

**课程设计报告**

**题目：基于SAT的二进制数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**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：**将二进制数独游戏[5，6]问题转化为SAT问题[6]，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[6-9]。(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]TanbirAhmed.An Implementation of the DPLL Algorithm.Masterthesis,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]CarstenSinz.Visualizing SAT Instances and Runsof the DPLL Algorithm.JAutom Reasoning (2007) 39:219–243

[5] Binary Puzzle：<http://www.binarypuzzle.com/>

[6] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Mathematics in Computer Science,(2017) 11:515–526

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

[8]InsLynce and JolOuaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

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

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

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

目 录

[1 引言 1](#_Toc51182776)

[1.1课题背景与意义 1](#_Toc51182777)

[1.2 国内外研究现状 1](#_Toc51182778)

[1.3 课程设计的主要研究工作 1](#_Toc51182779)

[1.3.1问题简介 1](#_Toc51182780)

[1.3.2主要内容 2](#_Toc51182781)

[2系统需求分析与总体设计 3](#_Toc51182782)

[2.1系统需求分析 3](#_Toc51182783)

[2.2系统总体设计 3](#_Toc51182784)

[3 系统详细设计 4](#_Toc51182785)

[3.1 有关数据结构的定义 4](#_Toc51182786)

[3.2 主要算法设计 6](#_Toc51182787)

[4 系统实现与测试 7](#_Toc51182788)

[4.1 系统实现 7](#_Toc51182789)

[4.2 系统测试 8](#_Toc51182790)

[5 总结与展望 14](#_Toc51182791)

[5.1 全文总结 14](#_Toc51182792)

[5.2 工作展望 14](#_Toc51182793)

[6 体会 15](#_Toc51182794)

[参考文献 16](#_Toc51182795)

# 1 引言

## 1.1课题背景与意义

在学习C语言和数据结构之后，初步掌握了用C语言等高级语言编写程序的基本技巧和理论知识，但对于如何处理较为复杂的问题还有待进一步掌握。本次课题便是一个很好的学习与锻炼机会，从最基础的数据结构开始逐步实现SAT问题的解决，对我们的综合能力是一次检验。在本次课设的完成过程中，也学到了很多东西，编程思维得到了提高。

## 1.2 国内外研究现状

DPLL算法是一个解决SAT问题的经典算法，国内外都有相当数量的论文供给参考，前人的研究为我本次的课设提供了丰富的思路。

## 1.3 课程设计的主要研究工作

### 1.3.1问题简介

**1 文字(literal)和子句(clause)**

对于任一布尔变元x，x与其非“¬x”称为**文字**。对于多个布尔变元，若干个文字的或运算l1∨l2∨…∨lk称为**子句**。

只含一个文字的子句称为**单子句**。不含任何文字的子句称为**空子句**。

**2 单子句规则**

若字句S中有一个单子句L，则L一定取真，从S中删除所有有L的子句，得到S1，若其为空集，则S可满足。

如果此时S1不是空集，那么对于S1中的每一个子句，若有¬L，则去掉这个文字，得到S2。S满足当且仅当S2满足。

1. **SAT问题概述**

SAT问题即为对于给定布尔变元集合以及其子句集合，若对于其子句的合取范式(CNF范式)，判断是否有一组对于布尔变元的赋值使得该CNF范式为真。当为真的时候，即我们称该问题是可以满足的，即SAT，此时我们输出给布尔变元赋值的结果。

**4 DPLL算法综述**

输入一个子句集S，使得其进入DPLL算法，若满足返回true，不满足返回为false。接下来是一个循环。从S中选择单子句L，利用单子句规则来化简L。如果此时化简结果是S为空集，证明可满足，返回true。如果此时S不为空集但是S中有子句是空子句，证明不可满足（无论取怎样的布尔值组都不可满足）。若是其他情况，继续循环，寻找S中的单子句，重复上面的步骤直到S中没有单子句，此时从S中选择一个v使其成为单子句，加入S递归调用DPLL，若返回值不为true，则v变为¬v，递归调用DPLL。

