

**课程设计报告**

**题目：基于SAT的双数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： 计算机本硕博**

**学 号： U202115666**

**姓 名： 刘文博**

**指导教师： 向文**

**报告日期： 2022.10.7**

**计算机科学与技术学院**

**任务书**

* **设计内容**

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

* **设计要求**

要求具有如下功能：

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

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

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

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

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

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

* **参考文献**

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

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

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

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

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

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

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

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

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

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

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

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

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

**目录**

**任务书** I

**参考文献** II

**1.引言** 1

1.1课题背景与意义 1

1.2国内外研究现状 2

1.3课程设计的主要研究工作 2

**2.系统需求分析与总体设计** 3

2.1系统需求分析 3

2.2系统总体设计 3

**3.系统详细设计** 4

3.1有关数据结构的定义 4

3.2主要算法设计 5

**4.系统实现与测试** 7

4.1系统实现 7

4.2系统测试 9

**5.总结与展望** 13

5.1全文总结 13

5.2工作展望 13

**6.体会** 14

**附录** 15

**1引言**

**1.1课题背景与意义**

**1.1.1课题背景**

SAT问题又称命题逻辑公式的可满足性问题（satisfiability problem），是判断对合取范式形式给出的命题逻辑公式是否存在一个真值指派使得该逻辑公式为真。SAT问题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。看似简单，却可广泛应用于许多实际问题如人工智能、电子设计自动化、自动化推理、硬件设计、安全协议验证等，具有重要理论意义与应用价值。

DPLL算法是基于树/二叉树的回溯搜索算法，主要使用两种基本处理策略：

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

单子句传播策略就是反复利用单子句规则化简S的过程。

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

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

**1.1.2课题意义**

SAT问题是第一个被证明的NP完全问题，而NP完全问题由于其极大的理论价值和困难程度，破解后将会在许多领域得到广泛应用，从而在计算复杂性理论中具有非常重要的地位。由于所有的NP完全问题都能够在多项式时间内进行转换，那么如果SAT问题能够得到高效解决，所有的NP完全问题都能够在多项式时间内得到解决。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。SAT问题的应用领域非常广泛，还能用于解决数学研究和应用领域中的旅行商问题和逻辑算数问题。许多实际问题，例如数据库检索、积木世界规划、超大规模集成电路设计、人工智能等都可以转换成SAT问题进而进行求解。可见对SAT问题求解的研究，具有重大意义。

**1.2国内外研究现状**

对于SAT问题的研究从没有停止过，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。对SAT问题的求解主要有完备算法和不完备算法两大类。不完备算法主要是局部搜索算法，这种算法不能保证一定找到解，但是求解速度快，对于某些SAT问题的求解，局部搜索算法要比很多完备算法更有效。完备算法出现的时间更早，优点是可以正确判断SAT问题的可满足性，在算例无解的情况下可以给出完备的证明。对于求解SAT问题的优化算法主要有启发式算法、冲突子句学习算法、双文字监视法等。

**1.3课程设计的主要研究工作**

1. 读入cnf文件，通过学习DPLL算法，能够对中小规模的算例进行求解，将结果保存在ref文件中并计算求解所用时间。
2. 将双数独问题抽象成相应的SAT问题，转化为cnf文件，将DPLL算法应用于双数独游戏中，满足用户需求。
3. 深入学习与理解SAT问题的特点和DPLL算法的设计与优化，撰写实验报告。**2系统需求分析与总体设计**

**2.1系统需求分析**

系统应能够读入cnf文件，对相应的SAT问题进行求解，如果可满足应将结果保存在res文件中，并计算求解所需时间。系统还应能够调用SAT求解模块根据挖洞法生成一个2\*9\*9的双数独并以合适的形式输出棋盘，用户可以根据意愿进行填写，或要求随机给出提示，棋盘应在用户做出操作后更新，当数独游戏结束时，应能输出数独的正确答案。

**2.2系统总体设计**

系统总体由五个模块组成，分别是包含调用库、函数原型、宏定义的define模块 、 能够读入cnf文件并建立相应数据结构的cnfparser模块 、 核心的使用DPLL算法的SAT问题求解器solver模块 、 将数独规则转化为相应cnf文件并调用其他模块求解以生成双数独的sudoku模块 以及 包含main函数的提供用户友好型菜单界面的display模块。（如图2-1所示）

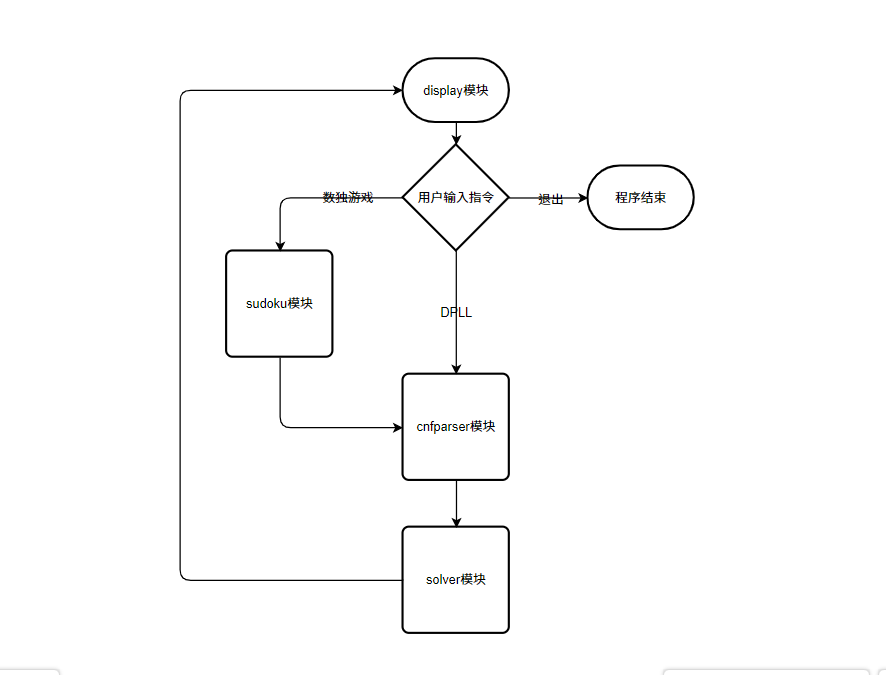


图2-1系统总体设计图

**3系统详细设计**

**3.1有关数据结构的定义**

本程序研究的对象是一个合取范式，包含若干子句，子句中又包含若干文字，本程序所使用的数据结构为十字链表。

其中最基本的节点为node,表示SAT命题中的文字，value保存文字的真值，next指向下一个文字。

clause表示子句，其中num表示子句中文字即node的个数，用head指向第一个文字，next指向下一子句。

sudproblem则是一个三维数组，用来保存双数独，original用来保存挖洞后的双数独，pattern用来保存未挖洞的双数独，num记录已填入数字的格子的数量。（如图3-1所示）

