華中科技大學

课 程 设 计 报 告

题目： 程序设计综合课程设计

课程名称	**程序设计综合课程设计**
专业班级	**计算机科学与技术 2107 班**
学　　号	**U202115538**
姓　　名	**陈侠锟**
指导教师	**周全**
报告日期	**2022 年 9 月 7 日**

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，具有重要理论意义于应用价值。本设计要求基于 DPLL 算法实现一个 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；精心设计问题中的变元、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：将双数独游戏 [5] 问题转化为 SAT 问题 [6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献 [3] 与 [6-8]。(15%)

目　　录

1 引言

1.1 课题背景意义

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。但目前不存在一种求解算法在最坏情况下的时间复杂度是多项式级别，其求解速度仍是制约 SAT 算法发展的一大难题。因此，世界各国的学者都在努力研究新的求解算法，以寻求一种高效的求解算法。从 1960 年至今，SAT 问题一直备受人们的关注，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。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。

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，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 求解器用于数独的求解中。

6. 实现数独游戏的求解与可玩性的交互。

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，分析问题中所涉及的数据对象，划分功能，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

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，基于 DPLL 过程实现一个高效 SAT 求解器，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，并进一步将双数独游戏问题转化为 SAT 问题，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。

SAT 求解器首先需要进行 cnf 文件的读入，将子句与文字存入合适的结构体内，之后，我们需要基于 DPLL 算法框架对 SAT 算例进行求解，并能够输出答案与自我检查答案是否正确，最后将求解结果保存到 res 文件中。此外，DPLL 过程运行时间的优化也是我们算关心的重要内容。

对于数独游戏，我们需要将双数独格局转换为 SAT 问题，以便通过 DPLL 算法来生成双数独格局。同时，游戏应具备一定的交互性，可让用户选择游戏的难度，进行输入求解，游戏也应当具备提示、检查等功能以完善设计。

2.2 系统总体设计

系统分为两个模块，分别是 SAT 问题求解模块和双数独游戏模块，其中 SAT 问题求解模块又可分为 CNF 解析模块和核心 DPLL 模块。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，并存储到数据结构中（CNF 解析模块）；
2. 遍历所有子句并输出（CNF 解析模块）；
3. 利用 DPLL 算法求解 SAT 问题（DPLL 模块）；
4. 将 DPLL 算法求得的解写入文件（CNF 解析模块）；
5. 打印 DPLL 求解得到的答案（CNF 解析模块）；
6. 验证 DPLL 求解的答案是否正确（CNF 解析模块）。

双数独游戏模块包括：

1. 数独完整棋盘的生成；
2. 基于挖洞法生成数独游戏的题面；

3. 与用户的交互，包括“输入求解”“提示”“检查已填内容”“显示答案”功能。

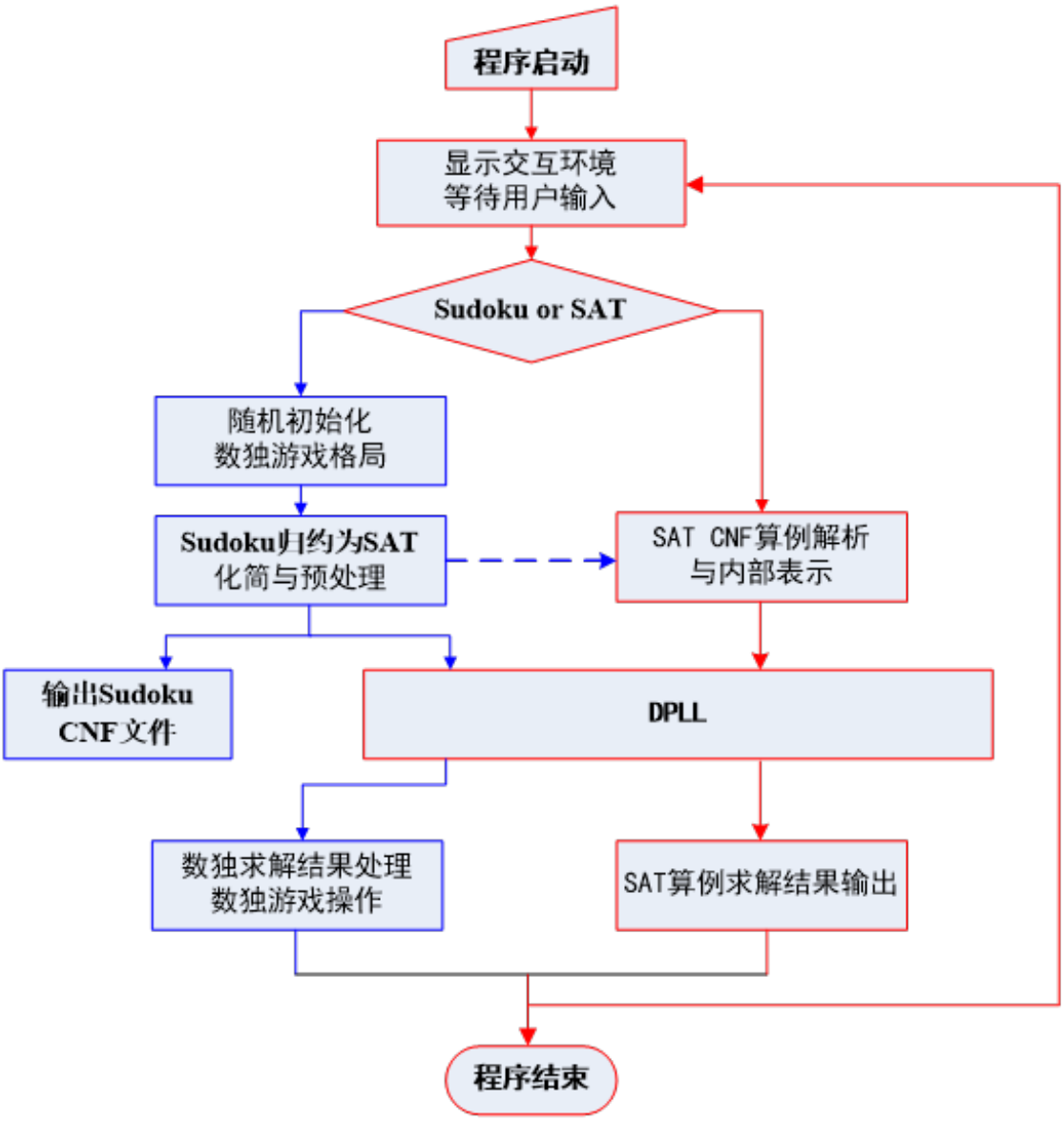

图 2-1　系统模块流程图

模块流程参考了任务书中已经有的内容。

3 系统详细设计

3.1 有关数据结构的定义

在 SAT 模块中，主要处理范式中各个子句与变元的存储。由于每个 CNF 公式的变元与子句数可能不同，同一个实例中子句长度也可能不同，为能够开辟合适的空间大小，我们采用类似邻接表的数据结构存储。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。

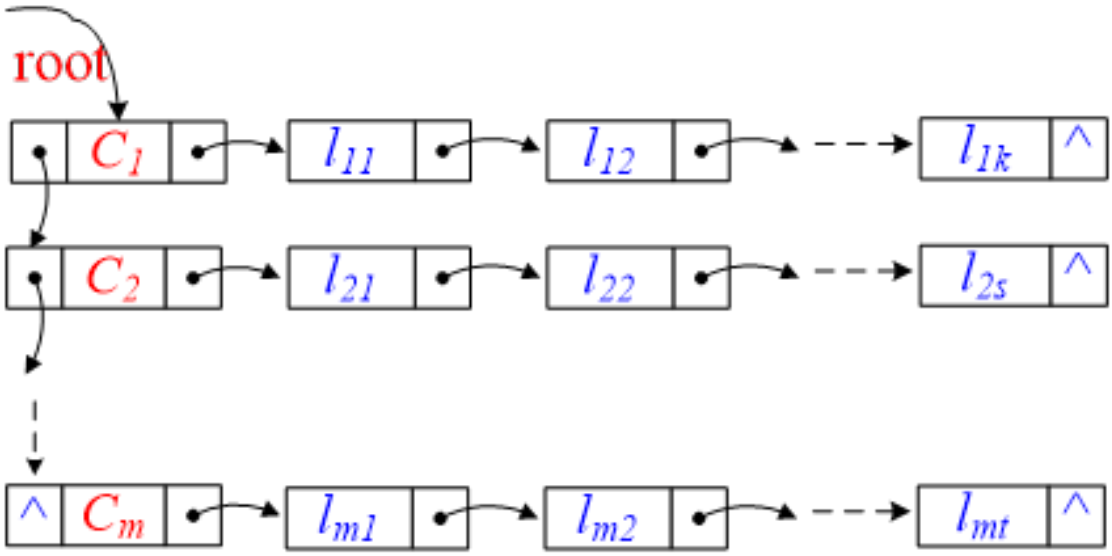

图 3-1 cnf 公式存储结构图

采用结构体的形式来保存子句与文字。

文字结构体中，有一个指针域，两个值域。指针域用来指向下一个与之相邻的文字，没有下一个相邻文字时指针值为 NULL。两个值域分别代表该文字的值（带正负号），和标志域（标志该文字是否在 dpll 的过程中被删除）。

子句结构体中，有两个指针域，两个值域。指针域用于指向下一个相邻的子句与该子句中第一个文字，没有所指时，指针值为 NULL。值域一存储该子句当前所含文字数量（随着文字的删除和恢复而改动），值域二为标志域，与文字结构体中的标志域类似。

为了方便我们记录 SAT 问题的具体情况，我们设置了问题结构体，其中包括记录 cnf 文件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，会用到与 SAT 模块中类似的数据结构。同样地，为了便于存储数独答案，我也设置了数独问题结构体，其中有存储数独答案和当前题面的数组，有记录当前题面非 0 数字个数的变量，同时设立 flag 数组判断是否为初始题面数字（避免用户在输入求解过程中修改初始题面的数字）。

3.2 主要算法设计

3.2.1 SAT 模块

CNF 文件解析：

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 求解：

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：将对应数据域赋值为递归深度，初始深度为 1）

之后，根据分裂规则，基于不同的变元选取策略（用户可选择）选择变元，假设选取到的变元为真，将其作为下一次递归中的目标单子句，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，说明原 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。

生成文件、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，还可以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 双数独模块

CNF 范式规约：

对于单数独而言，每一个格子可按语义编码为 1-9 之间数字构成的三位整数 ijn，i, j, n∈ $\{1, 2, \cdots, 9\}$，其中 i 表示单元格的行号，j 表示单元格的列号，n 表示单元格 <i,j> 填入的数字 n，用一定的转换公式将语义表示的格子变为用 1-729 中的数字表示的变元。

