**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：**

**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**

**1.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。

**2.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

[11] 严蔚敏等. 数据结构(C语言版). 北京：清华大学出版社

**目 录**

[1引言 1](#_Toc5551118)

[1.1 课题背景与意义 1](#_Toc5551119)

[1.2 国内外研究现状 1](#_Toc5551120)

[1.3 课程设计的主要研究工作 1](#_Toc5551121)

[2 系统需求分析与总体设计 1](#_Toc5551122)

[2.1 系统需求分析 2](#_Toc5551123)

[2.2 系统总体设计 2](#_Toc5551124)

[3 系统详细设计 3](#_Toc5551125)

[3.1 有关数据结构的定义 3](#_Toc5551126)

[3.2 主要算法设计 4](#_Toc5551127)

[4.系统实现与测试 11](#_Toc5551128)

[4.1 系统实现 11](#_Toc5551129)

[4.2 系统测试 19](#_Toc5551130)

[4.3程序优化 23](#_Toc5551131)

[5 总结与展望 24](#_Toc5551132)

[5.1 全文总结 25](#_Toc5551133)

[5.1 工作展望 25](#_Toc5551134)

[6体会 26](#_Toc5551135)

[附录 27](#_Toc5551136)

[1. 主程序 27](#_Toc5551137)

[2. 文件操作 31](#_Toc5551138)

[3. SAT求解，DPLL算法核心框架 32](#_Toc5551139)

[4. 数独求解模块 51](#_Toc5551140)

**1引言**

**1.1 课题背景与意义**

可满足性问题一直是人工智能领域研究的一个核心问题，其应用范围，不仅仅在人工智能，还包括计算机辅助设计，机器视觉，数据库等。合取范式的可满足性问题是理论计算机科学与人工智能的著名问题，寻求其有效算法，一直是计算机连及实际应用的重要任务。

本文简单的实现了基于DPLL的SAT算法，采用邻接表的数据结构和最短正字句的决策方式，并将算法应用在数独问题的求解过程中。

**1.2 国内外研究现状**

多年来，国际上已提出各种不同的局部搜索算法和回溯算法，使得S不同领域中SAT问题的解决能力增强，国际上提出的一大批采用回溯算法基本都是在DPLL回溯算法上，在变量决策，推理回溯等方面的优化。

在可满足问题研究的突破性进展，使得直接或者间接的推动人工智能等领域的突破性进展。目前SAT求解器可以高效地处理数百万变量规模的问题。然而,由于SAT问题本身的特性,除非P=NP,否则不存在最坏情况下多项式阶时间复杂度的SAT求解算法，因此设计出高度的SAT求解算法仍然是今天研究的热点。

**1.3 课程设计的主要研究工作**

DPLL算法是经典的SAT完备型求解算法，对给定的一个SAT问题实例，理论上可判定其是否满足，满足时可给出对应的一组解。本设计基于DPLL的算法与程序框架，实现一个完备SAT求解器，包括程序的改进也必须在此算法的基础上进行，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。

**2 系统需求分析与总体设计**

**2.1 系统需求分析**

本设计基于DPLL的算法与程序框架，实现一个完备SAT求解器，包括程序的改进也必须在此算法的基础上进行，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。

这部分应该写的是用户需求，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。

**2.2 系统总体设计**

这部分可根据用户需求，设计和规划一个系统，说明清楚系统应该有哪些功能模块，每个模块做什么。最后给出完整的系统模块结构图。

本任务只要包括六个主要的版块。

1. 输入输出功能，即执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存；
2. 公式解析与验证：读取cnf算例文件，解析文件，基于裂解表的物理结构，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[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。

以下是文中有图标的形式，要求对图标引用说明。

1. 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。

其完整的系统模块结构图如图2-1所示。

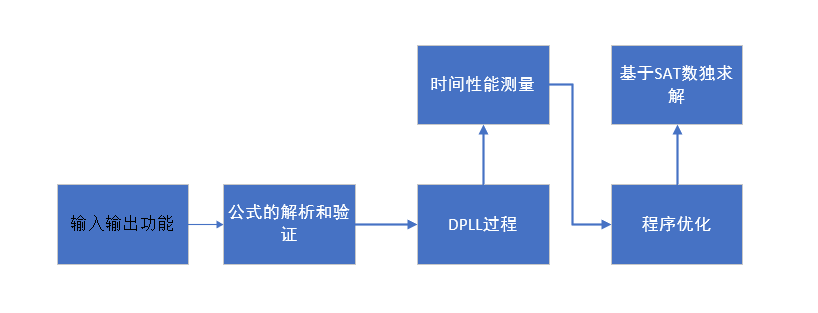


图2-1 系统模块结构图

**3 系统详细设计**

**3.1 有关数据结构的定义**

这部分要写的：（1）首先描述系统中要处理那些数据，每种类型的数据包括哪些数据项，每个数据项的数据类型，最后可用一个表格表示出来；（2) 描述这多种数据在系统中如何关联，可通过图直观的说明这多种数据间的关联。

1. 系统中压需要处理的数据，以及每种数据所包含的数据项，如表3-1所示。

表3-1 数据以及数据项

|  |  |  |
| --- | --- | --- |
| 需要处理的数据 | 所包含的数据项 | 数据类型 |
| CNF文件 | 字句行数 | 整型 |
| 变元个数 | 整型 |
| 字句 | 整型 |
| 变元 | 整型 |

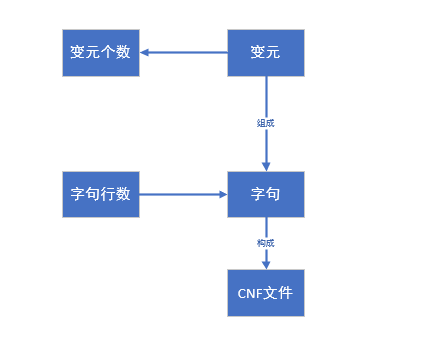
1. 数据之间的关联，如图3-1所示。

图3-1 多种数据间的关联

**3.2 主要算法设计**

给出主要部分流程图以及相关说明；

设计总体采用邻接表的数据结构，用来存储变元，字句等各种信息，具体数据结构及各模块的数据结构如下所述。

存储CNF的数据结构，如图3-2所示。

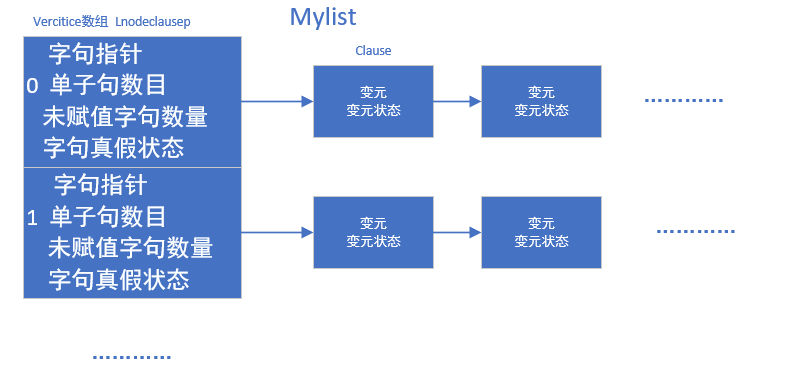


图3-2 存储CNF的数据结构

存储变元的及其赋值状态等各信息的数据结构，如图3-3所示。

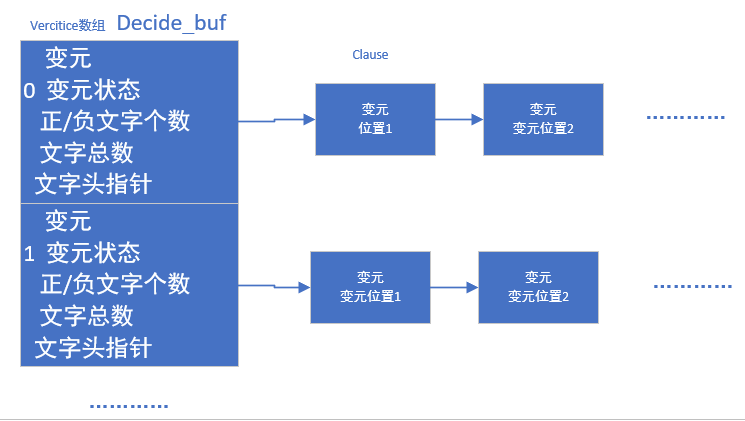


图3-3 存储变元信息的数据结构

1. 文件操作模块，包括读入变元，创建链表等步骤，其数据结构如图 3-4 所示。

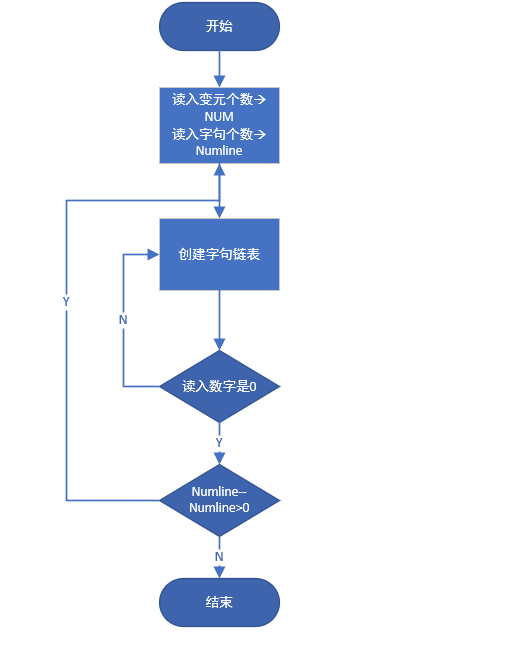


图 3-4 文件操作模块

1. DPLL模块，其主要代码框架如下。

**DPLL( *S*) :**

/\* *S*为公式对应的子句集。若其满足，返回TURE；否则返回FALSE. \*/

**{**

**while(*S*中存在单子句) {**//单子句传播

**在*S*中选一个单子句*L*；**

**依据单子句规则，利用*L*化简*S*；**

**if *S* = Φ return(TRUE);**

**else if (*S*中有空子句 ) return（FALSE）；**

**}**//while

**基于最短正字句策略选取变元*v*；//策略对DPLL性能影响很大**

**if DPLL（*S* ∪*v* ）return(TURE);**

**return DPLL(*S* ∪¬*v*);**

**}**

DPLL中，使用的函数有回溯，查找单子句，用但字句规则化简，查找当前最短正字句，判断冲突，判断cnf文件真假状态，配合DPLL完成CNF文件的求解。

其函数名如下，具体代码见工程文件。

status Conflict(MyList \*NewList)//判断冲突

status evalute\_All\_Cluase(MyList \*NewList)/判断cnf文件的当前真假状态

status isUnitClause(MyList \*NewList)//判断是否为单子句

status variable\_decide(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf) //最短正字句规则,返回当前最短的正字句序

status bcp\_deduce(MyList \*NewList, status j\_index,status p\_n\_text,Association\_Data \*Record\_Data,Decide\_buffer \*Decide\_buf) //执行BCP推导过程，利用化简策略对字句集化简