### 1.3.2主要内容

**1 创建数据结构**

包含文字，字句，CNF等结构和相关的函数功能，比如创建和删除文字。

**2 实现DPLL算法**

包括dpll的主函数和其调用的其他函数。

**3.实现数独模块**

包括导入棋盘，三个约束条件和输出。

**4 做最后的界面处理**

# 2系统需求分析与总体设计

## 2.1系统需求分析

本系统主要实现了解析cnf文件以及求解其中的SAT问题。以及解决二进制数独求解。

## 2.2系统总体设计

**程序中主要结构可以概述如下**：

1. 建立数据结构类型，声明所需的数据结构。
2. 在数据结构的基础上，编写相关的函数，如创建文字，删除字句等基本功能。
3. 编写上层函数，如寻找单子句，将文字进行排序等。
4. 编写输入输出函数，便于后续读写文件。
5. 编写DPLL函数。
6. 编写SAT主函数。
7. 编写数独处理函数
8. 编写main函数。

主要包括的文件有

structure.h（数据结构相关）

solveDPLL.h（DPLL算法）

sudoku.h（数独相关）

main.cpp（主函数）

# 3 系统详细设计

## 3.1 有关数据结构的定义

1. **CNF，字句和文字的存储。**

CNF文件的内容包括若干条字句，每个字句包含若干文字，所以我选用链表来存储CNF文件，其中每一节都是一个字句的头，字句头也是一条单链表，指向该字句包含的文字，结构如下：

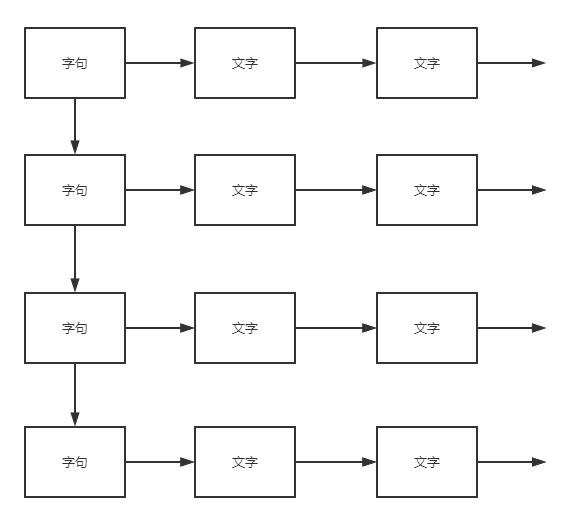


图3-1 CNF数据结构示意图

1. **其他数据结构。**

一开始我是只建立了字句Clause和文字Literal的，但编写dpll算法时发现该算法总是需要回溯过往的数据，如果每次都删除就无法实现，所以为解决这个问题，我在每个字句和文字处增加了状态量state（true or false），在最后输出结果的时候不考虑为false的子句或者文字。

之后我增加了一个数据结构Memory实时记录变为false的文字的位置，便于后续的修改

下面简要描述我建立的结构以及结构中的数据类型有Clause、literal、Memory、Sign。

对于四个结构简介如下：

表3-1 数据结构功能介绍

|  |  |
| --- | --- |
| 结构名 | 主要功能 |
| Clause | 子句链表，连接每个子句的头 |
| Literal | 文字链表，由Clause的sub指出，连接该子句所有文字 |
| Memory | 记忆链表，记录要删除文字的位置 |
| Sign | 符号链表，记录每个文字取的真值 |

**下表介绍了这些结构的主要数据类型：**

表3-2 结构中数据类型的介绍

|  |  |  |
| --- | --- | --- |
| **结构名** | **数据类型** | **数据意义** |
| **Clause** | clauselen | 子句的长度 |
| location | 子句的编号 |
| state | 存在性 |
| \*next | 指针类型，指向下一个子句 |
| \*sub | 指针类型，指向头文字 |
| **Literal** | location | 文字在该句中编号 |
| state | 存在性 |
| num | 这个文字代表的数字 |
| \*next | 指针，指向下一个文字 |
| **Memory** | deClauselocation | 记录state变为false字句的位置 |
| deLiteralLocation | 记录state变为false文字在句中的位子 |
| \*next | 指针，指向下一个变为false的节点 |
| **Sign** | num | 文字编号 |
| sgn | 文字真值（真为1假为-1） |
| \*next | 指针，指向下一个符号标记 |

