

**课程设计报告**

**题目：基于SAT的数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS2206**

**学 号： U202215525**

**姓 名： 冯子潇**

**指导教师： 纪俊文**

**报告日期： 2023年10月12日**

**计算机科学与技术学院**

**任务书**

* **设计内容**

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

* **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

**1引言** 1

1.1课题的背景与意义 1

1.2国内外研究现状 2

1.3课程设计的主要研究工作 3

**2系统需求分析与总体设计** 6

2.1系统需求分析 5

2.2系统总体设计 5

**3系统详细设计** 7

3.1有关数据结构的定义 7

3.2主要算法设计 8

3.2.1 SAT模块 8

3.2.2蜂窝数独游戏模块 9

**4系统实现与测试** 12

4.1系统实现 12

4.1.1 系统功能说明 12

4.1.2 常量与结构体声明 12

4.1.3 函数声明 14

4.2系统测试 12

4.2.1 SAT模块测试 23

4.2.2 数独模块测试 27

**5总结与展望** 32

5.1全文总结 32

5.2工作展望 33

**5体会** 34

**参考文献** 36

**附录** 37

**1引言**

**1.1课题背景与意义**

命题逻辑公式的可满足性问题（SAT）是数理逻辑、计算机科学、集成电路设计与验证和人工智能等领域中的核心问题，并且是第一个被证明出来的NP问题。SAT问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该问题的高效算法意义重大。

目前解决该问题的方法主要有完备的方法和不完备的方法两大类。完备的方法优点是保证能正确地判断SAT问题的可满足性，但其计算效率很低，平均的计算时间为多项式阶，最差的情况计算时间为指数阶，不适用于求解大规模的SAT问题。不完备的方法的优点是求解的时间比完备的方法快得多，但在很少数的情况下不能正确地判断SAT问题的可满足性。

从1960年至今，对于SAT问题的研究从没有停止过，世界各国的研究人员在这方面都做了大量的工作，提出了许多求解算法。每年可满足性理论和应用方面的国际会议都会组织一次SAT竞赛以求找到一组最快的SAT求解器，而且会详细展示一系列的高效求解器的性能。在1997年和2003年，H.Kautz与B.Selman两次列举出SAT搜索面临的挑战性问题，并于2011年和2007年，两度对当时的SAT问题研究现状进行了全面的综述。黄文奇提出的Solar算法在北京第三届SAT问题快速算法比赛中获得第一名。2003 年的SAT竞赛中，就有30多种解决方案针对从成千上万的基准问题中挑选出的一些SAT问题实例同台竞争。国内也经常会组织一些SAT竞赛及研讨会，这些都促进了SAT算法的飞速发展。尽管命题逻辑的可满足性问题理论研究已趋于成熟，但在SAT求解器被越来越多地应用到各种实际问题领域的今天，探寻解决SAT问题的高效算法仍然是个吸引人并且极具挑战性的研究方向。

SAT问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。对SAT问题的求解，可用于解决计算机和人工智能领域内的CSP问题（约束满足问题）、语义信息的处理和逻辑编程等问题，也可用于解决计算机辅助设计领域中的任务规划与设计、三维物体识别等问题。除此之外工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可转化成SAT问题。因此致力于寻找求解SAT问题的快速而有效的算法，不仅在理论研究上而且在许多应用领域都具有极其重要的意义。

**1.2国内外研究现状**

国际 SAT 算法竞赛从 1992 年开始，每一到两年举办一次，至今已经举办了二十届。本次挑战赛受到国际学术界和工业界的顶级研究人员的热烈关注。共有来自 14 个国家的 28 支队伍参加比赛。参赛单位包括纽约大学，威斯康辛大学，加拿大滑铁卢大学，德国林兹大学，比利时鲁汶大学，悉尼科技大学等。国内也有清华大学，北京大学，中科院等高校和研究所参加本次竞赛。本次竞赛共有Main、Parallel、Incremental、Agile、Random、No-limits 等 6 个组别，其中 Main组是针对于目前各类工业界难解问题的求解，是 SAT 竞赛中受关注度最高和竞争最激烈的组别。

华中科技大学计算机学院团队斩获第20届国际SAT算法竞赛冠军,除此之外,还有很多种计算 SAT 的求解器,主要有: GRASPI、 zChaff、 BerkMin 和MiniSATI9I 等，这些求解器几乎都是在 DPLL 算法或在对该算法进行优化得到

的算法的基础上实现的。DPLL算法是一种判定 SAT 问题的高效算法。早在 1960 年，Davis 和 Putnam就提出了最早的DPLL算法，当时称为DP算法，该算法的提出有效地降低了SAT问题的复杂性和求解器的空间限制这些棘手的情况，奠定了 DPLL 算法在判定SAT 问题时的地位。DP 算法通过在给定的表达式中进行变量消解，达到了降低搜索空间大小的目的。但当 Logemann 和 Loveland 尝试实现 DP 算法时，发现该算法在消解时占用过多的内存空间,这在当时的条件下是受限的。于是在1962年，Logemann 和 Loveland 等人对 DP 算法进行了改进，改变了变量消解的方式，形成了最初的 DPLL 算法叫。此时的 DPLL 算法采用的是分支回溯策略，即不断地选择变量进行分支赋值，当发生冲突时再进行回溯。这种 DPLL 算法求解效率较低，最多只能解决 10 变量的 SAT 问题，且求解范围受限，对随机生成的实例求解效果较好，对实际应用转化而来的实例求解效果不佳。1996年,Marques-Silva和Sakallah提出了GRASPl4l算法。在实际求解问题时，该算法可以尽早剪除不满足搜索空间,极大地降低了搜索时间，提高了求解效率。1997 年，H. Zhang 提出了 SATO 算法,它是在 DPLL 算法的基础\_上采用了智能回溯策略、搜索重新启动策略和 BCP 数据结构等，该算法的实现降低了原有的 SAT 求解器的求解:时间，并解决了之前不能处理的一些 SAT 问题。2001年，L.Zhang提出了zChaff算法，优化后的zChaff算法有效地提高了 BCP 推导的效率，找到了较为高效的学习方式，使得应用该算法解决 SAT 问题的效率实现了质的飞跃,奠定了其在 SAT 算法发展中的重要作用。使得在这之后的很多SAT 算法都以它为蓝本。2005 年，EenN和SorenssonN提出了 SAT 算法MiniSATI9I,这是迄今为止各方面性能都较好的 SAT 求解器。在 2006 年的 SAT国际竞赛中，该算法夺冠。随着研究的深入，优化技术的不断改进，DPLL 算法亦趋于完善。但无论DPLL 算法结合何种优化技术，在判定 SAT 问题时，总会有约 80%的时间耗费在 BCP 过程中,而判定变量的选取结果直接影响该过程的运行时间，所以判定变量选取策略在 DPLL 算法的优化技术研究中占有重要的地位，这也就成为了历年来 DPLL 算法优化领域的热门课题之一。

**1.3课程设计的主要研究工作**

主要研究工作已经在上面任务书里面有提及，主要是我们学习命题逻辑可满足性问题的相关理论知识，并对基于DPLL算法的sat问题求解器的关键技术和框架进行了研究与实现。

1. 在了解sat问题的发展，应用的基础上，学习CNF范式的基本理论知识，掌握解决命题逻辑可满足性问题的一般策略。

2. 精心设计数据结构来存储CNF范式的文字和语句。

3. 实现DPLL算法的递归过程，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间，并对分支变元的选取策略进行优化，给出相应问题的优化效率。

4. 求解出sat问题的答案，生成同名.res文件。

5. 将SAT求解器用于蜂窝数独的求解中。

6. 实现蜂窝数独游戏的求解与可玩性的交互，例如输入，检查输入是否正确，给与提示等。

大致的操作可以总结如下:

（1）阅读“程序设计”综合课程设计任务书，熟悉问题，查阅文献，了解问题背景及相关知识。

（2）对设计问题进行需求分析，分析解决问题需要的功能并进行划分，人

机交互需求与数据文件读写等，并对问题进行形式化表示。

（3）基于上述需求分析，进行系统设计，明确程序的模块结构；设计数据结构（逻辑结构及其物理结构），参考并设计主要子问题的求解算法。

（4）程序实现，基于系统设计，制定相应的实现方案，编写各程序模块，完成程序编写与调试任务。

（5）程序测试，设计测试用例对程序进行功能测试，性能测量及理论分析。程序优化，对设计方案中的结构，算法进行一定优化，测试与分析性能改善结果。在设计报告中明确说明优化策略与方案。

（6）设计总结，按规范化要求撰写“程序设计”综合课程设计报告。

**2系统需求分析与总体设计**

**2.1系统需求分析**

设计问题中变元、文字、子句、公式等有效的物理储存结构，实现SAT问题求解与数独游戏格局生成与求解两大功能。

首先需要对cnf文件进行读取，同时需要遍历各个子句，以显示读取操作的正确性。在这样的数据结构上，需要基于DPLL算法框架对SAT算例进行求解，实现结果的显示，并将结果写入文件中。此外，记录DPLL过程的运行时间，并计算优化算法后的优化效率。

对于蜂窝数独游戏，需要将蜂窝数独游戏转化为SAT问题，以便我们将SAT求解器运用到其中。可以直接以文件形式读取游戏，并求解输出答案，也可以通过挖空生成具有唯一解的游戏，不仅具备输入答案，检查输入值是否正确，此外还可以生成不同难度的数独格局，并给与提示等，使游戏具有简单的交互性。

**2.2系统总体设计**

系统包括两个模块，分别是SAT问题的求解和蜂窝数独游戏的求解。在程序开始时，根据用户的选择来执行相应模块的内容。

SAT模块包括：

1.从文件中读取CNF范式，并储存到数据结构中；

2.遍历所有子句；

3.利用DPLL算法求解SAT问题，并将结果写入文件中；

4.打印求解得到的答案；

5.验证所求解的答案是否正确；

共五个部分。

蜂窝数独游戏模块包括：

1.解决已有的蜂窝数独，以文件形式输入，求解打印答案，并将答案写入文件中；

2.基于挖洞法生成有唯一解的数独游戏，有简单的交互性，例如输入答案，检查答案是否正确，提供提示等；

共两个部分。

系统模块流程图如下：



图2.2-1 cnf公式存储结构图

模块流程的设计主要参照了任务书中已经给出的内容。

**3系统详细设计**

**3.1有关数据结构的定义**

在SAT模块中，首先需要按照格式处理文件中的各个子句与变元的存储。由于每个CNF公式变元与子句数目不尽相同，子句长度也不尽相同；此外，在算法设计中，会运用到添加子句与删除子句的操作，所以采用一链表的数据结构，将子句表示为文字构成的链表，整个公式则是由子句构成的链表，如图3.1所示。



图3.1-1 cnf公式存储结构图

采用结构体的形式来保存问题、子句与文字。

在子句结构体中，包括有两个int型变量，两个指针变量。两个int型变量分别用于表示子句中文字的个数，以及从公式中屏蔽子句时的递归深度，未被屏蔽时其值为0，便于回溯时将其复原；两个指针变量则分别指向下一个子句与子句中第一个文字，没有所指时指针值为NULL。

在文字结构体中，包括有两个int型变量，一个指针变量。两个int型变量分别用于表示带有符号的文字名，以及从子句中屏蔽文字时的递归深度，未被屏蔽时其值为0，便于回溯时将其复原；指针变量则指向下一个文字，没有所指时表示为NULL。

为了方便我们记录问题的具体情况，特别引入了问题结构体，其中包括记录cnf文件中文字数量与子句数量的变量，指向第一个子句的root指针，以及用于统计文字出现次数与记录答案的数组。

在数独模块中，会运用到与SAT模块中类似的数据结构。同样地，为了方便我们记录问题与统计答案，数独问题也采用结构体的形式，其中包含一个用于记录挖洞个数或非0个数的变量，以及两个二维数组，用于记录问题格局与答案。

表3-1 数据项及数据类型

|  |  |  |
| --- | --- | --- |
| 数据 | 记录方法 | 数据项（数据类型） |
| Sat问题 | 结构 | 文字数量和子句数量（int型） |
| 子句根节点（子句指针）  各元素出现次数（int型数组）  答案（int型数组） |
| 蜂窝数独问题 | 原始数值和答案（int型数组）  标志防止误输入（int型数组）  可输入个数（int型数组） |
| 子句 | 递归深度（int型）  文字数量（int型）  子句和文字指针（指针） |
| 文字 | 递归深度（int型）  文字数值（int型）  文字指针（指针） |

