

**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： CS2003班**

**学 号： U202015375**

**姓 名： 汪宇飞**

**指导教师： 卢萍**

**报告日期： 2021年9月29日**

**计算机科学与技术学院**

# 目 录

[**目 录 1**](#_Toc84888724)

[**任 务 书 2**](#_Toc84888725)

[**1 引言 4**](#_Toc84888728)

[**2 系统需求分析与总体设计 5**](#_Toc84888732)

[2.1 系统需求分析 5](#_Toc84888733)

[2.2 系统总体设计 5](#_Toc84888734)

[**3 系统详细设计 6**](#_Toc84888735)

[3.1 有关数据结构的定义 6](#_Toc84888736)

[3.2 主要算法设计 6](#_Toc84888737)

[**4 系统实现与测试 11**](#_Toc84888743)

[4.1 系统实现 11](#_Toc84888744)

[4.1.1 系统实现环境 11](#_Toc84888745)

[4.1.2 用户需求与系统目标 11](#_Toc84888746)

[4.1.3 数据结构定义 11](#_Toc84888747)

[4.1.4 系统包含的函数 12](#_Toc84888748)

[4.2 系统测试 15](#_Toc84888749)

[4.2.1 测试大纲 15](#_Toc84888750)

[4.2.2 SAT问题求解测试与优化 15](#_Toc84888751)

[4.2.3 数独游戏测试 27](#_Toc84888752)

[**5 总结与展望 32**](#_Toc84888753)

[**6 体会 33**](#_Toc84888756)

[**参考文献 35**](#_Toc84888757)

[**附录 37**](#_Toc84888757)

[def.h 37](#_Toc84888752)

[CourseDesign.cpp 38](#_Toc84888752)

CNF.cpp [43](#_Toc84888752)

[DPLL.cpp 47](#_Toc84888752)

[Sudoku.cpp 53](#_Toc84888752)

# 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%)

# 1 引言

## 1.1 课题背景与意义

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。

在计算机科学中，SAT问题是确定是否存在满足给定布尔公式的解释的问题。换句话说，它询问给定布尔公式的变量是否可以一致地用值TRUE或FALSE替换，公式计算结果为TRUE。如果是这种情况，公式称为可满足。另一方面，如果不存在这样的赋值，则对于所有可能的变量赋值，公式表示的函数为FALSE，并且公式不可满足。例如，公式“a AND NOT b”是可以满足的，因为可以找到值a = TRUE且b = FALSE，这使得（a AND NOT b）= TRUE。相反，“a AND NOT a”是不可满足的。

## 1.2 国内外研究现状

国内外对于SAT问题的研究一直在进行中。在1997年和2003年，Kautz与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，以及蔡少伟、葛冬冬等人的倾力研究。

## 1.3 课程设计的主要研究工作

本次课程设计是基于DPLL算法求解合取范式，对变元、文字、子句、公式等设计有效的物理存储结构，实现SAT求解器。

并在该求解器的基础上进行存储结构、分支变元选取策略等优化，以及实现基于SAT问题的数独游戏。

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

系统以DPLL算法为基础，根据SAT问题的数据对文字、子句等结构设计物理存储结构，实现SAT问题的求解，对输入的CNF文件进行读取并建立内部存储结构，以DPLL算法求解并生成包括解答结果以及时间的文件。

系统另一部分以该SAT求解器为基础，生成并实现数独游戏，能和用户实现简单的交互。

## 2.2 系统总体设计

对于课程设计要求，系统分为五个部分：def.h、CourseDesign.cpp、CNF.cpp、DPLL.cpp和Sudoku.cpp。

def.h是对于系统数据结构的定义以及一些函数、常量和变量的声明；CourseDesign.cpp是程序的主界面显示，实现程序的大体功能；CNF.cpp是读取cnf文件生成相应的数据结构并生成SAT问题解答文件；DPLL.cpp是对于DPLL算法的实现；Sudoku.cpp是生成完整的数独格局与挖空生成数独游戏。

系统具体模块结构图如图2-1所示。



图2-1 系统模块结构

# 3 系统详细设计

## 3.1 有关数据结构的定义

系统中需要处理的数据为合取范式（CNF），合取范式又由若干个子句（Clause）组成，子句由若干个文字（Literal）组成，以及最后求解完成时包括求解结果、各个文字的值、求解时间的回答。

系统处理的数据以及其包含的数据项如表3-1所示。

表3-1

|  |  |  |
| --- | --- | --- |
| 处理的数据 | 包含的数据项 | 数据类型 |
| CNF文件 | 子句数 | 整型 |
| 文字数 | 整型 |
| 子句 | 整型 |
| 文字 | 整型 |

所建成的内部存储数据结构关联如图3-1所示。



图3-1 数据结构关联

## 3.2 主要算法设计

### 3.2.1 操作界面模块

这一模块为主界面显示与操作选择模块，显示操作提示供选择，具体流程如图3-2所示。



图 3-2

### 3.2.2 CNF读取模块

这一模块为实现CNF文件的读取的模块，接受所输入的文件路径后打开文件并开始读取，同时建立内部存储结构，具体流程如图3-3所示。



图3-3

### 3.2.3 DPLL算法模块

这一模块为实现DPLL算法的模块，基于两种处理策略。

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

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

交错使用上述两种策略可不断地对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树,如下图3-4所示。



图3-4 DPLL算法搜索树

代码框架如下。

DPLL( S) :

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

{

while(S中存在单子句) {//单子句传播

在S中选一个单子句L；

依据单子句规则，利用L化简S；

if S = Φ return(TRUE);

else if (S中有空子句 ) return（FALSE）；

}//while

基于某种策略选取变元v； //策略对DPLL性能影响很大

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

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

}

### 3.2.4 数独生成模块

该模块为完整的数独格局生成模块，首先使用rand函数随机生成第一行，之后调用递归函数生成后续八行。

### 3.2.5 数独游戏模块

该模块为数独游戏生成模块，调用3.2.4中的生成完整数独函数后，对完整的数独格局进行挖洞，并调用SAT求解器判断是否有唯一解，具体流程如图3-5所示。



图3-5

其中调用SAT求解器判断是否有唯一解时，对于除挖去的数字以外的8个数字填入后生成CNF文件并求解，若均为无解则为是有唯一解，可以挖去。

# 4 系统实现与测试

## 4.1 系统实现

### 4.1.1 系统实现环境

软件：Visual Studio 2019

硬件：处理器为AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz，RAM为16GB

### 4.1.2 用户需求与系统目标

实现对一定规模的SAT问题的求解，并输出保存求解结果与求解时间，以及将这一SAT问题求解器应用与数独游戏中。

### 4.1.3 数据结构定义

因此数据结构设计如下。

typedef struct LNode {//文字的结点结构

int sta;//文字的编号

struct LNode \*next;