## 3.2 主要算法设计

**算法概述**

1. 遍历S查找其中的单子句，若存在，用单子句规则化简S，若S为空，DPLL函数输出true，若S不为空且S中有空子句，DPLL函数输出false。
2. 否则循环上一步，直到没有单子句为止
3. 若是执行完上面步骤还没有输出结果，则基于某种策略从中选取一个文字v，使其成为一个单子句，并入S，递归执行DPLL(S)。
4. 若是此操作返回true，则DPLL函数返回true，不然将v取反，递归执行DPLL

**DPLL实现策略**

第一次编写的DPLL算法要求我在选择了一个变元v执行DPLL，且结果为False的时候恢复整个链表，并把变元v取反。但由于之前已经全部删除包含v的字句，这一步几乎无法实现，若选择拷贝记录会导致内存占用过大，所以我选用了上文中提到的存在性标记state：

1 我对每一个子句和每一个文字都添加了一个state变量，存在为true，删除了就为false。

2 用Memory记录删除的位置，如果需要复原就去遍历Memory。

这样就解决了DPLL算法中最麻烦的问题。

**数独实现策略**

首先读入棋盘的方式为将棋盘初态以文字的形式存入文件，如第一行1，3位为1则记录为“11 13”，读入后将这些文字转换成单字句记录到sudoku.cnf文件中。之后调用limit1，limit2，limit3三个函数分别将三种约束以字句的形式加入该文件。

最后调用dpll算法处理刚刚生成的sudoku.cnf。最后用signList记录的前36个变元的值作为解输出为棋盘格式。

# 4 系统实现与测试

## 4.1 系统实现

1. **IDE**： visual studio 2019
2. **系统功能实现总流程**
3. 建立数据结构，编写基本操作函数
4. 从文件中读取数据存入对应的结构
5. 编写DPLL程序
6. 做最后的界面处理
7. **DPLL算法实现的结构**

