

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 综合程序设计课程设计**

**专业班级： 计算机科学与技术1707**

**学 号： U201714786**

**姓 名： 王占成**

**指导教师： 祝建华**

**报告日期： 2019年4月8日**

**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。
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。
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。

**目 录**

[任 务 书 1](#_Toc4606173)

[1 引言 1](#_Toc4606174)

[1.1 课题背景与意义 1](#_Toc4606175)

[1.1.1 课题背景 1](#_Toc4606176)

[1.1.2 课题意义 1](#_Toc4606177)

[1.2 国内外现状研究 1](#_Toc4606178)

[1.3 课程设计的主要研究工作 1](#_Toc4606179)

[2 系统需求分析与总体设计 2](#_Toc4606180)

[2.1 系统需求分析 2](#_Toc4606181)

[2.2 系统总体设计 2](#_Toc4606182)

[2.2.1 SAT问题求解 3](#_Toc4606183)

[2.2.2 标准数独文件读取与求解 3](#_Toc4606184)

[2.2.3 数独游戏 4](#_Toc4606185)

[2.2.4 系统模块结构图 5](#_Toc4606186)

[3 系统详细设计 5](#_Toc4606187)

[3.1 有关数据结构的定义 5](#_Toc4606188)

[3.2 主要算法设计 7](#_Toc4606189)

[3.2.1 DPLL算法设计 8](#_Toc4606190)

[3.2.2 挖洞法算法设计 9](#_Toc4606191)

[4 系统实现与测试 10](#_Toc4606192)

[4.1 系统实现 10](#_Toc4606193)

[4.1.1 ADT操作设计 10](#_Toc4606194)

[4.1.2 ADT操作实现 11](#_Toc4606195)

[4.2 系统测试 15](#_Toc4606196)

[5 总结与展望 25](#_Toc4606197)

[5.1 全文总结 25](#_Toc4606198)

[5.2 工作展望 26](#_Toc4606199)

[6 体会 27](#_Toc4606200)

[参考文献 27](#_Toc4606201)

[附录 28](#_Toc4606202)

# **1 引言**

## 1.1 课题背景与意义

## **1.1.1 课题背景**

命题逻辑公式的可满足性问题（SAT）是典型的NP完全问题，在计算机科学和人工智能等领域具有重要的理论意义与应用价值。同时，SAT问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法意义重大。

## 1.1.2 课题意义

通过本次课题的研究，来更深地认识和理解SAT问题的难度以及研究意义。同时实现较为简单的SAT求解器，来学习以及应用解决SAT问题的DPLL算法思想。拓展思维，提升自身能力。

## 1.2 国内外现状研究

到目前为止，仍不存在一种求解算法在最坏情况下的时间复杂度是多项式级别，其求解速度仍是制约SAT算法发展的一大难题。因此，世界各国的学者都在努力研究新的求解算法，以寻求出一种高效的求解算法。

## 1.3 课程设计的主要研究工作

根据完备算法对DPLL算法的多方面进行改进，基于DPLL算法框架，实现SAT算例的求解，得到具有较高效率的SAT问题求解器。与此同时，实现将数独问题集成到SAT问题求解器的功能，来将SAT问题的求解应用到实际生活中。

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

1. **输入输出功能：**读取用户的选择，进而实现用户选择的操作。读取用户输入的文件名，来从文件中读取内容或者创建相应的文件来存储相关内容。此功能具有简单的交互性。
2. **公式解析与验证：**读取cnf算例文件，解析文件，将文件的内容基于一定的物理结构存储起来，为DPLL过程做好准备。DPLL求解以后，将求解的结果和读取到的文件内容进行相应的结合，并且打印到屏幕上面，使用户能够简单地判断解析功能的正确性。
3. **DPLL过程：**基于DPLL算法框架，设计算法来实现对SAT问题的求解。
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，提高SAT问题求解器的求解效率。
6. **SAT应用：**将数独游戏问题转化为SAT问题，要实现将数独与SAT问题双向转化，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。能够判断用户输入答案的对错以及输出正确的答案。

## 2.2 系统总体设计

根据实验的要求以及C语言的相关规范标准，给出整个模块的总体架构：

模块包括4个头文件（.h）和5个实现文件（.c），如图2-1所示：

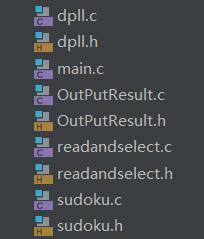


图2-1 代码文件图

模块的宏定义#define、类型别名typedef、结构定义和声明、函数原型被包括在头文件中。模块的ADT操作实现以及文件操作实现（函数）被包括在实现文件中

根据设计要求，可将整个系统划分为3个功能模块：

**1.** SAT问题求解

**2.** 标准数独文件读取与求解

**3.** 数独游戏

### 2.2.1 SAT问题求解

读取用户输入的cnf文件，将文件内容读取相应的数据结构中。然后对读取的SAT问题求解，将求解后的结果以及执行时间反馈给用户。根据用户的选择，可以对SAT问题的答案以文件形式保存，也可以将求解答案与SAT问题结合打印出来，人工判断所求的答案是否正确。此模块流程如图2-1所示：

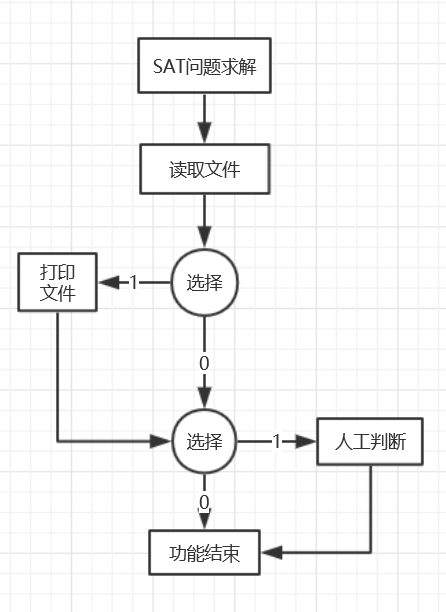


图2-2 SAT问题求解流程图

### 2.2.2 标准数独文件读取与求解

读取用户输入的cnf数独文件，将数独存储在相应的数据结构中。将数独游戏根据相应的算法转换为cnf问题，对cnf问题求解，然后再将cnf问题的解转化为数独游戏的解，最后把数独游戏的答案反馈给用户。整个过程实现了将数独问题集成到SAT问题求解器上求解和SAT问题与数独问题双向转换的功能。此模块流程如图2-2 所示：

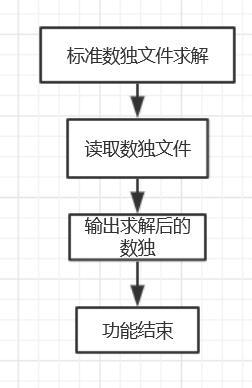


图2-3 标准数独文件读取与求解流程图

### 2.2.3 数独游戏

首先得到一个填充若干个数的初盘，对初盘求解得到终盘，对终盘使用挖洞法，生成具有唯一解的数独初盘。用户可以将自己求得的答案输入到程序中，让程序判断用户的答案是否正确，也可以让程序输出答案，自行与自己求得的答案进行对照。此模块流程如图2-3 所示：

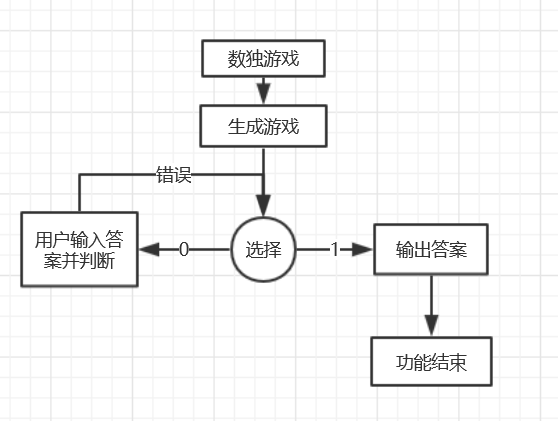


图2-4 数独游戏流程图

### 2.2.4 系统模块结构图

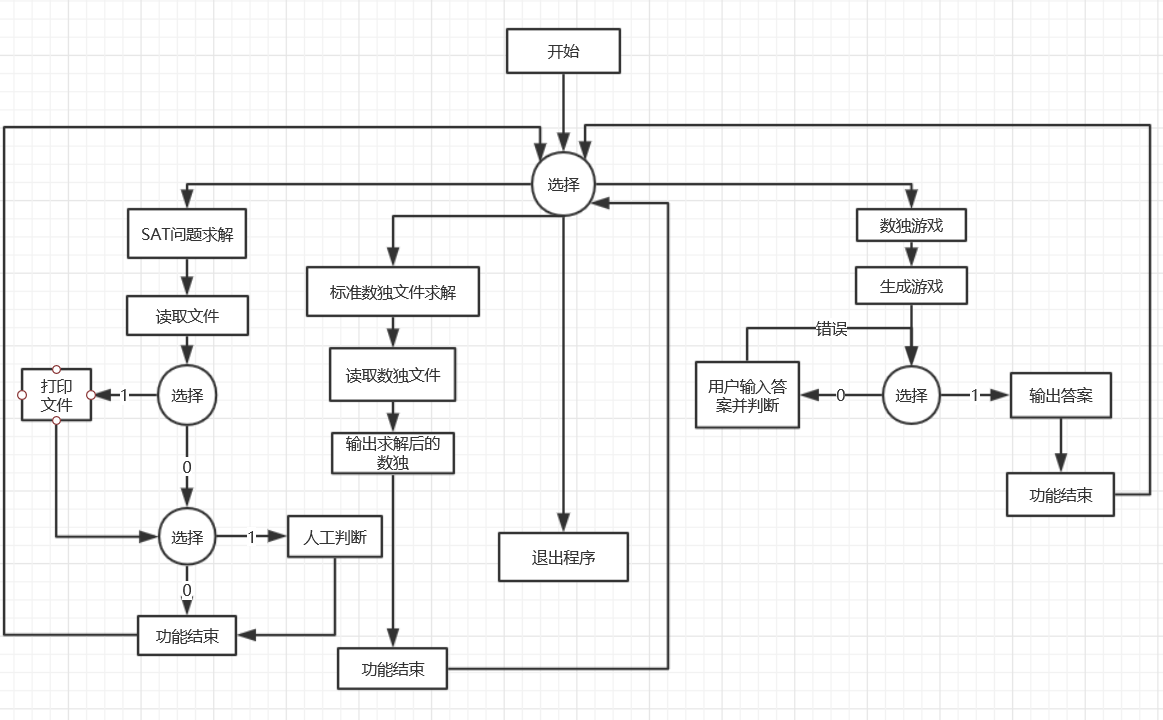


图2-5 系统模块结构图

# 3 系统详细设计

## 3.1 有关数据结构的定义

根据cnf文件的形式，可知cnf文件主要由子句和文字构成。每个子句包含多个文字，文字的类型是整型。因此，我们可以用一个使用双向链表的邻接表来实现物理结构的存储。

在本次实验中，根据实验的要求以及邻接表ADT的定义，给出基于C语言实现的邻接表ADT的结构定义以及别名：

子句中文字的存储结构：

typedef struct Clause{

struct Clause \* next\_word;

int word;