}LNode,\*lit;

typedef struct CNode {//子句的结构

struct CNode \*next;

struct LNode \*first\_l;//指向子句中的第一个文字

}CNode,\*cla;

typedef struct CNFG {//合取范式CNF结构

struct CNode \*root;//第一个子句

int c\_num, l\_num;//子句和文字的数量

}\*cnf;

typedef struct Ans {//SAT问题求解的答案

int time, state, num;//DPLL函数用时，SAT问题的可符合性，文字的个数

int LiteralState[MaxLitNum+1];//可符合时各个文字的正负状态

};

### 4.1.4 系统包含的函数

以下为各文件中的函数声明与其功能，以及各个函数间的调用关系，具体定义见附录。

#### 4.1.4.1 CNF.cpp

该文件内的函数如下：

int GetNumber(FILE\* fp) //获取CNF文件中的数字

status reset(cnf S) //重置S

cnf OperateCNF(FILE\* fp,cnf S) //读取CNF文件，创建CNF结构

status AnswerFile(char \*filename) //生成SAT问题的答案文件

OperateCNF函数为这一文件的主要函数，它调用了reset函数和GetNumber函数。

reset函数对存储结构cnf进行重置以读取接收新的CNF文件内容并建立新的cnf存储结构。

GetNumber函数对CNF文件中的数字进行读取以建立数字链表。

AnswerFile函数对SAT问题求解的结果进行文件存储并输出，其中包含了SAT问题的求解结果、变元状态以及求解时间。

#### 4.1.4.2 DPLL.cpp

该文件内的函数如下：

status DPLL(cnf S,int flag) //DPLL算法主函数

cnf DeleteLiteral(cnf S, int l) //单子句规则化简S

status EmptyClauseExist(cnf S) //判断是否存在空子句

status FreeCNF(cnf S) //销毁S

status LiteralExist(cla c, int l) //判断子句c是否包含文字l

int SelectClause(cnf S) //寻找S中出现次数最多的一个文字并返回它的编号

cnf Sew(cnf S, int i) //将数据为i的单子句作为新cnf的根后续复制S，返回新cnf

cla SingleClause(cnf S) //判断是否存在单子句并返回第一个单子句

DPLL函数为该文件的主函数，它调用了剩余的各个函数。

DeleteLiteral函数对cnf结构进行单子句规则化简后返回化简过的cnf结构。

EmptyClauseExist函数对cnf结构进行逐子句遍历，如果存在空子句则返回1，否则返回0。

FreeCNF函数是对cnf结构进行销毁，防止栈溢出。

LiteralExist函数对cnf结构进行逐文字遍历，若发现了给定的文字则返回1，否则返回0。

SelectClause函数对cnf结构进行逐文字遍历，统计并比较各个文字出现过的次数，返回出现过最多次的文字的编号。

Sew函数将给定的数据作为新cnf结构的根，并将已有的cnf结构复制其后生成新的cnf结构并将其返回。

SingleClause函数对cnf结构进行遍历，如果发现了单子句则返回该单子句。

#### 4.1.4.3 Sudoku.cpp

该文件内的函数如下：

cnf AddSew(cnf s, int i) {//将i作为单子句插入到S中

status CertainAnwser(int a[][COL]) {//判断挖洞后的数独是否有解

void CreateSudoku(int a[][COL]) { //生成数独

status DigHole(int a[][COL],int n) { //挖洞

int Fill(int a[][COL], int i, int j) {//递归填充数独元素

status JudgeNumber(int a[][COL], int row, int col, int n){//判断填入的数是否符合要求

void PrintSudoku(int a[9][9]){//输出数独

cnf SudokuToCNF(int a[][COL]) {//数独数组转换为cnf

status WhetherToDig(int a[][COL], int n, int m, int k) {//判断挖空后是否有唯一解

CreateSudoku函数和DigHole函数为该文件的主函数，各函数的调用情况如图4-1所示。



图4-1

CreateSudoku函数首先随机生成数独格局的第一行，随后调用Fill函数生成后续8行。

Fill函数采用递归思想，根据已填入的第一行数据对后续8行的数独格局进行填充。

DigHole函数对于已由CreateSudoku函数生成的数独格局进行逐个数字挖洞，若挖去后有唯一解即可挖，否则不可挖，通过这样的方式生成数独游戏。

WhetherToDig函数实现DigHole函数中的判断是否有唯一解，分别调用JudgeNumber函数和CertainAnswer函数进行判断。

JudgeNumber函数对于所填入的数字1进行数独游戏规则判断是否符合，即同一行、同一列以及同一九宫格内每个数字仅出现一次，符合规则则返回1，否则返回0。

CertainAnswer函数调用DPLL函数判断数独格局是否有唯一解，有则返回1复制返回0。

SudokuToCNF函数实现数独格局向cnf存储结构的转化，将已填入的数字作为单子句与基准CNF文件相加，生成cnf存储结构。

AddSew函数将给定的数据作为cnf存储结构的新根并返回该存储结构。

## 4.2 系统测试

### 4.2.1 测试大纲

根据要求，测试分为两部分，SAT问题求解测试与优化和数独游戏测试。

SAT问题求解测试与优化对大中小各6个算例进行测试与优化分析。在主界面选择相应选项后输入算例路径即可求得结果，完成测试。

数独游戏对生成完整数独、生成数独游戏以及试玩数独游戏进行测试。在主界面选择相应选项后，再选择相应的功能，完成测试。

### 4.2.2 SAT问题求解测试与优化

优化方式：SAT问题求解器的优化是对变元选择策略的优化。最初的变元选择策略是选择出现过最多次的文字作为变元，经过多次测试后发现有时选取子句集中出现的第一个文字作为变元会大大减少DPLL算法的运行时间，故同时提供这两种变元选取策略供选择。

为区别起见，以下表中“优化状态”中，“是”表示选取第一个变元，“否”表示选取出现次数最多的变元。

根据要求，大中小算例各选取了6个，其中unsat-5cnf-30、qg4-08和4是不满足的算例，测试与优化结果如下所示。

1. 小型算例

表4-1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试算例 | 变元数 | 子句数 | 优化状态 | 测试结果 | 优化率 |
| ais6 | 61 | 581 | 否 |  | 22.2% |
| 是 |  |
| sat-20 | 20 | 91 | 否 |  | 负优化 |
| 是 |  |
| unsat-5cnf-30 | 30 | 420 | 否 |  | 18.1% |
| 是 |  |
| prolem1-20 | 20 | 91 | 否 |  | 0 |
| 是 |  |
| tst\_v25\_c100 | 25 | 100 | 否 |  | 50.0% |
| 是 |  |
| problem11-100 | 100 | 600 | 否 |  | 负优化 |
| 是 |  |

