

**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： CS1706**

**学 号： U201714762**

**姓 名： 梁一飞**

**指导教师： 纪俊文**

**报告日期： 2019.3.10**

**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问题求解，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>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, 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 InternationalSymposium 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. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

# 目录

目录

[目录 III](#_Toc3832814)

[1 引言 5](#_Toc3832815)

[1.1课题背景与意义 5](#_Toc3832816)

[1.1.1 课题背景 5](#_Toc3832817)

[1.1.2 课题意义 5](#_Toc3832818)

[1.2 国内外研究现状 5](#_Toc3832819)

[1.3 课程设计的主要研究工作 5](#_Toc3832820)

[2 系统需求分析与总体设计 6](#_Toc3832821)

[2.1 系统需求分析 6](#_Toc3832822)

[2.2 系统总体设计 6](#_Toc3832823)

[3 系统详细设计 9](#_Toc3832824)

[3.1 有关数据结构的定义 9](#_Toc3832825)

[3.2 主要算法设计 10](#_Toc3832826)

[4 系统实现与测试 13](#_Toc3832827)

[4.1 系统实现 13](#_Toc3832828)

[4.1.1 数据结构用C语言定义如下 13](#_Toc3832829)

[4.1.2 各函数功能及实现 14](#_Toc3832830)

[4.2 系统测试 16](#_Toc3832831)

[4.2.1 基本测试 16](#_Toc3832832)

[4.2.2 优化测试 20](#_Toc3832833)

[4.2.3 不满足算例测试 22](#_Toc3832834)

[4.2.4 数独测试 23](#_Toc3832835)

[5.测试结果分析 25](#_Toc3832836)

[5 总结与展望 25](#_Toc3832837)

[5.1 全文总结 25](#_Toc3832838)

[5.2 工作展望 27](#_Toc3832839)

[6 体会 27](#_Toc3832840)

[参考文献 28](#_Toc3832841)

[附录 28](#_Toc3832842)

[1.DPLL.h 28](#_Toc3832843)

[2.SAT.cpp 29](#_Toc3832844)

# 1 引言

## 1.1课题背景与意义

### 1.1.1 课题背景

SAT问题是第一个被证明为NP难的问题，SAT在计算机科学、复杂性理论、密码系统、人工智能等领域发挥着至关重要的作用。然而，促使SAT问题成为持续研究热点的主要原因在于其在现实应用中的重要性。许多包含数以万计变量和数百万约束的组合问题都可运用SAT求解技术处理。数独游戏是一种源自18世纪末的瑞士的游戏，后在美国发展、并在日本得以发扬光大的数学智力拼图游戏。拼图是九宫格的正方形状，每一格又细分为一个九宫格。在每一个小九宫格中，分别填上1至9的数字，让整个大九宫格每一列、每一行的数字都不重复。 数独的玩法逻辑简单，数字排列方式千变万化。不少教育者认为数独是锻炼脑筋的好方法。

### 1.1.2 课题意义

研究基于SAT的数独游戏解法，不仅能快速解决数独问题，也能借此生成数独游戏，为数独游戏爱好者提供方便。在完成课题的过程中，也能使自己的编程以及思维能力得到提升。

## 1.2 国内外研究现状

对于基于SAT的数独游戏解法， 国内外研究出多种算法，可大体分为唯一法、排除法、回溯法这三种方法，其中基于回溯法的DPLL算法可以应用于求解。

## 1.3 课程设计的主要研究工作

本次课程设计，主要研究重点在于： 1.采用合适的数据结构保存cnf文件的数据 2.实现DPLL算法，用DPLL算法求解cnf文件 3.实现数独到cnf文件的转换，并用DPLL算法求解 4.将求解得到的结果导出为res文件 5.设计合适的UI实现用户交互。

其中，主要研究工作还在于如何实现DPLL算法以及如何优化DPLL算法使其能求解更复杂的问题。

------------------------------------------------------------------------------------

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

想要完整的求解基于SAT的数独问题，系统需要包含以下功能，分别是：

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；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。
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，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：**将数独游戏问题转化为SAT问题[，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。

## 2.2 系统总体设计

**1.系统总体设计思路如下**：1.将数独文件转换为cnf文件 2.将cnf文件中的数据读取到设计的数据结构中 3.进行DPLL过程解出结果 4.将结果输出为res文件，流程如图所示：

LoadtoFile

DPLL

LoadinFile

Transfersudo

图1.求解流程

如果不求解数独问题，而是对cnf文件进行求解，则可以省略第一步。为了方便与优化后DPLL算法对比，在系统中引入了优化前的DPLL1和优化后的DPLL2，方便直观对比。