**3.2 主要算法设计**

## 3.2.1 SAT模块

CNF文件处理：

首先初始化问题结构体，为文件读入做准备。

读入cnf文件，将各个子句的情况记录在数据结构中。之后可以调用函数展示所有子句，以验证读入过程的正确性。

DPLL过程：

在子句集中寻找单子句，假设找到单子句为L，更新答案数组。再根据单子句规则，屏蔽子句集中所有包含L的句子，同时屏蔽所有¬L的文字，更新子句中文字的个数。执行屏蔽操作时，更新各个变元出现的次数。

（屏蔽：将对应数据赋值为递归深度，初始深度为1）

之后，根据分裂规则，基于“选取最短子句中的第一个文字”、“选取最短子句中出现次数最多的文字”与“寻找当前首个文字”三种变元选取策略，假设选取到的变元为真，将其作为一个单子句插入到子句集中，再执行单子句传播规则。

如此往复，直至无法寻找到下一个可赋值的变元时，表明函数执行完毕；如果在分裂规则中出现了空子句，则表示出现了冲突，递归调用返回至上一级，恢复在上一级屏蔽的子句与文字，更新各个文字出现的次数。假设该文字的反面为真值，再次进入递归，以此类推，直至求解成功。

当所有的路径都无法求解成功时，函数执行完毕，原cnf范式不可满足。

保存求解结果并显示范式是否可以满足，同时显示求解时间，以反应程序的性能。

文件操作、答案输出与验证：

将问题结构体中数组所保存的答案录入同名但以.res结尾的文件中，文件中同时保存求解结果，求解答案与求解时间。同时，也可以选择直接输出答案。此外，通过遍历每一个子句并访问子句中文字的答案，来判断所求解的答案是否正确，并给出结果。

## 3.2.2 蜂窝数独游戏模块

蜂窝数独文件读入：

从文件中读入一个设计好的数独数组，此时数组按照原本的行作为行进行记录，初始化每个位置的代号ij，i代表行号，j代表列号，后续只用更换代号就可以将横行转化为斜行。其中没有填入数字的位置可以用任意的一个字符占位，均可以正确完成读入。

CNF范式归约：

将变元可按语义编码为1～9之间数字构成的三位整数ijk，i, j, k∈{1,2,…,9}，其中i表示单元格的行号，j表示单元格的列号，k表示单元格<i, j>填入的数字为k。负文字则表示不填入。再通过一定的方式将用语义表示的变元切换为连续的变元。

接下来处理横行约束，左斜行约束与右斜行约束。

首先，对于每一个小格而言，填入的数字应当是唯一的，也就是对于某一确定的格子，表明填入任意两个数字的变元不能同时为真

其次，对于每一行中，任意的两格不能填入相同的数字。

对于格子数不同的行约束规则有一定差别。

以横行约束为例。第五行一定有数字1-9，所以1-9约束为必填数字，要求该行必含数字1-9；第四行和第六行必填数字为2-8，则约束该行必含数字2-8，可选填数字组1或9；第三行和第七行必填数字为3-7，则约束该行必含数字3-7，可选填数字组为1-2，2和8或8-9；第二行和第八行必填数字为4-6，则约束该行必含数字4-6，可选填数字组为1、2、3或2、3、8或3、8、9；最后，第一行和第九行，约束该行必含数字5，可选填数字组为1、2、3、4或2、3、4、6或3、4、6、7或4、6、7、8或6、7、8、9。

将数组不同位置的代号更换，使原本左斜行变成横行（将左斜行当作横行重新记录每个位置的代号，相当于将图形旋转），所有的约束条件与横行一模一样，右斜行同理。

将以上条件全部处理成子句的形式，录入一个cnf文件中。

之后，再根据读入的数独格局，在cnf文件中写入单子句。

蜂窝数独格局生成：

基于挖洞法生成一个具有唯一解的数独。

首先，随机读入一个求解蜂窝数独的文件，使用DPLL过程求出其解，将其保存为最后的答案。之后随机地从数独中删除数字，将该数字删除后，把该位置的其它8个数字代入数独中进行验算，如果有解，则代表删除该数字后数独将不会再是唯一解，该位置的数字不能删除，找寻新的可删除位置。

每删除一个数字，缺空数即加一，直到增加到所设定的难度值停止挖空，打印除蜂窝数独。

实现可玩性与交互性：

用户可以输入填写位置与填入数据对数独格局进行操作，程序将给出输入数值的正确性，也可以随机直接提示答案。

文件输出：

可以将求解出的数独答案保存至同名但答案为.res一个文件中。同时，也可以选择直接输出答案。

**4系统实现与测试**

**4.1系统实现**

## 4.1.1 系统功能说明

系统包括两个部分：SAT和Sudoku。SAT是求解命题逻辑公式的可满足性问题，需要用户输入给定的cnf文件，通过SAT问题求解器来求解。Sudoku是求解数独游戏，需要用户输入棋盘的初始条件，系统自动将其转化为cnf子句集，调用SAT求解器，求解该数独游戏。

选择1.SAT SAT问题后，系统会出现SAT界面，如图4.1所示。

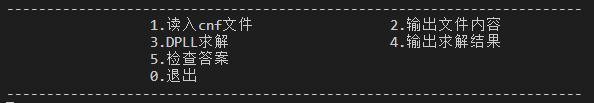


图4.1.1 SAT界面操作示意图

其中：

1.读入cnf文件：能从cnf文件中读取数据，为结构体分配存储空间，将数据存到结构体中；

2.输出文件内容：能够将结构体中的数据打印出来，显示为cnf字句集；

3.DPLL求解：是调用DPLL算法，求解该 sat问题,调用DPLL算法的时候同时显示算法的运行时间，调用完毕后会自动生成同名的res文件；

4.输出求解结果：能够将结果打印在显示屏上；

5.检查答案：能够检测DPLL算法求得的结果是否正确；

选择2.Sudoku 蜂窝数独后，系统会出现界面Sudoku界面，如图4.2所示。



图4.1.2 Sudoku界面操作示意图

其中：

1.求解数独sudoku solve能从数独文件中读取数独格局，并将其转化，形成同名的cnf文件，再调用SAT求解器对其求解。可以显示求解答案，也可以将数独答案显示到

2.生成数独 sudoku game可以随机挑选一个求解数独文件生成一个数独，并通过挖洞的方法来构建可玩的数独格局，用户可以选择不同的难度，对应数独中留空的数量。

## 4.1.2 常量与结构体声明

程序中的常量与结构体声明如下：

#define MAX\_STNUM 20000

#define MAX\_NUM 2000

#define ERROR 0

#define OK 1

#define false 0

#define true 1

typedef int status;

typedef struct element { //保存元素

int flag; //判断是否被删除（0代表未被删除，其余数字代表被删除时的递归层数）

int value; //记录元素(有正负)

struct element\* nextnode; //下一元素变量

}element;

typedef struct clause { //记录子句

int flag; //标记子句真假，mark=0表示子句为真,其他情况表示删除时的递归深度

int literal\_num; //记录子句中元素的个数

struct clause\* nextcla; //指向下一个子句

struct element\* first; //指向第一个元素

}clause;

typedef struct problem { //SAT问题

int vexnum, clanum; //记录布尔变元数量vexnum、子句数量clanum

struct clause\* root; //root结点中不存放数据

int element[MAX\_NUM + 1]; //各元素出现的次数（在无单子句而要选下一个删除关键字时使用）

int ans[MAX\_NUM + 1]; //（有解时的）解

}problem;

typedef struct sudproblem { //数独问题

int original[10][10]; //记录原始数值

int finalans[10][10]; //答案

int flag[10][10]; //防止玩家输错位置

int num; //非0个数

}sud;

## 4.1.3 函数声明

SAT模块的主要函数：

（1）void SATQuestion() SAT问题总页面

该函数会打印SAT问题的菜单。调用void \_ReadFile(char filename[200])读取文件；调用PrintFile打印文件子句；调用DPLL()进行求解；调用PrintAnswer()打印答案；调用CheckAnswer()检查答案是否正确。

（2）void init(); //初始化问题

函数无参数，无返回值，调用该函数将使问题P被初始化，释放P中原本用于储存子句与文字的空间，并将P中记录的子句数、变量数都初始化为0。初始化操作保证了我们在完成一个函数的求解之后，不用退出程序，继续求解下一个可满足性问题，对于数独部分是同样的道理。本函数将在每一次读取文件前被调用，保证读文件前，问题P是已经初始化的。

（3）void \_ReadFile(char filename[200]);//读取文件

函数参数为文件名，无返回值。先调用void init()进行初始化，再读取cnf文件中的各个子句，并将信息储存在刚刚被初始化的问题结构体中，并将注释行完整输出。

（4）void PrintFile();//输出文件内容（仅子句）

调用该函数，函数将打印出问题P中储存的所有子句与子句的具体情况，可以通过调用这个函数来检验文件读入过程是否正确。

如果问题P中没有储存具体信息，输出尚未读取文件。

（5）void DPLL(struct problem\* P);//DPLL模块

调用该函数，程序将开始进行SAT问题的求解与递归调用。首先调用RecoverBegin()将存储数据的链表恢复到初始状态，以便再进行一次求解（在计算优化率时需要对同一个问题多次求解，如果不调用此函数，第二次求解的时第一次已经求出的答案，求解所需时长永远为0ms）。

打印方法选择菜单，选择方法1，调用函数func1；选择方法2，调用函数func2；选择方法3，调用函数func3。求解完毕后该函数将给出可满足性问题的可满足性，递归过程所用时间。此外可以选择是否计算优化率，若选择计算优化率，则调用函数improve（opt）。最后调用函数FormAnsFile将结果生成为同名但以.res结尾的文件。

（6）status func1(int depth, int tar) 求解1

函数的其中一个参数depth指明了当前的递归深度，在DPLL函数中被首次调用时，depth的值将保存为1，之后每深入一层，depth值将加一，该值将在我们的回溯过程中起到至关重要的作用。

另外一个参数tar为当前运行的线索值，该值通常是子句集中保有的单子句对应的文字（带有正负号），调用FindSingleClause()寻找单子句，如果在上一层调用中，子句集中出现了单子句，我们将其保留为线索值，并根据单子句传播规则，将线索值传入下一层调用。线索值的存在将为我们的子句集化简工作带来极大便利，不必每次深入都重新寻找单子句。如果tar值为0，则表示当前子句集中，不含有单子句，则需要调用其它函数来进行下一步工作。

当tar有值时，函数调用函数DeleteTarget反复根据单子句传播规则对子句集进行化简，直至没有单子句。在化简后，我们将进行分裂策略，通过调用函数FindShortestLastLiteral()选取一个变元为真，在选取这个变元后，我们将其作为一个单子句，该变元作为单子句中的文字，加入子句集，进入下一层递归调用。

根据分裂规则，如果我们的子句集中出现了空子句，则代表这一种假设走不通，将调用函数RecoverLastLevel恢复到上一层调用，恢复被屏蔽的子句与文字，并做出反面假设，反应在决策树上是进入另外一个子树。

如此递归，直至我们找出解或所有假设道路均走不通，将求解情况作为返回值返回至DPLL函数，求解答案记录在问题P的结构体中。

变元选取策略详见对应函数status FindShortestFirstLiteral()、加入子句、找寻单子句、删除与屏蔽文字、恢复屏蔽并回到上一层”的各个函数将在下面的内容中逐一介绍。

（7）status func2(int depth, int tar) 求解2

除变元选取策略不同以外，其他都与求解1相同。变元选取策略详见对应函数status FindShortestMaxLiteral()。

（8）status func3(int depth, int tar) 求解3

除变元选取策略不同以外，其他都与求解1相同。变元选取策略详见对应函数status FindFirstLiteral()。

（9）status FindShortestFirstLiteral() 求解1：寻找最短子句中的最后一个元素

该函数会找到未被删除的子句，通过子句的长度找到最短的子句，并找到最短子句的最后一个元素并返回该元素的值。

（10）status FindShortestMaxLiteral() 求解2：寻找最短子句中出现最多的元素

该函数该函数会找到未被删除的子句，通过子句的长度找到最短的子句，并找到最短子句出现次数最多的元素并返回该元素的值。

（11）status FindFirstLiteral() 求解3：寻找所有（未删除）元素中的首个元素

该函数该函数会找到未被删除的子句，返回其中未被删除的首个元素。

（12）int FindSingleClause () 找寻一个单子句

该函数将在每一次递归调用solve函数时首先被调用。每次调用该函数，函数将遍历部分子句集，同时寻找单子句。由于每一个子句的信息结构体中保存了子句中当前文字的数目，我们实际上只需遍历每一个子句的信息结构体，不需要访问每一个文字。当我们找到这样的单子句时，我们将其中保存的文字（带正负号）作为函数的返回值返回，如果没有找到，函数将返回0。