1. 中型算例

表4-2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试算例 | 变元数 | 子句数 | 优化状态 | 测试结果 | 优化率 |
| sud00009 | 303 | 2851 | 否 |  | 负优化 |
| 是 |  |
| ais10 | 181 | 3151 | 否 |  | 负优化 |
| 是 |  |
| ais12 | 265 | 5666 | 否 |  | 6.7% |
| 是 |  |
| sud00012 | 232 | 1901 | 否 |  | 负优化 |
| 是 |  |
| tst\_v200\_c220 | 200 | 220 | 否 |  | 20.0% |
| 是 |  |
| tst\_v200\_c210 | 200 | 210 | 否 |  | 33.3% |
| 是 |  |

1. 大型算例

表4-3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试算例 | 变元数 | 子句数 | 优化状态 | 测试结果 | 优化率 |
| qg4-08 | 512 | 9685 | 否 |  | 负优化 |
| 是 |  |
| qg7-09 | 729 | 22060 | 否 |  | 负优化 |
| 是 |  |
| sw100-1 | 500 | 3100 | 否 |  | 34.8% |
| 是 |  |
| sw100-70 | 500 | 3100 | 否 |  | 98.9% |
| 是 |  |
| bw\_large.a | 459 | 4675 | 否 |  | 47.3% |
| 是 |  |
| 4 | 512 | 9185 | 否 |  | 负优化 |
| 是 | 运行时长超过五分钟 |