上文已有整体思路和疑难问题，这里仅展示函数调用关系：

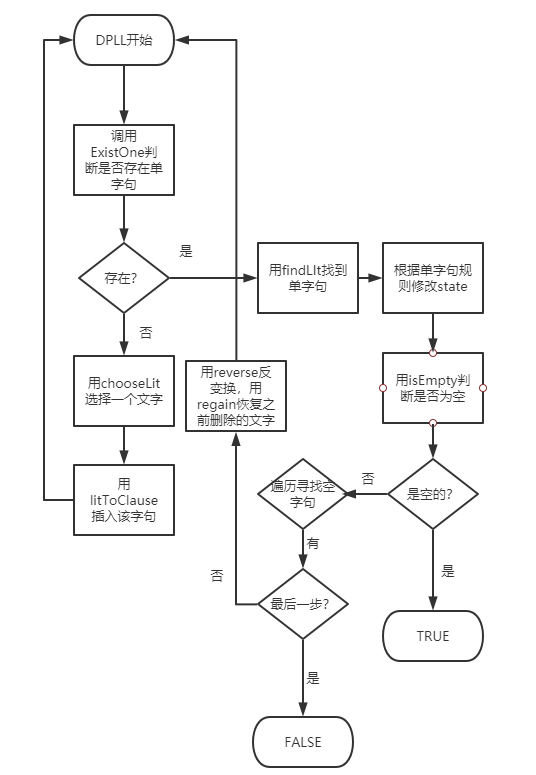


图4-1 DPLL结构流程示意图

1. **数独实现的结构**

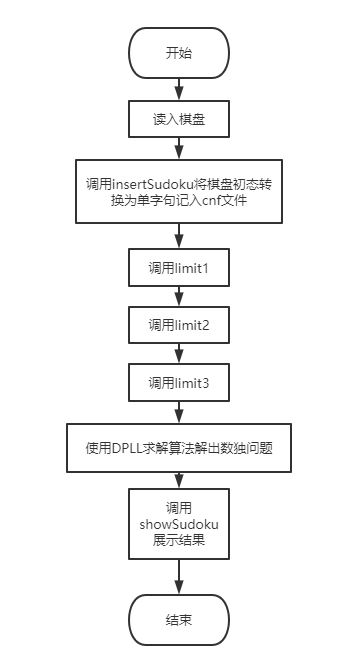


图4-2 数独问题求解流程示意图

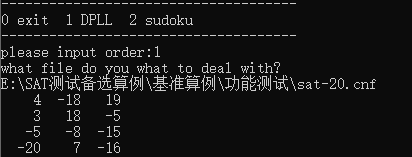
## 4.2 系统测试

**CNF求解模块测试：**

模块功能：读取CNF文件内容，并求解，之后生成.res文件输出求解结果，包括是否可满足，如果是，各项的取值如何，求解时间是多少。

**功能测试：**测试是否可以满足基本功能

* + 1. **sat-20.cnf测试如下**：



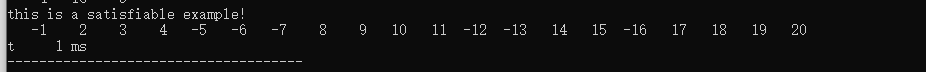


图4-3 sat-20.cnf输出结果

证明这是一个满足算例，输出的res文件截图如下：

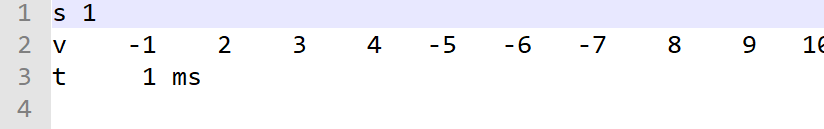


图4-4 sat-20.cnf相应输出的res文件

第一行输出S 1指该文件为可满足（satisfiable）文件，第二行为每个文字对应的取值，第三行是运行时间1ms

* + 1. **unsat-5cnf-30.cnf测试结果如下：**

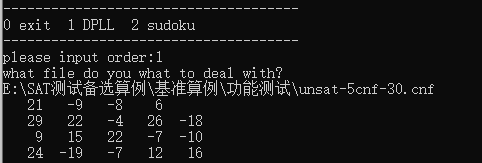




图4-5 unsat-5cnf-30.cnf输出结果

证明这是一个不满足算例，输出的res文件如下：

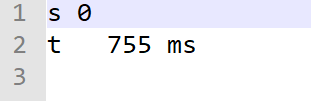
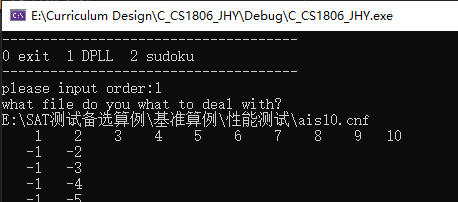


图4-6 unsat-5cnf-30.cnf相应输出的res文件

S为0表示不满足，DPLL运行时间为755ms。

**性能测试：**测试在满足功能的前提下，运行时间如何

* + 1. **ais10.cnf测试结果如下**



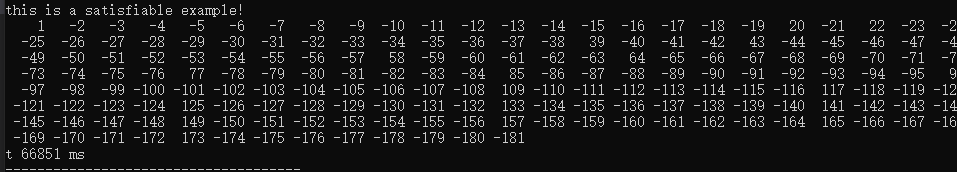


图4-7 ais10.cnf输出结果

证明这是一个可满足算例，输出的res文件如下：

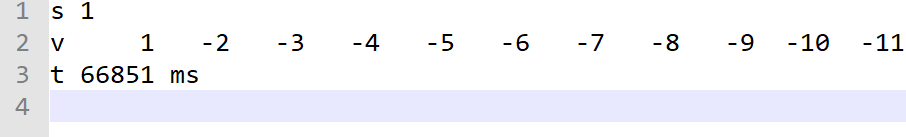
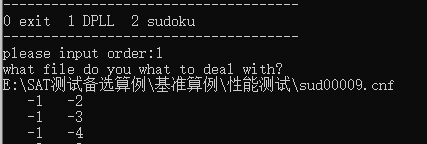


