\*& a, SATList\* b);**

**功能：将链表b的值复制到链表a中。**

**算法实现：对链表a依次开辟空间，并将b中每个句子和文字赋值给a中对应的结点。**

**时空效率分析：需遍历b中每个句子和文字，时间复杂度为O(n\*m)，需对a依次开辟空间，空间复杂度为O(n\*m)，n和m分别为句子数量和句子中文字的数量**

1. **int WriteFile(int result, double time, int value[]);**

**功能：将运行结果保存至同名文件，文件拓展名为.res,保存成功返回1，失败返回0。**

**算法实现：修改fileName的后三个字母，使其为res。然后依次写入运行结果result、求解记录value和运行时间Time。**

**时空效率分析：需遍历写入求解记录，时间复杂度为O(b)，空间复杂度为O(1)，b是变元数量。**

**4.1.2 蜂窝数独问题求解的主要函数及其功能**

**1.void CreateBinary( );**

**功能：读取初始化格局cnf文件，并将结果保存在res数组中，然后在交互界面展示初始化格局。**

**算法实现：开辟一个可容纳62个元素的数组res，然后读入初始化格局，最后打印初始化格局。**

**2.void SolvePuzzle( );**

**功能：求解输入棋盘，将结果保存在res数组中。**

**算法实现：根据棋盘已有数字和蜂窝数独满足的约束，建立CNF式子，再根据DPLL函数求解。**

**3. void drawHoneycomb();**

**功能：构造一个蜂巢，将res数组结果在蜂巢打印输出。**

**算法实现：通过循环结构构造蜂巢，再遍历res数组即可。**

**4.2系统测试**

**4.2.1 系统演示**

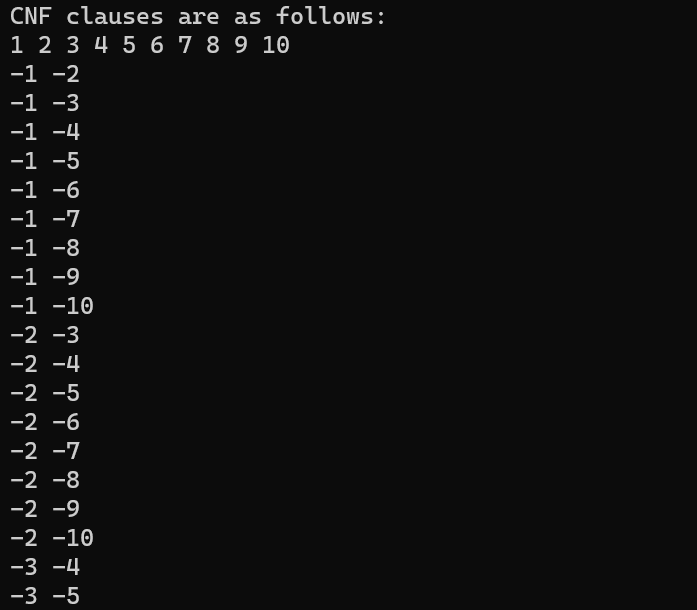
**系统交互界面如下图所示：**

**文本

描述已自动生成**

**图 4.2-1**

**cnf文件读取（读取文件：ais10.cnf）结果如下面各图所示：**

** 文本

描述已自动生成**

**图4.2-2 读取文件部分内容对比**

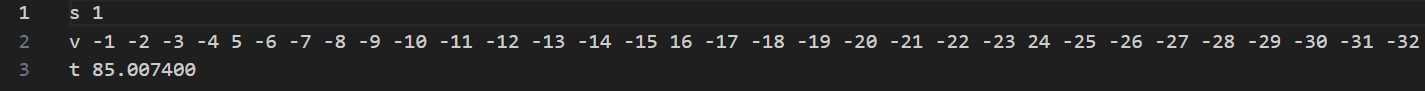
**DPLL求解算例（读取文件：ais10.cnf）如图所示：**

**文本

描述已自动生成**

**图4.2-3 求解算例成功**

**屏幕上有字

描述已自动生成**

**图 4.2-4 DPLL优化前后性能对比**

|  |  |  |
| --- | --- | --- |
| 求解文件 | 求解结果 | 求解时间 |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf  （可满足算例S级） | 1 | 49ms |
| problem1-20.cnf  （可满足算例S级） | 1 | 2ms |
| problem2-50.cnf  （可满足算例S级） | 1 | 12ms |
| problem3-100.cnf  （可满足算例S级） | 1 | 2248ms |
| problem6-50.cnf  （可满足算例S级） | 1 | 231ms |
| sud00012.cnf  （可满足算例M级） | 1 | 213ms |
| sud00021.cnf  （可满足算例M级） | 1 | 1062ms |
| sud00079.cnf  （可满足算例M级） | 1 | 101ms |
| sud00082.cnf  （可满足算例M级） | 1 | 194ms |