其DPLL框架流程图如图 3-5 所示。

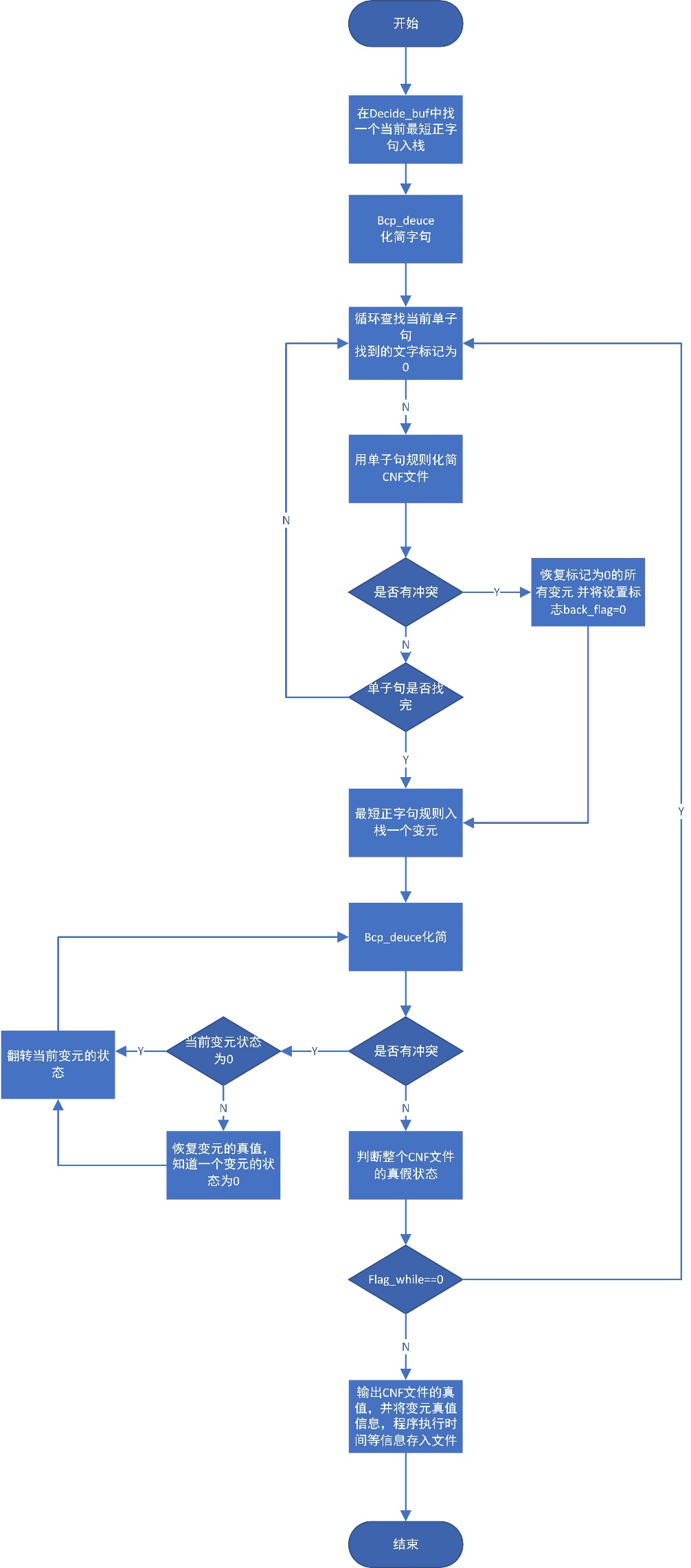


图 3-5 DPLL算法框架

1. 数独求解模块。

该模块包括两部分的内容；

1. 数独终盘的生成；主要包括生成9\*9的数独，第一步，随机生成15个数填入数独终盘，其中每生成一个数，都要根据数独的规则都要判断其填入的合法性；第二步，将所有的约束条件转换为CNF写入文件，并将已经填入的数字作为单子句也写入CNF文件；第三步，调用DPLL算法，求解CNF文件并转换为数独游戏生成数独终盘。

其生成流程图如图 3-6 所示。

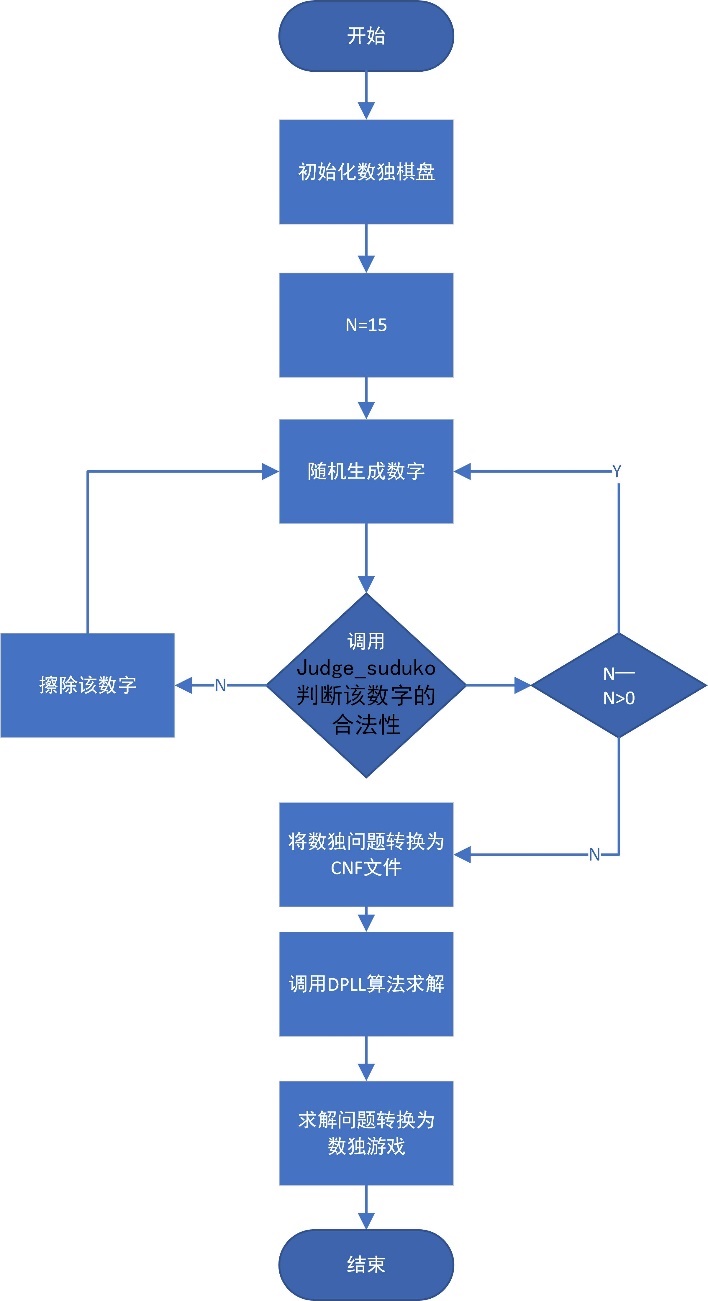


图 3-6 数独终盘生成

1. 将数独求解转换为DPLL文件。主要包括每个单元的约束条件，如对于（1，1）有 111 112 113 114 115 116 117 118 119 0

两两组合： -111 -112 0

-111 -113 0

…………

行约束，考虑第一行 包含1：111 121 131 141 151 161 171 181 191 0

包含2：112 122 132 142 152 162 172 182 192 0

…………

两两组合：-111 -121 0

-111 -131 0

列约束，考虑第一列 包含1：111 121 131 141 151 161 171 181 191 0

包含2：112 122 132 142 152 162 172 182 192 0

…………

两两组合：-111 -211 0

-111 -231 0

块约束，考虑第一块 包含1：111 121 131 211 221 231 311 321 331 0

包含2：112 122 132 212 222 232 312 322 332 0

…………

两两组合：-111 -211 0

-111 -231 0

单子句约束，如（3,6）包含 5有单子句 365 0

其流程图如图 3-7所示。

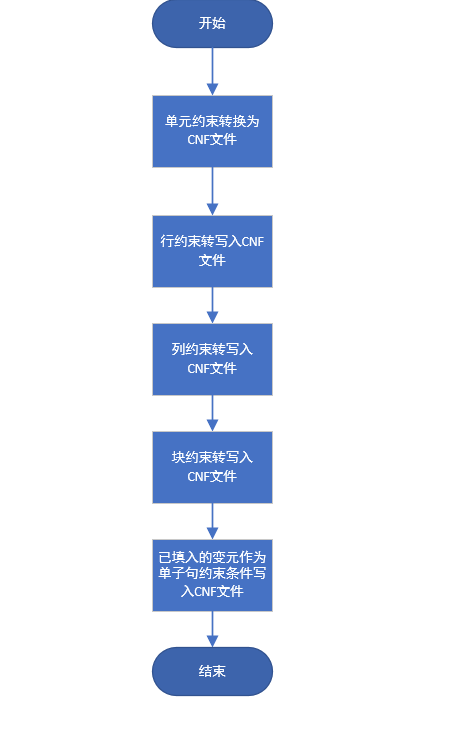


图 3-7 数独求解转换为CNF文件

1. 挖洞法生成数独游戏；

使用产生随机数的方式产生横坐标，纵坐标并擦掉相应位置的数字，回到生成步骤二，并将生成的CNF文件求解后转化为数独游戏，并输出；

其流程图如图 3-8 所示；

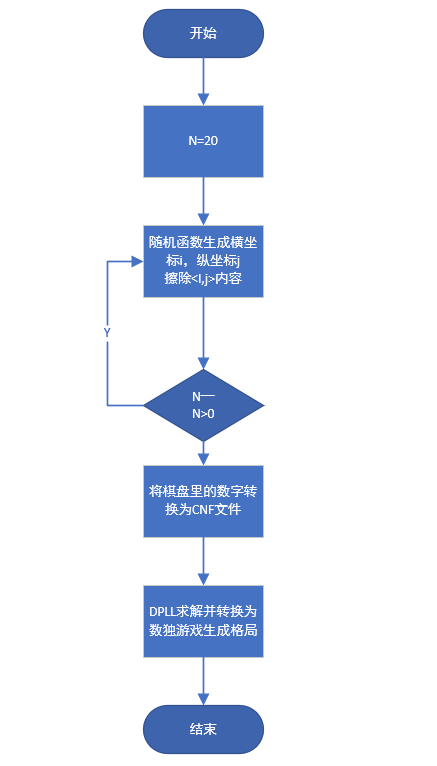


图 3-8 挖洞法生成数独游戏

1. 数独游戏的求解：数独游戏的步骤雷同步骤（1）有将数独棋盘转换为CNF文件，并用DPLL算法求解，最后转换为数独游戏即可。

**4.系统实现与测试**

**4.1 系统实现**

1. 系统实现环境

软件环境：visual studio2010；

硬件环境：intel 酷睿i7-7500 内存8G；

1. 用户需求：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，并将DPLL求解应用到数独求解具体问题中；
2. 处理流程，如图 4-1 所示。

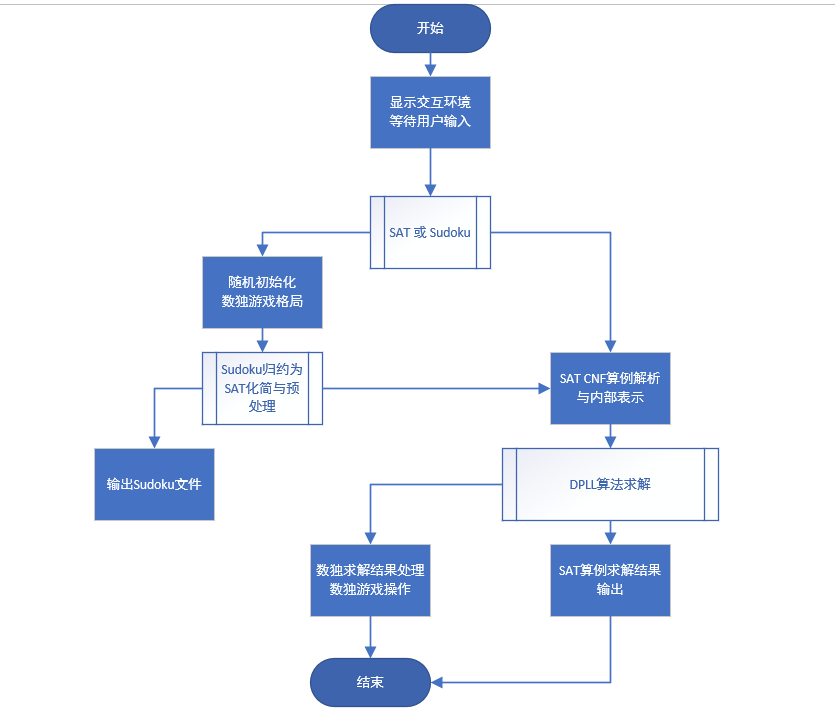


图 4-1 设计处理流程

1. 用C语言定义的各种数据结构；
2. CNF文件存储结构，采用邻接表定义，物理结构见图 3-1；

函数如下所示，完整函数参见附录；

typedef struct LnodeClause {

Clause \*firstnode;

int num\_positive\_text;//正文字个数

int num\_negative\_text;//负文字个数

int ClauseNum;//记录该字句的单子句数目

int Number\_of\_assigned\_words;//未赋值字句数量

int Determine;//判断该字句的真假状态 0 为假 ，1 为真，-1 为不确定

}LnodeClause, LnodeClausep[MAXLIST];

typedef struct MyList {

LnodeClausep vertices;

int num, numLine;