### 4.2.3 数独游戏测试

以下为数独游戏测试。

1. 生成完整数独

如图4-2所示。

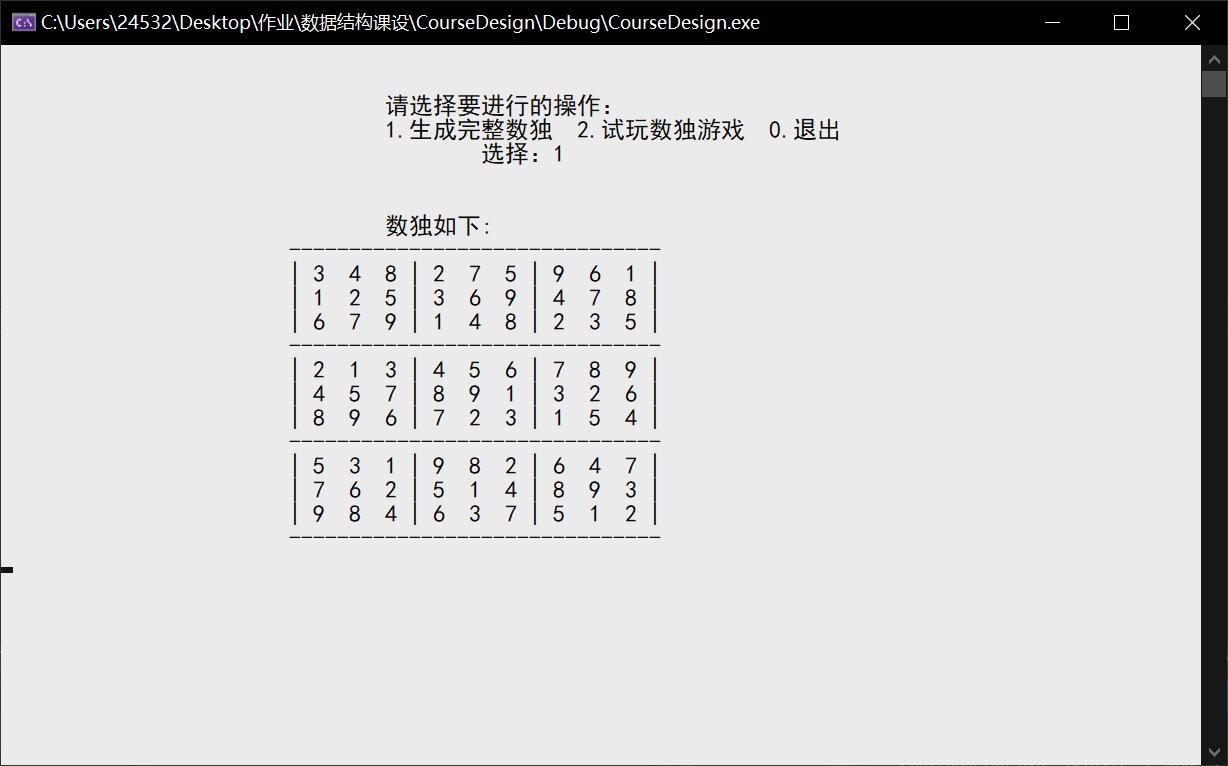


图4-2

1. 生成数独游戏

如图4-3所示。

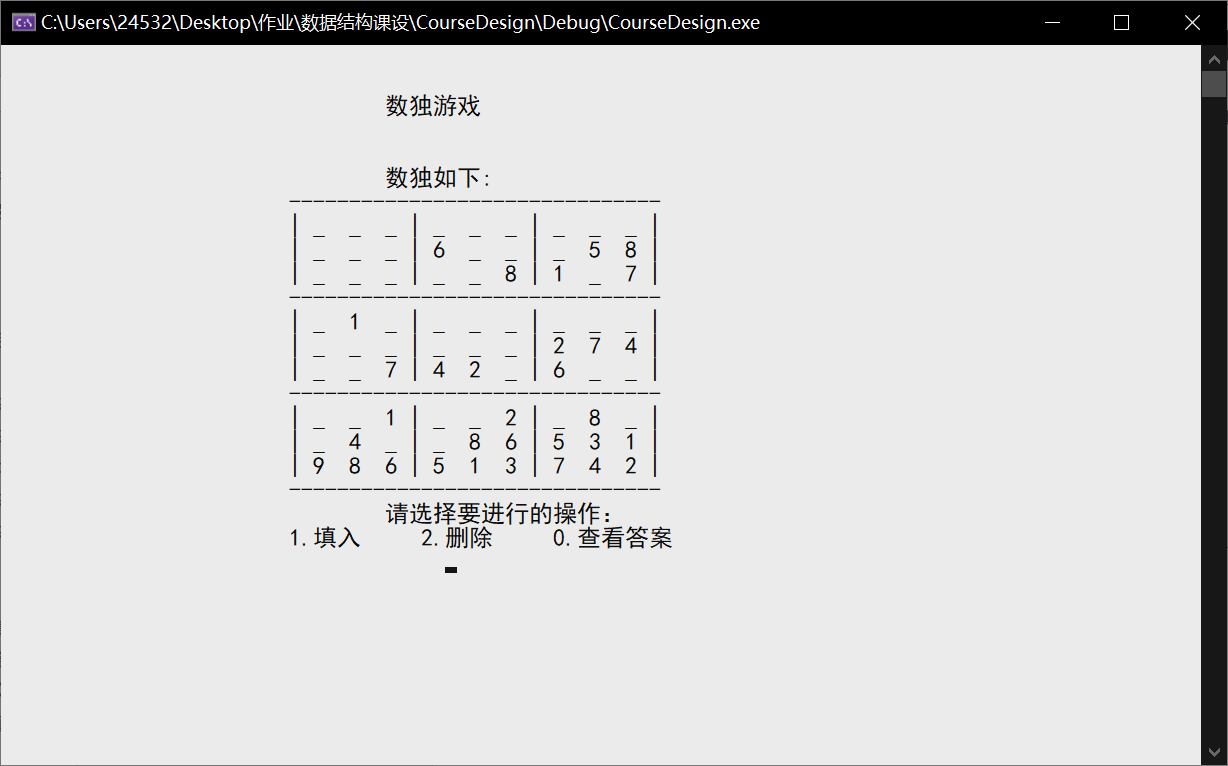


图4-3

1. 数独游戏试玩

基于（2）生成的数独游戏。

* + - 1. 查看答案

如图4-4所示。

* + - 1. 填入

1. 正确数字填入可填处，结果如图4-5所示。

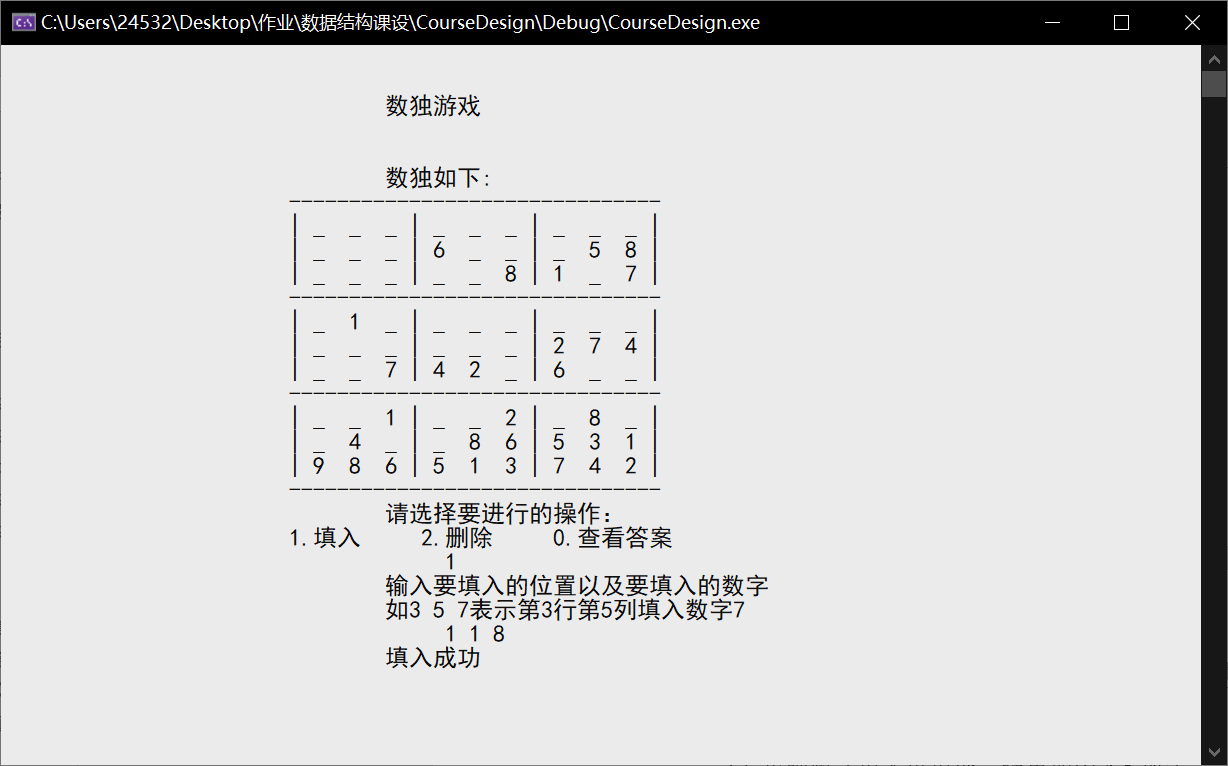


图4-5

1. 错误数字填入可填处，结果如图4-6所示。

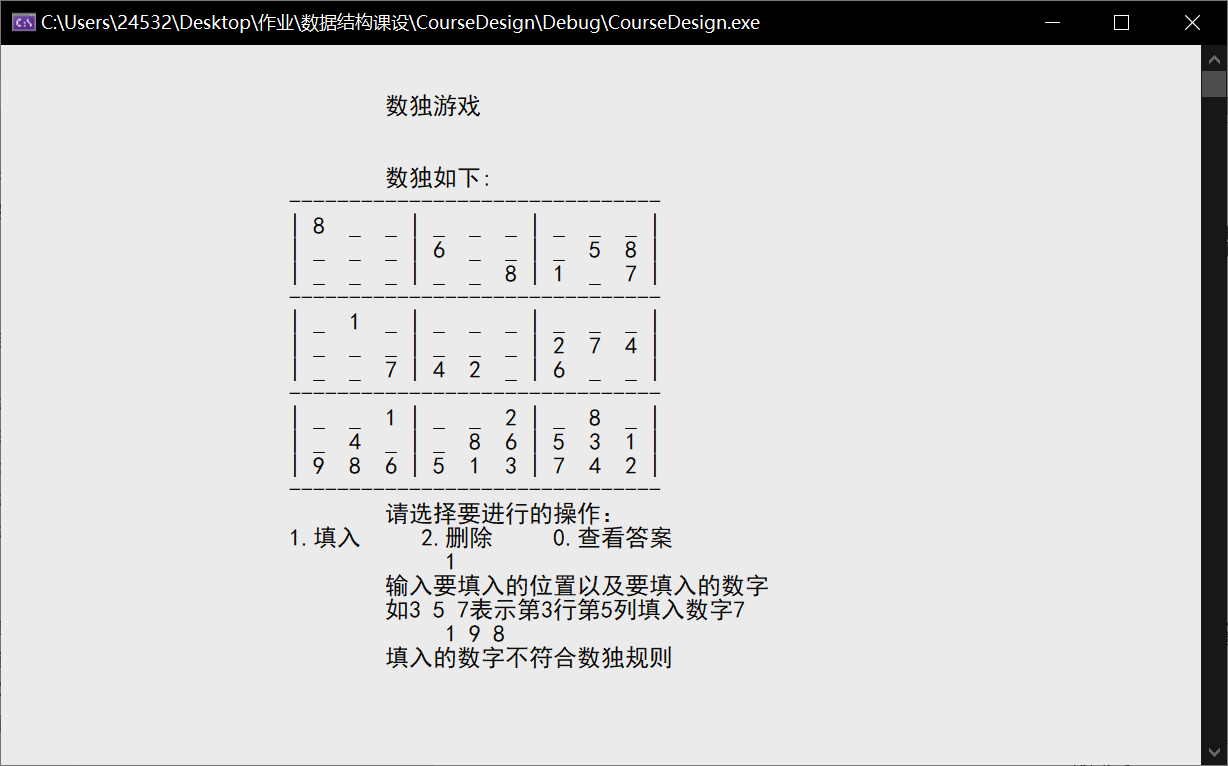


图4-6

1. 数字填入不可填出，结果如图4-7所示。



图4-7

* + - 1. 删除

1. 删除可删除数字，结果如图4-8所示。

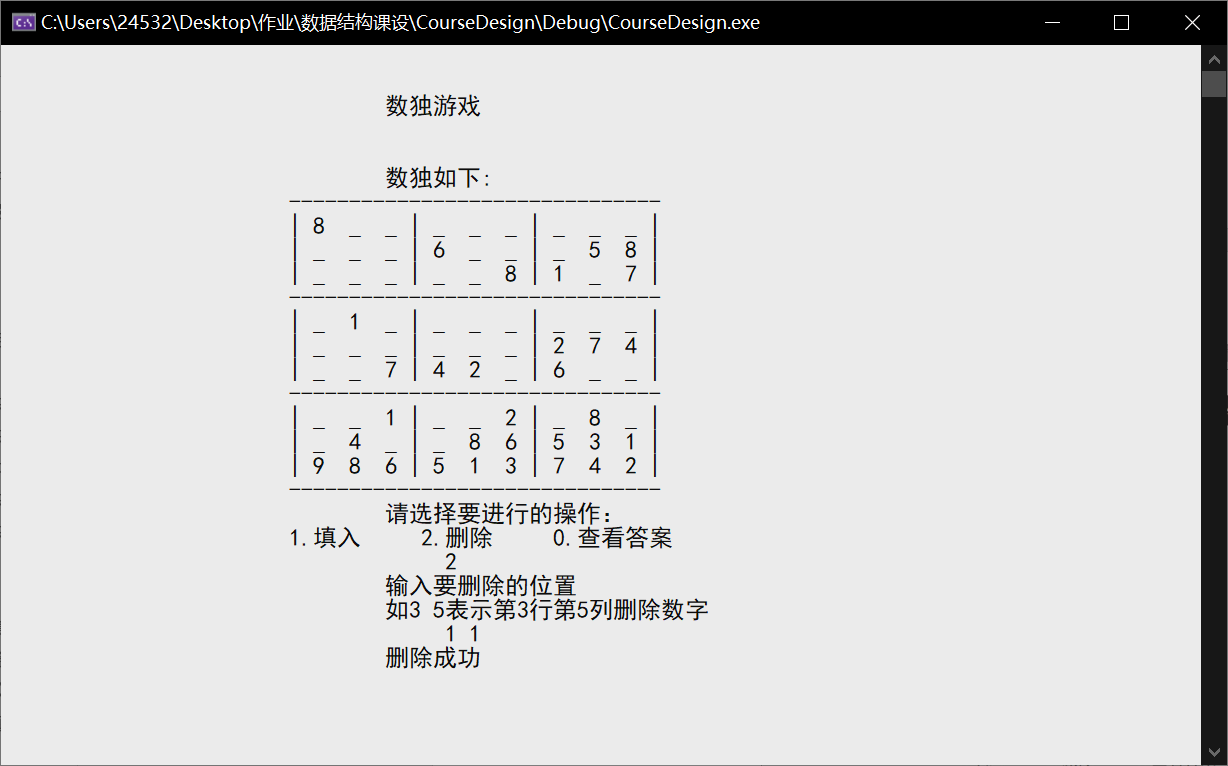


图4-8

1. 删除不可删除数字，结果如图4-9所示。



图4-9

# 5 总结与展望

## 5.1 全文总结

对自己的工作做个总结，主要工作如下：

（1）针对特定问题的数据类型与其关联设计了相应的邻接表数据结构。

（2）利用邻接表数据结构实现了CNF文件的读取和转换。

（3）利用邻接表数据结构实现的SAT问题求解器的编写。

（4）实现完整数独的生成。

（5）利用挖洞法和SAT问题求解器实现数独游戏的生成。

## 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）针对问题仔细分析要求与所给的信息，结合所学知识进行问题的具体化。

（2）对于具体问题多思考如何提取数据与数据之间的关系，以改造相应的数据结构。

（3）拓宽知识面，学习更多算法，深入了解体会其思想，并将其应用于实践中。

（4）努力学好专业课，掌握巩固已学知识，提升自己的编程水平。

# 6 体会

在暑假前老师发布这次课程设计的题目与要求后，我粗略看了一下，发现自己完全看不懂这些要求，于是最开始几天并没有继续着手做课程设计。但在之后我重新开始阅读题目和要求的时候，我强迫自己耐下性子，一行一行逐字逐句地看，终于有了一些眉目。在网上搜索了相关资料、对要求了解大致明白后，我开始写自己的代码。

中途也遇到了各种各样的困难，例如最开始的难题就是对CNF文件的处理，仅仅是读取CNF文件并存储这一部分我就写了很久，其中不停地调试几度让我失去耐心，但写完这一部分和其它一些部分之后，思路逐渐清晰了起来。

