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：计卓 2201

学　　号：U202215322

姓　　名：濮澍

指导教师：许贵平

报告日期：2023.10.7

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%)

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

SAT（可满足性问题）是一个经典的计算机科学问题，具有广泛的应用和深远的理论意义：

1.1.1 理论计算复杂性：

SAT 问题是计算机科学中的一个重要问题，被证明是 NP 完全问题的代表。这意味着如果存在一个高效算法解决了 SAT 问题，那么同样可以高效地解决许多其他 NP 完全问题，如图着色、旅行商问题、子集和问题等。因此，SAT 问题的研究对于理解计算复杂性理论和算法复杂性分析至关重要。

1.1.2 应用广泛：

SAT 问题的应用涵盖了多个领域，包括计算机辅助设计（CAD）、硬件和软件验证、自动规划、自动化定理证明、人工智能、生物信息学等。在 CAD 领域，SAT 用于电路设计中的布线问题、时序分析以及模块化测试。在人工智能中，SAT 用于解决知识表示和推理问题。因此，提高 SAT 问题的求解效率对于这些应用领域的性能和可靠性至关重要。

1.1.3 算法研究：

SAT 问题是算法设计和分析的理想测试平台。许多先进的算法和数据结构，如分支定界、随机化算法、启发式算法、学习算法和并行计算等，都在 SAT 问题的求解中得到了广泛的应用。因此，SAT 问题的研究对于算法研究的进展也具有重要作用。

1.2 国内外研究现状

1.2.1 国际研究现状：

1. **CDCL 算法：** 目前，Conflict-Driven Clause Learning (CDCL) 算法是解决 SAT 问题的主流方法。译为冲突驱动的子句学习算法，通过迭代地修改和学习约束来寻找解。此算法相对于 DPLL 最大的特点是非时序性回溯。
2. **并行化与分布式求解：** 随着硬件技术的进步，研究者致力于将 SAT 问题并行化和分布化，以提高求解速度。并行 SAT 求解器如 Plingeling 和

MapleCOMSPS 等在国际竞赛中表现出色。

3. **深度学习：** 随着近些年神经网络的发展，愈来愈多的问题被提供给深度学习。而 SAT 问题由于神经网络的可满足性被认为是可解的。

1.2.2 国内研究现状：

SAT 问题在中国计算机科学领域得到广泛研究，许多国内高校和研究机构都有 SAT 问题的研究团队。 这些团队在算法设计、应用研究以及竞赛中都取得了一定的成就。

1. **竞赛表现：** 中国的研究者和团队积极参与国际 SAT 竞赛，并在多次比赛中获得了优异的成绩。这反映了中国在 SAT 问题研究领域的活跃程度和竞争力。
2. **应用研究：** 中国的研究者也在 SAT 问题的应用领域进行了有益的探索，如在 EDA（电子设计自动化）领域，以及在智能制造、自动化规划等方面。

总的来说，SAT 问题在国际和国内都受到了广泛的研究和关注。国际上的先进技术和方法为中国的研究者提供了学习和合作的机会，同时国内也有许多研究团队积极贡献于该领域的发展。这个课题具有深远的理论意义和实际应用价值，对计算机科学和工程领域都具有重要的影响力。

1.3 课程设计的主要研究工作

本次课程设计主要依据任务书的指导，要求学生理解 SAT 问题的基本样式和求解方法，深入理解 DPLL 算法并通过代码实现基于 DPLL 算法的 SAT 求解器。给出一个具体问题（蜂窝数独求解），尝试将其转换成 SAT 问题并应用求解器进行

求解。

在具体实现过程中，要研究选择适当的数据结构，实现任务书的目标。代码要进行分块处理，并对函数进行规范书写和调用。难点在于 dpll 算法的代码实现和优化，以及蜂窝数独问题转化成 SAT 问题的过程。最终要研究实现具有交互功能的演示程序。

本次设计的主要研究工作可以分为以下几个关键方面：

1. 物理存储结构设计：

- 设计有效的数据结构来表示 SAT 问题中的变元、文字、子句和公式。这需要仔细选择数据结构以提高求解效率。

2. 输入输出功能：

- 开发用户友好的输入界面，允许用户指定程序执行参数，如优化选项等。

- 实现 SAT 算例 CNF 文件的读取和解析功能。

- 设计输出功能，能够将求解结果输出，并将执行时间记录并输出。

3. 公式解析与验证：

- 基于文献[1-3]的参考，建立公式的内部表示。

- 实现对解析正确性的验证功能，确保解析的准确性，包括逐行输出和显示每个子句。

4. DPLL 过程实现：

- 基于 DPLL 算法框架，实现 SAT 算例的求解。这是整个设计的核心部分。

- 避免使用 C++现有的 vector 等类库，而是自行设计数据结构，以满足性能要求。

5. 时间性能的测量：

- 使用相应的时间处理函数（例如，参考 time.h 库）记录 DPLL 过程的执行时间，以毫秒为单位。

- 将执行时间作为输出信息的一部分，使用户能够了解求解器的性能。

6. 程序优化：

- 针对 DPLL 算法的实现进行优化，包括但不限于存储结构的改进和分支变元选取策略的优化。

- 记录优化前后的执行时间，计算性能优化率，以衡量优化效果。

7. SAT 应用：

- 将蜂窝数独游戏问题转化为 SAT 问题，以实现数独游戏的求解。

- 集成数独游戏求解器到主要的 SAT 求解器中，并提供一定的交互性，使用户可以与游戏互动。

总体而言，主要的研究工作涉及到设计和实现一个高效的 SAT 求解器，包括输入输出功能、公式解析与验证、DPLL 过程的实现、时间性能测量和程序优化。此外，还需要将 SAT 求解器扩展到数独游戏应用领域，使其具备一定的交互性和可玩性。这些工作将为 SAT 问题求解和应用领域的研究提供重要的基础和实用价值。

2 系统需求分析与总体设计

2.1 系统需求分析

2.1.1 系统需求分析

目标：本设计旨在开发一个高效的 SAT 求解器，能够针对给定的中小规模 SAT 问题进行求解。具体目标包括：

1. 求解能力：能够解决 CNF 格式的 SAT 问题，确定是否存在一种变量赋值方式使得表达式为真。

2. 性能优化：实施性能优化，包括改进 DPLL 算法的存储结构、分支变元选取策略等，以提高求解效率。

3. 用户友好：提供用户友好的界面，允许用户指定程序执行参数，并能够记录求解时间和输出求解结果。

4. 可扩展性：可以将 SAT 问题应用于蜂窝数独游戏的求解，并提供一定的交互性，使用户可以与游戏互动。

2.1.2 事务处理流程：

1. 输入参数和 SAT 算例读取：

用户通过程序输入所需的参数，包括 SAT 算例的 CNF 文件路径、优化选项等。程序读取并解析 CNF 文件，构建内部表示。

2.公式解析与验证：

基于 CNF 文件内容，建立内部表示，包括变元、文字、子句和公式。验证解析的正确性，通过逐行输出和显示每个子句，与输入算例进行比对。

3. DPLL 过程求解：

利用 DPLL 算法框架，对 SAT 问题进行求解。在求解过程中使用自定义的数据结构来存储 CNF 公式和求解状态。

4. 时间性能测量：

使用时间处理函数记录 DPLL 过程的执行时间，以毫秒为单位。将执行时间作为输出信息的一部分。

5. 程序优化：

针对 DPLL 算法的实现进行优化，包括改进存储结构和分支变元选取策略。记录优化前后的执行时间，计算性能优化率。

6. SAT 应用：

将蜂窝数独游戏问题归约为 SAT 问题。集成数独游戏求解器到主要的 SAT

求解器中，并提供交互性，使用户能够与游戏互动。

7. 输出结果和保存：

输出 SAT 问题的解或判断无解的结果。将求解结果保存到文件，以备后续参考。

2.2 系统总体设计

系统总体由五个模块组成，分别是对数据结构进行定义和函数声明的 header 模块，DPLL 求解 cnf 文件的 DPLL 函数，求解下游任务的蜂窝数独函数和将读取进的蜂窝数独数字化为文字的函数，以及实现与用户交互，调用各类函数实现，读取 cnf 最终展示的 main 函数模块。

