华中科技大学

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：计算机 ACM1901 班

学　　号：U201915035

姓　　名：邹雅

指导教师：李开

报告日期：2021.3.15

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：

⑴ **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%)

⑵ **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%)

⑶ **DPLL 过程：**基于 DPLL 算法框架，实现 SAT 算例的求解。(35%)

⑷ **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%)

⑸ **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%)

⑹ **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

1.1 课题背景与意义

SAT 问题即命题逻辑公式的可满足性问题，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。

对个人而言，作为一名计算机科学的学生，在前三个学期已经学过 C 语言和串并行数据结构两门理论基础课以及这两门课的实验课，较为系统的了解了 C 语言和数据结构的基本知识，并且具有了初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。基于求解 SAT 问题并且设计二进制数独游戏程序是一个很好的项目实践过程。本课题的主要意义是让我将这些知识融会贯通，初步运用于实战，为后续学习和工作打下良好的基础。

1.2 国内外研究现状

从 1960 年至今，SAT 问题一直受到人们的关注，世界各国的研究人员都为之付出了大量的努力，提出了许多求解算法。每年可满足性理论和应用方面的国际会议都会组织一次 SAT 竞赛，在竞赛上会详细展示一系列的高效求解器的性能，并希望以此找到一组最快的 SAT 求解器。2003 年的 SAT 竞赛中针对从成千上万的基准问题中挑选出的一些 SAT 问题实例，有 30 多种解决方案同台竞争。国内也经常会组织一些 SAT 竞赛及研讨会这些都促进了 SAT 算法的迅速发展。

而其中非常重要的一种求解算法便是 DPLL 算法，它在 1962 年由马丁·戴维斯、希拉里·普特南、乔治·洛吉曼和多纳·洛夫兰德共同提出，作为早期戴维斯-普特南算法的一种改进。戴维斯-普特南算法是戴维斯与普特南在 1960 年发展的一种算法。很多求解 SAT 问题的高效算法都是通过改善 DPLL 算法而实现的，比如 CDCL。但是实际上 SAT 问题的求解与问题本身十分相关联，不同的问题使用不同的方式进行求解会有不同的效果，由此可见 SAT 问题的复杂性以及研究的意义。

尽管命题逻辑的可满足性问题理论研究已趋于成熟，但在 SAT 求解器被越来越多地应用到各种实际问题领域的今天，探寻解决 SAT 问题的高效算法仍然

是个吸引人并且极具挑战性的研究方向。

1.3 课程设计的主要研究工作

本次课程设计是根据课程要求所给出的 DPLL 的基本思想的基础上，思考出问题中的数据存储结构，在基于 DPLL 算法的思想上实现一个 SAT 求解器，该求解器有一定的输入和输出，并且对求解器进行一定的优化。

在进行优化之后，进一步理解和思考如何把二进制数独游戏转化为可解决的 cnf 文件，其中也涉及到选取数据存储结构，并且将其与 DPLL 算法进行结合，从而求解出该数独的解，同时，设计一个可互动的流程程序，从而可以实现基本的游戏功能。

2 系统需求分析与总体设计

2.1 系统需求分析

用户的需求可以分为以下两个基本需求，并且具有互动性，操作性。

需求一：在使用 DPLL 基础算法完成一个能够求解 SAT 问题的程序。程序通过读入.cnf 文件来输入一个 SAT 问题，通过程序的 DPLL 算法进行求解，并且将求解的结果按照一定的格式存放于.res 文档中。

需求二：将二进制数独游戏转化为 SAT 问题，基于上述 SAT 问题求解程序，实现一个简单易玩的数独游戏，该游戏具有一定的互动性，可玩性。能输出正确的答案，并且可以判断出用户是否解出了正确的答案，进行反馈。

除了以上基本功能，需要设计一个基本的程序流程，让用户可以选择需要的功能，并且可以控制退出程序。

2.2 系统总体设计

2.2.1 总体的设计

系统总体的设计主要体现在界面的设计上，在界面中选择要执行的满足需求的模块，以下为总体设计的流程图。

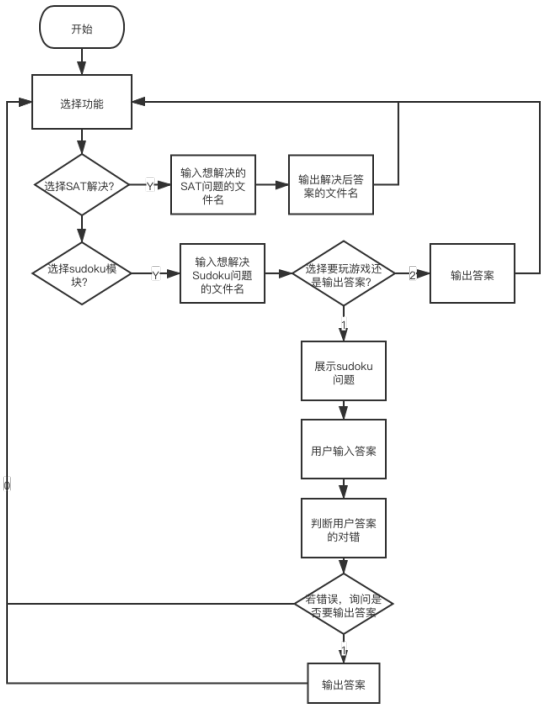


图 2-1 整体设计流程图

2.2.2 SAT 模块的设计

当用户选择求解 SAT 模块之后，首先会弹出一个提示，提示输入相应想要求解的 SAT 问题所保存的.cnf 文件的文件名，如果输入正确的文件名，则会调用程序求解的过程，并且会将解同步存放于同名同目录后缀名为.res 的文件中，并且显示出求解时间。