对于双数独，我们只需要再加一个整数 k，k 为 1 表示左上数独，k 为 2 表示右下数独, 这样便可用 729*2=1458 个变元表示双数独的所有格子（重叠部分的格子可用两种变元表示）。解决完变元的问题之后，我们需要处理约束条件，包括行约束、列约束、九宫格约束、单格约束（一个格子只能填入一个数字）、重叠部分等价约束，将以上约束规则全部处理成子句的形式，录入一个 cnf 文件中。之后，再根据已有的数独格局，在 cnf 文件中写入单子句（已有数独格局中已被填充数字的那些格子）。

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，随机选择一个位置填入任意的数字，将此初始棋盘转换为 cnf 文件后，用 dpll 求解以得到完整棋盘（也即数独游戏的最终解）。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，则代表该位置的数字一旦删除，数独便会出现多解，故该位置的数字不能删除，则寻找新的可删除位置。本程序可根据用户选择的难度不同挖掉不同数量的洞，最多可挖 90 个洞并保证数独的唯一解。

实现可玩性与交互性：

用户可以输入填写位置与填入数据进行数独的求解，同时也可选择“提示”“检查答案”“显示答案”功能。“提示”功能将会在用户未填入的格子中随机抽取一个填入正确的数字；“检查答案”可检查用户当前输入求解的内容是否正确；“显示答案”将直接显示数独的最终答案并终止游戏。

文件输出：

程序在初次 DPLL 求解生成完整棋盘后，将会把完整棋盘输出至 txt 文件中。

DPLL 求解：

双数独游戏中使用的 DPLL 过程与 SAT 模块中的相仿，此处不再赘述。

4 系统实现与测试

4.1 系统实现

4.1.1 系统实现环境

本程序在 Windows11 系统下采用 Dec-C++ 进行编译调试，编程语言选择 C 语言。

4.1.2 系统菜单演示

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行问题，需要用户输入给定的 cnf 文件，通过 SAT 求解器来求解。Sudoku 是求解数独游戏，根据用户选择的难度来生成数独题面，实现与用户的简单交互，能够让用户输入求解数独。主菜单如图 4-1 所示。

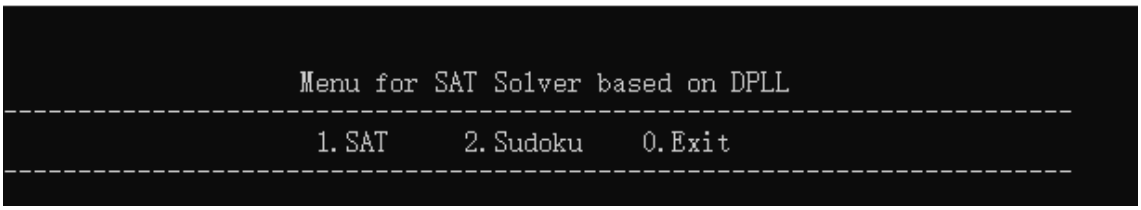

图 4-1 主菜单界面

选择 1.SAT 后，系统会出现 SAT 界面，如图 4-2 所示。

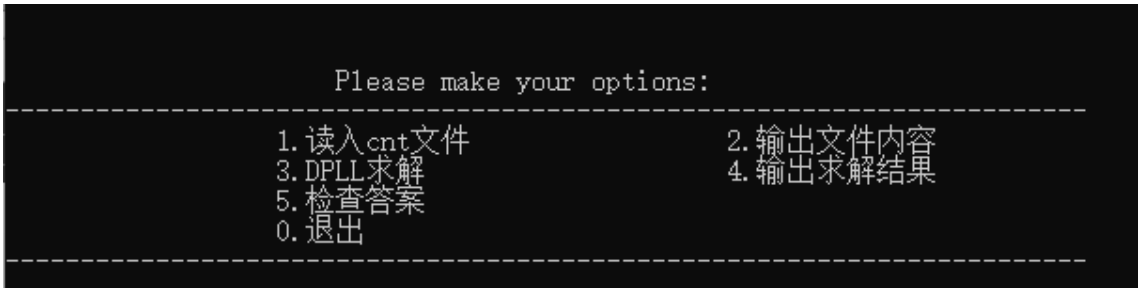

图 4-2 SAT 菜单界面

其中：

功能 1：实现 cnf 文件的读取，为结构体分配空间，将数据存储到结构体中，并输出该文件的变量数与子句数；

功能 2：将 cnf 文件中的数据打印出来，显示为 cnf 字句集。每一行数据前

面会多加一个数字代表当前行（子句）有多少文字；

功能 3：调用 DPLL 算法，求解 SAT 问题。求解完成时显示求解时间。求解完成后会生成同名的 res 文件并将求解结果输入进文件；

功能 4：将 DPLL 算法求解 SAT 得到的结果打印在屏幕上；

功能 5：检查 DPLL 算法求得的结果是否正确。

选择 2.Sudoku 后，系统会出现 Sudoku 界面，如图 4-3 所示。

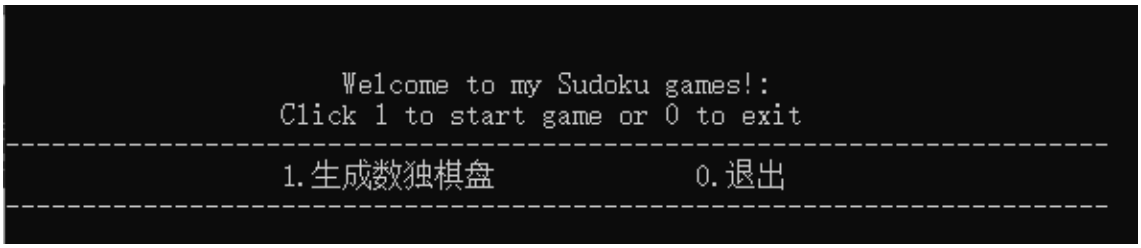

图 4-3　数独菜单界面

选择 1 即可生成一个完整的数独格局并保存在 txt 文件中，此后根据用户选择难度的不同基于挖洞法生成不同的数独题面。

4.1.3　常量与结构体声明

程序中的常量与结构体声明如下：

```
#define MAX_VARNUM 30000
#define ERROR 0
#define OK 1

typedef struct Literal{      //文字
   struct Literal* nextnode;  //下一个文字
   int value;              //该文字的值（带正负）
   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）
}node;

typedef struct Clause{       //子句
   struct Clause* nextcla;   //下一子句
   node* first;            //该子句的第一个文字
```

```
    int literal_num;        //该子句的文字数量
    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）
}clause;

typedef struct Question{     //SAT问题
    int vexnum;             //文字数
    int clanum;             //子句数
    int fre_lit[MAX_VARNUM+1];
        //各文字出现的次数（在无单子句而要选下一个删除关键字时使用）
    //选取出现频率最高的那个关键字作为待删除关键字
    //frequency_literal
    int ans[MAX_VARNUM+1];    //（有解时的）解
}question;

typedef struct Sudproblem{    //数独问题
    int original[16][16];     //待解棋盘（随着与用户的交互不断更新）
    int finalans[16][16];     //棋盘正解
    int flag[16][16];         //判断棋盘上的位置是否未初始题面所给数字的位置
    //为避免用户在输入求解时，误输入了初始题面所给的位置，故设立flag数组
    int num;                //非0个数
}sud;

typedef int status;
```

4.1.4 函数声明

SAT 模块的主要函数：

1) void SATQuestion()

进入 SAT 模块，显示 SAT 菜单界面，根据用户选择功能的不同调用不同的函数。

2) void init()

初始化函数，将 root 所指的空间及其相连的空间全部释放（如果有此空间）；将问题结构体 qq 中记录的子句数、变量数、用于保存 DPLL 求解答案和记录文字出现次数的数组全部初始化。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 SAT 问题。本函数将在每一次读取文件前被调用，保证在读入文件前，问题 qq 是已经初始化的。

3) void ReadFile(char filename[200])

调用该函数，打开作为参数被传入的文件名，函数将读取 cnf 文件中的各个子句，并将信息存储在已经被初始化的问题结构体 qq 中，并通过链表将各文字与各子句连接，root 指针指向第一个子句节点。函数可跳过文件的注释行，也可将注释行完整输出。

4) void PrintFile()

调用该函数，打印 cnf 文件中所有子句所含的文字情况（通过遍历文字节点和子句节点构成的链表实现），可以通过调用这个函数来检验文件读入过程是否正确。如果还没有读取文件，函数将会输出“尚未读取文件”来提醒用户。

5) void DPLL()

调用该函数，程序将开始 SAT 问题的求解与递归调用。注意，该函数仅是求解 SAT 问题的开头，而非递归调用的主体。在计算完成之后，函数将回答此问题可否满足，同时输出求解时间。最后，将生成与输入的文件同名但后缀名为.res 的文件。

求解 SAT 问题的关键函数与生成文件的函数将在后文介绍。

6) status func(int depth, int tar)

该函数会在 DPLL 函数中被首次调用，之后作为递归的主体函数被反复调用。

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

另一个参数 tar 为当前运行的线索值，当子句集中存在单子句时，tar 为单子句中对应的文字（带有正负号）。若在寻找单子句之后 tar 被赋值为 0，说明子句集中已没有单子句，此时要调用其他的函数为 tar 赋值代入下一层递归。

接下来介绍 func 函数的运行逻辑：

当 tar 有值时，我们反复根据单子句传播规则对子句集进行化简（即对对应的子句节点和文字节点进行屏蔽操作），直至没有单子句。在化简后，我们将进行分裂策略，通过一些策略选取一个变元为真（本程序共有 5 中选取变元的方式，将在后文说明）。在选取这个变元后，我们将其作为线索值，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，我们将恢复到上一层调用，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 qq 结构体中。

7) status FindSingleClause()

该函数在每一次递归调用 func 函数时首先被调用，通过遍历子句集查找其中的单子句。由于我们事先在子句节点中保存了该子句中（未被屏蔽的）文字数量，故只需要遍历所有的子句节点即可找到单子句。当我们找到单子句时，将其中保存的文字值（带正负号）作为函数的返回值返回；如果没有找到单子句，函数返回 0 作为未找到的提示。

8) status DeleteTarget(int depth, int single)

调用该函数，将从问题 qq 中，依据单子句传播规则，从子句集中删除文字或子句。

函数的参数 single 是我们的线索值，函数将删除子句集中含有 single 的所有子句，与含有-single 的文字，执行该操作的同时更新问题 qq 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。