}MyList;

1. 邻接表存储变元的出现次数，位置，赋值状态等各种信息；

函数，完整代码见附录；

typedef struct Decide\_buffer {//文字计数器

status Data;

status Data\_status;

int num\_positive\_text;//正文字个数

int num\_negative\_text;//负文字个数

int total\_number;//正负文字总数

Clause \*firstnode;

};

typedef struct Clause {

Data\_status elem;

struct Clause \*next;

}Clause, \*ClauseList;

//子句结点类型定义

1. 数组模拟进出栈

函数，完整代码见附录；

typedef struct Record\_Data\_status

{

status Data;//保存的数据

status positive\_text;//正文字

status negative\_text;//负文字

}\*Record\_Data\_status\_list;

1. CNF解析模块，使用的函数，完整代码详见附录；

读入文件：

status ReadFile(char \*filename,MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);

保存文件：

status SaveFile(char \*filename, MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf, status result, double time);

1. 核心DPLL模块，使用的函数；
2. status creatClaus(MyList \*NewList, status ID, status key, Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

根据传入的key的值，选择将新增的结点插入位置，每次新增结点使用malloc；

1. status Destory(MyList \*NewList,Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

如图 4-1-1所示；

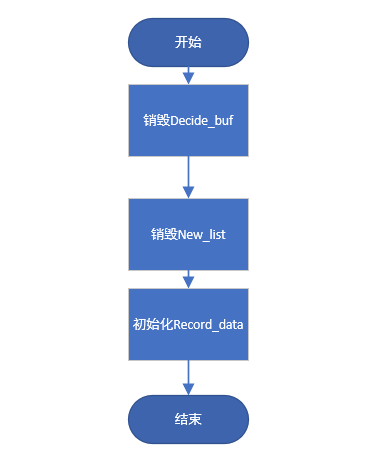


图 4-1-1 销毁邻接表

1. status evalute\_All\_Cluase(MyList \*NewList);//判断CNF文件字句的真假状态

即判断NewList所有字句的标记是否全部为1；

1. status variable\_decided();//从子句集中选择一个没有赋值的变量为其赋值，赋值成功//返回 1 ；若所有变量均已被赋值，则说明问题可满足
2. 根据NewList中Number\_of\_assigned\_words的值是否为1并且该字句状态为-1；

//返回 1 ；若所有变量均已被赋值，则说明问题可满足

1. status bcp\_deduce(MyList \*NewList, status j\_index, status p\_n\_text, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);//执行BCP过程

如图 4-1-2所示

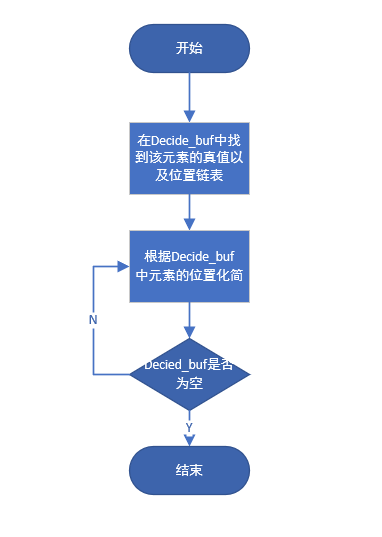


图 4-1-2 bcp\_deduce

1. status DPLL(MyList \*NewList, int index\_j, status temp\_p\_n\_text, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);//DPLL主要模块
2. void BackTrack\_Initilization(MyList \*NewList, status j\_index, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);//真值恢复函数

如图 4-1-3所示；

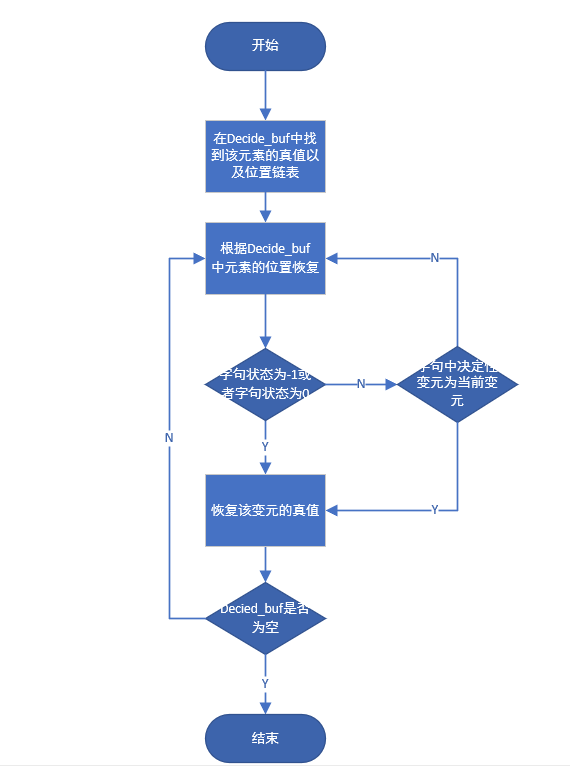


图 4-1-3 恢复函数

各函数之前的关系如图 4-1-4所示。



图 4-1-4 DPLL函数之间的关系

1. 数独模块,包括数独生成、归约、求解(Sudoku)，使用的函数；
2. void Tranlate(char \*filename, MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);//转换函数

其转换过程如图 4-1-5所示；

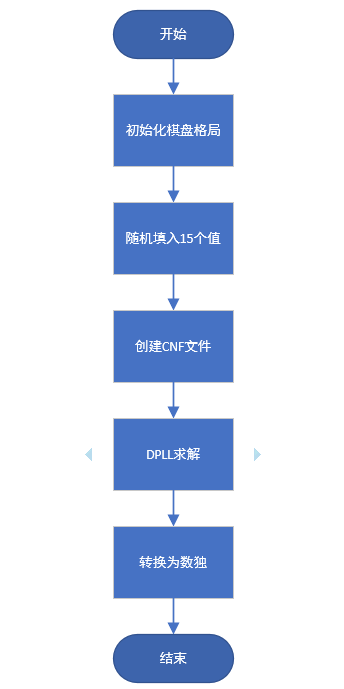


图 4-1-5 转换示意图

1. status creat\_sudoku\_file(char \*filename, status Sudoku\_buf[][9], status find\_word\_in\_sudoku);//生成CNF文件

生成CNF文件示意图，如图 4-1-6所示；

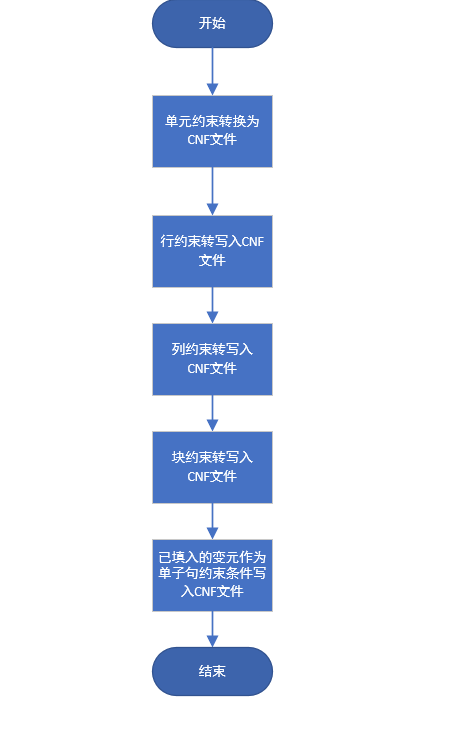


图 4-1-6生成CNF文件

1. status Translate\_to\_sudoku(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data\*Record\_Data, status find\_word\_in\_sudoku); //DPLL求解完成后转换为数独

该模块个函数之间的关系如图 4-1-5所示；



图4-2 数独求解函数之间的关系

**4.2 系统测试**

系统与用户交互界面如图 4-1所示。

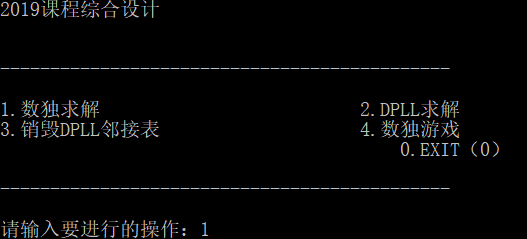


图 4-1 用户交互界面

1. CNF解析模块；
2. 模块说明；

该模块为本课程设计的核心模块，其是在DPLL框架下展开的；

1. 测试方法；

读取cnf算例文件，解析文件，基于一定的林基表的结构，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。