2.2.3 数独模块的设计

当用户选择数独游戏模块后，会提示用户输入当前目录下的数独游戏文件名，如果文件名正确，将会进行下一步，系统将自动将该文件里保存的数独盘局生成.cnf 文件，并且自动集成到 SAT 模块上求解。求解结束后，将会提示用户进行游戏或者直接查看答案或者退出。如果用户正确求解会有提示信息，如果用户求解错误将询问用户是否要查看答案，当用户选择要查看答案则输出答案并结束程序，若用户选择不查看则直接结束。

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 SAT 求解程序需要的数据结构

（1）系统需要处理的数据

系统需要处理的数据为 cnf 文件中读取的数据信息，以及存放的答案信息。

cnf 文件中存放的数据分别为：语句的总条数，变元的种类数，以及每个语句中单独的信息情况。

（2）系统的数据结构的描述

设计一个 CNF 结构，包含的信息有：当前子句所含的变元总数量，子句的数量，当前仍然有变元存在的子句数量，起始子句的地址，起始答案的地址。

设计一个子句 Clause 结构，包含的信息有：子句的变元长度，子句当前的状态，子句中第一个变元的地址，下一个子句的地址。

设计一个变元 Word 结构，包含的信息有：当前变元的值，当前变元的状态，下一个变元的地址。

其余有一些用于统计的数据结构在后续详细过程中再进行说明。

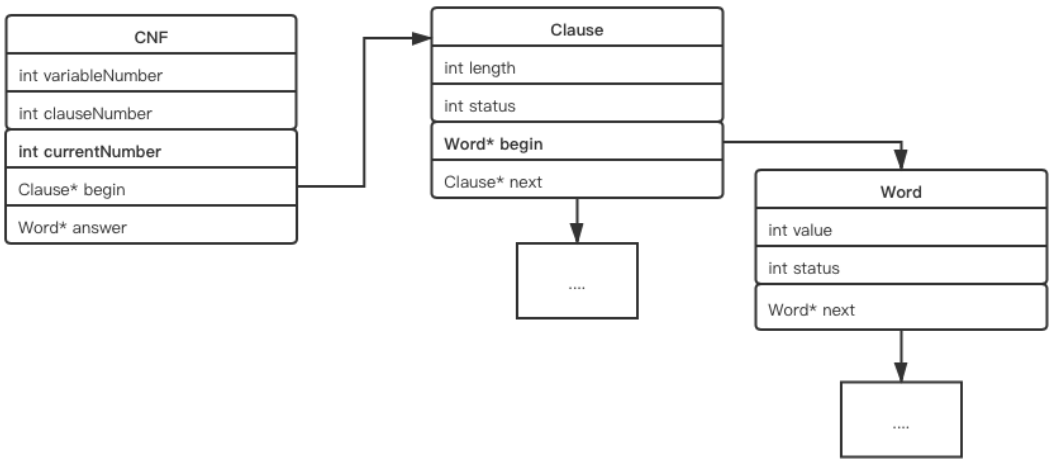


图 3-1 数据结构关系图

(i)合取语句的定义 CNF

数据结构的总体设计是一个与邻接表类似的数据结构。首先，使用一个大的结构存放读取及运算过程的所有信息。定义为 CNF。CNF 的定义如下所示。

```
typedef struct CNF{
    int variableNumber;//变元的数量
    int clauseNumber;//子句的数量
    int currentNumber;//当前还剩多少语句
    Clause* begin；//保存语句的起始地址
    Word * answer;//用来存放答案的起始地址
}total;
```

(ii) 子句的定义 Clause

正如表头中所表示的那样，子句与子句之间使用链表结构，每个字句中都保存了变元的起始地址。

```
typedef struct Clause{
    int length；
    int status；
    Word* begin；
    struct Clause *next；
}Clause;
```

(iii)变元的定义 Word

变元与变元之间使用单向链表的数据结构相互连接。

```
typedef struct Word{
    int value;
    int status;
    struct Word* next；
}vex;
```

3.1.2 数独游戏所需要的数据结构

（1）总体描述

数独游戏所需要的存储结构比 SAT 简单很多，需要处理的数据是创建的子句和变元。

（2）数据结构的描述

（i）变元存储的数据结构如下。

```
typedef struct Element
```

```
{
    int value;
    struct Element* next;
}Element;
```

（ii）子句存储的结构如下。

```
typedef struct Sentence
{
    Element* begin;
    struct Sentence* next;
}Sentence;
```

3.2 主要算法设计

3.2.1 SAT 求解的模块设计

(1)最核心的 DPLL 模块的设计

DPLL 模块的设计主要是参考任务书上给出的描述。首先判断当前是否满足子句数量为 0，如果子句数量为 0，说明 SAT 已经求出了正确的解，直接返回 TRUE；如果当前子句数量不为 0，则遍历当前所有的子句，判断是否存在空子句。如果存在空子句，说明之前的求得的解有错误，返回 FALSE 值。

若上述都不满足的情况下，DPLL 还需要进一步递归运行，求解。下一步判断是否存在单子句。如果存在单子句，则利用以下单子句规则：

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 S_1，如果它是空集，则 S 可满足。否则对 S_1 中的每个子句，如果它包含文字$\neg L$, 则从该子句中去掉这个文字，这样可得到子句集合 S_2。S 可满足当且仅当 S_2 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。

优先选取单子句的变元，进行下一层递归的 DPLL 过程，并且取返回值，如果返回值为 TRUE，则说明，当前选取了恰当的变元，并且求解出了正确的解，则当层递归也返回 TRUE。如果存在单子句且返回值为 FALSE，则说明当前选取的单子句的变元，带入 DPLL 后求得了错误的解。所以当层或者上一层或者之前的递归过程选取了错误的变元作为解，当层递归应当返回 FALSE。

如果不存在单子句，下一步则应用以下的分裂策略：

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 S_2；若 L 取假值（即¬L 成立）时，S 可化成 S_1.