演示程序具体操作流程如下：

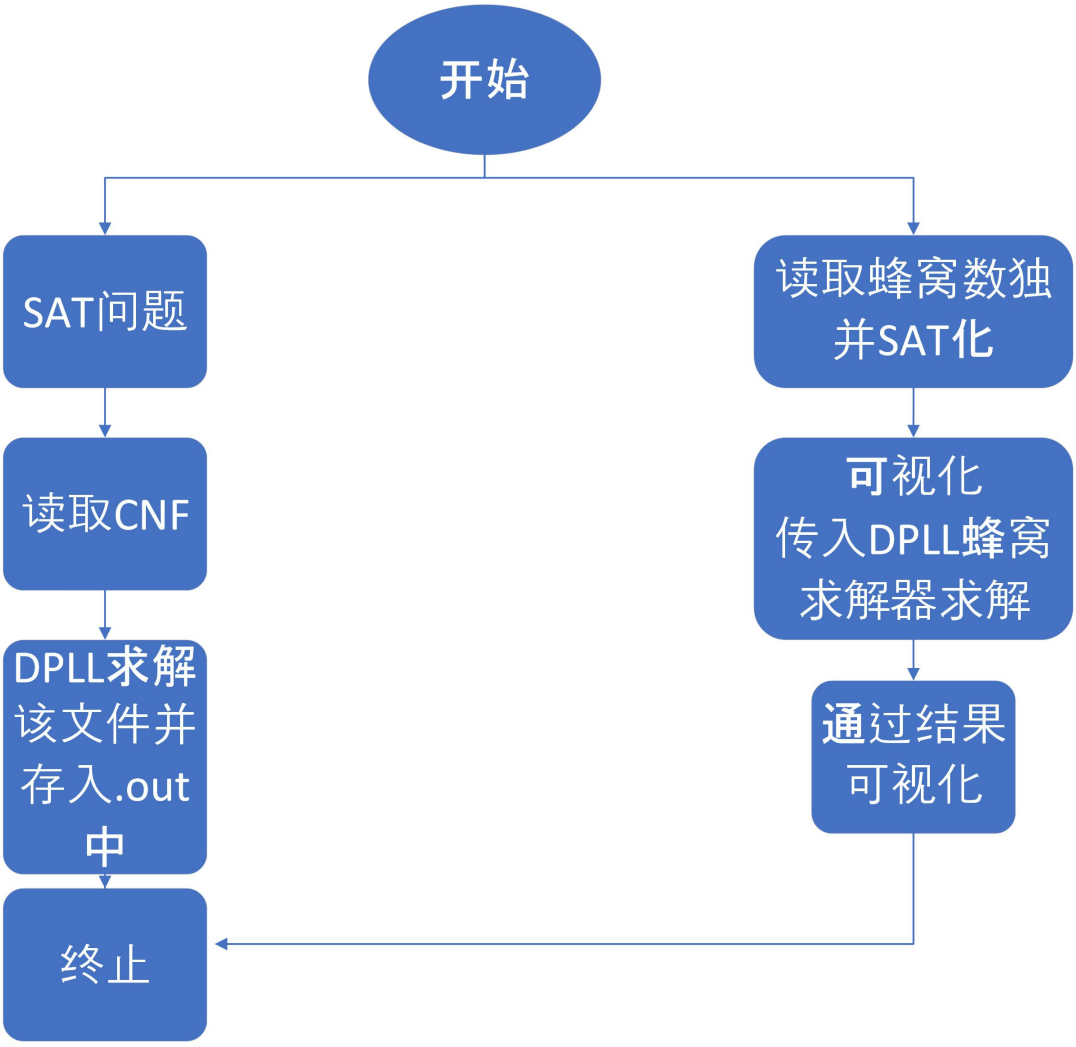

图 2-1 系统总体设计图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1数据结构定义：

优化前

Clauses结构体：

每一个都包含一个value并指向下一个Clauses。子句的首个Clauses中value表示子句中文字个数，0表示被删除。

ClauseSet结构体：

每一个都指向一个Clauses结构体表示一个子句，并有一个指针指向ClauseSet表示整句

优化后

3.1.2数据关联描述：

在程序中，文字通过Clauses表示，整句由ClauseSet串联

关系可以通过下图直观表示：

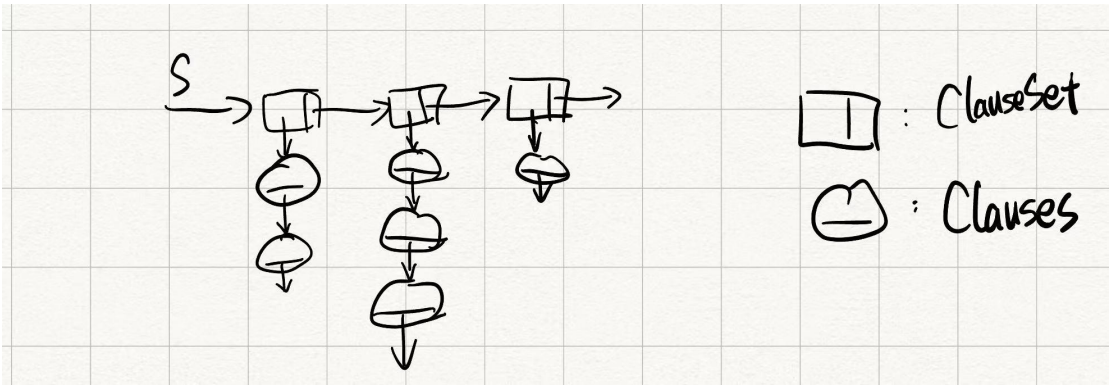

图 3-1 数据项之间的关联

3.2 主要算法设计

3.2.1 header模块：

提供必要的调用库和函数原型声明以及一些简单函数定义。

3.2.2 Dpll模块：

首先根据单子句规则进行删除，直至找不到单子句，调用分裂策略找到一个变元，先复制当前的数据，产生当前链表的拷贝，然后将新选择的变元进行赋值，并将其作为单子句插入拷贝中，对此时已插入新的单子句的链表拷贝递归调用下

