华中科技大学

课程设计报告

题目：基于 SAT 的对角线数独游戏求解程序

课程名称：程序设计综合课程设计

专业班级：计算机科学与技术 2305 班

学　　号：U202315573

姓　　名：叶润莹

指导教师：李丹

报告日期：2024.10.2

计算机科学与技术学院

任务书

□ 设计内容

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

□ 设计要求

要求具有如下功能：

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

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

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

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

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

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

口 参考文献

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

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

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

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

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

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

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

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

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

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

http://zhangroup.aporc.org/images/files/Paper_3485.pdf

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

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

目录

1 引言

1.1 课题背景与意义

1.1.1 课题背景

在现代游戏与逻辑推理研究领域，对角线数独作为数独的一种变体，因其独特的规则和挑战性而受到广泛关注。与传统数独相比，对角线数独不仅要求每行、每列和每个区域内的数字互不重复，还需确保主对角线和副对角线上的数字也符合这一要求。这种复杂性使得对角线数独的求解成为一个富有意义的计算机科学研究课题。

1.1.2 求解方法

基于 SAT（布尔满足问题）的求解程序为对角线数独提供了一种高效的解决方案。通过将数独问题形式化为布尔逻辑表达式，SAT 求解器能够利用其强大的搜索和推理能力，快速定位解的空间。这种方法不仅提高了求解的效率，还为研究算法的优化与复杂性理论提供了实践中的应用案例。

1.1.3 研究意义

在人工智能和游戏开发领域，此研究具有重要的现实意义。其一，成功的求解程序可以为数独游戏的爱好者提供更高效的解题工具；其二，探索布尔求解技术在其他逻辑游戏和优化问题中的应用，能够推动相关算法的进一步发展，具有广泛的应用前景与研究价值。综上所述，基于 SAT 的对角线数独求解程序的研究不仅能够丰富数独的解题方法，也为计算机科学领域的相关研究提供了新的思路与方向。

1.2 国内外研究现状

在国内，对角线数独的研究起步相对较晚，但随着数独及其变体的流行，相关研究逐渐增多。国内学者主要集中于数独的传统解法与优化算法的研究，通过回溯法、启发式搜索等方法提升求解效率。此外，部分研究也开始探索结合人工智能算法，如遗传算法和模拟退火等，来进行对角线数独的求解。这些研究虽然取得了一定的进展，但系统性与全面性的理论研究仍显不足。

在国际上，对角线数独和 SAT 求解的研究相对成熟，许多研究者将其视为逻

辑与计算复杂性结合的典范。国外的研究大多集中于使用 SAT 求解器对数独问题的建模与求解，相关文献中出现了各种针对 SAT 求解性能的优化策略，包括但不限于数据结构、剪枝技术和启发式方法。同时，不少研究探讨了对角线数独的不同变体与应用，使得该领域的研究成果不断丰富并具有更广泛的应用背景。

总体来看，尽管国内外在对角线数独的研究领域都取得了一定进展，但仍然存在差异。国内研究更倾向于实验与应用，而国外研究则在理论与方法论层面更为深入。因此，提高国内在这一领域的研究深度与广度，特别是在结合 SAT 求解技术的系统性研究，将是未来的重要发展方向。这不仅有助于推动数独及其相关技术的进一步应用，也将为学术界带来更为丰富的研究成果。

1.3 课程设计的主要研究工作

本设计要求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。同时能用此 SAT 求解器来解对角线数独游戏，并输出正确答案。

2 系统需求分析与总体设计

2.1 系统需求分析

这个系统应具有如下功能：

（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。

（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。

（3）DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。

（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）程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现，提供明确的性能优化率结果。

（6）SAT 应用：将对角线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。

2.2 系统总体设计

我设计的系统能实现两个功能：用 SAT 算法对给定的 cnf 文件进行求解，和玩对角线数独游戏。第一个功能包含读取文件、建立相应的数据结构并存储数据、运用 DPLL 算法进行求解、保存输出文件这几个模块，第二个功能包含从 20 个完整合法对角线数独格局中随机选取一个、基于挖洞法生成数独、将数独制约条件转化成 cnf 文件、最后利用功能一的模块进行求解这几个模块。