| sud00861.cnf  （可满足算例M级） | 1 | 46ms |
| bart17.shuffled-231.cnf  （可满足算例M级） | 1 | 50ms |
| sud00009.cnf  （可满足算例M级） | 1 | 207ms |
| eh-dp04s04.shuffled-1075.cnf  （可满足算例L级） | 1 | 7184ms |
| u-problem7-50.cnf  （不满足算例） | 0 | 785ms |
| tst\_v10\_c100.cnf  （不满足算例） | 0 | 0.0000ms |
| php-010-008.shuffled-as.sat05-1171.cnf  （不满足算例） | 0 | 13266ms |
| u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf  （不满足算例） | 0 | 295ms |
| ais10.cnf | 1 | 85ms |

**蜂窝数独游戏运行结果如图所示：**

**本次测试展示三个初始格局及最终求取结果。**

**图片包含 游戏机, 女人

描述已自动生成 图片包含 游戏机, 蜂窝

描述已自动生成**

**图 4.2-5 格局1展示**

**图片包含 形状

描述已自动生成 图片包含 游戏机, 蜂窝

描述已自动生成**

**图 4.2-6 格局2展示**

**图片包含 游戏机

描述已自动生成 形状

描述已自动生成**

**图 4.2-7 格局3展示**

**5 感悟与总结**

**课程设计重难点在于DPLL算法的实现，第一版程序能够算出规模较小的cnf文件，但遇到大文件容易出现内存爆满的情况，后来检查时发现是在DPLL函数递归调用过程中许多动态申请的空间没有释放掉，导致占用内存十分庞大。后来经过改进，虽然还是会出现内存崩掉的情况，但大部分文件还是可以处理。变元选取策略我最开始选择的是直接选择出现次数最多的文字，后来根据一些资料查阅，改进为启发式策略，这种策略下大部分文件都能快速得到解，但对一些特殊的文件和较大的文件，解起来十分慢也基本上解不出来，因为会出现内存不够的情况。在蜂窝数独游戏中，采用的是最简单最直接的根据已有cnf文件生成初始化格局，但是没有实现玩家自由填入数字操作，而是直接生成一个棋局和最后答案，让玩家先自己在纸上写好答案再和结果对比，在游戏交互性上显得不足。**

**若问收获，我想我最大的收获就是意识到自己代码能力薄弱，需要锻炼吧。孔子说学而不思则罔，思而不学则殆。过去的学习中，我总是拘泥于书本，并没有过多重视实践，这次课设使我变得清醒，毕竟以后的工作也要和代码打交道，代码能力不能是短板。这次课设也使我意识到，学习的过程中需要同伴，互相监督学习，互相鼓励支持，一个人是很难克制自己的，希望以后寻找一些学习上的同伴，共同进步。**

**最后，希望自己谨记经验教训，坚持下来。**

**附录**

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <ctime>

#include <chrono>

#include <cstring>

using namespace std;

using namespace std::chrono;

int value[100000000]; // 保存结果

int res[62] = {0}; // 打印蜂巢

int boolCount; // 布尔变元数量

int clauseCount; // 子句数量

char fileName[100]; // 文件名

double Time;