1. 运行结果；

测试用例1：如图4-2-1 基准满足算例sat-20.cnf。

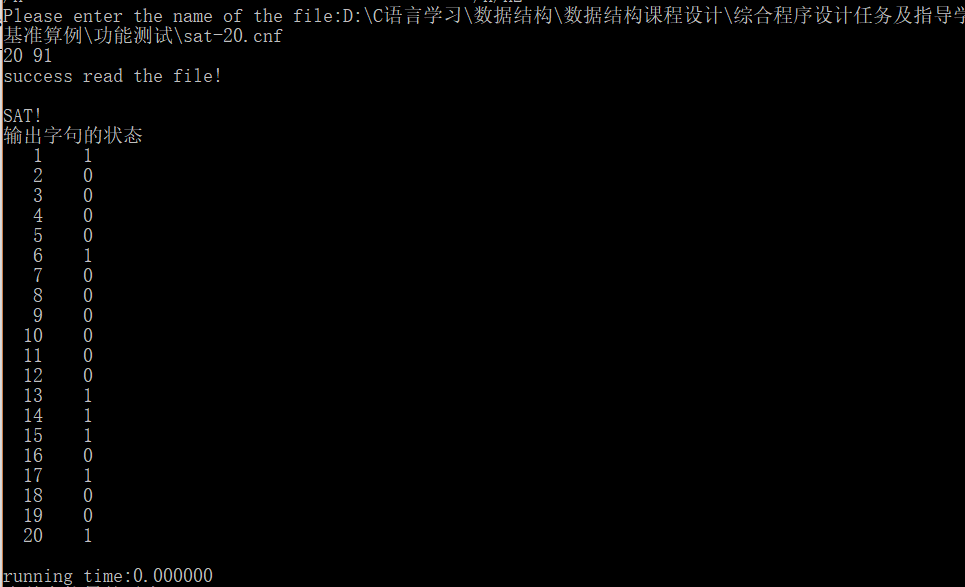


图 4-2-1基准满足算例sat-20.cnf

与预期结果一致；

测试用例二：如图4-2-2小型基准满足算例problem1-20.cnf所示。

运行截图：

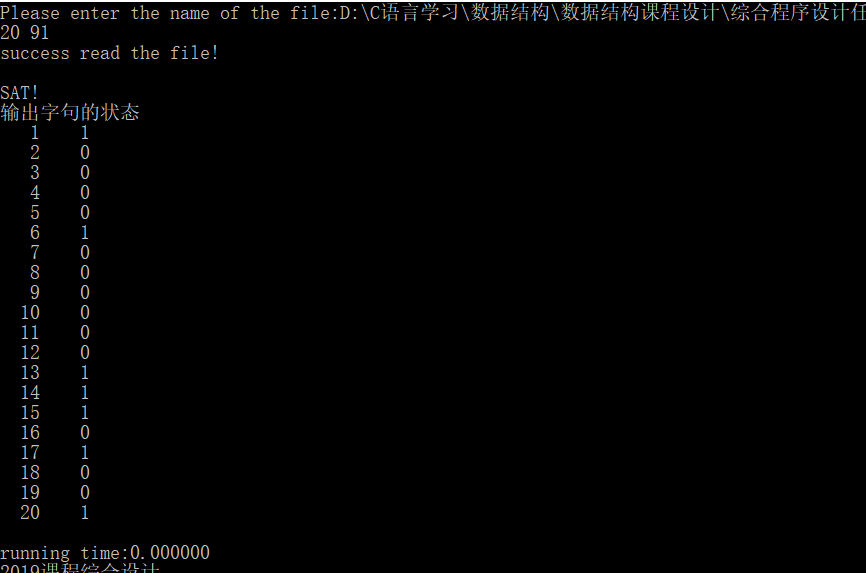
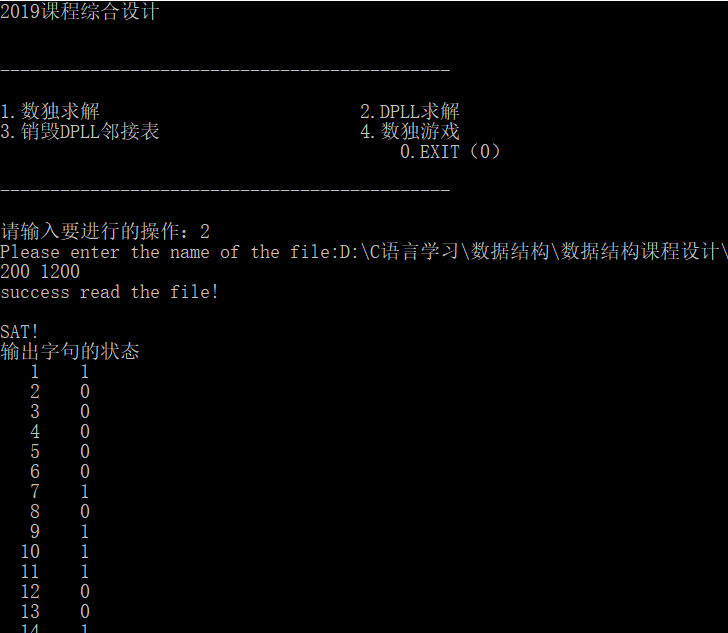


图4-2-2小型基准满足算例problem1-20.cnf

1. 运行结果分析：与预期一致；

测试算例3：中型基准满足算例problem12-200.cnf

运行截图：



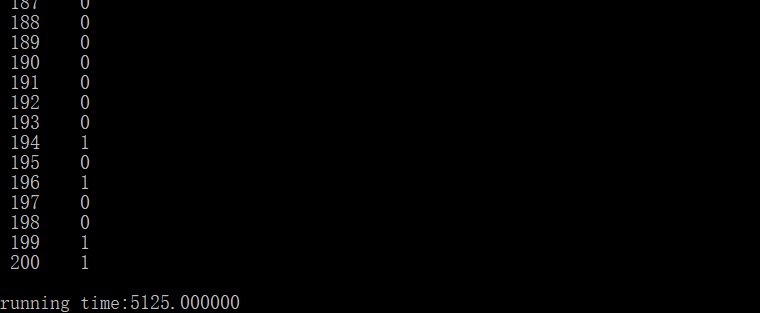


图 4-2-3 中型基准满足算例problem12-200.cnf

运行分析：与预期结果一致；

1. 数独模块,包括数独生成、归约、求解(Sudoku)模块
2. 模块说明：数独终盘的生成；主要包括生成9\*9的数独，第一步，随机生成15个数填入数独终盘，其中每生成一个数，都要根据数独的规则都要判断其填入的合法性；第二步，将所有的约束条件转换为CNF写入文件，并将已经填入的数字作为单子句也写入CNF文件；第三步，调用DPLL算法，求解CNF文件并转换为数独游戏生成数独终盘。
3. 测试方法：先随机生成棋盘，接着用DPLL算法生成数独终盘，最后用挖洞发生成一个数独游戏；
4. 运行结果，截图如图4-2-1 生成随机数独棋盘所示。

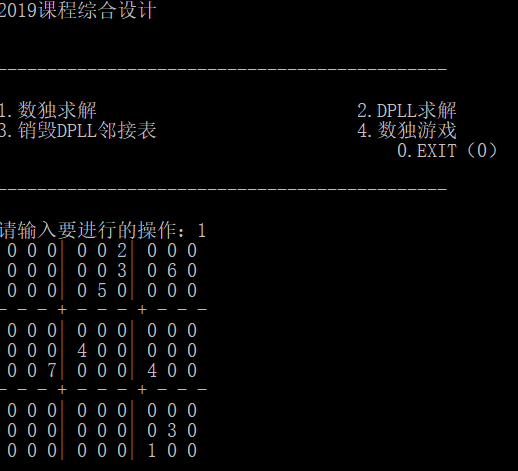


图 4-2-1 生成随机数独棋盘

如图 4-2-2生成数独终盘所示；

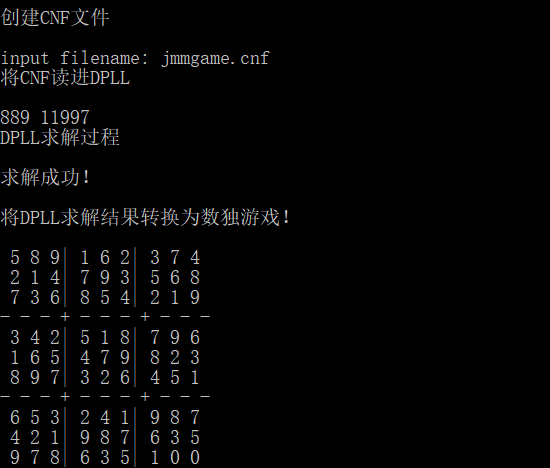


图 4-2-2生成数独终盘

**4.3程序优化**

1. 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。
2. 主要优化方向，包括代码指令减少，算法改进；

算法改进主要是决策方式的改变，即将最短正字句改变为变元在字句中出现的频率；

优化前后运行时间以及优化率如表4-3所示。

表 4-3 优化前后运行时间以及优化率



1. 优化分析，在（2）中可以观察到，优化之后某些算例反而出现负优化，究极原因，在分析算例后发现。对于变现出现频率相等或者说相差不大的算例，以变量出现频率作为决策方式，并不能达到预期结果。即优化还是需要针对实际算例的优化。

**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）利用邻接表实现了将CNF文件转换为可以求解的数据结构；

（2）基于邻接表的数据结构实现了最基础的SAT问题求解；

（3）将SAT问题求解应用于数独游戏的求解，实现数独游戏的生成。

**5.1 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. 学习中尽量多运用数据结构中所学的内容，在遇到一些问题时，多想想应该是使用怎么样的数据结构去存储和管理它们；
2. 多实践，将数据结构带到即将要学的专业课程中去，不断巩固提升所学内容；
3. 平时多看一些算法，及它们的应用，在学习别人的算法时提升自己的编程能力。

# 6体会

这次数据结构对我来说，感觉还是很有挑战性，从刚开始的文件读取，文件保存，通过请教老师和温习谭浩强的C语言与程序设计文件读取部分，在假期把这一部分完成。

后面慢慢的学会把程序写成一个个函数之后感觉是豁然开朗，感觉思路清晰了许多。在写到核心模块DPLL算法时，由于对文献变量决策阅读理解的错误，选取的静态变量出现频率的决策方式，导致一直卡在变量决策部分写不下去，在调试过程中发现，静态的变量决策根本不可能得到正确结果。在老师的耐心指导下，把决策方式修正为比较容易操作的最短正字句决策，有用几节课的时间调修改后出现的BUG。

修改之后，通过不断的调试，一直在变量回溯这一步出现问题，通过一遍一遍的修改逻辑，一步步的打印运行结果，一遍遍的调试，过程真的感觉十分艰难，当身边同学一个都完成时，自己颇为焦虑。

在DPLL写好之后，后面的数独求解，刚开始感觉到比较难的是把数度问题转换为CNF文件，十分感谢负责的老师，总是能够及时的发现我们的问题，在群里上传转换的文件，通过自己细细研读和同学的讨论，理解清楚思路后便开始着手写。用了半周的时间艰难的搞定。

这个课设总算是踉踉跄跄的画上一个句号，但真的是获益匪浅。当时着手操作时感觉好难好难，今天在这里写报告总结，不会再感到那么的难，不仅仅完成一次学习任务，对程序的设计方法，和刚学完的数据结构，都有了比较深入的认识和理解。最后，非常感谢负责的祝老师和热心的同学们，在成为一个合格程序员的路上有你们在，会更加精彩！

**附录**

源代码

1. 主程序

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <memory.h>

#include <algorithm>

#include "sat.h"

#include "MyFile.h"

#include "suduko.h"

#define ERROR 0

#define OK 1

#define max 9

#define min 1

int main()

{

status fun;

char filename[500];

char \*pfile = NULL;

status result = 0;

float time = 0;

int index\_j = 1;

int DPLL\_solve = 0;

clock\_t start, end;

Association\_Data \*Record\_Data= NULL;

MyList \*NewList = NULL;

Clause \*p = NULL;

Decide\_buffer \*Decide\_buf = NULL;

Decide\_buf = (Decide\_buffer\*)malloc(sizeof(Decide\_buffer)\*MAXLIST);

memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);

NewList = (struct MyList\*)malloc(sizeof(MyList));

NewList->num = 0;

NewList->numLine = 0;

memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);

Record\_Data = (Association\_Data\*)malloc(sizeof(Association\_Data)\*MAXLIST);

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

status temp\_p\_n\_text;

status Sudoku\_buf[9][9];

status Match\_buf[10][10][10];

int op = 1;

while (op)

{

//system("cls");

printf("2019课程综合设计\n\n");

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

printf("1.数独求解 2.DPLL求解 \n");

printf("3.销毁DPLL邻接表 4.数独游戏 \n");

printf(" 0.EXIT（0） \n");

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

printf("请输入要进行的操作：");

scanf("%d", &op);

switch (op)

{

case 1:

{

Record\_Data = NULL;

NewList = NULL;

p = NULL;

Decide\_buf = NULL;

Decide\_buf = (Decide\_buffer\*)malloc(sizeof(Decide\_buffer)\*MAXLIST);

memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);

NewList = (struct MyList\*)malloc(sizeof(MyList));

NewList->num = 0;

NewList->numLine = 0;

memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);

Record\_Data = (Association\_Data\*)malloc(sizeof(Association\_Data)\*MAXLIST);

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

Tranlate(filename, NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data);

}

break;

case 2:

{

Record\_Data = NULL;

NewList = NULL;

p = NULL;

Decide\_buf = NULL;

Decide\_buf = (Decide\_buffer\*)malloc(sizeof(Decide\_buffer)\*MAXLIST);

memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);

NewList = (struct MyList\*)malloc(sizeof(MyList));

NewList->num = 0;

NewList->numLine = 0;

memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);

Record\_Data = (Association\_Data\*)malloc(sizeof(Association\_Data)\*MAXLIST);

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

printf("Please enter the name of the file:");

scanf("%s", filename);

fun = ReadFile(filename,NewList, Record\_Data, Decide\_buf);

if (fun == ERROR)

printf("can't open the file!\n");

else if (fun == OK)

{

printf("success read the file!\n");

}

else if (fun == -1)

printf("Array space application failed!\n");

printf("\n");

temp\_p\_n\_text = 0;

start = clock();

result = DPLL(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);

end = clock();

time = (float)(end - start) \* 1000 / CLOCKS\_PER\_SEC;

if (result == OK)

printf("SAT!\n");

else

{

printf("NOT SAT!\n");