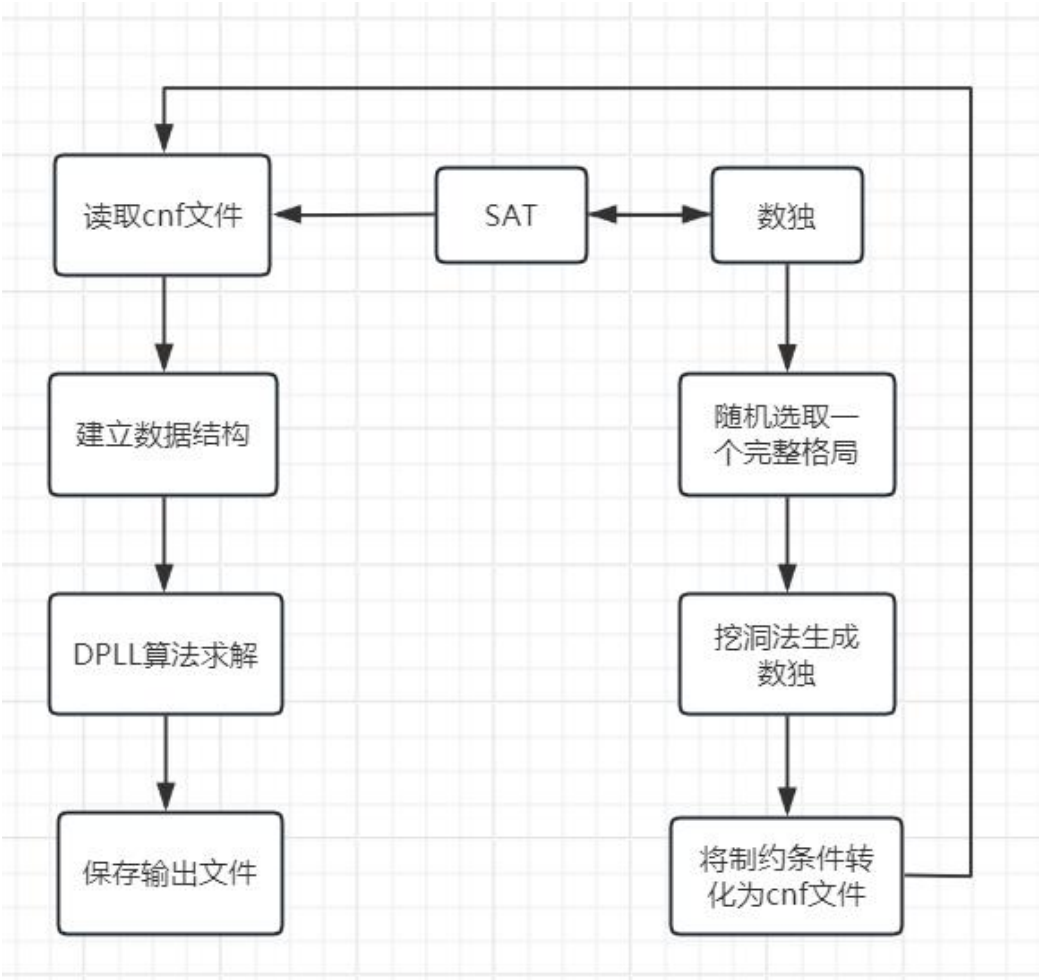

读取cnf文件
SAT
数独
建立数据结构
随机选取一个完整格局
DPLL算法求解
挖洞法生成数独
保存输出文件
将制约条件转化为cnf文件

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

系统中主要处理的数据是 cnf 文件中的子句及子句中的文字。我选择用单链表来储存这些数据。主要包括：

（1）定义结构 CNode 来记录子句中的文字，CNode 中包含该布尔变元的值、指向下一个文字 CNode 的指针 PNode。它们组成的链表表示一个完整子句。

（2）定义结构 CHead 来记录 cnf 文件中的子句，CHead 中包含子句中的文字个数、指向第一个文字的指针 PNode、指向下一个子句 CHead 的指针 PHead。它们组成的链表表示一个完整的 cnf 文件。

（3）同时，为了优化 DPLL 算法中选取变元的策略，我又定义结构数组 times 来记录每个布尔变元在 cnf 文件中出现的频次，其中包含布尔变元 x 的值、出现的总频次、x 和-x 分别出现的频次，以及该次出现时的符号（正负性）。

3.2 主要算法设计

对于SAT功能：

（1）读取cnf文件、建立数据结构：先手动跳过命令行，读取子句数和文字数，再依此建立数据结构并初始化，最后读取文件、在链表上添加Chead和CNode。

（2）DPLL算法求解：步骤一：先判断子句集中是否有单子句，若有则选定该单子句的文字为uni，将包含uni的单子句删除，并把非单子句中出现的-uni文字删除，返回一个新的子句集，若它为空则原子句集可被满足，若它含有空子句则无法被满足；步骤二：接着依照策略选取一个新的文字uni（直接选取下一个文字，或选取出现频次最高的文字），将uni作为单子句加入子句集，重复步骤一，若结果为无法满足，则将-uni作为单子句加入子句集再重复步骤一；循环步骤二，直到找到一组解或证明无解。

（3）DPLL算法的优化：主要优化点在于选取变元uni的策略上，即从“直接选取下一个文字”变为“选取出现频次最高的文字”的策略。同时，加入了clock()函数来计算优化前后的程序运行时间，以提供明确的性能优化率结果。

对于数独功能：

（1）挖洞法生成数独：先选取一个完整的对角线数独格局，再输入挖洞数，

然后开始挖洞：随机生成两个9以内的数字作为挖洞位置的坐标，先检查该位置是否已被挖洞，若是则再随机生成一组坐标，若不是则将该位置上的数字改为0，并将挖洞坐标存储在二维数组c中，以供检查。最后返回挖好洞的数独。

（2）将制约条件转化为cnf文件：对于已确定的位置上的数字，我们用单子句描述，如“222 0”；格约束：每一个格子中可填的数唯一在1~9内，则将对于一个格子取1~9的情况作为文字组成一个子句，对每个空缺格子都生成这样一个子句，如“111 112 113 114 115 116 117 118 119 0”，并生成约束其不能同时填两个数字的子句，如“-111 -112 0”；接着是行约束、列约束和对角线约束：某两个格子里不能填重复数字，我们生成子句来制约它，如“-111 -211 0”；最后是3*3盒子约束，与前面所说的行约束、列约束、对角线约束类似。

4 系统实现与测试

4.1 系统实现

系统实现的软件环境为 Dev-C++ 5.11；

C 语言定义数据类型如下：

```
typedef struct CNode{       //一个单纯的链表(记录一个文字)
        ElemType data;
        struct CNode *next;
}CNode, *PNode;

typedef struct CHead{       //链表的链表(记录一个子句)
    ElemType info;
    struct CHead *down;
    struct CNode *right;
}CHead, *PHead;

typedef struct times{
    double count;
    ElemType data;
    double positive;
    double negative;
    ElemType value;
}times;
```

各模块所需函数如下：

（1）输入输出文件：先通过 EntrySAT 函数进入功能一，在 EntrySAT 中调用 FileReader 来根据文件构建数据结构，构建过程中调用 IniClause 来构建新的 CHead 结构体存放子句、调用 IniNode 来构建新的 CNode 结构体存放文字；接着返回 EntrySAT，调用 DPLLSolver 求解，最后调用 SaveSATFile 来保存输出文件。

（2）DPLL 算法求解：在 DPLLSolver 函数中，有调用判断是否存在单子句的函数 isUniClause，判断是否存在空子句的函数 isEmptyClause，选取变元的策略函数 strategy1（优化前）和 strategy2（优化后），复制整个子句集的函数

Dulplication，将变元作为单子句加入子句集的函数 AddUniClause，简化子句集的函数 Simplification（其中包含删除单子句的 DelClause 和删除非单子句中文字的 DelNode，并会调用 FreeCNodes 函数来释放空间）。

（3）挖洞法生成数独：先调用函数 CreateSudokuToFile 来随机生成一个完整数独格局，再调用 CreatePlay 来挖洞。

（4）将制约条件转化为 cnf 文件并求解：由函数 TranToCNF 完成。之后由函数 EntrySUDOKU 来调用 DPLL 算法求解，完成后由 SaveSudokuFile 来保存输出文件，并可通过函数 print 将答案打印在屏幕上。

4.2 系统测试

功能一测试：用SAT算法对给定的cnf文件进行求解：

程序测试方法：运用verify程序进行验证，verify程序为验证程序，验证计算结果是否正确，使用方法为：cmd—verify.exe *.cnf *.res；（使用绝对路径或者相对路径，相对路径需要三个文件在同一文件夹下）。