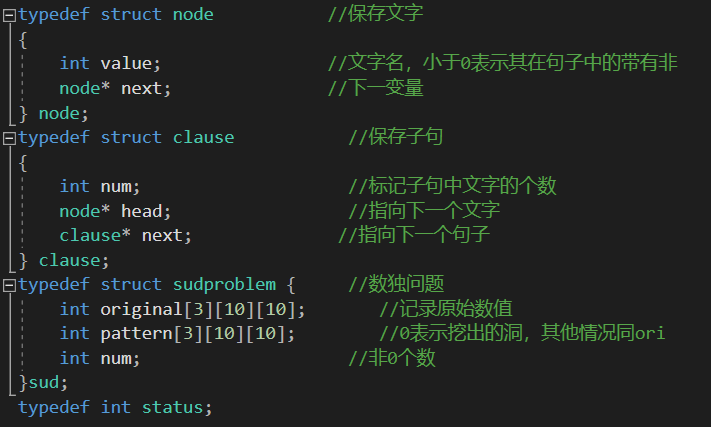


图3-1 数据结构定义

数据结构的逻辑关系如图3-2所示



图3-2 数据项之间的联系

**3.2主要算法设计**

solver模块：首先根据单子句规则进行删除，直至找不到单子句，调用启发函数找到一个变元，先复制当前的数据，然后加入对上述变元赋值的单子句进行下一次DPLL过程，返回值为true则求解完毕，返回至为false则在已复制的数据的基础上加入相反的单子句进行DPLL过程。（如图3-3所示）

sudoku模块：首先随机选择一个位置进行复制，接着根据双数独的规则生成包含1458个变元，24138个子句的cnf文件，之后调用solver模块求解该cnf文件对应的SAT问题，得到的解即为求解初始只有一个元素的双数独的解。将结果保存在pattern数组中，在其副本original基础上进行挖洞，在要挖洞的位置上试填其他数字，并用同上的方法求解双数独，如果可满足，则不能在此处挖洞，否则数独游戏将没有唯一性，反之可以挖洞，执行若干次后得到有唯一解的双数独棋盘。（如图3-4所示）