一次 DPLL，返回值为 true 则求解完毕，返回至为 false 则在当前链表中加入相反的单子句进行 DPLL 过程。

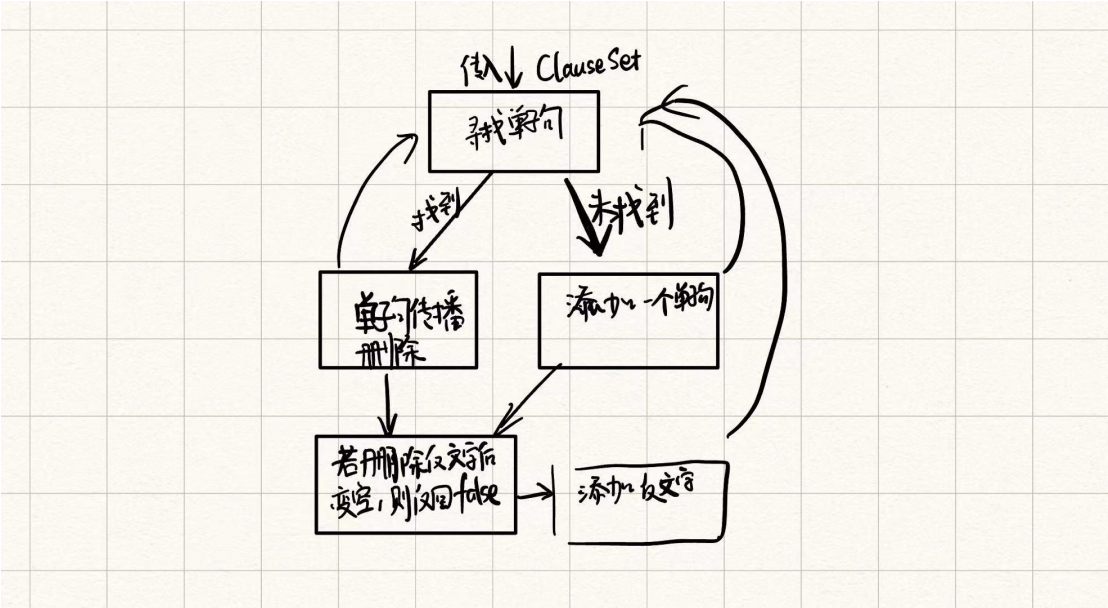

图 3-3 dpll 模块主要算法流程图

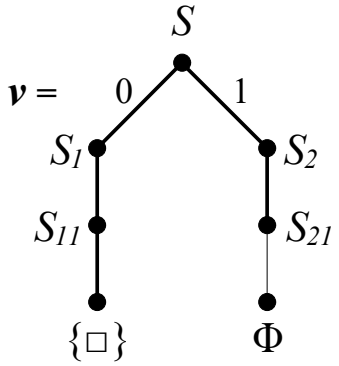

图 3-2 DPLL 算法搜索树

基于单子句传播与分裂策略的 DPLL 算法可以描述为一个如后所示的递归过程 DPLL(S)，DPLL 算法也可用非递归实现。

```
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 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
}
```

关于算法的优化，主要选择在单子句传播进行优化，通过线索将每个文字的下一个出现地点和反文字出现地点标记后，在传播时便不用再遍历整个句子，而是只删除有用信息。同时，在删除反文字后出现新的单子句后可以放入栈中，这样就不需再遍历寻找单子句。但因为需要改的结构，代码太多，最终只实现了一部分，bug 过多

3.2.3 SVS模块：

SVS 即 Single Value Spread 单子句传播算法，对标记文字整个子句进行删除（标记 0），对反文字单独删除，并在该子句头结点上标记 value--

3.2.4 DPLLhaniddoku模块：

通过 Encoding 函数将读入的 Haniddoku 编码，在 Encoding 中制作句子，先后加入格限制，行限制，两个对角线限制。再根据读入数据个性化添加已填数据限制

此时就已经将蜂窝数独问题转化为 SAT 求解问题。不过在编码时，个人直接将坐标文字化，因此需要在求解完成后再通过特定求解方法解码。

3.2.5 main模块：

实现与用户交互，打印出菜单界面，内部使用 switch 语句根据用户输入调用相应模块，同时给出提示并输出结果。

4 系统实现与测试

4.1 系统实现

系统环境：Ubuntu 20.04

开发环境：VSCode

硬件环境：Intel i7-12700H 32G 内存

数据类型定义如下：

```
typedef struct clauses{
    int value;
    struct clauses *next;
}Clauses;



typedef struct clauseSet{
    Clauses *head;
    struct clauseSet *next;
}ClauseSet;
```

图 4-1 数据结构定义

```
bool DPLL(ClauseSet *S); //DPLL整体框架算法
bool DPLLUpdation(ClauseSet_h *Sh,int n,int ist = 0,State *bak1 = NULL,int *bak2 = NULL);//优化后的DPLL算法（未实现）
bool DPLLHaniddoku(ClauseSet *S);//蜂窝数独DPLL算法
void Copy(ClauseSet *S, ClauseSet *TempS);//回溯用的复制整句函数
void Recover(ClauseSet_h *Sh, int ist,State *bak1,int *bak2);//优化用的回溯函数（未实现）
// void deletion(ClauseSet *S); //删除子句 被优化掉了
bool SVS(ClauseSet *S, int tag); //单子句传播
bool SVS(ClauseSet_h *Sh, int tag); //优化后的单子句传播（未实现）
```

图 4-2 dpll 函数列表

```
void PrintHaniddoku(int Hani[61]); //打印数独
ClauseSet *Encoding(int Hani[61]); //数独编码
void AddCellConstriant(ClauseSet *Han); //添加单元格约束
void AddRowConstriant(ClauseSet *Han); //添加行约束
void AddLDiagConstriant(ClauseSet *Han); //添加左下右上斜线约束
void AddRDiagConstriant(ClauseSet *Han); //添加右下左上斜线约束
ClauseSet * Add_must_fill(int r,int num); //重载行必填
ClauseSet * Add_must_fill(int r,int num,int flag); //重载对角线必填
```

图 4-3 dpllhaniddoku 函数列表

4.2 系统测试

4.2.1常用软件测试方法

分模块进行测试，先验证各模块功能是否正常，再对模块进行性能测试。

4.2.2各模块功能及设计目标

DPLL模块：通过DPLL算法设计SAT问题求解器，实现对中小规模的cnf算例的求解并记录求解所需时间，若满足则将解保存到相应的out文件中，同时设计一个优化算法，对已有算法进行优化，提升求解效率。

DPLLHaniddoku模块：依据难度在已有棋盘的基础上编码生成蜂窝数独棋盘，检查棋盘是否具有唯一解，通过输出提示与用户交互，跟据用户输入进行数独的填写，提示，输出答案等操作，能够及时更新数独棋盘。

4.2.3测试大纲

先选择功能测试算例，检查系统是否具有正确求解可满足和不可满足算例的功能，功能性测试算例通过之后再选择性能测试算例，根据变元与子句的多少综合判断算例规模之后，从易到难依次进行测试，观察算例的求解所需时间，判断DPLL算法的优化情况、试探程序能够较快求解的最大算例规模。

在进行完SAT求解器的测试后，进行蜂窝数独对应功能的测试，主要有生成数独棋盘，进行唯一解验证，与用户进行交互等。

4.2.4运行结果

先进行读取cnf文件以及保存结果的功能测试，运行结果如图4-4、4-5所示

图 4-4　读取 cnf 文件功能测试

```
└$ cat out/ais10.out
SATISFIED
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
-31 -32 -33 -34 -35 -36 -37 -38 39 -40
-41 -42 43 -44 -45 -46 -47 -48 -49 -50
-51 -52 -53 -54 -55 -56 -57 58 -59 -60
-61 -62 -63 64 -65 -66 -67 -68 -69 -70
-71 -72 -73 -74 -75 -76 77 -78 -79 -80
-81 -82 -83 -84 85 -86 -87 -88 -89 -90
-91 -92 -93 -94 -95 96 -97 -98 -99 -100
-101 -102 -103 -104 -105 -106 -107 -108 109 -110
-111 -112 -113 -114 -115 -116 117 -118 -119 -120
-121 -122 -123 -124 125 -126 -127 -128 -129 -130
-131 -132 133 -134 -135 -136 -137 -138 -139 -140
141 -142 -143 -144 -145 -146 -147 -148 149 -150
-151 -152 -153 -154 -155 -156 157 -158 -159 -160
-161 -162 -163 -164 165 -166 -167 -168 -169 -170
-171 -172 173 -174 -175 -176 -177 -178 -179 -180
-181 0.501413
```

图 4-5 保存结果的功能测试

再进行DPLL功能测试，分别选择满足算例（已在上方测试）和不满足算例进行测试，运行结果如图4-6所示。

```
                功能菜单
------------------------------------------------
1.DPLL判断cnf文件
2.求解蜂窝数独
3.DPLL优化后大改cnf文件
0.退出
------------------------------------------------
        请选择你的操作[0~2]:
1
Please input the file name: unsat-5cnf-30
Done
^C
(PuEnv) ┌─pu@Pu-Thinkbook14plus in ~/PuFiles/Experiment/DDPL Program
└$ cat ./out/unsat-5cnf-30.out
UNSATISFIED
0.052463
```