（13）status DeleteTarget(int depth, int single) 删除目标子句和元素

函数的参数depth是线索值，函数将删除子句集中含有depth的所有子句，与含有-depth的文字，执行该操作的同时更新问题P中保留的各个变量出现的次数，方便我们进行下一次的单子句传播操作或者是分裂操作。

首先找到未被删除的子句搜索是否有目标元素，如果找到目标元素且其元素设定为真，则删除该子句；如果找到目标元素且其元素设定为假则删除该元素，若删除后子句为空子句则说明设定值错误，返回ERROR，之后调用的函数将会恢复递归至上一层，并做出反面假设。

（14）void RecoverLastLevel(int depth) 返回上一层递归时的状态

该函数通常是在函数DeleteTarget返回ERROR后紧跟着被调用，意味着这样的假设不能让我们成功求解，但对应的文字或者子句已经被屏蔽，我们也已经在问题结构体P中对子句集进行了修改。我们不得不将造成影响的修改恢复，并做出反面假设。

函数的整型参数depth将作为我们恢复操作的重要依据。在之前，我们从子句集中屏蔽子句或文字时，将标志数据域的值设为了depth，此时派上了用场，当我们恢复至上一层时，在当前depth层做出的修改都必须被复原。遍历整个子句集中的所有文字，当标志数据域的值等于当前递归层数时，意味着此次屏蔽操作在该层被完成，我们将其重设为0，意味着子句或文字恢复到子句集中，同时更新子句总数与各个变元出现的次数，防止出现错误。

（15）void FormAnsFile(int res, int time, char filename[200]); 将答案保存在res文件中

该函数将在求解函数完成求解后，在DPLL函数中被调用。调用该函数，将形成与cnf文件同名但以res结尾的文件。

文件中保留求解结果，求解时间与求解答案。s表示求解结果，1表示满足，0表示不满足，-1表示在限定时间内未完成求解；v后记录求解答案，满足时，每个变元的赋值序列，-1表示第一个变元1取假，2表示第二个变元取真，用空格分开，十个变量占一行，不满足时，将不会记录答案，只显示v；t后保存求解时间，以毫秒为单位，仅计算DPLL执行时间。

（16）void PrintAnswer() 打印求解答案

直接将答案数组中的答案逐个打印，-1表示第一个变元1取假，2表示第二个变元取真，十个变量占据一行。

（17）status CheckAnswer() 检查答案

根据问题P中的记录，将求解答案代入到P的各个子句中，通过判断每个子句的真假性来检查答案是否正确，将检查结果作为返回值返回。

对于不满足的算例，也可以进行检查。不过很显然，对于不满足算例的检查结果，每一次都会是“答案错误！”。

（18）void improve(int opt) 求优化率

该函数传入值为最初使用的方法的消耗时间，首先调用函数RecoverBegin()将存储数据的链表恢复到初始状态，以便再进行一次求解。再使用另一种方法求解并计算优化率。（哪种方法使用时间较短，就当作优化后的算法所消耗的时间）

（19）void RecoverBegin() 返回初始状态

将问题记录恢复成未删除任何元素和子句的刚读入文件时的状态。

蜂窝数独模块的主要函数：

（1）void Sudoku() 蜂窝数独模块

该函数是开始蜂窝数独模块的函数，会打印蜂窝数独菜单，提供可选择的选项并调用相应的函数。

选择求解数独，调用函数void initsudo()进行数独初始化，并初始化蜂窝数独的记录数组，即按照横行为每个位置赋予代号ij，i代表行数，j代表列数，并调用函数void CreateSudoku()生成数独。

选择数独游戏，调用函数void initsudo()进行数独初始化，并初始化蜂窝数独的记录数组，即按照横行为每个位置赋予代号ij，i代表行数，j代表列数，并调用函数void \_CreateSudoku() 生成数独。然后提供玩家三种难度进行选择，根据所选择的难度调用函数DigHole(diff)挖洞，调用函数Printorigin()打印当前题面。

接下来打印蜂窝数独游戏的菜单。选择输入求解调用函数InputToSolve()；选择提示调用函数hint()；选择显示答案调用函数Printfinal()；选择检查调用函数CheckCurrentSolve()。

（2）void initsudo() 数独初始化

调用该函数将使数独sudo被初始化，清除sudo中本来存有的数独格局。初始化操作保证了我们在完成一个数独的求解之后，不用退出程序，继续求解下一个数独，对于数独部分是同样的道理。本函数将在每一次读取数独文件前被调用，保证读文件前，数独sudo是已经初始化的。

（3）void CreateSudoku() 生成数独（求解数独用）

调用该函数，输入蜂窝数独的数据，函数会将数据填入数独数组中，调用函数FormCnfFile转化成cnf问题并生成cnf文件，调用函数\_ReadFile读入文件并用函数func1求解（利用SAT模块求解），最后调用函数AnsTransSudo最将答案保存在答案数组并调用函数Printfinal()将蜂窝数独答案打出。

（4）void \_CreateSudoku() 生成数独（数独游戏用）

调用该函数，随机选取一份蜂窝数独的数据，函数会将数据填入数独数组中，调用函数FormCnfFile转化成cnf问题并生成cnf文件，调用函数\_ReadFile读入文件并用函数func1求解（利用SAT模块求解），最后调用函数AnsTransSudo最将答案保存在答案数组。注意不将答案打出。

（5）void Printorigin() 打印现在的题面

调用该函数，函数将打印数独sudo中存有的目前正在进行的蜂窝数独（数独游戏当前进行的数独的样子）。

（6）void Printfinal() 打印答案题面

调用该函数，函数将打印数独sudo中存有的蜂窝数独答案。

（7）void FormCnfFile(int flag) 将数独数字转换为自然语言编码并存入cnf文件

调用该函数，将形成一个与FileName同名但以cnf为后缀的文件，也就是将数独文件归约为cnf范式，并以文件的形式保存，方便对其求解。

将变元可按语义编码为1～9之间数字构成的三位整数ijk，i, j, k∈{1,2,…,9}，其中i表示单元格的行号，j表示单元格的列号，k表示单元格<i, j>填入的数字为k。负文字则表示不填入。再通过一定的方式将用语义表示的变元切换为连续的变元。

接下来处理横行约束，左斜行约束与右斜行约束。

首先，对于每一个小格而言，填入的数字应当是唯一的，也就是对于某一确定的格子，表明填入任意两个数字的变元不能同时为真

其次，对于每一行中，任意的两格不能填入相同的数字。

对于格子数不同的行约束规则有一定差别。

以横行约束为例。第五行一定有数字1-9，所以1-9约束为必填数字，要求该行必含数字1-9；第四行和第六行必填数字为2-8，则约束该行必含数字2-8，可选填数字组1或9；第三行和第七行必填数字为3-7，则约束该行必含数字3-7，可选填数字组为1-2，2和8或8-9；第二行和第八行必填数字为4-6，则约束该行必含数字4-6，可选填数字组为1、2、3或2、3、8或3、8、9；最后，第一行和第九行，约束该行必含数字5，可选填数字组为1、2、3、4或2、3、4、6或3、4、6、7或4、6、7、8或6、7、8、9。

将数组不同位置的代号更换，使原本左斜行变成横行（将左斜行当作横行重新记录每个位置的代号，相当于将图形旋转），所有的约束条件与横行一模一样，右斜行同理。

将以上条件全部处理成子句的形式，录入一个cnf文件中。

（8）int VarTrans(int i, int j, int k);// 变量转换函数，将语义编码转换为自然顺序编码。

调用该函数，输入i，j，n三个整型变量，意为i行j列填入数字n。函数将通过公式i \* 100 + j \* 10 + k把相对离散的变量转换为相对连续的变量，给DPLL的过程提供方便。

（9）void AnsTransSudo()//将ans数组中保存的答案，转移到sudo.finalans数组中

调用该函数，访问问题结构，将各个变量的取值转换为数独的答案，将答案保存在数独的答案数组中。

（10）void DigHole(int diff) 对已求解的数独进行挖洞

调用该函数，在已有数独格局的基础上对数独进行挖洞。随机地从数独中删除数字，将该数字删除后，调用函数IfCanDig把该位置的其它8个数字代入数独中进行验算，如果有解，则代表删除该数字后数独将不会再是唯一解，该位置的数字不能删除，找寻新的可删除位置。

（11）status IfCanDig(int x, int y) 判断某位置能否被删除

调用该函数，判断(x,y)位置的数字被删除后数独是否仍为唯一解，将可行性表征为函数返回值。首先把该位置的其它8个数字代入数独中进行验算，如果有解，则代表删除该数字后数独将不会再是唯一解，该位置的数字不能删除，找寻新的可删除位置。

（12）void InputToSolve() 数独游戏输入模块

输入x、y、v，x代表行，y代表列，v代表数值，进行交互，若输入位置错误，则输出“请输入合法的位置以及填充的数字！”，若输入题面上已有的数字的位置则输出“您输入的位置是题目所给数字的位置，此位置数字不可更改！”

（13）void hint() 提示

调用该函数，随机选取一个空着的格子显示答案上的数值。

（14）status CheckCurrentSolve() 检查当前输入的解是否有误

调用该函数，检查已输入的数字与答案上是否一致。

**4.2系统测试**

## 4.2.1 SAT模块测试

SAT模块的测试均从主菜单进入SAT部分后开始进行，在预期结果中，括号包含的内容为对显示结果的解释，不是实际的显示内容。

1.文件读入测试

本测试读入的文件为sat-20.cnf。

测试1测试函数是否能成功读入文件并显示文件中的注释内容

测试2测试当文件名输入错误时，能否给出错误提示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 测试输入 | 预期结果 | 运行结果 |
| 1 | 1 sat-20.cnf | （显示文件变量与子句的个数）  已成功读取文件！ | 如下图  符合预期 |
| 2 | 1 123aaa | 无法打开文件。 | 符合预期 |



图4.2.1 文件读入测试结果

2.子句显示功能测试

本测试读入的文件为sat-20.cnf。

测试1测试函数是否能正确显示各个子句；

测试2测试当文件尚未被读入时，程序能否给出判断

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 测试输入 | 预期结果 | 运行结果 |
| 1 | 2 | （显示文件中的各个子句） | 如下图 |
| 2 | 2 | 尚未读取文件！ | 符合预期 |

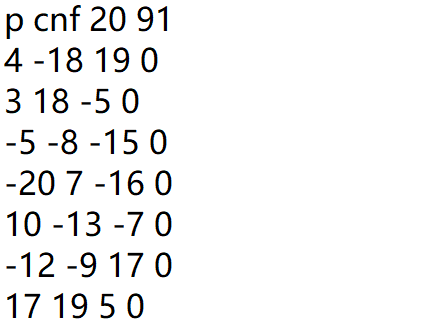
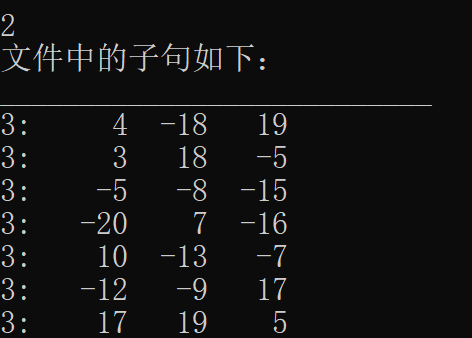