**2.系统设计如图所示**：

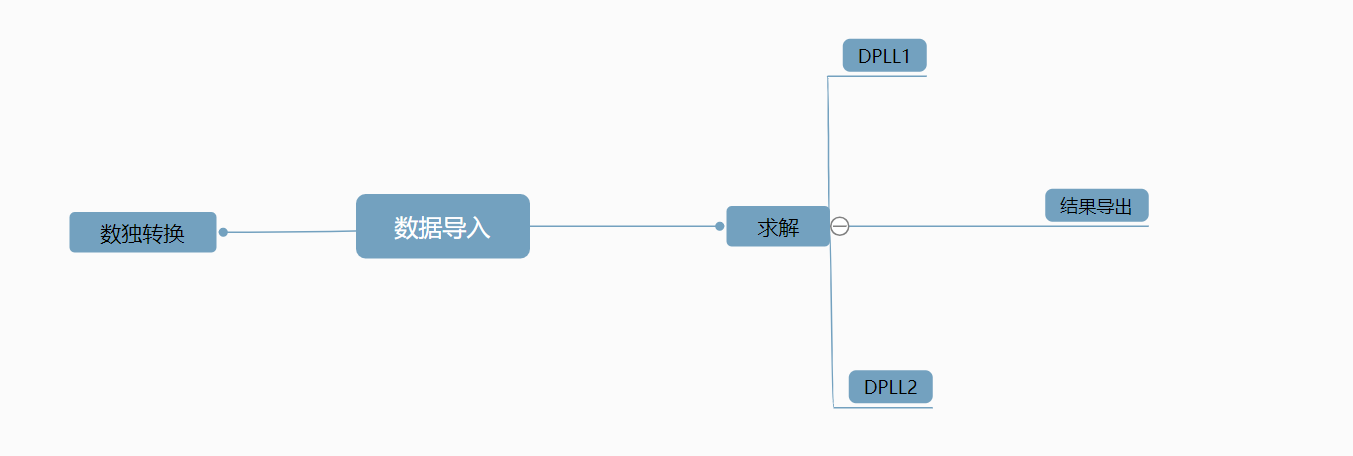


图2.系统设计

在结果导出这里设计了一个分支选项，如果操作的是数独，则会以九宫格形式打印出来并且生成res文件，否则只会生成res文件。

DPLL1和DPLL2都会打印TRUE（有解）或FALSE（无解），以及算法运行所需要的时间。

系统UI界面如图所示：

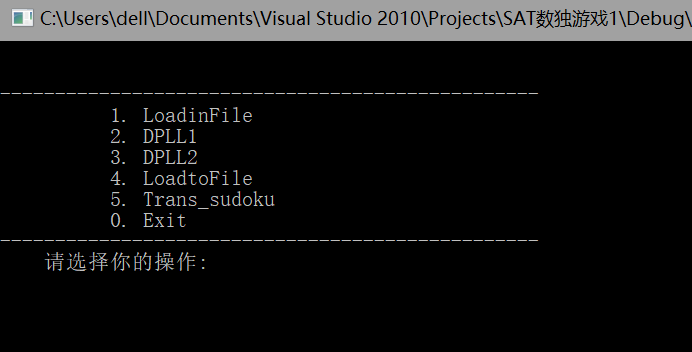


图3.系统界面

另外，为了直观的显示数独游戏，采取九宫格打印输出的办法，输出效果如图所示：



图3.初始数独输出

求解后的结果依旧以九宫格形式输出，输出如图所示：

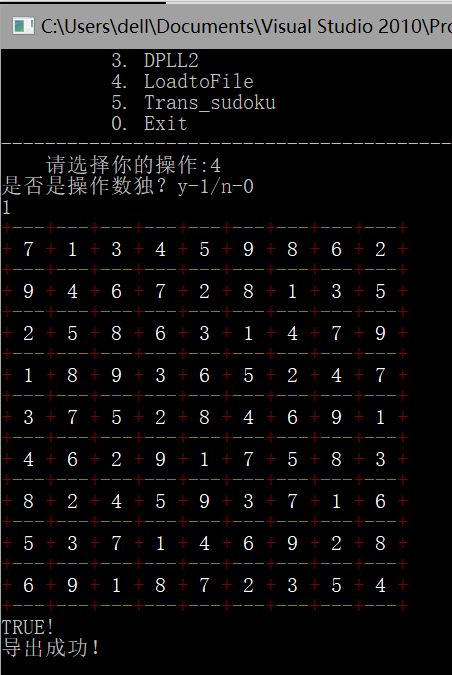


图4.求解数独输出

------------------------------------------------------------------------------------

# 3 系统详细设计

## 3.1 有关数据结构的定义

在每个cnf文件的开始，由‘c’开头的是若干注释说明行；‘p’开头的行说明公式的总体信息，包括：范式为CNF；公式有200个布尔变元，由1到200的整数表示；320个子句。之后每行对应一个子句，0为结束标记。

为了存储cnf文件的数据，设计了如图所示结构来存储：



图5.数据结构示意图

具体来说，即为两个链表（子句链表与变元链表）以及一个根节点。

**1.子句链表**： 内部包含两个指针，PL指向第一个变元，next2指向下一条子句，每一个结点的结构如图所示：



图6.子句结点结构

**2.变元链表**：内部包含一个整型数据Elem：储存变元的值，以及一个指向下一个变元结点的指针next1,每一个结点的结构如图所示：



图7.变元结点结构

**3.根节点**：内部包含一个指针指向第一个子句链表结点，一个整形变量用来储存子句数量，另一个整形变量用来储存变元数量，这是实现DPLL算法的重要数据类型。

**4.其他预处理定义如下**：

#define TRUE 1//用于函数返回值

#define FALSE 0//用于函数返回值

#define DESCRIPTION "p cnf "//用于跳过读取cnf文件时遇到的p cnf

#define LENGTH 1000//用于设置数组长度

## 3.2 主要算法设计

**1.数独转换成cnf文件**：

将数独转换成cnf文件，首先明确变元个数，由于九宫格数独总共有81个空，而每个空可以填入1~9，故总共应有729个变元。这729个变元应满足以下五个规则：1.每行9个变元不同 2.每列9个变元不同 3.每点至少一个解 4.每点至多一个解 5.每个小九宫格9个变元不同。满足这5个规则，则子句应有81+324\*27=8829条。数独的初始数据由txt文件直接导入，格式为1~81格，未填的数为0，填好的为所填的数。算法流程如下图所示：

数独文件

行规则

列规则

点规则1

点规则2

九宫格规则

Cnf文件

图8.transsudo流程

算法时间复杂度：O(1)

**2.将cnf文件数据导入数据结构**：

因为cnf文件开始几行为注释，所以要先跳过注释，在读到p cnf时停下，后面第一个数值为变元个数，第二个数值为子句个数，用两个数组记录这两个数，然后输入到根节点中储存起来。接下来分配第一个子句的空间，然后开始记录变元值，直到读取到字符’0’，开始创建下一个子句，依次循环，直到记录数等于子句数为止。算法流程如图所示：

跳过注释

读取子句数n，变元数m

k=n

创建完成

创建下一条子句

k=0,k++;

k<n

读到’0’