图 4-6 不可满足算例 DPLL 功能测试

接下来进行性能测试，选取多个数据集并给出优化前后时间变化。以五分钟

为时间最大限制。

表 3-1 其他数据集的测试结果

数据集名称	优化前用时/ms	优化后用时/ms	优化率%
sat-20.cnf	0.814	0.641	21
Problem1-20.cnf	0.741	0.678	8
Problem2-50.cnf	13.424	0.708	179
Problem3-100.cnf	108.264	9.252	110
Problem6-50.cnf	21.3	0.974	222
Sud00001.cnf	31642.2	14.869	226014
Sud00009.cnf	552.2	36.337	1433
Sud00012.cnf	508.623	15.454	3286
Sud00021.cnf	23395	166.413	143750
Sud00079.cnf	227.013	32.081	593
Unsat-5cnf-30.cnf	45.7	38.96	48
u-problem7-50.cnf	84.117	4.872	2000
u-homer14.shuffled-300.cnf	超时	超时	0
eh-dp04s04.shuffled-1075.cnf	超时	超时	0
e-par32-3.shuffled-3176.cnf	超时	超时	0
ec-vda_gr_rcs_w9.shuffled-6498.cnf	超时	超时	0
eh-vmpc_25.renamed-as.sat05-1913-625.cnf	超时	超时	0
ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf	超时	超时	0
平均优化率%			-

通过表格可以发现，在数据集不大的情况下优化取得了较好的结果，在数据集较大的情况下优化没有实现太多改进，在某些特定情况下，存在负优化的情况，但我们从整体上可以认为这是在特定数据集下的特定情况。可以认为优化基本成功。

接下来进行蜂窝数独功能的测试。首先展示根据难度生成棋盘功能。如图4-7

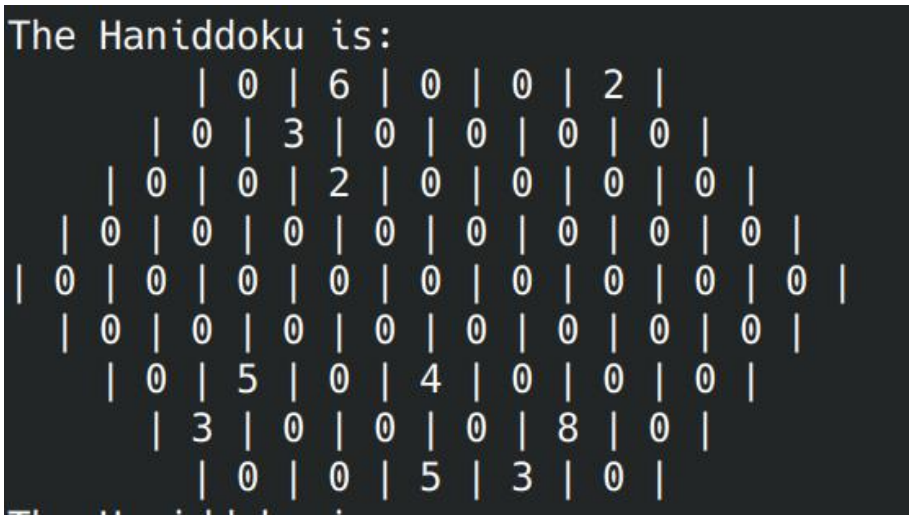

图 4-7 蜂窝数独棋盘生成

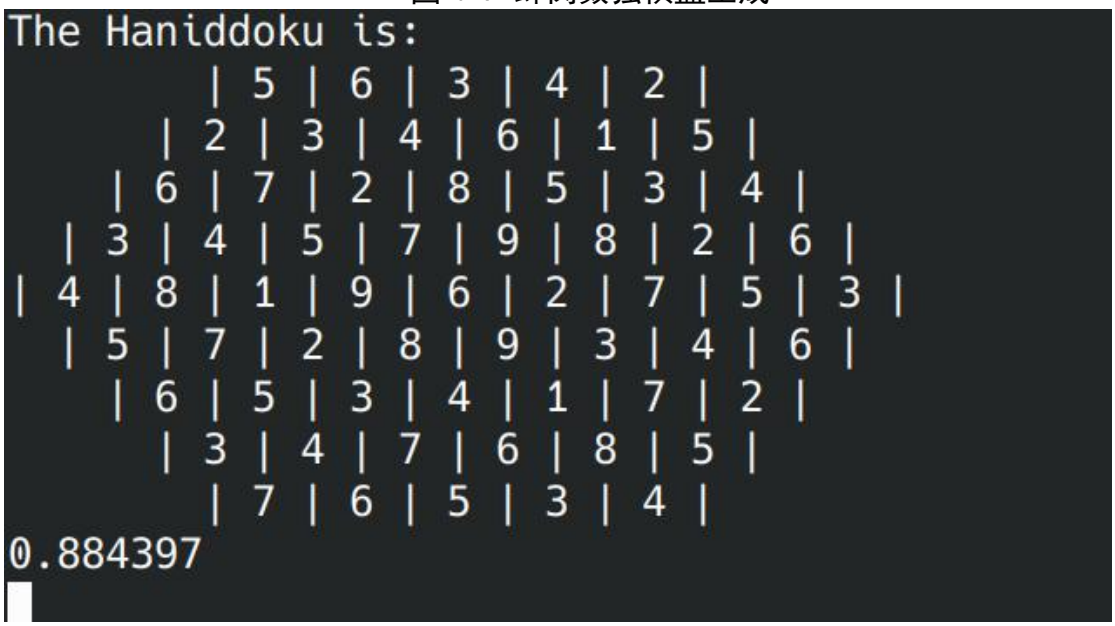

图 4-8 蜂窝数独查看答案功能

4.2.5运行结果分析

根据运行结果，该系统读入、打印cnf文件功能正常，能够对SAT问题进行求解且可以将结果保存在out文件中，通过进一步的性能测试，该系统能够在合适的时间内对小、中规模的算例验证其满足性并求解。在数独游戏中，根据运行结果，该系统能够自动读取或基于用户输入或者输出答案，棋盘可以相应更新。综上所述，该系统各功能正常完备，具有较好的性能，完成了相应的设计目标。

根据上述测试内容，可以认为程序已完成任务书规定的全部内容，实现基于DPLL算法的SAT求解和蜂窝数独转化成SAT问题并求解的程序设计。

5 总结与展望

5.1 全文总结

对自己的工作做个总结，主要工作如下：

（1）了解 SAT 问题，认识析取范式的概念，读取 cnf 文件，建立相应的数据结构进行存储；

（2）学习并优化 DPLL 算法，对相应的 SAT 问题进行求解，判断其满足性并记录求解所需时间；

（3）设计蜂窝数独游戏,将蜂窝数独游戏的规则转换为 SAT 问题进行求解。

成功完成了输入输出功能，公式解析与验证，DPLL 过程，时间性能的测量，程序优化，SAT 应用：蜂窝数独游戏的功能，完成课程设计的全部目标。

5.2 工作展望

在今后的研究中，将围绕着如下几个方面开展工作

（1）优化 DPLL 算法中的回溯策略，避免因为选取副本进行回溯占用太多内存，使之能够在合理的时间内化简大算例。

（2）优化变元选取策略，尝试多种变元选取的方式，实现对选取出现次数最多的变元的算法优化，提升求解器的适应性。使求解大规模，不同类型算例的能力更强。

（3）加强蜂窝数独棋盘的图形化，设计更好的交互界面，使之更容易被用户接受。

（4）提升蜂窝数独cnf文件的生成效率，同时尝试逐步给出用户正确答案的提示，提升程序可玩性。

6 体会

通过课程设计教学与实践环节，我进一步正确理解与应用了专业知识，增强和提高了分析问题与解决问题的综合能力。这个过程加深了我对于求解实际问题的基本科研步骤的体会与理解，也增强和提升了我的信息搜索和分析技能，培养了技术总结的基本技能，锻炼了课程设计报告的撰写能力。

这次的课程设计课程是我学习计算机以来做过的最大型、最完整的项目。为了理解SAT问题、DPLL算法和析取范式，我投入了大量时间查阅资料、阅读相关内容，以及学习对应的数学知识。经过认真细心的反复阅读相关资料和任务书，我对本次程序设计课程的主要内容有了大致了解和认识。