调用选取变元的模块，按照选取变元的规则，选取变元进入下一层递归。如果下一层递归返回 TRUE，说明选取了正确的变元，应当将现在选取的变元作为解。返回 TRUE。如果下一层递归返回值为 FALSE，说明选择的变元不正确。在本程序中，如果选取了不正确的变元，则选择当前变元的非，进入下一层的递归，如果返回值为 TRUE，说明选取了正确的变元，并作为 SAT 问题的解，当层递归也返回 TRUE。在此如果返回值为 FALSE，说明根据规则，当层选取的变元或者之前选择的变元不是 SAT 问题的解，则应当返回 FALSE。至此，本层递归所有的情况均对应有相应的返回值，完成了相应的任务。DPLL 主程序的运行流程到此结束。运行流程图如下所示。

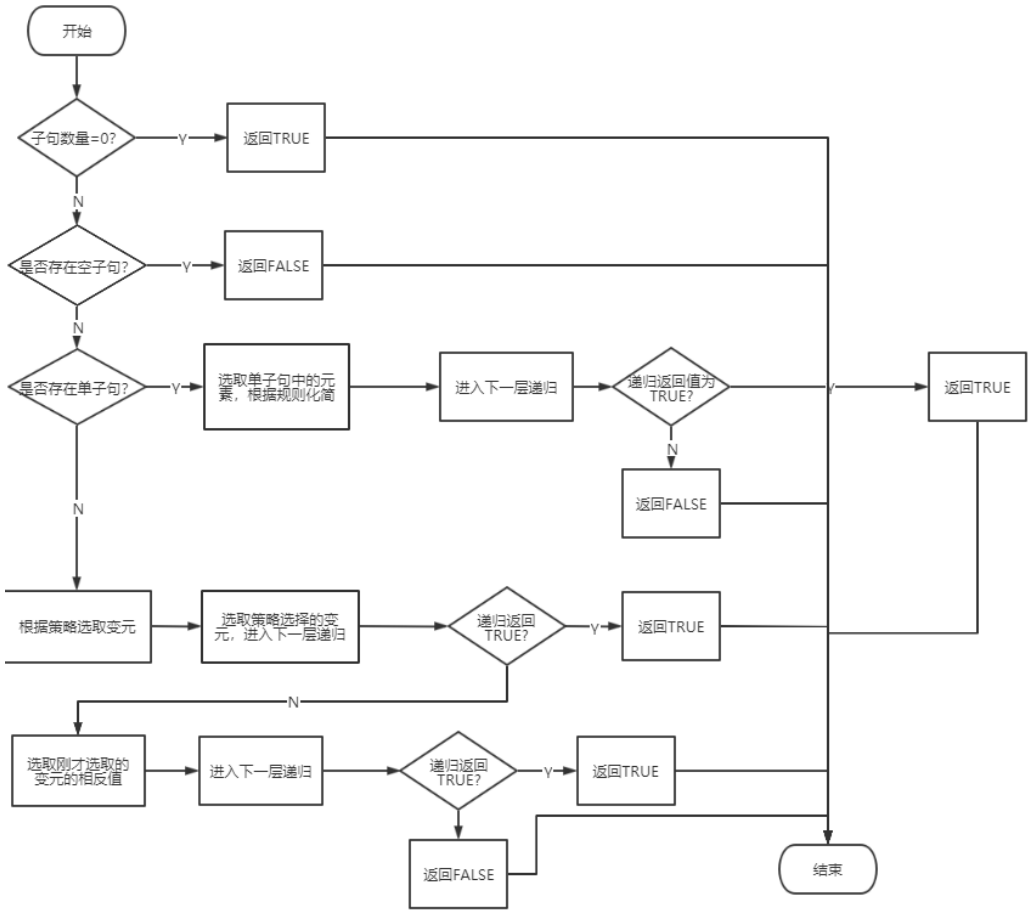


图 3-2 DPLL 算法运行图

（2）变元选取策略的算法

在变元选取策略这里，我进行了优化。

优化前采取的变元选取策略是，直接选择出现的第一个变元，这样做的好处

是，节省了选取变元算法本身的时间，所以对于一些小型算例来说，这样的算法的时间会比较快。但是这样选取变元策略的弊端就是，对于稍大的算例，使用这样的变元进行计算，往往效率非常低下，递归过程非常长，导致一些大型的算例无法在可行的时间内得出结果。

优化后采取的变元选取策略是，首先，找出长度最长的子句，记录子句长度，然后找出长度最短的子句，记录子句的长度。计算这两个长度的平均值，进行记录。再次遍历所有的子句，记录比平均值长度低的所有子句的每个变元出现的频率，并进行排序。每次选择变元的时候，都选择出现频率最高的变元。这样做的优点是，在一些大型算例，子句长度不统一的情况下，能够有效的使算例出现单子句。从而提升求解的效率。使得一些大型算例得以进行。缺点就是，对于一些中小型的算例，变元选取的算法本身消耗的时间较长，效率不如方法一直接选取变元的速度。从而在小型算例上，时间相对慢一些。