（1）功能测试-满足算例：主要检测程序是否能处理一个可以满足的cnf文件，检测结果如下：

```
C:\Users\y-one>cd "C:\Users\y-one\Desktop\大二上程序设计课设\test"

C:\Users\y-one\Desktop\大二上程序设计课设\test>verify.exe sat-20.cnf sat-20.res

v  value
1  0
2  1
3  1
4  1
5  0
6  0
7  0
8  1
9  1
10  1
11  1
12  0
13  0
14  1
15  1
16  0
17  1
18  1
19  1
20  1

Assignment is satisfying!
```

（2）功能测试-不满足算例：主要检测程序是否能处理一个无法满足的cnf文件并输出报错信息，检测结果如下：

```
C:\Users\y-one\Desktop\大二上程序设计课设\test>verify.exe unsat-5cnf-30.cnf unsat-5cnf-30.res

Clause 12  (25 29 24 11)  is not satisfied!
Clause 62  (2 4 12 6 1)  is not satisfied!
Clause 67  (9 23 17 24)  is not satisfied!
Clause 135  (29 22 10 7)  is not satisfied!
Clause 172  (30 24 16 20 11)  is not satisfied!
Clause 187  (16 21 24 18)  is not satisfied!
Clause 202  (28 21 20 26 9)  is not satisfied!
Clause 230  (26 7 27 2 22)  is not satisfied!
Clause 231  (15 9 26 11)  is not satisfied!
Clause 233  (5 8 3 26)  is not satisfied!
Clause 256  (1 23 2 20 6)  is not satisfied!
Clause 338  (12 7 23 26)  is not satisfied!
Clause 359  (25 30 27 21 19)  is not satisfied!
Clause 363  (5 2 16 1 28)  is not satisfied!
Clause 372  (1 16 20 22 30)  is not satisfied!
```

（3）性能测试：主要检查程序在处理含有较多文字和子句数的 cnf 文件时的性能，检查结果如下：

```
C:\Users\y-one\Desktop\大二上程序设计课设\test>verify.exe ais10.cnf ais10.res

v  value
1  0
2  0
3  0
4  0
5  0
6  0
7  0
8  0
9  0
10  1
11  1
12  0
13  0
14  0
15  0
16  0
17  0
18  0
19  0
20  0
21  0
22  0
23  0
24  0
25  0
```

```
155  0
156  0
157  1
158  0
159  0
160  0
161  0
162  0
163  0
164  0
165  1
166  0
167  0
168  0
169  0
170  0
171  0
172  0
173  1
174  0
175  0
176  0
177  0
178  0
179  0
180  0
181  0

Assignment is satisfying!
```

功能二测试：玩对角线数独游戏：

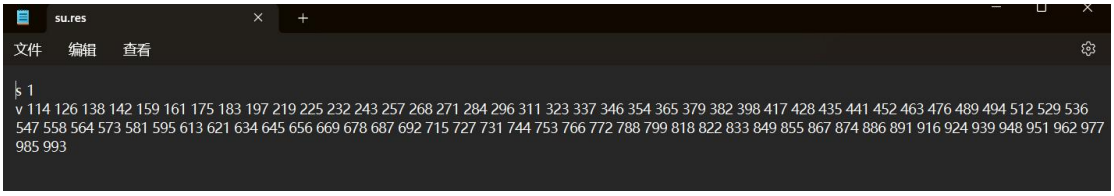
程序测试方法：运用 Dev-C++自带的调式运行功能。

功能测试：主要检查程序是否可以完成基于 SAT 的对角线数独游戏的生成和求解，检查结果如下：

```
   ---------------------------------
1  | 4  _  8 | 2  9  _ | 5  _  7  |
   |         |         |          |
2  | _  _  _ | 3  7  _ | _  _  _  |
   |         |         |          |
3  | _  _  _ | 6  _  _ | 9  2  8  |
   ---------------------------------
4  | 7  _  _ | _  _  3 | 6  9  _  |
   |         |         |          |
5  | 2  _  6 | 7  _  4 | _  1  _  |
   |         |         |          |
6  | _  1  4 | 5  _  9 | _  7  _  |
   ---------------------------------
7  | 5  _  1 | 4  _  6 | 2  _  _  |
   |         |         |          |
8  | 8  2  3 | 9  5  _ | 4  6  _  |
   |         |         |          |
9  | 6  _  _ | _  1  _ | _  5  _  |
   ---------------------------------


        Welcome to My Sudoku!
 -------------------------------------
 |   1. Answer     |   0. Exit       |
 -------------------------------------
 请输入你的选择[0~1]:1
数独填写成功!请进入文件查看!

请按回车键回到主菜单!
```



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）通过程序实现了基于 SAT 的 cnf 文件的求解，主要包括输入输出功能、解析文件并建立相应的数据结构功能、运用 DPLL 算法求解功能、时间性能的测量功能等。

（2）SAT 应用：将对角线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，相应功能有挖洞法生成数独、将制约条件转化为 cnf 文件等。

5.2 工作展望

在今后的研究中，可围绕着如下几个方面开展工作：

（1）在生成对角线数独时，可设计一种算法自动生成对角线数独的合法完整初始格局，而不用从互联网上读取。

（2）目前的程序在处理文字和子句数量较多的算例时依然存在运行时间长、输出结果不稳定、可能异常终止的情况，今后可研究和比较不同的 SAT 求解器（如 CDCL、DPLL 等）在对角线数独问题上的性能，探讨如何通过改进 SAT 求解算法。

（3）可提升程序的交互性：如求解数独时令游戏可玩，填到错误数字时会报出错误等。

6 体会

在完成“基于 SAT 的对角线数独游戏求解程序”的设计与实现后，我深刻体会到编程不仅是一项“技术活”，更是一种思维方式的磨练。通过将复杂的数独问题转化为 SAT 形式、在解决过程中反复探索与调整算法，我对逻辑推理和算法设计有了更深刻的理解。此外，看到程序能够顺利求解不同难度的数独谜题，无疑给了我极大的成就感。这个过程让我意识到，面对看似困难的挑战，只要坚持探索与实践，总能找到解决问题的路径。同时，我也认识到算法优化和用户体验的重要性，今后将更加注重在技术创新与人性化设计之间寻找平衡。这次程序设计不仅提升了我的编程能力，更激发了我更深入研究人工智能与组合优化领域的兴趣。

参考文献

[1] 袁凌，祝建华，许贵平，李建军，周全. 数据结构——从概念到算法[M]. 人民邮电出版社，2023.