图4.2.2 子句显示测试结果

不难看出，程序输出的各个子句与cnf文件中的子句是相同的，出于节省篇幅的考虑，没有给出全部的子句内容。这证明了子句显示的功能是较为可靠的。

3.DPLL过程测试

测试程序能否求解成功，并显示求解时间。可选择是否计算优化率，显示优化率。

在四个测试中，不退出程序，进行完一个测试后测试下一个算例，测试程序是否具有一定的连贯性。

测试1读入小型满足算例；

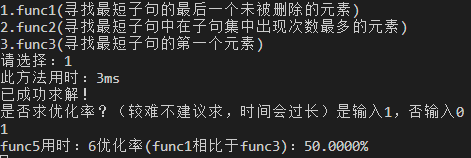


图4.2.3-1 小型算例测试结果

测试2读入中型满足算例；

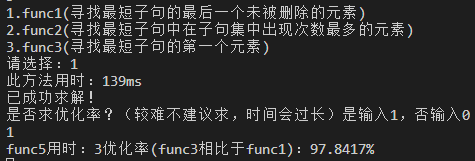


图4.2.3-2 中型算例测试结果

测试3读入大型满足算例；

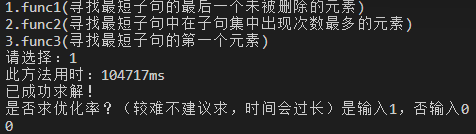


图4.2.3-3 大型算例测试结果

测试4读入不满足算例。

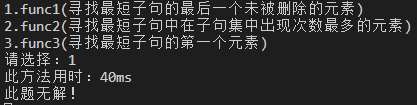


图4.2.3-4 不满足算例测试结果

4.答案打印与文件形成功能测试

此项函数比较简单，测试从简。在求解sat-20.cnf后进行答案的打印，并检查形成的文件，检验是否正确，测试结果如下：



图4.2.4答案打印测试结果

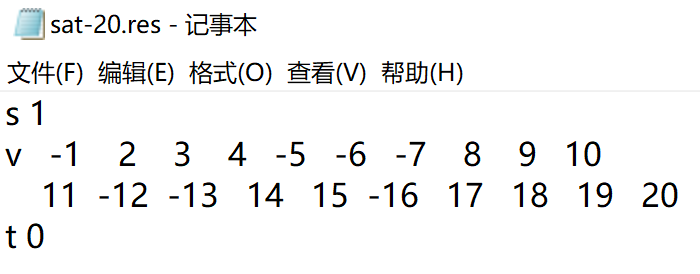


图4.2.5 文件生成测试结果

可以看出，最后打印的答案与生成的文件中的答案是一致的，文件中也包含了求解结果和求解时间等内容。

5.答案检验功能测试

检验答案的过程十分简单，测试中主要检验求解sat-20.cnf的答案与不满足算例的答案是否正确。如果正确则显示“所求解正确！”。很显然，不满足算例的答案总是不正确的。

6.DPLL优化性能测试

测试函数的优化性能，主要是针对变元选取策略进行优化。优化前采用“选取当前子句集中第一个变元”；优化后采用“当前子句集中出现次数最多的一个变元”。对于这两种选取变元的策略，代入不同的算例进行运算，计算求解时间与优化率。

测试结果如下表：（表中文件名后带有×标记的为不满足算例）

表4.2.6 优化性能测试表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 算例名  (以.cnf为后缀) | 变量数/  子句数 | 优化前用时(ms) | 优化后用时(ms) | 优化率 |
| sat-20 | 20/91 | 0 | 0 | 0．0000% |
| problem3-100 | 100/340 | 5 | 4 | 20.0000% |
| problem11-100 | 100/600 | 11 | 4 | 63.6364% |
| problem12-200 | 200/1200 | 60 | 10 | 83.3333% |
| bart17.shuffled-231 | 231/1166 | 140 | 5 | 96.4286% |
| eh-dp04s04.shuffled-1075 | 1075/3152 | 806580 | 108473 | 86.5515% |
| php-010-008.shuffled-as.sat05-1171(×) | 80/370 | 3930 | 3154 | 19.7455% |
| u-5cnf\_3500\_3500\_30f1.shuffled-30(×) | 30/420 | 72 | 33 | 54.1667% |

对于部分大算例而言，优化的结果是显著的，对于中小型算例而言，由于优化过程比较浅显，也能起到一定的优化作用。

## 4.2.2 数独模块测试

数独模块的测试均从主菜单进入数独部分后开始进行，在预期结果中，括号包含的内容为对显示结果的解释，不是实际的显示内容。

1.数独求解功能测试

输入数独文件，测试函数能否正确求解，具体求解所用时间与求解结果如下图所示：

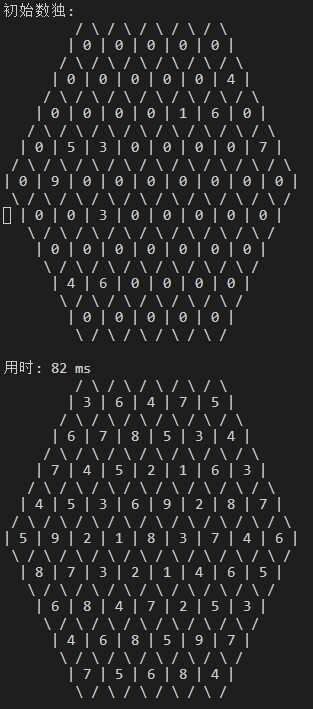
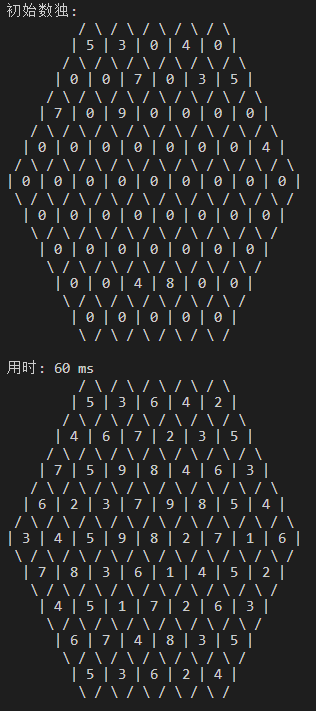
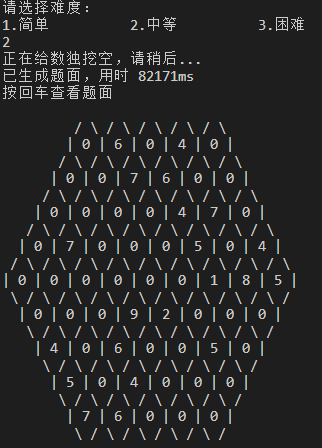
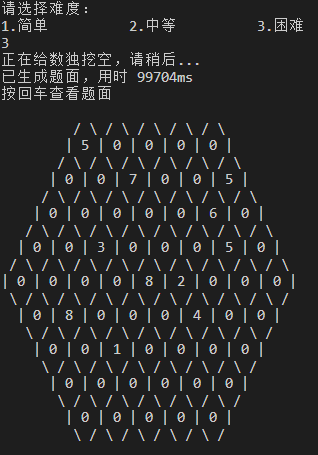
 

图4.2.7 数独求解测试结果

可以看出，数独在解决有10个左右提示字的情况下求解是十分正确的，速度也是十分高效的。

2.数独格局生成与交互

测试程序能否生成不同难度的数独格局，并与用户完成简单的交互，生成的不同难度的格局如下：

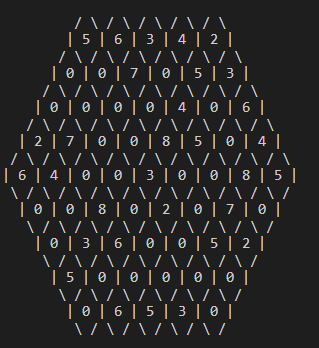
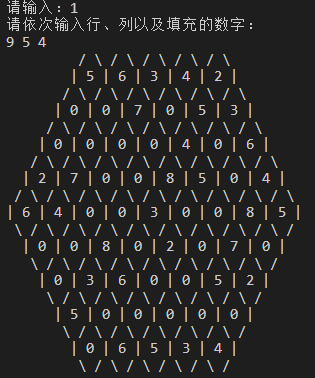
  

（1）简单格局30空 （2）中等格局40空 （3）困难格局50空

图4.2.8 生成的不同难度的数独格局

可以与用户实现简单的交互。选择选项1，用户可以在数独中填入答案：

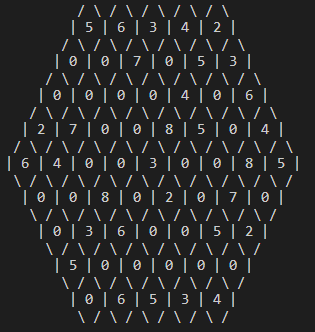
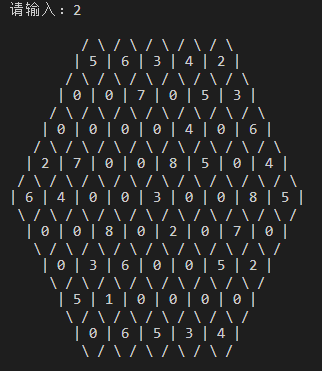
在9 5位置填入4

输入前 输入后

图4.2.9 向蜂窝数独输入数字

选择选项2，将会随机给出一个提示数字

提示前 提示后

图4.2.10 蜂窝数独随机提示数字

选择选项3，直接显示数独答案，并退出回答蜂窝数独菜单界面。



图4.2.11 蜂窝数独显示答案

选择选项4，检查已输入的数值是否正确。都正确则显示“当前求解全部正确”，错误则显示“当前解答有误！”。

输入0则退出，返回到数独菜单界面。

**5总结与展望**

**5.1全文总结**

本次课程设计主要完成了关于SAT问题的求解器的研究，了解并尝试构建了DPLL算法，完成了数独游戏格局的构造与求解。在实验过程中，主要完成了以下几个方面的工作：

（1）了解可满足性问题：

学习了解了可满足性问题的概念与研究意义，复习了离散数学中有关合取范式的相关命题逻辑知识。通过查阅一定的文献资料，了解了可满足性问题研究的发展历程与研究现状。了解到国内外关于可满足性问题的相关研究，特别是我校对于该问题的研究成果。通过查阅其它资料，认识了可满足性问题的研究历史与研究前景。

（2）完成了课程设计的初期设计任务：

了解了cnf文件的基本形式与一般特征，设计了提取文件信息的方法，并设计了数据结构用于储存这些信息，回顾了数据结构课程中的知识，设计了链表的储存结构。同时完成了菜单设计，可以完成简单的交互。最后设计了文件输出与答案检验功能，可以将求解答案按照确定的格式储存在同名文件中，或者检查所求解的答案是否正确。

（3）学习DPLL算法，完成了求解器的设计：

理解了DPLL算法求解SAT问题的过程，了解了DPLL算法处理问题的依据与具体思路，设计了DPLL的求解框架，设计了分裂策略中的变元续期内策略，在算法构建过程中，完成了子句集化简、语句的回溯等任务。并采用了一定的方法对变元选取策略进行优化。

（4）解决蜂窝数独问题并设计简单的蜂窝数独游戏：

理解数独问题与SAT问题的联系，将数独格局按照条件归约成标准的合取范式语句，形成cnf文件，并写入子句集，再调用SAT求解器对数独进行求解。通过挖洞法形成数独格局，并在生成过程中保证形成的数独是唯一解。同样地，完成了数独游戏的文件操作。

设计简单的蜂窝数独游戏，游戏拥有基本的输入功能，提示功能，显示答案功能，检查功能，具有一定的交互性。

**5.2工作展望**

在今后的研究中，围绕着如下几个方面开展工作。

（1）了解和学习更多关于SAT算法的知识，研究更加巧妙地算法，提高问题地求解能力和求解效率，可以在 DPLL 的求解过程中寻求更高效的启发式变量决策策略，以减少算法的回溯次数和搜索空间。

（2）为算法加入合适的CNF文件的预处理器，如引入高级前向推理机制，对 CNF文字和公式进行预处理和化简，以便能更高效地进行对其进行求解。

（3）可以尝试将本文中提到的算法应用到更大规模的实际问题中，比如硬件验证、自动化推理等领域。

（4）算法采用的数据结构和具体实现有待改进，可以引入更巧妙的数据结构来表示文字和子句信息，也可以优化代码实现以增强程序的内聚性。

（5）可以通过对命题逻辑的可满足性问题的逻辑结构进行分析，采用一种基于 解方程组的 SAT 问题求解思路。即先将合取范式转换为等价的方程组形式，然后 再对方程组进行求解，进而得到对应 SAT 问题的解。

# 6 体会

本次课程设计与以前的 C 语言程序设计实验和数据结构实验相比，难度更大，工程量也更多，经过本次实验，我受益匪浅，也有很多体会：

首先，完成一个程序完整的功能，前期的设计和准备是非常必要的，要先根据课程提供的文件理解题意，并在网络上搜索文献了解相关知识，了解DPLL过程和具体实现方式，在前期需要完成很多的准备工作。如果这部分工作缺失或者是不够完善，就容易在已经开始工程后反复地进行确认，或者在完成部分功能后发现设计不完善反复进行更改，这样的习惯无疑会为程序设计带来麻烦和阻力。从这次的学习中，我更加体会到首先要正确地、完整地理解题意，从一开始尽量做好程序基本框架的设计。这次我在开始前没有很好的想清楚程序的框架设计，导致反复地对各种功能函数以及文件、子句的结构进行更改，浪费非常多的的时间与精力。