（3）子句处理与恢复

子句处理：当选择了一个处理的变元之后，在所有的子句中通过循环查找该变元，如果子句中存在该变元，则将整个子句的 status 改为 0，即该子句在该变元为真的情况下可满足，将子句的数量-1，如果子句中存在该变元的非，则将对应变元的 status 改为 0，则该变元为假，当前子句的长度-1。

子句恢复：当发现选择了一个错误的变元时调用，对子句处理进行反处理，具体流程如同子句处理，将所有的子句处理的过程逆向即可。

3.2.2 数独游戏的算法

数独游戏的算法主要是将数独游戏转化为 cnf 文件，再利用 SAT 求解模块求解即可。

与普通数独游戏相似，n 阶二进制数独游戏要求在 n×n 的网格中每个单元填入一个数字 1 或 0，必须满足约束：（1）在每一行、每一列中不允许有连续的 3 个 1 或 3 个 0 出现；（2）在每一行、每一列中 1 与 0 的个数相同；（3）不存在重复的行与重复的列。根据这些约束，以及在任务书中的说明，使用循环等操作，添加所有的子句即可完成。

添加完之后，求解只需要将生成的.cnf 文件集成到 DPLL 算法上进行求解，对求解的结果进行处理，去除添加的辅助变元，就是对应数独的解。

4 系统实现与测试

4.1 系统实现

这部分从用户需求出发，明确你做的系统要实现的目标，能处理一些什么样的事务、事务处理流程等。

4.1.1 软硬件环境

硬件环境：MacBook Pro (macOS BigSur)

软件环境：Xcode

4.1.2 数据类型

这部分已经在 3.1 中给出。

4.1.3 函数原型及关系

函数原型：

(1) int DPLLMain(char* filename,int isSudoku);

函数功能：DPLL 算法的主函数，负责 DPLL 模块的调用以及一些文件操作

算法思想：类似于 main 函数的调用函数。

(2) int simplify(CNF* current,int tag);

函数功能：将 tag 对应的变元对于所有的子句进行处理。

算法思想：循环，判断。

(3) int backward(CNF* current,int tag);

函数功能：simplify 的反函数，恢复 simplify 函数做的改变

算法思想：循环，判断。

(4) int DPLL(CNF* current,Word* curAns);

函数功能：进行 DPLL 功能的实现。

算法思想：DPLL 算法的递归函数。

(5) int chooseVariable(CNF* current);

函数功能：选择变元。

算法思想：排序，选择。

(6) CNF* createCNF(char* origin);

函数功能：创建当前问题对应的 CNF 结构，并且给各个指针，变量附初值。

算法思想：初始化。

(7) int transferNumber(char** content);

函数功能：将文件中保存的值变为对应数字，同时去除 0、“\0”、空格等

算法思想：文件的读取。

(8) char* readFile(char* filename);

函数功能：将文件保存的内容读取出来，保存为字符串，返回首指针。

算法思想：文件操作。

(9) int createCNFFileForSudoku(Sentence* start,int sentenceNumber,int elementNumber);

函数功能：读取.txt 文件，创建链表保存数独相关问题。

算法思想：文件的读取操作，动态分配内存。

(10) int createVariableClause(Sentence* start,int status,int row,int column);

函数功能：为数独游戏创建变元子句。

算法思想：动态分配内存。

(11) int limitedcontinuity(Sentence* start,int type);

函数功能：根据连续性约束为数独游戏添加子句。

(12) int limitedSameNumber(Sentence* start);

函数功能：根据同一行 0 和 1 个数相同约束添加子句。

(13) int addLimitInRow(Sentence* start,int a,int b,int c,int d,int row);

函数功能：为同一行 0 和 1 个数相同约束而创建在行的约束。

（14）int addLimitInColumn(Sentence* start,int a,int b,int c,int d,int column);

函数功能：为同一行 0 和 1 个数相同约束而创建在列的约束。

（15）int addElement(Element* start,int value);

函数功能：添加一个变元元素。

（16）int addNotSameRowOrColumn(Sentence* start);

函数功能：根据不同的行和列不能相同约束添加子句。

（17）int displaySudoku(Word* answerStart,char *filename,int choice);