循环创建变元结点

图9.LoadinFile流程

时间复杂度：O（m\*n）

**3.DPLL1**：

这是未优化前的算法，具体过程为，首先循环删除数据结构中的所有单子句。在所有单子句删除完毕后，检查子句集，如果子句集为空，则说明有解，如果子句集含空子句，则说明无解。否则进入下一步，此时子句集中不含单子句，于是选取第一条子句的第一个元素，创建一条单子句加到根节点后面，使其成为第一条子句。然后继续循环删除，直到子句集为空或者子句集含空子句为止，这一步采用递归的方式实现。算法流程图如下：

删除子句集中所有单子句所含的正负变元

子句集不为空且不含空子句

将第一条子句的第一个元素添加为单子句

子句集为空 子句集含空子句

FALSE

TRUE

图10.DPLL1流程

**4.DPLL2**：

优化后的算法，与DPLL1的主要区别是，在没有单子句后，选取的是最短子句的第一个元素添加为单子句，这样优化的根据是这种策略可以更快的得到新的原子句集所含的单子句，节省添加单子句的效率，但是对于某些特殊情况不能做到完全优化，比如第一个单子句所含的第一个元素可以删除大多数子句中的变元。算法流程如图所示：

删除子句集中所有单子句所含的正负变元

子句集不为空且不含空子句

将最短子句的第一个元素添加为单子句

子句集为空 子句集含空子句

FALSE

TRUE

图11.DPLL2流程

**5.将结果导出为res文件：**

在DPLL算法运行时，设置了一个整形指针（即数组）记录下了每个变元的真假，真则为1，假则为0，如变元40即便是c[39]=1；在输出文件时，如果操作的是数独文件，则会以九宫格形式输出数独，否则则以普通形式输出解，即每个变元后有1或0表示真或假。输出文件时，将导入的cnf文件的文件名后三位，即后缀.cnf改为.res，如果cnf文件有解，则输出s 1;v 解;t 时间。若无解，则输出 s 0;v;t 时间。算法流程如图所示：

选择操作类型

直接打印

非数独

数独

输出为res文件

打印九宫格

图12.LoadtoFile流程

------------------------------------------------------------------------------------

# 4 系统实现与测试

## 4.1 系统实现

### 4.1.1 数据结构用C语言定义如下

**1.变元链表：**

typedef struct Node1{

int Elem;

struct Node1 \*next1;

}Node1;

**2.子句链表：**

typedef struct Node2{

Node1 \*pL;

struct Node2 \*next2;

}Node2;

**3.根节点：**

typedef struct root{

Node2 \*p;

int NumOfClause;//子句数

int NumOfVari;//变元数

}root;

### 4.1.2 各函数功能及实现

**1.导入cnf文件：**

函数定义为：int LoadinFile(char filename[], root \*head);

具体实现：在3.2里已叙述

**2.判断单子句：**

函数定义为：int IsUnitClause(root head, int \*Elem);

具体实现：传入根节点和一个整形指针，循环遍历整个数据结构，在碰到第一个单子句时，返回单子句所在的位置，整形指针记录该单子句所含的变元值。

时间复杂度：O（n）

**3.销毁子句：**

函数定义为int Destroy(Node2 \*p, root \*head);

具体实现：传入一个子句指针，设置两个指针进行遍历和删除，删除这个指针中的所有元素并释放该指针所占的空间，最后将根节点的子句数减一。

时间复杂度：O（n）

**4.销毁根节点**

函数定义为：int DestroyHead(root \*head);

具体实现：传入根节点，设置两个指针进行遍历和删除，且在子句指针移动时使用销毁子句功能销毁子句，然后移动到下一子句进行销毁，直到销毁所有子句,最后释放根节点所占空间。

时间复杂度：O（m\*n）

**5.删除子句**

函数定义为：int RemoveClause(root \*head, int Elem);

具体实现：传入根节点和要删除的变元Elem,Elem由判断单子句功能得到，设置4个指针进行遍历和删除。首先设置两层循环，即，循环子句链表和变元链表，找到变元等于Elem的结点进行删除，直至销毁所有的Elem变元结点，然后再次循环，销毁所有变元等于-Elem的结点进行删除。如果首子句即为含Elem的单子句，则删除时要另外注意将根节点指针指向下一条子句以免出错。删除功能采用销毁子句功能销毁指针实现。

时间复杂度：O（）

**6.添加子句**

函数定义为：root\* AddClause(root \*head, int VariElem)

具体实现：先定义一个新的根节点Newhead，然后设置2个指针创建新的单子句，并将Newhead第一个指针指向新的单子句，然后设置新的4个指针遍历复制原根节点，接在新的单子句后面，返回Newhead，这样，便使原根节点添加了一条新的单子句。

时间复杂度:O(m\*n)

**7.判断空子句**

函数定义为：int IsEmptyClause(root \*head)

具体实现：设置一个子句指针进行循环遍历所有子句，如果有某一子句的变元指针指向NULL，则说明是空子句，返回0。否则，返回1。

时间复杂度：O（n）

**8.判断是否删除结束**

函数定义为：int IsOver(root \*head)

具体实现：传入根节点，如果根节点的指针p指向NULL，则删除结束，返回1.否则，返回0.

时间复杂度：O（1）

**9.DPLL1**

函数定义为：int DPLL1(root \*head)

具体实现：在3.2里已叙述

时间复杂度：O（2^n）

**10.DPLL2**

函数定义为：int DPLL1(root \*head)

具体实现：在3.2里已叙述

时间复杂度：O（2^n）

**11.将数独转化成cnf文件**

函数定义为：int transudo(char filename[])