其次，对于我本人对课程设计的时间规划也是一大挑战，我们在完成工作量较大的工作时要合理地规划时间，这样能产生事半功倍的效果，我们也不会感到压力或者容易产生焦虑情绪。要在各个时间节点完成好相应的任务，保证工作的有效推进，否则整个工程就将是一团乱麻，不知从何下手，也对当前的任务与进度没有一个精确客观的认识。

再者，本次课程设计无疑对我网上查找资料的能力是一大锻炼，我们在任务书中了解到的DPLL算法相关的内容是远远不够的，这样，我们独立查找一些资料与阅读一些文献就显得十分必要了，而优化算法也是必然要通过阅读资料来进行。我认为这次课程设计使得我在这方面的能力得到了很大的提升。

经过这次程序设计，我巩固了C语言与数据结构方面的知识，对于结构体，指针，链表，多文件编译与程序模块化有了更深入的理解，并且更加牢固地掌握了这些课内知识。

我在课程设计中锻炼了自己的编程能力，我们需要自行设计数据结构，自行搭建框架，自行完成操作处理，有许多事情都是自己独立完成的，这方面的能力也无疑得到了锻炼。

我还在实验过程中提升了自己的时间管理能力，学习能力，并且培养了自己心态。精心安排工作时间，查阅文献资料并尝试复刻其中的内容，不论代码是否频繁报错还是无法完成任务，我有有更好的 方法去应对，相信以后可以在应对类似的或者是难度更高的任务更加从容，完成得更好。

**参考文献**

将报告中引用的参考文献放在此处。

[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>

[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

**附录**

## 1 常量与结构体定义definition.h

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<windows.h>

#include<time.h>

#define MAX\_STNUM 20000

#define MAX\_NUM 2000

#define ERROR 0

#define OK 1

#define false 0

#define true 1

typedef int status;

typedef struct element { //保存元素

int flag; //判断是否被删除（0代表未被删除，其余数字代表被删除时的递归层数）

int value; //记录元素(有正负)

struct element\* nextnode; //下一元素变量

}element;

typedef struct clause { //记录子句

int flag; //标记子句真假，mark=0表示子句为真,其他情况表示删除时的递归深度

int literal\_num; //记录子句中元素的个数

struct clause\* nextcla; //指向下一个子句

struct element\* first; //指向第一个元素

}clause;

typedef struct problem { //SAT问题

int vexnum, clanum; //记录布尔变元数量vexnum、子句数量clanum

struct clause\* root; //root结点中不存放数据

int element[MAX\_NUM + 1]; //各元素出现的次数（在无单子句而要选下一个删除关键字时使用）

int ans[MAX\_NUM + 1]; //（有解时的）解

}problem;

typedef struct sudproblem { //数独问题

int original[10][10]; //记录原始数值

int finalans[10][10]; //答案

int flag[10][10]; //防止玩家输错位置

int num; //非0个数

}sud;

## 2 函数声明dfSAT.c

#include"definition.h"

extern char FileName[100] ;//文件名

extern struct problem qq;

extern sud sudo;

clause\* root = NULL;

int arr1[65];

float time1=0,time2=0;

/\*sat问题用到的函数\*/

void init() //初始化问题

{

clause\* cla\_tem = root; //根节点

while (cla\_tem != NULL) {

element\* liter\_tem = cla\_tem->first;//指向第一个元素

while (liter\_tem != NULL) {

element\* rel\_liter = liter\_tem;

liter\_tem = liter\_tem->nextnode;//指向下一个元素

free(rel\_liter);

}

clause\* rel\_cla = cla\_tem; //release\_clause

cla\_tem = cla\_tem->nextcla;

free(rel\_cla);

}

root = NULL;

memset(qq.ans, 0, sizeof(qq.ans)); //初始化数值

memset(qq.element, 0, sizeof(qq.element));

qq.clanum = 0; qq.vexnum = 0;

}

void \_ReadFile(char filename[200]) //读取文件

{

init(); //初始化

FILE\* fp = NULL;

if (!(fp = fopen(filename, "r"))) {

printf("无法打开文件！");

getchar();

}

else {

char c;

char str[5000] = { '\0' };

while ((c = fgetc(fp)) == 'c') {

fgets(str, 5000, fp);

}

fgetc(fp); //读空格

fscanf(fp, "%s%d%d", str, &qq.vexnum, &qq.clanum);

root = (clause\*)malloc(sizeof(clause));

root->literal\_num = 0;

clause\* tem1 = root; //指向子句

element\* tem2\_r = NULL, \* tem2\_l = NULL; //指向元素，采用尾插法插入

int num\_cur;

int cnt = 0; //计数

for (int i = 1; i <= qq.clanum; i++) {

if (i != 1) { //第一次的tem1->nextcla就是root指向的位置，不分配空间

tem1->nextcla = (clause\*)malloc(sizeof(clause));

tem1 = tem1->nextcla;

tem1->literal\_num = 0;

}

tem1->flag = 0;

tem2\_l = NULL; tem2\_r = NULL;

fscanf(fp, "%d", &num\_cur);

while (num\_cur != 0) {

tem1->literal\_num++;

tem2\_r = (element\*)malloc(sizeof(element));//分配空间

tem2\_r->value = num\_cur;

qq.element[abs(num\_cur)]++; //元素个数

tem2\_r->flag = 0;

if (!cnt) {

tem1->first = tem2\_r;

cnt++;

}

else {

tem2\_l->nextnode = tem2\_r;

}

tem2\_l = tem2\_r;

fscanf(fp, "%d", &num\_cur);

}

tem2\_l->nextnode = NULL; //子句的最后元素挂空

cnt = 0;

}

tem1->nextcla = NULL; //最后子句挂空

}

fclose(fp);

}

void PrintFile() {//输出文件内容（仅子句）

if (qq.clanum == 0 && qq.vexnum == 0) {

printf("尚未读取文件！\n");

getchar();

return;

}

printf("该文件所有子句如下：\n");

clause\* tem1 = root;

element\* tem2;

while (tem1 != NULL) {

tem2 = tem1->first;

printf(" %d :",tem1->literal\_num);

while (tem2 != NULL) {

printf("%5d", tem2->value);

tem2 = tem2->nextnode;

}

printf("\n");

tem1 = tem1->nextcla;

}

printf("---------------------------------------------\n");

getchar();

}

void RecoverBegin() {//返回初始状态

memset(qq.element, 0, sizeof(qq.element));//初始化各元素数量记录

qq.clanum = 0;

for (clause\* cla = root; cla; cla = cla->nextcla) {

cla->flag = 0; //恢复子句节点的标志域

cla->literal\_num = 0; //重置子句节点中记录的元素数

for (element\* lit = cla->first; lit; lit = lit->nextnode) {

lit->flag = 0; //恢复各元素节点的标志域

qq.element[abs(lit->value)]++; //恢复各元素出现的次数

cla->literal\_num++; //恢复子句节点中记录的元素数

}

qq.clanum++;

}

}

status FindSingleClause()//寻找一个单子句

{

int sin\_cla = 0, judge = 0;

for (clause\* p1 = root; p1; p1 = p1->nextcla)

{

if (p1->literal\_num == 1 && !p1->flag) //子句中未被删除的元素为1且该子句未被删除

{

for (element\* p2 = p1->first; p2; p2 = p2->nextnode) //找单子句中元素

{

if (!p2->flag) //未被删除的是目标元素

{

sin\_cla = p2->value;

judge = 1;

break;

}

}

if (judge == 1) break;

}

}

return sin\_cla; //没找到单子句时返回0

}

status DeleteTarget(int depth, int single)//删除目标子句和元素

{

for (clause\* cla = root; cla; cla = cla->nextcla)

{

if (cla->flag) continue; //该子句已经删除，判断下一子句

element\* lit = cla->first;

while (lit)

{

if (lit->flag) //该元素已经删除，判断下一元素

{

lit = lit->nextnode;

continue;

}

if (lit->value == single)//找到目标

{

cla->flag = depth; //删除子句

element\* lit2 = cla->first;

while (lit2) { //对应的数量（element数组）减一

if (!lit2->flag) //该子句中在之前的操作中未被删除的才算数

{

qq.element[abs(lit2->value)]--;

lit2 = lit2->nextnode;

continue;

}

lit2 = lit2->nextnode;

}

qq.clanum--; //总子句数减一

break;

}

else if (lit->value == -single) { //仅删除该元素

lit->flag = depth;

qq.element[abs(single)]--;

cla->literal\_num--; //该子句所含元素数减一

if (!cla->literal\_num) return ERROR; //说明删除后出现了空子句

lit = lit->nextnode;

continue;

}

lit = lit->nextnode;

}

}

return OK;

}

status FindShortestLastLiteral() {//寻找最短子句中最后一个元素

int len = MAX\_NUM;

int maxx = 0, res = 0;

for (clause\* cla = root; cla; cla = cla->nextcla) {

if (cla->flag) continue; //子句已删除

if (cla->literal\_num < len) { //寻找最短子句

len = cla->literal\_num;

for (element\* lit = cla->first; lit; lit = lit->nextnode)

{

if (!lit->flag && maxx < qq.element[abs(lit->value)]) //找到最后一个元素

res = lit->value;

}

}

}

return res;

}

void RecoverLastLevel(int depth) //（递归）返回上一层递归时的状态

{

for (clause\* cla = root; cla; cla = cla->nextcla) {

if (cla->flag == depth) { //说明该子句在本层被修改

cla->flag = 0; //恢复子句有效性

qq.clanum++; //恢复总子句数

for (element\* lit = cla->first; lit; lit = lit->nextnode) {//寻找子句中在该层被改动的元素

if (lit->flag == depth) {

lit->flag = 0; //恢复元素有效性

cla->literal\_num++; //恢复子句节点记录的元素数

qq.element[abs(lit->value)]++;//恢复该元素出现的次数

continue;

}

else if (!lit->flag) {

qq.element[abs(lit->value)]++;

}

}

}

else {

element\* lit2 = cla->first; //子句没有被删除，但是元素可能被删除

while (lit2) {

if (lit2->flag == depth) { //有被删除的元素

lit2->flag = 0; //恢复元素有效性

cla->literal\_num++; //该子句元素数+1

qq.element[abs(lit2->value)]++;//该元素数+1

lit2 = lit2->nextnode;

continue;

}

lit2 = lit2->nextnode;

}

}

}

}

status FindShortestFirstLiteral() {//寻找最短子句中的第一个元素

int len = MAX\_NUM;

for (clause\* cla = root; cla; cla = cla->nextcla) {

if (cla->flag) continue;

if (cla->literal\_num < len) {

len = cla->literal\_num;

for (element\* lit = cla->first; lit; lit = lit->nextnode) {

if (!lit->flag) return lit->value;

}

}

}

return 0;

}

status FindShortestMaxLiteral() {//寻找最短子句中出现最多的元素

int len = MAX\_NUM;

int maxx = 0, res = 0;

for (clause\* cla = root; cla; cla = cla->nextcla) {

if (cla->flag) continue; //子句已删除

if (cla->literal\_num <= len) { //寻找最短子句

len = cla->literal\_num;

for (element\* lit = cla->first; lit; lit = lit->nextnode)

{

if (!lit->flag && maxx < qq.element[abs(lit->value)]) {//找频率最高的元素

maxx = qq.element[abs(lit->value)];

res = lit->value;

}

}

}

}

return res;

}

status FindFirstLiteral() {//寻找所有（未删除）元素中的首个元素

clause\* cla = root;

while (cla) {

if (cla->flag) { //子句已删除

cla = cla->nextcla;

continue;

}

else {

element\* lit = cla->first;

while (lit) {

if (lit->flag) {//元素已删除

lit = lit->nextnode;

continue;

}

return lit->value;

}

}

}

return 0;//没有元素，已经找完

}

status func1(int depth, int tar) //递归

{

if (qq.clanum == 0) return OK; //求解完成，剩余子句数为0

int single = 0;

if (!tar) //表示

single = FindSingleClause(); //寻找一个单子句（这里得到的single可正可负）

else

single = tar;

while (single) { //如果能找到单子句

qq.ans[abs(single)] = single; //存储结果

int res = DeleteTarget(depth, single); //删除目标子句与元素

if (res == ERROR) return ERROR; //删除的时候出现了空子句，不满足要求

single = FindSingleClause();