在学习的过程中，我首先独立思考，然后在遇到问题时积极地与同学们进行探讨。这种合作讨论的方式帮助我形成了基本的思路，也在遇到某些问题无法解决时给予我新的思路和启发。这对于当下或是未来的研究工作来说都是至关重要的。

在设计DPLL算法的优化过程中，我发现针对不同的算例，其最优变元选取策略也有所不同。最初的算法中，我的代码可以求解内存较大的算例，但却无法求解某些小规模的算例。这让我意识到分裂变元的选取可能在某些特殊的算例中存在偏向性，这就要求我们不断尝试，突破思维定式，寻找到合适的变元选取策略。

在蜂窝数独的设计与求解中，为了生成空白棋盘的cnf约束，我以近乎手动的方式编写了许多模块来完成cnf文件的书写工作。同时，在连续性约束代码的表达方面，我发现往往复杂的规律可以找到简洁的数学表达形式，这启发我面对一些难以一眼得到思路的问题时，不妨静下心来，由特殊到一般，慢慢寻找数学关系。这次的课程设计不仅仅是对知识的应用，更是对独立思考、团队合作和问题解决能力的锻炼，让我受益匪浅。

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

附录

```cpp
#include <iostream>
#include <cstring>
#include <fstream>
#include <string>
#include <ctime>
#include <unordered_map>

#include "DDPL_Implementation.h"

using namespace std;
ClauseSet *S;
const int sz = 10000;
char USELESS[10000];
Stack stack(sz);
Stack SingleValueStack(sz);
Stack RecoveryStack(s/*  */z);
int HaniMat[960] = {0};
int *ans;
int ClauseNum = 0;
int cpr[10] = {0,5,6,7,8,9,8,7,6,5};
int numust[10] = {0,1,3,5,7,9,7,5,3,1};
int rowbegin[10] = {0,0,5,11,18,26,35,43,50,56};
int cnt = 0;

//把0-61转化为蜂窝数独坐标 蜂窝数独每行分别有 5,6,7,8,9,8,7,6,5个格子
unordered_map<int , int> Hash = {
    {0,110},{1,120},{2,130},{3,140},{4,150},{5,210},{6,220},{7,230},{8,240},{9,250},{10,260},{11,310},
    {12,320},{13,330},{14,340},{15,350},{16,360},{17,370},{18,410},{19,420},{20,430},{21,440},{22,450},{23,460},
    {24,470},{25,480},{26,510},{27,520},{28,530},{29,540},{30,550},{31,560},{32,570},{33,580},{34,590},{35,610},
    {36,620},{37,630},{38,640},{39,650},{40,660},{41,670},{42,680},{43,710},{44,720},{45,730},{46,740},{47,750},
    {48,760},{49,770},{50,810},{51,820},{52,830},{53,840},{54,850},{55,860},{56,910},{57,920},{58,930},{59,940},
    {60,950}
```

```
int tag = 0;
//本程序对较大文件用时过长！！！
int main(){
    //文件读取
    clock_t start_t,finish_t;
    int n,m;//n为变元数，m为子句数
    string Line;

    string FileName;
    ifstream fpin;
    ofstream fpout;

    int op = 1;
    while(op){
        system("clear");
        cout << endl;
        cout << endl;
        cout << "       功能菜单 "<< endl;
        cout << "------------------------------------------------" << endl;
        cout << "1.DPLL判断cnf文件"<< endl;
        cout << "2.求解蜂窝数独" << endl;
        cout << "3.DPLL优化后大改cnf文件" << endl;
        cout << "0.退出"<< endl;
        cout << "------------------------------------------------" << endl;
        cout << "   请选择你的操作[0-2]:" << endl;
        cin >> op;

        switch(op){
            case 0:
                return 0;
            case 1:{

                cout << "Please input the file name: ";
                cin >> FileName;
                string Fin = FileName + ".cnf";
                fpin.open(Fin,ios::in);
                start_t = clock();
                if(!fpin.is_open()){
                    std::cerr<<"cannot open the file"<<endl;
                    return 0;
                }
                char c;
                while((c = fpin.get()) == 'c'){
                    fpin.getline(USELESS,10000); // 需要建立Useless数组，否则会报错
                }
                if(c == 'p'){
                    fpin >> Line >> n >> m;
                }
                //读取子句
                ans = new int[n+1];
                S = new ClauseSet;
                S->next = new ClauseSet;
                ClauseSet *pS = S->next;
                S->head = NULL;
                for(int i = 0 ; i < m ; i++){
                    pS->head = new Clauses;
                    Clauses *p = pS->head;
                    Clauses *pre = p;
                    p->value = 0;
                    p->next = new Clauses;
                    p = p->next;
                    if(fpin.eof()) break;
                    while((fpin >> p->value) && p->value != 0){
                        pS->head->value++;
                        // booleans[abs(p->value)]++;
                        p->next = new Clauses;
                        p = p->next;
                        pre = pre->next;
                    }
                    pre->next = NULL;
                    delete p;
                    if(i != m-1){
                        pS->next = new ClauseSet;
                        pS = pS->next;
                    }
                    else pS->next = NULL;
                }
                fpin.close();
                string Fout = "out/" + FileName + ".out";
                fpout.open(Fout,ios::out);
                if(!fpout.is_open()){
                    std::cerr<<"cannot open the file"<<endl;
                    return 0;
                }
                if(DPLL(S)){
                    fpout << "SATISFIED" << endl;
                    for(int i = 1 ; i < n+1 ; i++){
                        if(ans[i] == 1)
                            fpout << i << ' ';
                        else if(ans[i] == -1)
                            fpout << -i << ' ';
                        else fpout << "error" << endl;
                        if(i % 10 == 0) fpout << endl;
                    }
                }
                else{
                    fpout << "UNSATISFIED" << endl;
                }
                finish_t = clock();
                fpout << (double)(finish_t-start_t) / CLOCKS_PER_SEC << endl;
                fpout.close();
                cout << "Done" << endl;
                delete []ans;
                getchar();
                getchar();

            }
                break;
```

```
case 2:
    {
    // S = GenerateHaniddoku();
    // cout << "Sudoku print" << endl;
    // PrintHaniddoku(S);
    // if(!SolveHaniddoku(S)) cout << "UNSATISFIED" << endl;
    // else{
    //     cout << "SATISFIED" << endl;
    //     PrintHaniddoku(S);
    // }
    srand(time(NULL));
    int offset = rand()%100;
    char *HaniChar;
    HaniChar = new char[64];
    FILE *fp;
    fp = fopen("easy_hanidoku.txt","r");
    fseek(fp,offset*63,SEEK_SET);
    fgets(HaniChar,63,fp);
    int Hani[61];
    for(int i = 0 ; i < 61 ; i++) {
        Hani[i] = HaniChar[i] - '0';
    }
    int _answerH[61] = {0};
    // cout << "Please input the Haniddoku: ";
    // for(int i = 0 ; i < 60 ; i++){
    //     cin >> Hani[i];
    // }
    start_t = clock();
    PrintHaniddoku(Hani);
    S = Encoding(Hani);
    ans = new int[960];
    if(DPLL(S)){ //61 * 9 =

        string FileName = "Hani";
        fpout.open("out/" + FileName + ".out");
        int i,j;
        fpout << "s 0" << endl;
        for(i = 111, j = 0; i <= 959 ; i++){
            if(ans[i] > 0){
                fpout << i << ' ';
                _answerH[j++] = i%10;
            }
            if (ans[i] < 0)
                fpout << -i << ' ';
            if(i%10 == 0)
                fpout << endl;
        }

        PrintHaniddoku(_answerH);
        // Transition(_answerH,stack);
        // PrintHaniddoku(_answerH);
    }
    else{
        fpout << "s 0" << endl;
    }
    fpout << endl;
    finish_t = clock();
    fpout << (double)(finish_t-start_t) / CLOCKS_PER_SEC << endl;
    fpout.close();
    cout << (double)(finish_t-start_t) / CLOCKS_PER_SEC << endl;
    delete []ans;
    delete []HaniChar;
    getchar();
    getchar();
    }
    break;
case 3:{
    ClauseSet_h *Sh;
    cout << "Please input the file name: ";
    cin >> FileName;
    string Fin = FileName + ".cnf";
    fpin.open(Fin,ios::in);
    if(!fpin.is_open()){
        std::cerr<<"cannot open the file"<<endl;
        return 0;
    }
    char c;
    while((c = fpin.get()) == 'c'){
        fpin.getline(USELESS,10000); // 需要建立Useless数组，否则会报错
    }
    if(c == 'p'){
        fpin >> Line >> n >> m;
    }
    //读取子句
    positives = new Literal *[n+1];
    negatives = new Literal *[n+1]; // 初始化
    posClause = new Clauses_h *[n+1];
    negClause = new Clauses_h *[n+1];
    posSet = new ClauseSet_h *[n+1];
    negSet = new ClauseSet_h *[n+1];//不一定指向NULL
    //用new给指针数组分配空间，每个指针指向什么
    for(int i = 0 ; i < n+1 ; i++){
        positives[i] = NULL;
        negatives[i] = NULL;
        posClause[i] = NULL;
        negClause[i] = NULL;
        posSet[i] = NULL;
        negSet[i] = NULL;
    }
    ans = new int[n+1];
    Sh = new ClauseSet_h;
    Sh->next = new ClauseSet_h;
    ClauseSet_h *pS = Sh->next;
    Sh->head = NULL;
    for(int i = 0 ; i < m ; i++){
        pS->head = new Clauses_h;
        Clauses_h *p = pS->head;
        Clauses_h *pre = p;
        p->l = new Literal;
        p->l->value = 0;
        p->state = Exist;
        p->next = new Clauses_h;
        p = p->next;
```