在执行删除时，我们将修改问题中保留的子句总数与每一个子句中的文字数。当某一子句的文字数为 0 时，也就是出现了空子句。根据分裂策略，这标志着我们的假设出现了错误，函数会给 func 函数返回 ERROR 以提示，之后 func 函数将会恢复递归至上一层，并做出反面假设。

需要注意的是，在执行删除操作时，我们并不希望更改链表结构，以便于进行回溯时将子句或文字恢复，所以此处的删除并不是实际意义上的删除，我们使用“屏蔽”二字来解释这个操作更为妥当。正因如此，我将递归层数作为函数的其中一个参数，在执行屏蔽操作时，将子句或文字的标志数据域中的值更改为

depth，当之后的操作中回溯至 depth 层时，可以很方便地找到哪些数据是在该层被“删除”的，从而让相应子句或文字重新恢复到子句集中。这样的设计给数据的删除与恢复带来了极大的便利，节约了我们备份一整个子句集的时间和空间。

9) void RecoverLastLevel(int depth)

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 qq 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。之前我们在屏蔽子句或文字时，将标志数据域的值设为了 depth，在此时便作为了是否进行恢复的标志，在本函数中，当前 depth 层做出的修改都将被复原。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。

10) void RecoverBegin()

调用该函数，使 qq 结构体恢复到刚读入文件结束的状态。该函数在每一次 func 函数执行之前被调用，使得一次读入文件之后可以用多种方法进行求解而不用重新读入文件。该函数将遍历子句节点与文字节点组成的结构体，将所有节点的标志数据域赋值为 0，恢复其有效性，同时更新 qq 结构体中记录文字出现次数的数组，更新记录子句总数的变量，以确保结构体回到刚读入文件结束的状态。

11) void FormAnsFile(int res, int time, char filename[200])

该函数将在 func 函数完成求解后，在 DPLL 函数中被调用。调用该函数，将形成与相应 cnf 文件同名后缀名为 res 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，满足时将记录每个变元的赋值，-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，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。

12) void PrintAnswer()

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。

13) void CheckAnswer()

根据问题 qq 中的记录，将求解答案代入到 root 指向的各个子句中，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。不过显然，对于不满足算例的检查结果，每一次都会是“所求解错误!”。

以下是执行分裂策略时选取变元所用的 5 种方法：

14) status FindShortestLastLiteral()

寻找最短子句的最后一个变元。我们在子句节点中设置了变量用来记录该子句中未被屏蔽的文字数，以便我们快速找到最短子句。找到最短子句后，我们选取最后一个未被屏蔽的文字节点，返回其值。

15) status FindShortestFirstLiteral()

寻找最短子句的第一个变元。思路与第一种方法类似，但返回值是最短子句中第一个未被屏蔽的文字的值。

16) status FindShortestMaxLiteral()

寻找最短子句的出现次数最多的变元。此方法要用到问题结构体 qq 中记录文字出现次数的数组，用来判断最短子句中哪个文字在整个子句集中出现次数最多，并返回其值。

17) status FindMaxLiteral()

寻找子句集中出现次数最多的变元。直接遍历问题结构体 qq 中记录文字出现次数的数组，选取出现次数最多的文字作为返回值。

18) status FindFirstLiteral()

寻找子句集中第一个未被屏蔽的变元，并返回其值。

双数独模块主要函数：

19) void Sudoku()

进入双数独模块，显示双数独界面，生成初始数独后允许用户选择题目的难度。生成数独与挖洞法完成后显示算法执行时间。双数独题面生成后，进入游戏界面，为用户提供“输入求解”“来点提示”“显示答案”“检查”“退出”五个选项，并根据用户选择功能的不同调用不同的函数。

20) void initsudo()

调用 init 函数初始化 root 指针和问题结构体 qq。初始化双数独结构体 sudo，清楚 sudo 中本来存有的数独格局。初始化操作保证了我们可以在不退出程序的前提下连续进行多次数独游戏。本函数将在每一次生成新的数度之前被调用，确保上一次的双数独不会影响下一次的数独格局

21) void CreateSudoku()

调用该函数，生成数独格局。首先随机选取棋盘上一位置填入 1-9 中的一个文字，将此初始棋盘存入 cnf 文件，读入此 cnf 文件并利用 DPLL 算法求解，将解再次转换成数独格局的形式并存入 sudo 结构体，并将生成的数独格局存入 txt 文件。

22) void AnsTransSudo()

调用该函数，将问题结构体 qq 中存储的初始数独棋盘，转换成数独格局的形式存入数独结构体 sudo 中。

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

调用该函数，将数独格局中的语义编码转换为自然顺序编码。输入 k，i，j，n 四个整型变量，意为第 k 个数独的第 i 行第 j 列填入数字 n，函数通过公式 $(k-1)*729+(i-1)*81+(j-1)*9+n$ 把 1111-2999 共 1458 个离散的变量转换为连续的 1458 个变量，为 DPLL 过程提供方便。

24) void FormSudoFile()

调用该函数，生成数独文件。该函数将在生成初始数独格局之后被调用，将数独结构体 sudo 中存储的数独保存至 txt 文件中，以便在程序外部查看。

25) void FormCnfFile(int flag)

调用该函数，将当前数独格局转换成自然语义编码存入 cnf 文件，以方便 DPLL 过程的求解。

函数首先将当前数独棋盘的特殊规则写入文件（将已填充的数字的语义编码转换为自然顺序编码 m，在文件中写入“m 0”，以表示当前位置已有数字），之后将基础规则（包括行约束规则、列约束规则、九宫格约束规则、单格约束规则、重叠部分等价规则）写入文件。文件的变元数共 1458 个，子句数为基础规则数 + 特殊规则数。

形成 cnf 文件之后，函数会调用 ReadFile 函数，将刚生成的文件读取，以备 DPLL 求解。

26) void DigHole(int diff)

调用该函数，基于挖洞法生成双数独题面。参数 diff 是用户在 Sudoku 函数中选择的题目难度，函数根据 diff 的不同挖不同数量的洞以生成难度不同的题目。挖洞的位置随机生成，直至挖完目标数量的洞，函数才会结束。

27) status IfCanDig(int x, int y)

调用该函数，判断 (x,y) 位置的数字被删除（挖洞）后数独是否仍为唯一解。函数将把不同于此位置当前数字的另外 8 个数字依次填入此位置，并基于改动后的数独棋盘生成 cnf 文件，使用 DPLL 求解。若有解，则说明删掉此位置后数独将不再是唯一解，故此位置数字不能删除，寻找新的删除位置。函数会将删除可行性表征为函数返回值。

28) void PrintSudoku()

调用该函数，打印当前数独题面，包括挖洞法生成的初始数独题面和用户已填入的数字。

29) void PrintAnsSudoku()

调用该函数，打印数独的最终解。此函数将在 Sudoku 函数中用户选择了“显示答案”之后被调用，打印最终解之后，将会结束游戏。

30) void InputToSolve()

调用该函数，进入输入求解模块。用户可以输入目标行列以及想要填入的数字进行数独的解答，如果输入了错误的位置或错误的填充数字，程序将给出相应的提示。

31) status CheckCurrentSolve()

调用该函数，检查当前输入求解的部分是否全部正确。若当前所求解有误，系统只会给出“当前解答有误”的提示，不会告诉具体错误的位置。

32) void hint()

调用该函数，使用数独游戏的提示功能。程序将随机选择棋盘上一个已被挖洞且还未填入数字的位置，填入正确的答案。

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。

1. 读入文件功能测试

此功能可读取 cnf 文件，将其中的注释部分打印，数据信息保存在问题结构体 qq 中，本测试将测试程序是否能成功读入文件并显示文件中的注释内容。测试 1 为正确文件名，测试 2 为错误文件名，可测试程序能否对异常输入做出相应的提示。测试结果如表格及图片所示：

序号	测试输入	目标文件	预期结果	运行结果
1	1	sat-20.cnf	打印文件注释内容，打印变量数、子句数，打印“已成功读取文件”	见图 4-4，符合预期
2	1	abcdefg	无法打开文件	见图 4-5，符合预期

表 4-1 读入文件功能测试

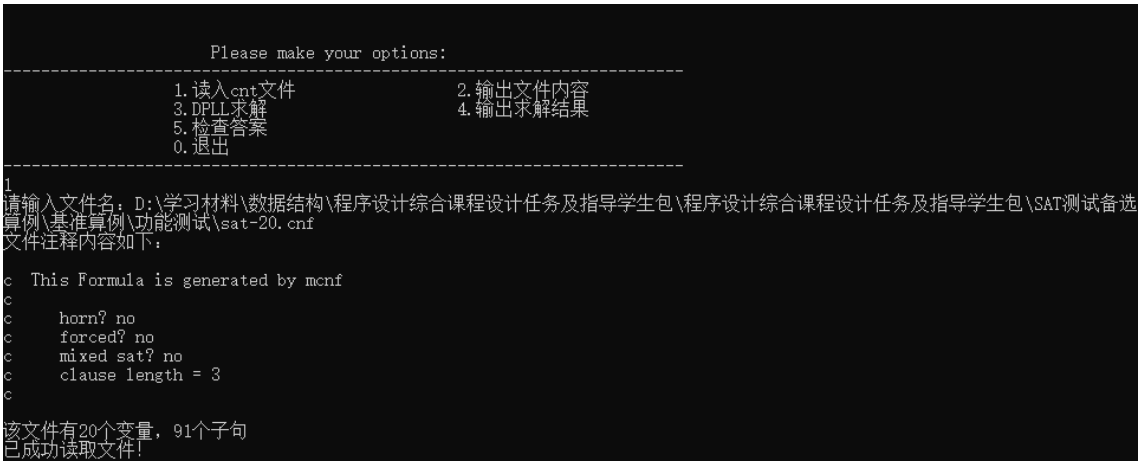

图 4-4　读入文件功能测试 1

图 4-5　读入文件功能测试 2

2. 输出文件内容功能测试

此功能可将已读入的 cnf 文件中有关子句的信息全部打印出来。本测试将测试程序是否能成功打印已读入文件的子句信息，并且是否能对未读入文件时的情况做出相应反应。

测试 1、2 均为正确文件名，测试 3 为未读入文件时的情况。测试结果如表格及图片所示：

序号	测试输入	目标文件	预期结果	运行结果
1	2	sat-20.cnf	打印文件中的子句内容	见图 4-6，符合预期
2	2	unsat-5cnf-30.cnf	打印文件中的子句内容	见图 4-7，符合预期
3	2	未读入文件	尚未读取文件	见图 4-8，符合预期