// 十字链表结构体

typedef struct SATNode {

int data; // 数据域

SATNode\* next;

} SATNode;

typedef struct SATList {

SATNode\* head; // 表头

SATList\* next;

} SATList;

int chess[10][10];

// 函数声明

int ReadFile(const char\* fileName, SATList\*& cnf);

void destroyClause(SATList\*& cnf);

int isUnitClause(SATNode\* cnf);

int evaluateClause(SATNode\* cnf, int v[]);

int removeClause(SATList\*& cnf, SATList\*& root);

int removeNode(SATNode\*& cnf, SATNode\*& head);

int addClause(SATList\* cnf, SATList\*& root);

int emptyClause(SATList\* cnf);

int DPLL(SATList\*& cnf, int value[], int Ans);

int better\_DPLL(SATList\*& cnf, int value[]);

void CopyClause(SATList\*& a, SATList\* b);

int WriteFile(int result, int value[]);

void CreateBinary();

void SolvePuzzle();

void drawHoneycomb();

int main(void) {

SATList\* CNFList = NULL, \* lp;

SATNode\* tp;

int op = 1, i, result;

while (op) {

system("cls");

cout << "\n\n";

cout << " Function Menu \n";

cout << "-------------------------------------------------\n";

cout << "1. Read CNF File 2. Traverse and Output Clauses\n";

cout << "3. Solve with DPLL and Save 4. Solve with improved DPLL and Save \n";

cout << "5. Binary Sudoku Game 0. Exit\n";

cout << "-------------------------------------------------\n";

cout << "Select your operation [0~5]: ";

cin >> op;

switch (op) {

case 1:

cout << "Enter the CNF file to read: ";

cin >> fileName;

ReadFile(fileName, CNFList);

getchar();

getchar();

break;

case 2:

if (CNFList == nullptr)

cout << "File not imported\n";

else {

cout << "CNF clauses are as follows:\n";

for (lp = CNFList; lp != nullptr; lp = lp->next) {

for (tp = lp->head; tp != nullptr; tp = tp->next) {

cout << tp->data << " ";

}

cout << endl;

}

}

getchar();

getchar();

break;

case 3:

if (CNFList == nullptr)

cout << "File not imported\n";

else {

for (int i = 1; i <= boolCount; i++)

value[i] = 1; // Initialize, all set to 1

result = DPLL(CNFList, value, boolCount);

cout << "Solution result: " << result << endl;

if (result == 1) {

if (WriteFile(result, value) == 1)

cout << "Results saved to a file with the same name (.res)" << endl;

else

cout << "Failed to save results" << endl;

}

}

getchar();

getchar();

break;

case 4:

if (CNFList == nullptr)

cout << "File not imported\n";

else {

for (int i = 1; i <= boolCount; i++)

value[i] = 1; // Initialize, all set to 1

result = better\_DPLL(CNFList, value);

cout << "Solution result: " << result << endl;

if (result == 1) {

if (WriteFile(result, value) == 1)

cout << "Results saved to a file with the same name (.res)" << endl;

else

cout << "Failed to save results" << endl;

}

}

getchar();

getchar();

break;

case 5:

CreateBinary();

SolvePuzzle();

drawHoneycomb();

getchar();

getchar();

break;

case 0:

break;

}

}

system("pause");

return 0;

}

/\*

\* 函数名称: ReadFile

\* 接受参数: SATList\*&

\* 函数功能: 用文件指针fp打开用户指定的文件，并读取文件内容保存到给定参数中，读取成功返回1，失败返回0

\* 返回值: int

\*/

int ReadFile(const char\* fileName, SATList\*& cnf) {

FILE\* fp = fopen(fileName, "r");

if (!fp) {

printf("File open failed!\n");

return 0;

}

char ch;

int number, i;

SATList\* lp;

SATNode\* tp;

// 跳过以 'c' 开头的行

while ((ch = getc(fp)) == 'c') {

while ((ch = getc(fp)) != '\n')

continue; // 跳过整行

}

// 确保已经到达 'p cnf' 开头的行

if (ch!= 'p') {

printf("Invalid file format!\n");

fclose(fp);

return 0;