附录

```
//@Global.h
#ifndef WORKSHOP_GLOBAL_H
#define WORKSHOP_GLOBAL_H

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

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR -1
#define MAXN 1200
#define COL 9
#define ROW 9
#define AC 0
#define WA -1

typedef int ElemType;
typedef int status;

typedef struct CNode{    //一个单纯的链表(记录一个结点)
      ElemType data;
      struct CNode *next;
}CNode, *PNode;

typedef struct CHead{    //链表的链表(记录一个子句)
   ElemType info;
   struct CHead *down;
   struct CNode *right;
```

```
}CHead, *PHead;

typedef struct times{
    double count;
    ElemType data;
    double positive;
    double negative;
    ElemType value;
}times;

times counter[MAXN];
int player[ROW][COL];

void test(CHead **C);
int opp(int x);

CHead* IniClause(CHead **C);
CNode* IniNode(CNode **right);
status FileReader(CHead **C, char filename[]);

status SaveSATFile(int re, double t, char filename[]);
status SaveSudokuFile(int re, char filename[]);

CHead* isUniClause(CHead *C);
status isEmptyClause(CHead *C);

status DPLLSolver(CHead **C);
CHead* Duplication(CHead **C);
int strategy2(CHead **C);
int strategy1(CHead **C);

status EntrySUDOKU(CHead **C);
int EntrySAT(char FileName[]);
void CreatePlay(int a[][COL], int b[][COL], int numDigits);
status CreateSudokuToFile(char SudokuFile[], int holes);
status TranToCNF(int a[][COL], int holes, char SudokuFile[]);
```

```
void print(int a[][COL]);

#endif

//@main.c
#include "Global.h"
int main()
{
    int op = 1, choice = 1;
   int holes = 0;
    int pd, re, uni = 111, flag = 1, flagg = 1, flag2 = 1, i, j, k;
    PHead C = NULL, temp1 = NULL, temp2 = NULL;
    char FileName[200];
    char SudokuFileName[200] = "su.txt";
    while(op){
       system("cls");
        printf("\n\n");
        printf("       Welcome to My System!\n");
        printf("   ------------------------------\n");
        printf("       1. Sat        2. Sudoku\n");
        printf("       0. Exit\n");
        printf("   ------------------------------\n");
        printf("       请输入你的选择[0~2]:");
        scanf("%d", &op);
        switch(op){
           case 1:
               printf("请输入文件名:\n");
               scanf("%s", FileName);
               if((pd=EntrySAT(FileName))==OK){
                   printf("成功!请进入对应文件中查看!\n\n");
                   printf("请按回车键回到主菜单!\n");
               }
               else{
                   printf("unsatisfy!\n\n");
                   printf("请按回车键回到主菜单!\n");
               }
```

```
            getchar();getchar();
            break;
        case 2:
            printf("请输入挖洞数:\n");
            scanf("%d", &holes);
            for(i=0; i<COL; i++){
                for(j=0; j<ROW; j++){
                    player[i][j] = 0;
                }
            }
            CreateSudokuToFile(SudokuFileName, holes);
            FileReader(&C, SudokuFileName);
            system("cls");
            print(player);
            printf("\n\n");
            printf("        Welcome to My Sudoku!\n");
            printf(" ------------------------------------\n");
            printf(" |   1. Answer     |   0. Exit       |\n");
            printf(" ------------------------------------\n");
            printf(" 请输入你的选择[0~1]:");
            scanf("%d", &op);
            while(op && op!=3){
                switch(op){
                    case 1:
                        re = EntrySUDOKU(&C);
                        pd = SaveSudokuFile(re, SudokuFileName);
                        if(pd==OK){
                            printf("数独填写成功！请进入文件查
看!\n\n");
                            printf("请按回车键回到主菜单!\n");
                        }
                        else{
                            printf("该数独棋盘无解!\n");
                        }
                        getchar();getchar();
                        system("cls");
```

```
                        printf("\n");
                        printf("       Welcome to My System!\n");
                                                        printf("
------------------------------\n");
                                printf("         1. Sat          2.
Sudoku\n");
                            printf("       0. Exit\n");
                                                        printf("
------------------------------\n");
                            printf("       请输入你的选择[0~2]:");
                         scanf("%d", &op);
                        getchar();
                        break;
                    case 0:
                        break;
                    default:
                        printf("输入非法!请按回车键重新选择!\n");
                        getchar();getchar();
                        break;
                }//end of switch(interior)
            }//end of while(interior)
        case 0:
            break;
        default:
            printf("输入非法!请按回车键重新选择!\n");
            getchar();getchar();
            break;
    }//end of switch
}//end of while
printf("欢迎下次再使用本系统!\n");
return 0;
}//end of main

//@file DPLLSolver.c
#include "Global.h"
status FreeCNodes(CHead **node)    //释放空间
```

```
{
    CNode *p = NULL, *q = NULL;