}

int max\_lit = FindShortestLastLiteral(); //寻找最短子句中最后一个元素

if (!max\_lit) return OK; //如果没找到，说明所有元素出现次数都已经变成0，此时说明已经正确求解

if (func1(depth + 1, max\_lit) == OK) { //当已经没有单子句的时候才进行递归 //赋值为真

return OK; //如果递归返回了OK，说明找到了解

}

//上一个if没有return说明没找到解

RecoverLastLevel(depth + 1); //把在下一层做的更改复原

if (func1(depth + 1, -max\_lit) == OK) {//赋值为假

return OK;

}

RecoverLastLevel(depth + 1);

return ERROR; //不能求解

}

status func2(int depth, int tar) { //递归

if (qq.clanum == 0) return OK; //求解完成，剩余子句数为0

int single = 0;

if (!tar) single = FindSingleClause(); //寻找一个单子句（single可正可负）

else single = tar;

while (single) { //如果能找到单子句

qq.ans[abs(single)] = single; //存储结果

int res = DeleteTarget(depth, single); //删除目标子句与元素

if (res == ERROR) return ERROR; //删除的时候出现了空子句，不满足要求

single = FindSingleClause();

}

int max\_lit = FindShortestMaxLiteral(); //寻找最短子句中出现频率最多的元素

if (!max\_lit) return OK; //如果没找到，说明所有元素出现次数都已经变成0，此时说明已经正确求解

if (func2(depth + 1, max\_lit) == OK) { //当已经没有单子句的时候才进行递归

return OK; //如果递归返回了OK，说明找到了解

}

//上一个if没有return说明没找到解

RecoverLastLevel(depth + 1); //把在下一层做的更改复原

if (func2(depth + 1, -max\_lit) == OK) {

return OK;

}

RecoverLastLevel(depth + 1);

return ERROR; //不能求解

}

status func3(int depth, int tar) { //递归

if (qq.clanum == 0) return OK; //求解完成，剩余子句数为0

int single = 0;

if (!tar) single = FindSingleClause(); //寻找一个单子句

else single = tar;

while (single) { //如果能找到单子句

qq.ans[abs(single)] = single; //存储结果

int res = DeleteTarget(depth, single); //删除目标子句与元素

if (res == ERROR) return ERROR; //删除的时候出现了空子句，不满足要求

single = FindSingleClause();

}

int max\_lit = FindFirstLiteral(); //寻找（未被删除的元素中）首个元素

if (!max\_lit) return OK; //如果没找到，说明所有元素出现次数都已经变成0，此时说明已经正确求解

if (func3(depth + 1, max\_lit) == OK) { //当已经没有单子句的时候才进行递归

return OK; //如果递归返回了OK，说明找到了解

}

//上一个if没有return说明没找到解

RecoverLastLevel(depth + 1); //把在下一层做的更改复原

if (func3(depth + 1, -max\_lit) == OK) {

return OK;

}

RecoverLastLevel(depth + 1);

return ERROR; //不能求解

}

void FormAnsFile(int res, int time, char filename[200]) {//将答案保存在res文件中

FILE\* fp;

int len = strlen(filename);

filename[len - 3] = 'r'; //文件后缀名

filename[len - 2] = 'e';

filename[len - 1] = 's';

fp = fopen(filename, "w+");

fprintf(fp, "s %d", res);

fprintf(fp, "\nv");

if (res != 1); //不能求解

else {

for (int i = 1; i <= qq.vexnum; i++) { //写入答案

fprintf(fp, " %d", qq.ans[i]);

}

}

fprintf(fp, "\nt %d", time); //写入求解时间

fclose(fp);

}

void improve(int opt)//求优化率

{

int res;

time\_t t1, t2;

t1 = clock();

RecoverBegin(); //将存储数据的链表恢复到初始状态，以便再进行一次求解

res = func3(1, 0);

t2 = clock();

time2=t2-t1;

printf("func5用时：%d",t2-t1);

if(time1<time2)

printf("优化率(func%d相比于func3)：%.4f%%\n",opt,((time2-time1)/time2)\*100);

else if(time1>time2)

printf("优化率(func3相比于func%d)：%.4f%%\n",opt,((time1-time2)/time1)\*100);

else

printf("优化率：0.0000%%\n");

}

void DPLL() {//DPLL模块

if (qq.clanum == 0 && qq.vexnum == 0)

{

printf("尚未读取文件！\n");

getchar();

return;

}

RecoverBegin(); //将存储数据的链表恢复到初始状态，以便再进行一次求解

printf("1.func1(寻找最短子句的最后一个未被删除的元素)\n2.func2(寻找最短子句中在子句集中出现次数最多的元素)\n3.func3(寻找最短子句的第一个元素)\n");

printf("请选择：");

int opt, res; //res记录求解结果（有解还是无解）

scanf("%d", &opt);

time\_t t1, t2;

if (opt == 1)

{

t1 = clock();

res = func1(1, 0);

}

else if (opt == 2)

{

t1 = clock();

res = func2(1, 0);

}

else if (opt == 3)

{

t1 = clock();

res = func3(1, 0);

}

else {

printf("请输入1~3中的数字！\n");

getchar();

}

t2 = clock();

time1=t2-t1;

printf("此方法用时：%dms\n", t2 - t1);

if (res == OK)

{

printf("已成功求解！\n");

printf("是否求优化率？（较难不建议求，时间会过长）是输入1，否输入0\n");

int x;

scanf("%d",&x);

if(x)

improve(opt);

}

else printf("此题无解！\n");

FormAnsFile(res, t2 - t1, FileName); //将答案保存在res文件中

getchar(); getchar();

}

void PrintAnswer() {//打印答案

printf("答案如下：");

for (int i = 1; i <= qq.vexnum; i++) {

if (!qq.ans[i]) printf("%5d", i); //说明是无关变量，取正值即可

else printf("%5d", qq.ans[i]);

if (i % 10 == 0) ;

}

getchar();

}

void CheckAnswer() {//检查答案

int res = 0, flag = 0;

for (clause\* cla = root; cla; cla = cla->nextcla) {

flag = 0;

for (element\* lit = cla->first; lit; lit = lit->nextnode) {

if (lit->value == qq.ans[abs(lit->value)]) {

//一个子句中存在元素是真的，则子句为真

flag = 1;

break;

}

}

if (!flag) break; //子句中没有一个元素与求得的解相同，错误

}

if (flag) printf("所求解正确！\n");

else printf("所求解错误！\n");

getchar();

}

void SATQuestion() {

int op = 1;

while (op) {

//fflush(stdin);

//system("cls");

printf("------------------------------------------------------------------------\n");

printf(" 1.读入cnf文件 2.输出文件内容 \n");

printf(" 3.DPLL求解 4.输出求解结果 \n"); //选项3同时输出时间性能和保存文件

printf(" 5.检查答案 \n");

printf(" 0.退出 \n");

printf("------------------------------------------------------------------------\n");

scanf("%d",&op);

if (op == 1) {

fflush(stdin);

printf("请输入文件名：");

scanf("%s", FileName);

\_ReadFile(FileName);

printf("该文件有%d个变量，%d个子句\n", qq.vexnum, qq.clanum);

printf("已成功读取文件！\n");

getchar();

}

else if (op == 2) {

fflush(stdin);

PrintFile();

getchar();

}

else if (op == 3) {

fflush(stdin);

DPLL();

getchar();

}

else if (op == 4) {

fflush(stdin);

PrintAnswer();

getchar();

}

else if (op == 5) {

fflush(stdin);

CheckAnswer();

getchar();

}

else if (op != 0) {

fflush(stdin);

printf("输入错误！请重新输入！");

getchar();

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initsudo() {//数独初始化

sudo.num = 0;

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

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

if ((i < 5 && j >(4 + i % 5)) || (i > 5 && j > (9 - i % 5)))

{

sudo.finalans[i][j] = -1;

sudo.original[i][j] = -1;

sudo.flag[i][j] = -1; //蜂窝数独范围外标注-1

continue;

}

sudo.finalans[i][j] = 0;

sudo.original[i][j] = 0;

sudo.flag[i][j] = 0;//数独内初始化为0

}

}

}

status VarTrans(int i, int j, int k)

{//将数独的1-9数字转化成自然顺序编码

return i \* 100 + j \* 10 + k;

}

void FormCnfFile(int flag) {//将数独数字转换为自然语言编码并存入cnf文件

//flag仅仅是用于判断此函数被哪个函数调用了，以确定生成的文件名是什么样的

char cnfname[200] = { '\0' };

if (flag == 1) strcpy(cnfname, "prepare.cnf");

else if (flag == 2) strcpy(cnfname, "dighole.cnf");

FILE\* fp = NULL;

int dis[9][2] = { {0,0},{0,1},{0,2},{1,0},{1,1},{1,2},{2,0},{2,1},{2,2} };

if (!(fp = fopen(cnfname, "w"))) {

printf("无法打开文件\n");

getchar();

}

else

{

fprintf(fp, "p cnf %d %d\n", 959, 7468 + sudo.num);

int k = 1, i = 1, j = 1, count, n, q = 1;

{//将特有规则写入文件（总共sudo.num条语句）

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

{

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

if (sudo.flag[i][j] == 1) fprintf(fp, "%d 0\n", i \* 100 + j \* 10 + sudo.original[i][j]);

}

}

}

//将基础规则写入文件

//只有行规则 分三次写入

//1.

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

{

if (i == 5)//第五行有数字1-9

{

//必填数字

for (k = 1; k <= 9; k++)

{

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

{

fprintf(fp, "%d ", VarTrans(i, j, k));

}

fprintf(fp, "0\n");

}

//格约束--两格不能同时填一个数字

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

{

for (k = 1; k <= 9; k++)

{

for (q = k + 1; q <= 9; q++)

{

fprintf(fp, "%d %d 0\n", -VarTrans(i, j, k), -VarTrans(i, j, q));

}

}

}

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

{

for (k = 1; k <= 9; k++)

{

fprintf(fp, "%d ", VarTrans(i, j, k));

}

fprintf(fp, "0\n");

}

//无重复数字--一行中数字不重复

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 9; q++)

{

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

{

fprintf(fp, "%d %d 0\n", -VarTrans(i, j, k), -VarTrans(i, q, k));

}

}

}

}

if (i == 4 || i == 6)//第四行和第六号

{

//必填数字未2-8

for (k = 2; k <= 8; k++)

{

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", VarTrans(i, j, k));

}

fprintf(fp, "0\n");

}

//约束选填--1或9

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", VarTrans(i, j, 1));

}

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", VarTrans(i, j, 9));

}

fprintf(fp, "0\n");

//格约束--两格不能同时填一个数字

for (j = 1; j <= 8; j++)

{

for (k = 1; k <= 9; k++)

{

for (q = k + 1; q <= 9; q++)

{

fprintf(fp, "%d %d 0\n", -VarTrans(i, j, k), -VarTrans(i, j, q));

}

}

}

for (j = 1; j <= 8; j++)

{

for (k = 1; k <= 9; k++)

{

fprintf(fp, "%d ", VarTrans(i, j, k));

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 8; q++)

{

for (j = q + 1; j <= 8; j++)

{

fprintf(fp, "%d %d 0\n", -VarTrans(i, j, k), -VarTrans(i, q, k));

}

}

}

}

if (i == 3 || i == 7)///第三行和第七行

{

//必填数字3-7

for (k = 3; k <= 7; k++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束12或28或89

for (k = 1; k <= 2; k++)

{

for (q = 8; q <= 7 + k; q++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + q);

}

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//格约束--两格不能同时填一个数字

for (j = 1; j <= 7; j++)

{

for (k = 1; k <= 9; k++)

{

for (q = k + 1; q <= 9; q++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + j \* 10 + q));

}

}

}

for (j = 1; j <= 7; j++)

{

for (k = 1; k <= 9; k++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 7; q++)

{

for (j = q + 1; j <= 7; j++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + q \* 10 + k));

}

}

}

}

if (i == 2 || i == 8)//第二行和第八行

{

//必填数字4-6

for (k = 4; k <= 6; k++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束123或237或378或789

for (k = 1; k <= 3; k++)

{

for (q = 7; q <= 6 + k; q++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + q);

}

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//格约束--两格不能同时填一个数字

for (j = 1; j <= 6; j++)