}

getc(fp); getc(fp); getc(fp); getc(fp);

// 读取布尔变元数量和子句数量

fscanf(fp, "%d %d", &boolCount, &clauseCount);

// 从这里开始读取 CNF 数据

// ... (后续的代码和逻辑)

cnf = new SATList;

cnf->next = nullptr;

cnf->head = new SATNode;

cnf->head->next = nullptr;

lp = cnf;

tp = cnf->head;

for (i = 0; i < clauseCount; ++i, lp = lp->next, tp = lp->head) {

fscanf(fp, "%d", &number);

while(number != 0) {

tp->data = number;

fscanf(fp, "%d", &number);

if (number == 0)

tp->next = nullptr;

else{

tp->next = new SATNode;

tp = tp->next;

}

}

lp->next = new SATList;

lp->next->head = new SATNode;

if (i == clauseCount - 1) {

lp->next = nullptr;

break;

}

}

fclose(fp);

return 1;

}

/\*

\* 函数名称: destroyClause

\* 接受参数: SATList\*&

\* 函数功能: 销毁链表

\* 返回值: int

\*/

void destroyClause(SATList\*& cnf)

{

SATList\* lp1, \* lp2;

SATNode\* tp1, \* tp2;

for (lp1 = cnf; lp1 != NULL; lp1 = lp2)

{

lp2 = lp1->next;

for (tp1 = lp1->head; tp1 != NULL; tp1 = tp2)

{

tp2 = tp1->next;

free(tp1);

}

free(lp1);

}

cnf = nullptr;

}

/\*

\* 函数名称: isUnitClause

\* 接受参数: SATNode\*

\* 函数功能: 判断是否为单子句，是返回1，不是返回0

\* 返回值: int

\*/

int isUnitClause(SATNode\* cnf)

{

if (cnf != NULL && cnf->next == NULL)

return 1;

else

return 0;

}

/\*

\* 函数名称: evaluateClause

\* 接受参数: SATList\*

\* 函数功能: 评估子句的真假状态，真返回1，假返回0

\* 返回值: int

\*/

int evaluateClause(SATNode\* cnf,int v[])

{

SATNode\* tp = cnf;

while (tp != NULL)

{

if (tp->data > 0 && v[tp->data] == 1 ||

tp->data < 0 && v[-tp->data] == 0)

return 1;

}

return 0;

}

/\*

\* 函数名称: removeClause

\* 接受参数: SATList\*,SATList\*

\* 函数功能: 在已有的十字链表中删除指定的子句，删除成功返回1，失败返回0

\* 返回值: int

\*/

int removeClause(SATList\*& cnf, SATList\*& root)

{

SATList\* lp = root;

if (lp == cnf) root = root->next; //删除为头

else

{

while (lp != nullptr && lp->next != cnf) lp = lp->next;

lp->next = lp->next->next;

}

free(cnf);

cnf = nullptr;

return 1;

}

/\*

\* 函数名称: removeNote

\* 接受参数: SATNode\*,SATNode\*

\* 函数功能: 在指定的子句中删除指定的文字，删除成功返回1，失败返回0

\* 返回值: int

\*/

int removeNode(SATNode\*& cnf, SATNode\*& head)

{

SATNode\* lp = head;

if (lp == cnf) head = head->next; //删除为头

else

{

while (lp != nullptr && lp->next != cnf) lp = lp->next;

lp->next = lp->next->next;

}

free(cnf);

cnf = NULL;

return 1;

}

/\*

\* 函数名称: addClause

\* 接受参数: SATList\*,SATList\*

\* 函数功能: 在已有的十字链表中添加指定的子句，添加成功返回1，失败返回0

\* 返回值: int

\*/

int addClause(SATList\* cnf, SATList\*& root)

{

//直接插入在表头

if (cnf != NULL)

{

cnf->next = root;

root = cnf;

return 1;

}

return 0;