再之后处理最核心的DPLL算法时，也是遇到了类似的困难，在询问室友的帮助以及查阅资料后，花了很长时间终于写出可以正常运行的代码了。

总的来说，这次课程设计对于我而言是一次难题，各个部分都花了很长时间，但它也让我意识到了自己的不足，找到以后的学习的主要方向，同时它也让我复习巩固了已有的知识和学习了解了新的知识，对我有很大的帮助。

最后，感谢指导老师、助教以及我的室友，你们的帮助让我十分受益！

# 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

[11] 王静康,张凤宝,夏淑倩等.论化工本科专业国际认证与国内认证的“实质性”.高等工程教育研究,2014,5:1-4

[12] Stone J A, Howard L P. A simple technique for observing periodic nonlinearities in Michelson interferometers. Precision Engineering,1998,22(4):220-232

[13] 朱印红,袁衍明.Dreamweaver完美网页设计——技术入门篇.(第一版).北京:中国电力出版社,2006:19～20

[14] Lewis S L. Physics and chemistry of the solar system.北京:北京大学出版社,2014.1～2

[15] 陈剑.上博简《民之父母》“而得既塞於四海矣”句解释[EB/OL］.简帛研究网站，<http://www.bamboosilk.org/Wssf/2003/chenjian03.htm．2003-01-18>

附录

## def.h

//此文件为头文件，声明各个函数和定义

#pragma once

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<time.h>

#include<ctype.h>

#define status int

#define MaxLitNum 4000

#define OK 1

#define NOPE 0

#define TRUE 1

#define FALSE 0

#define pos 1

#define neg -1

typedef struct LNode {//文字的结点结构

int sta;//文字的编号

struct LNode \*next;