    p = (*node)->right;
    q = p->next;
    if(q==NULL){
        free(p);
    }
    while(q!=NULL){
        free(p);
        p = q;
        q = p->next;
    }
    return OK;
}
status DelClause(int x, CHead **C)    //去含 x 的子句
{
    PHead p = *C, q = NULL;
    PNode m = NULL;
    int flag = 1;
    while(p!=NULL){
        m = p->right;
        while(m!=NULL){
            if(m->data==x){
                flag = 0;
                break;
            }
            m = m->next;
        }
        if(flag==1){
            break;
        }
        else if(flag==0){
            q = *C;
            *C = (*C)->down;
            p = *C;
            FreeCNodes(&q);
```

```
            free(q);
            q = NULL;
            flag = 1;
        }
    }
    if(*C==NULL){
        return OK;
    }
    p = *C;
    q = p->down;
    flag = 0;
    while(q!=NULL){
        m = q->right;
        while(m!=NULL){
            if(m->data==x){
                flag = 1;
                p->down = q->down;
                FreeCNodes(&q);
                free(q);
                q = NULL;
                break;
            }
            m = m->next;
        }
        if(flag==0){
            p = p->down;
            q = q->down;
        }
        else if(flag==1){
            q = p->down;
            flag = 0;
        }
    }
    return OK;
}
status DelNode(int x, CHead **C)    //去多子句中的结点
```

```
{
    PHead temp1 = *C;
    PNode temp2 = NULL, p = NULL, q = NULL;
    while(temp1!=NULL){
        temp2 = temp1->right;
        while(temp2->data==opp(x)){
            temp1->right = temp2->next;
            free(temp2);
            temp2 = temp1->right;
            temp1->info--;
        }
        p = temp2;
        q = p->next;
        while(q!=NULL){
            if(q->data==opp(x)){
                p->next = q->next;
                free(q);
                q = p->next;
                temp1->info--;
            }
            else{
                p = q;
                q = p->next;
            }
        }
        temp1 = temp1->down;
    }
    return OK;
}
status Simplification(int uni, CHead **C)    //在C中去掉uni
{
    DelClause(uni, C);    //去单子句
    DelNode(uni, C);      //去多子句中的结点
    return OK;
}
status AddUniClause(int x, CHead **C)    //将一个值为x的新单子句加入
```

```
C_copy1
{
    PHead ne = NULL;
    ne = (PHead)malloc(sizeof(CHead));
    if(!ne) exit(ERROR);
    ne->down = *C;
    ne->right = NULL;
    ne->info = 0;
    *C = ne;
    ne->info = 1;
    PNode node = IniNode(&(ne->right));
    node->data = x;
    counter[abs(x)].count += 1;
    if(x>0){
        counter[abs(x)].positive += 1;
    }
    else{
        counter[abs(x)].negative += 1;
    }
    node = IniNode(&(node->next));
    return OK;
}
status DPLLSolver(CHead **C)
{
    PHead dnow = NULL;
    int uni = 0;
    while((dnow=isUniClause(*C))!=NULL){
        uni = dnow->right->data;
        uni > 0 ? (counter[abs(uni)].value = 1) : (counter[abs(uni)].value
= -1);//value 记录 uni 正负
        Simplification(uni, C);
        if(*C==NULL) return TRUE;
        else if(isEmptyClause(*C)) return FALSE;    //有空子句,无法满足
    }
    uni = strategy2(C);    //!!!
    uni > 0 ? (counter[abs(uni)].value = 1) : (counter[abs(uni)].value
```

```
= -1);
    PHead copy1 = Duplication(C);
    AddUniClause(uni, &copy1);
    if(DPLLSolver(&copy1)) return TRUE;    //递归调用
    else{
        PHead copy2 = Duplication(C);
        uni   >   0   ?   (counter[abs(uni)].value   =   -1)   :
(counter[abs(uni)].value = 1);
        AddUniClause(opp(uni), &copy2);    //将-uni 加入 C_copy2
        return DPLLSolver(&copy2);         //递归调用
    }
}
CHead* isUniClause(CHead *C)    //找单子句
{
    CHead *temp = C;
    while(temp!=NULL){
        if(temp->info==1){    //返回第一个单子句出现的地址
            return temp;
        }
        temp = temp->down;
    }
    return NULL;
}
status isEmptyClause(CHead *C)    //判断是否有空子句
{
    CHead *temp = C;
    while(temp!=NULL){
        if(temp->info==0){
            return OK;
        }
        temp = temp->down;
    }
    return FALSE;
}
CHead* Duplication(CHead **C)    //复制一个 C,并返回新复制体地址
{
```

```
    PHead p = *C, q = NULL, dt = NULL, dnow = NULL;
    PNode rnow = NULL, rt = NULL;
    while(p!=NULL){
        dnow = IniClause(&q);
        dnow->info = p->info;
        rt = p->right;
        while(rt!=NULL){
            rnow = IniNode(&(dnow->right));
            rnow->data = rt->data;
            rt = rt->next;
        }
        p = p->down;
    }
    return q;
}
int strategy1(CHead **C)    //直接选取第一个值作为uni!
{
    return (*C)->right->data;
}
int strategy2(CHead **C)    //返回出现次数最多的x
{
    PHead dnow = *C;
    PNode rnow = NULL;
    int x, pos = 0;
    double max = 0.0;
    while(dnow!=NULL){    //找count最大的x
        rnow = dnow->right;
        while(rnow!=NULL){
            x = rnow->data;
            if(counter[x].count>max){
                max = counter[x].count;
                pos = x;
            }
            rnow = rnow->next;
        }
        dnow = dnow->down;
```

```
    }
    if(pos==0) pos = (*C)->right->data;    //找不到就用 strategy1 了
    if(counter[pos].positive>counter[pos].negative) return pos;    //返回正负值中占大头的