图4-8 ais10.cnf相应输出的res文件

S为1代表可满足算例，v后面输出真值，DPLL运行时间为66851ms

* + 1. **sud00009.cnf测试结果如下**



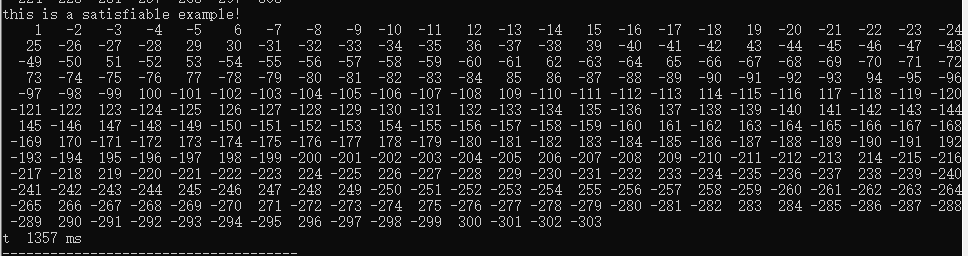


图4-9 sud00009.cnf输出结果

这是一个可满足算例，输出res文件如下：

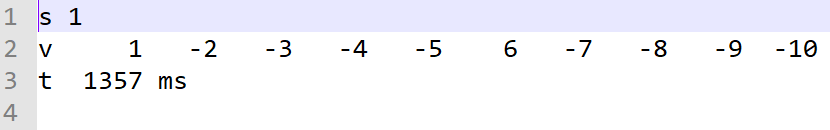


图4-10 sud00009.cnf相应输出的res文件

S为1代表可满足算例，v后面输出真值，DPLL运行时间为1357ms。

* + 1. **优化比例**

优化方案是从选取第一个文字改为选取众数，但是很多文件都不能用优化后的方案运算，跟踪调试也没找到问题所在。

其中可以优化且有优化效果的文件sud00009.cnf输出如下



图4-11 sud00009.cnf优化后输出结果

可以看出，确实在选取众数后计算时间有所缩短在，这种优化方案是可行的。

表4-1 SAT部分样例优化率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件名 | 所属类型 | 是否满足 | 优化前运行时间 | 优化后运行时间 | 优化比例 |
| sat-20.cnf | 基准算例-功能测试 | 是 | 1 ms | 0 ms | 100% |
| sud00009.cnf | 基准算例-性能测试 | 是 | 1357ms | 908ms | 33% |

* + 1. **其他测试**

可以优化的测试样例是优化后的运行时间

表4-2 SAT测试样例及其输入结果一览图

|  |  |  |  |
| --- | --- | --- | --- |
| 文件名 | 所属类型 | DPLL运行时间 | 是否满足 |
| sat-20.cnf | 基准算例-功能测试 | 0 ms | 是 |
| unsat-5cnf-30.cnf | 基准算例-性能测试 | 13 ms | 否 |
| ais10.cnf | 基准算例-性能测试 | 66582 ms | 是 |
| sud00009.cnf | 基准算例-性能测试 | 908ms | 是 |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 满足算例-S | 938 ms | 是 |
| problem1-20.cnf | 满足算例-S | 1 ms | 是 |
| problem2-50.cnf | 满足算例-S | 1 ms | 是 |
| problem3-100.cnf | 满足算例-S | 17 ms | 是 |
| problem6-50.cnf | 满足算例-S | 2 ms | 是 |
| problem8-50.cnf | 满足算例-S | 14 ms | 是 |
| problem9-100.cnf | 满足算例-S | 6 ms | 是 |
| problem11-100.cnf | 满足算例-S | 73 ms | 是 |
| tst\_v25\_c100.cnf | 满足算例-S | 0 ms | 是 |
| bart17.shuffled-231.cnf | 满足算例-M | 23 ms | 是 |
| problem5-200.cnf | 满足算例-M | 97 ms | 是 |
| problem12-200.cnf | 满足算例-M | 89 ms | 是 |
| sud00001.cnf | 满足算例-M | 384 ms | 是 |
| sud00009.cnf | 满足算例-M | 1492 ms | 是 |
| sud00012.cnf | 满足算例-M | 234 ms | 是 |
| sud00021.cnf | 满足算例-M | 8219 ms | 是 |
| sud00079.cnf | 满足算例-M | 1033 ms | 是 |
| sud00082.cnf | 满足算例-M | 40 ms | 是 |
| sud00861.cnf | 满足算例-M | 266 ms | 是 |
| tst\_v200\_c210.cnf | 满足算例-M | 3 ms | 是 |
| tst\_v10\_c100.cnf | 不满足算例 | 2 ms | 否 |
| u-5cnf\_3500\_3500\_30f1.shuffled-30 | 不满足算例 | 830 ms | 否 |
| u-problem7-50.cnf | 不满足算例 | 4 ms | 否 |