表 4-2　输出文件内容功能测试

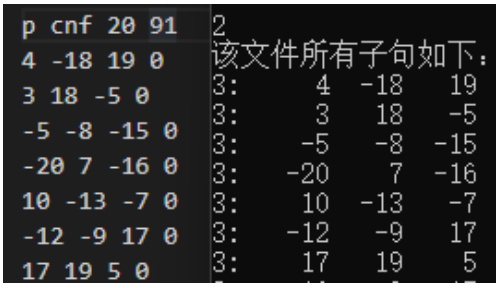

图 4-6　输出文件内容功能测试 1

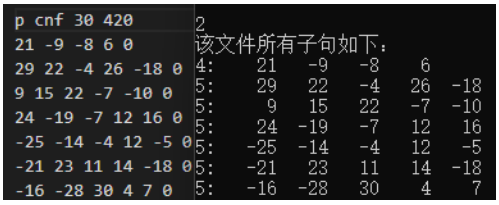

图 4-7　输出文件内容功能测试 2

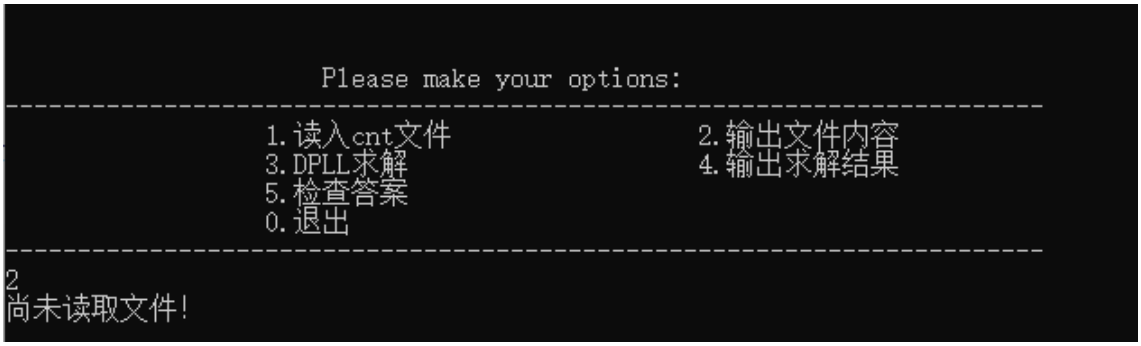

图 4-8　读入文件功能测试 3

测试 1、2 中，左侧是 cnf 原文件中的内容，右侧是程序输出的内容，可以看出，两者是一样的，这说明输出文件内容功能可以正常使用。

3.DPLL 功能测试

此功能可实现 SAT 问题的求解。本测试将测试程序对大中小型算例及对不可满足算例的求解，并输出求解时间。

为测试方便，本测试只选用 3 种求解方法进行连续求解，3 种求解方法只有执行分裂策略时的变元选取方法不同，前文已有介绍。

函数及对应方法为：func1：寻找最短子句的最后一个未被删除的文字；func2：寻找最短子句的第一个未被删除的文字；func3：寻找最短子句中在子句集中出现次数最多的文字。

测试结果如表格及图片所示：

序号	测试输入	目标文件	文件大小	运行结果
1	3	problem1-20.cnf	小型，20 个变量，91 个子句	见图 4-9
2	3	sud00012.cnf	中型，232 个变量，1901 个子句	见图 4-10
3	3	eh-dp04s04.shuffled-1075.cnf	大型，1075 个变量，3152 个子句	见图 4-11
4	3	u-dp04u03.shuffled-825.cnf	不可满足，825 个变量，2411 个子句	见图 4-12

表 4-3　DPLL 功能测试

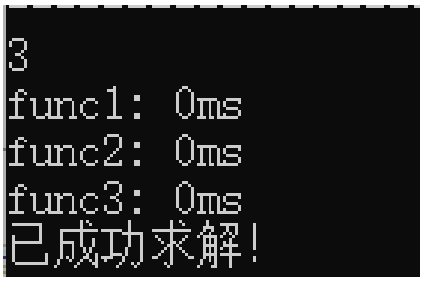

图 4-9　DPLL 测试 1

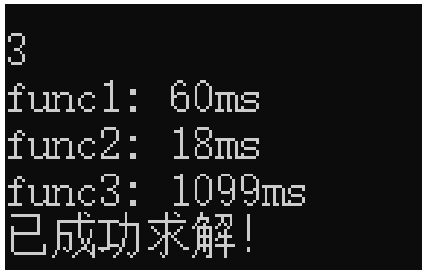

图 4-10　DPLL 测试 2

图 4-11　DPLL 测试 3

图 4-12　DPLL 测试 4

4. 输出求解结果功能测试

此功能在 DPLL 求解之后使用，将输出 DPLL 求解的结果。

本测试只测试该功能是否能正常使用以及打印到终端的答案与输出文件中的内容是否一致，故只选取两个小型算例进行测试。测试结果如表格及图片所示：

序号	测试输入	目标文件	文件大小	运行结果
1	5	problem1-20.cnf	小型，20 个变量，91 个子句	见图 4-13
2	5	u-dp04u03.shuffled-825.cnf	不可满足，825 个变量，2411 个子句	见图 4-14

表 4-4　输出求解结果功能测试

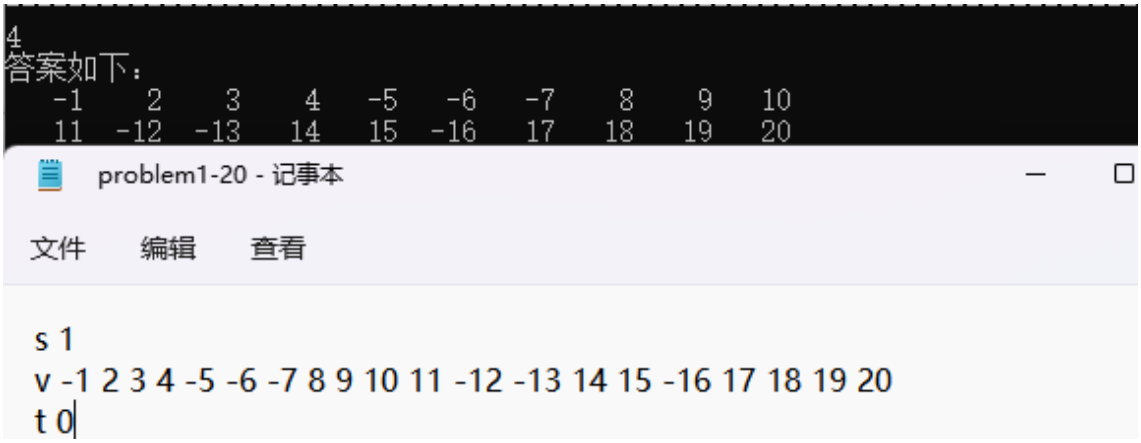

图 4-13　输出求解结果功能测试 1

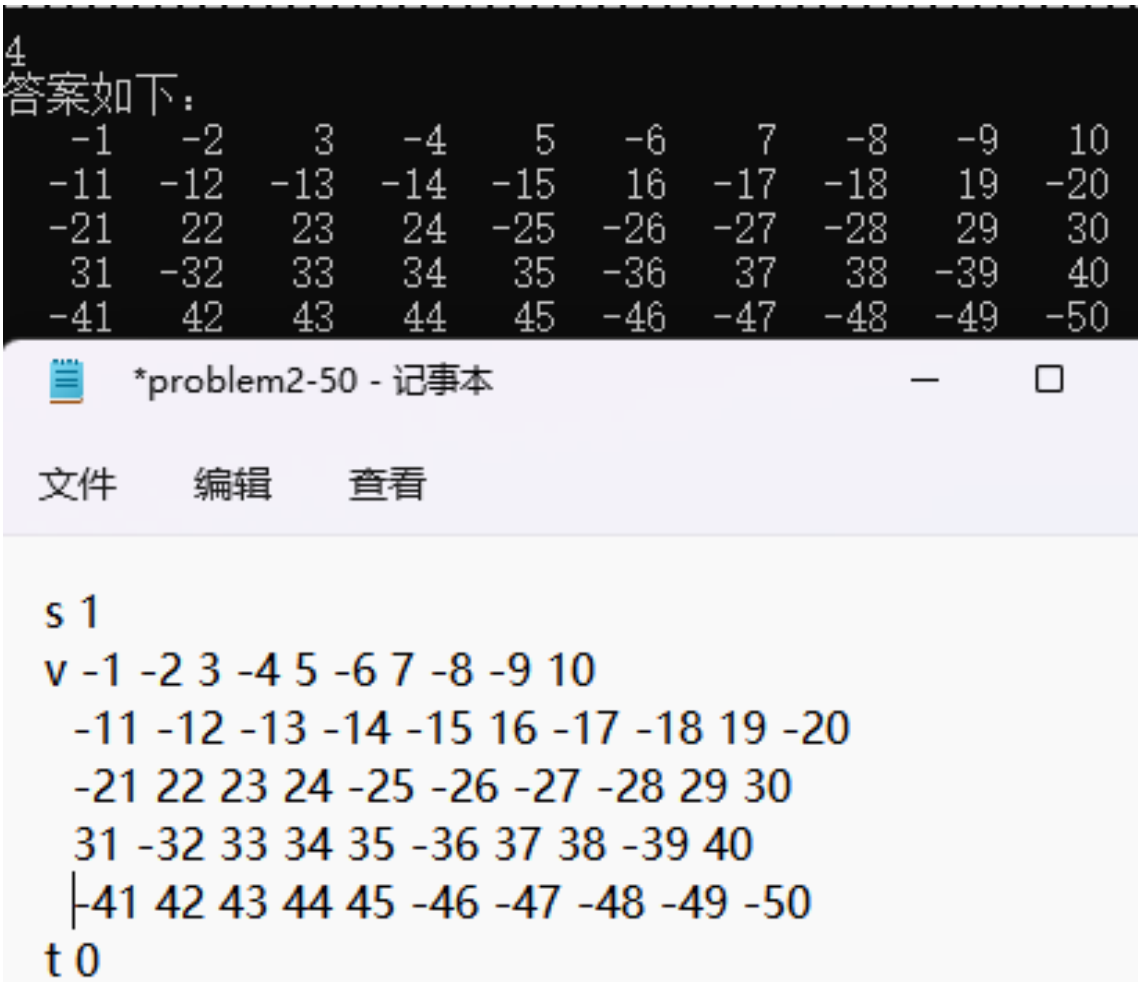

图 4-14　输出求解结果功能测试 2

可以看出，终端上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可行性和求解时间等内容。（这里为了截图方便，文件中的求解结果以 10 个数字为一行，而正常输出的 res 文件里所有求解结果都在一行中。）

5. 检查答案功能测试

此功能在 DPLL 求解之后使用，可实现对 DPLL 求解结果的验证。对于不满足算例的求解也可以使用该功能，但很显然，不满足算例的答案总是不正确的，故程序总会输出“所求解错误!”