}

SaveFile(filename, NewList, Record\_Data, Decide\_buf, result, time);

printf("输出字句的状态\n");

for (int i = 1; i < NewList->num + 1; i++)

printf("%4d %4d\n", Decide\_buf[i].Data, Decide\_buf[i].Data\_status);

printf("\n");

printf("running time:%f\n", time);

//print(NewList, Record\_Data, Decide\_buf,1);

//print(NewList, Record\_Data, Decide\_buf, index\_j);

//free(Decide\_buf);

//free(NewList);

};

break;

case 3:

Destory(NewList, Decide\_buf, Record\_Data);

break;

case 4:

{

Record\_Data = NULL;

NewList = NULL;

p = NULL;

Decide\_buf = NULL;

Decide\_buf = (Decide\_buffer\*)malloc(sizeof(Decide\_buffer)\*MAXLIST);

memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);

NewList = (struct MyList\*)malloc(sizeof(MyList));

NewList->num = 0;

NewList->numLine = 0;

memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);

Record\_Data = (Association\_Data\*)malloc(sizeof(Association\_Data)\*MAXLIST);

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

}

sudoku\_game(NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data);

break;

case 0:

exit(0);

break;

default:

break;

}

}

return 0;

}

1. **文件操作**

status ReadFile(char \*filename,MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);

status SaveFile(char \*filename, MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf, status result, double time);

#include <algorithm>

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include "MyFile.h"

#include "sat.h"

#include "quicksort.h"

#define FLEN 100

int Num = 0;

int NumLine = 0;

status ReadFile(char \*filename, MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf)

{

FILE \*fp = NULL;

char ch;

status key = 0;

char tempFp[FLEN + 1];

if ((fp = fopen(filename, "r")) == NULL)

return ERROR;

ch = fgetc(fp);

while (ch != 'p')

{

fgets(tempFp, FLEN, fp);

ch = fgetc(fp);

}

//printf("%c\n", ch);

fgets(tempFp, (sizeof(int) / 4)\*5, fp);

fscanf(fp, "%d %d", &Num, &NumLine);

printf("%d %d\n", Num, NumLine);

NewList->num = Num; NewList->numLine = NumLine;

for (int i = 0; i < NewList->numLine; i++)

{

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

creatClaus(NewList, i, key, Decide\_buf,Record\_Data);

//printf("%d ", key);

while (key!=0)

{

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

if (key != 0)

creatClaus(NewList, i, key, Decide\_buf, Record\_Data);

}

}

//对计数器的变量数量做排序

//quickSort(Record\_Data,1, NewList->num);

fclose(fp);

return OK;

}

status SaveFile(char \*filename,MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf,status result,double time)

{

FILE \*fp = NULL;

char path[30] = ".res";

char file[100];

strncat(filename, path, 50);

//char filepath[30] = "D:\\SAT测试备选算例\\Result";

//sprintf(file, "%s/%s", filepath, filename);

if ((fp = fopen(filename, "w")) == NULL)

return ERROR;

if (result == OK)

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

else if (result == ERROR)

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

else

fprintf(fp, "%s %d\n", "s", -1);

fprintf(fp, "%s ", "v");

for (int i = 1; i < NewList->num + 1; i++)

{

if (Decide\_buf[i].Data\_status == 1)

fprintf(fp, "%d ", Decide\_buf[i].Data);

else if (Decide\_buf[i].Data\_status == 0)

fprintf(fp, "%d ", -Decide\_buf[i].Data);

else

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

}

fprintf(fp,"\n");

fprintf(fp, "%s %lf\n", "t", time);

fclose(fp);

}

1. **SAT求解，DPLL算法核心框架**

#pragma once

typedef int status;

#define ERROR 0

#define OK 1

#define NumClaus 20000

#define MAXLIST 20000

#define INITIALIZATION 9999

typedef struct Record\_Data\_status

{

status Data;//保存的数据

status positive\_text;//正文字

status negative\_text;//负文字

}\*Record\_Data\_status\_list;

typedef struct Data\_status {

status Data;

status Data\_situation;

}Data\_statusp[MAXLIST];

typedef struct Clause {

Data\_status elem;

struct Clause \*next;

}Clause, \*ClauseList;

//子句结点类型定义

typedef struct LnodeClause {

Clause \*firstnode;

int num\_positive\_text;//正文字个数

int num\_negative\_text;//负文字个数

int ClauseNum;//记录该字句的单子句数目

int Number\_of\_assigned\_words;//未赋值字句数量

int Determine;//判断该字句的真假状态 0 为假 ，1 为真，-1 为不确定

}LnodeClause, LnodeClausep[MAXLIST];

typedef struct MyList {

LnodeClausep vertices;

int num, numLine;

}MyList;

typedef struct Association\_Clause {

status num;

Association\_Clause \*next;

};

typedef struct Decide\_buffer {//文字计数器

status Data;

status Data\_status;

int num\_positive\_text;//正文字个数

int num\_negative\_text;//负文字个数

int total\_number;//正负文字总数

Clause \*firstnode;

};

typedef struct Association\_Data {

status Data;

status mark\_num;

int totalNum;

};

status creatClaus(MyList \*NewList, status ID, status key, Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

status destoryClause(MyList \*NewList, status num);//销毁单子句

status DestoryDecide\_buf(Decide\_buffer \*Decide\_buf, int i);

status Destory(MyList \*NewList,Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

status isUnitClause(MyList \*NewList);//判断是否为单子句\*/

status evaluateClause(MyList \*NewList,status id);//评估字句的真假状态

status evalute\_All\_Cluase(MyList \*NewList);//判断CNF文件字句的真假状态

//status variable\_decided();//从子句集中选择一个没有赋值的变量为其赋值，赋值成功

//返回 1 ；若所有变量均已被赋值，则说明问题可满足

status bcp\_deduce(MyList \*NewList, status j\_index, status p\_n\_text, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);//执行BCP过程

status Conflict(MyList \*NewList);

status DPLL(MyList \*NewList, int index\_j, status temp\_p\_n\_text, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);

void BackTrack\_Initilization(MyList \*NewList, status j\_index, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);

status Check\_repet\_ID(Decide\_buffer \*Decide\_buf, status key, status ID);

void print2(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf);

void print3(MyList \*NewList, status Data, Decide\_buffer \*Decide\_buf);

void print(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf, status index);

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include "stack.h"

#include "sat.h"

status Check\_repet\_ID(Decide\_buffer \*Decide\_buf, status key,status ID)

{//检查关联链表，防止记录一个文字多次

Clause \*p = NULL;

if (Decide\_buf[key].firstnode == NULL)

return OK;

else

p = Decide\_buf[key].firstnode;

while (p!=NULL)

{

if (p->elem.Data == ID)

return ERROR;

p = p->next;

}

return OK;

}

status creatClaus(MyList \*NewList, status ID, status key, Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)

{

Clause \*p = NULL;

NewList->vertices[ID].Determine = -1;

p = (struct Clause\*)malloc(sizeof(Clause));

p->elem.Data = key;

p->elem.Data\_situation = -1;

p->next = NULL;

Clause \*q\_Record = NULL;

q\_Record = (Clause\*)malloc(sizeof(Clause));

q\_Record->elem.Data = ID;

q\_Record->next = NULL;

if (key > 0)

{

if (Decide\_buf[key].Data == 0)

Decide\_buf[key].Data = key;

Decide\_buf[key].num\_positive\_text++;

NewList->vertices[ID].num\_positive\_text++;

Decide\_buf[key].total\_number++;

Decide\_buf[key].Data\_status = -1;

if (Check\_repet\_ID(Decide\_buf, key, ID) == OK)

{//记录文字出现的位置

if (Decide\_buf[key].firstnode == NULL)

Decide\_buf[key].firstnode = q\_Record;

else

{

q\_Record->next = Decide\_buf[key].firstnode;

Decide\_buf[key].firstnode = q\_Record;

}

}

}

else if (key < 0)

{

if (Decide\_buf[-key].Data == 0)

Decide\_buf[-key].Data = -key;

Decide\_buf[-key].num\_negative\_text++;

Decide\_buf[-key].total\_number++;

NewList->vertices[ID].num\_negative\_text++;

Decide\_buf[-key].Data\_status = -1;

if (Check\_repet\_ID(Decide\_buf, -key, ID) == OK)

{

if (Decide\_buf[-key].firstnode == NULL)

{

Decide\_buf[-key].firstnode = q\_Record;

}

else

{

q\_Record->next = Decide\_buf[-key].firstnode;

Decide\_buf[-key].firstnode = q\_Record;

}

}

}

NewList->vertices[ID].ClauseNum++;

NewList->vertices[ID].Number\_of\_assigned\_words = NewList->vertices[ID].ClauseNum;

if (NewList->vertices[ID].firstnode == NULL)

NewList->vertices[ID].firstnode = p;

else

{

p->next = NewList->vertices[ID].firstnode;

NewList->vertices[ID].firstnode = p;

}

return OK;

}

status Destory(MyList \*NewList,Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)

{

for (int i = 0; i < NewList->numLine; i++)

{

destoryClause(NewList, i);

}

for (int i = 1; i < NewList->num + 1; i++)

{

DestoryDecide\_buf(Decide\_buf, i);

}

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);

memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);

memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);

//free(Decide\_buf);

//free(NewList);

return OK;

}

status DestoryDecide\_buf(Decide\_buffer \*Decide\_buf, int i)

{

Clause \*p = Decide\_buf[i].firstnode;

Clause \*q = p;

while (p)

{

q = p->next;

free(p);

p = q;

}

return OK;

}

status destoryClause(MyList \*NewList ,status num)

{

Clause \*p\_New = NewList->vertices[num].firstnode;

Clause \*q = p\_New;

while (p\_New)

{

q = p\_New->next;

free(p\_New);

p\_New = q;

}

NewList->vertices[num].firstnode = NULL;

return OK;

}

status isUnitClause(MyList \*NewList)//判断是否为单子句

{ //单子句来源于两部分，一是字句数量为1，二是赋值化简得到的单子句

//而赋值化简得到的单子句怎么搞

MyList \*p = NULL;

Clause \*q = NULL;

for (int i = 0; (i < NewList->numLine); i++)

{

if ((NewList->vertices[i].ClauseNum == 1) && NewList->vertices[i].Determine == -1)

return i;

else if (NewList->vertices[i].Number\_of\_assigned\_words == 1 && NewList->vertices[i].Determine == -1)

return i;

}

return -1;

}

status evaluateClause(MyList \*NewList,status id)

{//判断一条的字句的状态,同时给出该字句的三种状态，真(1)，假(0)，无法确定(-1)

//字句的真假状态赋值

Clause \*p = NULL;

int temp\_Record\_Num = NewList->vertices[id].ClauseNum;

//记录字句个数，如果循环做完，假字句个数没有改变，则该条字句状态为假

p = NewList->vertices[id].firstnode;

while (p)

{

if (p->elem.Data\_situation == 1)

{

NewList->vertices[id].Determine = 1;

return OK;

}

else if (p->elem.Data\_situation == 0)

--temp\_Record\_Num;

p = p->next;

}

if (temp\_Record\_Num == 0)

{

NewList->vertices[id].Determine = 0;

return ERROR;

}

else if (temp\_Record\_Num != 0)

{

NewList->vertices[id].Determine = -1;

return OK;

}

return OK;

}

status evalute\_All\_Cluase(MyList \*NewList)

{

MyList \*p = NULL;

for (int i = 0; i < NewList->numLine; i++)

{

if (NewList->vertices[i].Determine == 0 || NewList->vertices[i].Determine == -1)

return ERROR;

}

return OK;

}

status bcp\_deduce(MyList \*NewList, status j\_index,status p\_n\_text,Association\_Data \*Record\_Data,Decide\_buffer \*Decide\_buf)