    else return opp(pos);
}
int EntrySAT(char FileName[])
{
    PHead C = NULL;
    int re, i;
    clock_t start_t, end_t;
    double total_t;
    FileReader(&C, FileName);    //根据文件构建数据结构
    start_t = clock();           //记录开始时间
    re = DPLLSolver(&C);
    end_t = clock();             //记录结束时间
    total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC * 1000;    //计算总时间
    SaveSATFile(re, total_t, FileName);    //保存输出文件
    if(re==FALSE) return ERROR;
    return OK;
}
status SaveSATFile(int re, double t, char filename[])    //保存输出文件
{
    int i;
    for(i=0; filename[i]!='\n'; i++){    //更改文件名
        if(filename[i]=='.'){
            filename[i+1] = 'r';
            filename[i+2] = 'e';
            filename[i+3] = 's';
        }
    }
    FILE *fp;
    fp = fopen(filename, "w");
    if(!fp) exit(-1);
```

```
    if(re==TRUE){
        fprintf(fp, "s 1\n");    //表示 satisfy
        fprintf(fp, "v ");
        for(i=1; counter[i].data!=0 && i<MAXN; i++){
            if(counter[i].value==1){
                fprintf(fp, "%d ", i);
            }
            else if(counter[i].value==-1){
                fprintf(fp, "%d ", -i);
            }
        }
        fprintf(fp, "\n");
    }
    else{                        //表示 unsatisfy
        fprintf(fp, "s 0\n");
    }
    fprintf(fp, "t %f", t);
    fclose(fp);
    return OK;
}

//@file FileReader.c
#include "Global.h"
status FileReader(CHead **C, char filename[])    //根据文件构建数据结
构
{   FILE *fp = NULL;
    fp = fopen(filename, "r");
    if(!fp) exit(ERROR);
    char ch;
    int num1, num2;
    while((ch=fgetc(fp))!='p'){    //手动跳过命令行
        if(ch=='c'){
            while((ch=fgetc(fp))!='\n'){
                ;
            }
        }
```

```
}
while((ch=fgetc(fp))!='f'){
    ;
}
fscanf(fp, "%d %d", &num1, &num2);
int i;
for(i=0; i<=num1; i++){    //初始化 counter
    counter[i].count = 0.0;
    counter[i].data = 0;
    counter[i].negative = 0.0;
    counter[i].positive = 0.0;
    counter[i].value = 0;
}
int x;
PHead dnow = NULL;
PNode rnow = NULL;
i = num2;
while(i>0){
    i--;
    dnow = IniClause(C);    //新建一个子句
    do{
        fscanf(fp, "%d", &x);
        counter[abs(x)].data = abs(x);
        rnow = IniNode(&(dnow->right));    //在该子句上新建一个结点
        rnow->data = x;
        dnow->info++;
    }while(x!=0);
}
fclose(fp);
i = num2;
dnow = *C;
while(i>0){
    i--;
    rnow = dnow->right;    //遍历子句上的每个结点
    do{                    //记录总出现频率和正负出现频率
        x = rnow->data;
```

```
            counter[abs(x)].count += pow(0.5, (double)(dnow->info - 1));
            if(x>0){
                counter[abs(x)].positive += pow(0.5, (double)(dnow->info
- 1));
            }
            else{
                counter[abs(x)].negative += pow(0.5, (double)(dnow->info
- 1));
            }
            rnow = rnow->next;
        }while(x!=0);
        dnow = dnow->down;    //遍历每个子句
    }
    return OK;
}
CHead* IniClause(CHead **C)    //初始化一个新的 CHead 结构体，并将其添
加到链表的末尾
{
    CHead *ne = NULL, *temp = *C;
    ne = (PHead)malloc(sizeof(CHead));
    if(!ne) return FALSE;
    ne->down = NULL;
    ne->right = NULL;
    ne->info = -1;
    if(*C==NULL){
        *C = ne;
        return ne;
    }
    while(temp->down!=NULL){
        temp = temp->down;
    }
    temp->down = ne;
    return ne;
}
CNode* IniNode(CNode **right)    //初始化一个新的链表节点，并将其添加
到指定链表的末尾
```

```
{
    CNode *ne = NULL, *temp = *right;
    ne = (PNode)malloc(sizeof(CNode));
    if(!ne) return FALSE;
    ne->next = NULL;
    ne->data = 0;
    if(*right==NULL){
        *right = ne;
        return ne;
    }
    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next = ne;
    return ne;
}
int opp(int x)
{
    return -x;
}

//@Sudoku.c
#include "Global.h"
status CreateSudokuToFile(char SudokuFile[], int holes)
{
     int sudoku2[ROW][COL],i,j;
     int play[ROW][COL]={0};
     srand(time(0));
     int rn = (rand() % 20) + 1;
     switch(rn){
        case 1:{
            int
sudoku[ROW][COL]={{7,1,3,5,2,6,8,4,9},{5,8,2,4,9,1,6,3,7},{9,6,4,7,3,
8,1,2,5},{8,5,6,1,7,4,2,9,3},{3,7,1,2,5,9,4,8,6},{2,4,9,8,6,3,5,7,1},
{4,3,7,6,1,2,9,5,8},{1,2,5,9,8,7,3,6,4},{6,9,8,3,4,5,7,1,2}};
            for(i=0;i<ROW;i++){
```

```
            for(j=0;j<COL;j++)
                sudoku2[i][j]=sudoku[i][j];
         }
        break;
    }
    case 2:{
        int sudoku[ROW][COL] = {
             {4, 6, 8, 2, 9, 1, 5, 3, 7},
             {9, 5, 2, 3, 7, 8, 1, 4, 6},
             {1, 3, 7, 6, 4, 5, 9, 2, 8},
             {7, 8, 5, 1, 2, 3, 6, 9, 4},
             {2, 9, 6, 7, 8, 4, 3, 1, 5},
             {3, 1, 4, 5, 6, 9, 8, 7, 2},
             {5, 7, 1, 4, 3, 6, 2, 8, 9},
             {8, 2, 3, 9, 5, 7, 4, 6, 1},
             {6, 4, 9, 8, 1, 2, 7, 5, 3}
         };
         for(i=0;i<ROW;i++){
            for(j=0;j<COL;j++)
                sudoku2[i][j]=sudoku[i][j];
         }
        break;
    }
    case 3:{
        int
sudoku[ROW][COL]={6,7,4,1,3,2,5,8,9,2,9,5,8,4,7,6,1,3,8,3,1,5,6,9,7,2,
4,1,4,8,3,9,6,2,5,7,3,5,7,2,8,1,9,4,6,9,6,2,4,7,5,8,3,1,7,1,3,6,2,8,4,
9,5,4,2,6,9,5,3,1,7,8,5,8,9,7,1,4,3,6,2};
        for(i=0;i<ROW;i++){