}LNode,\*lit;

typedef struct CNode {//子句的结构

struct CNode \*next;

struct LNode \*first\_l;//指向子句中的第一个文字

}CNode,\*cla;

typedef struct CNFG {//合取范式CNF结构

struct CNode \*root;//第一个子句

int c\_num, l\_num;//子句和文字的数量

}\*cnf;

typedef struct Ans {//SAT问题求解的答案

int time, state, num;//DPLL函数用时，SAT问题的可符合性，文字的个数

int LiteralState[MaxLitNum+1];//可符合时各个文字的正负状态

};

//以下是对一些数据的声明

extern Ans ans;

extern int digcount;

//以下是对于SAT的函数声明

cnf OperateCNF(FILE\* fp,cnf S);

status AnswerFile(char\* filename);

status DPLL(cnf S,int flag);//DPLL算法函数

void TraverseCNF(cnf S);

status FreeCNF(cnf S);

//以下是对于数独的函数声明

void PrintSudoku(int a[][9]);

void CreateSudoku(int a[][9]);

status DigHole(int a[][9], int n);

status JudgeNumber(int a[][9], int row, int col, int n);

## CourseDesign.cpp

#include"def.h"

int op = 1, sudokuop, sudoku[9][9], answer[9][9], setted[9][9];

char filepath[100];

FILE\* fp;

cnf S;

void main() {

clock\_t beg, end;

while (op) {

system("cls");

printf("\n\n");

printf("\t\t\t\t欢迎使用本系统，请选择要使用的功能：\n");

printf("\t\t\t\t1.SAT问题 2.数独 0.退出\n");

printf("\t\t\t\t\t选择：");

scanf("%d", &op);

switch (op) {

case 0:

getchar();

break;

case 1:

printf("请输入CNF文件路径：\n");

scanf("%s", filepath);

fp = fopen(filepath, "r");

if (!fp) {

printf("打开文件失败\n");

getchar(); getchar();

break;

}

S = (cnf)malloc(sizeof(CNFG));

S->root = NULL;

S = OperateCNF(fp,S);

//TraverseCNF(S);

int flag;

printf("\n请选择决策方式：\n");

printf("1.选择出现的第一个文字 2.选择出现次数最多的文字\n\n");

printf("选择：");

scanf("%d", &flag);

beg = clock();

ans.state = DPLL(S,flag);

end = clock();

ans.time = end - beg;

if (ans.state) {

printf("该SAT是可满足的\n\n");

for (int i = 1; i <= ans.num; i++) {

if (ans.LiteralState[i]) { printf("%5d ", ans.LiteralState[i] \* i); }

else { printf("%5d", i); }

if (!(i % 15)) printf("\n");

}

printf("\n\n");

}

else { printf("该SAT是不可满足的\n\n"); }

printf("DPLL算法用时为%d ms\n\n", ans.time);

AnswerFile(filepath);

getchar(); getchar();

break;

case 2:

sudokuop = 1;

while (sudokuop) {

system("cls");

printf("\n\n");

printf("\t\t\t\t请选择要进行的操作：\n");

printf("\t\t\t\t1.生成完整数独 2.试玩数独游戏 0.退出\n");

printf("\t\t\t\t\t选择：");

scanf("%d", &sudokuop);

switch (sudokuop) {

case 0:

break;

case 1:

CreateSudoku(sudoku);

PrintSudoku(sudoku);

getchar(); getchar();

break;

case 2:

CreateSudoku(sudoku);

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) { answer[i][j] = sudoku[i][j]; }

}

if (!DigHole(sudoku,49)) { printf("生成数独游戏失败\n"); }

else {

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) { setted[i][j] = sudoku[i][j]; }

}

int playop = 1, m, n, k;

while (playop) {

system("cls");

printf("\n\n");

printf("\t\t\t\t数独游戏\n");

PrintSudoku(sudoku);

printf("\t\t\t\t请选择要进行的操作：\n");

printf("\t\t\t1.填入 2.删除 0.查看答案\n\t\t\t\t ");

scanf("%d", &playop);

switch (playop) {

case 0:

printf("\t\t\t\t答案如下\n");

PrintSudoku(answer);

break;

case 1:

printf("\t\t\t\t输入要填入的位置以及要填入的数字\n");

printf("\t\t\t\t如3 5 7表示第3行第5列填入数字7\n\t\t\t\t ");

scanf("%d%d%d", &n, &m, &k);

if (m >=1 && m <=9 && n>=1 && n <= 9&&k>=1&&k<=9){

if (setted[n - 1][m - 1]) {

printf("\t\t\t\t该位置无法填入\n");

getchar(); getchar();

continue;

}

if (!JudgeNumber(sudoku, n-1, m-1, k)) {

printf("\t\t\t\t填入的数字不符合数独规则\n");

getchar(); getchar();

continue;

}

sudoku[n - 1][m - 1] = k;

printf("\t\t\t\t填入成功\n");

getchar(); getchar();

continue;

}

else {

printf("\t\t\t\t请输入正确的位置以及数字编号\n");

getchar(); getchar();

continue;

}

case 2:

printf("\t\t\t\t输入要删除的位置\n");

printf("\t\t\t\t如3 5表示第3行第5列删除数字\n\t\t\t\t ");

scanf("%d%d", &n, &m);

if (m >= 1 && m <= 9 && n >= 1 && n <= 9) {

if (setted[n-1][m-1]) {

printf("\t\t\t\t该位置无法删除\n");

getchar(); getchar();

continue;

}

sudoku[n - 1][m - 1] = 0;

printf("\t\t\t\t删除成功\n");

getchar(); getchar();

continue;

}

else {

printf("\t\t\t\t请输入正确的位置以及数字编号\n");

getchar(); getchar();

continue;

}

default:

printf("\t\t\t\t无该选项\n");

getchar(); getchar();

continue;

}

}

}

getchar(); getchar();

break;

default:

printf("无该选项\n");

getchar();

break;

}//end of switch

}//end of while

getchar();

break;

default:

printf("无该选项\n");

getchar(); getchar();

break;

}//end of switch

}//end of while

printf("\n\t\t\t\t使用结束 欢迎再次使用本系统\n\n");

}//end of main

## CNF.cpp

//CNF.cpp：此文件是对于合取范式CNF有关操作的函数的定义

#include"def.h"

Ans ans;

int GetNumber(FILE\* fp) {//获取CNF文件中的数字

char c;

int flag = 1, num = 0;

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

if (c == '-') {

flag = -1;

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

}

else if (c == '0') {

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

if (c == '\r') { fscanf(fp, "%c", &c); }

return num;

}

else if (feof(fp)) { return 0; }

while (c != ' ' && c != '\n' && c != '\r') {

num = num \* 10 + c - '0';

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

}

if (c == '\r') { fscanf(fp, "%c", &c); }

return flag \* num;

}

status reset(cnf S) {//重置S

S->c\_num = 0; S->l\_num = 0;

cla ctemp,cnext;

lit ltemp,lnext;

ctemp = S->root;

while (ctemp) {

cnext = ctemp->next;

ltemp = ctemp->first\_l;

while (ltemp) {

lnext = ltemp->next;

free(ltemp);

ltemp = lnext;