具体实现：在3.2里已叙述

**12.将结果导出为res文件**

这一功能没有单独定义函数，而是直接在主函数里实现，具体实现在3.2里已叙述。

**13.各函数调用关系如图所示:**

DPLL1&DPLL2

Y

N

way1 way2

RemoveClause

IsUnitClauseuse

Reback

AddClause

IsOver

IsEmptyClasue

TRUE

FALSE

图13.函数关系调用图

## 4.2 系统测试

### 4.2.1 基本测试

首先对给定的基准样例进行测试，测试所需的功能包括：将cnf文件数据导入数据结构、运用DPLL算法解出cnf文件、将解得结果导出为res文件。

基准算例包括一个满足算例、一个不满足算例、以及两个性能测试算例。下面用一个表格给出测试结果。

满足算例：sat-20.cnf 变元：20 子句：91

不满足算例：unsat-5cnf-30.cnf 变元：30 子句：420

性能测试1：ais10.cnf 变元：181 子句：3151

性能测试2：sud00009.cnf 变元：303 子句：2851

表1.基准算例测试1

|  |  |  |
| --- | --- | --- |
| 算例 | sat-20.cnf | unsat-5cnf-30.cnf |
| 文件导入 |  |  |
| D  P  LL |  |  |
| 文件导出 |  |  |

表2.基准算例测试2

|  |  |  |
| --- | --- | --- |
| 算例 | ais10.cnf | sud00009.cnf |
| DPLL1 |  |  |
| DPLL2 |  |  |
| 优化率 | 96.9% | 63.4% |

下面进行数独求解测试。首先导入数独文件，数独由txt文件直接导入，如图所示：

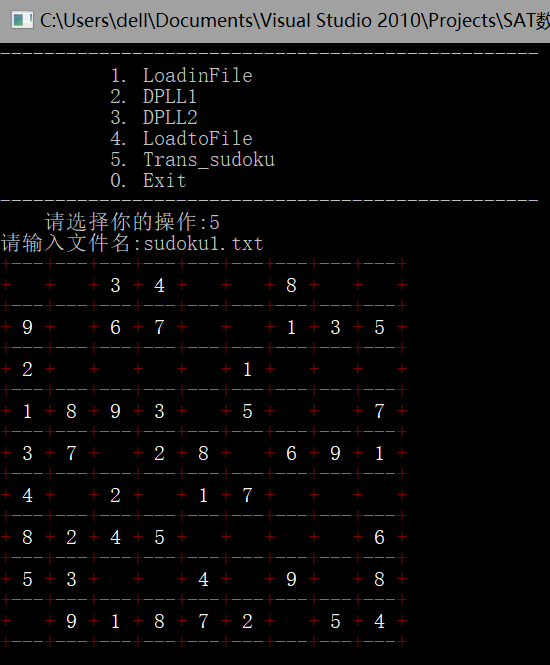
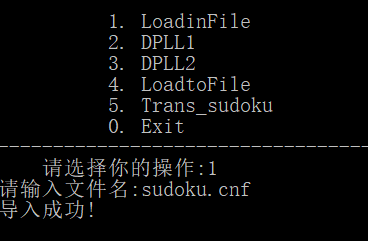
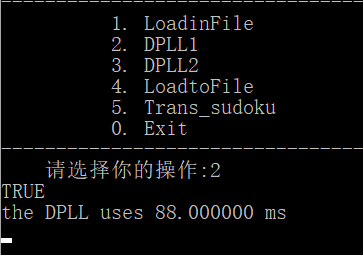


图14.导入数独文件

接下来进行求解，求解步骤如下图所示：





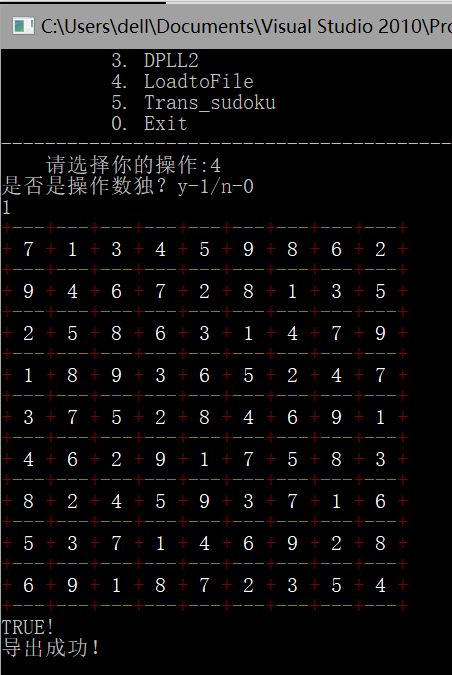


图15.数独求解

由最后生成的九宫格，可直观的看出解答是正确的。由此，所有的基本功能均可实现。下面进行大量算例测试以及优化对比。

### 4.2.2 优化测试

表3.测试对比表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 算例 | 是否满足 | DPLL1 | DPLL2 | 优化率 |
| 1 | problem1-20.cnf | 是 |  |  | 0 |
| 2 | problem2-50.cnf | 是 |  |  | 81.8% |
| 3 | problem3-100.cnf | 是 |  |  | -7.3% |
| 4 | problem6-50.cnf | 是 |  |  | 50% |
| 5 | problem8-50.cnf | 是 |  |  | -45.5% |
| 6 | problem9-100.cnf | 是 |  |  | 93.8% |
| 7 | problem11-100.cnf | 是 |  |  | -63.5% |
| 8 | tst\_v25\_c100.cnf | 是 |  |  | 100% |
| 9 | bart17.shuffled-231.cnf | 是 |  |  | -13.9% |
| 10 | sud00012.cnf | 是 |  |  | -62.5% |
| 11 | sud00021.cnf | 是 |  |  | 93.0% |
| 12 | sud00079.cnf | 是 |  |  | 40.3% |
| 13 | problem5-200.cnf | 是 |  | 无 | 0% |
| 14 | sud00082.cnf | 是 |  |  | -58% |
| 15 | sud00861.cnf | 是 |  |  | 60.2% |
| 16 | tst\_v200\_c210.cnf | 是 |  | 无 | 0% |
| 17 | ais8.cnf | 是 |  |  | 62.9% |
| 18 | problem12-200.cnf | 是 | 无 |  | 100% |
| 19 | sud00001.cnf | 是 |  |  | -20.2% |
| 20 | sud00009.cnf | 是 |  |  | 62.6% |