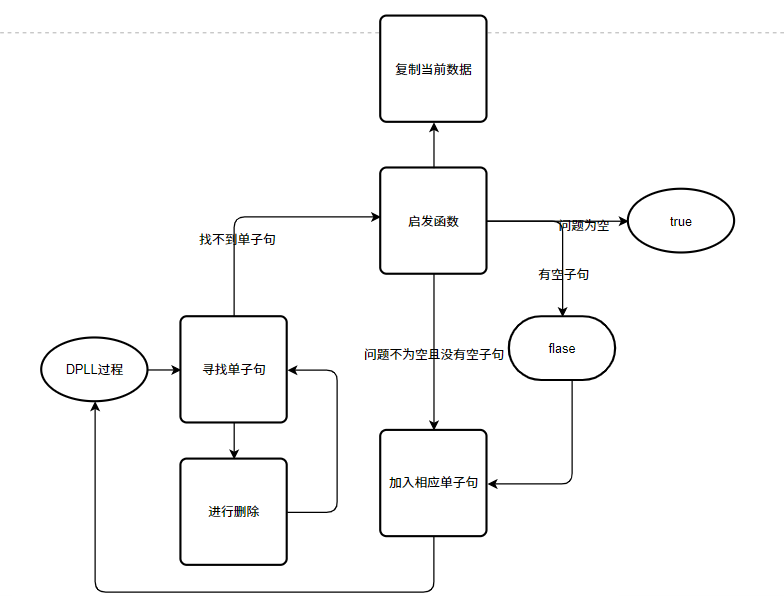


图3-3 solver模块主要算法流程图

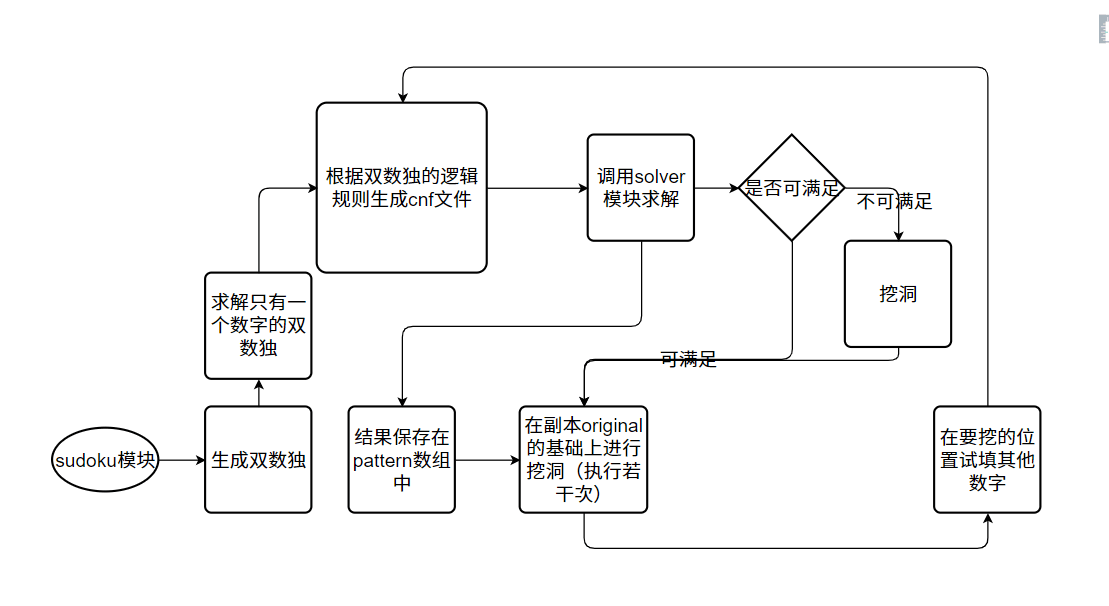


图3-4 sudoku模块流程图

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

cnfparser模块：读入cnf文件并依次建立相应数据结构。

define模块：提供必要的调用库和函数原型声明以及宏定义。

**4系统实现与测试**

**4.1系统实现**

系统的软件环境：运行环境Windows11，开发环境Visual Studio Code

系统的硬件环境：CPU为11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz

GPU为NVIDIA GeForce RTX 3050 Laptop GPU

系统所包含的函数及相应说明参见图4-1。



图4-1 各函数及其说明

sudoku模块中各函数的作用及调用关系如图4-2所示

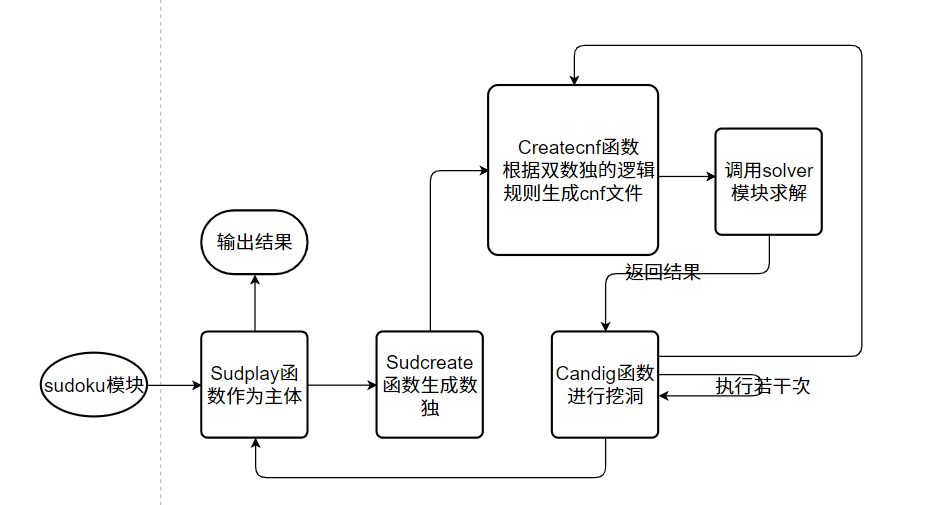


图4-2 sudoku模块各函数功能及调用关系图

solver模块中各函数的作用及调用关系如图4-3所示

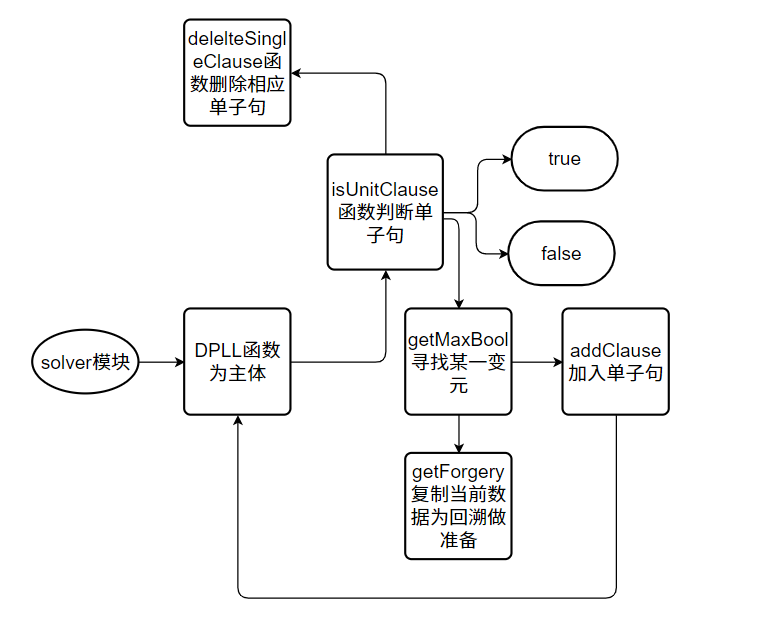


图4-3 solver模块各函数功能及调用关系图

**4.2系统测试**

4.2.1常用软件测试方法

分模块进行测试，先验证各模块功能是否正常，再对模块进行性能测试。

4.2.2各模块功能及设计目标

solver模块：通过DPLL算法设计SAT问题求解器，实现对中小规模的cnf算例的求解并记录求解所需时间，若满足则将解保存到相应的res文件中。

sudoku模块：生成具有唯一解的双数独棋盘，通过输出提示与用户交互，跟据用户输入进行数独的填写，提示，输出答案等操作，能够及时更新数独棋盘并实现数独难度可选择。