}Clause;

其中，next\_word指向子句中该文字的下一个文字，word表示该文字的值。

子句的存储结构：

typedef struct ClauseLink{

int positive\_number;//一行子句中正文字数目

int number;//一行子句中文字的数目

int isnew;//子句是不是新创建的

Clause \* present\_clause;

struct ClauseLink \* next\_clause;

struct ClauseLink \* last\_clause;

}ClauseLink;

其中，positive\_number表示该子句中正文字的数目，number表示该子句中文字的数目，isnew表示该文字是否是文件中本来存储的，present\_clause指向当前子句的第一个文字，next\_clause指向下一个子句，last\_clause指向上一个子句。

邻接表的结构如图3-1所示：

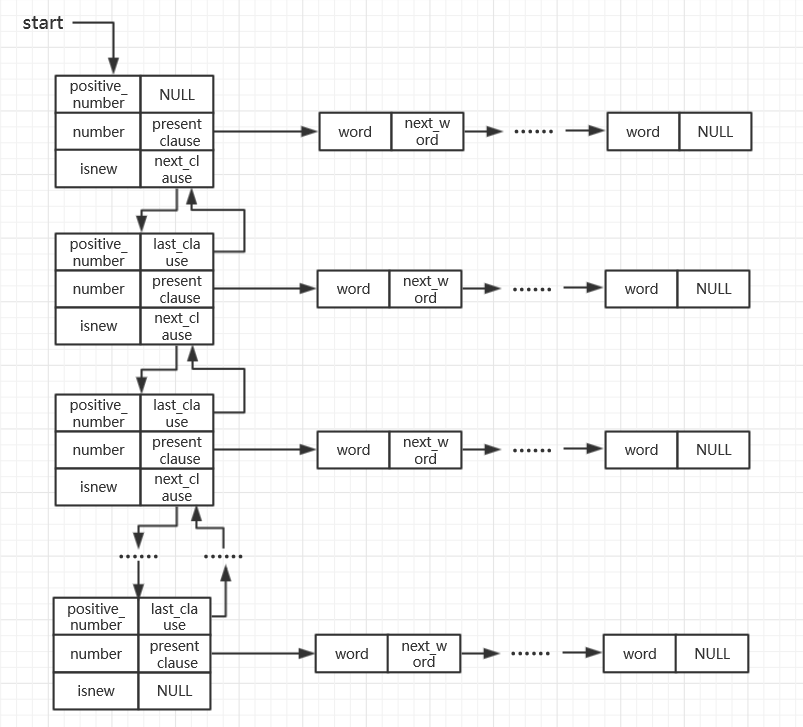


图3-1 邻接表的结构图

另外，根据实验的要求以及DPLL算法的实现，本次实验还定义了如下的栈结构：

typedef struct Stack{

int word;

int degree;

int wdorsen;//0:文字 :子句

struct Stack \*highstack;

ClauseLink \*delorchang;

}Stack;

其中，word表示的是操作的文字，degree表示的是被操作文字所在的深度，wdorsen表示的是当前栈存储的地址是被删除文字所在的子句地址还是被删除子句的地址，higstack指向上一个被入站的结构，delorchang指向被操作的子句地址。

栈的结构如图3-2所示：

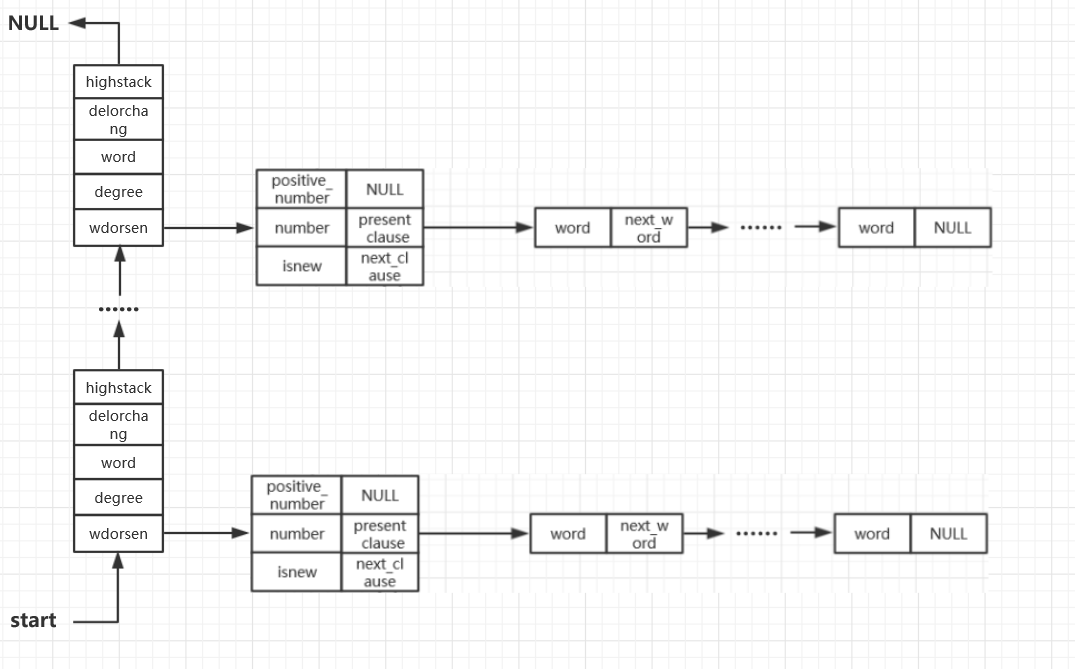


图3-2 栈的结构图

## 3.2 主要算法设计

在这次实验设计中，主要涉及到的算法设计是DPLL算法设计以及挖洞法的算法设计

### 3.2.1 DPLL算法设计

**DPLL中处理子句的算法设计：**

传入子句的指针，根据所选择的文字，对子句进行相应的操作。其中，涉及到的操作是消除与所选文字相反的负文字以及消除包含所选文字的子句，将消除的文字和子句放入栈中。具体的设计如下：

遍历指针所指向的子句，若当前子句中包含与被选取文字相反的负文字，创建新的结构入栈，delorchange保存当前操作子句的地址，word保存决策变量选取的文字，degree保存当前操作子句时所在的递归层数，wdorsen是标志位，此时标志位置0，代表当前对子句进行消除文字的操作，最后，删除子句中操作的负文字，并且释放掉空间；若当前子句中包含被选取文字，创建新的结构入栈，delorchange保存当前操作子句的地址，word保存决策变量选取的文字，degree保存当前操作子句时所在的递归层数，wdorsen是标志位，此时标志位置1，代表当前对子句进行消除子句的操作，最后，从子句集中删除该子句。

**DPLL中恢复子句集的算法设计：**

根据传入的递归层数，将栈中所有含有递归层数与当前递归层数相同的结构进行恢复操作，根据栈中保存的子句地址delorchange，被选取的文字word以及标志位wdorsen对子句集进行文字恢复或子句添加的操作，当恢复文字的时候，将新创建的文字置于子句首；当恢复子句的时候，将被删除的子句重新置于子句集首。

**DPLL中选取决策变量的算法设计：**

在此次程序设计中，采取正子句的规则进行决策变量的选取。在每个子句中均保存着当前子句包含的文字数量和正文字数量。在选取决策变量的函数中，对子句集开始遍历，当遇到文字数量与正文字数量相同的子句中，选取该子句中第一个正文字作为决策变量，并作为返回值返回到调用函数中。若没有遇到正子句，则按照变量的序号从小到大选取第一个未赋值的正文字作为决策变量，并作为返回值返回到调用函数中。

**DPLL算法设计：**

DPLL算法在选取决策变量方面进行了简单的优化，DPLL的大体结构如下所示：

**DPLL**

**{**

**while(S中存在单子句) {**

**在S中选一个单子句L；**

**依据单子句规则，利用L化简S（对子句集进行遍历，调用处理单子句的函数HandleClause）；**

**if S = Φ return(TRUE);**

**else if (S中有空子句 ) return（FALSE）；**

**}**

**基于某种策略选取变元v（调用选取决策变量函数SelectWord）；**

**S ∪ v；**

**if DPLL**

**return(TURE);**

**恢复栈中所有递归层数大于当前递归层数进行的操作；**

**S ∪ - v；**

**return DPLL;**

**}**

### 3.2.2 挖洞法算法设计

1. 首先生成至多填充11个文字合法的数独初盘。并对该初盘求解得到数独终盘，保存该终盘，作为答案。

2. 然后复制一份终盘，开始对复制份进行挖洞操作。

3. 假设在一次“挖洞”尝试中，我们试图抹去一个填有数字６的格子

步骤ｌ 将数字3依次替换为除3以外１～９的其他八个数字。 检查是否违反游戏规则。

步骤２ 调用求解器对完成步骤１后的数独题进行求解．

步骤３ 一旦求解器能得到一个解． 便立即终止求解器。 同时宣布抹去3后得到的题至少有两解。因为针对这一空格，原本填入3时已有一个来自初始终盘的解。

步骤４

只有当所有除3外的八个数依次经过步骤１的检验，同时步骤２中求解器遍历搜索后回复无解，才能保证抹去数字3后的数独题的解具有唯一性。即判定此次“挖洞”操作是合法的。

步骤5

进行剪枝优化，挖过的洞不再进行第二次挖洞，从第一行第一个开始从左到右、从上到下重复上面的步骤进行挖洞，直到挖到最后一个格子为止。

4. 最后生成具有唯一解的数独终盘，并且将终盘展示给用户。

# 4 系统实现与测试

## 4.1 系统实现

### 4.1.1 ADT操作设计

根据实验的要求以及SAT问题求解算法和数独问题算法，该实验的应包括如下的操作函数：

ClauseLink \* ReadFromFile(int \*bool\_number);（读取文件）

bool HandleClause(ClauseLink \*TempCL,Stack \*\*stack,int word,ClauseLink \*\*clauseLink,int degree);（处理子句）

