

**课程设计报告**

**题目：基于SAT的数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS2002**

**学 号： U202015342**

**姓 名： 徐子路**

**指导教师： 李丹**

**报告日期： 2021.9.26**

**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数独游戏求解，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%)

**3.参考文献**

[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 for Programming, 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 International Symposium 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. SAT 2015, 223-237360

**目 录**

[1 引言 1](#_Toc85047542)

[1.1 课题背景与意义 1](#_Toc85047543)

[2 系统需求分析与总体设计 2](#_Toc85047544)

[2.1 系统需求分析 2](#_Toc85047545)

[2.2 系统总体设计 2](#_Toc85047546)

[3 系统详细设计 4](#_Toc85047547)

[3.1 有关数据结构的定义 4](#_Toc85047548)

[3.2 主要模块设计 7](#_Toc85047549)

[4 系统实现与测试 11](#_Toc85047550)

[4.1 系统实现 11](#_Toc85047551)

[4.2 系统测试 29](#_Toc85047552)

[5 优化策略与方案 38](#_Toc85047553)

[5.1 优化策略 38](#_Toc85047554)

[5.2 优化结果 38](#_Toc85047555)

[6 体会 40](#_Toc85047556)

[参考文献 41](#_Toc85047557)

[附录 42](#_Toc85047558)

# 1 引言

## 1.1 课题背景与意义

**1.1.1背景**

布尔逻辑的可满足性问题（SATISFIABLITY problem）是历史上第一个被证明的NP-complete问题，简称为SAT。布尔表达式是由布尔变量和运算符（NOT , AND , OR）所构成的表达式。如果对于变量的某个true,false赋值，使得一个布尔表达式的值为true，则该布尔表达式是可满足的。可满足性问题就是判定一个给定的合取范式的布尔公式是否是可满足的。

**1.1.2意义**

SAT问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可转化成SAT问题。因此致力于寻找求解SAT问题的快速而有效的算法，不仅在理论研究上而且在许多应用领域都具有极其重要的意义。

本次课程设计主要是利用在数据结构课上学到知识自主设计一个数据结构，用该数据结构存储cnf文件的信息，使用改进的DPLL算法编写程序实现SAT问题的求解。在设计过程中还要完成主界面的设计，并且将数独游戏的求解转化为SAT问题，利用之前编写的DPLL对其进行求解。

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

本设计要求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，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。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（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。

**程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现，提供明确的性能优化率结果。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。

**SAT应用：**将二进制数独游戏问题转化为SAT问题，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。

## 2.2 系统总体设计

本系统主要分为两个模块：SAT求解模块和数独模块。SAT求解模块所含有的功能有读取文件、展示子句、DPLL求解、重置和退出。数独模块含有的功能有创造棋盘，打印棋盘，填充棋盘、开始游戏、重置和退出。系统总体设计的结构图如图1。

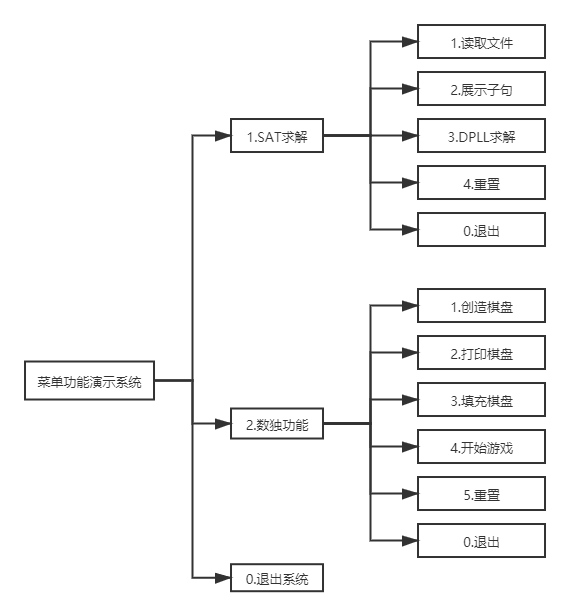
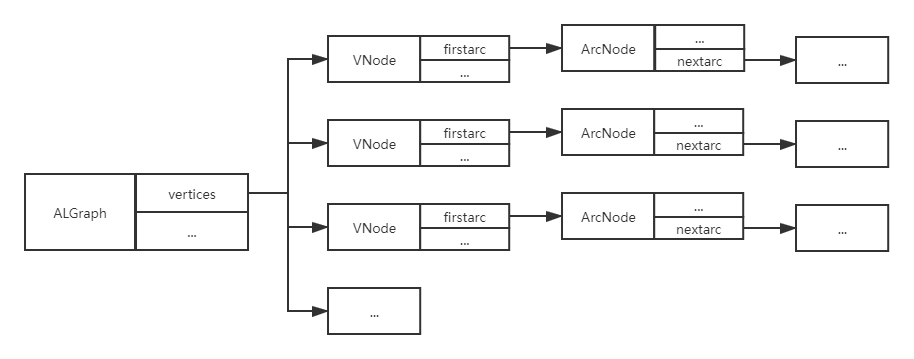


图1 系统总体设计图

# 3 系统详细设计

## 3.1 有关数据结构的定义

系统采取图结构，用邻接表保存cnf文件中的信息，并定义一个索引表和数据记录结构体formula方便对邻接表进行操作。邻接表总体结构如图2所示.