函数功能：用于显示数独游戏，让用户输入，增加互动性，可玩性。

算法思想：简单的输入输出。

函数调用关系如下所示（箭头上的数字表示系统提示的选择）：

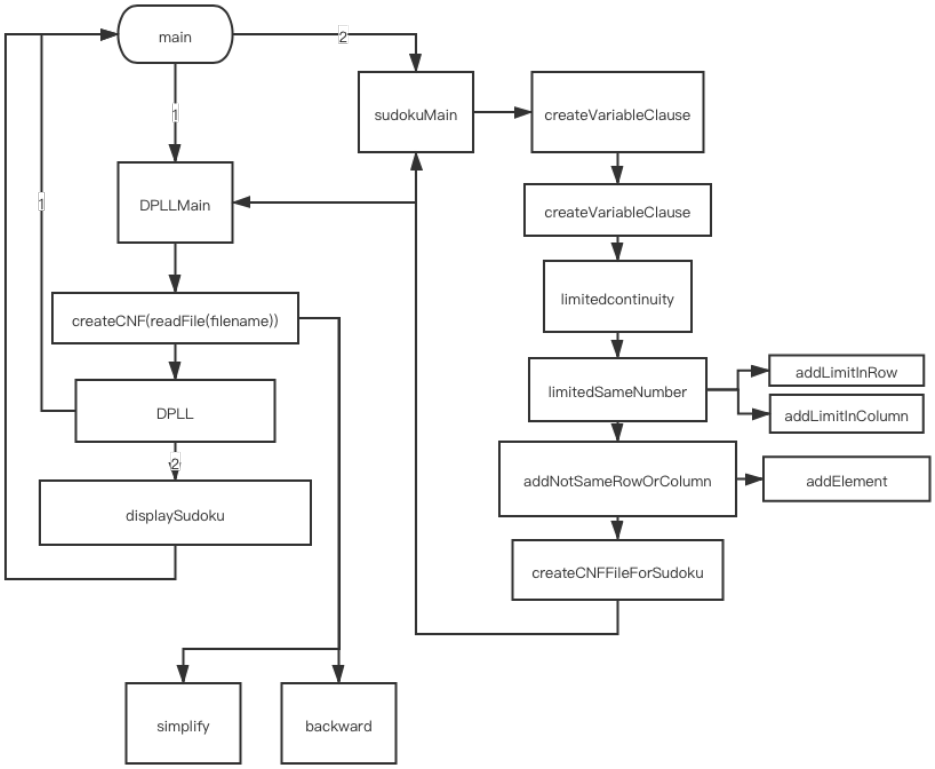


图 4-1 函数调用关系图

4.2 系统测试

系统测试主要分为两种测试，性能测试和功能测试。功能测试的内容是测试整个程序的功能，性能测试针对 SAT 求解，测试 SAT 求解过程的性能。

4.2.1 功能测试

设计思路：

使用基准算例测试 SAT 功能，并截图，再读取存放数独的文件，进行数独游戏的测试，并截图。

测试开始：

开始运行，会显示界面。

图 4-2 主界面测试图

在界面中，选择 1.进入 SAT 求解过程。提示输入文件名。

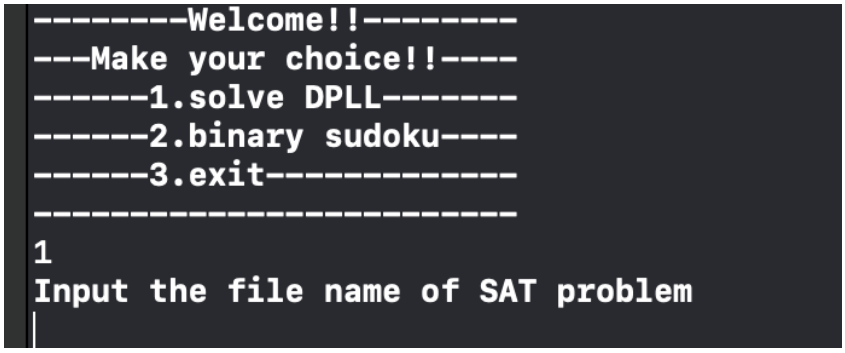


图 4-3 功能 1 操作测试图

输入对应的基准算例（可满足）位置，回车。

出现存储文件的信息，说明生成了.res 文件。

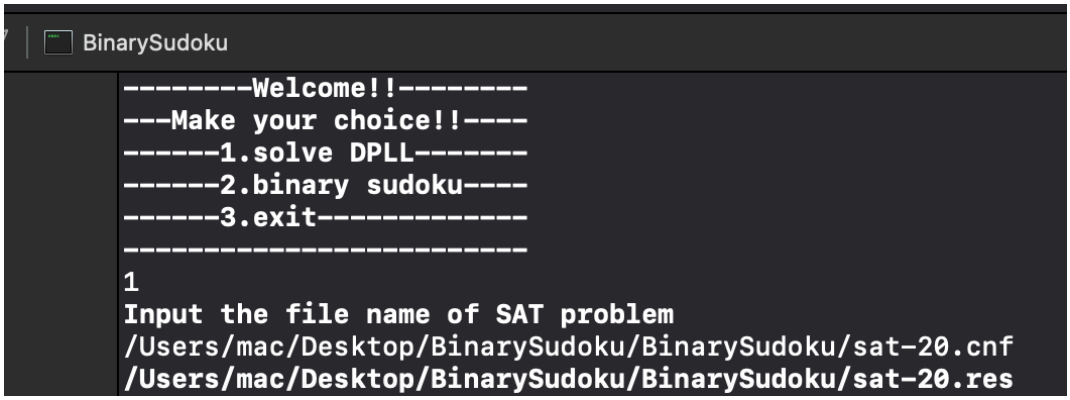


图 4-4 可满足算例生成结果

这时，到目录下寻找该文件。发现出现文件。