bool FlashBack(ClauseLink \*\*clauseLink,Stack \*\*stack,int degree,int \*word\_positive\_value);（恢复子句集的状态）

int Selectword(ClauseLink \*clauseLink,int \*word\_positive\_value,int bool\_number);（选取决策变量）

bool MergeWord(ClauseLink \*\*clauseLink,int word);（将决策变量并入子句集）

bool DPLL(ClauseLink \*\*clauseLink,int \*word\_positive\_value,Stack \*\*stack,int degree,int bool\_number);（DPLL算法核心）

bool Print\_yes\_result(int \*word\_positive\_value, int bool\_number, double time);（输出有解的结果）

bool Print\_no\_result(double time);（输出无解的结果）

bool OutPutClause(int \*word\_positive\_value,int \*bool\_number);（打印文件内容以及结果）

bool Read\_sudo(Sudo \*sudo);（读取数独文件）

bool Trans\_sudo\_to\_cnf(Sudo \*sudo);（将数独转换为cnf文件）

ClauseLink \*Read\_sudo\_cnf(int \*bool\_number);（读取数独cnf文件）

bool Trans\_cnf\_to\_sudo(Sudo \*sudo, int \*word\_positive\_value, int bool\_number);（cnf文件转化为数独）

bool Produce\_sudo(Sudo \*sudo);（产生填充11个空的数独）

bool Judge\_sudo(Sudo \*sudo,int row,int col);（判断当前数独是否满足数独的基本要求）

bool Generate\_sudo(Sudo \*sudo);（挖洞法产生数独游戏）

bool Read\_result(Sudo \*sudo);（读取用户答案并且判断正误）

根据相关ADT的操作特性，再定义如下的结构：

typedef struct Sudo{

int matrix[ROW][COL];

}Sudo;

其中，matrix指向的只是一个二维数组，用来存储数独各个格子的值。

### 4.1.2 ADT操作实现

ReadFromFile:

该函数接受一个int指针，修改指针所指位置的值，修改为cnf文件中变元的数目。

该操作首先读取用户输入的文件名，检查是否能打开对应的cnf文件。若能打开对应文件的话，跳过cnf文件相关的注释和说明，将子句集读入到邻接表中进行存储，并且将相关信息的读取到邻接表中，子句之间用双链表进行连接。最后返回子句集邻接表的表头。

该邻接表的初始长度为0，邻接表的大小由子句的数量和文字的数目决定。

该操作的时间复杂度O(n+e)，空间复杂度O(n+e)。

HandleClause：

该操作接受一个ClauseLink指针，一个Stack双重指针，一个int类型word，一个ClauseLink双重指针和一个int整型。

该操作接受TempCL后，首先判断当前子句是否是新创建的。若是新创建的，则修改头指针clauseLink指向，让其指向下一个子句并且释放上一个子句的空间；若不是新创建的，根据word文字，遍历子句的每个文字并对符合条件的文字进行相应的操作，将消除的文字和消除的子句入stack栈进行存储，并且标记他们被操作时所在的递归深度。根据操作的结果返回相应的布尔类型。

该操作的时间复杂度O(n),空间复杂度O(e)。

FlashBack：

该操作接受一个ClauseLink双重指针，一个Stack双重指针，一个int类型degree和一个int指针。

该操作首先根据接受的degree来进行比较，来将栈中存储的深度值均和degree相等的文字或子句还原到子句集中，使得子句集恢复到上一层的状态，同时使word\_positive\_value数组中相应文字的赋值恢复到未赋值状态。每恢复一个子句或者文字后，将之前存储其相应信息的栈所占用的空间释放掉。根据操作的结果返回相应的布尔类型，恢复的文字和子句数量取决于与degree的栈数有多少层。

该操作的时间复杂度O(n),空间复杂度O(1)。

Selectword：

该操作接受一个ClauseLink指针，一个int指针和一个int类型的整数。

该操作首先根据传入的子句集头指针clauseLink来遍历整个子句集寻找正子句，若找到正子句，则选取正子句中未赋值的变量作为决策变量，并且将该决策变量以返回值的形式传入到调用函数中；若未找到正子句，则遍历赋值数组word\_positive\_value，从小开始寻找第一个未赋值变量，找到后将其作为决策变量返回到调用函数中。

该操作的时间复杂度为O(n),空间复杂度为O(1) 。

bool MergeWord(ClauseLink \*\*clauseLink,int word)：

该操作接受一个ClauseLink双重指针和一个int类型。

该操作根据传入的整型word来创建一个单子句，并且将该单子句置于子句集的头部，让子句集的头指针指向这个新创建的单子句。

该操作的时间复杂度为O(1)，空间复杂度为O(1)。

DPLL：

该操作接受一个ClauseLink指针，一个int指针，一个Stack双重指针和两个int类型整数。

该操作首先遍历整个子句集寻找单子句，若找到单子句，则进行单子句规则进行相关文字的消除和相关子句的消除。并且不断地重复寻找单子句和消除这个循环的过程，直到没有单子句。若没有单子句，首先判断子句集是否为空，若为空，则返回布尔类型true；若不为空，判断是否存在空子句，若存在空子句，则返回布尔类型false。若上述两个情况均不存在，则执行SelectWord操作寻找决策变量，然后执行操作MergeWord，然后再次进行操作DPLL。同时记录下一次DPLL的返回值，对其返回值进行判断，若返回值为false，则执行FlashBack操作，翻转上一次选择的文字，然后执行操作MergeWord，再次递归执行DPLL操作，并将其返回值返回。该过程是个不断递归的过程。

该操作的时间复杂度为O(n2)，空间复杂度为O(logn)。

Print\_yes\_result：

该操作接受一个int指针，一个int类型整数和一个浮点数time。

该操作根据一维数组word\_positive\_value将各个文字的赋值以相应的格式输出，并且输出时间time。

该操作的时间复杂度为O(n)，空间复杂度O(1)。

Print\_no\_result：

该操作接受一个浮点数time。

该操作将无解的情况以相应的格式输出，并且输出时间time。

该操作的时间复杂度为O(1)，空间复杂度O(1)。

OutPutClause：

该操作接受两个int指针。

该操作首先执行操作ReadFromFile重新将文件内容读取到邻接表中，然后根据一维数组word\_positive\_value将各个文字的赋值以规定的格式和文件内容组合起来并且打印出来，让用户能够快速直接地判断所给的解是否正确。

该操作的时间复杂度为O(n+e)，空间复杂度为O(n+e)。

Read\_sudo：

该操作接受一个Sudo指针。

该操作首先将读取用户输入的cnf格式文件的文件名，将文件中存储的数独读取到机构sudo中的二维数组中。

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

Trans\_sudo\_to\_cnf：

该操作接受一个Sudo指针。

该操作首先根据数独转化为SAT问题的规则将数独转化为相应的cnf子句集并且存储到中间文件qsx.cnf中。

该操作的时间复杂度取决于数独游戏中初盘的情况和数独转换为SAT问题的规则。

该操作的时间复杂度同样取决于数独游戏中初盘的情况和数独转换为SAT问题的规则。

Read\_sudo\_cnf：

该操作接受一个int指针。

该操作读取在Trans\_sudo\_to\_cnf中创建的qsx.cnf中间文件，将文件中的内容读取到双向链表邻接表中。然后将邻接表的表头指针返回到调用函数中。

该操作的时间复杂度为O(n+e)，空间复杂度为O(n+e)。

Trans\_cnf\_to\_sudo：

该操作接受一个Sudo指针，一个int指针和一个int类型的整数。

该操作首先将一维数组word\_positive\_value中文字的赋值为正数的存储到一维数组中，然后再根据相应的转换规则将求得的数独终盘输入到sudo中的matrix存储。

该操作的时间复杂度为O(n)，空间复杂度为O(1)。

Produce\_sudo：

该操作接受一个Sudo指针。

该操作随机产生十一个数根据相应的转换原则将所得到数转换为1-9存储到sudo中的二维数组matrix中。并且检查产生的数独初盘是否符合数独的基本要求，根据不同的检查结果返回不同的布尔类型。

该操作的时间复杂度为O(1)，空间复杂度为O(1)。

Judge\_sudo：

该操作接受一个Sudo指针和两个int类型的整数。

该操作检查以sudo中二维数组matrix存储的数独游戏是否满足数独的基本要求，最后根据不同的检查结果返回不同的布尔类型。

该操作的时间复杂度为O(n)，空间复杂度为O(1)。

Generate\_sudo：

该操作的接受一个Sudo指针。

该操作首先将sudo中二位数组matrix存储的数独复制一份。然后根据挖洞原则在备份的数独上面进行操作，最后将挖洞生成的数独初盘打印出来。

该操作的时间复杂度为O(n2)，空间复杂度为O(n)。

Read\_result：

该操作接受一个Sudo指针。该操作首先读取用户输入的数独答案，然后将用户求得的结果与sudo中matrix存储的正确结果相比较，检查用户的结果是否正确。然后根据检查结果打印不同的告知信息和返回不同的布尔类型。

该操作的时间复杂度为O(n)，空间复杂度为O(1)。

## 4.2 系统测试

开发本系统时的操作环境为Windows 10

电脑配置：Intel core i5-7200U 8G内存

编程环境

IDE：Clion

Toolchains：

Environment：MinGW version：5.0

C Compiler：mingw32-gcc

C++ Compiler：mingw32-g++

演示系统的界面如图4-1所示：

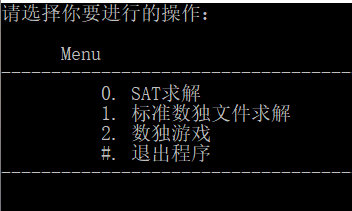


图4-1 演示界面

该演示系统操作简单，输入对应的数字并按下回车即可，输入0 - 2表明运行对应的功能，且相应操作的结果会被输出到屏幕上，输入 # 退出系统。

**SAT求解功能：**

设计目标：实现SAT算例的求解。

设计功能：SAT算例cnf文件的读取，解析，求解以及执行结果的输出与文件保存；读取cnf算例文件并实现对解析正确性的验证；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。