### 4.2.3 不满足算例测试

表4.不满足算例测试

|  |  |  |
| --- | --- | --- |
| 序号 | 算例 | 测试结果 |
| 1 | tst\_v10\_c100.cnf |  |
| 2 | u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf |  |
| 3 | u-problem7-50.cnf |  |
| 4 | u-problem10-100.cnf |  |
| 5 | unsat-5cnf-30.cnf |  |

### 4.2.4 数独测试

表5.数独测试

|  |  |  |
| --- | --- | --- |
| 序号 | 求解前 | 求解后 |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

### 5.测试结果分析

对于优化对比测试，可以看出优化后的DPLL2算法在大多数cnf文件上为正优化，少部分文件会出现负优化的情况，在另一部分文件上优化效率甚至超过了90%，可以算是一个合适的优化算法。在不满足问题测试上，测试结果均为FALSE，可看出对于不满足问题，DPLL1算法同样可靠，在这里我用DPLL2算法分析对比了解答利率，同样呈现大部分正优化，少数负优化的情况，故没有再缀述。对于数独问题的解答正确性，可以通过打印的九宫格直观的判断，可以看出解出来后的数独九宫格满足行规则，列规则，九宫格规则，以及单解。综合以上测试结果，可以看出各函数功能均是正确的。

------------------------------------------------------------------------------------

# 5 总结与展望

## 5.1 全文总结

在本次课程设计中，我主要做了如下工作：

**1.分析设计数据结构**：首先观察cnf文件的数据类型，包括子句和变元，变元包括在每一条子句当中。而子句是一条连一条的，所以在数据结构的设计上，我采用了两个链表来储存子句，和子句所包含的变元，而为了函数操作方便，则必须再设计一个根节点，根据cnf文件的数据，我设计的根节点包括子句数量和变元数量，以及一个指向第一个子句的指针，这样便能很好的被函数调用，实现相应的功能。

**2.设计多个函数，最终实现DPLL算法：**DPLL算法的核心在于单子句化简，所以首先要设计判断单子句以及返回单子句信息的函数，返回后要进行删除化简，所以需要设计删除单子句的函数，而DPLL算法的难点在于单子句删除完毕后，如何继续进行化简，当单子句删除完毕后，我的想法是必须添加单子句，所以此时设计了添加单子句的函数。但是以什么策略添加单子句也是一个难点，我采用了将第一条子句的第一个元素添加为单子句，于是解决了这个问题。之后再设计两个判断TRUE和FALSE的函数，就实现了DPLL算法。

**3.优化DPLL算法**：经过分析，我认为优化DPLL算法的关键在于单子句删除完毕后，采用何种策略添加单子句上。因为DPLL算法核心在于单子句化简，所以需要尽可能快的产生新的单子句，所以我将选取第一条子句的第一个变元作为新的单子句改为了选取最短子句的第一个变元作为单子句，这样便能快速的产生新的单子句，进而化简效率更高，在实际测试中，这种方式确实能优化解决大多数cnf文件，但也有少部分cnf文件出现了负优化的情况，经过分析得知，虽然这种方式能更快的产生单子句，但是对于某些特殊文件，删除效率却不如直接将第一条子句的第一个元素添加为单子句快，但是能做到多数文件的优化，说明优化还是成功的。

**4.将数独转换为cnf文件**：在这一步设计时，首先解决的是变元问题，因为数独每一个可以填9个数，而总共有81格，所以变元数应该有729个。问题在于这729个变元需要满足什么条件才能得到正确的81个解，在查阅数独游戏的规则后，首先我明确了三个约束条件：1.每行9个数字不同 2.每列9个数字不同 3.每个9宫格9个数字不同。 但是光这三个约束条件似乎不够，在运行中出现了有的点多解和无解的情况。所以此时还需要两个约束条件：1.每点至多一个解 2.每点至少一个解。于是通过这5个约束条件，数独顺利的转换成了可解的cnf文件。

## 5.2 工作展望

虽然本次的课程设计顺利的完成了，但是却不够完美，还有以下几点可以继续改进和优化：

**1.DPLL算法的优化：**虽然在改进选取策略后，算法得到了优化，但是还远远不够，许多大型算例还是无法求解出来，主要原因还是因为采取了递归的实现方式，所以今后的研究方向在于如何用非递归的方式实现DPLL算法，在查阅了相关资料后，发现可以用压栈的方式实现，这将是我以后研究改进的方向

**2.数独游戏：**这一部分可以继续进步的空间就更大了，首先我的数独是直接导入的，所以今后的工作可以尝试用挖洞法自己生成，另外我的数独没有可玩性，只是导入后直接解出来，所以下一步的工作需要实现用户交互的可玩性，再就是UI界面不够美观，还可以继续优化。

**3.数据结构的改进：**由于设计的数据结构有两个链表，每次遍历都需要m\*n的时间复杂读，所以今后的工作方向应该考虑如何改进这个数据结构，使得遍历需要的时间复杂度更小，以提高算法的时间效率。

-----------------------------------------------------------------------------------

# 6 体会

本次实验过程中，最主要的就是要时刻检查，因为实现DPLL算法需要调用多个函数，每一步都不能出错，所以每次写完一个函数后都要检查一遍，在写完导入cnf文件后，我又写了一个打印链表的函数来检查是否成功导入了cnf文件，而在写删除和添加子句时，也要不时打印一遍检查删除和添加是否正确，就这样一步一脚印的写完了DPLL，在DPLL算法成功运行的那一刻是有很大的满足感的，因为之前的调试和检查花了大量的时间和精力。

在输出数独时，也遇到了一个问题，就是输出的解答之后的数独结果虽然正确，但不是原来的数独了，是一个新的数独，这就让我百思不得其解，在仔细观察了数独转换后的cnf文件，以及解答输出的res文件，我发现这两者都没有错误，就是在打印的时候出错了，在仔细对照后，我发现所有格子中的元素都比正确的解答少了1，然后找到了问题所在，因为是729个变元，所以应该是在1~729间循环，找到正确的解之后%9打印出来，而由于个人编程习惯的问题，我是在0~728之间循环，虽然也循环了729次，但是会导致结果比正确结果少1，于是我成功的解决了这个问题。但是在解决问题的过程中间，我走了很大的弯路，刚开始看到结果不对后，我以为是自己的DPLL和转换数独出现了问题，甚至一度想推倒重来，多亏了冷静下来仔细分析输出的文件，才找到了问题所在，问题很小，但是差点让我将问题严重化了，这告诫我以后在写代码的过程中，出了问题一定要冷静分析，小心调试，不要轻易的就想推倒重来，增加工作量。