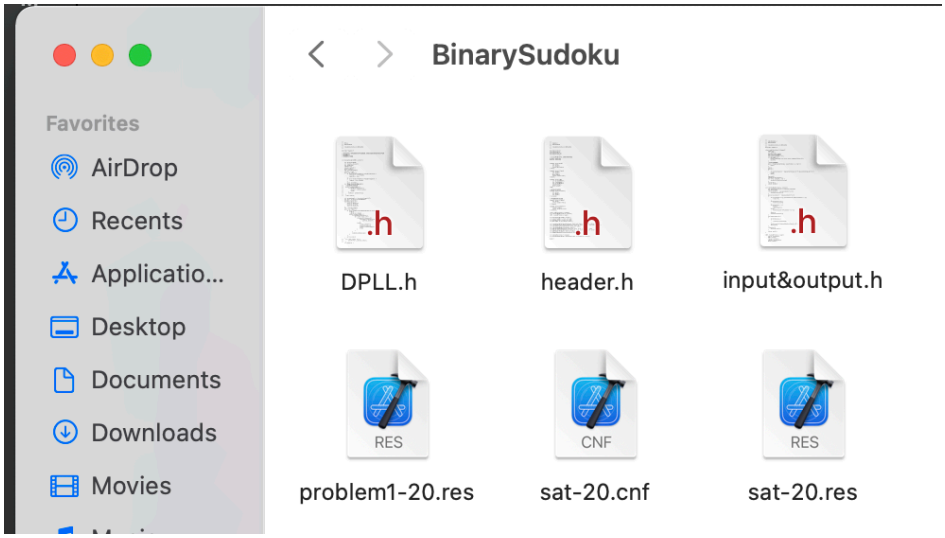


图 4-5 根目录查看

打开该文件，显示对应信息。s 为 1 表示可满足，v 后面存放了解，t 为时间，

单位是毫秒。

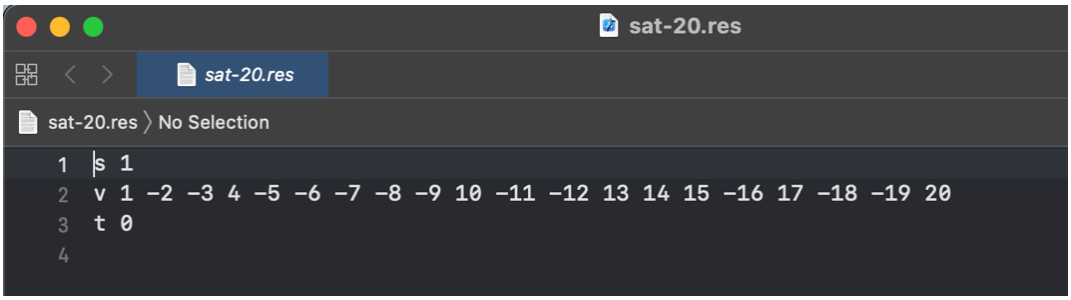


图 4-6 查看生成的答案

同理，以同样的操作运行基准算例中的不满足 SAT，也出现对应信息。

然后，进行数独游戏的功能测试。数独游戏存放的格式如下：3 代表不确定，0 和 1 都是数独的初值。

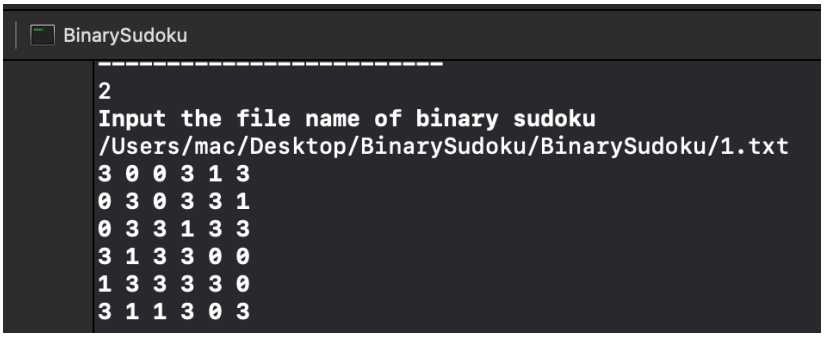


图 4-7 数独流程

输入 1.txt，出现对应的界面，并且在根目录下出现.cnf 文件。

BinarySudoku.cnf

图 4-8 数独生成. cnf 图

打开.cnf 文件。可以看到 SAT 的所有信息。信息正确。

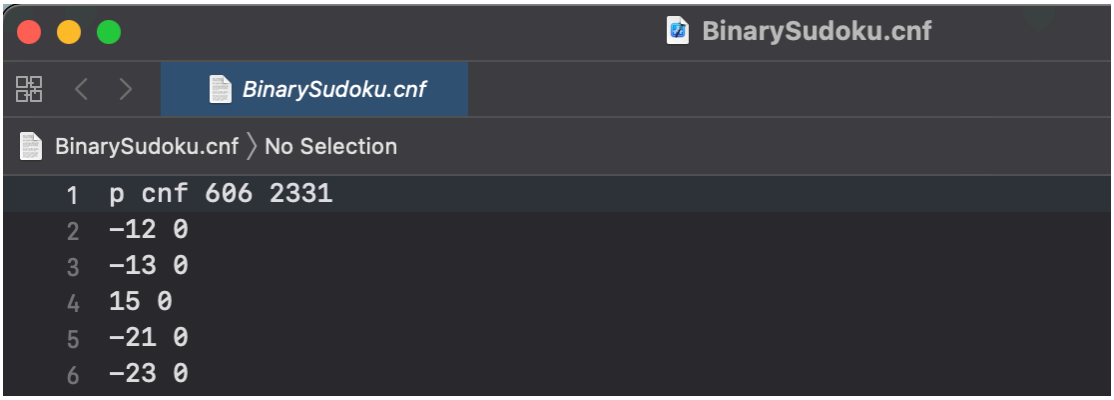


图 4-9 查看数独生成. cnf 程序图

图 4-10 提示图

首先，选择 2 显示答案。则会直接得到数独的答案。

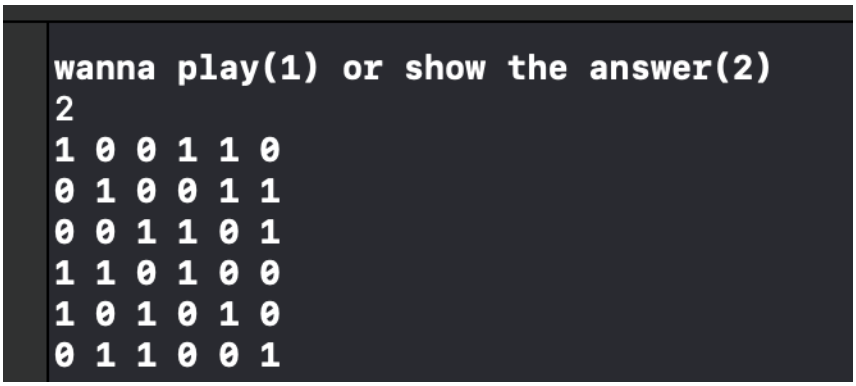