1）邻接表

图2 邻接表总体结构图

邻接表中含有四个成员：clause\_Gcount用于记录当前邻接表中的子句数目，

clause\_max用于记录子句的最大数目，clause\_ini用于记录初始化时分配的空间（在数独模块中构建邻接表时需要一个足够大的空间，而这个邻接表的子句数是未知的，因此需要一个足够大的clause\_ini去确保不会越界，而在SAT模块中，由于子句数目在文件中就已经确定好了，clause\_ini变量就失去了必要性），vertices则是邻接表的头结点数组（动态分配）。邻接表结构如图3所示。

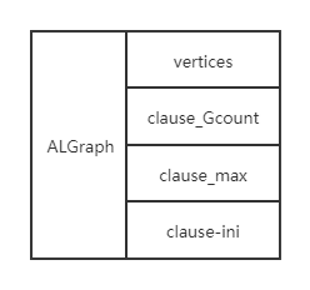


图3 邻接表结构图

每个头结点由一个名为VNode的结构体定义，其中包含以下成员：sequence用于记录子句序号，mark记录该子句是否被删除，literal\_Gcount记录子句中有多少文字（变元），firstarc则为指向该子句第一个文字结点的指针。VNode的结构图如图4所示。

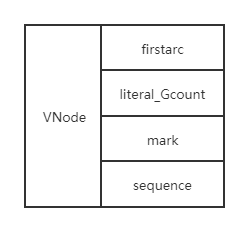


图4 头结点结构图

每一个文字结点又由一个名为ArcNode的结构体定义，其中包含以下成员：literal表示文字的序号（或取值），mark记录该文字是否被删除，nextarc则是指向下一个文字结点的指针。ArcNode的结构图如图5所示。

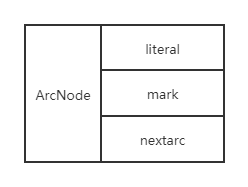


图5 文字结点结构图

2）索引表

formula结构体主要记录了邻接表的相关信息并提供了对邻接表中头结点和文字结点进行索引的功能。formula的总体结构图如图6。

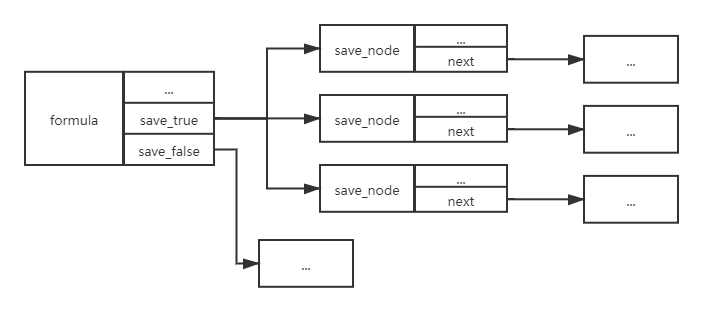


图6 formula总体结构图

其主要包含以下成员：ans记录SAT问题的结果（0为不可满足，1为可满足），time记录程序运行的时间，clause\_count记录子句的数目，literal\_count记录文字的数目，literal\_frequence为一个记录每个文字出现频率的数组（正文字记录为2n-1,负文字记录为-2n），literal\_tof为记录文字目前取值的数组，literal\_sequence为一个记录文字选择次序的数组（在DPLL每层选择文字进行处理时使用），save\_true和save\_false为两个命名为save\_form结构体数组，分别记录正负文字对应子句。formula的结构图如图7所示。

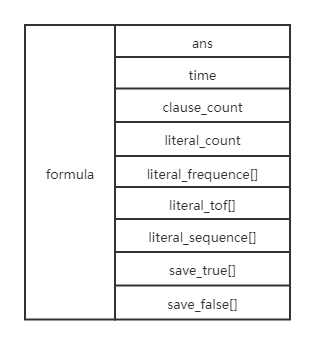


图7 formula结构图

save\_form中只有一个成员，save\_node。每一个文字都在formula中对应一个save\_form，记录文字对应的子句以及在子句中的位置。save\_node中含有以下成员：node为指向邻接表一文字结点的指针，mark\_clause为指向邻接表中一头结点的指针，way记录该文字对应的这个子句被删除的方式（-1表示未删除，0表示子句被整个删除，1表示单个文字被删除）save\_node的结构图如图8所示。

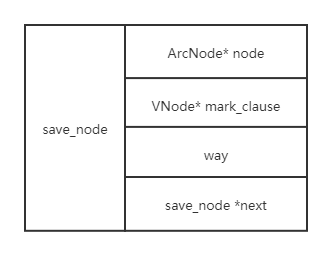


图8 save\_node结构图

## 3.2 主要模块设计

一、系统主控模块



图9 系统主控图

本系统共两个模块，SAT问题求解模块和数独求解模块，其中的核心部分是用DPLL算法求解SAT问题。

1.SAT问题求解模块

在SAT问题求解模块中，用户首先需要输入文件名，读取需要求解的SAT问题的cnf文件，这部分功能定义在功能1.ReadFile中。之后用户可以选择功能2.DisplayClauses将读取的文件显示出来。读取文件后，用户可以选择3.DPLL功能求解这个SAT问题。现对每一个功能进行详细说明。