-----------------------------------------------------------------------------------

# 参考文献

[1]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.2009.11

[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>

-----------------------------------------------------------------------------------

# 附录

# 1.DPLL.h

#define TRUE 1

#define FALSE 0

#define DESCRIPTION "p cnf "

#define LENGTH 1000

typedef struct Node1{

int Elem;

struct Node1 \*next1;

}Node1;//句链表

typedef struct Node2{

Node1 \*pL;

struct Node2 \*next2;

}Node2;//行链表

typedef struct root{

Node2 \*p;

int NumOfClause;

int NumOfVari;

}root;//用于操作的根节点

int LoadinFile(char filename[], root \*head);

int Destroy(Node2 \*p, root \*head);

int IsUnitClause(root head, int \*Elem);

int RemoveClause(root \*head, int Elem);

root\* AddClause(root \*head, int VariElem);

int IsEmptyClause(root \*head);

int IsOver(root \*head);

int DestroyHead(root \*head);

int DPLL1(root \*head);

int DPLL2(root \*head);

int transudo(char filename[]);

int sudos(int x,int y,int d);

# 2.SAT.cpp

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#include "DPLL.h"

int \*f;

int tmin;

int sudo[81];

int main(){

int op=1;

int a;

root head;

clock\_t start, end;

double duration;

char filename1[1000];

char filename2[1000];

//LoadinFile(filename1, &head);

while(op){

system("cls"); printf("\n\n");

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

printf(" 1. LoadinFile \n");

printf(" 2. DPLL1\n");

printf(" 3. DPLL2\n");

printf(" 4. LoadtoFile\n");

printf(" 5. Trans\_sudoku\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作:");

scanf("%d",&op);

switch(op){

case 1:

printf("请输入文件名:");

scanf("%s",filename1);

LoadinFile(filename1, &head);

printf("导入成功!\n");

getchar();getchar();

break;

case 2:

start = clock();

if (a=DPLL1(&head))

printf("TRUE\n");

else

printf("FALSA\n");

end = clock();

duration = (double)(end - start) ;

printf("the DPLL uses %f ms\n", duration);

getchar();getchar();

break;

case 3:

start = clock();

if (a=DPLL2(&head))

printf("TRUE\n");

else

printf("FALSA\n");

end = clock();

duration = (double)(end - start) ;

printf("the DPLL uses %f ms\n", duration);

getchar();getchar();

break;

case 4:

if (a == TRUE) {

int h;

printf("是否是操作数独？y-1/n-0\n");

scanf("%d",&h);

if(h==0){

for (int t = 0; t < head.NumOfVari; t++) {

printf("%2d %d\n", \*(f + t),t+1);

}

}

else if(h==1){

int d,o,l,k;

k=0;

d=0;

for (int t = 0; t < head.NumOfVari; t++){

if(\*(f+t)==1){

if((t+1)%9==0)

sudo[d]=9;

else

sudo[d]=(t+1)%9;

d++;

}

}

for(o=0;o<19;o++){

if(o%2==0){

for(l=0;l<37;l++){

if(l%4==0)

printf("\033[;31m+\033[0m");

else

printf("\033[;33m-\033[0m");

}

printf("\n");

}

else{

for(l=0;l<37;l++){

if(l%4==0&&l!=36){

printf("\033[;31m+\033[0m");

printf("\033[;;1m %d \033[0m",sudo[k]);

k++;

}

if(l==36)

printf("\033[;31m+\033[0m");

}

printf("\n");

}

}

}

printf("TRUE!\n");

int length = strlen(filename1);

filename1[length - 1] = 's';

filename1[length - 2] = 'e';

filename1[length - 3] = 'r';

FILE \*fp;

if ((fp = fopen(filename1, "w")) == NULL) {

printf("File open error!\n");

break;

}

fprintf(fp, "s 1\n");

fprintf(fp, "v");

for (int t = 0; t < head.NumOfVari; t++) {

if(\*(f+t)==1)fprintf(fp," %d",t+1);

else fprintf(fp, " %d", -t - 1);

}

fprintf(fp, "\n");

fprintf(fp, "t %f ms\n", duration);

fclose(fp);

}

else {

printf("FALSE!\n");

int length = strlen(filename1);

filename1[length - 1] = 's';

filename1[length - 2] = 'e';

filename1[length - 3] = 'r';

FILE \*fp;

if ((fp = fopen(filename1, "w")) == NULL) {

printf("File open eror!\n");

break;

}

fprintf(fp, "s 0\n");

fprintf(fp, "v\n");

fprintf(fp, "t %f ms\n", duration);

fclose(fp);

}

printf("导出成功！\n");

memset(f,0,sizeof(f));

getchar();getchar();

break;

case 5:

printf("请输入文件名:");

scanf("%s",filename2);

FILE \*f2;

f2=fopen(filename2,"r");

if(f2==NULL){

printf("wrong!\n");

break;

}

int i,o,l,k;

k=0;

for(i=0;i<81;i++){

sudo[i]=fgetc(f2)-48;

}

for(o=0;o<19;o++){

if(o%2==0){

for(l=0;l<37;l++){

if(l%4==0)

printf("\033[;31m+\033[0m");

else

printf("\033[;33m-\033[0m");

}

printf("\n");

}

else{

for(l=0;l<37;l++){

if(l%4==0&&l!=36){

printf("\033[;31m+\033[0m");

if(sudo[k]==0)

printf(" ");

else

printf("\033[;;1m %d \033[0m",sudo[k]);

k++;

}

if(l==36)

printf("\033[;31m+\033[0m");

}

printf("\n");