测试分析：在对SAT求解功能进行测试的时候，应该选取满足的算例文件以及不满足的算例文件进行多次测试，验证功能的鲁棒性。在测试的过程中，让程序输出文件中的子句和答案的结合，便于人工验算功能的正确性，同时让程序生成相应的结果文件，来验证SAT求解功能的完整性。测试表格如下：

表4-1 SAT求解功能测试表

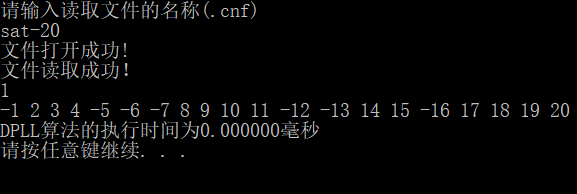
|  |  |  |
| --- | --- | --- |
| 满足的cnf算例文件 | 不满足的cnf算例文件 | 预期结果 |
| sat-20.cnf | unsat-5cnf-30.cnf | 有解，无解 |
| ais10.cnf | u-dp04u03.shuffled-825.cnf | 有解，无解 |
| sud00009.cnf | u-problem7-50.cnf | 有解，无解 |

各个文件的测试结果如下图所示(因空间有限，解析功能的验证可在功能中按照提示输出解析后的文件自行验证)：

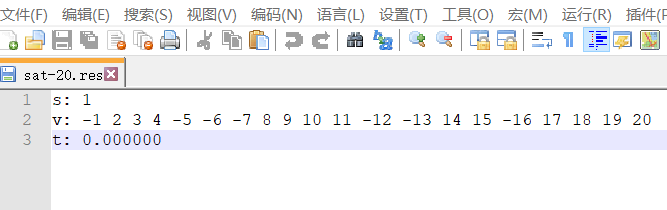
**注：为了文件内容便于截图，在截图过程中会对排版稍加变化。在测试之前已经进行了完整的测试并截图。因电脑不同情况下运行状况不同，故截图中的时间可能不一致。**

sat-20.cnf：

运行结果：

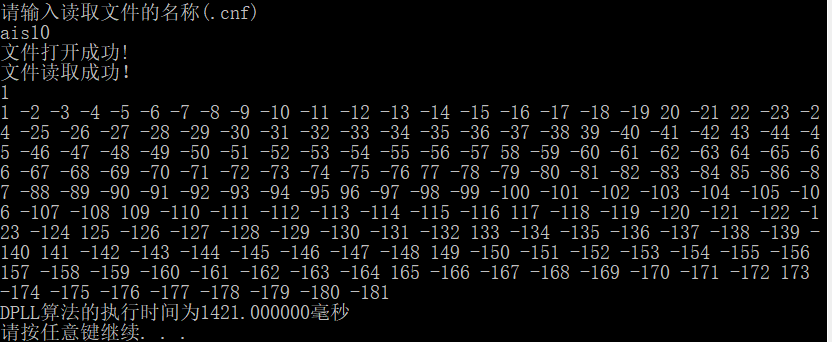


输出文件：

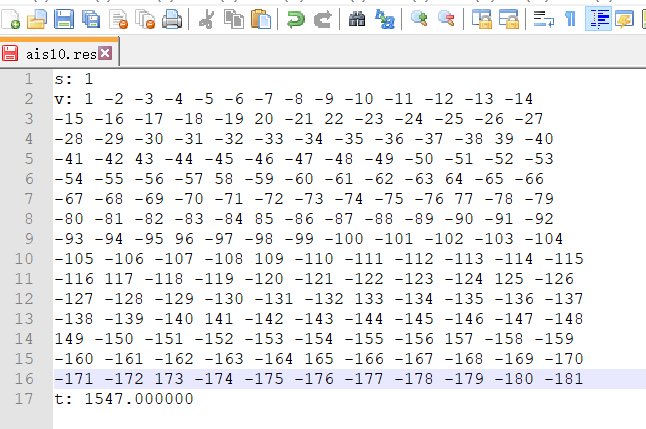


ais10.cnf：

运行结果：

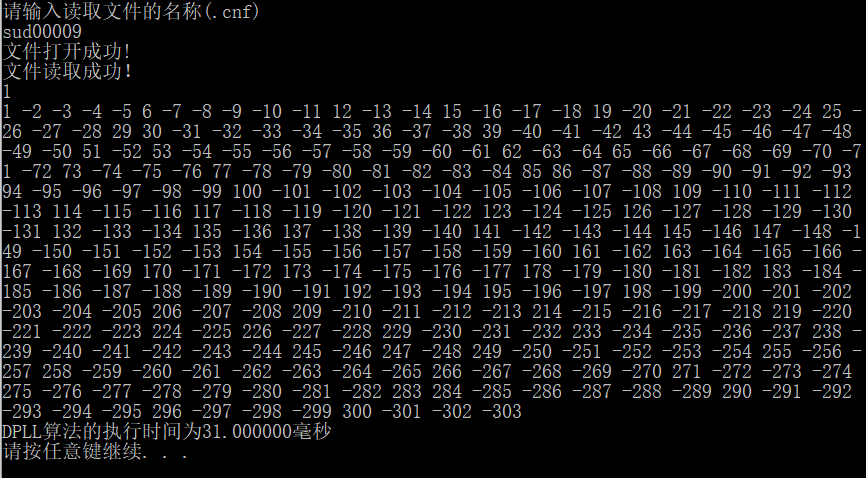


输出文件：

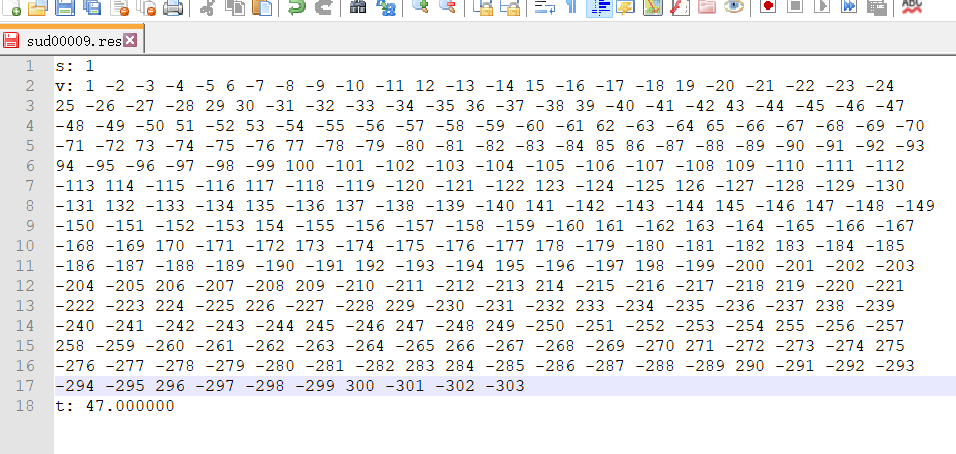


sud00009.cnf：

运行结果：

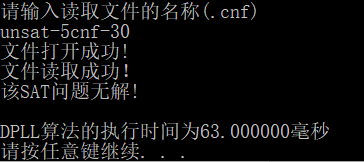


输出文件：

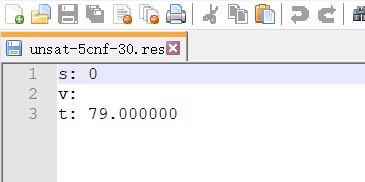


unsat-5cnf-30.cnf：

运行结果：

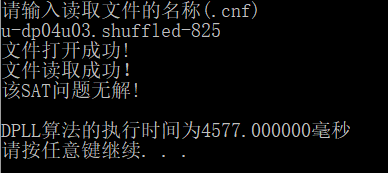


输出文件：

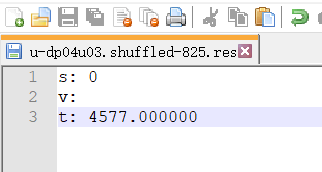


u-dp04u03.shuffled-825.cnf：

运行结果：

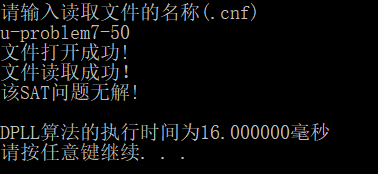


输出文件：

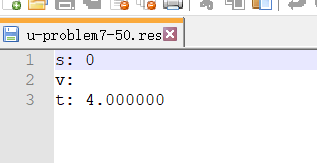


u-problem7-50.cnf：

运行结果：



输出文件：



测试结果如表：

表4-2 SAT求解功能测试对比表

|  |  |  |
| --- | --- | --- |
| 满足的cnf算例文件 | 不满足的cnf算例文件 | 与预期结果是否一致 |
| sat-20.cnf | unsat-5cnf-30.cnf | 均一致 |
| ais10.cnf | u-dp04u03.shuffled-825.cnf | 均一致 |
| sud00009.cnf | u-problem7-50.cnf | 均一致 |