}

/\*

\* 函数名称: emptyClause

\* 接受参数: SATList\*

\* 函数功能: 判断是否含有空子句，是返回1，不是返回0

\* 返回值: int

\*/

int emptyClause(SATList\* cnf)

{

SATList\* lp = cnf;

while (lp != nullptr)

{

if (lp->head == nullptr) return 1;

lp = lp->next;

}

return 0;

}

/\*

\* 函数名称: CopyClause

\* 接受参数: SATList\*,SATList\*

\* 函数功能: 将链表b的值复制到链表a中

\* 返回值: void

\*/

void CopyClause(SATList\*& a, SATList\* b)

{

SATList\* lpa,\*lpb;

SATNode\* tpa,\*tpb;

a = new SATList;

a->head = new SATNode;

a->next = nullptr;

a->head->next = NULL;

for (lpb = b, lpa = a; lpb != NULL; lpb = lpb->next, lpa = lpa->next)

{

for (tpb = lpb->head, tpa = lpa->head; tpb != NULL; tpb = tpb->next, tpa = tpa->next)

{

tpa->data = tpb->data;

tpa->next = new SATNode;

tpa->next->next = NULL;

if (tpb->next == NULL)

{

free(tpa->next);

tpa->next = NULL;

}

}

lpa->next = new SATList;

lpa->next->head = new SATNode;

lpa->next->next = NULL;

lpa->next->head->next = NULL;

if (lpb->next == NULL)

{

free(lpa->next->head);

free(lpa->next);

lpa->next = nullptr;

}

}

}

/\*

\* 函数名称: DPLL

\* 接受参数: SATList \*

\* 函数功能: 求解SAT问题，给出满足条件时的一个式子,若有解则返回1，无解返回0

\* 返回值: int

\*/

int better\_DPLL(SATList\*& cnf, int value[]) {

auto start = high\_resolution\_clock::now();

SATList\* tp = cnf, \* lp = cnf, \* sp;

SATNode\* dp;

int\* count = new int[boolCount \* 2 + 1]; // 记录每个文字出现次数的数组

int re, MaxWord, max;

// 查找并处理单子句

while (true) {

while (tp != nullptr && isUnitClause(tp->head) == 0) {

tp = tp->next;

}

if (tp == nullptr)

break;

else {

// 单子句规则简化

if (tp->head->data > 0) {

value[tp->head->data] = 1;

} else {

value[-tp->head->data] = 0;

}

re = tp->head->data;

for (lp = cnf; lp != nullptr; lp = sp) {

sp = lp->next;

// 查找含有核心文字的句子

for (dp = lp->head; dp != nullptr; dp = dp->next) {

if (dp->data == re) {

removeClause(lp, cnf); // 删除子句，简化式子

break;

}

if (dp->data == -re) {

removeNode(dp, lp->head); // 删除文字，简化子句

break;

}

}

}

tp = cnf; // 继续简化

}

// 极简化规则简化后

if (cnf == nullptr) {

delete[] count;

return 1; // 所有子句都满足

}

else if (emptyClause(cnf)) {

delete[] count;

destroyClause(cnf);

return 0; // 出现空子句，不满足

}

}

for (int i = 0; i <= boolCount \* 2; i++) {

count[i] = 0;

}

// 计算子句中各文字出现次数

for (lp = cnf; lp != nullptr; lp = lp->next) {

for (dp = lp->head; dp != nullptr; dp = dp->next) {

if (dp->data > 0) {

count[dp->data]++;

} else {

count[boolCount - dp->data]++;

}

}

}

max = 0;

// 找到出现次数最多的正文字

for (int i = 2; i <= boolCount; i++) {

if (max < count[i]) {

max = count[i];

MaxWord = i;

}

}

if (max == 0) {

// 若没有出现正文字，找到出现次数最多的负文字

for (int i = boolCount + 1; i <= boolCount \* 2; i++) {

if (max < count[i]) {

max = count[i];

MaxWord = boolCount-i;

}

}

}

delete[] count;