4.2.3测试大纲

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

4.2.4运行结果

先进行读取、打印cnf文件以及保存结果的功能测试，运行结果如图4-4、4-5、4-6所示

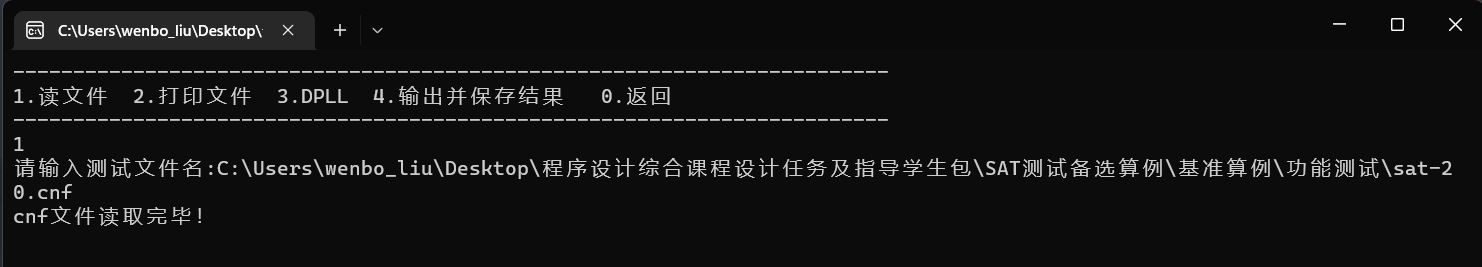


图4-4 读取cnf文件功能测试

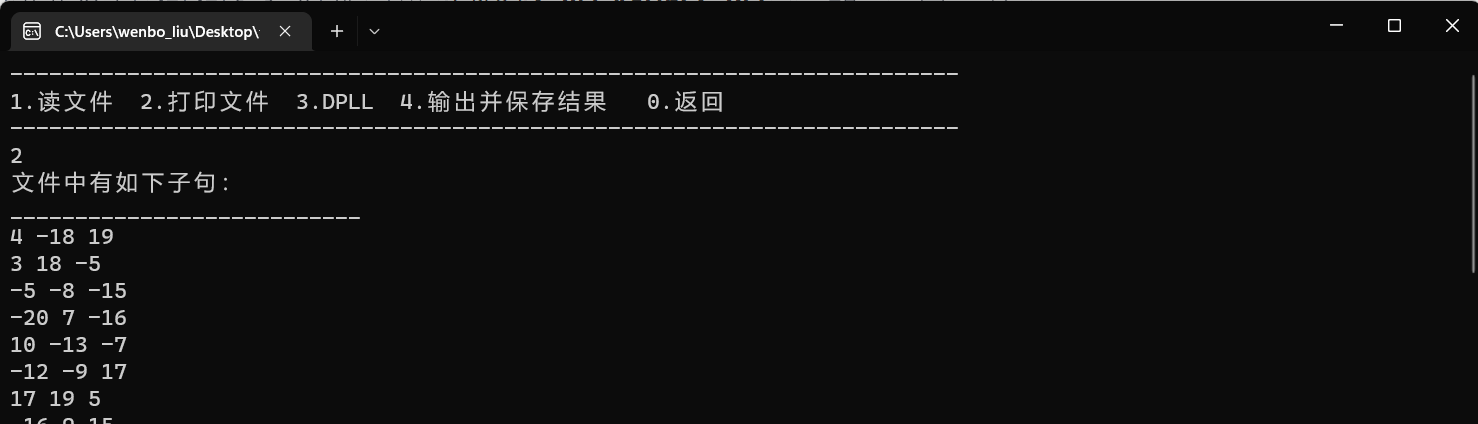


图4-5 打印cnf文件功能测试

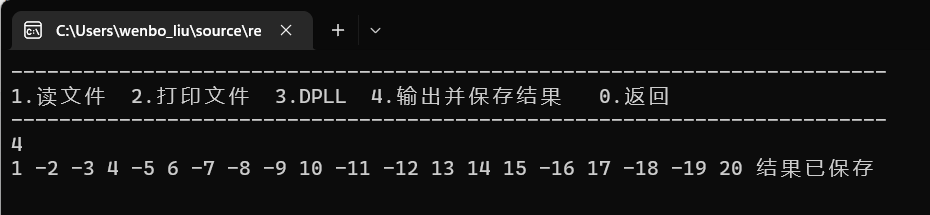


图4-6 输出并保存结果的功能测试

再进行DPLL功能测试，分别选择满足算例和不满足算例进行测试，运行结果如图4-7、4-8所示

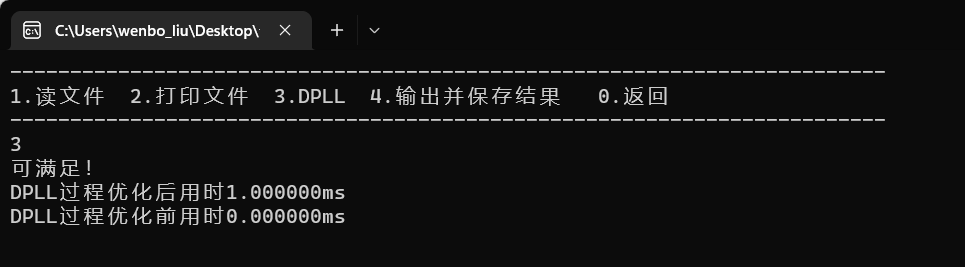


图4-7 可满足算例DPLL功能测试

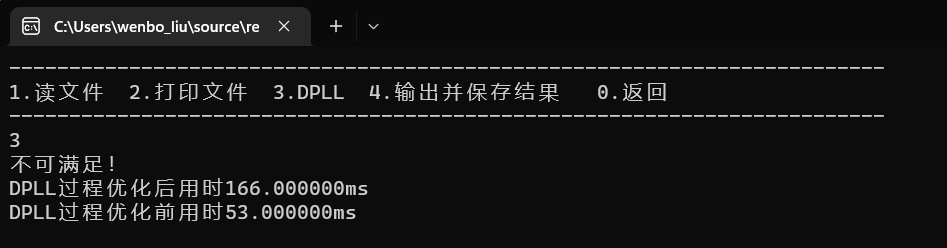


图4-8 不可满足算例DPLL功能测试

之后进行DPLL过程性能测试，依次选取小、中、大规模算例，运行结果如图4-9、4-10、4-11所示

（7cnf20\_90000\_90000\_7.shuffled-20.cnf）

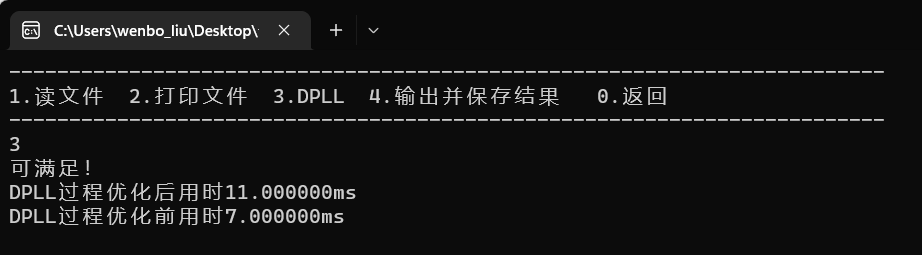


图4-9 小规模算例DPLL性能测试

（bart17.shuffled-231.cnf）

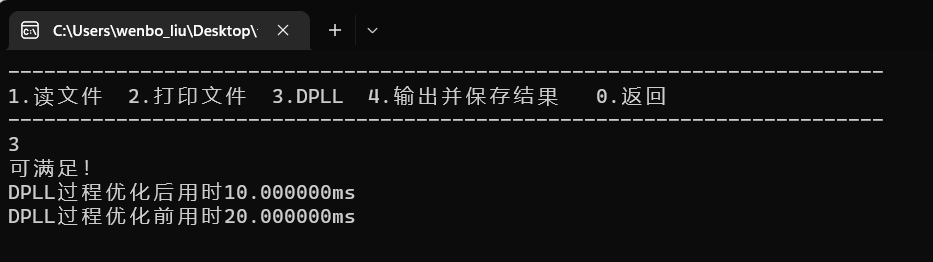


图4-10 中规模算例DPLL性能测试

（eh-dp04s04.shuffled-1075.cnf）



图4-11 大规模算例DPLL性能测试

最后进行双数独游戏的功能测试，检验数独游戏的生成、填写、输出答案的功能，运行结果如图4-12、4-13、4-14所示



图4-12 数独游戏生成功能测试

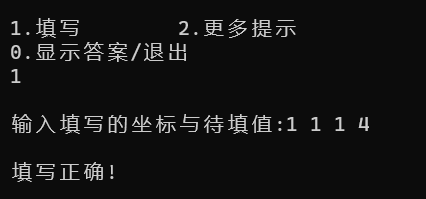


图4-13 数独游戏填写功能测试



图4-14 数独游戏输出答案功能测试

4.2.5运行结果分析

根据运行结果，该系统读入、打印cnf文件功能正常，能够对SAT问题进行求解且可以将结果保存在res文件中，通过进一步的性能测试，该系统能够在合适的时间内对小、中、大规模的算例验证其满足性并求解。在数独游戏中，根据运行结果，该系统能够生成具有唯一解的数独棋盘，可以让用户进行填写或者输出答案，棋盘可以相应更新。综上所述，该系统各功能正常完备，具有较好的性能，完成了相应的设计目标。

**5总结与展望**

**5.1全文总结**

本次实验的主要工作如下：

（1）读取cnf文件，建立相应的数据结构；

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

（3）将双数独游戏的规则转换为SAT问题进行求解，生成双数独棋盘。

**5.2工作展望**

在今后的研究中，将围绕着如下几个方面开展工作

（1）优化DPLL算法中的启发函数，使之能够在合理的时间内化简大算例。

（2）加强双数独棋盘的图形化，使之更容易被用户接受。

（3）研究针对数独棋盘生成的DPLL优化算法，减少生成棋盘所需的时间。

# 6体会

这次实验对于我们来说是一次不小的挑战，一开始面对设计一个复杂系统的要求感到无所适从，但真正动手做起来才能明白，系统都由一个个模块组成，理清它们之间的调用关系，在让编程大大简化的同时还可以让系统更加有条理。在以后的学习中，我们要建立起模块化的思想，面对复杂问题要敢于实践。同时，当算法面对大规模数据性能低下时，应多做相关的研究与思考，在原有算法的基础上多尝试几种优化方法，只有这样才能满足各种各样的编程需要。总而言之，这次课程设计，我受益良多。