{ //执行BCP推导过程，利用化简策略对字句集化简

//BCP就是反复应用单元字句规则对合取范式的子句集进行化简，直到子句集中不再存在

//单元字句的过程单元字句规则： 对于子句w = (x1 || -x2 || x3 || -x4）

//当在x1 = 0，x2 = 1，x3 = 0的条件下，该子句就变为单元子句，很显然，

//为使该子句可满足，子句w中的文字-x4必需取值为 0，

status temp\_p\_n\_text = 0;

status temp\_Data = 0;

status Data = 0;

int flag = 1;

status fun = -1;

Data = Record\_Data[j\_index].Data;

Decide\_buf[Data].Data\_status = p\_n\_text;

Clause \*p\_NewList = NULL;

Clause \*p\_Association = NULL;

p\_Association = Decide\_buf[Record\_Data[j\_index].Data].firstnode;

while (p\_Association)

{

if (NewList->vertices[p\_Association->elem.Data].Determine == -1)

{//对字句当前的状态有判断

p\_NewList = NewList->vertices[p\_Association->elem.Data].firstnode;

flag = 1;

while (p\_NewList)

{

if (p\_NewList->elem.Data == Data || p\_NewList->elem.Data == -Data)

{

if (p\_NewList->elem.Data > 0)

{

p\_NewList->elem.Data\_situation = Decide\_buf[Data].Data\_status;

NewList->vertices[p\_Association->elem.Data].num\_positive\_text--;

}

else

{

p\_NewList->elem.Data\_situation = (Decide\_buf[Data].Data\_status == 1 ? 0 : 1);

NewList->vertices[p\_Association->elem.Data].num\_negative\_text--;

}

NewList->vertices[p\_Association->elem.Data].Number\_of\_assigned\_words--;

}

p\_NewList = p\_NewList->next;

}

evaluateClause(NewList, p\_Association->elem.Data);

}

p\_Association = p\_Association->next;

}

//system("cls");

//print3(NewList, Data, Decide\_buf);

return OK;

}

status Conflict(MyList \*NewList)

{//判断是否存在冲突

for (int i = 0; i < NewList->numLine; i++)

{

if (NewList->vertices[i].Determine == 0)

return ERROR;

}

return OK;

}

void print(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf, status index)

{

printf("当前在栈里的元素\n");

printf("序号 变元 真值 化简方式 \n");

for (int i = 1; i<index+2; i++)

printf("%5d %5d %5d %5d\n",i, Record\_Data[i].Data, Decide\_buf[Record\_Data[i].Data].Data\_status,Record\_Data[i].mark\_num);

printf("\n\n");

printf("当前所有 字句 的真假状态\n");

for (int i = 0; i < NewList->numLine; i++)

{

Clause \*p = NewList->vertices[i].firstnode;

printf("%-5d %-5d\n", i, NewList->vertices[i].Determine);

}

printf("\n\n");

printf("当前CNF文件中变元的的真值情况\n");

for (int i = 0; i < NewList->numLine; i++)

{

printf("%-2d %-2d ", i,NewList->vertices[i].Determine);

Clause \*p = NewList->vertices[i].firstnode;

while (p)

{

printf("%-2d ", p->elem.Data\_situation);

p = p->next;

}

printf("\n");

}

printf("\n\n");

/\*printf("当前元素所在位置:%d\n", Record\_Data[index].Data);

/\*Clause \*p\_Ass = Decide\_buf[Record\_Data[index].Data].firstnode;

while (p\_Ass)

{

printf("%d ", p\_Ass->elem.Data);

p\_Ass = p\_Ass->next;

}\*/

}

status variable\_decide(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf)

{//最短正字句规则,返回当前最短的正字句序号

status MaxNum = -1;

status Cluase\_Num = 100;

status ID = -1;

for (int i = 0; i < NewList->numLine; i++)

{

if (NewList->vertices[i].num\_positive\_text == NewList->vertices[i].Number\_of\_assigned\_words && NewList->vertices[i].ClauseNum <= Cluase\_Num)

{//在每次回溯之后，字句长度都会变短

if (NewList->vertices[i].Determine == -1)

{

Cluase\_Num = NewList->vertices[i].ClauseNum;

ID = i;

}

}

}

return ID;

}

void print2(MyList \*NewList, Association\_Data \*Record\_Data, Decide\_buffer \*Decide\_buf)

{//输出当前所有变元的赋值情况

printf("变元当前的真值：\n");

for (int i = 1; i < NewList->num + 1; i++)

printf("%2d %2d\n", i, Decide\_buf[i].Data\_status);

}

status DPLL(MyList \*NewList,int index\_j, status temp\_p\_n\_text,Association\_Data \*Record\_Data,Decide\_buffer \*Decide\_buf)

{

status fun = 0;

int flag\_while = 1;

int flag = 1;

int j\_index = 0;

status fun\_single\_flag = -1;

status temp\_Data = -1;

status back\_flag = 0;

status isUnitClause\_flag = 0;

status bcp\_flag = 1;

status postitive\_num = 0;

status negative\_num = 0;

status Decide\_p\_n\_text = 0;

status fun\_Decide = -1;

status fun\_conflict\_one = -1;

status fun\_conflict\_two = -1;

/\*for (int i = 1; i < NewList->num+1; i++)

{

postitive\_num += Decide\_buf[i].num\_positive\_text;

negative\_num += Decide\_buf[i].num\_negative\_text;

}

if (postitive\_num > negative\_num)

Decide\_p\_n\_text = 1;

else\*/

Decide\_p\_n\_text = 0;

//第一次给最短正字句赋值，并入栈

fun\_Decide = variable\_decide(NewList, Record\_Data, Decide\_buf);

if (fun\_Decide == -1)

{//刚开始没有满足条件的最短正字句

if (Decide\_buf[1].Data\_status == -1)

Record\_Data[1].Data = Decide\_buf[1].Data;

}

else {//有满足条件的正字句

Clause \*p = NewList->vertices[fun\_Decide].firstnode;

while (p)

{

if (p->elem.Data\_situation == -1)

{

Record\_Data[1].Data = p->elem.Data;

break;

}

p = p->next;

}

}

bcp\_deduce(NewList, index\_j, Decide\_p\_n\_text, Record\_Data, Decide\_buf);

while (flag\_while == 1)

{

status back\_flag = 1;

fun\_single\_flag = isUnitClause(NewList);

if (fun\_conflict\_two != ERROR)

{//这里是为了防止翻转后有矛盾继续执行

while (fun\_single\_flag != -1 && back\_flag == 1)

{//当前单子句没有化简完的时候,循环化简当前单子句

status temp\_Data = 0;

Clause \*p = NewList->vertices[fun\_single\_flag].firstnode;

while (p)

{//找到单子句

if (p->elem.Data\_situation == -1)

{

temp\_Data = p->elem.Data;

break;

}

p = p->next;

}

while (Record\_Data[index\_j].Data != -1)

index\_j++;

if (Record\_Data[index\_j].Data == -1)

{//把单子句文字入栈

if (temp\_Data > 0)

{

Record\_Data[index\_j].Data = temp\_Data;

Record\_Data[index\_j].mark\_num = 0;

Decide\_buf[Record\_Data[index\_j].Data].Data\_status = 1;

}

else if (temp\_Data < 0)

{

Record\_Data[index\_j].Data = -temp\_Data;

Record\_Data[index\_j].mark\_num = 0;

Decide\_buf[Record\_Data[index\_j].Data].Data\_status = 0;

}

bcp\_deduce(NewList, index\_j, Decide\_buf[Record\_Data[index\_j].Data].Data\_status, Record\_Data, Decide\_buf);

}

fun\_conflict\_one = Conflict(NewList);//是否有矛盾

if (fun\_conflict\_one == ERROR)

{

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

--index\_j;

if (Record\_Data[index\_j].mark\_num == -1)

back\_flag = 0;//遇到最短正字句规则化简的文字，跳出循环

else

{//全部恢复由于单子句规则化简的文字

while (Record\_Data[index\_j].mark\_num != -1)

{

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

--index\_j;

back\_flag = 0;

}

}

}

fun\_single\_flag = isUnitClause(NewList);

}

}

//system("cls");

//print(NewList, Record\_Data, Decide\_buf, index\_j);

fun = evalute\_All\_Cluase(NewList);

if (fun == OK)

{

flag\_while = 0;

return OK;

}

if (back\_flag == 1 && fun\_conflict\_one != ERROR)

{//上一次单子句化简没有矛盾，新元素继续入栈

while (Record\_Data[index\_j].Data != -1)

index\_j++;

if (Record\_Data[index\_j].Data == -1)

{

fun\_Decide = variable\_decide(NewList, Record\_Data, Decide\_buf);//寻找当前最短正字句

if (fun\_Decide != -1)

{//有满足条件的最短正字句，该正字句的首个变元入栈

Clause \*p\_temp = NewList->vertices[fun\_Decide].firstnode;

while (p\_temp)

{

if (p\_temp->elem.Data\_situation == -1)

{//因为是最短正字句，所以，元素一定是正值

Record\_Data[index\_j].Data = p\_temp->elem.Data;

temp\_p\_n\_text = Decide\_p\_n\_text;

break;

}

p\_temp = p\_temp->next;

}

}

else

{//没有满足条件的正字句,就按照变元顺序选择一个没有赋值的变元

for (int i = 1; i < NewList->num + 1; i++)

{

if (Decide\_buf[i].Data\_status == -1)

{

Record\_Data[index\_j].Data = Decide\_buf[i].Data;

temp\_p\_n\_text = Decide\_p\_n\_text;

break;

}

}

}

//system("cls");

//print2(NewList, index\_j, temp\_p\_n\_text,Record\_Data, Decide\_buf);

//print(NewList, Record\_Data, Decide\_buf, j\_index);

bcp\_deduce(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);

}

}

if (fun\_conflict\_two != ERROR)//还是为了防止在翻转后没法判断是不是有矛盾

{

fun\_conflict\_one = Conflict(NewList);

}

if ((fun\_conflict\_one != ERROR) && (index\_j < NewList->num + 1))

{//没有冲突

fun = evalute\_All\_Cluase(NewList);

if (fun == OK)

{

flag\_while = 0;

return OK;

}

}

if (fun\_conflict\_one == ERROR || back\_flag == 0)

{//发生冲突 发生冲突的值是在最短正字句赋值过程中产生的，不是单子句化简产生

if (Decide\_buf[Record\_Data[index\_j].Data].Data\_status == Decide\_p\_n\_text)

{//就近翻转, 还停留在index\_j这一层，所以不需要考虑其他情形

temp\_Data = Record\_Data[index\_j].Data;

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

Record\_Data[index\_j].Data = temp\_Data;

temp\_p\_n\_text = (Decide\_p\_n\_text == 1 ? 0 : 1);

bcp\_deduce(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);

}

else if (Decide\_buf[Record\_Data[index\_j].Data].Data\_status == (Decide\_p\_n\_text == 0 ? 1 : 0))

{//从右边回溯上来

if (index\_j == 1 && Decide\_buf[Record\_Data[index\_j].Data].Data\_status == (Decide\_p\_n\_text == 0 ? 1 : 0))

{//从右子树回溯到根，说明整个二叉树已经遍历完成

flag\_while = 0;

return ERROR;

}

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

--index\_j;

if (index\_j == 1 && Decide\_buf[Record\_Data[index\_j].Data].Data\_status == (Decide\_p\_n\_text == 0 ? 1 : 0))

{//从右子树回溯到根，说明整个二叉树已经遍历完成

flag\_while = 0;

return ERROR;

}

while (((Record\_Data[index\_j].mark\_num == 0)&& index\_j > 1)||(Decide\_buf[Record\_Data[index\_j].Data].Data\_status == 1)&& index\_j > 1)

{//当栈里面的值赋值是 1 或者是单子句化简得到的值（做了标记，即mark\_num的值是 0），都需要恢复

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

--index\_j;

if (Decide\_buf[Record\_Data[index\_j].Data].Data\_status == 0 && Record\_Data[index\_j].mark\_num == -1)

break;

}

if (index\_j == 1 && Decide\_buf[Record\_Data[index\_j].Data].Data\_status == (Decide\_p\_n\_text == 0 ? 1 : 0))