{

for (k = 1; k <= 9; k++)

{

for (q = k + 1; q <= 9; q++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + j \* 10 + q));

}

}

}

for (j = 1; j <= 6; j++)

{

for (k = 1; k <= 9; k++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 6; q++)

{

for (j = q + 1; j <= 6; j++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + q \* 10 + k));

}

}

}

}

if (i == 1 || i == 9)//第一行和第九行

{

//必填数字5

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + 5);

}

fprintf(fp, "0\n");

//选填约束1234或2346或3467或4678或6789

for (k = 1; k <= 4; k++)

{

for (q = 6; q <= 5 + k; q++)

{

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + q);

}

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//格约束--两格不能同时填一个数字

for (j = 1; j <= 5; j++)

{

for (k = 1; k <= 9; k++)

{

for (q = k + 1; q <= 9; q++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + j \* 10 + q));

}

}

}

for (j = 1; j <= 5; j++)

{

for (k = 1; k <= 9; k++)

{

fprintf(fp, "%d ", i \* 100 + j \* 10 + k);

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 5; q++)

{

for (j = q + 1; j <= 5; j++)

{

fprintf(fp, "%d %d 0\n", -(i \* 100 + j \* 10 + k), -(i \* 100 + q \* 10 + k));

}

}

}

}

}

//2.//更换每个位置的代号数字，相当于斜着看数独，这样后面的代码几乎完全一样

int arr[10][10];

memset(arr, 0, sizeof(arr));

arr[1][1] = 15; arr[1][2] = 26; arr[1][3] = 37; arr[1][4] = 48; arr[1][5] = 59;

arr[2][1] = 14; arr[2][2] = 25; arr[2][3] = 36; arr[2][4] = 47; arr[2][5] = 58; arr[2][6] = 68;

arr[3][1] = 13; arr[3][2] = 24; arr[3][3] = 35; arr[3][4] = 46; arr[3][5] = 57; arr[3][6] = 67; arr[3][7] = 77;

arr[4][1] = 12; arr[4][2] = 23; arr[4][3] = 34; arr[4][4] = 45; arr[4][5] = 56; arr[4][6] = 66; arr[4][7] = 76; arr[4][8] = 86;

arr[5][1] = 11; arr[5][2] = 22; arr[5][3] = 33; arr[5][4] = 44; arr[5][5] = 55; arr[5][6] = 65; arr[5][7] = 75; arr[5][8] = 85; arr[5][9] = 95;

arr[6][1] = 21; arr[6][2] = 32; arr[6][3] = 43; arr[6][4] = 54; arr[6][5] = 64; arr[6][6] = 74; arr[6][7] = 84; arr[6][8] = 94;

arr[7][1] = 31; arr[7][2] = 42; arr[7][3] = 53; arr[7][4] = 63; arr[7][5] = 73; arr[7][6] = 83; arr[7][7] = 93;

arr[8][1] = 41; arr[8][2] = 52; arr[8][3] = 62; arr[8][4] = 72; arr[8][5] = 82; arr[8][6] = 92;

arr[9][1] = 51; arr[9][2] = 61; arr[9][3] = 71; arr[9][4] = 81; arr[9][5] = 91;

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

{

if (i == 1 || i == 9)

{

//必填数字

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 5);

}

fprintf(fp, "0\n");

//选填约束

for (k = 1; k <= 4; k++)

{

for (q = 6; q <= 5 + k; q++)

{

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 5; q++)

{

for (j = q + 1; j <= 5; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 2 || i == 8)

{

//必填数字

for (k = 4; k <= 6; k++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束

for (k = 1; k <= 3; k++)

{

for (q = 7; q <= 6 + k; q++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 6; q++)

{

for (j = q + 1; j <= 6; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 3 || i == 7)

{

//必填数字

for (k = 3; k <= 7; k++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束

for (k = 1; k <= 2; k++)

{

for (q = 8; q <= 7 + k; q++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 7; q++)

{

for (j = q + 1; j <= 7; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 4 || i == 6)

{

//必填数字

for (k = 2; k <= 8; k++)

{

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//约束选填

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 1);

}

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 9);

}

fprintf(fp, "0\n");

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 8; q++)

{

for (j = q + 1; j <= 8; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 5)

{

//必填数字

for (k = 1; k <= 9; k++)

{

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

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 9; q++)

{

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

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

}

//3.//更换每个位置的代号数字，相当于斜着看数独，这样后面的代码几乎完全一样

memset(arr, 0, sizeof(arr));

arr[1][1] = 59; arr[1][2] = 68; arr[1][3] = 77; arr[1][4] = 86; arr[1][5] = 95;

arr[2][1] = 48; arr[2][2] = 58; arr[2][3] = 67; arr[2][4] = 76; arr[2][5] = 85; arr[2][6] = 94;

arr[3][1] = 37; arr[3][2] = 47; arr[3][3] = 57; arr[3][4] = 66; arr[3][5] = 75; arr[3][6] = 84; arr[3][7] = 93;

arr[4][1] = 26; arr[4][2] = 36; arr[4][3] = 46; arr[4][4] = 56; arr[4][5] = 65; arr[4][6] = 74; arr[4][7] = 83; arr[4][8] = 92;

arr[5][1] = 15; arr[5][2] = 25; arr[5][3] = 35; arr[5][4] = 45; arr[5][5] = 55; arr[5][6] = 64; arr[5][7] = 73; arr[5][8] = 82; arr[5][9] = 91;

arr[6][1] = 14; arr[6][2] = 24; arr[6][3] = 34; arr[6][4] = 44; arr[6][5] = 54; arr[6][6] = 63; arr[6][7] = 72; arr[6][8] = 81;

arr[7][1] = 13; arr[7][2] = 23; arr[7][3] = 33; arr[7][4] = 43; arr[7][5] = 53; arr[7][6] = 62; arr[7][7] = 71;

arr[8][1] = 12; arr[8][2] = 22; arr[8][3] = 32; arr[8][4] = 42; arr[8][5] = 52; arr[8][6] = 61;

arr[9][1] = 11; arr[9][2] = 21; arr[9][3] = 31; arr[9][4] = 41; arr[9][5] = 51;

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

{

if (i == 1 || i == 9)

{

//必填数字

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 5);

}

fprintf(fp, "0\n");

//选填约束

for (k = 1; k <= 4; k++)

{

for (q = 6; q <= 5 + k; q++)

{

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 5; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 5; q++)

{

for (j = q + 1; j <= 5; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 2 || i == 8)

{

//必填数字

for (k = 4; k <= 6; k++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束

for (k = 1; k <= 3; k++)

{

for (q = 7; q <= 6 + k; q++)

{

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 6; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 6; q++)

{

for (j = q + 1; j <= 6; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 3 || i == 7)

{

//必填数字

for (k = 3; k <= 7; k++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//选填约束

for (k = 1; k <= 2; k++)

{

for (q = 8; q <= 7 + k; q++)

{

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + q);

}

for (j = 1; j <= 7; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 7; q++)

{

for (j = q + 1; j <= 7; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 4 || i == 6)

{

//必填数字

for (k = 2; k <= 8; k++)

{

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//约束选填

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 1);

}

for (j = 1; j <= 8; j++)

{

fprintf(fp, "%d ", arr[i][j] \* 10 + 9);

}

fprintf(fp, "0\n");

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 8; q++)

{

for (j = q + 1; j <= 8; j++)

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

if (i == 5)

{

//必填数字

for (k = 1; k <= 9; k++)

{

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

{

fprintf(fp, "%d ", arr[i][j] \* 10 + k);

}

fprintf(fp, "0\n");

}

//无重复数字

for (k = 1; k <= 9; k++)

{

for (q = 1; q <= 9; q++)

{

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

{

fprintf(fp, "%d %d 0\n", -(arr[i][j] \* 10 + k), -(arr[i][q] \* 10 + k));

}

}

}

}

}

}

fclose(fp);

return;

}

void Printorigin()//打印现在的题面

{

printf(" / \\ / \\ / \\ / \\ / \\ \n");

printf(" | %d | %d | %d | %d | %d | \n",sudo.original[1][1],sudo.original[1][2],sudo.original[1][3],sudo.original[1][4],sudo.original[1][5]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ \n");

printf(" | %d | %d | %d | %d | %d | %d | \n",sudo.original[2][1],sudo.original[2][2],sudo.original[2][3],sudo.original[2][4],sudo.original[2][5],sudo.original[2][6]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | \n",sudo.original[3][1],sudo.original[3][2],sudo.original[3][3],sudo.original[3][4],sudo.original[3][5],sudo.original[3][6],sudo.original[3][7]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | %d | \n",sudo.original[4][1],sudo.original[4][2],sudo.original[4][3],sudo.original[4][4],sudo.original[4][5],sudo.original[4][6],sudo.original[4][7],sudo.original[4][8]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ \n");

printf("| %d | %d | %d | %d | %d | %d | %d | %d | %d |\n",sudo.original[5][1],sudo.original[5][2],sudo.original[5][3],sudo.original[5][4],sudo.original[5][5],sudo.original[5][6],sudo.original[5][7],sudo.original[5][8],sudo.original[5][9]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ /\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | %d | \n",sudo.original[6][1],sudo.original[6][2],sudo.original[6][3],sudo.original[6][4],sudo.original[6][5],sudo.original[6][6],sudo.original[6][7],sudo.original[6][8]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | %d | %d | \n",sudo.original[7][1],sudo.original[7][2],sudo.original[7][3],sudo.original[7][4],sudo.original[7][5],sudo.original[7][6],sudo.original[7][7]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | %d | \n",sudo.original[8][1],sudo.original[8][2],sudo.original[8][3],sudo.original[8][4],sudo.original[8][5],sudo.original[8][6]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | \n",sudo.original[9][1],sudo.original[9][2],sudo.original[9][3],sudo.original[9][4],sudo.original[9][5]);

printf(" \\ / \\ / \\ / \\ / \\ / \n");

putchar('\n');

}

void Printfinal()//打印答案题面

{

printf(" / \\ / \\ / \\ / \\ / \\ \n");

printf(" | %d | %d | %d | %d | %d | \n",sudo.finalans[1][1],sudo.finalans[1][2],sudo.finalans[1][3],sudo.finalans[1][4],sudo.finalans[1][5]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ \n");

printf(" | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[2][1],sudo.finalans[2][2],sudo.finalans[2][3],sudo.finalans[2][4],sudo.finalans[2][5],sudo.finalans[2][6]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[3][1],sudo.finalans[3][2],sudo.finalans[3][3],sudo.finalans[3][4],sudo.finalans[3][5],sudo.finalans[3][6],sudo.finalans[3][7]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[4][1],sudo.finalans[4][2],sudo.finalans[4][3],sudo.finalans[4][4],sudo.finalans[4][5],sudo.finalans[4][6],sudo.finalans[4][7],sudo.finalans[4][8]);

printf(" / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ \n");

printf("| %d | %d | %d | %d | %d | %d | %d | %d | %d |\n",sudo.finalans[5][1],sudo.finalans[5][2],sudo.finalans[5][3],sudo.finalans[5][4],sudo.finalans[5][5],sudo.finalans[5][6],sudo.finalans[5][7],sudo.finalans[5][8],sudo.finalans[5][9]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ /\n");

printf(" | %d | %d | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[6][1],sudo.finalans[6][2],sudo.finalans[6][3],sudo.finalans[6][4],sudo.finalans[6][5],sudo.finalans[6][6],sudo.finalans[6][7],sudo.finalans[6][8]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[7][1],sudo.finalans[7][2],sudo.finalans[7][3],sudo.finalans[7][4],sudo.finalans[7][5],sudo.finalans[7][6],sudo.finalans[7][7]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | %d | \n",sudo.finalans[8][1],sudo.finalans[8][2],sudo.finalans[8][3],sudo.finalans[8][4],sudo.finalans[8][5],sudo.finalans[8][6]);

printf(" \\ / \\ / \\ / \\ / \\ / \\ / \n");

printf(" | %d | %d | %d | %d | %d | \n",sudo.finalans[9][1],sudo.finalans[9][2],sudo.finalans[9][3],sudo.finalans[9][4],sudo.finalans[9][5]);

printf(" \\ / \\ / \\ / \\ / \\ / \n");

putchar('\n');

}

void AnsTransSudo()//将ans数组中保存的答案，转移到sudo.finalans数组中

{

int k, i, j, n;

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

if (i < 5)

{

for (j = 1; j <= (4 + i % 5); j++)