BAK2.cpp > main()

```
                negClause = new Clauses_h *[n+1];
                posSet = new ClauseSet_h *[n+1];
                negSet = new ClauseSet_h *[n+1];//不一定指向NULL
                //用new给指针数组分配空间，每个指针指向什么
                for(int i = 0 ; i < n+1 ; i++){
                    positives[i] = NULL;
                    negatives[i] = NULL;
                    posClause[i] = NULL;
                    negClause[i] = NULL;
                    posSet[i] = NULL;
                    negSet[i] = NULL;
                }
                ans = new int[n+1];
                Sh = new ClauseSet_h;
                Sh->next = new ClauseSet_h;
                ClauseSet_h *pS = Sh->next;
                Sh->head = NULL;
                for(int i = 0 ; i < m ; i++){
                    pS->head = new Clauses_h;
                    Clauses_h *p = pS->head;
                    Clauses_h *pre = p;
                    p->l = new Literal;
                    p->l->value = 0;
                    p->state = Exist;
                    p->next = new Clauses_h;
                    p = p->next;
                    p->l = new Literal;
                    if(fpin.eof()) break;
                    while((fpin >> p->l->value) && p->l->value != 0){
                        //添加结点
                        p->state = Exist;
                        if(p->l->value > 0){
                            if(posClause[p->l->value] == NULL){
                                positives[p->l->value] = p->l;
                                posClause[p->l->value] = p;
                                posSet[p->l->value] = pS;
                                p->l->numnext = nullptr;
                                p->l->antinumnext = nullptr;
                            }
                            else{
                                positives[p->l->value] = p->l;
                                p->l->numnext = posSet[p->l->value];
                                posSet[p->l->value] = pS;
                                p->l->antinumnext = posClause[p->l->value];
                                posClause[p->l->value] = p;
                            }
                        }
                        else{
                            if(negClause[-(p->l->value)] == NULL){
                                negatives[-p->l->value] = p->l;
                                negClause[-p->l->value] = p;
                                negSet[-p->l->value] = pS;
                                p->l->numnext = nullptr;
                                p->l->antinumnext = nullptr;
                            }
                            else{
                                negatives[-p->l->value] = p->l;
                                p->l->numnext = negSet[-p->l->value];
                                negSet[-p->l->value] = pS;
                                p->l->antinumnext = negClause[-p->l->value];
                                negClause[-p->l->value] = p;
                            }
                        }
                        pS->head->l->value++;
                        p->next = new Clauses_h;
                        p = p->next;
                        p->l = new Literal;
                        pre = pre->next;
                    }
                    pre->next = NULL;
                    delete p->l;
                    delete p;
                    if(i != m-1){
                        pS->next = new ClauseSet_h;
                        pS = pS->next;
                    }
                    else pS->next = NULL;
                }
                fpin.close();
                string Fout = "out/" + FileName + ".out";
                fpout.open(Fout,ios::out);
                if(!fpout.is_open()){
                    std::cerr<<"cannot open the file"<<endl;
                    return 0;
                }
                start_t = clock();
                if(DPLLUpdation(Sh,n)){
                    fpout << "SATISFIED" << endl;
                    for(int i = 1 ; i < n+1 ; i++){
                        if(ans[i] == 1)
                            fpout << i << ' ';
                        else if(ans[i] == -1)
                            fpout << -i << ' ';
                        else fpout << "error" << endl;
                        if(i % 10 == 0) fpout << endl;
                    }
                }
                else{
                    fpout << "UNSATISFIED" << endl;
                }
                finish_t = clock();
                fpout << (double)(finish_t-start_t) / CLOCKS_PER_SEC << endl;
                fpout.close();
                cout << "Done" << endl;
                delete []ans;
                getchar();
                getchar();

            }
```