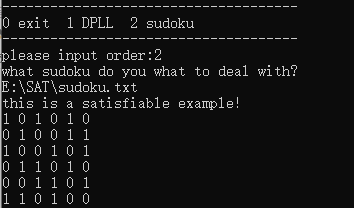
经过多次测试，发现无法运算L算例，可能是程序结构仍然有待优化。

**数独求解模块测试：**

模块功能：读取棋盘文件内容，生成对应的cnf文件，并求解，之后生成.res文件输出求解结果。

**功能测试：**测试是否可以满足基本功能

读入棋盘为：11 -14 -23 -24 26 -32 -33 36 -51 -52 54 62 -65 -66

求解如下，可以看出答案是正确的。

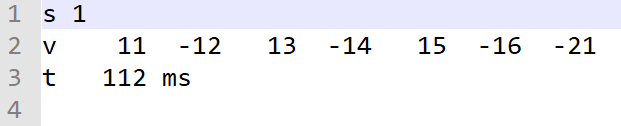
图4-12 数独输出棋盘

图4-12 数独输出文件

运行时间是112ms，和中型算例差不多。

根据参考资料[5]，得到了一些棋盘进行测试，具体如下：

表4-3 数独求解情况

|  |  |  |
| --- | --- | --- |
| 棋盘 | DPLL运行时间 | 是否正确 |
| 13 -21 -22 24 -31 54 -65 | 112 ms | 是 |
| 14 -23 26 -31 -35 42 43 61 -65 | 112 ms | 否 |
| 11 -14 -23 -24 26 -32 -33 36 -51 -52 54 62 -65 -66 | 112 ms | 是 |

猜测运算时间相同是因为约束条件完全一致，区别只有初值。

# 5 总结与展望

## 5.1 全文总结

**在这次课设中，我的主要工作如下：**

设计了合理的数据结构来存储CNF文件，解决了DPLL算法的求解，进而解决SAT问题。通过选取众数的方式对DPLL算法做了优化，有一定效果。最后通过构建约束条件将二进制数独转换为cnf文件并用DPLL求解，可解决一类数独问题。

## 5.2 工作展望

**在这次课程设计中依然有许多需要优化的地方：**

我编写的DPLL算法用了递归的处理方式，显然不如迭代的效率高，多次测试L文件都出现了栈溢出现象，递归是有极限的，但我尝试改成迭代都没有很好的实现，甚至出现了解析错误，最后只能在递归的基础继续处理。优化算法对部分算例产生了负优化，甚至无法求解。

**通过此次课设的经验，希望在今后的编程学习中，我能更好的做到：**

能够更加熟悉链表等数据结构，有了一定的运用编码解决大型问题的能力，进一步熟悉了一个项目的管理方式，掌握了简单的算法思想。

# 6 体会

# 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]TanbirAhmed.An Implementation of the DPLL Algorithm.Masterthesis,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]CarstenSinz.Visualizing SAT Instances and Runsof the DPLL Algorithm.JAutom Reasoning (2007) 39:219–243

[5]Binary Puzzle：<http://www.binarypuzzle.com/>