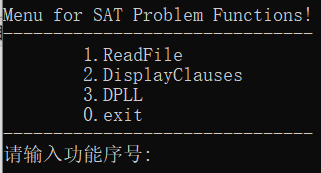


图10 SAT求解模块图

1）ReadFile

在实现ReadFile功能时，首先打印菜单，菜单中有两个选项，0为从最后检查文件列表选择，1为从当前文件列表中选择。之后输入文件名，调用readfile函数构造邻接表和索引表。ReadFile功能的流程图如图9.

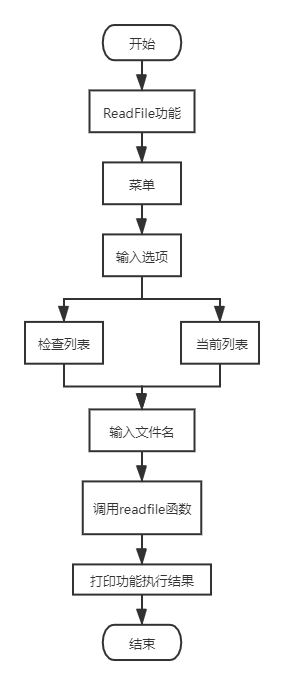


图11 ReadFile功能流程图

2）DisplayClauses

实现DisplayClauses功能只需要调用函数GraphPrint，这里不再详述。

3）DPLL

实现DPLL功能时，首先调用SAT\_Solve函数进行求解。然后调用makefile函数新建记录求解结果的res文件，最后销毁调用readfile函数时构建的邻接表和索引表。（DPLL函数的详细说明在4.1系统实现中）。

2.数独求解模块

数独求解模块大致分为两个版块，第一部分分为三个功能，首先需要使用1.CreateMap功能创建一个只有唯一解的棋盘，然后使用3.FillMap功能调用SAT求解模块的相关函数将棋盘化为SAT问题求解。功能2.PrintMap可以打印当前状态下的棋盘。第一个部分主要是用来测试程序能否正确对数独问题进行求解，因此与以交互为主的第二部分分开。第二部分只有一个功能，即4.PlayGame。在这个功能中，用户可以自主选择创建棋盘的方式并开始游戏。现对每一个功能进行详细说明。

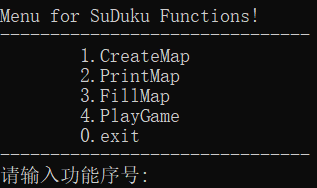


图12 数独求解模块图

1. CreateMap

在CreateMap功能中，调用CreateUniqueMap函数来创建棋盘。（该函数详见4.1系统实现）

1. PrintMap

在PrintMap功能中，调用PrintMap函数来打印棋盘。

1. FillMap

在FillMap功能中，首先调用Transform函数将棋盘转换为邻接表和索引表，然后调用MakeCnfFile函数生成cnf文件。再调用DPLL函数对转换的SAT问题求解。

1. PlayGame

在PlayGame功能中，调用Game函数开始游戏。

# 4 系统实现与测试

（4.1和4.2系统测试均用优化后的程序测试）

## 4.1 系统实现

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。

这部分可首先叙述一下你的系统实现的软硬件环境；

根据3.1的设计，用C语言定义各种数据类型；

程序代码部分在这里不需要给出来，只需要叙述清楚在系统中包括哪些函数，各函数的说明，如何利用这些函数实现系统各模块的功能，以及函数间的调用关系（可用图表示出来）。程序详见附录。

1.系统实现环境

本系统在windows10操作系统中实现，采用 Dev-C++ version 5.5.2编译器编写。其设备规格如图13.



图13 系统实现的设备规格图

2.数据类型

在3.1中已经给出数据结构定义的图例和解释，此处给出#define的相关定义和数据定义的C语言源码。

1. define定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

typedef int status;

1. 数据结构定义

typedef struct ArcNode {

int literal;

int mark;

struct ArcNode \*nextarc;

} ArcNode;

typedef struct VNode{

ArcNode \*firstarc;

int literal\_Gcount;

int mark;

int sequence;//子句序号

} VNode,\*AdjList;

typedef struct {

AdjList vertices;

int clause\_Gcount; //邻接表中当前的子句数目

int clause\_max;//子句最大数目

int clause\_ini;//分配空间时用，主要针对数独

} ALGraph;

typedef struct save\_node{//文字对应的每个子句

ArcNode\* node;//查找该文字在子句中是否被删除

VNode\* mark\_clause;//查找该子句是否被删除

int way;//记录该记录删除的方式,0:整个子句删除 1:单个文字删除

save\_node\* next;