}

}

transudo(filename2);

getchar();getchar();

break;

case 0:

break;

return 0;

}

}

}

int Destroy(Node2 \*p, root \*head){

Node1 \*temp1, \*temp2;

for (; p->pL != NULL;){

for (temp1 = p->pL, temp2 = p->pL; temp1->next1 != NULL; temp1 = temp1->next1)

temp2 = temp1;

if (temp1 ==p->pL){

p->pL = NULL;

free(temp1);

}

else

{

free(temp1);

temp2->next1 = NULL;

}

}

free(p);

head->NumOfClause--;

return 0;

}//传入一个子句指针，回收该子句及其所有文字

int IsUnitClause(root head, int \*Elem){

int i;

Node2 \*temp2;

for (temp2 = head.p, i = 1; temp2 != NULL; temp2 = temp2->next2, i++){

if (temp2->pL->next1 == NULL)

{

\*Elem = temp2->pL->Elem;

return i;

}

}

return i = 0;

}//传入一个cnf公式，和一个整数指针，若有单子句则返回第一个单子句的位置和该单子句文字的值，否则返回0

int RemoveClause(root \*head, int Elem){

int i, j, k;//i标识子句是否是首子句，j标识当前子句是否含L，i为0标识首子句含L，i为1标识首子句不含L

Node1 \*temp11, \*temp12;

Node2 \*temp21 = head->p, \*temp22 = head->p;

for (i = 1; temp21 != NULL; i++){//控制子句循环

j = 1;

for (temp11 = temp21->pL, temp12 = temp21->pL, k = 1; temp11 != NULL; k++){//循环子句中的每一个文字

if (temp11->Elem == Elem){//若存在单子句规则要简化的文字L，则简化

if (i == 1){//若第一个子句含有L

head->p = temp21->next2;

Destroy(temp21, head);

i--;//使i为0，更新第一个子句

}

else{//其他子句含有L

temp22->next2 = temp21->next2;

Destroy(temp21, head);

i++;

}

j = 0;

break;//跳出第二重循环

}

if (temp11->Elem == -Elem){

if (k == 1){

temp21->pL = temp11->next1;

free(temp11);

temp11 = temp21->pL;

temp12 = temp21->pL;

k--;

}

else{

temp12->next1 = temp11->next1;

free(temp11);

temp11 = temp12->next1;

}

continue;

}

temp12 = temp11;

temp11 = temp11->next1;

}

if (i == 0)

{

temp21 = head->p; temp22 = head->p;

}

else if (i == 1 || (i>1 && j == 0))

temp21 = temp22->next2;

else{

temp22 = temp21;

temp21 = temp22->next2;

}

}

return 0;

}//传入一个root型和一个elem，删除root中所有包含elem文字的子句

root\* AddClause(root \*head, int VariElem){

Node2 \*tempCp, \*tempCp2, \*tempCp3;

Node1 \*tempLp, \*tempLp2, \*tempLp3;

root \* NewHead;

int i;

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

NewHead->NumOfClause = head->NumOfClause + 1;

NewHead->NumOfVari = head->NumOfVari;

tempLp = (Node1\*)malloc(sizeof(Node1));

tempCp = (Node2\*)malloc(sizeof(Node2));

tempCp->pL = tempLp;

tempLp->next1 = NULL;

tempLp->Elem = VariElem;

NewHead->p = tempCp;

for (tempCp2 = head->p, tempCp3 = tempCp; tempCp2 != NULL; tempCp2 = tempCp2->next2){

tempCp = (Node2\*)malloc(sizeof(Node2));

tempCp3->next2 = tempCp;

for (tempLp2 = tempCp2->pL, i = 1; tempLp2 != NULL; tempLp2 = tempLp2->next1){

if (i == 1){

tempLp = (Node1\*)malloc(sizeof(Node1));

tempCp->pL = tempLp;

tempLp3 = tempLp;

tempLp->Elem = tempLp2->Elem;

i++;

}

else{

tempLp3 = tempLp;

tempLp = (Node1\*)malloc(sizeof(Node1));

tempLp3->next1 = tempLp;

tempLp->Elem = tempLp2->Elem;

}

}

tempLp->next1 = NULL;

tempCp3 = tempCp;

}

tempCp->next2 = NULL;

return NewHead;

}//添加子句

int IsEmptyClause(root \*head){//若含空子句，则返回0；否则返回1

Node2 \*tempCp;

for (tempCp = head->p; tempCp != NULL; tempCp = tempCp->next2)

if (tempCp->pL == NULL)

return 0;

return 1;

}

int IsOver(root \*head){

if (head->p == NULL)

return 1;

return 0;

}//是否删除结束

int DestroyHead(root \*head){

Node2 \*tempCp, \*tempCp2;

for (tempCp = head->p; tempCp != NULL; tempCp = tempCp2){

tempCp2 = tempCp->next2;

Destroy(tempCp, head);

}

free(head);

return 0;

}

int LoadinFile(char filename[], root \*head){

FILE \* file;

char str[LENGTH], stemp[LENGTH], ch = '0';

int i, j, k, l, m;//临时变量

Node1 \*temp1;

Node2 \*temp2;

file = fopen(filename, "r");

if (file == NULL)

{

printf("wrong!\n");

return FALSE;

}

for (; fgets(str, LENGTH, file);)//跳过注释，读到"p cnf "

if (strstr(str, DESCRIPTION) != NULL)

break;

for (i = 6; str[i] != ' '; i++)

stemp[i - 6] = str[i];

stemp[i - 6] = '\0';//stemp记录变元数

for (m = i; str[i] != '\0'; i++)

str[i - m] = str[i];

str[i - m] = '\0';//str记录子句数

head->NumOfVari = atoi(stemp);//读取变元数，atoi(字符串转整形）

f=(int\*)malloc(sizeof(int)\*atoi(stemp));

memset(f,0,atoi(stemp));

head->NumOfClause = atoi(str);//读取子句数

head->p = (Node2\*)malloc(sizeof(Node2));//分配第一个子句

temp2 = head->p;

fgets(str, LENGTH, file);

ch = '0';

for (i = 0, j = 0, l = 0; str[i] != '0' || ch != ' '; i++){//i控制当前子句字符位置,j控制当前行位置，l判断是否为第一行,此for循环创建第一个子句

if (str[i] != ' ')

stemp[j++] = str[i];

else{

stemp[j] = '\0';

if (l == 0){

temp1 = (Node1\*)malloc(sizeof(Node1));

temp2->pL = temp1;

temp1->Elem = atoi(stemp);

l++, j = 0;