测试结果如表格及图片所示：

序号	测试输入	目标文件	文件大小	运行结果
1	4	problem1-20.cnf	小型，20 个变量，91 个子句	见图 4-15
2	4	problem2-50.cnf	小型，50 个变量，80 个子句	见图 4-16

表 4-5　检查答案功能测试

图 4-15　检查答案功能测试 1

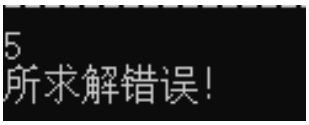

图 4-16　检查答案功能测试 2

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。本测试将选用三种不同的方法对读入的 cnf 文件进行求解，以 func4 函数得到的求解时间为基准，计算 func1 和 func3 对相同问题的优化率。

函数及对应方法为：func1：寻找最短子句的最后一个未被删除的文字；func3：寻找最短子句中在子句集中出现次数最多的文字；func4：寻找子句集中出现次数最多的文字。

（注：不同的算例想要减少 DPLL 的求解时间，采用的变元选取策略未必相同，当算例越大，这种区分越明显。因此不可能有一种算法能够起到绝对的优化作用，即使是使用寻找子句集中第一个未被删除的变元这种策略，在求解某些算例时用时也会极短，故在测试过程中可能存在负优化现象，属正常现象。）

测试结果如下表：（表中文件名后带有 × 标记的为不满足算例）

算例名（以.cnf 为后缀）	变量数/子句数	func1 用时(ms)	func3 用时(ms)	func4 用时(ms)	func1 优化率	func3 优化率
problem11-100	100/600	12	5	31	61.3%	83.9%
problem3-100	100/340	3	61	206	98.5%	70.4%
sud00009	303/2851	7	7362	10050	99.9%	26.7%
sud00079	301/2810	24	13192	1595	98.5%	-727.1%
sud00861	297/2721	56	6779	96	41.7%	-6961.5%
eh-dp04s04.shuffled-1075	1075/3192	129076	2793	4332	-288.0%	35.5%
u-dp04u03.shuffled-825(×)	825/2411	10271	1337	1078	-852.8%	-24.0%
u-problem7-50(×)	50/100	0	7	37	99.9%	81.1%
php-010-008.shuffled-as.sat05-1171(×)	80/370	5515	2920	3960	-39.7%	26.3%
u-problem10-100(×)	100/200	3	38570	546240	99.9%	92.9%

表 4-6　优化性能测试表

从表格可以看出，对于七成的算例而言，func1 与 func3 能够起到正向优化作用，且 func1 的优化性能更好。算例的大小与采用哪种方法计算更好并无确定的联系，需要分析算例中的子句结构才能用对应的方法节省计算的时间。

4.2.2　双数独模块测试

双数独模块测试均从主菜单进入双数独部分之后开始进行。

1. 数独生成测试

本测试将测试用户选择三种难度（简单、普通、困难）后，程序能否生成正常的数独。测试结果如表格及图片所示：

序号	选择难度	挖洞数量	生成数独总用时（ms）	运行结果
1	1（简单）	50	8571	见图 4-17、4-18
2	2（普通）	70	12747	见图 4-19、4-20
3	3（困难）	90	24850	见图 4-21、4-22

表 4-7　数独生成测试

图 4-17　简单数独生成时间

图 4-18　简单数独题面

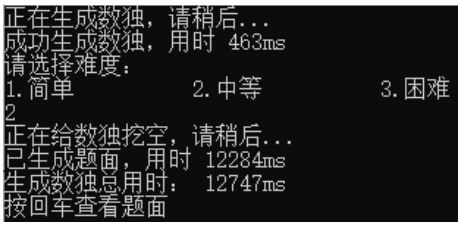

图 4-19　普通数独生成时间

图 4-20　普通数独题面

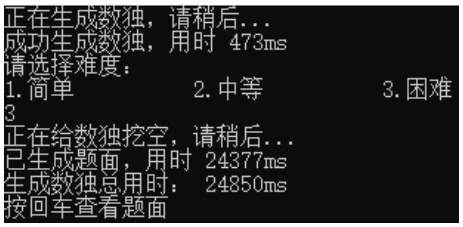

图 4-21 困难数独生成时间

图 4-22 困难数独题面

可以看出，生成数独的时间随着难度的增加而增加，这是因为基于挖洞法生成的数独，难度越大挖洞越多，且因为挖洞位置随机选取，选到已挖洞位置和非棋盘区域的次数会更多，从而进一步增加了生成数独的时间。但即使是困难难度的数独题面，生成时间也在 30s 以内，可以看出生成数独的速度是非常快的。

2. 用户交互测试

程序在用户选择完难度之后便会随机生成数独题面，生成完毕之后按回车键即可进入游戏界面，如图 4-23 所示。

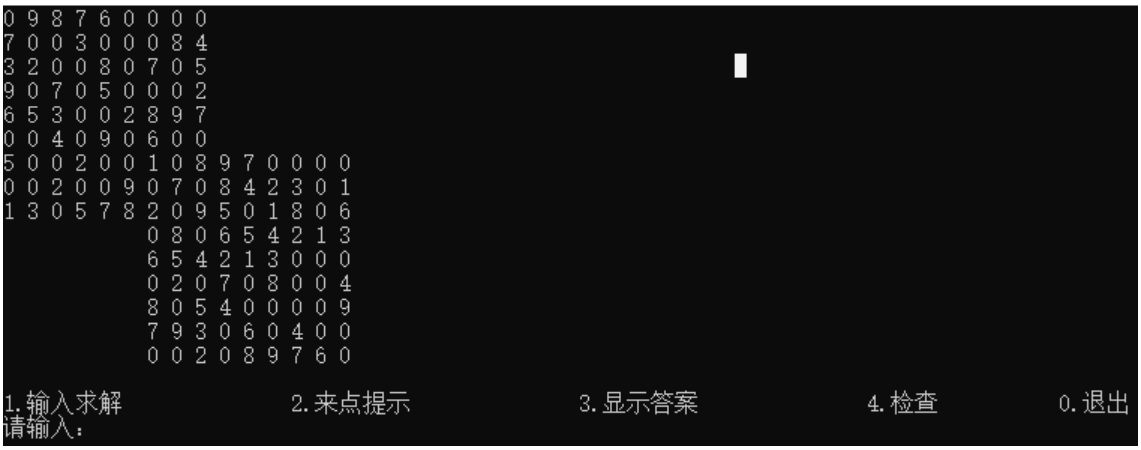

图 4-23 数独游戏界面

上图的数独格局正是数独生成测试中生成的困难数独题面，在进入游戏界面后，有如图所示的四种功能（退出除外）供用户选择，之后的测试将在此数独题面下进行。

功能一：输入求解数独

用户可以输入“1”以进入输入求解功能。用户需要手动输入要填写的目标位置（行、列）和要填入的数字，填写完成之后，下一次题面显示将会包含用户新填写的数字。

进入求解功能后输入“1 1 1”，下一次显示的题面如图 4-24 所示。

图 4-24　输入求解后的界面显示

可以看出，在第一行第一列的位置显示了数字“1”。

此外，为了防止游戏过程中用户误改初始题面所给数字，特意在数独结构体 sudo 中设置了 flag 数组，专门用来记录某数字是否为初始数字。利用 flag 数组，可以很容易的给用户提示，如图 4-25 所示。

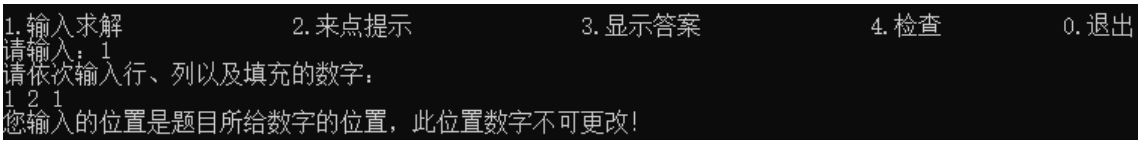

图 4-25　输入位置错误后的界面显示

此外，当用户输入位置正确但填入数字不在 0-9 范围内时，也会有对应的提示。

功能二：提示

用户可以输入“2”以获取提示，程序将随机选择一个未填入数字的位置，填入正确的数字。

使用一次提示功能后的界面显示如图 4-26 所示。

图 4-26　提示一次后的界面显示

与图 4-24 相对比，可以看出在 12 行 7 列的位置（图中高亮位置）填入了一个数字 3，即为程序所提示的数字。

功能三：检查

用户可以输入 “3”，以检查当前解答是否全部正确，为保证游戏的难度，程序只会给出当前输入求解是否全部正确的判断，而不会给出具体错误的位置。

在功能一的测试中，我们在一行一列输入了数字 1，显然初始题面第 9 行第 1 列已经有了数字 1，所以我们的解答应该是错误的，故检查功能的反馈如图 4-27 所示。

图 4-27　第一次检查

在将 1 行 1 列的数字改为 4 之后，再次检查，所得结果如图 4-28 所示。

图 4-28　第二次检查

功能四：显示答案

由于显示答案后游戏会直接结束，故将此功能放至最后。用户可输入“4”以直接显示本数独的最终答案，如图 4-29 所示。

图 4-29　显示答案后的界面显示

此外，当用户全部填写正确之后，系统将会显示求解成功的提示，如图 4-30 所示。

图 4-30　求解完成后的界面显示

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，完成了双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：

学习并了解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。通过查阅文献资料，了解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，此外，还认识了可满足性问题的研究历史与研究前景。

(2) 完成课程设计的初期设计任务：

我在了解 cnf 文件的基本形式与一般特征之后，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，复习了数据结构课程中的知识。此外，我完成了菜单系统的设计，可以实现简单的交互。最后，我设计了文件输出与答案检验功能，可以将求解答案按照指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，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) 解决数独问题：

理解双数独问题与 SAT 问题的联系，将双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 求解器对双数独进行求解获得双数独终解。通过挖洞法形成双数独题面，并在生成过程中保证形成的双数独是唯一解。对双数独游戏系统，设计了“输入求解”“来点提示”“显示答案”“检查”四个功能，使游戏更加完善。