}save\_node;

typedef struct save\_form{//每个文字对应一个save

save\_node\* p;//子句的指针

}save\_form;

typedef struct formula{//记录邻接表中各种信息

int ans;//结果

int time;//运行时间

int clause\_count;//子句的数目

int literal\_count;//文字的数目

int\* literal\_frequency;//记录文字出现的频率（正负分别记录）

int\* literal\_tof;//记录文字目前的取值（1表示真，0表示假，-1表示未完成）

int\* literal\_sequence;//记录数据提取顺序

save\_form \*save\_true;//正文字对应子句

save\_form \*save\_false;//负文字对应子句

}formula;

3.函数说明

1）基本功能函数（定义在头文件Graph & Formula.h中）

①数组赋值

函数声明：void reset(int \*s,int n,int x)

函数功能：将大小为n的int类型数组中所有元素赋值为x。

②初始化邻接表

函数声明：void InitialGraph(ALGraph &G)

函数功能：对邻接表G进行初始化。

③销毁邻接表

函数声明：void DestroyGraph(ALGraph &G)

函数功能：将邻接表G的所有成员归0，释放分配的空间。

④添加文字

函数声明：void AddVertex(ALGraph &G,int n,int literal)

函数功能：在读取cnf文件时，向邻接表G中第n个子句添加文字literal。

函数思想：声明ArcNode\*指针变量p和q。为q分配空间并将literal赋值给q->literal。用指针p遍历G.vertices[n]，运用图结点的添加操作将q添加到其末尾。

⑤删除子句

函数声明：void DeleteClause(ALGraph &G,formula &Formula,int n)

函数功能：将邻接表G中的第n个子句删除（将该子句的mark标为0）。

⑥删除文字

函数声明：void DeleteLiteral(ALGraph &G,formula &Formula,int literal)

函数功能：删除邻接表G中所有含literal的文字，并对索引表进行相应的修改。

函数思想：分literal>0和literal<0两种情况进行操作。如果literal>0，声明一个save\_node\*类型的指针p，将其赋值为Formula.save\_true[literal].p，如果p为空则直接return;否则通过p遍历含literal的所有结点，将p->way赋值为1（表示该节点被直接删除而不是因为子句删除被删除），将p->mark标为0，同时修改该文字所在子句literal\_Gcount(p->markclause->literal\_Gcount)。如果literal<0，则将p赋值为Formula.save\_false[literal].p，其他相同。

⑦打印邻接表

函数声明：void GraphPrint(ALGraph G)

函数功能：将邻接表G中每一个子句所含有的文字打印出来。

⑧初始化索引表

函数声明：void FormulaInitial(formula &Formula)

函数功能：将索引表Formula中所有成员初始化并给其中的数组分配空间。

⑨销毁索引表

函数声明：void DestroyFormula(formula &Formula)

函数功能：将索引表Formula中所有成员归零，并释放之前分配的空间。

⑩寻找出现频率最高的文字

函数声明：int SearchLiteral\_M(formula Formula)

函数功能：遍历Formula中的成员frequence数组，寻找其中出现频率最高的文字（frequence数组中，文字x取正时对应frequence[2\*x-1]，文字x取负时对应frequence[2\*x]，详见SAT问题求解模块中的readfile函数）。

⑪文字排序

函数声明：void SequenceSort(formula &Formula)

函数功能：对索引表Formula中按所有文字出现频率的大小进行排序（同一文字符号不同则算作不同文字）

函数思想：用一个for循环（循环变量为i）对1~2\*Formula.literal\_count进行遍历，每一次循环调用SearchLiteral\_M函数并将返回值赋值给x，再将x赋值给Formula.literal\_sequence[i]，根据x的正负确定x在frequence中的次序，并将frequence中x对应的值归0，这样第i次循环选择出第i大的数，最后Formula.literal\_sequence成为按文字出现频率从大到小的数组。

⑫匹配索引表

函数声明：void MatchFormula(ALGraph G,formula &Formula)

函数功能：根据已经建立好的邻接表G，构建相应的索引表，使Formula.save\_true和Formula.save\_false数组每一个元素i对应一个链表，链表的每一个元素含有指向文字i所在某一子句的各类指针，包括文字i所在该子句的头结点的指针（VNode \*）、文字i在该子句中所处结点的指针（ArcNode \*）。（具体的结构见3.1图8）

函数思想：该函数的流程图如图14。i是循环变量，每一个i对应一个子句。p是头结点的指针，用来遍历整个子句链表。创造一个save\_node类型的结点，记录当前结点的相关信息后，接到其对应文字的索引表里，直到遍历完整个邻接表。