由上可见，SAT求解功能是比较完善的。

**DPLL性能优化率：**

设计目标：实现DPLL算法的优化

设计功能：在选取决策变量的时候，选取正子句中的第一个文字作为决策变量，对其赋值为正，然后再进行DPLL相应的算法。

测试分析：在对优化功能进行测试的时候，应先后用不同的决策变量选取方法对SAT问题求解,并对同一文件多次求解取平均值，并进行多个文件求解，从而得到更可靠的优化率。

测试分析如表(以下表格执行时间为平均值，\*\*为时间过长或程序崩掉)：

表4-3 DPLL性能优化测试表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试文件 | 变元数 | 子句数 | 优化后执行时间 | 优化前执行时间 | 优化率 |
| sat-20.cnf | 20 | 91 | 2ms | 3ms | 33.3% |
| unsat-5cnf-30.cnf | 30 | 420 | 66ms | 85ms | 22.4% |
| ais10.cnf | 181 | 3151 | 1434ms | 1246ms | -15.1% |
| sud00009.cnf | 303 | 2851 | 25ms | 28ms | 10.7% |
| 7cnf20\_90000\_90000\_7.shuffled-20 | 20 | 1532 | 0ms | 15ms | 100% |
| problem1-20 | 20 | 91 | 0ms | 0ms | 0% |
| problem2-50 | 50 | 80 | 0ms | 219ms | 100% |
| problem3-100 | 100 | 340 | 31ms | 3749ms | 99.2% |
| problem6-50 | 50 | 100 | 0ms | 141ms | 100% |
| problem8-50 | 50 | 300 | 15ms | 0ms | -∞% |
| problem9-100 | 100 | 200 | 28509ms | 71084ms | 59.9% |
| problem11-100 | 100 | 600 | 16ms | 125ms | 87.2% |
| tst\_v25\_c100 | 25 | 100 | 0ms | 0ms | 0% |
| bart17.shuffled-231 | 231 | 1166 | 14ms | \*\* | 100% |
| problem5-200 | 200 | 320 | 89736ms | \*\* | 100% |
| problem12-200 | 200 | 1200 | 202ms | 95399ms | 99.8% |
| sud00001 | 301 | 2780 | 31ms | 31ms | 0% |
| sud00009 | 303 | 2851 | 47ms | 31ms | -51.6% |
| sud00012 | 232 | 1901 | 76ms | 125ms | 39.2% |
| sud00021 | 308 | 2911 | 53ms | 141ms | 62.4% |
| sud00079 | 301 | 2810 | 64ms | 32ms | -100% |
| sud00082 | 224 | 1762 | 62ms | 94ms | 34.0% |
| sud00861 | 297 | 2721 | 94ms | 78ms | -20.5% |
| tst\_v200\_c210 | 200 | 210 | 14ms | 16ms | 12.5% |
| ec-iso-ukn009.shuffled-as.sat05-3632-1584 | 1584 | 16587 | 557520ms | \*\* | 100% |
| ais6 | 61 | 581 | 0ms | 0ms | 0% |
| ais8 | 113 | 1520 | 63ms | 47ms | -34.0% |
| ais12 | 265 | 5666 | 47692ms | 37554ms | -27.0% |
| CBS\_k3\_n100\_m403\_b10\_0 | 100 | 403 | 0ms | 1156ms | 100% |
| CBS\_k3\_n100\_m403\_b10\_999 | 100 | 403 | 125ms | 1281ms | 90.2% |
| flat30-99 | 90 | 300 | 0ms | 16ms | 100% |

测试结论：分支变量选取策略的优化对于某些SAT问题求解效率确实有一定的优化，优化的效率取决于具体的SAT问题，某些复杂的SAT问题在优化分支变量选取策略后求解效率反而更加的低下。但是，对于大多数SAT问题求解来说，在优化分支变量选取策略后执行效率都会提高。所以说，不同的SAT问题相同的选取决策变量的方法选取出的决策变量并不一定是最有效的。这也是SAT问题不能得到最优求解算法所遇到的阻碍之一。

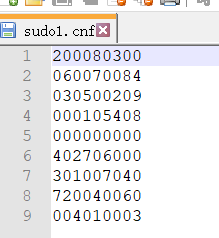
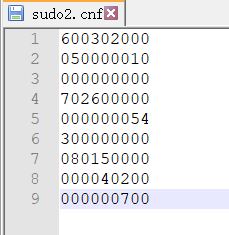
**标准数独文件求解功能：**

设计目标：实现对cnf文件中数独格式的数独游戏求解。

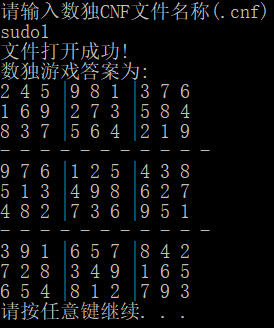
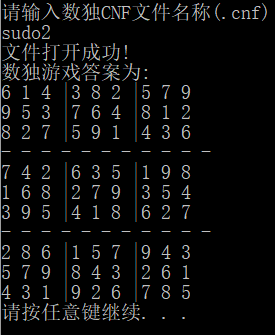
设计功能：读取数独文件(.cnf格式)，并对文件中的数独游戏求解

测试分析：标准数独文件中数独的空位用0进行表示，对多个标准数独文件进行求解，验证程序的鲁棒性。

验证文件：sudo1.cnf sudo2.cnf

运行结果如图：

由上图可知，所得的数独答案均符合数独要求，故标准数独文件求解的功能是比较完善的。

**数独游戏：**

设计目标：生成数独游戏，并且具有简单的交互性。

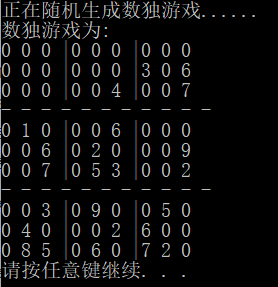
设计功能：根据挖洞法生成具有唯一解的数独，并且具有简单的交互性，能够判断用户输入的答案是否正确。

测试分析：生成多个数独，在其中输入错误的答案以及正确的答案或直接输出正确的答案，验证功能的完整性以及正确性。

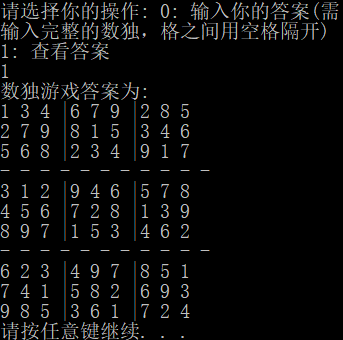
数独游戏的生成以及输入如下图所示：

数独游戏1：

生成数独如下：

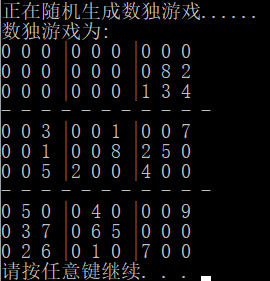


输出答案如下：

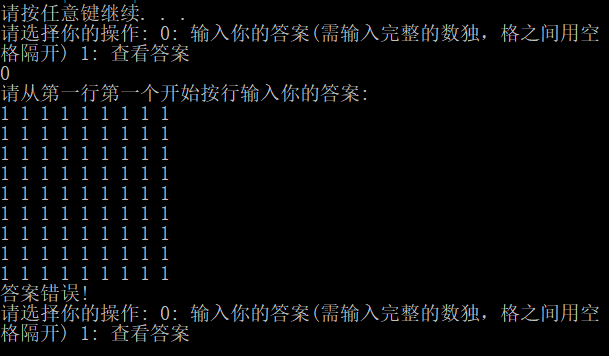


数独游戏2：

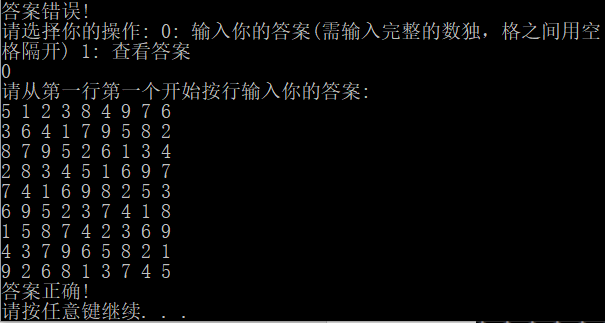
生成数独如下：



输入错误的答案：



输入正确的答案：



由上面的测试可得，数独游戏具有一定的交互性，且生成的数独游戏完好，具有唯一的解。

# 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) 在本次课程设计中，工作主要分为三部分

1.文件的读取、输出和解析

2.DPLL算法的设计

3.数独游戏的生成与求解

在各个不同的部分，均遇到了不同的难题。

**文件方面：**

要注意跳过文件开头的注释，但是仍要读取其中关于cnf文件的信息：变元的数目以及子句的数量。有一个比较容易犯错的地方，在我仔细观察后，发现有的文件的尾部仍然存在注释，如果要以读取文件尾部结束子句读取的话，会有一定的难度而且比较麻烦。在这里直接利用cnf文件中子句的数目，当读取的子句数量达到文件中所给出的数量时，就停止文件的读取，关闭文件。这样一来，读取文件的工作就得到了圆满的完成。

**DPLL方面：**

在DPLL算法方面，首先遇到的难题是当选取的变量造成SAT问题无解时，如何恢复选取变量以后对子句集所造成的改变。而这个问题，又影响了我如何具体编写消除文字和消除子句的程序。最终，选择了将被消除文字的子句地址和被删除的子句的地址统一成相同的结构，并且置于类似于栈的结构中。由于恢复操作需要删除在选择变量之后对子句集造成的所有改变，所以栈中的每一个结构又保存了入栈时的递归层数。当选取决策变量无解时，便恢复栈中所有递归层数大于选择决策变量时递归层数的操作，进而问题得到了解决。

**数独游戏方面：**

数独游戏比较困惑我的方面时如何将数独问题转换为SAT问题。在阅读相关文献以后，知道了数独游戏的约束如何转化为SAT问题，此问题就得到了良好的解决。

(2)在进行程序编程以前，首先写出了总体的模块，根据不同的模块创建不同的头文件和实现文件。对程序的设计也有了良好的思路，从而让自己的代码更加易读易懂。在写程序报告的时候，所写出的总体模块图也能应用到自己的报告中，让自己的思路更加清晰。

## 5.2 工作展望

在今后的研究中，围绕着如下几个方面开展工作：

(1)如何修改读取cnf文件时所创建的数据结构，例如：十字链表结构以及膨胀数据结构，均能够提高求解SAT问题的效率。

(2)采用多种选取变量的方法，根据读取的SAT问题中子句的数目以及变元的数目，选择更合适的分支变元选取策略。例如：计数器法、最短正子句法等。

(3)对DPLL算法实现非递归方式，改进算法的执行步骤，使得优化后的DPLL算法具有更高的求解效率。

(4)生成数独游戏时，可对挖洞的路径进行改变，进而根据用户的选择生成不同难度的数独游戏。例如：蛇形挖洞法等。

# 6 体会

在本次课程设计中，我受益匪浅。对SAT问题有了一定深度的了解，理解了SAT问题的难度所在以及研究SAT问题所具有的重大意义。在了解SAT问题背景和历史发展进程时，确实很敬佩科学家们坚持不懈的精神。本次实验中，第一次真正地接触了算法，真正地理解了什么叫做算法，数据结构不同，求解思路不同，当然所得的问题求解效率也会大大地不同。

在研究问题之前，要充足地阅读文献，并且写出自己的思路，画出自己所想的程序模块图，这样才能让自己编程序更加的快速高效，而不是在编写了很多的代码之后，意识到了自己程序的不足，前功尽弃。

在这次的课程设计中，也非常地感谢老师对我的精心指导，让我有足够的动力来面对第一次课程设计的困难，谢谢老师！

# 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. 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]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]基于“挖洞”思想的数独游戏生成算法https://www.doc88.com/p-6873249794810.html