// 在一个分支中搜索

SATList\* lp1 = new SATList;

lp1->head = new SATNode;

lp1->head->data = MaxWord;

lp1->head->next = nullptr;

lp1->next = nullptr;

CopyClause(tp, cnf);

addClause(lp1, tp);

if (better\_DPLL(tp, value) == 1) {

return 1;

}

destroyClause(tp);

// 在另一个分支中搜索

SATList\* lp2 = new SATList;

lp2->head = new SATNode;

lp2->head->data = -MaxWord;

lp2->head->next = nullptr;

lp2->next = nullptr;

addClause(lp2, cnf);

re = better\_DPLL(cnf, value);

destroyClause(cnf);

auto end = high\_resolution\_clock::now();

duration<double> duration = end - start;

Time = duration.count();

return re;

}

/\*

\* 函数名称: 优化前的DPLL

\* 接受参数: SATList \*

\* 函数功能: 求解SAT问题，给出满足条件时的一个式子,若有解则返回1，无解返回0

\* 返回值: int

\*/

int DPLL(SATList\*& cnf, int value[], int Ans) {

auto start = high\_resolution\_clock::now();

SATList\* tp = cnf, \* lp = cnf, \* sp;

SATNode\* dp;

int\* count = new int[boolCount \* 2 + 1]; // 记录每个文字出现次数的数组

int re, MaxWord, max;

// 查找并处理单子句

while (true) {

while (tp != nullptr && isUnitClause(tp->head) == 0) {

tp = tp->next;

}

if (tp == nullptr)

break;

else {

// 单子句规则简化

if (tp->head->data > 0) {

value[tp->head->data] = 1;

} else {

value[-tp->head->data] = 0;

}

re = tp->head->data;

for (lp = cnf; lp != nullptr; lp = sp) {

sp = lp->next;

// 查找含有核心文字的句子

for (dp = lp->head; dp != nullptr; dp = dp->next) {

if (dp->data == re) {

removeClause(lp, cnf); // 删除子句，简化式子

break;

}

if (dp->data == -re) {

removeNode(dp, lp->head); // 删除文字，简化子句

break;

}

}

}

tp = cnf; // 继续简化

}

// 极简化规则简化后

if (cnf == nullptr) {

delete[] count;

return 1; // 所有子句都满足

}

else if (emptyClause(cnf)) {

delete[] count;

destroyClause(cnf);

return 0; // 出现空子句，不满足

}

}

for (int i = 0; i <= boolCount \* 2; i++) {

count[i] = 0;

}

if (Ans > 0)

Ans = Ans / 2;

// 在一个分支中搜索

SATList\* lp1 = new SATList;

lp1->head = new SATNode;

lp1->head->data = Ans;

lp1->head->next = nullptr;

lp1->next = nullptr;

CopyClause(tp, cnf);

addClause(lp1, tp);

if (better\_DPLL(tp, value) == 1) {

return 1;

}

destroyClause(tp);

// 在另一个分支中搜索

SATList\* lp2 = new SATList;

lp2->head = new SATNode;

lp2->head->data = -Ans;

lp2->head->next = nullptr;

lp2->next = nullptr;

addClause(lp2, cnf);

re = better\_DPLL(cnf, value);

destroyClause(cnf);

auto end = high\_resolution\_clock::now();

duration<double> duration = end - start;

Time = duration.count();

return re;

}

/\*

\* 函数名称: WriteFile

\* 接受参数: int,int,int[]

\* 函数功能: 将运行结果保存至同名文件，文件拓展名为.res,保存成功返回1，失败返回0

\* 返回值: int

\*/

int WriteFile(int result, int value[]) {

FILE\* fp;

int i;

for (i = 0; fileName[i] != '\0'; i++) {

// 修改扩展名为 ".res"

if (fileName[i] == '.' && fileName[i + 4] == '\0') {

fileName[i + 1] = 'r';

fileName[i + 2] = 'e';

fileName[i + 3] = 's';

break;

}

}

fp = fopen(fileName, "w");

if (!fp) {

printf("Failed to open file for writing!\n");

return 0;