{

for (k = 1; k <= 9; k++)

{

if (qq.ans[i \* 100 + j \* 10 + k] > 0)

{

sudo.finalans[i][j] = k;

}

}

}

}

if (i == 5)

{

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

{

for (k = 1; k <= 9; k++)

{

if (qq.ans[i \* 100 + j \* 10 + k] > 0)

{

sudo.finalans[i][j] = k;

}

}

}

}

if (i > 5)

{

for (j = 1; j <= (9 - i % 5); j++)

{

for (k = 1; k <= 9; k++)

{

if (qq.ans[i \* 100 + j \* 10 + k] > 0)

{

sudo.finalans[i][j] = k;

}

}

}

}

}

}

void \_CreateSudoku() //生成数独用

{

srand((unsigned)time(NULL));

int x = rand() % 6 ;

int i;

int a[7][62]={{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,4,0,4,0,0,0,0,5,1,7,3,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0},

{5,3,0,4,0,0,0,7,0,3,5,7,0,9,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,8,0,0,0,0,0,0,0},

{0,2,0,6,4,6,0,0,7,0,0,0,7,9,0,0,0,0,3,0,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0},

{3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,5,2,0,0,6,0,0,0,0,0,0,0,0,0,4,7,6,0,0,0,0,0,5,0,0,0,2,0,8,0,0,0,0,0,0,7,0,0,0,0,0},

{0,0,0,4,0,0,0,7,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,2,5,0,4,2,0,0,7,0,5,3,0},

{0,0,0,0,0,0,1,5,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,3,0,6,4,3,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,7,0,4,5,0,0,0,0,0,0,0,0,0,2,4,3,4,0,0,0,0,0,2,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,6,0,0,0,0,0,0}};

sudo.num = 61;

for (i = 1; i <= 61;i++ )

{

sudo.flag[arr1[i] / 10][arr1[i] % 10] = 1;

sudo.original[arr1[i] / 10][arr1[i] % 10] = a[x][i-1];

}

FormCnfFile(1); //将初始数独盘导入cnf文件，便于后续求解已得到完整数独题面

char cnfname[200] = "prepare.cnf";

\_ReadFile(cnfname); //读取刚刚生成的cnf文件，准备DPLL求解以生成数独题面

func1(1, 0);

AnsTransSudo(); //将ans数组中保存的答案，转移到sudo.finalans数组中

sudo.num = 61;

return;

}

void CreateSudoku() {//求解数独用

int i = 1;

char temp;

sudo.num = 0;

printf("输入:");

for (i = 1; i <= 61; )

{

temp = getchar();

if (temp >= '0' && temp <= '9')

{

if (temp != '0')

{

sudo.num++;

sudo.flag[arr1[i] / 10][arr1[i] % 10] = 1;

sudo.original[arr1[i] / 10][arr1[i] % 10] = temp - '0';

}

i++;

}

}

printf("初始数独:\n");

Printorigin();

getchar();

FormCnfFile(1); //将初始数独盘导入cnf文件，便于后续求解已得到完整数独题面

char cnfname[200] = "prepare.cnf";

\_ReadFile(cnfname); //读取刚刚生成的cnf文件，准备DPLL求解以生成数独题面

time\_t t1, t2;

t1 = clock();

func1(1, 0);

t2 = clock();

printf("用时: %d ms\n", t2 - t1);

AnsTransSudo(); //将ans数组中保存的答案，转移到sudo.finalans数组中

Printfinal();

sudo.num = 61;

return;

}

status IfCanDig(int x, int y)

{//判断当前位置可否挖洞

if(sudo.original[x][y] == 0 || sudo.finalans[x][y] == -1)

return ERROR;

else{

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

{

if(i == sudo.original[x][y])

continue;

else

{

sudo.original[x][y] = i;

FormCnfFile(2);

char cnfname[200] = "dighole.cnf";

\_ReadFile(cnfname);

int res = func1(1,0);

if(res == OK){ //说明解不唯一，则该点不能删除

sudo.original[x][y] = sudo.finalans[x][y]; //恢复

return ERROR;

}

else{

sudo.original[x][y] = 0; //可以挖洞

sudo.num--;

return OK;

}

}

}

}

}

void DigHole(int diff)

{ //挖洞法创造数独题面

int count\_del = 0; //记录已经删除（挖洞）的个数

int target = 0; //目标挖洞数

if(diff == 1) target = 30;

else if(diff == 2) target = 40;

else if(diff == 3) target = 50;

while(1){

if(count\_del >= target) break;

int x = rand() % 9 + 1;

int y = rand() % 9 + 1;

if((x <= 5 && y >x+4) || (x > 5 && y >14-x ))

continue;

if(IfCanDig(x,y) == OK){

sudo.flag[x][y] = 1; //在输入求解时用到，flag为1说明可以输入数字，flag为0说明是题面初始数字，不能更改

count\_del++;

}

}

}

void InputToSolve(){//输入求解模块

printf("请依次输入行、列以及填充的数字：\n");

int x, y, v;

scanf("%d %d %d",&x,&y,&v);

if((x < 1 || x > 9) ||(x<=5&&y>x+4 )||(x>5&&y>14-x)|| (v < 1 || v > 9)){

printf("请输入合法的位置以及填充的数字！\n");

getchar();

return;

}

else if(sudo.flag[x][y] == 0){

printf("您输入的位置是题目所给数字的位置，此位置数字不可更改！\n");

getchar();

return;

}

sudo.original[x][y] = v;

sudo.num++;

}

void hint(){//提示

while(1){

int x = rand() % 9 + 1;

int y = rand() % 9 + 1;

if (!sudo.original[x][y])//未填入的位置

{ if((x <= 5 && y >x+4) || (x > 5 && y >14-x ))

continue;

sudo.original[x][y] = sudo.finalans[x][y];

sudo.num++;

break;

}

}

}

status CheckCurrentSolve(){//检查当前输入的解是否有误

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

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

if(sudo.finalans[i][j] == -1) continue;

if(sudo.original[i][j] != sudo.finalans[i][j] && sudo.original[i][j] != 0) return ERROR;

}

}

return OK;

}

void Sudoku() {//数独模块

int op = 1;

while (op) {

//fflush(stdin);

//system("cls");

printf("------------------------------------------------------------------------\n");

printf(" 1.求解数独 sudoku solve 2.生成数独 sudoku game 0.退出 back \n");

printf("------------------------------------------------------------------------\n");

scanf("%d",&op);

if (op == 1)

{

//fflush(stdin);

initsudo();//初始化棋盘

//system("cls");

memset(arr1, 0, sizeof(arr1));//初始化每个位置的序号，后续只用更换序号就可以将斜行转化

arr1[1] = 11; arr1[2] = 12; arr1[3] = 13; arr1[4] = 14; arr1[5] = 15;

arr1[6] = 21; arr1[7] = 22; arr1[8] = 23; arr1[9] = 24; arr1[10] = 25; arr1[11] = 26;

arr1[12] = 31; arr1[13] = 32; arr1[14] = 33; arr1[15] = 34; arr1[16] = 35; arr1[17] = 36; arr1[18] = 37;

arr1[19] = 41; arr1[20] = 42; arr1[21] = 43; arr1[22] = 44; arr1[23] = 45; arr1[24] = 46; arr1[25] = 47; arr1[26] = 48;

arr1[27] = 51; arr1[28] = 52; arr1[29] = 53; arr1[30] = 54; arr1[31] = 55; arr1[32] = 56; arr1[33] = 57; arr1[34] = 58; arr1[35] = 59;

arr1[36] = 61; arr1[37] = 62; arr1[38] = 63; arr1[39] = 64; arr1[40] = 65; arr1[41] = 66; arr1[42] = 67; arr1[43] = 68;

arr1[44] = 71; arr1[45] = 72; arr1[46] = 73; arr1[47] = 74; arr1[48] = 75; arr1[49] = 76; arr1[50] = 77;

arr1[51] = 81; arr1[52] = 82; arr1[53] = 83; arr1[54] = 84; arr1[55] = 85; arr1[56] = 86;

arr1[57] = 91; arr1[58] = 92; arr1[59] = 93; arr1[60] = 94; arr1[61] = 95;

CreateSudoku();//创造数独

getchar(); getchar();

}

else if(op==2)

{

initsudo();//初始化棋盘

//system("cls");

memset(arr1, 0, sizeof(arr1));

arr1[1] = 11; arr1[2] = 12; arr1[3] = 13; arr1[4] = 14; arr1[5] = 15;

arr1[6] = 21; arr1[7] = 22; arr1[8] = 23; arr1[9] = 24; arr1[10] = 25; arr1[11] = 26;

arr1[12] = 31; arr1[13] = 32; arr1[14] = 33; arr1[15] = 34; arr1[16] = 35; arr1[17] = 36; arr1[18] = 37;

arr1[19] = 41; arr1[20] = 42; arr1[21] = 43; arr1[22] = 44; arr1[23] = 45; arr1[24] = 46; arr1[25] = 47; arr1[26] = 48;

arr1[27] = 51; arr1[28] = 52; arr1[29] = 53; arr1[30] = 54; arr1[31] = 55; arr1[32] = 56; arr1[33] = 57; arr1[34] = 58; arr1[35] = 59;

arr1[36] = 61; arr1[37] = 62; arr1[38] = 63; arr1[39] = 64; arr1[40] = 65; arr1[41] = 66; arr1[42] = 67; arr1[43] = 68;

arr1[44] = 71; arr1[45] = 72; arr1[46] = 73; arr1[47] = 74; arr1[48] = 75; arr1[49] = 76; arr1[50] = 77;

arr1[51] = 81; arr1[52] = 82; arr1[53] = 83; arr1[54] = 84; arr1[55] = 85; arr1[56] = 86;

arr1[57] = 91; arr1[58] = 92; arr1[59] = 93; arr1[60] = 94; arr1[61] = 95;

printf("正在生成数独，请稍后...\n");

\_CreateSudoku();//创造数独

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

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

if ((i < 5 && j >(4 + i % 5)) || (i > 5 && j > (9 - i % 5)))

{

continue;

}

sudo.original[i][j] = sudo.finalans[i][j] ;

}

}

printf("请选择难度：\n");

printf("1.简单 2.中等 3.困难\n");

int diff;

scanf("%d",&diff);

printf("正在给数独挖空，请稍后...\n");

time\_t t3, t4;

t3 = clock();

DigHole(diff);//（根据不同难度）挖洞法生成题面

t4 = clock();

printf("已生成题面，用时 %dms\n",t4-t3);//挖洞用时

printf("按回车查看题面\n");

getchar();getchar();

int opp = 1;

while(opp){

//system("cls");

Printorigin(); //打印当前题面

if(sudo.num == 61){//双数独共153个数字

if(CheckCurrentSolve())

{

printf("当前求解全部正确！\n");

printf("你已完成该数独的求解！\n");

break;

}

else

{ printf("当前解答有误！\n");

continue;}

getchar();

}

printf("1.输入求解 2.提示 3.显示答案 4.检查 0.退出\n");

printf("请输入：");

scanf("%d",&opp);

if(opp == 1){

fflush(stdin);

InputToSolve(); //输入求解

}

else if(opp == 2){

fflush(stdin);

hint(); //提示

getchar();

}

else if(opp == 3){

//fflush(stdin);

//system("cls");

Printfinal(); //打印最终答案

printf("按任意键退出");

getchar();

break;

}

else if(opp == 4){ //检查当前解答是否有误

fflush(stdin);

if(CheckCurrentSolve()) printf("当前求解全部正确！\n");

else printf("当前解答有误！\n");

getchar();

}

else if(opp != 0){

fflush(stdin);

printf("输入错误！请重新输入！");

getchar();

}

}

}

}

}

## 3 主函数dfmain.cpp

#include"dfSAT.c"

struct problem qq; //记录问题

sud sudo;

char FileName[100] = { '\0' };

int main(void)

{

int choice =1;

while (choice) {

printf("---------------------------------------------------\n");

printf(" 1.SAT SAT问题 2.Sudoku 蜂窝数独 0.Exit 退出 \n");

printf("---------------------------------------------------\n");

scanf("%d",&choice);

if (choice == 1) { //SAT求解

fflush(stdin);

SATQuestion(); //进入SAT模块

}

else if (choice == 2) { //数独问题

fflush(stdin);

Sudoku();

}

else if (choice != 0) {

fflush(stdin);

printf("输入错误！请重新输入1或2或0！");

getchar(); getchar();

}

}

free(root);

return 0;

}