图 4-11 直接查看答案测试图

重新加载，开始数独游戏。选择解题，则会出现对应的画面。画面提示为输入要输入答案的位置。

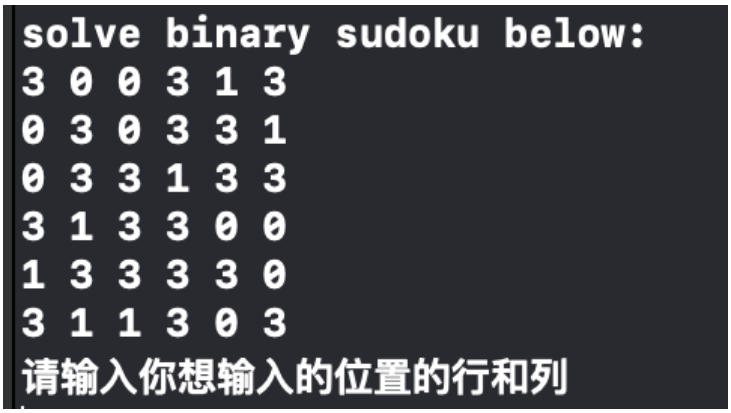
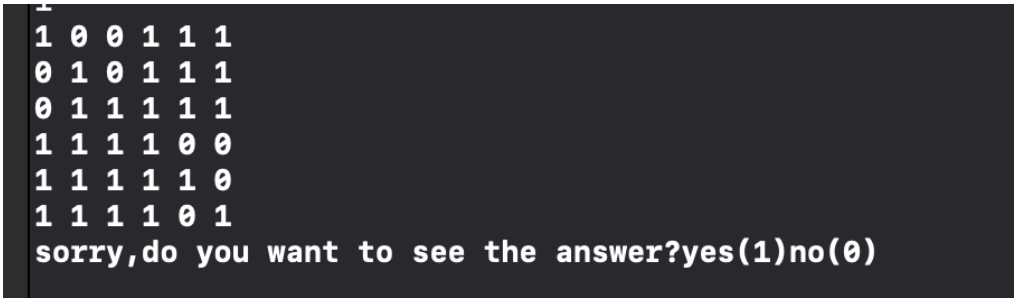


图 4-12 玩游戏过程测试图（1）

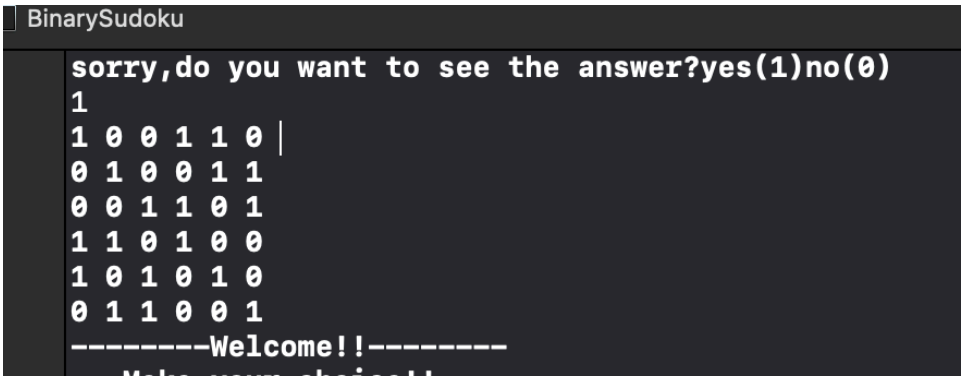
填入元素，并回车，可以继续输入下一个位置的答案。

全部填 1，输入错误，此时会提示不是正确答案，需要选择功能重新解答或者直接查看答案。



4-13 答案错误图

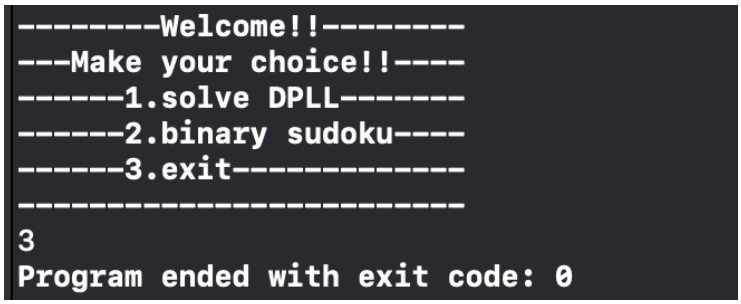
选择查看答案。



4-14 查看游戏答案

一轮游戏结束，重新回到程序选择模块。

在主菜单选择功能 3，程序退出。



4-15 退出图

4.2.2 性能测试

要求：不少于 18 个 SAT 算例，其中可满足的算例不少于 15 个，不满足的算例不少于 3 个，大中小算例各占三分之一。小型算例变元数为 100 个左右；中型算例变元数介于 200-500 个； 大型算例变元数 600 个以上。