}

fprintf(fp, "s %d\nv ", result); // 求解结果

if (result == 1) {

// 保存解值

for (i = 1; i <= boolCount; i++) {

if (value[i] == 1) {

fprintf(fp, "%d ", i);

} else {

fprintf(fp, "%d ", -i);

}

}

}

fprintf(fp, "\nt %lf", Time \* 1000); // 运行时间（毫秒）

fclose(fp);

return 1;

}

void CreateBinary()

{

int op , k;

FILE \* fp;

char ch;

char inputfile [100];

cout << "Please choose a file to initialize your puzzle : ";

cin >> inputfile;

strcpy (fileName ,inputfile);

fileName[2]='9';

fp = fopen (inputfile , "r");

k = 1;

while (fscanf (fp ,"%c", &ch) != EOF){

if (ch != '\n')

res [k++] = ch - '0';

}

fclose (fp);

cout << " Your Initial puzzle is : ";

drawHoneycomb();

}

void SolvePuzzle( )

{

const char \* cnf\_file = "limit.cnf";

char ch;

SATList\* CNFList = nullptr;

ReadFile (cnf\_file, CNFList);

FILE \* fp = fopen (fileName , "r");

for (int i = 1; i <= 9; i++)

for (int j = 1; j <= 9; j++){

fscanf (fp ,"%c", &ch );

chess[i][j] = ch - '0';

}

fclose (fp);

for (int i = 1; i <= 9; i++)

for (int j = 1; j <= 9; j++){

if (chess[i][j] > 0){

SATList\* cnf = new SATList;

cnf->head = new SATNode;

cnf->next = nullptr;

cnf->head->data = i \* 100 + j \* 10 + chess[i][j];

cnf->head->next = nullptr;

addClause (cnf,CNFList);

}

} // 添加单子句

boolCount = 1000;

if (better\_DPLL (CNFList,value) == 1){

cout << "Suceesful solve the puzzle! "<< endl << "The ultimate puzzle is :" << endl;

for (int i = 1; i <= 9; i++)

for (int j = 1; j <= 9; j++)

for (int l = 1; l <= 9; l++){

if (value [i \* 100 + j \* 10 + l] && l != 0)

chess[i][j] = l;

}

int count = 1;

for (int i = 1; i <= 9; i++)

for (int j = 1; j <= 9; j++){

if (chess[i][j])

res [count++] = chess[i][j];

}

}

else

cout << "fail to solve the puzzle!" << endl;

}

void drawHoneycomb() {

int rowcount = 19, colomncount\_1 = 5, spacecount = 9, colomncount\_2 = 6;

int i, j, count = 1;

cout << endl<<endl<<endl;

count = 1;

for (i = 1; i <= 10; i++, spacecount--) {

for (int k = 1; k <= spacecount; k++)

cout << " ";

if (i % 2) {

for (j = 1; j <= colomncount\_1; j++) {

if (j != colomncount\_1)

cout << "/" << " " << "\\" << " ";

else

cout << "/" << " " << "\\" << endl;

}

colomncount\_1++;

} else {

for (j = 1; j <= colomncount\_2; j++) {

if (j != colomncount\_2)

cout << "|" << " " << res [count ++] /\*在这个位置填入数字\*/<< " ";

else

cout << "|" << endl;

}

colomncount\_2++;

}

}

spacecount = 1;

colomncount\_1 = 9, colomncount\_2 = 9;

for (i = 11; i <= 19; i++, spacecount++) {

for (int k = 1; k <= spacecount; k++)

cout << " ";

if (i % 2) {

for (j = 1; j <= colomncount\_1; j++) {

if (j != colomncount\_1)

cout << "\\" << " " << "/" << " ";

else

cout << "\\" << " " << "/" << endl;

}

colomncount\_1--;

} else {

for (j = 1; j <= colomncount\_2; j++) {

if (j != colomncount\_2)

cout << "|" << " " << res [count ++] << " ";

else

cout << "|" << endl;

}

colomncount\_2--;

}

}

}