}

else{

temp1->next1 = (Node1\*)malloc(sizeof(Node1));

temp1 = temp1->next1;

temp1->Elem = atoi(stemp);

j = 0;

}

}

ch = str[i];

}

temp1->next1 = NULL;//将最后一行的指针链向空

for (k = 1; k < head->NumOfClause; k++){//k控制子句数目

temp2->next2 = (Node2\*)malloc(sizeof(Node2));

temp2 = temp2->next2;

fgets(str, LENGTH, file);//读当前子句字符串

ch = '0';//置前一个字符为‘0’

for (i = 0, j = 0, l = 0; str[i] != '0' || ch != ' '; i++){//i控制当前子句字符位置,j控制当前行位置，l判断是否为第一行,此for循环创建第一个子句

if (str[i] != ' ')

stemp[j++] = str[i];

else{

stemp[j] = '\0';

if (l == 0){

temp1 = (Node1\*)malloc(sizeof(Node1));

temp2->pL = temp1;

temp1->Elem = atoi(stemp);

l++, j = 0;

}

else{

temp1->next1 = (Node1\*)malloc(sizeof(Node1));

temp1 = temp1->next1;

temp1->Elem = atoi(stemp);

j = 0;

}

}

ch = str[i];

}

temp1->next1 = NULL;//将最后一行的指针链向空

}

temp2->next2 = NULL;//将最后一个子句的指针链向空

fclose(file);

return 0;

}

int DPLL1(root \*head){

int e;

root \*p;

while (IsUnitClause(\*head, &e) != 0){

RemoveClause(head, e);

if(e>0)

\*(f+e-1)=TRUE;

else

\*(f-e-1)=FALSE;//文件输出所用

if (IsEmptyClause(head) == 0)

return FALSE;

if (IsOver(head) == 1)

return TRUE;

}

if (e = DPLL1(p = AddClause(head, head->p->pL->Elem))){

DestroyHead(p);

return TRUE;

}

if (e == 0)

DestroyHead(p);

e = DPLL1(p = AddClause(head, -head->p->pL->Elem));

DestroyHead(p);

return e;

}

int DPLL2(root \*head){

int e;

int ifmin=99;

int min;

Node1 \*m;

Node2 \*n;

root \*p;

while (IsUnitClause(\*head, &e) != 0){

RemoveClause(head, e);

if(e>0)

\*(f+e-1)=TRUE;

else

\*(f-e-1)=FALSE;//文件输出所用

if (IsEmptyClause(head) == 0)

return FALSE;

if (IsOver(head) == 1)

return TRUE;

}

for (n=head->p; n != NULL;n=n->next2){//控制子句循环

min=0;

for (m = n->pL; m != NULL;m=m->next1){//循环子句中的每一个文字

min=m->Elem;

}

if (min<ifmin){

ifmin=min;

tmin=n->pL->Elem;

}

}

if (e = DPLL2(p = AddClause(head,tmin))){

DestroyHead(p);

return TRUE;

}

if (e == 0)

DestroyHead(p);

e = DPLL2(p = AddClause(head, -tmin));

DestroyHead(p);

return e;

}

int sudos(int x,int y,int d){

return ((x-1)\*9 +y-1)\*9 +d;

}

int transudo(char filename[]){

int clause = 0;

clause += 81+324\*27;

FILE \*fp\_out;

fp\_out = fopen("sudoku.cnf","w");

fprintf(fp\_out,"p cnf 729 %d\n",clause);

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

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

if(sudo[(i-1)\*9 +j-1] == 0){

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

fprintf(fp\_out,"%d ",sudos(i,j,d));

fprintf(fp\_out,"0\n");

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

for(int l = d+1;l <=9;l++)

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(i,j,l));

}

else{

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

if(sudo[(i-1)\*9 +j-1] != d)

fprintf(fp\_out, "%d 0\n",-1\*sudos(i,j,d));

else

fprintf(fp\_out, "%d 0\n",sudos(i,j,d));

}

}//行列不同

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

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

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

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

fprintf(fp\_out,"%d %d 0\n",-1\*sudos(i,j,d), -1\*sudos(i,k,d));

}//每点至少一个

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

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

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

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

fprintf(fp\_out,"%d %d 0\n",-1\*sudos(j,i,d), -1\*sudos(k,i,d));

}//每点至多一个

//以下为九宫格不同

for(int i = 1;i <= 3;i++){

for(int j = 1;j <=3;j++)

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

for(int l = 1;l <=3;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

for(int i = 1;i <= 3;i++){

for(int j = 4;j <=6;j++)

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

for(int l = 4;l <=6;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

for(int i = 1;i <= 3;i++){

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

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

for(int l = 7;l <=9;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

for(int i = 4;i <= 6;i++){

for(int j = 1;j <=3;j++)

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

for(int l = 1;l <=3;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

for(int i = 4;i <= 6;i++){

for(int j = 4;j <=6;j++)

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

for(int l = 4;l <=6;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

for(int i = 4;i <= 6;i++){

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

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

for(int l = 7;l <=9;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

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

for(int j = 1;j <=3;j++)

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

for(int l = 1;l <=3;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

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

for(int j = 4;j <=6;j++)

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

for(int l = 4;l <=6;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

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

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

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

for(int l = 7;l <=9;l++){

if((i-1)\*3 +j >= (k-1)\*3 +l )

continue;

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

fprintf(fp\_out,"%d %d0\n",-1\*sudos(i,j,d),-1\*sudos(k,l,d));

}

}

fclose(fp\_out);

return 0;

}