[7] Sudoku as a SAT Problem ：<https://github.com/caistrong/Blog/blob/master/demos/sudoku-sat/Sudoku%20as%20a%20SAT%20Problem.pdf>

# 附录

**dpll.h:**

//处理子句

bool HandleClause(ClauseLink \*TempCL,Stack \*\*stack,int word,ClauseLink \*\*clauseLink,int degree);

//恢复子句的状态

bool FlashBack(ClauseLink \*\*clauseLink,Stack \*\*stack,int degree,int \*word\_positive\_value);

//选取决策变量

int Selectword(ClauseLink \*clauseLink,int \*word\_positive\_value,int bool\_number);

//将决策变量并入子句集

bool MergeWord(ClauseLink \*\*clauseLink,int word);

//DPLL算法核心

bool DPLL(ClauseLink \*\*clauseLink,int \*word\_positive\_value,Stack \*\*stack,int degree,int bool\_number);

**dpll.c:**

bool HandleClause(ClauseLink \*TempCL,Stack \*\*stack,int word,ClauseLink \*\*clauseLink,int degree){

if(TempCL->isnew == 0) {

Clause \*lastclause = TempCL -> present\_clause;

for (Clause \*clause = TempCL->present\_clause; clause != NULL;lastclause = clause, clause = clause->next\_word) {

//消除文字

if(clause->word + word == 0){

Stack \*stack1 = (Stack\*)malloc(sizeof(Stack));

stack1->delorchang = TempCL;

stack1->wdorsen = 0;

stack1->word = word;

stack1->degree = degree;

stack1->highstack = \*stack;

\*stack = stack1;

TempCL->number--;

if(clause->word > 0)

TempCL->positive\_number--;

//文字是否位于首位

if(clause == TempCL->present\_clause){

TempCL->present\_clause = clause-> next\_word;

//有空间散落

} else{

lastclause->next\_word = clause-> next\_word;

free(clause);

clause =lastclause;

}

continue;

}

//消除子句

if(clause-> word == word){

Stack \*stack1 = (Stack\*)malloc(sizeof(Stack));

stack1->delorchang = TempCL;

stack1->wdorsen = 1;

stack1->word = word;

stack1->degree = degree;

stack1->highstack = \*stack;

\*stack = stack1;

if(TempCL -> last\_clause == NULL) {

\*clauseLink = TempCL->next\_clause;

if(!(\*clauseLink))

return true;

else

(\*clauseLink)->last\_clause = NULL;

} else {

TempCL->last\_clause->next\_clause = TempCL->next\_clause;

if(TempCL->next\_clause)

TempCL->next\_clause->last\_clause = TempCL->last\_clause;

}

}

}

} else{

\*clauseLink = TempCL-> next\_clause;

if(TempCL->next\_clause != NULL)

(\*clauseLink)->last\_clause = NULL;

}

return true;

}

bool FlashBack(ClauseLink \*\*clauseLink,Stack \*\*stack,int degree,int \*word\_positive\_value){

while((\*stack)->degree == degree){

word\_positive\_value[abs((\*stack)->word) - 1] = -1;

if((\*stack)->wdorsen == 1){//恢复被删除的子句

(\*stack)->delorchang ->next\_clause = \*clauseLink;

(\*stack)->delorchang->last\_clause = NULL;

(\*clauseLink) -> last\_clause = (\*stack)->delorchang;

(\*clauseLink) = (\*stack)->delorchang;

Stack \*stack1 = \*stack;

(\*stack) = (\*stack)->highstack;

free(stack1);

}else if((\*stack)->wdorsen == 0){//恢复被删除的文字

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

clause->word = 0 - (\*stack)->word;

clause->next\_word = (\*stack)->delorchang->present\_clause;

(\*stack)->delorchang ->present\_clause = clause;

(\*stack)->delorchang->number++;

if(clause->word > 0)

(\*stack)->delorchang->positive\_number++;

Stack \*stack1 = \*stack;

(\*stack) = (\*stack)->highstack;

free(stack1);

}

}

return true;

}

int Selectword(ClauseLink \*clauseLink,int \*word\_positive\_value,int bool\_number){

int word = 0;

for (ClauseLink \*temp = clauseLink;temp!= NULL;temp = temp -> next\_clause) {

if(temp->number == temp->positive\_number && temp->number != 0){

word = temp->present\_clause->word ;

break;

}

}

if (word == 0) {//没有最短正子句时赋值

for (int i = 0; i < bool\_number; ++i) {

if (word\_positive\_value[i] == -1) {

word = i + 1;

break;

}

}

}

return word;

}

bool MergeWord(ClauseLink \*\*clauseLink,int word){

//将赋值的文字作为单元子句置于子句集首

ClauseLink \*newclauseLink = (ClauseLink \*) malloc(sizeof(ClauseLink));

newclauseLink->number = 1;

newclauseLink->isnew = 1;

if(word > 0)

newclauseLink -> positive\_number = 1;

else if(word < 0)

newclauseLink -> positive\_number = 0;

newclauseLink->present\_clause = (Clause \*) malloc(sizeof(Clause));

newclauseLink->present\_clause->word = word;

newclauseLink->present\_clause->next\_word = NULL;

newclauseLink->next\_clause = \*clauseLink;

newclauseLink->last\_clause = NULL;

if(\*clauseLink != NULL)

(\*clauseLink)->last\_clause = newclauseLink;

\*clauseLink = newclauseLink;

//结束

return true;

}

bool DPLL(ClauseLink \*\*clauseLink,int \*word\_positive\_value,Stack \*\*stack,int degree,int bool\_number){

int word;

while (true){

word = 0;

for(ClauseLink \*TempCL = \*clauseLink;TempCL != NULL;TempCL = TempCL -> next\_clause){

if(TempCL -> number == 1){

word = TempCL -> present\_clause -> word;

break;

}

}

if(!word)

break;

//给文字赋值

if(word > 0)

word\_positive\_value[word - 1] = 1;

else

word\_positive\_value[abs(word) - 1] = 0;

//赋值结束

//单子句规则

for(ClauseLink \*TempCL = \*clauseLink;TempCL != NULL;TempCL = TempCL -> next\_clause){

HandleClause(TempCL,stack,word,clauseLink,degree);

}

//判断

if(!(\*clauseLink))

return true;//找到了解

else{

for(ClauseLink \*TempCL = \*clauseLink;TempCL != NULL;TempCL = TempCL -> next\_clause){

if(TempCL-> number == 0) {

return false;//当前无解

}

}

}

}

word = Selectword(\*clauseLink,word\_positive\_value,bool\_number);//选取变量

MergeWord(clauseLink,word);//将变量并入子句集

if(DPLL(clauseLink,word\_positive\_value,stack,degree+1,bool\_number))

return true;

word = -word;

while ((\*stack)->degree > degree)

FlashBack(clauseLink,stack,(\*stack)->degree,word\_positive\_value);

MergeWord(clauseLink,word);//将变量并入子句集

return DPLL(clauseLink,word\_positive\_value,stack,degree+1,bool\_number);

}

**OutPutResult.h:**

//输出有解的结果

bool Print\_yes\_result(int \*word\_positive\_value, int bool\_number, double time);

//输出无解的结果

bool Print\_no\_result(double time);

//打印文件内容以及结果

bool OutPutClause(int \*word\_positive\_value,int \*bool\_number);

**OutPutResult.c:**

bool Print\_yes\_result(int \*word\_positive\_value, int bool\_number, double time){

FILE \*fp;

char file\_name[60];

char file\_type[] = ".res";

while (true) {

printf("请输入创建生成的CNF文件名称(.res)\n");

scanf("%s", file\_name);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

strcat(file\_name, file\_type);

fp = fopen(file\_name, "w");

if (!fp)

printf("创建文件失败!\n");

else {

printf("创建文件成功!\n");

break;

}

}

fprintf(fp,"s: 1\nv: ");

for (int i = 1; i <= bool\_number; i++) {

if (word\_positive\_value[i - 1] == -1)

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

else if (word\_positive\_value[i - 1] == 0)

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

else if (word\_positive\_value[i - 1] == 1)

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

}

fprintf(fp,"\nt: %f",time);

fclose(fp);

return true;

}

bool Print\_no\_result(double time){

FILE \*fp;

char file\_name[60];

char file\_type[] = ".res";

while (true) {

printf("请输入创建生成的CNF文件名称(.res)\n");

scanf("%s", file\_name);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

strcat(file\_name, file\_type);

fp = fopen(file\_name, "w");

if (!fp)

printf("创建文件失败!\n");

else {

printf("创建文件成功!\n");

break;

}

}

fprintf(fp,"s: 0\nv: \nt: %f",time);

fclose(fp);

return true;

}

bool OutPutClause(int \*word\_positive\_value,int \*bool\_number){

printf("打印CNF子句需要再次输入一次文件名，请按照提示进行!\n");

ClauseLink \*clauseLink = ReadFromFile(bool\_number);

for (ClauseLink \*TemporaryCL = clauseLink;TemporaryCL != NULL;TemporaryCL = TemporaryCL -> next\_clause) {

int a = 0;

for (Clause \*clause = TemporaryCL -> present\_clause;clause != NULL; clause = clause -> next\_word) {

printf("%d(",clause->word);

if(word\_positive\_value[abs(clause -> word) - 1] == -1){

printf("\*) ");

}else if(clause -> word > 0){

printf("%d) ",word\_positive\_value[clause -> word - 1]);

if(word\_positive\_value[clause -> word - 1] == 1)

a = 1;

}else if(clause -> word < 0){

printf("%d) ",1 - word\_positive\_value[abs(clause -> word) - 1]);

if(word\_positive\_value[abs(clause->word) - 1] == 0)

a = 1;

}

}

if(a == 1)

printf(" ---true");

else

printf(" ---false");

printf("\n");

}

return true;

}

**readandselect.h:**

typedef struct Clause{

struct Clause \* next\_word;

int word;

}Clause;

typedef struct ClauseLink{

int positive\_number;//一行子句中正文字数目

int number;//一行子句中文字的数目

int isnew;//子句是不是新创建的

Clause \* present\_clause;

struct ClauseLink \* next\_clause;

struct ClauseLink \* last\_clause;

}ClauseLink;

typedef struct Stack{

int word;

int degree;

int wdorsen;//0:文字 :子句

struct Stack \*highstack;

ClauseLink \*delorchang;

}Stack;