5.2 工作展望

对于以后关于 SAT 问题的求解，有以下几点展望：

1. 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。

2. 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。

3. 可以改进在执行分离策略时选择变元的策略，从而使在 DPLL 的求解过程尽可能减少算法的回溯次数和搜索空间。

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。

6 体会

即使已经完成了 C 语言程序设计实验和数据结构实验，这一此的课程设计对我来说仍然是巨大的挑战，工程量、难度均比之前有较大提升，在最开始，我甚至有了“刚学会 1+1，就要写微积分作业”的感觉，但在仔细分析了题目以及四处查找资料过后，我对本次程序设计的认识逐渐明朗起来，思路也一点一点理顺，总结体会如下：

其一，前期的设计和准备是非常必要的。刚开始任务书上对 SAT 问题的介绍，对文字和子句的解释，让我一开始就有了理解上的困难。等好不容易理解了题意，又为如果设计数据结构存储文字及子句犯了愁。解决完数据结构的问题又开始了解 DPLL 算法、寻找优化的策略 …… 数独游戏的实现更是遥遥无期。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，我们可能在已经开始工程后反复地进行确认，这样的习惯无疑是十分糟糕的。对于平时的算法编程题我们首先更应正确地、完整地理解题意，这样子我们才能做好设计。在设计前期，关于程序基本框架的设计一定要找准。对于本次程序设计来说，像复习数据结构的知识这一点，就是不可或缺的前期工作之一。不进行前期的准备工作，甚至还有做到一半发现有无可修正的错误而重新开始的风险。因此，在程序设计前进行前期的准备工作是十分必要的。

其二，万事开头难。正因为第一点中所说的，我们有极其重要且大量的准备工作要做，我在真正写程序的时候迟迟不敢开始，因为总感觉还有哪里没有理清楚。但其实在这时题目的大致框架已经清晰，剩下的无非是一些具体函数的实现以及策略优化的内容。在今后的程序编写设计当中，不能过于犹豫，因为有些错误只靠前期构思是无法发现的，我们不可能通过一开始的准备工作就实现程序的无一纰漏。只有开始编写程序，才能在过程中涌现更多的思路。

其三，优先完成更重要的部分。在 SAT 问题基本实现的时候，我们很容易就会陷入到策略的优化上面，但是以目前的知识水平来看，我们几乎不可能实现全部算例的求解，有可能花费了两三天时间确是毫无进展。那么在这种已经可以求解但优化不够好的情况下，我们不如先进行之后的工作，比如数独游戏转换成 SAT 问题的实现，比如程序的特色功能的设计等等。这样即使程序优化不够完善，也可以有其他的部分来弥补。在实现某功能时，我们也要优先完成更重要的部分，比如 DPLL 的求解，先把主要求解的函数搞定（实现基本的框架），才

能搞清楚在求解过程中需要哪些函数，比如寻找单子句、分离策略时寻找变元等等。就像一棵树，先有了根和树干，才能开支散叶。程序中的 SAT 求解就是我们的树根树干。如果先写完了很多小函数，在真正实现求解的过程中会发现小函数不符合要求、或者根本用不上，这样相当于是白花了时间。

总结：我再一次复习了 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，在之后编写程序时也能更好的运用这一类知识。

我在课程设计中锻炼了自己的编程能力，数据结构的设计，搭建框架，完成操作处理 ··· 有许多事情都是自己独立完成的，这些事情也让我在工程设计方面的能力得到了提升。

我还在实验过程中提升了自己的时间管理能力，学习能力，不断调试、反复试错的过程也锻炼了我在编程时的心态，这对我来说是莫大的提升。通过查阅文献资料并尝试复刻其中的内容，我提高了查找文献资料的能力。我相信以后可以在应对类似的或者是难度更高的任务时更加从容，完成得更好。

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

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

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

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

[9] Sudoku Puzzles Generating: from Easy to Evil. http://zhangroup.aporc.org/images/files/Paper_3485.pdf

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

附录

Global.h

常量、结构体定义与函数声明。

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

using namespace std;

#define MAX_VARNUM 30000
#define ERROR 0
#define OK 1

typedef struct Literal{      //文字
   struct Literal* nextnode;  //下一个文字
   int value;              //该文字的值（带正负）
   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）
}node;

typedef struct Clause{       //子句
   struct Clause* nextcla;   //下一子句
   node* first;            //该子句的第一个文字
   int literal_num;        //该子句的文字数量
   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）
}clause;

typedef struct Question{     //SAT问题
   int vexnum;             //文字数
   int clanum;             //子句数
```

```
    int fre_lit[MAX_VARNUM+1];
        //各文字出现的次数（在无单子句而要选下一个删除关键字时使用）
    //选取出现频率最高的那个关键字作为待删除关键字
    //frequency_literal
    int ans[MAX_VARNUM+1];    //（有解时的）解
}question;

typedef struct Sudproblem{    //数独问题
    int original[16][16];     //待解棋盘（随着与用户的交互不断更新）
    int finalans[16][16];     //棋盘正解
    int flag[16][16];       //判断棋盘上的位置是否未初始题面所给数字的位置
    //为避免用户在输入求解时，误输入了初始题面所给的位置，故设立flag数组
    int num;              //非0个数
}sud;

typedef int status;

//SAT求解CNF文件
void SATQuestion();               //SAT模块
void init();                      //SAT初始化
void ReadFile(char filename[200]);   //读取文件
void PrintFile();                 //输出文件内容（仅子句）
void DPLL();                      //DPLL模块
void RecoverBegin();              //返回初始状态
status FindSingleClause();        //寻找单子句
status DeleteTarget(int depth, int single); //删除目标子句和文字
//以下是求解SAT问题的五个关键函数，分别使用了五种不同的方法
status func1(int depth, int tar);   //方法：寻找最短子句的最后一个文字
status func2(int depth, int tar);   //方法：寻找最短子句的第一个文字
status func3(int depth, int tar);   //方法：寻找最短子句出现次数最多的文字
status func4(int depth, int tar);
    //方法，寻找所有（未被删除的文字中）出现次数最多的文字
status func5(int depth, int tar);
    //方法：寻找所有（未被删除的文字中）第一个文字
//以上
```

```
void RecoverLastLevel(int depth);   // (递归时使用) 返回上一层递归时的状态
void PrintAnswer();                 //打印答案
void CheckAnswer();                 //检查答案
//以下是五种求解SAT问题的方法，对应上面的五个func函数
status FindShortestLastLiteral();   //寻找最短子句的最后一个文字 (对应func1)
status FindShortestFirstLiteral();
    //寻找最短子句的第一个文字 (对应func2)
status FindShortestMaxLiteral();
    //寻找最短子句中在问题里出现最多的文字 (对应func3)
status FindMaxLiteral();            //寻找一个出现最多的文字 (对应func4)
status FindFirstLiteral();          //寻找第一个文字 (对应func5)
//以上
void FormAnsFile(int res, int time, char
    filename[200]);//将答案保存在res文件中

//数独功能
void Sudoku();                  //数独模块
void initsudo();                //数独初始化
void CreateSudoku();            //创造一个数独
void AnsTransSudo();
    //将ans数组中保存的答案，转移到sudo.finalans数组中
status VarTrans(int k, int i, int j, int
    n);//将数独的1-9数字转化成自然顺序编码
void FormSudoFile();            //创造数独之后，将数独保存至txt文件中
void FormCnfFile(int flag);
    //将数独数字转换为自然语言编码并存入cnf文件
void DigHole(int diff);             //挖洞法创造数独题面
status IfCanDig(int x, int y);      //判断当前位置可否挖洞
void PrintSudoku();                 //打印当前题面
void PrintAnsSudoku();              //打印数独答案
void InputToSolve();                //输入求解模块
status CheckCurrentSolve();         //检查当前输入求解的内容是否正确
void hint();                    //数独的提示功能
```

main.cpp

主函数模块, 负责主菜单页面的显示。

```
#include "Global.h"

/* run this program using the console pauser or add your own getch,
    system("pause") or input loop */
int choice = 1;
char FileName[200] = {'\0'};
question qq;
sud sudo;
clause * root = NULL;

int main(void){
   while(choice){
      fflush(stdin);
      system("cls"); printf("\n\n");
      printf("                Menu for SAT Solver based on DPLL
                \n");
      printf("-------------------------------------------------------------------------\n")
      printf("                 1.SAT    2.Sudoku   0.Exit
                        \n");
      printf("-------------------------------------------------------------------------\n")
      cin >> 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();       //进入数独模块
      }
      else if(choice != 0){
         fflush(stdin);
         printf("输入错误! 请重新输入1或2或0! ");
         getchar();
      }
```

```
    }
    free(root);
    return 0;
}
```

CnfParser.cpp

CNF 解析模块，负责 SAT 模块的菜单显示、cnf 文件的读取、SAT 问题答案的打印、检查及答案文件的生成。

```
#include "Global.h"


extern char FileName[200];
extern question qq;
extern clause* root;

void SATQuestion(){
    int op = 1;
    while(op){
        fflush(stdin);
        system("cls"); printf("\n\n");
        printf("                    Please make your options:
                        \n");
        printf("------------------------------------------------------------------------\n")
        printf("                1.读入cnt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")
        cin >> 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");
            fflush(stdin);
            getchar();
        }
        else if(op == 2){
            fflush(stdin);
            PrintFile();
        }
        else if(op == 3){
            fflush(stdin);
            DPLL();
        }
        else if(op == 4){
            fflush(stdin);
            PrintAnswer();
        }
        else if(op == 5){
            fflush(stdin);
            CheckAnswer();

        }
        else if(op != 0){
            fflush(stdin);
            printf("输入错误! 请重新输入! ");
            getchar();
        }
    }
}
```

```
void init(){
   clause* cla_tem = root;
   while(cla_tem != NULL){
      node* liter_tem = cla_tem->first;
      while(liter_tem != NULL){
         node* rel_liter = liter_tem; //release_literal
         liter_tem = liter_tem->nextnode;
         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));
   memset(qq.fre_lit,0,sizeof(qq.fre_lit));
   qq.clanum = 0; qq.vexnum = 0;
}

void ReadFile(char filename[200]){
   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'};
      //   printf("文件注释内容如下: \n\n");
      while((c = fgetc(fp)) == 'c'){
         //      cout << c << ' ';
         fgets(str, 5000, fp);
         //      cout << str;
      }
```