            for(j=0;j<COL;j++)
                sudoku2[i][j]=sudoku[i][j];
         }
        break;
    }
    case 4:{
        int
```

```
sudoku[ROW][COL]={6,1,7,3,4,9,2,8,5,2,3,9,6,5,8,7,1,4,5,8,4,2,1,7,6,3,
9,4,6,1,9,2,3,8,5,7,7,2,5,1,8,4,9,6,3,8,9,3,7,6,5,4,2,1,3,5,2,4,7,6,1,
9,8,1,4,8,5,9,2,3,7,6,9,7,6,8,3,1,5,4,2};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 5:{
            int
sudoku[ROW][COL]={1,8,6,2,3,4,7,9,5,7,3,9,6,8,5,2,1,4,4,2,5,9,1,7,6,3,
8,3,7,2,4,5,9,8,6,1,8,6,4,3,7,1,5,2,9,9,5,1,8,6,2,4,7,3,5,1,3,5,2,8,9,
4,7,5,4,7,1,9,6,3,8,2,2,9,8,7,4,3,1,5,6};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 6:{
            int
sudoku[ROW][COL]={2,6,3,9,7,8,5,4,1,7,4,9,6,1,5,8,2,3,5,8,1,4,2,3,6,7,
9,6,3,2,8,4,7,1,9,5,9,5,8,1,3,2,7,6,4,4,1,7,5,9,6,2,3,8,3,7,4,2,5,1,9,
8,6,1,9,6,7,8,4,3,5,2,8,2,5,3,6,9,4,1,7};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 7:{
            int
sudoku[ROW][COL]={2,6,7,8,1,9,3,4,5,3,9,4,6,7,5,1,8,2,8,5,1,3,2,4,7,6,
9,7,2,6,5,3,1,8,9,4,1,8,9,7,4,2,5,3,6,5,4,3,9,6,8,2,1,7,4,7,2,1,9,3,6,
5,8,9,3,5,2,8,6,4,7,1,6,1,8,4,5,7,9,2,3};
```

```
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 8:{
            int
sudoku[ROW][COL]={9,2,4,3,1,7,5,6,8,5,6,7,9,8,2,4,1,3,1,8,3,5,6,4,7,9,
2,6,5,2,1,7,3,9,8,4,4,7,8,6,5,9,3,2,1,3,1,9,4,2,8,6,7,5,8,3,6,7,4,1,2,
5,9,7,9,1,2,3,5,8,4,6,2,4,5,8,9,6,1,3,7};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 9:{
            int
sudoku[ROW][COL]={7,4,6,8,1,2,5,9,3,5,3,2,6,7,9,4,8,1,8,1,9,5,4,3,2,6,
7,9,5,7,1,3,6,8,4,2,6,8,3,2,5,4,7,1,9,4,2,1,7,9,8,3,5,6,2,7,4,9,8,1,6,
3,5,3,9,5,4,6,7,1,2,8,1,6,8,3,2,5,9,7,4};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 10:{
            int
sudoku[ROW][COL]={8,4,7,2,3,9,1,6,5,3,5,6,8,1,7,4,9,2,1,2,9,5,4,6,3,8,
7,4,6,3,7,5,8,9,2,1,9,8,1,4,2,3,7,5,6,2,7,5,6,9,1,8,4,3,5,3,4,1,8,2,6,
7,9,6,1,2,9,7,4,5,3,8,7,9,8,3,6,5,2,1,4};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
```

```
            }
            break;
        }
        case 11:{
            int
sudoku[ROW][COL]={4,5,2,3,8,9,6,1,7,3,1,7,2,5,6,9,4,8,6,9,8,4,1,7,5,3,
2,1,8,4,5,7,3,2,6,9,2,7,9,1,6,4,3,8,5,5,6,3,8,9,2,4,7,1,8,3,1,9,4,5,7,
2,6,7,2,5,6,3,1,8,9,4,9,4,6,7,2,8,1,5,3};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
            }
            break;
        }
        case 12:{
            int
sudoku[ROW][COL]={7,3,4,2,8,1,5,6,9,1,5,6,9,7,4,2,8,3,8,2,9,5,6,3,1,4,
7,3,9,5,1,4,6,7,2,8,4,6,8,7,2,9,3,5,1,2,7,1,3,5,8,6,9,4,9,8,7,6,3,5,4,
1,2,6,4,2,8,1,7,9,3,5,5,1,3,4,9,2,8,7,6};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
            }
            break;
        }
        case 13:{
            int
sudoku[ROW][COL]={1,2,4,5,8,6,7,9,3,9,7,6,3,1,4,8,2,5,8,5,3,2,7,9,4,1,
6,5,3,9,6,4,8,1,7,2,6,8,1,7,9,2,3,5,4,2,4,7,1,3,5,6,8,9,4,9,5,8,6,7,2,
3,1,3,6,8,9,2,1,5,4,7,7,1,2,4,5,3,9,6,8};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
            }
            break;
        }
```

```
        case 14:{
            int
sudoku[ROW][COL]={9,8,4,1,5,6,2,7,3,6,7,1,2,3,9,5,4,8,3,5,2,8,7,4,9,1,
6,1,9,7,3,2,8,4,6,5,4,3,8,9,6,5,7,2,1,5,2,6,7,4,1,3,8,9,7,4,5,6,1,3,8,
9,2,8,1,3,4,9,2,6,5,7,2,6,9,5,8,7,1,3,4};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 15:{
            int
sudoku[ROW][COL]={5,2,3,8,1,6,7,9,4,4,6,7,9,3,5,2,1,8,8,1,9,2,7,4,6,3,
5,2,9,8,1,6,7,5,4,3,3,7,5,4,8,9,1,2,6,6,4,1,5,2,3,9,8,7,7,5,2,3,9,8,4,
6,1,1,3,4,6,5,2,8,7,9,9,8,6,7,4,1,3,5,2};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 16:{
            int
sudoku[ROW][COL]={2,1,7,8,6,5,3,4,9,8,3,9,1,7,4,5,2,6,5,4,6,2,3,9,7,1,
8,9,5,4,7,2,3,6,8,1,3,2,1,5,8,6,4,9,7,6,7,8,4,9,1,2,3,5,4,8,5,6,1,2,9,
7,3,7,6,3,9,4,8,1,5,2,1,9,2,3,5,7,8,6,4};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 17:{
            int
sudoku[ROW][COL]={1,4,7,6,3,8,5,9,2,6,8,5,7,2,9,3,4,1,9,2,3,4,1,5,8,7,
```

```
6,4,1,2,5,8,6,7,3,9,3,7,8,1,9,2,4,6,5,5,6,9,3,4,7,2,1,8,2,9,1,8,7,4,6,
5,3,8,5,4,9,6,3,1,2,7,7,3,6,2,5,1,9,8,4};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 18:{
            int
sudoku[ROW][COL]={9,5,8,6,4,7,2,3,1,1,7,4,2,5,3,6,9,8,3,2,6,9,1,8,5,7,
4,4,8,3,1,7,6,9,2,5,2,1,5,3,8,9,4,6,7,6,9,7,4,2,5,1,8,3,8,4,2,7,6,1,3,
5,9,5,3,1,8,9,2,7,4,6,7,6,9,5,3,4,8,1,2};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 19:{
            int
sudoku[ROW][COL]={8,3,9,5,6,7,1,4,2,4,5,7,8,1,2,3,6,9,1,6,2,4,9,3,7,8,
5,3,8,6,7,2,1,9,5,4,9,1,5,6,4,8,2,7,3,7,2,4,3,5,9,8,1,6,2,4,8,1,3,5,6,
9,7,6,9,1,2,7,4,5,3,8,5,7,3,9,8,6,4,2,1};
            for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
             }
            break;
        }
        case 20:{
            int
sudoku[ROW][COL]={3,7,5,2,8,6,1,9,4,4,6,2,1,5,9,8,7,3,8,9,1,7,3,4,6,5,
2,9,8,4,5,1,2,3,6,7,5,3,7,6,9,8,4,2,1,2,1,6,3,4,7,5,8,9,6,4,8,9,7,3,2,
1,5,7,5,3,8,2,1,9,4,6,1,2,9,4,6,5,7,3,8};
            for(i=0;i<ROW;i++){
```

```
                for(j=0;j<COL;j++)
                    sudoku2[i][j]=sudoku[i][j];
            }
            break;
        }
    }
    CreatePlay(sudoku2, play, holes);
    for(i=0; i<COL; i++){
        for(j=0; j<ROW; j++){
            player[i][j] = play[i][j];
        }