}

free(ctemp);

ctemp = cnext;

}

S->root = NULL;

memset(ans.LiteralState, 0, MaxLitNum + 1);

return OK;

}

cnf OperateCNF(FILE\* fp,cnf S) {//读取CNF文件，创建CNF结构

char c;

int temp,flag;

cla clap,clapre;

lit litp,litnext;

reset(S);

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

while (c == 'c') {

while (c == 'c') {

while (c != '\n' && c != '\r') { fscanf(fp, "%c", &c); }

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

if (c == '\n'){ fscanf(fp, "%c", &c); }

}

}

fscanf(fp, " cnf ");

S->l\_num = GetNumber(fp), S->c\_num = GetNumber(fp);

ans.num = S->l\_num;

while (1) {

temp = GetNumber(fp);

if (feof(fp)) { break; }

clap = (cla)malloc(sizeof(CNode));

clap->next = NULL;

if (!S->root) { S->root = clap; }

else { clapre->next = clap; }

if (temp) {

litp = (lit)malloc(sizeof(LNode));

clap->first\_l = litp;

litp->sta = temp;

litp->next = NULL;

temp = GetNumber(fp);

while (temp) {

litnext = (lit)malloc(sizeof(LNode));

litnext->sta = temp;

litnext->next = NULL;

litp->next = litnext;

litp = litnext;

temp = GetNumber(fp);

}

}

else {

clap->first\_l = NULL;

}

clapre = clap;

}

return S;

}

status AnswerFile(char \*filename) {//生成SAT问题的答案文件

FILE\* fp;

int p = 0;

while (filename[p] != 0) { p++; }

while (filename[p] != '.') { p--; }

filename[++p] = 'r';

filename[++p] = 'e';

filename[++p] = 's';

filename[++p] = 0;

fp = fopen(filename, "w");

if(ans.state){ fprintf(fp, "s 1\r\n"); }

else{ fprintf(fp, "s 0\r\n"); }

fprintf(fp, "v ");

for (int i = 1; i <= ans.num; i++) {

if (ans.LiteralState[i]) { fprintf(fp, "%5d ", i \* ans.LiteralState[i]); }

else { fprintf(fp, "%5d ", i); }

}

fprintf(fp, "\r\n");

fprintf(fp, "t %d\r\n", ans.time);

fclose(fp);

printf("SAT问题答案已保存至相应文件中\n");

return OK;

}

void TraverseCNF(cnf S) {//遍历生成的CNF结构并输出

printf("\n\n");

cla c = S->root;

lit l;

while (c) {

l = c->first\_l;

while (l) {

printf("%5d ", l->sta);

l = l->next;

}

printf("0\n");

c = c->next;

}

}

## DPLL.cpp

//DPLL.cpp：此文件是对于DPLL算法的实现

#include"def.h"

int count\_pos[MaxLitNum + 1], count\_neg[MaxLitNum + 1];

status LiteralExist(cla c, int l) {//判断子句c是否包含文字l

int flag = 0;

if (!c) { return 0; }

lit temp = c->first\_l;

while (temp) {

if (temp->sta == l) { flag = 1; break; }

else { temp = temp->next; }

}

return flag;

}

cnf DeleteLiteral(cnf S, int l) {//单子句规则化简S

//以下是删除包含l的子句

cla ctemp, cnext;

lit ltemp, lnext;

while (S->root && LiteralExist(S->root, l)) {

ctemp = S->root;

S->root = S->root->next;

ltemp = ctemp->first\_l;

while (ltemp) {

lnext = ltemp->next;

free(ltemp);

ltemp = lnext;

}

free(ctemp);

S->c\_num--;

}

ctemp = S->root;

while (ctemp) {

while (ctemp->next) {

if (LiteralExist(ctemp->next, l)) {

cnext = ctemp->next;

ltemp = cnext->first\_l;

while (ltemp) {

lnext = ltemp->next;

free(ltemp);

ltemp = lnext;

}

ctemp->next = cnext->next;

free(cnext);

S->c\_num--;

}

else { ctemp = ctemp->next; }

}

ctemp = ctemp->next;

}

//以下是删除子句中的-l(待修改)

ctemp = S->root;

while (ctemp) {

while (ctemp->first\_l && (ctemp->first\_l->sta + l == 0)) {

ltemp = ctemp->first\_l;

ctemp->first\_l = ctemp->first\_l->next;

free(ltemp);

}

ltemp = ctemp->first\_l;

while (ltemp) {

while (ltemp->next) {

if (ltemp->next->sta + l == 0) {

lnext = ltemp->next;

ltemp->next = lnext->next;

free(lnext);

}

else { ltemp = ltemp->next; }

}

ltemp = ltemp->next;

}

ctemp = ctemp->next;

}

S->l\_num--;

return S;

}

status EmptyClauseExist(cnf S) {//判断是否存在空子句

cla p = S->root;

while (p) {

if (!p->first\_l) { return OK; }

p = p->next;

}

return NOPE;

}

cla SingleClause(cnf S) {//判断是否存在单子句并返回第一个单子句

cla p = S->root;

lit q;

while (p) {

q = p->first\_l;

if (q) {

if (!q->next) { return p; }

}

p = p->next;

}

return NULL;

}

int SelectClause(cnf S) {//寻找S中出现次数最多的一个文字并返回它的编号

memset(count\_pos, 0, MaxLitNum + 1);

memset(count\_neg, 0, MaxLitNum + 1);

cla ctemp = S->root;

lit ltemp;

while (ctemp) {

ltemp = ctemp->first\_l;

while (ltemp) {

if (ltemp->sta > 0) { count\_pos[ltemp->sta]++; }

else { count\_neg[abs(ltemp->sta)]++; }

ltemp = ltemp->next;

}

ctemp = ctemp->next;

}

int maxi = 0, max = 0, flag=1;

for (int i = 1; i <= ans.num; i++) {

if (count\_pos[i] > max) {

max = count\_pos[i];

maxi = i;

flag = 1;

}

}

for (int i = 1; i <= ans.num; i++) {

if (count\_neg[i] > max) {

max = count\_neg[i];

maxi = i;

flag = 0;

}

}

return flag ? maxi : -maxi;

}

cnf Sew(cnf S, int i) {//将数据为i的单子句作为新cnf的根，后续复制S，返回新cnf

cnf news = (cnf)malloc(sizeof(CNFG));

cla v = (cla)malloc(sizeof(CNode));

lit u = (lit)malloc(sizeof(LNode));

v->first\_l = u; v->next = NULL;

u->sta = i; u->next = NULL;

news->root = v;

news->c\_num = S->c\_num + 1;

news->l\_num = S->l\_num;

cla stemp = S->root, newtemp = news->root, cnext;

lit slit, nlit, lnext;

while (stemp) {

cnext = (cla)malloc(sizeof(CNode));

newtemp->next = cnext;

slit = stemp->first\_l;

cnext->next = NULL;

if (slit) {

nlit = (lit)malloc(sizeof(LNode));

cnext->first\_l = nlit;

nlit->sta = slit->sta; nlit->next = NULL;

slit = slit->next;

while (slit) {

lnext = (lit)malloc(sizeof(LNode));

lnext->next = NULL; lnext->sta = slit->sta;

nlit->next = lnext;

nlit = lnext;

slit = slit->next;