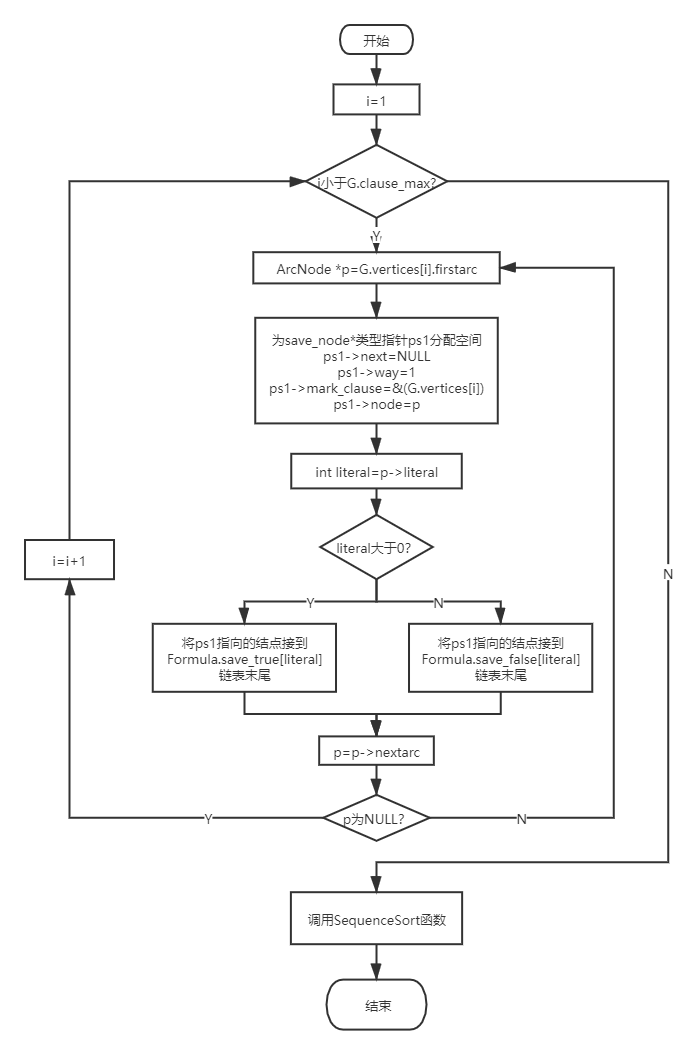


图14 MatchFormula函数流程图

⑬文字取值打印

函数声明：void Literal\_Print(formula Formula)

函数功能：显示每一个文字的取值情况（真/假/未赋值）。

2）SAT问题求解模块

①读取文件

函数声明：status readfile(char \*FileName,formula &Formula,ALGraph &G);

函数功能：读取名为FileName的cnf文件，如果读取成功，构建相应的邻接表和索引表，返回OK；否则返回ERROR。

函数思想：运用while循环过滤注释，如果无法读取到cnf的相关说明（p,cnf等字符串），返回ERROR，函数结束。否则，读入文件中记录的文字数和子句数，分别赋值给Formula.literal\_count和Formula.clause\_count。调用FormulaInitial函数并对邻接表G中G.clause\_Gcount、G.clause\_max、G.clause\_ini进行赋值。调用InitialGraph函数对邻接表进行初始化。然后用一个for循环（循环变量为i），每一层代表一个子句，依次读入cnf文件中子句中的文字，调用AddVertex函数将这些文字添加到邻接表G中，并在索引表Formula中的成员frequence数组里做好记录（文字取正时frequence [2\*literal-1]加1，取负时frequency[-literal\*2]加1）。循环完成后，调用MatchFormula函数依照构造好的邻接表构造索引表。

②删除函数中的永真式

函数声明：void Delete\_Tautologies(ALGraph &G,formula &Formula)

函数功能：删除邻接表G所有字句中存在的永真式。

函数思想：用一个for循环（循环变量为i）遍历所有的文字。声明save\_node\*类型的指针p1和p2，p1赋值为Formula.save\_true[i].p（文字i取正值时所在的所有字句链表的头指针，详见3.1Formula结构），p2赋值为Formula.save\_false[i].p。

遍历p2所在的链表，对于每一个结点，判断其对应的字句序号是否与p1对应的字句序号相同。（p1->mark\_clause->sequence == p2->mark\_clause->sequence）。

如果相同，说明有一个子句同时存在文字i的正负两种取值，则该子句是永真式，删除该子句。

③单子句规则删除文字（注：仅代表删除方式与单子句规则一致）

函数声明：void DeleteUnitartLiteral(ALGraph &G,formula &Formula,int literal)

函数功能：根据单子句规则，删除literal所在的所有字句，并把literal的反在邻接表各个子句中依次删除。

函数思想：该函数的流程图如图15。根据literal的正负，确定其取值以及索引表的位置（save\_true或save\_false）。删除文字所在子句前先根据p->mark是否为0来判断该子句是否被删除过。如果没有则将p->mark标为0并将p->way也标为0（p->way不仅代表了删除的方式，同样记录了是因为哪个文字删除的，因为只有这个文字对应索引表中p->way为0，该子句中其他文字虽然也被删除，但索引表中p->way为-1）之后指针p后移直到p为NULL。