{//从右子树回溯到根，说明整个二叉树已经遍历完成

flag\_while = 0;

return ERROR;

}

temp\_Data = Record\_Data[index\_j].Data;

BackTrack\_Initilization(NewList, index\_j, Record\_Data, Decide\_buf);

Record\_Data[index\_j].Data = temp\_Data;

temp\_p\_n\_text = (Decide\_p\_n\_text == 0 ? 1 : 0);

bcp\_deduce(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);

}

}

//再次判断是不是有矛盾

fun\_conflict\_two = Conflict(NewList);

//system("cls");

//print(NewList, Record\_Data, Decide\_buf, index\_j);

}

}

void BackTrack\_Initilization(MyList \*NewList, status j\_index,Association\_Data \*Record\_Data,Decide\_buffer \*Decide\_buf)

{//回溯过程中恢复特定变量的真值

Clause \*p\_NewList = NULL;

Clause \*p\_Association = NULL;

status Data = Record\_Data[j\_index].Data;

status temp\_Data = 0;

status Data\_flag = 1;

//system("cls");

//print(NewList, Record\_Data, Decide\_buf, j\_index);

p\_Association = Decide\_buf[Record\_Data[j\_index].Data].firstnode;

Record\_Data[j\_index].Data = -1;

Record\_Data[j\_index].mark\_num = -1;

Decide\_buf[Data].Data\_status = -1;

while (p\_Association)

{

Data\_flag = 1;

if (NewList->vertices[p\_Association->elem.Data].Determine == -1)

{//字句真假状态没有确定

p\_NewList = NewList->vertices[p\_Association->elem.Data].firstnode;

while (p\_NewList)

{

if (p\_NewList->elem.Data == Data || p\_NewList->elem.Data == -Data)

{

p\_NewList->elem.Data\_situation = -1;

if (p\_NewList->elem.Data > 0)

++NewList->vertices[p\_Association->elem.Data].num\_positive\_text;

else

++NewList->vertices[p\_Association->elem.Data].num\_negative\_text;

++NewList->vertices[p\_Association->elem.Data].Number\_of\_assigned\_words;

}

p\_NewList = p\_NewList->next;

}

}

else if (NewList->vertices[p\_Association->elem.Data].Determine == 0)

{// 只能是 最后一个文字的赋值导致该字句为假，只恢复该文字的状态，再判断字句的真假状态

p\_NewList = NewList->vertices[p\_Association->elem.Data].firstnode;

while (p\_NewList)

{

if (p\_NewList->elem.Data == Data || p\_NewList->elem.Data == -Data)

{

p\_NewList->elem.Data\_situation = -1;

if (p\_NewList->elem.Data > 0)

++NewList->vertices[p\_Association->elem.Data].num\_positive\_text;

else

++NewList->vertices[p\_Association->elem.Data].num\_negative\_text;

++NewList->vertices[p\_Association->elem.Data].Number\_of\_assigned\_words;

}

p\_NewList = p\_NewList->next;

}

evaluateClause(NewList, p\_Association->elem.Data);

}

else if (NewList->vertices[p\_Association->elem.Data].Determine == 1)

{//

status temp\_flag = 1;

Data\_flag = 1;

p\_NewList = NewList->vertices[p\_Association->elem.Data].firstnode;

while (p\_NewList && temp\_flag == 1)

{

//判断是 当前变量 赋值使得该字句为真

//如果是前面变量赋值导致该字句为真，则后面的变量在该字句中的值一直是 -1

//如果不是该变量赋值导致该字句为真，则前面变量的值根本不需要做变动

if (p\_NewList->elem.Data == Data || p\_NewList->elem.Data == -Data)

{

if (p\_NewList->elem.Data\_situation == 1)

{

Data\_flag = 0;

temp\_flag = 0;

}

}

if (Data\_flag == 1)

p\_NewList = p\_NewList->next;

}

if (Data\_flag == 0)

{//如果是由当前文字的赋值导致该字句为真

p\_NewList = NewList->vertices[p\_Association->elem.Data].firstnode;

while (p\_NewList)

{

if (p\_NewList->elem.Data == Data || p\_NewList->elem.Data == -Data)

{

p\_NewList->elem.Data\_situation = -1;

if (p\_NewList->elem.Data > 0)

++NewList->vertices[p\_Association->elem.Data].num\_positive\_text;

else

++NewList->vertices[p\_Association->elem.Data].num\_negative\_text;

++NewList->vertices[p\_Association->elem.Data].Number\_of\_assigned\_words;

}

p\_NewList = p\_NewList->next;

}

evaluateClause(NewList, p\_Association->elem.Data);

}

}

p\_Association = p\_Association->next;

}

//system("cls");

//print3(NewList, Data, Decide\_buf);

}

void print3(MyList \*NewList, status Data, Decide\_buffer \*Decide\_buf)

{

Clause \*p\_ass = Decide\_buf[Data].firstnode;

printf("当前文字的值: %2d ",Data);

printf("文字状态: %2d\n", Decide\_buf[Data].Data\_status);

while ( p\_ass)

{

printf("字句序号 ：%2d 文字所在字句的状态 ：%2d ",p\_ass->elem.Data,NewList->vertices[p\_ass->elem.Data].Determine );

Clause \*p\_N = NewList->vertices[p\_ass->elem.Data].firstnode;

while (p\_N)

{

if (p\_N->elem.Data == Data || p\_N->elem.Data == -Data)

{

printf("文字：%2d 真值：%2d\n", p\_N->elem.Data, p\_N->elem.Data\_situation);

}

p\_N = p\_N->next;

}

p\_ass = p\_ass->next;

}

}

1. **数独求解模块**

#pragma once

#include "sat.h"

#include "MyFile.h"

void Tranlate(char \*filename, MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

void Ini\_sudoku(status Sudoku\_buf[][9], status Match\_buf[][10][10]);

status Creat\_suduko(status Sudoku\_buf[][9]);

status Judge\_suduko(status Sudoku\_buf[][9], int row, int col, int value);

void Display(status Sudoku\_buf[][9]);

status Check\_Sudoku\_ID(MyList \*NewList);

void Creat\_CNF\_POSITION(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

void Creat\_CNF\_ROW(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

void Creat\_CNF\_COL(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

void Creat\_CNF\_BLOCK(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

status num\_position(int x, int y, int value);

status creat\_sudoku\_file(char \*filename, status Sudoku\_buf[][9], status find\_word\_in\_sudoku);

status Translate\_to\_sudoku(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data, status find\_word\_in\_sudoku);

void sudoku\_game(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

status Judge\_hole(char \*filename, MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data);

status Judge\_rule\_sudoku(status Sudoku\_buf[][9]);

status find\_word\_in\_sudoku\_fun(status Sudoku\_buf[][9]);

**#include <time.h>**

**#include<stdlib.h>**

**#include "suduko.h"**

**void Ini\_sudoku(status Sudoku\_buf[][9], status Match\_buf[][10][10])**

**{**

**for (int i = 0; i < 9; i++)**

**{**

**for (int j = 0; j < 9; j++)**

**{**

**Sudoku\_buf[i][j] = 0;**

**}**

**}**

**for (int i = 0; i <= 9; i++)**

**{**

**for (int j = 0; j <= 9; j++)**

**{**

**for (int k = 0; k <= 9; k++)**

**{**

**Match\_buf[i][j][k] = 0;**

**}**

**}**

**}**

**}**

**status Creat\_suduko(status Sudoku\_buf[][9])**

**{//随机生成15个数填进三位数组**

**status i = 0, j = 0, v = 0;**

**time\_t t = time(NULL);**

**srand(t);**

**int count = 0;**

**while (count < 11)**

**{**

**i = (rand() % 9);**

**j = (rand() % 9);**

**v = (rand() % 9) + 1;**

**if (Judge\_suduko(Sudoku\_buf, i, j, v) == OK)**

**{**

**Sudoku\_buf[i][j] = v;**

**count++;**

**}**

**}**

**return OK;**

**}**

**status Judge\_suduko(status Sudoku\_buf[][9],int row,int col,int value)**

**{**

**for (int i = 0; i < 9; i++)**

**{//判断行**

**if (Sudoku\_buf[row][i] == value)**

**return ERROR;**

**}**

**for (int j = 0; j < 9; j++)**

**{//判断列**

**if (Sudoku\_buf[j][col] == value)**

**return ERROR;**

**}**

**//判断块**

**int x = (row / 3) \* 3;**

**int y = (col / 3) \* 3;**

**for (int i = x; i < x + 3; i++)**

**{**

**for (int j = y; j < y + 3; j++)**

**if (Sudoku\_buf[i][j] == value)**

**return ERROR;**

**}**

**return OK;**

**}**

**void Display(status Sudoku\_buf[][9])**

**{**

**status flag = 0;**

**for (int i = 0; i < 9; i++)**

**{**

**if (i == 3 || i == 6)**

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

**for (int j = 0; j < 9; j++)**

**{**

**if (j == 3 || j == 6)**

**printf("|");**

**printf("%2d", Sudoku\_buf[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**void Match(status Sudoku\_buf[][9], status Match\_buf[][10][10])**

**{//匹配函数**

**for(int i=0;i<9;i++)**

**for (int j = 0; j < 9; j++)**

**{**

**if (Match\_buf[i][j] == 0)**

**{//如果该位置为空**

**for (int value = 1; value <= 9; value++)**

**{**

**if (Judge\_suduko(Sudoku\_buf, i, j, value) == OK)**

**{//可以填值的位置标志位 1**

**++Match\_buf[i + 1][j + 1][0];**

**Match\_buf[i + 1][j + 1][value] = 1;**

**}**

**}**

**}**

**}**

**}**

**status Check\_Sudoku\_ID(MyList \*NewList)**

**{//检查邻接表里面还没有生成字句的序号**

**int i = 0;**

**MyList \*p = NewList;**

**while (p)**

**{**

**if (p->vertices[i].firstnode == NULL)**

**return i;**

**i++;**

**}**

**}**

**status num\_position(int x,int y,int value)**

**{**

**return (x \* 100 + y \* 10 + value);**

**}**

**void Creat\_CNF\_POSITION(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{//创建每个位置的约束条件**

**int ID = 0;**

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

**{**

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

**{**

**ID = Check\_Sudoku\_ID(NewList);**

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

**{//一个位置生成 9 条字句**

**creatClaus(NewList, ID, num\_position(i, j, k), Decide\_buf, Record\_Data);**

**}**

**for (int m = 1; m <= 9; m++)**

**{//每个位置的字句两两组合**

**ID = Check\_Sudoku\_ID(NewList);**

**creatClaus(NewList, ID, -num\_position(i, j, m), Decide\_buf, Record\_Data);**

**for (int n = m; n <= 9; n++)**

**{**

**creatClaus(NewList, ID, -num\_position(i, j, n), Decide\_buf, Record\_Data);**

**ID = Check\_Sudoku\_ID(NewList);**

**creatClaus(NewList, ID, -num\_position(i, j, m), Decide\_buf, Record\_Data);**

**}**

**}**

**}**

**}**

**}**

**void Creat\_CNF\_ROW(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{//创建行约束条件**

**int ID = 0;**

**int value = 1;**

**int flag\_value = 0;**

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

**{**

**ID = Check\_Sudoku\_ID(NewList);**

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

**{//创建字句的约束**

**creatClaus(NewList, ID, num\_position(i, j, value), Decide\_buf, Record\_Data);**

**}**

**value++;**

**}**

**int temp\_i = 0;**

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

**{//值**

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

**{//行**

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

**{//列**

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

**{//下一列**

**creatClaus(NewList, ID, -num\_position(i, j, k), Decide\_buf, Record\_Data);**

**if(temp\_i%2==0)**

**ID = Check\_Sudoku\_ID(NewList);**

**}**

**}**

**}**

**}**

**}**

**void Creat\_CNF\_COL(MyList \*NewList,status Sudoku\_buf[][9],status Match\_buf[][10][10],Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{//创建列约束条件**