//从cnf文件中读取

ClauseLink \* ReadFromFile(int \*bool\_number);

**readandselect.c:**

ClauseLink \*ReadFromFile(int \*bool\_number) {//将文件读取到链表中

FILE \*fp = NULL;

ClauseLink \*lastClauseLink;

ClauseLink \*headClauseLink;

ClauseLink \*clauseLink;

Clause \*headclause;

char file\_name[60];

char file\_type[] = ".cnf";

printf("请输入读取文件的名称(.cnf)\n");

while (true) {

scanf("%s", file\_name);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

strcat(file\_name, file\_type);

fp = fopen(file\_name, "r");

if (!fp)

printf("输入的文件不存在，请重新输入!\n");

else {

printf("文件打开成功!\n");

break;

}

}

char words[100];

char first\_word;

while (true) {//跳过cnf格式文件的开头部分

fscanf(fp, "%c", &first\_word);

if (first\_word == 'c') {

fgets(words, 99, fp);//读取（跳过）注释部分的一整行，若此行字符数目超过99，循环后执行else部分

} else if (first\_word == 'p') {

fscanf(fp, "%s", words);

break;//开始进入正文信息部分

} else

fgets(words, 99, fp);

}

\*bool\_number = 0;

fscanf(fp, "%d", bool\_number);

int clause\_number = 0;

int word = 0;

fscanf(fp, "%d", &clause\_number);

clauseLink = lastClauseLink = headClauseLink = (ClauseLink \*) malloc(sizeof(ClauseLink));

clauseLink -> last\_clause = NULL;

//依次访问每个子句

for (int i = 0;; i++) {

int a;//文字数目

int b;//正文字数目

if (i == clause\_number) {

free(headClauseLink);

lastClauseLink->next\_clause = NULL;

printf("文件读取成功！\n");

fclose(fp);

return clauseLink;

}

//为了方便首先在循环外读取第一个文字

fscanf(fp, "%d", &word);

//为了方便首先在循环外分配一次空间

headclause = headClauseLink->present\_clause = (Clause \*) malloc(sizeof(Clause));

//读取子句中的文字

for (a = 1, b = 0;; a++) {

headclause->word = word;

if (word > 0)

b++;

//读取下一个文字

fscanf(fp, "%d", &word);

//判断该子句是否还有下一个文字

if (word == 0) {

headclause->next\_word = NULL;

break;

}

headclause->next\_word = (Clause \*) malloc(sizeof(Clause));

headclause = headclause->next\_word;

}

//初始化每个从句的状态(从句的正文字数目以及是否可以访问)

headClauseLink->positive\_number = b;

headClauseLink->number = a;

headClauseLink->isnew = 0;

lastClauseLink = headClauseLink;

headClauseLink->next\_clause = (ClauseLink \*) malloc(sizeof(ClauseLink));

headClauseLink->next\_clause->last\_clause = lastClauseLink;

headClauseLink = headClauseLink->next\_clause;

}

}

**sudoku.h:**

typedef struct Sudo{

int matrix[ROW][COL];

}Sudo;

//读取数独文件

bool Read\_sudo(Sudo \*sudo);

//将数独转换为cnf文件

bool Trans\_sudo\_to\_cnf(Sudo \*sudo);

//读取数独cnf文件

ClauseLink \*Read\_sudo\_cnf(int \*bool\_number);

//将cnf文件转化为数独

bool Trans\_cnf\_to\_sudo(Sudo \*sudo, int \*word\_positive\_value, int bool\_number);

//产生填充11个空的数独

bool Produce\_sudo(Sudo \*sudo);

//判断当前数独是否满足数独的基本要求

bool Judge\_sudo(Sudo \*sudo,int row,int col);

//挖洞法产生数独游戏

bool Generate\_sudo(Sudo \*sudo);

//读取用户答案并且判断正误

bool Read\_result(Sudo \*sudo);

**sudoku.c:**

FILE \*fp\_1 = NULL;

bool Read\_sudo(Sudo \*sudo){

char file\_name[60];

char file\_type[] = ".cnf";

while (true) {

printf("请输入数独CNF文件名称(.cnf)\n");

scanf("%s", file\_name);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

strcat(file\_name, file\_type);

fp\_1 = fopen(file\_name, "r");

if (!fp\_1)

printf("输入的文件不存在，请重新输入!\n");

else {

printf("文件打开成功!\n");

break;

}

}

char a;

for (int x = 0; x < ROW; ++x) {

for (int y = 0; y < COL; ++y) {

fscanf(fp\_1,"%c",&a);

sudo->matrix[x][y] = a - '0';

}

fscanf(fp\_1,"%c",&a);

}

fclose(fp\_1);

fp\_1 = NULL;

return true;

}

bool Trans\_sudo\_to\_cnf(Sudo \*sudo){

char file\_name[60] = "qsx.cnf\0";

fp\_1 = fopen(file\_name, "w");

if (!fp\_1)

exit(0);

int bool\_number = 729;

int clause\_number = 8829;

//统计子句数量

for (int x = 0; x < ROW; ++x) {

for (int y = 0; y < COL; ++y) {

if (sudo->matrix[x][y] != 0) {

clause\_number++;

}

}

}

fprintf(fp\_1,"p cnf %d %d\n",bool\_number,clause\_number);

//single clause

for (int x = 0; x < ROW; ++x) {

for (int y = 0; y < COL; ++y) {

if (sudo->matrix[x][y] != 0) {

fprintf(fp\_1, "%d %d\n", x \* 81 + y \* 9 + sudo->matrix[x][y], 0);

clause\_number++;

}

}

}

//There is at least one number in each entry(格限制):

for (int x = 0; x <= 8; ++x) {

for (int y = 0; y <= 8; ++y) {

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

fprintf(fp\_1,"%d ",x \* 81 + y \* 9+ z);

fprintf(fp\_1,"%d\n",0);

}

}

//Each number appears at most once in each row(行限制):

for (int y = 0; y <= 8; ++y) {

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

for (int x = 0; x <= 7; ++x)

for (int i = x+1; i <= 8; ++i)

fprintf(fp\_1,"%d %d %d\n",0 - (y\*81 + x\*9 + z),0 - (y\*81 + i\*9 + z),0);

}

//Each number appears at most once in each column(列限制):

for (int x = 0; x <= 8; ++x) {

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

for (int y = 0; y <= 7; ++y)

for (int i = y+1; i <= 8; ++i)

fprintf(fp\_1,"%d %d %d\n",0-(y\*81 + x\*9 + z),0-(i\*81 + x\*9 + z),0);

}

//Each number appears at most once in each 3x3 sub-grid(九宫格限制):

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

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

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

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

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

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

fprintf(fp\_1,"%d %d %d\n",0 - (((3\*i+x)-1)\*81 + ((3\*j+y)-1)\*9 + z),0-(((3\*i+x)-1)\*81 + ((3\*j+k)-1)\*9 + z),0);

}

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

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

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

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

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

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

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

fprintf(fp\_1,"%d %d %d\n",0 - (((3\*i+x)-1)\*81 + ((3\*j+y)-1)\*9 + z),0 - (((3\*i+k)-1)\*81 + ((3\*j+l)-1)\*9 + z),0);

}

fclose(fp\_1);

return true;

}

ClauseLink \*Read\_sudo\_cnf(int \*bool\_number){

fp\_1 = fopen("qsx.cnf","r");

ClauseLink \*lastClauseLink;

ClauseLink \*headClauseLink;

ClauseLink \*clauseLink;

Clause \*headclause;

char words[100];

char first\_word;

while (true) {//跳过cnf格式文件的开头部分

fscanf(fp\_1, "%c", &first\_word);

if (first\_word == 'c') {

fgets(words, 99, fp\_1);//读取（跳过）注释部分的一整行，若此行字符数目超过99，循环后执行else部分

} else if (first\_word == 'p') {

fscanf(fp\_1, "%s", words);

break;//开始进入正文信息部分

} else

fgets(words, 99, fp\_1);

}

\*bool\_number = 0;

fscanf(fp\_1, "%d", bool\_number);

int clause\_number = 0;

int word = 0;

fscanf(fp\_1, "%d", &clause\_number);

clauseLink = lastClauseLink = headClauseLink = (ClauseLink \*) malloc(sizeof(ClauseLink));

clauseLink -> last\_clause = NULL;

//依次访问每个子句

for (int i = 0;; i++) {

int a;//文字数目

int b;//正文字数目

if (i == clause\_number) {

free(headClauseLink);

lastClauseLink->next\_clause = NULL;

fclose(fp\_1);

fp\_1 = NULL;

return clauseLink;

}

//为了方便首先在循环外读取第一个文字

fscanf(fp\_1, "%d", &word);

//为了方便首先在循环外分配一次空间

headclause = headClauseLink->present\_clause = (Clause \*) malloc(sizeof(Clause));

//读取子句中的文字

for (a = 1, b = 0;; a++) {

headclause->word = word;

if (word > 0)

b++;

//读取下一个文字

fscanf(fp\_1, "%d", &word);

//判断该子句是否还有下一个文字

if (word == 0) {

headclause->next\_word = NULL;

break;

}

headclause->next\_word = (Clause \*) malloc(sizeof(Clause));

headclause = headclause->next\_word;

}

//初始化每个从句的状态(从句的正文字数目以及是否可以访问)

headClauseLink->positive\_number = b;

headClauseLink->number = a;

headClauseLink->isnew = 0;

lastClauseLink = headClauseLink;

headClauseLink->next\_clause = (ClauseLink \*) malloc(sizeof(ClauseLink));

headClauseLink->next\_clause->last\_clause = lastClauseLink;

headClauseLink = headClauseLink->next\_clause;

}

}

//数独cnf的解都存储在一个一维数组中

bool Trans\_cnf\_to\_sudo(Sudo \*sudo, int \*word\_positive\_value,int bool\_number){

int result[ROW \* COL];

for (int j = 0; j < bool\_number; ++j) {

if(word\_positive\_value[j] == 0)

word\_positive\_value[j] = -(j+1);

else if(word\_positive\_value[j] == 1)

word\_positive\_value[j] = j + 1;

else if(word\_positive\_value[j] == -1)

return false;

}

for (int i = 0,a = 0; i < bool\_number; ++i) {

if (word\_positive\_value[i] > 0) {

result[a] = word\_positive\_value[i] % 9;

if (!result[a])

result[a] = 9;

a++;