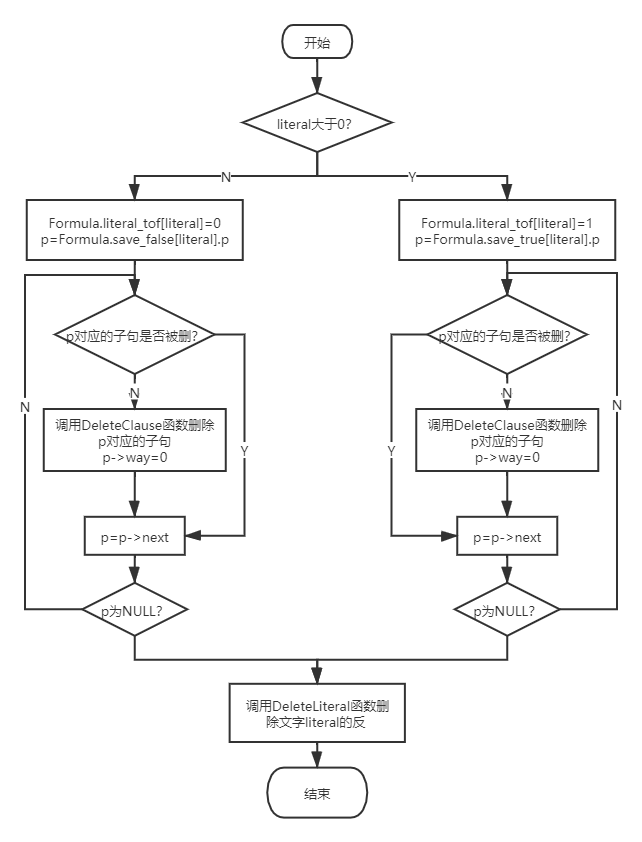


图15 DeleteUnitaryLiteral函数流程图

④恢复被删除的文字

函数原型：void RestoreLiteral(ALGraph &G,formula &Formula,int literal)

函数功能：将邻接表和索引表恢复到该文字被删除前的状态。

函数思想：与DeleteUnitaryLiteral的流程一致，将前者修改的部分依次改回来即可，不同的地方在于恢复时要根据结点被删除的方式判断能否恢复。在该函数中，首先恢复被删除的子句。用指针p遍历索引表中文字literal对应链表，对每个结点如果p->way等于0，说明该节点指向的子句被整个删除，对其进行恢复。但是在对该子句其他文字对应索引表进行修改时，要判断这些文字是否因为其他原因被删除，如果没有删除才能恢复。

⑤判断是否有空子句

函数原型：status JudgeEmptyClause(ALGraph G)

函数功能：遍历邻接表的所有头结点，根据其记录的信息判断对应子句是否为空，若存在空子句则TRUE，否则返回FALSE。

⑥寻找操作的文字

函数原型：int SearchLiteral(ALGraph G,formula &Formula,int &flag)

函数功能：如果当前邻接表中存在单子句，则选择第一个单子句中的文字作为下一层DPLL中删除的文字，并将flag标记为1。如果不存在单子句，则根据最初记录在邻接表中的Formula.literal\_sequence数组依次选择未被删除过的文字作为删除对象，并将flag标记为0。

⑦DPLL算法

函数原型：status DPLL(ALGraph &G,formula &Formula,int literal)

函数功能：通过DPLL算法判断邻接表G对应的SAT问题是否可满足，如果可满足则给出每个文字的取值。

函数思想：该函数的流程图如图16。每一层的flag用于判断该层在选下一次DPLL删除文字x的方式，flag取0表示单子句，取1表示按sequence数组中的文字顺序。如果是单子句则不需要将-x赋值为真。最初传入的literal为0。

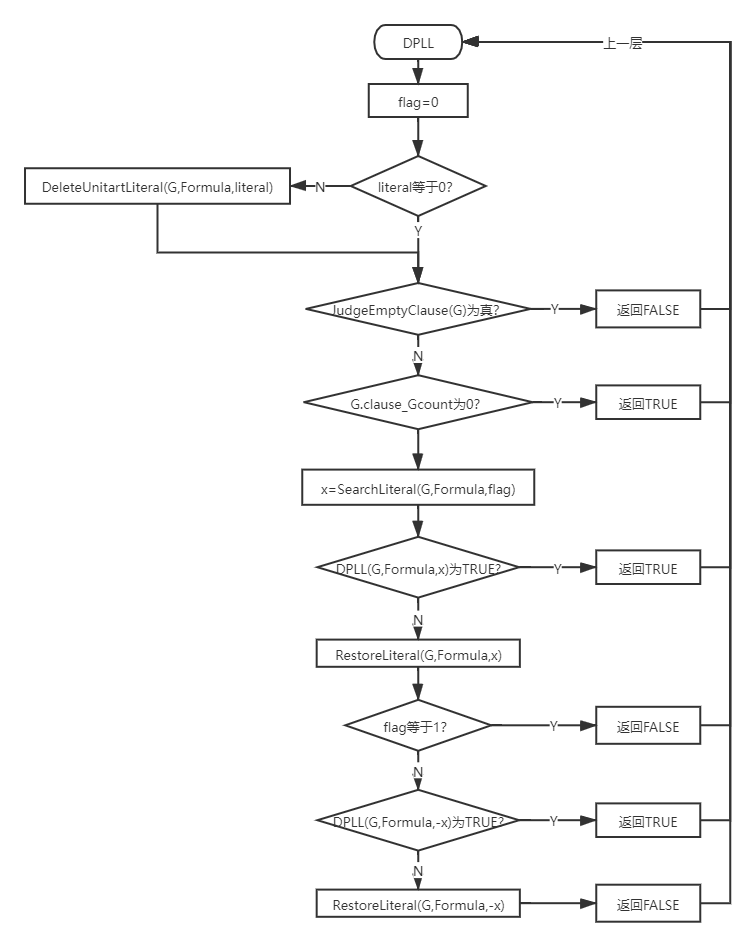


图16 DPLL函数流程图

⑧SAT问题求解

函数声明：void SAT\_Solve(ALGraph &G,formula &Formula)