BAK2.cpp > main()

```
                negClause = new Clauses_h *[n+1];
                posSet = new ClauseSet_h *[n+1];
                negSet = new ClauseSet_h *[n+1];//不一定指向NULL
                //用new给指针数组分配空间，每个指针指向什么
                for(int i = 0 ; i < n+1 ; i++){
                    positives[i] = NULL;
                    negatives[i] = NULL;
                    posClause[i] = NULL;
                    negClause[i] = NULL;
                    posSet[i] = NULL;
                    negSet[i] = NULL;
                }
                ans = new int[n+1];
                Sh = new ClauseSet_h;
                Sh->next = new ClauseSet_h;
                ClauseSet_h *pS = Sh->next;
                Sh->head = NULL;
                for(int i = 0 ; i < m ; i++){
                    pS->head = new Clauses_h;
                    Clauses_h *p = pS->head;
                    Clauses_h *pre = p;
                    p->l = new Literal;
                    p->l->value = 0;
                    p->state = Exist;
                    p->next = new Clauses_h;
                    p = p->next;
                    p->l = new Literal;
                    if(fpin.eof()) break;
                    while((fpin >> p->l->value) && p->l->value != 0){
                        //添加结点
                        p->state = Exist;
                        if(p->l->value > 0){
                            if(posClause[p->l->value] == NULL){
                                positives[p->l->value] = p->l;
                                posClause[p->l->value] = p;
                                posSet[p->l->value] = pS;
                                p->l->numnext = nullptr;
                                p->l->antinumnext = nullptr;
                            }
                            else{
                                positives[p->l->value] = p->l;
                                p->l->numnext = posSet[p->l->value];
                                posSet[p->l->value] = pS;
                                p->l->antinumnext = posClause[p->l->value];
                                posClause[p->l->value] = p;
                            }
                        }
                        else{
                            if(negClause[-(p->l->value)] == NULL){
                                negatives[-p->l->value] = p->l;
                                negClause[-p->l->value] = p;
                                negSet[-p->l->value] = pS;
                                p->l->numnext = nullptr;
                                p->l->antinumnext = nullptr;
                            }
                            else{
                                negatives[-p->l->value] = p->l;
                                p->l->numnext = negSet[-p->l->value];
                                negSet[-p->l->value] = pS;
                                p->l->antinumnext = negClause[-p->l->value];
                                negClause[-p->l->value] = p;
                            }
                        }
                        pS->head->l->value++;
                        p->next = new Clauses_h;
                        p = p->next;
                        p->l = new Literal;
                        pre = pre->next;
                    }
                    pre->next = NULL;
                    delete p->l;
                    delete p;
                    if(i != m-1){
                        pS->next = new ClauseSet_h;
                        pS = pS->next;
                    }
                    else pS->next = NULL;
                }
                fpin.close();
                string Fout = "out/" + FileName + ".out";
                fpout.open(Fout,ios::out);
                if(!fpout.is_open()){
                    std::cerr<<"cannot open the file"<<endl;
                    return 0;
                }
                start_t = clock();
                if(DPLLUpdation(Sh,n)){
                    fpout << "SATISFIED" << endl;
                    for(int i = 1 ; i < n+1 ; i++){
                        if(ans[i] == 1)
                            fpout << i << ' ';
                        else if(ans[i] == -1)
                            fpout << -i << ' ';
                        else fpout << "error" << endl;
                        if(i % 10 == 0) fpout << endl;
                    }
                }
                else{
                    fpout << "UNSATISFIED" << endl;
                }
                finish_t = clock();
                fpout << (double)(finish_t-start_t) / CLOCKS_PER_SEC << endl;
                fpout.close();
                cout << "Done" << endl;
                delete []ans;
                getchar();
                getchar();

            }
```

```
bool DPLL(ClauseSet *S){
    //找单子句
    ClauseSet *pS = S->next , *preS = S;
    Clauses *pC, *preC;

    while(pS){
        pC = pS->head;
        if(pC->value == 0){
            if(pS->next == NULL) preS->next = NULL;
            else preS->next = pS->next;
            deletion(pS);
            pS = preS->next;
            continue;
        }
        if(pC->value == 1){
            //单子句传播
            tag = pC->next->value;
            // booleans[abs(tag)] = 0;
            // if(pS->next == NULL) preS->next = NULL;
            // else preS->next = pS->next;
            // delete pS->head->next;
            // delete pS->head;
            // delete pS;
            if(tag > 0) ans[tag] = 1;
            else ans[-tag] = -1;
            pS->head->value = 0;
            if(!SVS(S,tag)) {
                return false;//SingleValueSpread
            }
            preS = S;
            pS = S->next;
        }
        else{
            preS = pS;
            pS = pS->next;
        }
    }
    if(S->next == NULL) return true;
    //找到单子句 进行单子句传播 (用value = 0做标记为删除)
    //如果传播后有空子句(即删除了-tag后为空) 返回false
    //每次找到单子句并处理后都要从头扫描单子句 若headvalue = 0 则删除该子句 (注意更新数组)
    //若从头扫描到尾都没有单子句 进行下一步
    //排序
    // SORT(booleans,labels,n);
    //添加新的变元 (数量最多)
    int v = S->next->head->next->value;
    // int TempBool[10000] = {0};
    // for(int i = 1; i <= n ; i++){
    //     TempBool[i] = booleans[i];
    // }
    //复制
    ClauseSet *TempS;
    TempS = new ClauseSet;
    TempS->head = NULL;
    ClauseSet *pp = new ClauseSet;
    Clauses *ppc;
    pp->head = new Clauses;
    ppc = pp->head;
    ppc->value = 1;
    ppc->next = new Clauses;
    ppc = ppc->next;
    ppc->next = NULL;
    ppc->value = v;
    pp->next = S->next;
    S->next = pp;
    // if(stack.push(v)){
    //     cout << "Stack Overflow" << endl;
    //     exit(EXIT_SUCCESS);
    // }
    Copy(S,TempS);
    TempS->next->head->next->value = -v;
    if(DPLL(S)) return true;
    //回溯 需要回溯booleans
    else{
        //  for(ClauseSet *delS = S->next ; delS ;){
        //      if(delS->next == NULL) S->next = NULL;
        //      else S->next = delS->next;
        //      deletion(delS);
        //      delS = S->next;
        // }
        // delete S;
        //此处未完成冗余删除!!!!
        S = TempS;
    //     for(int i = 1; i <= n ; i++){
    //     booleans[i] = TempBool[i];
    // }
        return(DPLL(S));
    }

}
```

```
void Copy(ClauseSet *S, ClauseSet *TempS){

    ClauseSet *p = S->next,*q;
    Clauses *p1, *q1;
    q = TempS;
    while(p){
        q->next = new ClauseSet;
        q = q->next;
        p1 = p->head;
        q->head = new Clauses;
        q1 = q->head;
        q1->value = p1->value;
        p1 = p1->next;
        while(p1){
            // cout << "p1 :"<< p1->value<<endl;
            q1->next = new Clauses;
            q1 = q1->next;
            q1->value = p1->value;
            // cout << "q1:" << q1->value << endl;
            p1 = p1->next;
        }
        q1->next = NULL;
        p = p->next;
    }
    q->next = NULL;
}

// void SORT(int *booleans, KV *labels,int n){
//     for(int i = 0 ; i <= n ; i++){
//         labels[i].index = i;
//         labels[i].value = booleans[i];
//     }
//     sort(labels, labels + n + 1, cmp);
// }
//指针未赋值?
void deletion(ClauseSet *S){

    if(S->head == NULL){
        delete S;
        return;
    }
    Clauses *pre = S->head;
    Clauses *p = pre->next;
    while(p){
        if(p->next == NULL) pre->next = NULL;
        else pre->next = p->next;
        delete p;
        p = pre->next;
    }
    delete pre;
    delete S;
}

bool SVS(ClauseSet *S,int tag){
    ClauseSet *pS = S->next;
    Clauses *pC, *preC;
    while(pS){
        preC = pS->head;
        if(preC->value == 0){
            pS = pS->next;
            // cout << "pS->head->value == 0" << endl;
            continue;
        }
        pC = preC->next;
        while(pC){
            if(pC->value == tag){
                pS->head->value = 0;
                // for(preC = pS->head->next ; preC ; preC = preC->next){
                //     if(tag != preC->value){
                //         booleans[abs(preC->value)]--;
                //     }
                // }
                break;
            }
            else if(pC->value == -tag){
                pS->head->value--;