**附录**

**define.h**

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define OK 1

#define ERROR 0

#define SUDOKU 1

#define SAT 0

typedef struct node //保存文字

{

int value; //文字名，小于0表示其在句子中的带有非

node\* next; //下一变量

} node;

typedef struct clause //保存子句

{

int num; //标记子句中文字的个数

node\* head; //指向下一个文字

clause\* next; //指向下一个句子

} clause;

typedef struct sudproblem { //数独问题

int original[3][10][10]; //记录原始数值

int pattern[3][10][10]; //0表示挖出的洞，其他情况同ori

int num; //非0个数

}sud;

typedef int status;

int ABS(int a); //辅助计算函数

int read(); //快读

status ReadCNF(clause\*& cnf, int satORsudoku); //读取cnf文件

void destroyClause(clause\*& cnf); //删除SAT问题

void removeClause(clause\*& cnf, clause\*& cl); //删除子句

void removeNode(clause\* cl, node\*& nd); //删除文字

status addClause(clause\* cl, clause\*& cnf); //加入单子句

bool isUnitClause(clause\* cl); //子句判断是否为单子句

void delelteSingleClause(clause\* s, clause\*& cnf); //单子句规则

bool emptyClause(clause\* cnf); //子句判空

void getForgery(clause\*& forgery, clause\* cnf); //获取副本

int getMaxBool(clause\* cnf); //选取出现次数最多的变元

int getMaxBool\_2(clause\* cnf); //选取总出现次数最多的变元

int getMaxBool\_Sudoku(clause\* cnf); //针对数独的选择出现次数最多的变元的算法

int getNextBool\_3(clause\* cnf); //选取在短子句总出现次数最多的变元

int getNextBool\_4(clause\* cnf); //选取在短子句出现次数最多的变元

bool DPLL(clause\*& cnf, int\* v, int satORsudoku); //DPLL算法

status printCNF(int s, int\* v, double t); //创建res文件

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

//输入变量名i行j列填入数字n

void SudPlay(); //可玩数独

void Sudcreate(); //生成一个可以玩的数独

status Candig( int x, int y, int z); //判断是否为可以挖洞

void hint(); //给出提示

void CreateCnf(); //生成cnf文件

**cnfparser.cpp**

#include"define.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#pragma warning(disable:6031)

const int inf = 1061109567;

extern int boolCnt, clauseCnt, sumOfAns, branchRule;

extern char fileName[100];

extern clause\* CNF ;

int ABS(int a)

{

return a > 0 ? a : -a;

}

int read(FILE\* fp)

{

int flag = 0;

int x = 0, f = 1;

char c;

c = getc(fp); //读入字符

while (c < '0' || c > '9') //非数字

{

if (c == '-')

f = -1, c = getc(fp);

else

return inf;

}

while (c >= '0' && c <= '9') //读入数字

{

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

c = getc(fp);

if (c == -1) //文件结束

flag = 1;

}

//跳出循环时c应为空格 换行 或者EOF

if (!flag && c != ' ' && c != '\n')

return inf;

return x \* f; //返回读入数字的值

}

status ReadCNF(clause\*& cnf, int satORsudoku)

{

FILE\* fp;

char ch;

node\* pn;

clause\* pc;

char check[5] = { ' ', 'c', 'n', 'f', ' ' };

if (satORsudoku == SAT)

fp = fopen(fileName, "r");

else

fp = fopen("sud.cnf", "r");

if (!fp)

return ERROR;

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

while ((ch = getc(fp)) != '\n') //当行首为c时读到换行符

continue;

//跳出循环时ch应为‘p’

if (ch != 'p')

return ERROR;

for (int i = 0; i < 5; i++)

if ((ch = getc(fp)) != check[i]) //读取‘p cnf ’部分

return ERROR;

if ((boolCnt = read(fp)) == inf || (clauseCnt = read(fp)) == inf)//读入变元数和子句数

return ERROR;

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

cnf->next = NULL;

cnf->head = (node\*)malloc(sizeof(node));

cnf->head->next = NULL;

cnf->num = 0;

pc = cnf; //pc是一个指向子句的指针

pn = cnf->head; //pn是一个指向文字的指针

//初始化问题

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

{

int dat;

if ((dat = read(fp)) == inf)

return ERROR;

while (dat)//dat不是终止符0时，循环读取后面的数据

{

pc->num++;

pn->value = dat;

pn->next = (node\*)malloc(sizeof(node));

if ((dat = read(fp)) == inf)

return ERROR;

if (!dat)

pn->next = NULL;

pn = pn->next;

}

pc->next = (clause\*)malloc(sizeof(clause));

pc->next->num = 0;

pc->next->head = (node\*)malloc(sizeof(node));

if (i == clauseCnt)

{

pc->next = NULL;

break;

}

pc = pc->next;

pn = pc->head;

}

//读取cnf文件

fclose(fp);

return OK;

}

status printCNF(int s, int\* v, double t)

{

FILE\* fp;

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

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

{

fileName[i + 1] = 'r', fileName[i + 2] = 'e', fileName[i + 3] = 's';

break;

}

if (!(fp = fopen(fileName, "w+")))//创建res文件

{

printf("文件创建失败!\n");

return ERROR;

}

fprintf(fp, "s %d\nv ", s);

if (s)//如果可满足

for (int i = 1; i <= boolCnt; i++)//写入结果

{

if (v[i])

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

else

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

}

fprintf(fp, "\nt %lf", t \* 1000);

fclose(fp);

return OK;

}

**solver.cpp**

#include"define.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#pragma warning(disable:6031)

const int inf = 1061109567;

extern int boolCnt, clauseCnt, sumOfAns, branchRule, \* ans;

extern clause\* CNF;

void destroyClause(clause\*& cnf)

{

clause\* ppc, \* pc2; //指向子句的指针

node\* pn1, \* pn2; //指向文字的指针

ppc = cnf;

while (ppc)

{

pn1 = ppc->head; //依次删除子句

while (pn1)

pn2 = pn1->next, free(pn1), pn1 = pn2; //依次删除文字

pc2 = ppc->next, free(ppc), ppc = pc2;

}

cnf = NULL;

}

void removeClause(clause\*& cnf, clause\*& cl)

{

if (cl == cnf) //如果是头子句

cnf = cnf->next; //头子句的next变为头子句

else

{

clause\* ppc = cnf;

while (ppc && ppc->next != cl)

ppc = ppc->next; //定位要删除的子句

ppc->next = ppc->next->next;

}

node\* pn1, \* pn2;

for (pn1 = cl->head; pn1;)

pn2 = pn1->next, free(pn1), pn1 = pn2; //删除文字

free(cl);

cl = NULL;