```
//    cout << endl;
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
node* tem2_r = NULL, *tem2_l = NULL; //指向文字，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，故不用再开辟空间
      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 = (node*)malloc(sizeof(node));
      tem2_r->value = num_cur;
      qq.fre_lit[abs(num_cur)]++;
      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;     //结尾
            cnt = 0;
        }
        tem1->nextcla = NULL;           //结尾
    }
    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;
    node* tem2;
    while(tem1 != NULL){
        tem2 = tem1->first;
        cout << tem1->literal_num <<": ";
        while(tem2 != NULL){
            printf("%5d",tem2->value);
            tem2 = tem2->nextnode;
        }
        cout << endl;
        tem1 = tem1->nextcla;
    }
    printf("-----------------------------------------\n");
    getchar();
}

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
int opt, res;      //res记录求解结果（有解还是无解）

printf("1.func1(last)  2.func2(first)  3.func3(shortmax)
    4.func4(max)  5.func5(simple)\n");
printf("请选择你要使用的方法：");
scanf("%d",&opt);
time_t t1, t2;
t1 = clock();
if(opt == 1) res = func1(1,0);
else if(opt == 2) res = func2(1,0);
else if(opt == 3) res = func3(1,0);
else if(opt == 4) res = func4(1,0);
else if(opt == 5) res = func5(1,0);
else{
    printf("请输入1~5中的数字！\n");
    getchar();
}
t2 = clock();
printf("此方法用时：%dms\n",t2-t1);
if(res == OK) printf("已成功求解！\n");
else printf("此题无解！\n");

//实验报告测试专用
// time_t t1,t2;
// t1 = clock();
// res = func1(1,0);
// t2 = clock();
// printf("func1: %dms\n",t2-t1);
// time_t t3,t4;
// t3 = clock();
// RecoverBegin();
```

```
    // func3(1,0);
    // t4 = clock();
    // printf("func3: %dms\n",t4-t3);
    // time_t t5,t6;
    // t5 = clock();
    // RecoverBegin();
    // res = func4(1,0);
    // t6 = clock();
    // printf("func4: %dms\n",t6-t5);
    // if(res == OK) printf("已成功求解! \n");
    // 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
    cout << "答案如下: " << endl;
    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) cout << endl;
    }
    getchar();
}

void CheckAnswer(){//检查答案
    int res = 0, flag = 0;
    for(clause* cla = root; cla; cla = cla->nextcla){
        flag = 0;
        for(node* lit = cla->first; lit; lit = lit->nextnode){
            if(lit->value == qq.ans[abs(lit->value)]){
                //一个子句中只要有一个是真的, 那么子句就是真的, 说明此解是正确的
                flag = 1;
                break;
```

```
            }
        }
        if(!flag) break;                //flag非1即0，0说明求得的解是错误的
        //能到这里说明一个子句中没有一个文字与求得的解相同
    }

    if(flag) printf("所求解正确！\n");
    else printf("所求解错误！\n");
    getchar();
}

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
            if(!qq.ans[i]) fprintf(fp," %d",i);
            else fprintf(fp, " %d", qq.ans[i]);
        }
    }
    fprintf(fp, "\nt %d", time);      //写入求解时间
    fclose(fp);
}
```

Solve.cpp

DPLL 求解模块，负责 SAT 问题的核心求解部分。

```
#include "Global.h"

extern char FileName[200];
extern question qq;
extern clause* root;

void RecoverBegin(){//返回初始状态
   memset(qq.fre_lit, 0, sizeof(qq.fre_lit));
   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文字数
      for(node* lit = cla->first; lit; lit = lit->nextnode){
         lit->flag = 0;           //恢复各文字节点的标志域
         qq.fre_lit[abs(lit->value)]++;  //恢复各文字出现的次数
         cla->literal_num++;          //恢复子句节点中记录的文字数
      }
      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){
          //首先要求该子句中未被删除的文字只有一个，然后要求该子句没有被删除
         for(node* p2 = p1->first; p2; p2 = p2->nextnode){
             //已经找到了对应单子句，现在要找到具体是该子句中的哪个元素
            if(!p2->flag){
                //还没有被删除的元素就是目标元素
               sin_cla = p2->value;
               judge = 1;                    //减少计算量
               break;
            }
         }
```

```
            if(judge == 1) break;                    //减少计算量
        }
    }
    return sin_cla;                                  //没找到单子句时返回0
}

status DeleteTarget(int depth, int single){//删除目标子句和文字

    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
        node* lit = cla->first;
        while(lit){
            if(lit->flag){                           //该文字已经删除，判断下一文字
                lit = lit->nextnode;
                continue;
            }
            if(lit->value == single){
                cla->flag = depth;                   //删除子句
                node* lit2 = cla->first;
                while(lit2){
                    //需再从头遍历一遍看里面有哪些文字，对应的数量（fre_lit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.fre_lit[abs(lit2->value)]--;
                        lit2 = lit2->nextnode;
                        continue;
                    }
                    lit2 = lit2->nextnode;
                }
                qq.clanum--;                         //总子句数要减一
                break;
            }
            else if(lit->value == -single){          //仅删除该文字
                lit->flag = depth;
```

```
            qq.fre_lit[abs(single)]--;
            cla->literal_num--;                    //该子句所含文字数减一
            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文字
   int len = MAX_VARNUM;
   int maxx = 0, res = 0;
   for(clause* cla = root; cla; cla = cla->nextcla){
      if(cla->flag) continue;
      if(cla->literal_num < len){  //寻找最短子句
         len = cla->literal_num;
         for(node* lit = cla->first; lit; lit = lit->nextnode){
            if(!lit->flag && maxx < qq.fre_lit[abs(lit->value)]) res =
                lit->value;
         }
      }

   }
   return res;
}

status FindShortestFirstLiteral(){//寻找最短子句中的第一个文字
   int len = MAX_VARNUM;
   for(clause* cla = root; cla; cla = cla->nextcla){
      if(cla->flag) continue;
      if(cla->literal_num < len){
```

```
            len = cla->literal_num;
            for(node* 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频率最高的文字
      int len = MAX_VARNUM;
      int maxx = 0, res = 0;
      for(clause* cla = root; cla; cla = cla->nextcla){
         if(cla->flag) continue;
         if(cla->literal_num <= len){  //寻找最短子句
            len = cla->literal_num;
            for(node* lit = cla->first; lit; lit = lit->nextnode){
               if(!lit->flag && maxx < qq.fre_lit[abs(lit->value)]){
                  maxx = qq.fre_lit[abs(lit->value)];
                  res = lit->value;
               }
            }
         }

      }
      return res;
}

status FindMaxLiteral(){//寻找所有（未删除）文字中，出现频率最高的文字
      int maxx = 0;
      int ans = 0;
      for(int i = 1; i <= qq.vexnum; i++){
         if(maxx < qq.fre_lit[i]){
            maxx = qq.fre_lit[i];
```

```
            ans = i;
        }
    }
    return ans;
}

status FindFirstLiteral(){//寻找所有（未删除）文字中的首个文字
    clause* cla = root;
    while(cla){
        if(cla->flag){
            cla = cla->nextcla;
            continue;
        }
        else{
            node* lit = cla->first;
            while(lit){
                if(lit->flag){
                    lit = lit->nextnode;
                    continue;
                }
                return lit->value;
            }
        }
    }
    return 0;
}

status func1(int depth, int tar){       //求解主要函数，递归思想
    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（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); //删除目标子句与文字
        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文字（注意这里得到的max_lit一定是正的）
    if(!max_lit) return OK;
        //如果没找到，说明所有文字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
        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){
        return OK;
    }
    RecoverLastLevel(depth+1);
    return ERROR;                   //走到这一步就说明已经不能求解了
}

status func2(int depth, int tar){      //求解主要函数，递归思想
    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（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); //删除目标子句与文字
        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 = FindShortestFirstLiteral();
```

```
    //寻找最短子句中第一个文字（注意这里得到的max_lit一定是正的）
  if(!max_lit) return OK;
    //如果没找到，说明所有文字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;                   //走到这一步就说明已经不能求解了
}

status func3(int depth, int tar){      //求解主要函数，递归思想
  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（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); //删除目标子句与文字
    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文字（注意这里得到的max_lit一定是正的）
  if(!max_lit) return OK;
    //如果没找到，说明所有文字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;                     //走到这一步就说明已经不能求解了
}

status func4(int depth, int tar){      //求解主要函数，递归思想
    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（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); //删除目标子句与文字
        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 = FindMaxLiteral();
        //寻找（未被删除的文字中）出现次数最多的文字
    if(!max_lit) return OK;
        //如果没找到，说明所有文字出现次数都已经变成0，此时说明已经正确求解
    if(func4(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(func4(depth+1, -max_lit) == OK){
        return OK;
    }
    RecoverLastLevel(depth+1);
```

```
    return ERROR;                    //走到这一步就说明已经不能求解了
}

status func5(int depth, int tar){      //求解主要函数，递归思想
    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（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); //删除目标子句与文字
        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（未被删除的文字中）首个文字
    if(!max_lit) return OK;
        //如果没找到，说明所有文字出现次数都已经变成0，此时说明已经正确求解
    if(func5(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(func5(depth+1, -max_lit) == OK){
        return OK;
    }
    RecoverLastLevel(depth+1);
    return ERROR;                      //走到这一步就说明已经不能求解了
}

void RecoverLastLevel(int depth){// (递归时使用) 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（删除操作）
            cla->flag = 0;             //恢复子句有效性
```

```
            qq.clanum++;                //恢复总子句数
            for(node*lit = cla->first; lit; lit =
                lit->nextnode){//寻找这一子句中在该层被改动的文字
               if(lit->flag == depth){
                  lit->flag = 0;        //恢复文字有效性
                  cla->literal_num++;     //恢复子句节点记录的文字数
                  qq.fre_lit[abs(lit->value)]++;//恢复该文字出现的次数
                  continue;
               }
               else if(!lit->flag){
                  qq.fre_lit[abs(lit->value)]++;
               }
            }
         }
         else{
            node* lit2 = cla->first;
                //子句没有被删除，但是其中的文字可能被删除了，要找一找有没有
            while(lit2){
               if(lit2->flag == depth){  //如果有在该层被删除的文字
                  lit2->flag = 0;         //恢复文字有效性
                  cla->literal_num++;     //恢复子句节点记录的文字数
                  qq.fre_lit[abs(lit2->value)]++;//恢复该文字出现的次数
                  lit2 = lit2->nextnode;
                  continue;
               }
               lit2 = lit2->nextnode;
            }
         }
      }
   }
```

Sudoku.cpp

双数独模块，负责双数独游戏的内容。