由于大型算例不容易得出结果，大部分的大型算例由数独建立的.cnf 代替。该变元数量为固定值 606 个。

预计测试内容：

表 4-23 预计测试内容

算例名称	算例大小	可否满足
7cnf20_90000_90000_7.shuffled-20.cnf	小	满足
problem1-20.cnf		
problem2-50.cnf		
problem3-100.cnf		
tst_v25_c100.cnf		
unsat-5cnf-30.cnf		不满足
bart17.shuffled-231.cnf	中	满足
problem12-200.cnf		
sud00012.cnf		
sud00082.cnf		
tst_v200_c210.cnf		
u-5cnf_3500_3500_30f1.shuffled-30.cnf		不满足
eh-dp04s04.shuffled-1075.cnf	大	满足
BinarySudoku1.cnf		
BinarySudoku2.cnf		
BinarySudoku3.cnf		
BinarySudoku4.cnf		
u-dp04u03.shuffled-825.cnf		不满足

测试结果：

表 4-24 测试实际结果

算例名称	算例大小	可否满足	优化前时间 ms	优化后时间 ms	优化率
7cnf20_90000_90000_7.shuffled-20.cnf	小	满足	2653	1776	0.33
problem1-20.cnf			0	0	/
problem2-50.cnf			0	0	/
problem3-100.cnf			311	297	0.04
tst_v25_c100.cnf			0	0	/
unsat-5cnf-30.cnf		不满足	107	107	0
bart17.shuffled-231.cnf	中	满足	310	265	0.17
problem12-200.cnf			320	110	0.65
sud00012.cnf			47	6203	-130.98
sud00082.cnf			15	8474	-550.66
tst_v200_c210.cnf			0	0	/
u-5cnf_3500_3500_30f1.shuffled-30.cnf		不满足	94	937	-8.96
eh-dp04s04.shuffled-1075.cnf	大	满足	16953000	12504	1.00
BinarySudoku1.cnf			47	31	0.34
BinarySudoku2.cnf			46	16	0.65
BinarySudoku3.cnf			47	16	0.66

BinarySudoku4.cnf			47	15	0.68
u-dp04u03.shuffled-825.cnf		不满足	无穷大	1750	1.00

4.2.3 运行结果分析

从功能测试上可以看出，功能被完整的实现了。从性能测试上来看，性能测试上大多数的算例得到了优化，算例规模越大，优化越明显。但是，在中小算例上出现了负优化的情况，也是与选取变元算法本身的时间在中小算例中会占比较高等因素有关。

5 总结与展望

5.1 全文总结

在本次课设中，我主要完成了以下几个工作

1.查阅了相关文献，对 SAT 问题有了更深刻的了解。

2.设计了能满足需求且相对高效的数据结构。

3.实现了完整的 SAT 求解算法。

4.对程序进行了模块化，各个函数各司其职。

5.对在程序中出现的 bug 进行了修改，使得程序可运行并得到正确的结果。

6.设计了可玩的数独游戏流程。

7.对程序完成了完整的测试，得到了相关的测试信息。

8.对测试的结果进行了分析，并且思考了可以进行进一步优化的方向。

5.2 工作展望

首先，我认为在选取变元的策略上，一定有更加高效的选择。此外，针对不同规模的算例，也可以选择不同的选取变元的策略。

在数独游戏方面，本程序直接读取现有的数独存放文本文档。在查阅了相关资料之后，发现了可以用挖洞法来实现生成数独游戏。而且数独游戏的实现比较单一，在本程序中只能实现 6*6 的数独游戏，如果能实现不同规模的数独游戏可以更好。

在大算例的处理上，虽然已经达到任务要求，但是绝大多数给出的大型算例还是无法解决。今后在算法的优化上可以做得更好一些。

此外，应当给程序加上更利于用户交互的 GUI，比如运用 QT 写一个窗口。这一点也是可以继续优化的。

6 体会

本次课设是我用 C 语言写过的最大的一个项目了，这让我的 C 语言编程能力得到了一定的提升，让我能更好地用 C 语言来思考任务。

整个项目耗时大约两个星期。刚开始读课设任务书时，觉得十分困难，直到后来心慢慢的静了下来，反复认真阅读，一点一点地设计每一个模块，用了相当长的时间才把最初的优化之前的 SAT 求解写了出来。在构思过程中，由于模块之间的衔接以及很多算法上的问题，我会经常性陷入很长的思考，在电脑前反复揣摩反复修改程序。当我第一次把整体框架写完并且运行的时候，出现了无法运行的 bug，我一开始十分气馁，但在上网查阅之后，发现是头文件的问题，修改了之后便运行成功了。这不仅锻炼了我的编程能力，也让我更好地能面对编程中出现问题的局面。

后来在优化 DPLL 算法时，又查阅了大量的资料并且反复试验，最终确定了现在的选择变元的方式。转化成数独的过程也让我思考了很久，如何有效地添加子句这一问题让我思考了很久。

这次课设也让我接触到了英文的文献，读的时候还是有一些困难，但总体来说技术文章不算很难懂，读完之后发现英文的文献读起来更能与计算机的内容接轨，在代码上更易懂一些。这些阅读也直接地帮助了我实现数独功能。

总之，在本次课程设计任务过程中，我学到了很多也体会了很多，对于我来说是一次相当有意义的尝试。

参考文献

[1]TanbirAhmed.An Implementation of the DPLL Algorithm.Masterthesis,Concordia University,Canada,2009

[2] 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

[3]CarstenSinz.Visualizing SAT Instances and Runsof the DPLL Algorithm.JAutom Reasoning (2007) 39:219–243

[4] Binary Puzzle：http://www.binarypuzzle.com/

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