}

void removeNode(node\*& head, node\*& nd)

{

node\* ppn = head;

if (ppn == nd) //如果要删除头节点

head = head->next; //头节点的next成为头节点

else

{

while (ppn && ppn->next != nd)

ppn = ppn->next; //定位节点

ppn->next = ppn->next->next;

}

free(nd);

nd = NULL;

}

status addClause(clause\* cl, clause\*& cnf)

{

if (cl)//首插法插入单子句

{

cl->next = cnf;

cnf = cl;

return OK;

}

return ERROR;

}

bool isUnitClause(clause\* cl)

{

if (cl->head != NULL) {

if (cl->head->next == NULL)

return true;

}

return false;

}

void delelteSingleClause(clause\* s, clause\*& cnf)

{

clause\* tmp;

int n = s->head->value;//要删除的变元的值

for (clause\* ppc = cnf; ppc; ppc = tmp) //遍历所有子句

{

tmp = ppc->next;//用temp存储当前子句的next 当前子句被删除后可以找到下一子句

for (node\* ppn = ppc->head; ppn; ppn = ppn->next)

{

if (ppn->value == n)

{

removeClause(cnf, ppc);//如果包含该变元 删除该子句

break;

}

if (ppn->value == -n)

{

removeNode(ppc->head, ppn);//如果包含该变元的反 删除该节点

ppc->num--;

break;

}

}

}

}

bool emptyClause(clause\* cnf)

{

for (clause\* p = cnf; p; p = p->next)

if (!p->head)

return true;

return false;

}

void getForgery(clause\*& forgery, clause\* cnf)

{

clause\* ppc, \* pc; //指向子句的指针

node\* ppn, \* pn; //指向文字的指针

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

forgery->head = (node\*)malloc(sizeof(node));

forgery->next = NULL;

forgery->head->next = NULL;

forgery->num = 0;

//初始化问题

for (pc = cnf, ppc = forgery; pc != NULL; pc = pc->next, ppc = ppc->next)//复制当前cnf 为回溯做准备

{

for (pn = pc->head, ppn = ppc->head; pn != NULL; pn = pn->next, ppn = ppn->next)

{

ppc->num++;

ppn->value = pn->value;

ppn->next = (node\*)malloc(sizeof(node));

ppn->next->next = NULL;

if (pn->next == NULL)

free(ppn->next), ppn->next = NULL;

}

ppc->next = (clause\*)malloc(sizeof(clause));

ppc->next->head = (node\*)malloc(sizeof(node));

ppc->next->next = NULL;

ppc->next->head->next = NULL;

ppc->next->num = 0;

if (pc->next == NULL)

free(ppc->next->head), free(ppc->next), ppc->next = NULL;

}

}

int getMaxBool(clause\* cnf) //选择出现次数最多的变元

{

int\* cnt = (int\*)malloc(sizeof(int) \* (boolCnt \* 2 + 1));

for (int i = 0; i <= boolCnt \* 2; i++) //初始化cnt数组

cnt[i] = 0;

for (clause\* pc = cnf; pc; pc = pc->next)

for (node\* pn = pc->head; pn; pn = pn->next)

{

if (pn->value > 0)

cnt[pn->value]++;

else

cnt[boolCnt - pn->value]++;

}

int maxBool, maxTimes = 0;

for (int i = 1; i <= boolCnt; i++) //选择变元的正面

if (cnt[i] > maxTimes)

maxTimes = cnt[i], maxBool = i;

if (maxTimes == 0) //只剩下变元的反

{

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

if (cnt[i] > maxTimes)

maxTimes = cnt[i], maxBool = boolCnt - i;

}

free(cnt);

return maxBool;

}

int getMaxBool\_Sudoku(clause\* cnf) //对于变元数已知的数独问题 不再动态分配cnt数组的存储空间

{

int cnt[2917] = { 0 };

for (clause\* pc = cnf; pc; pc = pc->next)

for (node\* pn = pc->head; pn; pn = pn->next)

{

if (pn->value > 0)

cnt[pn->value]++;

else

cnt[1458 - pn->value]++;

}

int maxBool, maxTimes = 0;

for (int i = 1; i <= 1458; i++) //先找变元的反面

if (cnt[i + 1458] > maxTimes)

maxTimes = cnt[i], maxBool = -i;

if (maxTimes == 0) //只剩下变元的正面

{

for (int i = 1; i <= 1458; i++)

if (cnt[i] > maxTimes)

maxTimes = cnt[i], maxBool = i;

}

return maxBool;

}

int getMaxBool\_2(clause\* cnf) //选取总出现次数最多的变元

{

int minNode = boolCnt;

int\* cnt = (int\*)malloc((boolCnt + 1) \* sizeof(int));

for (int i = 0; i < boolCnt; i++)

cnt[i] = 0;

for (clause\* pc = cnf; pc; pc = pc->next)

if (pc->num < minNode)

minNode = pc->num;

for (clause\* pc = cnf; pc; pc = pc->next)

if (pc->num == minNode)

for (node\* pn = pc->head; pn; pn = pn->next)

cnt[ABS(pn->value)]++;

int maxAppear = 0, maxBool;

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

if (cnt[i] > maxAppear)

maxAppear = cnt[i], maxBool = i;

free(cnt);

return maxBool;

}

double J(int n)

{

return pow(2.0, (double)(-n));

}

int getNextBool\_3(clause\* cnf) //选取在短子句总出现次数最多的变元

{

double\* weight = (double\*)malloc(sizeof(double) \* (boolCnt + 1));

for (int i = 0; i <= boolCnt; i++)

weight[i] = 0.0;

for (clause\* pc = cnf; pc; pc = pc->next)

for (node\* pn = pc->head; pn; pn = pn->next)

weight[ABS(pn->value)] += J(pc->num);

double maxWeight = 0;

int maxBool=0;

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

if (weight[i] > maxWeight)

maxWeight = weight[i], maxBool = i;

free(weight);

return maxBool;

}

int getNextBool\_4(clause\* cnf) //选取在短子句出现次数最多的变元

{

double\* weight = (double\*)malloc(sizeof(double) \* (boolCnt \* 2 + 1));

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

weight[i] = 0.0;

for (clause\* pc = cnf; pc; pc = pc->next)

for (node\* pn = pc->head; pn; pn = pn->next)

{

if (pn->value > 0)

weight[pn->value] += J(pc->num);

else

weight[boolCnt - pn->value] += J(pc->num);

}

double maxWeight = 0.0;

int maxBool=0;

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

if (weight[i] + weight[i + boolCnt] > maxWeight)

{

maxWeight = weight[i] + weight[i + boolCnt], maxBool = i;

}

if (weight[maxBool] < weight[maxBool + boolCnt])

maxBool = -maxBool;

free(weight);

return maxBool;

}

bool DPLL(clause\*& cnf, int\* v, int satORsudoku)