}

}

else { cnext->first\_l = NULL; }

newtemp = newtemp->next;

stemp = stemp->next;

}

return news;

}

status FreeCNF(cnf S) {//销毁S

cnf cntem = S;

cla cltem, cnext;

lit ltem, lnext;

if (cntem) {

cltem = cntem->root;

if (cltem) {

cnext = cltem->next;

while (cltem && cnext) {

ltem = cltem->first\_l;

if (ltem) {

lnext = ltem->next;

while (ltem && lnext) {

free(ltem); ltem = lnext; lnext = lnext->next;

}

free(ltem);

}

free(cltem); cltem = cnext; cnext = cnext->next;

}

free(cltem);

}

free(cntem);

}

S = NULL;

return OK;

}

status DPLL(cnf S,int flag) {

int sc;

cla sin = SingleClause(S);

while (sin) {

ans.LiteralState[abs(sin->first\_l->sta)] = sin->first\_l->sta > 0 ? pos : neg;

S = DeleteLiteral(S, sin->first\_l->sta);

if (!S->root) {

FreeCNF(S);

return TRUE;

}

else if (EmptyClauseExist(S)) {

FreeCNF(S);

return FALSE;

}

sin = SingleClause(S);

}//end of while

if (flag == 1) { sc = S->root->first\_l->sta; }

else if (flag == 2) { sc = SelectClause(S); }

cnf temp;

temp = Sew(S, sc);

if (DPLL(temp,flag)) {

FreeCNF(S);

return TRUE;

}

//FreeTree(temp);

temp = (cnf)malloc(sizeof(CNFG));

temp = Sew(S, -sc);

FreeCNF(S);

return DPLL(temp,flag);

}

## Sudoku.cpp

//Sudoku.cpp：此文件是对数独的实现

#include"def.h"

#define MaxLit 729

#define CORRECT 0

#define WRONG -1

#define COL 9

#define ROW 9

#define tran(i,j,k) i\*81+j\*9+k

//int Digged[ROW][COL];

void PrintSudoku(int a[9][9]){//输出数独

int i, j;

printf("\n\n\t\t\t\t数独如下:\n");

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

for (i = 0; i < 9; i++) {

printf("\t\t\t");

for (j = 0; j < 9; j++) {

if (j % 3 == 0) { printf("|"); }

if (a[i][j]) { printf("%2d ", a[i][j]); }

else { printf(" \_ "); }

}

printf("|\n");

if ((i + 1) % 3 == 0) { printf("\t\t\t-------------------------------\n"); }

}

}

status JudgeNumber(int a[][COL], int row, int col, int n){//判断填入的数是否符合要求

int i, j;

// 判断行是否重复

for (i = 0; i < 9; i++) {

if (a[row][i] == n) { return FALSE; }

}

// 判断列是否重复

for (j = 0; j < 9; j++) {

if (a[j][col] == n) { return FALSE; }

}

// 判断所在小九宫格是否重复

int x = (row / 3) \* 3;

int y = (col / 3) \* 3;

for (i = x; i < x + 3; i++) {

for (j = y; j < y + 3; j++) {

if (a[i][j] == n) { return FALSE; }

}

}

return OK;

}

int Fill(int a[][COL], int i, int j) {//递归填充数独元素

if (i < ROW && j < COL) {

int x, y, k, flag = 0;

int temp[COL + 1] = { CORRECT };

for (x = 0; x < i; x++) { temp[a[x][j]] = WRONG; }

for (x = 0; x < j; x++) { temp[a[i][x]] = WRONG; }

for (x = i / 3 \* 3; x <= i; x++) {

if (x == i) {

for (y = j / 3 \* 3; y < j; y++) { temp[a[x][y]] = WRONG; }

}

else {

for (y = j / 3 \* 3; y < j / 3 \* 3 + 3; y++) { temp[a[x][y]] = WRONG; }

}

}

for (k = 1; k <= COL && flag == 0; k++) {

if (temp[k] == CORRECT) {

flag = 1;

a[i][j] = k;

if (j == COL - 1 && i != ROW - 1) {

if (Fill(a, i + 1, 0) == CORRECT) { return CORRECT; }

else { flag = 0; }

}

else if (j != COL - 1) {

if (Fill(a, i, j + 1) == CORRECT) { return CORRECT; }

else { flag = 0; }

}

}

}

if (!flag) {

a[i][j] = 0;

return WRONG;

}

}

return CORRECT;

}

void CreateSudoku(int a[][COL]) { //生成数独

//随机生成第一行

int i, j;

srand((unsigned)time(nullptr));

for (i = 0; i < COL; i++) {

a[0][i] = rand() % 9 + 1;

j = 0;

while (j < i) {

if (a[0][i] == a[0][j]) {

a[0][i] = rand() % 9 + 1;

j = 0;

}

else j++;

}

}

//递归生成后i行

Fill(a, 1, 0);

}

cnf AddSew(cnf s, int i) {//将i作为单子句插入到S中

cla c = (cla)malloc(sizeof(CNode));

lit l = (lit)malloc(sizeof(LNode));

l->sta = i; l->next = NULL;

c->first\_l = l;

c->next = s->root;

s->root = c;

s->c\_num++;

return s;

}

cnf SudokuToCNF(int a[][COL]) {//数独数组转换为cnf

FILE\* fp;

cnf c;

fp = fopen("D:\SudokuRule.cnf", "r");

c = (cnf)malloc(sizeof(CNFG));

c->root = NULL;

OperateCNF(fp, c);

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

for (int j = 0; j < COL; j++) {

if (a[i][j]) { c = AddSew(c, tran(i, j, a[i][j])); }

}

}

return c;

}

status CertainAnwser(int a[][COL]) {//判断挖洞后的数独是否有解

cnf s = SudokuToCNF(a);

return DPLL(s, 1);

}

status WhetherToDig(int a[][COL], int n, int m, int k) {//判断挖空后是否有唯一解

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

if (i != k && JudgeNumber(a, n, m, i)) {

a[n][m] = i;

if (CertainAnwser(a)) {

a[n][m] = k;

return NOPE;

}

}

}

a[n][m] = 0;

return OK;

}

status DigHole(int a[][COL],int n) {

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

if (!n) { return OK; }

if (WhetherToDig(a, i, j, a[i][j])) { n--; }

}

}

if (n) { return NOPE; }

}