**int ID = 0;**

**int sign[10] = { 0 };**

**for (int j = 0; j < 9; j++)**

**{**

**for (int i = 0; i < 9; i++)//第 i + 1 列**

**{//遍历寻找该列已经出现的数字**

**if (Sudoku\_buf[i][j] != 0)**

**sign[Sudoku\_buf[i][j]] = j + 1;//sign[该位置的值]=位置横坐标 j**

**}**

**for (int m = 1; m <= 9; m++)**

**{//创建单子句**

**if (sign[m] != 0)**

**{**

**ID = Check\_Sudoku\_ID(NewList);**

**creatClaus(NewList, ID, num\_position(sign[m], j, m), Decide\_buf, Record\_Data);**

**}**

**}**

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

**{//创建 111 121 131 141 151 161 171 181 191 形式的约束规则**

**if (sign[k] == 0)**

**{**

**ID = Check\_Sudoku\_ID(NewList);**

**for (int m = 1; m <= 9; m++)**

**{**

**creatClaus(NewList, ID, num\_position(k, sign[m], m), Decide\_buf, Record\_Data);**

**}**

**}**

**}**

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

**{//创建两两约束的规则**

**}**

**}**

**}**

**void Creat\_CNF\_BLOCK(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{//创建块约束条件**

**int ID = Check\_Sudoku\_ID(NewList);**

**int x = 0, y = 0;**

**for (int block = 1; block <= 9; block++)**

**{**

**switch (block)**

**{**

**case 1:**

**ID = Check\_Sudoku\_ID(NewList);**

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

**{**

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

**{**

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

**{**

**}**

**}**

**}**

**break;**

**case 2:**

**break;**

**case 3:**

**break;**

**case 4:**

**break;**

**case 5:**

**break;**

**case 6:**

**break;**

**case 7:**

**break;**

**case 8:**

**break;**

**case 9:**

**break;**

**}**

**}**

**}**

**void Creat\_single\_clause(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{//创建单子句**

**int ID = 0;**

**for (int i = 0; i < 9; i++)**

**{**

**for (int j = 0; j < 9; j++)**

**{**

**if (Sudoku\_buf[i][j] != 0)**

**{**

**ID = Check\_Sudoku\_ID(NewList);**

**creatClaus(NewList, ID, num\_position(i + 1, j + 1, Sudoku\_buf[i][j]), Decide\_buf, Record\_Data);**

**}**

**}**

**}**

**}**

**//将数独转换成DPLL函数**

**void Tranlate(char \*filename, MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{**

**Match\_buf[10][10][10];**

**for (int i = 0; i < 10; i++)**

**for (int j = 0; j < 10; j++)**

**for (int k = 0; k < 10; k++)**

**Match\_buf[i][j][k] = 0;**

**Ini\_sudoku(Sudoku\_buf,Match\_buf);//初始化**

**Creat\_suduko(Sudoku\_buf);//随机填一些数创建数独**

**status find\_word\_in\_sudoku = 0;**

**find\_word\_in\_sudoku = find\_word\_in\_sudoku\_fun(Sudoku\_buf);**

**Display(Sudoku\_buf);**

**printf("\n\n");**

**printf("创建CNF文件\n\n");**

**creat\_sudoku\_file(filename,Sudoku\_buf,find\_word\_in\_sudoku);**

**printf("将CNF读进DPLL\n\n");**

**ReadFile(filename,NewList, Record\_Data, Decide\_buf);**

**int index\_j = 1;**

**int temp\_p\_n\_text = 0;**

**printf("DPLL求解过程\n\n");**

**DPLL(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);**

**status result = DPLL(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);**

**if (result == OK)**

**printf("求解成功！\n\n");**

**else if (result == ERROR)**

**printf("求解失败！\n");**

**printf("将DPLL求解结果转换为数独游戏！\n\n");**

**Translate\_to\_sudoku(NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data,find\_word\_in\_sudoku);**

**Display(Sudoku\_buf);**

**}**

**status creat\_sudoku\_file(char \*filename, status Sudoku\_buf[][9],status find\_word\_in\_sudoku)**

**{**

**int i = 1;**

**int j = 1;**

**int n = 1;**

**int r = 0;**

**int s = 0;**

**int m = 0;**

**int a = 0;**

**int b = 0;**

**int c = 0;**

**int d = 0;**

**int count = 0;**

**int numline = 11988+find\_word\_in\_sudoku;**

**int num = 889;**

**FILE \*fp;**

**//char filename[50];**

**char filepath[50];**

**char file[100];**

**printf("input filename: ");**

**scanf("%s", filename);**

**//printf("input filepath: "); scanf("%s", filepath);**

**//sprintf(file, "%s\%s", filepath, filename);**

**if ((fp = fopen(filename, "w")) == NULL)**

**printf("读取失败！\n");**

**fprintf(fp, "%s %d %d\n", "p cnf", num, numline);**

**for (i = 1; i<10; i++)**

**for (n = 1; n < 10; n++)**

**{**

**for (j = 1; j < 10; j++)**

**fprintf(fp, " %d%d%d ", i, j, n);**

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

**count++;**

**}**

**for (j = 1; j<10; j++)**

**for (n = 1; n < 10; n++)**

**{**

**for (i = 1; i < 10; i++)**

**fprintf(fp, " %d%d%d ", i, j, n);**

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

**count++;**

**}**

**for (r = 0; r<3; r++)**

**for (s = 0; s < 3; s++)**

**for (n = 1; n < 10; n++)**

**{**

**for (i = 1; i < 4; i++)**

**for (j = 1; j < 4; j++)**

**fprintf(fp, " %d%d%d ", 3 \* r + i, 3 \* s + j, n);**

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

**count++;**

**}**

**for (i = 1; i < 10; i++)**

**for (j = 1; j < 10; j++)**

**for (n = 1; n < 9; n++)**

**for (m = n + 1; m < 10; m++)**

**{**

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

**count++;**

**}**

**for (i = 1; i < 10; i++)**

**for (j = 1; j < 10; j++)**

**{**

**for (n = 1; n < 10; n++)**

**fprintf(fp, " %d%d%d ", i, j, n);**

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

**count++;**

**}**

**for (i = 1; i<10; i++)**

**for (n = 1; n < 9; n++)**

**for (m = n + 1; m<10; m++)**

**for (j = 1; j < 10; j++)**

**{**

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

**count++;**

**}**

**for (i = 1; i<10; i++)**

**for (n = 1; n < 9; n++)**

**for (m = n + 1; m<10; m++)**

**for (j = 1; j < 10; j++)**

**{**

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

**count++;**

**}**

**for (r = 0; r<3; r++)//块约束**

**for (s = 0; s < 3; s++)**

**for (n = 1; n < 10; n++)**

**for (i = 0; i < 8; i++)**

**for (j = i + 1; j < 9; j++)**

**{**

**fprintf(fp, " -%d%d%d -%d%d%d 0 \n", 3 \* r + i % 3 + 1, 3 \* s + i / 3 + 1, n, 3 \* r + j % 3 + 1, 3 \* s + j / 3 + 1, n);**

**count++;**

**}**

**//创建单子句**

**for (int i = 0; i < 9; i++)**

**{**

**for (int j = 0; j < 9; j++)**

**{**

**if (Sudoku\_buf[i][j] != 0)**

**{**

**fprintf(fp, " %d%d%d 0\n", i + 1, j + 1, Sudoku\_buf[i][j]);**

**}**

**}**

**}**

**//printf("%d\n", count);**

**fclose(fp);**

**return OK;**

**}**

**status Translate\_to\_sudoku(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data, status find\_word\_in\_sudoku)**

**{**

**int x = 0, y = 0, value = 0;**

**status Data = 0;**

**for (int i = 111; i < NewList->num + 1 + find\_word\_in\_sudoku\*10; i++)**

**{**

**if (Decide\_buf[i].Data\_status == 1)**

**{**

**Data = Decide\_buf[i].Data;**

**value = Data % 10;**

**Data = Data / 10;**

**y = Data % 10;**

**Data = Data / 10;**

**x = Data % 10;**

**Sudoku\_buf[x - 1][y - 1] = value;**

**}**

**}**

**return OK;**

**}**

**void sudoku\_game(MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{**

**int hole = 0;**

**int x = 0, y = 0, value = 0;**

**status Data = 0;**

**char filename[50];**

**printf("生成数独终盘\n\n");**

**Tranlate(filename, NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data);**

**printf("\n\n");**

**printf("请输入要挖的洞的个数:");**

**scanf("%d", &hole);**

**printf("\n\n");**

**for (int k = 0; k < hole; k++)**

**{//洞的个数**

**x = (rand() % 9);**

**y = (rand() % 9);**

**Sudoku\_buf[x][y] = 0;**

**}**

**printf("挖洞后的棋盘！\n\n");**

**Display(Sudoku\_buf);**

**printf("数独游戏求解!\n\n");**

**Destory(NewList, Decide\_buf, Record\_Data);**

**{**

**Record\_Data = NULL;**

**NewList = NULL;**

**Decide\_buf = NULL;**

**Decide\_buf = (Decide\_buffer\*)malloc(sizeof(Decide\_buffer)\*MAXLIST);**

**memset(Decide\_buf, 0, sizeof(Decide\_buffer)\*MAXLIST);**

**NewList = (struct MyList\*)malloc(sizeof(MyList));**

**NewList->num = 0;**

**NewList->numLine = 0;**

**memset(NewList->vertices, 0, sizeof(LnodeClause)\*MAXLIST);**

**Record\_Data = (Association\_Data\*)malloc(sizeof(Association\_Data)\*MAXLIST);**

**memset(Record\_Data, -1, sizeof(Association\_Data)\*MAXLIST);**

**}**

**Tranlate(filename, NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data);**

**printf("求解结果!\n\n");**

**Display(Sudoku\_buf);**

**}**

**status Judge\_hole(char \*filename, MyList \*NewList, status Sudoku\_buf[][9], status Match\_buf[][10][10], Decide\_buffer \*Decide\_buf, Association\_Data \*Record\_Data)**

**{**

**Display(Sudoku\_buf);**

**status find\_word\_in\_sudoku = 0;**

**find\_word\_in\_sudoku = find\_word\_in\_sudoku\_fun(Sudoku\_buf);**

**printf("\n\n");**

**printf("创建CNF文件\n\n");**

**creat\_sudoku\_file(filename, Sudoku\_buf,find\_word\_in\_sudoku);**

**printf("将CNF读进DPLL\n\n");**

**ReadFile(filename, NewList, Record\_Data, Decide\_buf);**

**int index\_j = 1;**

**int temp\_p\_n\_text = 0;**

**printf("DPLL求解过程\n\n");**

**DPLL(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);**

**status result = DPLL(NewList, index\_j, temp\_p\_n\_text, Record\_Data, Decide\_buf);**

**if (result == OK)**

**printf("求解成功！\n\n");**

**else if (result == ERROR)**

**printf("求解失败！\n");**

**printf("将DPLL求解结果转换为数独游戏！\n\n");**

**Translate\_to\_sudoku(NewList, Sudoku\_buf, Match\_buf, Decide\_buf, Record\_Data,find\_word\_in\_sudoku);**

**if (Judge\_rule\_sudoku(Sudoku\_buf) == OK)**

**return OK;**

**else**

**return ERROR;**

**}**

**status Judge\_rule\_sudoku(status Sudoku\_buf[][9])**

**{**

**for (int i = 0; i < 9; i++)**

**{**

**for (int j = 0; j < 9; j++)**

**{**

**if (Sudoku\_buf[i][j] == 0)**

**return ERROR;**

**}**

**}**

**return OK;**

**}**

**status find\_word\_in\_sudoku\_fun(status Sudoku\_buf[][9])**

**{**

**int count = 0;**

**for (int i = 0; i < 9; i++)**

**{**

**for (int j = 0; j < 9; j++)**

**{**

**if (Sudoku\_buf[i][j] != 0)**

**count++;**

**}**

**}**

**return count;**

**}**