{

int flag = 1;

clause\* pc;

while (flag)//能找到单子句

{

flag = 0;

pc = cnf;

while (pc && !isUnitClause(pc)) //遍历寻找单子句

pc = pc->next;

if (pc != NULL)//找到单子句pc

{

if (pc->head->value > 0) //在结果数组中保存pc的值

v[pc->head->value] = 1;

else

v[-pc->head->value] = 0;

delelteSingleClause(pc, cnf); //根据单子句规则化简

if (cnf == NULL) //子句已完全化简

return true;

else if (emptyClause(cnf)) //存在空子句

return false;

flag = 1; //找到了单子句

}

}

int maxBool;

if (satORsudoku == SUDOKU)

maxBool = getMaxBool\_Sudoku(cnf);

else

{

if (branchRule == 1)

maxBool = getMaxBool(cnf);

else if (branchRule == 2)

maxBool = getMaxBool\_2(cnf);

else if (branchRule == 3)

maxBool = getNextBool\_3(cnf);

else if (branchRule == 4)

maxBool = getNextBool\_4(cnf);

}

clause\* newSingleClause = (clause\*)malloc(sizeof(clause)), \* forgery;

newSingleClause->head = (node\*)malloc(sizeof(node));

newSingleClause->next = NULL;

newSingleClause->head->value = maxBool;

newSingleClause->head->next = NULL;

newSingleClause->num = 1;

getForgery(forgery, cnf); //复制当前cnf 为回溯做准备

addClause(newSingleClause, forgery); //加入单子句

if (DPLL(forgery, v, satORsudoku) == true)

return true;

//如果结果为false

destroyClause(forgery); //删除副本

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

newSingleClause->head = (node\*)malloc(sizeof(node));

newSingleClause->next = NULL;

newSingleClause->head->value = -maxBool;

newSingleClause->head->next = NULL;

newSingleClause->num = 1;

addClause(newSingleClause, cnf); //加入相反的单子句

bool ans = DPLL(cnf, v, satORsudoku); //DPLL

destroyClause(cnf); //删除副本

return ans; //返回ans

}

**suduku.cpp**

#include"define.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#pragma warning(disable:6031)

extern int boolCnt, clauseCnt, sumOfAns, branchRule;

extern int \* ans ;

extern char fileName[100];

extern clause\* CNF ;

extern sud S;

int VarTrans(int f, int i, int j, int n) {

return (f - 1) \* 729 + (i - 1) \* 81 + (j - 1) \* 9 + n;

}

status Candig( int x, int y, int z) //修改都在original的基础上

{

if (S.original[x][y][z] == 0) return ERROR;

for (int i = 1; i <= 9; i++) //试填除原来数字之外的8个数字，若都不能满足说明可以挖洞

{

if (i == S.pattern[x][y][z]) continue;//和原数字相同，则比较下一个数字

S.original[x][y][z] = i;//给空格处填入一个数字，检查其是否有解

CreateCnf();

ReadCNF(CNF, SUDOKU);

ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

for (int i = 0; i <= boolCnt; i++)

ans[i] = 1;

int res=DPLL(CNF, ans, SUDOKU);

if (!res) continue;

else

{//若能满足，说明不能挖洞，给数独恢复其值

S.original[x][y][z] = S.pattern[x][y][z];

return ERROR;

}

}

S.original[x][y][z] = 0;//可以挖洞

S.num--;

return OK;

}

void hint() {

int x, y, z, flag;

flag = 1;

srand(time(NULL));

while (flag) {

x = rand() % 2 + 1;

y = rand() % 9 + 1;

z = rand() % 9 + 1;

if (!S.original[x][y][z]) {

S.original[x][y][z] = S.pattern[x][y][z];

flag = 0;

}

}

}

void CreateCnf()

{

FILE\* fp = fopen("sud.cnf", "w");

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

fprintf(fp, "p cnf %d %d\n", 1458, 24138 + S.num \* 9);//双数独 fijn==2\*9\*9\*9=1458 基本要求 162个格子\*（1+36） 行约束 18行\*（9+36\*9） 列约束同理 盒子约束 18个盒子\*（9+36\*9）等价约束 18\*9=162

int x, y, z, n;

int f, i, j, count;

{//将数独游戏所给提示数对应的单子句集录入cnf文件

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

{

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

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

{

if (S.original[f][i][j]) {

fprintf(fp, "%d 0\n", VarTrans(f, i, j, S.original[f][i][j]));

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

{

if (x == S.original[f][i][j]) continue;

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

}

}

}

}

}

}

{//将基础规则录入cnf文件

for (f = 1; f <= 2; f++) {

for (i = 1; i <= 9; i++)//对于9行，每一行有(9\*C92+9)个子句

{

for (n = 1; n <= 9; n++)//111或121或131……n为填入的数字，9次循环，每循环一次填入一个句子

{

for (j = 1; j <= 9; j++)//j为列号，

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

fprintf(fp, "0\n");

}

for (n = 1; n <= 9; n++)//n为填入的数字，9次循环

for (count = 1; count <= 9; count++)//这里循环输入非111或非121，等，共C92个句子

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

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

}

}

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

{

for (j = 1; j <= 9; j++)//对于9列，每一列有(9\*C92+9)个子句

{

for (n = 1; n <= 9; n++)//111或211或311……n为填入的数字，9次循环，每循环一次填入一个句子

{

for (i = 1; i <= 9; i++)//i为行号

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

fprintf(fp, "0\n");

}

for (n = 1; n <= 9; n++)//n为填入的数字，9次循环

for (count = 1; count <= 9; count++)//这里循环输入非111或非211，等，共C92个句子

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

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

}

}

for (f = 1; f <= 2; f++) {

for (count = 0; count < 9; count++)//对于9个九宫格，每个格子有(9\*C92+9)个子句

{

x = 1 + d[count][0] \* 3;

y = 1 + d[count][1] \* 3;//每个宫格左上角格子的位置

for (n = 1; n <= 9; n++)//n为填入的数字

{

for (int m = 0; m < 9; m++)

fprintf(fp, "%d ", VarTrans(f, x + d[m][0], y + d[m][1], n));

fprintf(fp, "0\n");

}

for (n = 1; n <= 9; n++)//n为填入的数字，9次循环

for (int m1 = 0; m1 < 9; m1++)//控制格子中的9个位置，共C92个句子

for (int m2 = m1 + 1; m2 < 9; m2++)

fprintf(fp, "%d %d 0\n", -VarTrans(f, x + d[m1][0], y + d[m1][1], n), -VarTrans(f, x + d[m2][0], y + d[m2][1], n));

}

}

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

{

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

for (j = 1; j <= 9; j++)//对于81个格子，每个格子有(C92+1)个句子

{

for (n = 1; n <= 9; n++)//n为填入的数字

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

fprintf(fp, "0\n");

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

for (count = n + 1; count <= 9; count++)

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

}

}

}

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

{

for (i = 7; i <= 9; i++)