}

}

for(int row = 0;row < ROW;++row){

for (int col = 0; col < COL; ++col) {

sudo->matrix[row][col] = result[9\*row + col];

}

}

return true;

}

bool Produce\_sudo(Sudo \*sudo){

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

sudo->matrix[row][col] = 0;

}

}

srand((unsigned)time(NULL));

//随机填充11个空

for (int i = 0; i < 11; i++) {

int keyword = rand() % 729;

int row = keyword / 81;

int col = (keyword % 81) / 9;

int data = keyword % 9 + 1;

sudo->matrix[row][col] = data;

}

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

if(sudo->matrix[row][col] != 0){

for (int j = 0; j < COL; ++j) {

if(sudo->matrix[row][col] == sudo->matrix[row][j] && j != col)

return false;

}

for (int i = 0; i < ROW; ++i) {

if(sudo->matrix[row][col] == sudo->matrix[i][col] && i != row)

return false;

}

for (int i = (row/3)\*3; i < (row/3)\*3 + 3; ++i) {

for (int j = (col/3)\*3; j < (col/3)\*3 + 3; ++j) {

if(sudo->matrix[row][col] == sudo->matrix[i][j] && (i != row || j != col))

return false;

}

}

}

}

}

return true;

}

bool Judge\_sudo(Sudo \*sudo,int row,int col){

if(sudo->matrix[row][col] != 0){

for (int j = 0; j < COL; ++j) {

if(sudo->matrix[row][col] == sudo->matrix[row][j] && j != col)

return false;

}

for (int i = 0; i < ROW; ++i) {

if(sudo->matrix[row][col] == sudo->matrix[i][col] && i != row)

return false;

}

for (int i = (row/3)\*3; i < (row/3)\*3 + 3; ++i) {

for (int j = (col/3)\*3; j < (col/3)\*3 + 3; ++j) {

if(sudo->matrix[row][col] == sudo->matrix[i][j] && (i != row || j != col))

return false;

}

}

}

return true;

}

//挖洞法产生数独

bool Generate\_sudo(Sudo \*sudo){

Sudo \*sudo1 = (Sudo\*)malloc(sizeof(Sudo));

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

sudo1->matrix[row][col] = sudo->matrix[row][col];

}

}

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

int i = 1;

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

if(i != sudo->matrix[row][col]){

sudo1->matrix[row][col] = i;

//先判断是否满足数独基本要求

if(!Judge\_sudo(sudo1,row,col))

continue;

//判断结束

//判断是否有解

Trans\_sudo\_to\_cnf(sudo1);

int bool\_number;//布尔变量数量

int degree = 1;//所在的层

ClauseLink \*clauseLink;//子句链表头

clauseLink = Read\_sudo\_cnf(&bool\_number);

int word\_positive\_value[bool\_number];//正文字的赋值数组

Stack \*stack = (Stack\*)malloc(sizeof(Stack));

stack->degree = -1;

stack->wdorsen = -1;

stack->highstack = NULL;

stack->word = 0;

stack->delorchang = NULL;

for (int p = 0; p < bool\_number; ++p) {

word\_positive\_value[p] = -1;//初始化数组

}

if (DPLL(&clauseLink,word\_positive\_value,&stack,degree,bool\_number))

break;

else

continue;

}

}

if (i == 10)

sudo1->matrix[row][col] = 0;

else

sudo1->matrix[row][col] = sudo->matrix[row][col];

}

}

//打印数独游戏

printf("数独游戏为:\n");

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

printf("%d ",sudo1->matrix[row][col]);

if(col == 2 || col == 5)

printf("|");

}

printf("\n");

if(row == 2 || row == 5)

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

}

free(sudo1);

return true;

}

bool Read\_result(Sudo \*sudo){

printf("请从第一行第一个开始按行输入你的答案:\n");

int result[ROW][COL];

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

scanf("%d",&(result[row][col]));

}

}

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

if(sudo->matrix[row][col] != result[row][col]) {

printf("答案错误!\n");

return false;

}

}

}

printf("答案正确!\n");

return true;

}

**main.c:**

int main() {

while (true) {

char a;

while (true) {

//用户选择要进行的操作

printf("请选择你要进行的操作：\n");

printf("\n");

printf(" Menu \n");

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

printf(" 0. SAT求解\n");

printf(" 1. 标准数独文件求解\n");

printf(" 2. 数独游戏\n");

printf(" #. 退出程序\n");

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

scanf("%c", &a);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

if (a != '0' && a != '1' && a != '#' && a != '2') {

printf("输入的选择不合法，回到菜单!\n");

continue;

} else

break;

}

//根据用户的选择执行不同的模块

switch (a) {

case '0': {

int bool\_number;//布尔变量数量

int degree = 1;//所在的层

clock\_t start, finish;

ClauseLink \*clauseLink;//子句链表头

clauseLink = ReadFromFile(&bool\_number);

int word\_positive\_value[bool\_number];//正文字的赋值数组

Stack \*stack = (Stack\*)malloc(sizeof(Stack));

stack->degree = -1;

stack->wdorsen = -1;

stack->highstack = NULL;

stack->word = 0;

stack->delorchang = NULL;

start = clock();

for (int i = 0; i < bool\_number; ++i) {

word\_positive\_value[i] = -1;//初始化数组

}

if (DPLL(&clauseLink,word\_positive\_value,&stack,degree,bool\_number)) {

printf("1\n");

for (int i = 1; i <= bool\_number; i++) {

if (word\_positive\_value[i - 1] == -1)

printf("\*%d ", i);

else if (word\_positive\_value[i - 1] == 0)

printf("-%d ", i);

else if (word\_positive\_value[i - 1] == 1)

printf("%d ", i);

}

finish = clock();

printf("\nDPLL算法的执行时间为%f毫秒\n", (double) (finish - start) / CLOCKS\_PER\_SEC\*1000);

system("pause");

printf("是否打印CNF结果? 是：1 否：0\n");

char choice;

while (true) {

scanf("%c", &choice);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

if (choice == '1') {

Print\_yes\_result(word\_positive\_value,bool\_number, (double) (finish - start) / CLOCKS\_PER\_SEC\*1000);

break;

} else if (choice == '0')

break;

else

printf("输入的指令不合法，请重新输入!\n");

}

//人工判断

printf("是否人工判断CNF结果? 是(需要再次输入文件名)：1 否：0\n");

while (true) {

scanf("%c", &choice);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

if (choice == '1') {

OutPutClause(word\_positive\_value,&bool\_number);

break;

} else if (choice == '0')

break;

else

printf("输入的指令不合法，请重新输入!\n");

}

system("pause");

} else {

printf("该SAT问题无解!\n");

finish = clock();

printf("\nDPLL算法的执行时间为%f毫秒\n", (double) (finish - start) / CLOCKS\_PER\_SEC\*1000);

system("pause");

printf("是否打印CNF结果? 是：1 否：0\n");

char choice;

while (true) {

scanf("%c", &choice);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

if (choice == '1') {

Print\_no\_result((double) (finish - start) / CLOCKS\_PER\_SEC\*1000);

break;

} else if (choice == '0')

break;

else

printf("输入的指令不合法，请重新输入!\n");

}

system("pause");

}

//释放多余空间

free(stack);

break;

}

case '1': {

Sudo \*sudo = (Sudo\*)malloc(sizeof(Sudo));

Read\_sudo(sudo);

Trans\_sudo\_to\_cnf(sudo);

int bool\_number;//布尔变量数量

int degree = 1;//所在的层

ClauseLink \*clauseLink;//子句链表头

clauseLink = Read\_sudo\_cnf(&bool\_number);

int word\_positive\_value[bool\_number];//正文字的赋值数组

Stack \*stack = (Stack\*)malloc(sizeof(Stack));

stack->degree = -1;

stack->wdorsen = -1;

stack->highstack = NULL;

stack->word = 0;

stack->delorchang = NULL;

for (int i = 0; i < bool\_number; ++i) {

word\_positive\_value[i] = -1;//初始化数组

}

if (DPLL(&clauseLink,word\_positive\_value,&stack,degree,bool\_number)){

Trans\_cnf\_to\_sudo(sudo,word\_positive\_value,bool\_number);

printf("数独游戏答案为:\n");

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

printf("%d ",sudo->matrix[row][col]);

if(col == 2 || col == 5)

printf("|");

}

printf("\n");

if(row == 2 || row == 5)

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

}

}else {

printf("数独无解!\n");

}

system("pause");

//释放多余空间

free(stack);

break;

}

case '2': {

printf("正在随机生成数独游戏......\n");

Sudo \*sudo = (Sudo\*)malloc(sizeof(Sudo));

int bool\_number = 729;//布尔变量数量

int degree = 1;//所在的层

ClauseLink \*clauseLink;//子句链表头

int word\_positive\_value[729];//正文字的赋值数组

Stack \*stack = (Stack\*)malloc(sizeof(Stack));

stack->degree = -1;

stack->wdorsen = -1;

stack->highstack = NULL;

stack->word = 0;

stack->delorchang = NULL;

for (int i = 0; i < 729; ++i) {

word\_positive\_value[i] = -1;//初始化数组

}

while(true) {

if(Produce\_sudo(sudo) && Trans\_sudo\_to\_cnf(sudo)){

clauseLink = Read\_sudo\_cnf(&bool\_number);

if(!DPLL(&clauseLink,word\_positive\_value,&stack,degree,bool\_number))

continue;

Trans\_cnf\_to\_sudo(sudo,word\_positive\_value,bool\_number);

break;

}

}

//挖洞法产生

Generate\_sudo(sudo);

system("pause");

//读取用户输入并且判断正误

char c;

while (true){

printf("请选择你的操作: 0: 输入你的答案(需输入完整的数独，格之间用空格隔开) 1: 查看答案\n");

scanf("%c",&c);

setbuf(stdin, NULL);//使stdin输入流由默认缓冲区转为无缓冲区

if(c == '0'){

if(Read\_result(sudo))

break;

} else if(c == '1'){

printf("数独游戏答案为:\n");

for (int row = 0; row < ROW; ++row) {

for (int col = 0; col < COL; ++col) {

printf("%d ",sudo->matrix[row][col]);

if(col == 2 || col == 5)

printf("|");

}

printf("\n");

if(row == 2 || row == 5)

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

}

break;

} else

printf("输入的指令不合法，请重新输入!\n");

}

system("pause");

//释放多余空间

free(stack);

break;

}

case '#': {

printf("即将退出程序!感谢您的使用!\n");

return 0;

}

}

}

}