函数功能：对SAT问题进行求解，打印求解结果并计算求解时间。

函数思想：函数流程图如图17。在SAT函数中，首先声明clock\_t类型的变量start和finish，调用头文件time.h中的clock()函数赋值给start开始计时。调用Delete\_Tautologies函数删除重言式，然后调用DPLL函数并记录结果。再次调用函数clock()赋值给finish，并把finish和start之差以毫秒为单位记录到索引表中，至此SAT\_Solve函数结束。

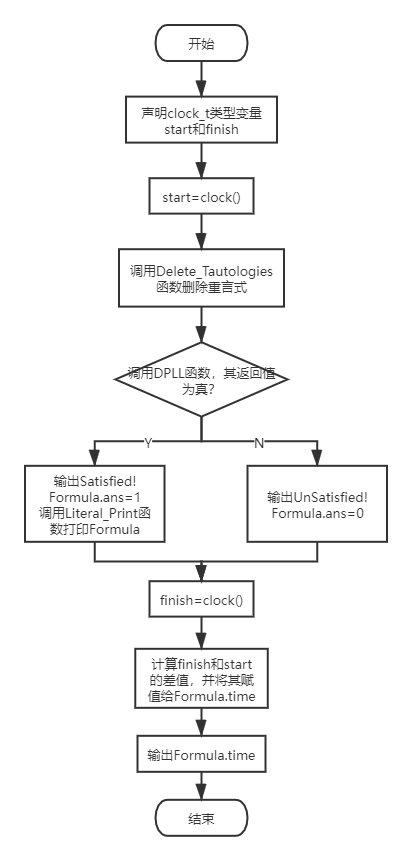


图17 SAT\_Solve函数流程图

⑨生成res文件

函数声明：status makefile(formula Formula,char \*FileName)

函数功能：根据索引表中对SAT问题求解结果的记载，生成相应的res文件。

3）数独模块

a.数独问题转化成SAT问题及相关基础功能

①打印棋盘

函数声明：void PrintMap(int ChessMap[][10])

函数功能：将传入的二维数组以数独格局打印出来

②判断能否正确填入

函数声明：status JudgeCorrect(int ChessMap[][10],int x,int n,int m)

函数功能：判断数字x能否填入棋盘ChessMap中第n行第m列。

函数思想：先判断棋盘中第n行是否有文字x，如果有则返回ERROR；然后判断第m列是否有文字x，如果有则返回ERROR；最后判断n行m列所在的大方块中有无文字x，如果有则返回ERROR。如果上述判断都满足，则返回OK。

③创建棋盘

函数声明：void CreateMap(int ChessMap[][10])

函数功能：创建一个只含有17个数字、随机分布且满足数独规则的棋盘。

函数思想：首先初始化棋盘ChessMap，调用srand函数初始化随机种子，用for循环（循环变量为k，循环17次），每层循环随机取得行数i、列数j、填入的数字x。如果棋盘中第i行j列还未填入数字，并且调用函数JudgeCorrect(ChessMap,x,i,j)的返回值为真，则将x赋值给ChessMap[i][j]，否则k减一。

④赋值棋盘

函数声明：void ChessMapCpy(int ChessMap[][10],int ChessMap\_S[][10])

函数功能：将ChessMap中的内容复制给ChessMap\_S

⑤文字匹配

函数声明：void LiteralMatch(formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：将棋盘中每一个空每一个可能的取值依次编号，每一个编号对应cnf文件中一个文字（变元），并将文字的总数记录到索引表中。

函数思想：用一个变量sq记录文字的序号，用双层for循环（循环变量为i,j）遍历棋盘每一个空，对每一个空用for循环（循环变量为x）从1到9遍历，然后依次用JudgeCorrect函数判断x能否填入，如果能填入则将Value[i][j][x]赋值为sq，sq自增1。

⑥读取文字

函数声明：void ReadLiteral(ALGraph &G,formula &Formula,int x)

函数功能：若x不为0，调用AddVertex函数将文字x添加到邻接表G当前最后一个子句的最后一个子句中，并对索引表进行相应的修改。若x为0，则G的最后一个子句下移。（G.clause\_Gcount和G.clause\_max加一）

⑦单格取值限制

函数声明：void UnitRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中每一个单格，寻找其中能填的所有取值，然后将这些取值两两匹配，将每组对应的文字取负值，作为邻接表的一个子句。（因为每一个格子只能填一个数，所以对于这个格子所有能填的数，不可能同时填入）

⑧行取值限制

函数声明：void RowRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中的每一行，寻找能填的所有取值，然后将这些取值两两匹配，将每组对应的文字取负值，作为邻接表的一个子句。