{

for (j = 7; j <= 9; j++)

{

fprintf(fp, "%d %d 0\n", -VarTrans(1, i, j, n), VarTrans(2, i - 6, j - 6, n));

fprintf(fp, "%d %d 0\n", VarTrans(1, i, j, n), -VarTrans(2, i - 6, j - 6, n));

}

}

}

}

//将规则录入cnf文件

fclose(fp);

}

void Sudcreate() {

int x, y, z, v;

int f, i, j, n;

srand(time(NULL));

x = rand() % 2 + 1;

y = rand() % 9 + 1;//随机确定数独一个位置的数值

z = rand() % 9 + 1;

v = rand() % 9 + 1;

for (int f = 0; f <= 2; f++)

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

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

S.original[f][i][j] = S.pattern[f][i][j] = 0;

S.num = 0;//初始化数独

S.original[x][y][z] = v;

S.num = 1;

CreateCnf();//生成cnf文件

ReadCNF(CNF, SUDOKU);

ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

for (int i = 0; i <= boolCnt; i++)

ans[i] = 1;

DPLL(CNF, ans, SUDOKU);//求解双数独

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

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

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

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

if (ans[VarTrans(f, i, j, n)]) S.pattern[f][i][j] = n;

S.num = 162;

}

void SudPlay()

{

int choice = 1, diff, op;

int x, y, z, v;

while (choice) {

system("cls");

printf("请选择难度：\n");

printf(" 1.简单 2.中等 3.困难 \n");

scanf("%d", &diff);

printf("正在生成数独，请稍等……");

Sudcreate();//随机形成一个数独

for (int f = 1; f <= 2; f++)

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

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

S.original[f][i][j] = S.pattern[f][i][j];

int dignum = 0,xx,yy,zz;//给数独挖洞

do

{

xx = rand() % 2 + 1;

yy = rand() % 9 + 1;//随机确定数独一个位置的数值

zz = rand() % 9 + 1;

Candig(xx, yy, zz);

dignum++;

} while (dignum < diff \* 40);

op = 1;

while (op)

{

system("cls");

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

for (int i = 1; i <= 6; i++)

{

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

printf("%3d", S.original[1][i][j]);

printf("\n");

}

for (int i = 7; i <= 9; i++)

{

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

printf("%3d", S.original[1][i][j]);

for (int j = 4; j <= 9; j++)

printf("%3d", S.original[2][i - 6][j]);

printf("\n");

}

for (int i = 4; i <= 9; i++)

{

printf(" ");

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

printf("%3d", S.original[2][i][j]);

printf("\n");

}

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

printf("\n1.填写 2.更多提示\n0.显示答案/退出\n");

scanf("%d", &op);

if (op == 2) hint();

if (op == 1)

{

printf("\n输入填写的坐标与待填值:");

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

if (!S.original[x][y][z] && v == S.pattern[x][y][z])

{

printf("\n填写正确!\n");

S.original[x][y][z] = S.pattern[x][y][z];

getchar(); getchar();

}

else {

printf("\n答案错误\n");

getchar(); getchar();

}

}

}

for (int i = 1; i <= 6; i++)

{

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

printf("%3d", S.pattern[1][i][j]);

printf("\n");

}

for (int i = 7; i <= 9; i++)

{

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

printf("%3d", S.pattern[1][i][j]);

for (int j = 4; j <= 9; j++)

printf("%3d", S.pattern[2][i - 6][j]);

printf("\n");

}

for (int i = 4; i <= 9; i++)

{

printf(" ");

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

printf("%3d", S.pattern[2][i][j]);

printf("\n");

}

printf("1.再来一局 0.退出\n");

scanf("%d", &choice);

}

}

**display.cpp**

#include"define.h"

#pragma warning(disable : 4996)

#pragma warning(disable : 6031)

int inf = 1061109567;

int boolCnt, clauseCnt, sumOfAns, branchRule;//变元数量 子句数量 选择策略

char fileName[200]; //cnf文件文件名

int \*ans=NULL; //结果数组

sud S; //数独问题

clause\* CNF=NULL; //SAT问题

int main()

{

int op = 1, op1 = 1, op2 = 1, flag\_cnf, difficulty;

clock\_t start, finish;

double time;

bool res;

while (op)

{

system("cls");

printf(" 菜单 \n");

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

printf(" 1.DPLL 2.双数独 0.退出 \n");

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

scanf("%d", &op);

switch (op)

{

case 1:

flag\_cnf = 1, op1 = 1;

while (op1)

{

system("cls");

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

printf("1.读文件 2.打印文件 3.DPLL 4.输出并保存结果 0.返回\n");

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

scanf("%d", &op1);

switch (op1)

{

case 1:

printf("请输入测试文件名:");

scanf("%s", fileName);

if (ReadCNF(CNF, SAT) == OK)

flag\_cnf = 0, printf("cnf文件读取完毕！\n");

else

printf("cnf文件读取失败!\n");

getchar(), getchar();

break;

case 2:

if (flag\_cnf)

printf("尚未未读取cnf文件!\n ");

else

{

printf("文件中有如下子句：\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

for (clause\* lp = CNF; lp; lp = lp->next)

{

for (node\* tp = lp->head; tp; tp = tp->next)

printf("%d ", tp->value);

printf("\n");

}

}

getchar(), getchar();

break;

case 3:

if (flag\_cnf)

printf("尚未未读取cnf文件!\n ");

else

{ branchRule = 1;

ans = (int\*)malloc(sizeof(int) \* (boolCnt + 1));

for (int i = 0; i <= boolCnt; i++)

ans[i] = 1;

start = clock();

res = DPLL(CNF, ans, SAT);

finish = clock();

if (!res)

printf("不可满足！\n");

else

{

printf("可满足！\n");

}

time = (double)(finish - start);

printf("DPLL过程优化后用时%lfms\n", time );

destroyClause(CNF);

ReadCNF(CNF, SAT);

branchRule = 4;

for (int i = 0; i <= boolCnt; i++)

ans[i] = 1;

start = clock();

res = DPLL(CNF, ans, SAT);

finish = clock();

time = (double)(finish - start);

printf("DPLL过程优化前用时%lfms\n", time );

}

getchar(), getchar();

break;

case 4:

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

{

if (ans[i])

printf("%d ", i);

else

printf("%d ", -i);

}

case 5:

if (flag\_cnf)

printf("尚未未读取cnf文件!\n ");

else

{

if (printCNF(res, ans, time))

printf("结果已保存\n");

else

printf("结果保存失败!\n");

}

getchar(), getchar();

break;

case 0:

break;

}

}

break;

case 2:

op2 = 1;

while (op2)

{

system("cls");

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

printf(" 1.数独游戏 0.返回 \n");

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

scanf("%d", &op2);

switch (op2)

{

case 1:

SudPlay();

getchar();

break;

default:

break;

}

}

break;

case 0:

break;

}

}

return 0;

}