    }
    TranToCNF(play, holes, SudokuFile);
    return OK;
}
void CreatePlay(int a[][COL], int b[][COL], int numDigits)
{
    int i, j, k;
    srand((unsigned)time(NULL));
    for(i=0; i<ROW; i++)          //b 先照搬
        for(j=0; j<COL; j++)
            b[i][j] = a[i][j];
    int c[numDigits][2];          //用于存储已经被挖的坐标
    int m, flag = 0;
    for(i=0; i<numDigits; i++){    //开始挖洞
        j = rand() % 9;           //随机生成行列
        k = rand() % 9;
        flag = 0;
        for(m=0; m<i; m++)        //检查该位置是否已被挖掉
            if(j==c[m][0] && k==c[m][1])
                flag = 1;
        if(flag==0){
            b[j][k] = 0;
            c[i][0] = j;
            c[i][1] = k;
        }
```

```
        else i--;
    }
}
status TranToCNF(int a[][COL], int holes, char SudokuFile[])
{
    FILE *fp = fopen(SudokuFile, "w");
    if(!fp) exit(ERROR);
    fprintf(fp, "p cnf 729 %d\n", 6642+81-holes);
    int x, y, z;
    for(x=0; x<ROW; x++){    //确定的数:单子句
        for(y=0; y<COL; y++){
            if(a[x][y]!=0){
                fprintf(fp, "%d 0\n", (x+1)*100 + (y+1)*10 + a[x][y]);
            }
        }
    }
    for(x=1; x<=9; x++){    //填的数是1~9
        for(y=1; y<=9; y++){
            for(z=1; z<=9; z++){
                fprintf(fp, "%d ", x*100 + y*10 + z);
            }
            fprintf(fp, "0\n");
        }
    }
    int i, j, k;
    for(y=1; y<=9; y++){    //行约束
        for(z=1; z<=9; z++){
            for(x=1; x<=8; x++){
                for(i=x+1; i<=9; i++){
                    fprintf(fp, "%d ", opp(x*100 + y*10 +z));
                    fprintf(fp, "%d 0\n", opp(i*100 + y*10 + z));
                }
            }
        }
    }
    for(x=1; x<=9; x++){    //列约束
```

```
        for(z=1; z<=9; z++){
            for(y=1; y<=8; y++){
                for(i=y+1; i<=9; i++){
                    fprintf(fp, "%d ", opp(x*100 + y*10 +z));
                    fprintf(fp, "%d 0\n", opp(x*100 + i*10 + z));
                }
            }
        }
    }
    for(z=1; z<=9; z++){    //3*3 子网络中的列约束
        for(i=0; i<=2; i++){
            for(j=0; j<=2; j++){
                for(x=1; x<=3; x++){
                    for(y=1; y<=3; y++){
                        for(k=y+1; k<=3; k++){
                            fprintf(fp,"%d ",opp((3*i+x)*100+(3*j+y)*10
+ z));
                            fprintf(fp,  "%d  0\n",  opp((3*i+x)*100  +
(3*j+k)*10 + z));
                        }
                    }
                }
            }
        }
    }
    int m;
    for(z=1; z<=9; z++){    //3*3 子网络中的行约束
        for(i=0; i<=2; i++){
            for(j=0; y<=2; j++){
                for(x=1; x<=3; x++){
                    for(y=1; y<=3; y++){
                        for(k=x+1; k<=3; k++){
                            for(m=1; m<=3; m++){
                                fprintf(fp,  "%d  ",  opp((3*i+x)*100  +
(3*j+y)*10 + z));
                                fprintf(fp, "%d 0\n", opp((3*i+k)*100 +
```

```
(3*j+m)*10 + z));
                            }
                        }
                    }
                }
            }
        }
    }
    for(i=1;i<9;i++){    //对角线约束其一
        for(j=i+1;j<=9;j++){
            for(k=1;k<=9;k++)
                fprintf(fp,"-%d -%d 0\n",i*100+i*10+k,j*100+j*10+k);
        }
    }
    for(i=1;i<9;i++){    //对角线约束其二
        for(j=i+1;j<=9;j++){
            for(k=1;k<=9;k++)
                                                fprintf(fp,"-%d         -%d
0\n",i*100+(10-i)*10+k,j*100+(10-j)*10+k);
        }
    }
    fclose(fp);
    return OK;
}
status EntrySUDOKU(CHead **C)
{
    int re, i;
    re = DPLLSolver(C);
    if(re==OK) return re;
    else return ERROR;
}
status SaveSudokuFile(int re, char filename[])
{
    int i, j, k, x;
    for(i=0; filename[i]!='\n'; i++){
        if(filename[i]=='.'){
```

```
            filename[i+1] = 'r';
            filename[i+2] = 'e';
            filename[i+3] = 's';
        }
    }
    FILE *fp;
    fp = fopen(filename, "w");
    if(!fp) exit(-1);
    if(re==TRUE){        //数独有解
        fprintf(fp, "s 1\n");
        fprintf(fp, "v ");
        for(i=1; i<=9; i++){
            for(j=1; j<=9; j++){
                for(k=1; k<=9; k++){
                    x = i*100 + j*10 + k;
                    if(counter[x].value==1){
                        fprintf(fp, "%d ", x);
                    }
                }
            }
        }
        fprintf(fp, "\n");
    }
    else{                //数独无解
        fprintf(fp, "s 0\n");
    }
    fclose(fp);
    return OK;
}
void print(int a[][COL])
{
    int i, j;
    printf("     1  2  3    4  5  6    7  8  9\n");
    for(i=0; i<ROW; i++){
        if(i%3==0) printf("   --------------------------------\n");
        printf("%d  ", i+1);
```

```
        for(j=0; j<COL ; j++){
            if(j%3==0) printf("| ");
            if(a[i][j]!=0) printf("%d  ", a[i][j]);
            else printf("_  ");
        }
        if(i%3==2) printf("|\n");
        else{
            printf("|\n");
            printf("   |          |          |          |\n");
        }
    }
    printf("   ---------------------------------\n");
}
```