```
#include "Global.h"
```

```
extern sud sudo;
extern question qq;
extern clause* root;

void Sudoku(){//数独模块
   int op = 1;
   while(op){
      fflush(stdin);
      system("cls"); printf("\n\n");
      printf("                    Welcome to my Sudoku games!:
          \n");
      printf("               Click 1 to start game or 0 to exit
          \n");
      printf("-------------------------------------------------------------------------\n")
      printf("               1.生成数独棋盘          0.退出
          \n");
      printf("-------------------------------------------------------------------------\n")
      cin >> op;
      if(op == 1){
         fflush(stdin);
         initsudo();//初始化棋盘
         system("cls");
         printf("正在生成数独，请稍后...\n");
         time_t t1, t2;
         t1 = clock();
         CreateSudoku();//创造数独
         t2 = clock();
         printf("成功生成数独，用时 %dms\n",t2-t1);
         for(int i = 1; i <= 15; i++){
            for(int j = 1; j <= 15; j++){
               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("生成数独总用时：  %dms\n",t2-t1+t4-t3);
printf("按回车查看题面\n");
getchar();getchar();
int opp = 1;
while(opp){
   system("cls");
   PrintSudoku();  //打印当前题面
   if(sudo.num == 153){//双数独共153个数字
      printf("你已完成该数独的求解！\n");
      getchar();
      break;
   }
   cout << endl;
   printf("1.输入求解   2.来点提示   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
   }
   else if(opp == 3){
```

```
                fflush(stdin);
                system("cls");
                PrintAnsSudoku(); //打印最终答案
                printf("下次再努力求解吧\n");
                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();
            }
        }
    }
    else if(op != 0){
        fflush(stdin);
        printf("输入错误! 请重新输入! ");
        getchar();
    }
  }
}

void initsudo(){//数独初始化
  init();
  sudo.num = 0;
  for(int i = 1; i <= 15; i++){
    for(int j = 1; j <= 15; j++){
      if((i <= 6 && j > 9) || (i > 9 && j <= 6)){
```

```
            sudo.finalans[i][j] = -1;
            sudo.original[i][j] = -1;
            sudo.flag[i][j] = -1;
            continue;
        }
        sudo.finalans[i][j] = 0;
        sudo.original[i][j] = 0;
        sudo.flag[i][j] = 0;
    }
  }
}

void CreateSudoku(){   //创造一个数独
  srand((unsigned)time(NULL));
  int x = rand() % 9 + 1;
     //随机抽取上方棋盘的一个位置，填入1~9中的某个数字
  int y = rand() % 9 + 1;
  int v = rand() % 9 + 1;
  sudo.original[x][y] = v;
  sudo.num = 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);          //求解生成数独题面（答案存于qq.ans数组中）
  AnsTransSudo();
     //将ans数组中保存的答案，转移到sudo.finalans数组中
  sudo.num = 153;
  FormSudoFile();        //将生成的数独题面存入txt文件
}

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(k = 1; k <= 2; k++){ //第k个数独（1代表左上数独，2代表右下数独）
```

```
      for(i = 1; i <= 9; i++){//第i行
         for(j = 1; j <= 9; j++){//第j列
            for(n = 1; n <= 9; n++){//数字n
               if(qq.ans[VarTrans(k,i,j,n)] == VarTrans(k,i,j,n)){
                  if(k == 2){
                     sudo.finalans[i+6][j+6] = n;
                  }
                  else sudo.finalans[i][j] = n;
               }
            }
         }
      }
   }
}

status VarTrans(int k, int i, int j, int
    n){//将数独的1-9数字转化成自然顺序编码
   return (k-1)*729 + (i-1)*81 + (j-1)*9 + n;//即指导文档中的公式
}


void FormSudoFile(){//创造数独之后，将数独保存至txt文件中
   FILE* fp = NULL;
   if(!(fp = fopen("SudoProblem.txt", "w"))){
      printf("无法打开文件！ ");
      getchar();
   }
   else{
      for(int i = 1; i <= 15; i++){
         for(int j = 1; j <= 15; j++){
            if(sudo.finalans[i][j] == -1) fprintf(fp, " ");
            else fprintf(fp, "%2d", sudo.finalans[i][j]);
         }
         fprintf(fp, "\n");
      }
```

```
    }
    fclose(fp);
}

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}
        };//九宫格约束用
    if(!(fp = fopen(cnfname, "w"))){
        printf("无法打开文件\n");
        getchar();
    }
    else{
        fprintf(fp,"p cnf %d %d\n",729*2, 11988*2+162+sudo.num);
        int k, i, j, count, n;
        {//将特有规则写入文件（总共sudo.num条语句）
            //避免两个数独的重叠区重复录入，故分区块写入特有规则
            for(i = 1; i <= 9; i++){
                for(j = 1; j <= 9; j++){
                    if(sudo.original[i][j]) fprintf(fp, "%d 0\n",
                        VarTrans(1,i,j,sudo.original[i][j]));
                }
            }
            for(i = 7; i <= 9; i++){
                for(j = 10; j <= 15; j++){
                    if(sudo.original[i][j]) fprintf(fp, "%d 0\n",
                        VarTrans(2,i,j,sudo.finalans[i][j]));
                }
            }
            for(i = 10; i <= 15; i++){
```

```
        for(j = 7; j<= 15; j++){
            if(sudo.original[i][j]) fprintf(fp, "%d 0\n",
                VarTrans(2,i,j,sudo.finalans[i][j]));
        }
    }
}

{//将基础规则写入文件（单数独有11988条，共两个数独）
    for(k = 1; k <= 2; k++){
        for(i = 1; i <= 9; i++){
            //行约束，每行有9+36*9条规则，共9行
            for(n = 1; n <= 9; n++){          //9条规则
                for(j = 1; j <= 9; j++){
                    fprintf(fp, "%d ", VarTrans(k,i,j,n));
                }
                fprintf(fp,"0\n");
            }
            for(n = 1; n <= 9; n++){          //9*36条规则
                for(count = 1; count <= 9; count++){ //36条规则
                    for(j = count+1; j <= 9; j++){
                        fprintf(fp, "%d %d 0\n", -VarTrans(k,i,count,n),
                            -VarTrans(k,i,j,n));
                    }
                }
            }
        }

        for(j = 1; j <= 9; j++){
            //列约束，每行有9+36*9条规则，共9行
            for(n = 1; n <= 9; n++){          //9条规则
                for(i = 1; i <= 9; i++){
                    fprintf(fp, "%d ", VarTrans(k,i,j,n));
                }
                fprintf(fp,"0\n");
            }
```

```
    for(n = 1; n <= 9; n++){          //9*36条规则
      for(count = 1; count <= 9; count++){ //36条规则
        for(i = count+1; i <= 9; i++){
          fprintf(fp, "%d %d 0\n", -VarTrans(k,count,j,n),
              -VarTrans(k,i,j,n));
        }
      }
    }
  }

  for(count = 0; count < 9; count++){
     //九宫格约束，每个九宫格9+36*9条规则，共9个九宫格
    int x = 1 + dis[count][0] * 3;
    int y = 1 + dis[count][1] * 3;
    //[x][y]是每个九宫格左上角格子的位置
    for(int n = 1; n <= 9; n++){  //9条规则
      for(int m = 0; m < 9; m++)
      fprintf(fp, "%d ", VarTrans(k, x+dis[m][0], y+dis[m][1],
          n));
      fprintf(fp, "0\n");
    }
    for(int n = 1; n <= 9; n++){  //9*36条规则
      for(int p = 0; p < 9; p++){  //36条规则
        for(int q = p+1; q < 9; q++){
          fprintf(fp, "%d %d 0\n",-VarTrans(k, x+dis[p][0],
              y+dis[p][1], n), -VarTrans(k, x+dis[q][0],
              y+dis[q][1], n));
        }
      }
    }
  }

  for(i = 1; i <= 9; i++){        //81个格子，每个格子有1+36条规则
    for(j = 1; j <= 9; j++){
      for(n = 1; n <= 9; n++){
```

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

            for(count = 1; count <= 9; count++){//36条规则
              for(n = count+1; n <= 9; n++){
                fprintf(fp, "%d %d 0\n", -VarTrans(k,i,j,count),
                    -VarTrans(k,i,j,n));
              }
            }
          }
        }
      }

      {//将重叠部分的数独规则写入文件（共9*18 = 162条）
        for(int x1 = 7, x2 = 1; x1 <= 9 && x2 <= 3; x1++,x2++){
          for(int y1 = 7, y2 = 1; y1 <= 9 && y2 <= 3; y1++,y2++){
            for(n = 1; n <= 9; n++){
              fprintf(fp, "%d %d 0\n", -VarTrans(1,x1,y1,n),
                  VarTrans(2,x2,y2,n));
              fprintf(fp, "%d %d 0\n", VarTrans(1,x1,y1,n),
                  -VarTrans(2,x2,y2,n));
            }
          }
        }
      }
    }
    fclose(fp);

}

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 = 50;
    else if(diff == 2) target = 70;
    else if(diff == 3) target = 90;
    while(1){
        if(count_del >= target) break;
        int x = rand() % 15 + 1;
        int y = rand() % 15 + 1;
        if((x <= 6 && y > 9) || (x > 9 && y <= 6)) 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++;
        }
    }
}

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 PrintSudoku(){//打印当前题面
    for(int i = 1; i <= 15; i++){
        for(int j = 1; j <= 15; j++){
            if(sudo.original[i][j] == -1){
                printf(" ");
                continue;
            }
            printf("%d ",sudo.original[i][j]);
        }
        cout << endl;
    }
}

void PrintAnsSudoku(){//打印数独答案
    for(int i = 1; i <= 15; i++){
        for(int j = 1; j <= 15; j++){
            if(sudo.finalans[i][j] == -1){
                printf(" ");
                continue;
            }
            printf("%d ",sudo.finalans[i][j]);
        }
        cout << endl;
    }
}
```

```
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;
   cin >> x >> y >> v;
   if((x < 1 || x > 15) || (y < 1 || y > 15) || (v < 0 || v > 9)){
      printf("请输入合法的位置以及填充的数字! \n");
      getchar();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();getchar();
      return;
   }
   sudo.original[x][y] = v;
   if(v == sudo.finalans[x][y]) sudo.num++;

}

status CheckCurrentSolve(){//检查当前输入的解是否有误
   for(int i = 1; i <= 15; i++){
      for(int j = 1; j <= 15; 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 hint(){
   while(1){
      int x = rand() % 15 + 1;
      int y = rand() % 15 + 1;
```

```
        if((x <= 6 && y > 9) || (x > 9 && y <= 6) || sudo.original[x][y])
            continue;
        sudo.original[x][y] = sudo.finalans[x][y];
        sudo.num++;
        break;
    }
}
```