                if(pS->head->value == 0)
                    return false;
                if(pC->next == NULL) preC->next = NULL;
                else preC->next = pC->next;
                delete pC;
                pC = preC->next;
            }
            else{
                pC=pC->next;
                preC = preC->next;
            }
        }
        pS=pS->next;
    }
    return true;
}


//优化方向 递归和复制 (假删除)
```

```
// 1: 1-9
// 2: Row and diagnol must be filled with consecutive numbers(it's not ok to fill i
//      2.1 Size 5: 12345 23456 34567 45678 56789
// 3. 只有5是必选

void PrintHaniddoku(int Hani[61]){
    cout << "The Haniddoku is: " << endl;
    int i;
    cout << "        |";
    for(i = 0 ; i < 5 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "      |";
    for(; i < 11 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "    |";
    for(; i < 18 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "  |";
    for(; i < 26 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "|";
    for(; i < 35 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "  |";
    for(; i < 43 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "    |";
    for(; i < 50 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "      |";
    for(; i < 56 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    cout << "        |";
    for(; i < 61 ; i++){
        cout << ' '<< Hani[i] << " |";
    }
    cout << endl;
    return;
}

void Transition(int _answerH[61], Stack stack){
    int temp;
    while(!stack.isEmpty()){
        stack.pop(temp);
        int row = temp / 100;
        int col = (temp % 100) / 10;
        int num = temp % 10;
        _answerH[rowbegin[row]+col-1] = num;
    }
    return;
}



bool DPLLHaniddoku(ClauseSet *S){
    //找单子句
    ClauseSet *pS = S->next , *preS = S;
    Clauses *pC, *preC;
    int index = 0;
    while(pS){
        pC = pS->head;
        if(pC->value == 0){
            if(pS->next == NULL) preS->next = NULL;
            else preS->next = pS->next;
            deletion(pS);
            pS = preS->next;
            continue;
        }
        if(pC->value == 1){
            //单子句传播
            tag = pC->next->value;
            // booleans[abs(tag)] = 0;
            // if(pS->next == NULL) preS->next = NULL;
            // else preS->next = pS->next;
            // delete pS->head->next;
            // delete pS->head;
            // delete pS;

            pS->head->value = 0;
            if(!SVS(S,tag)) {
                return false;
            }//SingleValueSpread
            preS = S;
            pS = S->next;
        }
        else{
            preS = pS;
            pS = pS->next;
        }
    }
    if(S->next == NULL) return true;
    //找到单子句 进行单子句传播 (用value = 0做标记为删除)
    //如果传播后有空子句(即删除了-tag后为空) 返回false
    //每次找到单子句并处理后都要从头扫描单子句 若headvalue = 0 则删除该子句 (注意更新数组)
    //若从头扫描到尾都没有单子句 进行下一步
    //排序
    // SORT(booleans,labels,n);
```

```
//每次找到单子句并处理后都要从头扫描单子句 若headvalue = 0 则删除该子句 (注意更新数组)
//若从头扫描到尾都没有单子句 进行下一步
//排序
// SORT(booleans,labels,n);
//添加新的变元 (数量最多)
int v = S->next->head->next->value;
// int TempBool[10000] = {0};
// for(int i = 1; i <= n ; i++){
//     TempBool[i] = booleans[i];
// }
//复制


ClauseSet *TempS;
TempS = new ClauseSet;
TempS->head = NULL;
ClauseSet *pp = new ClauseSet;
Clauses *ppc;
pp->head = new Clauses;
ppc = pp->head;
ppc->value = 1;
ppc->next = new Clauses;
ppc = ppc->next;
ppc->next = NULL;
ppc->value = v;
pp->next = S->next;
S->next = pp;
Copy(S,TempS);
TempS->next->head->next->value = -v;
if(DPLLHaniddoku(S)) return true;
//回溯 需要回溯booleans
else{
    //  for(ClauseSet *delS = S->next ; delS ;){
    //      if(delS->next == NULL) S->next = NULL;
    //      else S->next = delS->next;
    //      deletion(delS);
    //      delS = S->next;
    // }
    // delete S;
    //此处未完成冗余删除！！！！
    S = TempS;
//     for(int i = 1; i <= n ; i++){
//     booleans[i] = TempBool[i];
// }
    return(DPLLHaniddoku(S));
}


auseSet *Encoding(int Hani[61]){
//Initializing
ClauseSet *Han = new ClauseSet;
Han->head = NULL;
Han->next = NULL;
AddCellConstriant(Han);
AddRowConstriant(Han);
AddLDiagConstriant(Han);
AddRDiagConstriant(Han);
int i = 0;
while(i < 61){
    if(Hani[i] == 0){
        i++;
        continue;
    }
    ClauseSet *pS = new ClauseSet;
    pS->next = Han->next;
    pS->head = new Clauses;
    pS->head->value = 1;
    pS->head->next = new Clauses;
    pS->head->next->value = Hash[i]+Hani[i];
    pS->head->next->next = NULL;
    Han->next = pS;
    i++;
    ClauseNum++;
}
return Han;

Check 1

id AddCellConstriant(ClauseSet *Han){
int r = 1;

while(r <= 9){
    for(int c = 1 ; c <= cpr[r]; c++){
        for(int i = 1 ; i <= 8 ; i++){
            for(int j = i + 1 ; j <= 9 ; j++){
                //先放负值
                ClauseSet *pS;
                pS = new ClauseSet;
                pS->next = Han->next;
                Clauses *pC = new Clauses;
                int show = -(100 * r +10 * c);
                pC->value = 2;
                pC->next = new Clauses;
                pC->next->value = show - i;
                pC->next->next = new Clauses;
                pC->next->next->value = show - j;
                pC->next->next->next = NULL;
                ClauseNum++;
                pS->head = pC;
                Han->next = pS;
            }
        }
        //再放正值
        ClauseSet *pS;
        pS = new ClauseSet;
        pS->next = Han->next;
        Clauses *pC = new Clauses;
        int show = 100 * r +10 * c;
        pC->value = 9;
        pS->head = pC;
        for(int i = 1; i <= 9 ; i++ ){
            pC->next = new Clauses;
            pC = pC->next;
```

```
void AddRowConstriant(ClauseSet *Han){
    int r = 1;
    ClauseSet *pS;
    while(r <= 9){
        //放入约束项
        for(int i = 1 ; i <= 9 ; i++){
            for(int c1 = 1 ; c1 <= cpr[r] - 1 ; c1++){
                for(int c2 = c1 + 1; c2 <= cpr[r] ; c2++ ){
                        pS = new ClauseSet;
                        pS->next = Han->next;
                        Clauses *pC = new Clauses;
                        pC->value = 2;
                        pC->next = new Clauses;
                        pC->next->value = -(100 * r + 10 * c1 + i);
                        pC->next->next = new Clauses;
                        pC->next->next->value = -(100 * r + 10 * c2 + i);
                        pC->next->next->next = NULL;
                        pS->head = pC;
                        Han->next = pS;
                        ClauseNum++;
                }
            }
        }
        //放入必填项 cpr[r] = 5 -> 5必填 / cpr[r] = 6-> 4,5,6必填 / cpr[r] = 7-> 3,4,5,6,7必填
        // cpr[r] = 8-> 2,3,4,5,6,7,8必填 / cpr[r] = 9-> 1,2,3,4,5,6,7,8,9必填
        int turn = numust[r];
        int num = 5;
        int flag = 1;

        while(turn){
                pS = Add_must_fill(r,num);
                pS->next = Han->next;
                Han->next = pS;
                ClauseNum++;
                if(turn%2){
                    num+=flag;
                }
                else num-=flag;
                flag++;
                turn--;
        }
        //填入选择项
        // 1234 2346 3467 4678 6789
        ch_row[r](Han);
        // ClauseNum
        r++;
    }
}

void AddLDiagConstriant(ClauseSet *Han){
    int flag = 1;
    ClauseSet *pS;
    Clauses *pC;
    int rowj , rowl;
    int base;
    for(int d = 1; d <= 9 ; d++){
        base = LDiag[d] / 10;
        for(int i = 1 ; i <= 9 ; i++){
            for(int j = 0 ; j < cpr[d] - 1  ; j++){
                rowj = LDiag[d] / 10 + j;
                for(int l = j + 1 ; l < cpr[d] ; l++){
                    rowl = LDiag[d] / 10 + l;

                    pS = new ClauseSet;
                    pS->next = Han->next;
                    pS->head = new Clauses;
                    pC = pS->head;
                    pC->value = 2;
                    pC->next = new Clauses;
                    pC = pC->next;
                    if(rowj <= 5)
                        pC->value = - (10*(LDiag[d]+ 11 * j) + i);
                    else
                        pC->value = - (10 * (LDiag[d] + 11 * (5 - base)  + 10 * (rowj - 5)) + i);
                    pC->next = new Clauses;
                    pC = pC->next;
                    pC->next = NULL;
                    if(rowl <= 5)
                        pC->value = - (10*(LDiag[d]+11 * l) + i);
                    else
                        pC->value = - (10 * (LDiag[d] + 11 * (5-base) + 10 * (rowl - 5)) + i);
                    Han->next = pS;
                    ClauseNum++;
                    }
                }
        }

        //放入必填项
        int turn = numust[d];
        int num = 5;
        int flagg = 1;
        while(turn){
            pS = Add_must_fill(d,num,flag);
            pS->next = Han->next;
            Han->next = pS;
            ClauseNum++;
            if(turn%2){
                num+=flagg;
            }
            else
                num-=flagg;
            flagg++;
            turn--;
        }

        //放入选填项
        ch_ldiag[d](Han);
}
```