⑨列取值限制

函数声明：void LineRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中的每一列，寻找能填的所有取值，然后将这些取值两两匹配，将每组对应的文字取负值，作为邻接表的一个子句。

⑩块取值限制

函数声明：void BlockRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中的每一块（共9个3×3块），寻找能填的所有取值，然后将这些取值两两匹配，将每组对应的文字取负值，作为邻接表的一个子句。

⑪单格取值判定

函数声明：void UnitEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中每一个单格，寻找其中能填的所有取值，然后将它们对应的文字组成一个子句添加到邻接表中。

⑫行取值判定

函数声明：void RowEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中每一行，寻找其中能填的所有取值，然后将它们对应的文字组成一个子句添加到邻接表中。

⑬列取值判定

函数声明：void LineEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中每一列，寻找其中能填的所有取值，然后将它们对应的文字组成一个子句添加到邻接表中。

⑭块取值判定

函数声明：void BlockEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：对于棋盘中每一块，寻找其中能填的所有取值，然后将它们对应的文字组成一个子句添加到邻接表中。

⑮数独问题转换

函数声明：void Transform(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：将一个数独棋盘转化为SAT问题，构建相应的邻接表和索引表。

函数思想：首先调用LiteralMatch函数初始化文字变元，将G.clause\_initial赋值为6000后调用InitialGraph函数初始化邻接表，再调用FormulaInitial函数初始化索引表。之后调用之前的判定和取值限制函数生成子句补充邻接表，最后调用MatchFormula函数根据邻接表构造相应的索引表。该函数流程图如图18。



图18 Transform函数流程图

⑯结果匹配

函数声明：void AnswerMatch(formula &Formula,int ChessMap[][10],int Value[][10][10])

函数功能：根据索引表中SAT问题的求解结果，将其转换成棋盘上的数字（LiteralMatch函数的功能反过来）。

b.数独游戏交互模块及游戏模块相关函数。

①判断多解

函数原型：status JudgeUnique(int ChessMap[][10],int PlayMap[][10],int n,int m)

函数功能：判断棋盘ChessMap中第n行第m列是否能填入多个能确保棋盘解唯一数字。

函数思想：首先定义三维int类型数组value并进行初始化，然后调用LiteralMatch函数将棋盘每个空格可能的取值转化成文字。用for循环（循环变量为x）遍历1~9每一个取值，如果该格是最初ChessMap的17个数之一或者调用JudgeCorrect(PlayMap,x,n,m)的取值为FALSE，继续循环；否则将x赋值给PlayMap[n][m]，调用Transform函数将PlayMap转化为邻接表和索引表。调用DPLL函数对该邻接表和索引表进行求解，如果返回值为TRUE，则计数器num自增1。每层循环都需要判断num是否大于0，如果大于1说明传入的PlayMap中第n行第m列有多于1个x使填入x后棋盘的解唯一，这时将PlayMap还原成传入时的状态，并返回FALSE。

②判断空棋盘

函数声明：status JudgeEmpty(int Hole[][10])

函数功能：判断数组Hole是否所有元素都为0

③寻找下一个洞

函数声明：int SearchHole(int PlayMap[][10],int Hole[][10])

函数功能：随机寻找下一个洞的行坐标n和列坐标m，如果找到则返回10\*n+m。

函数思想：运用随机数产生行n和列m，调用JudgeEmpty函数判断Hole是否为空，如果返回值为TRUE，则返回0，否则用while循环每层循环取随机数n和m，直到Hole[n][m]不为0时，返回n和m的组合数。

④挖洞法

函数声明：status DigHole(int n,int m,int ChessMap[][10],int PlayMap[][10],int Hole[][10],int &num)

函数功能：ChessMap为最初产生的含17个数的棋盘，PlayMap为求解后填满的棋盘。Hole记录挖过的洞，num记录还需要挖多少洞。该函数是一个递归函数，最后的效果是在PlayMap上随机挖num个洞，并确保挖完后棋盘有唯一解。

函数思想：首先判断是否是第一次挖洞。如果是第一次挖洞（n和m均为0），直接用随机数给n和m赋值，然后调用DigHole开始挖洞。如果不是第一次，则先判断num的大小，如果为0说明已经挖完，返回OK。若没有，调用JudgeUnique函数判断该空格是否有多解，如果有多解直接返回ERROR；否则让PlayMap[n][m]为0，Hole[n][m]也为0，num自减1（表示该格已被挖）。调用ChessMapCpy函数将当前的Hole保存到hole\_s数组中，调用SearchHole函数寻找下一个要挖的洞x。使用while循环（终止条件是x等于0），每一次循环都对x对应的n和m调用DigHole函数进行挖洞，如果返回值为OK则返回OK，否则用Hole数组标记该x对应的n和m为0（标记为0表示不能挖或者已经挖过），然后继续调用 SearchHole函数寻找下一个要挖的洞x。如果在一层DigHole中跳出了这个while循环，说明当前棋盘不存在一个挖洞方案使将本层传入的n行m列挖去后棋盘其有唯一解，这时进行回溯。令num自增1，把PlayMap和Hole还原成挖洞前的状态，返回ERROR。该函数流程图如图19。

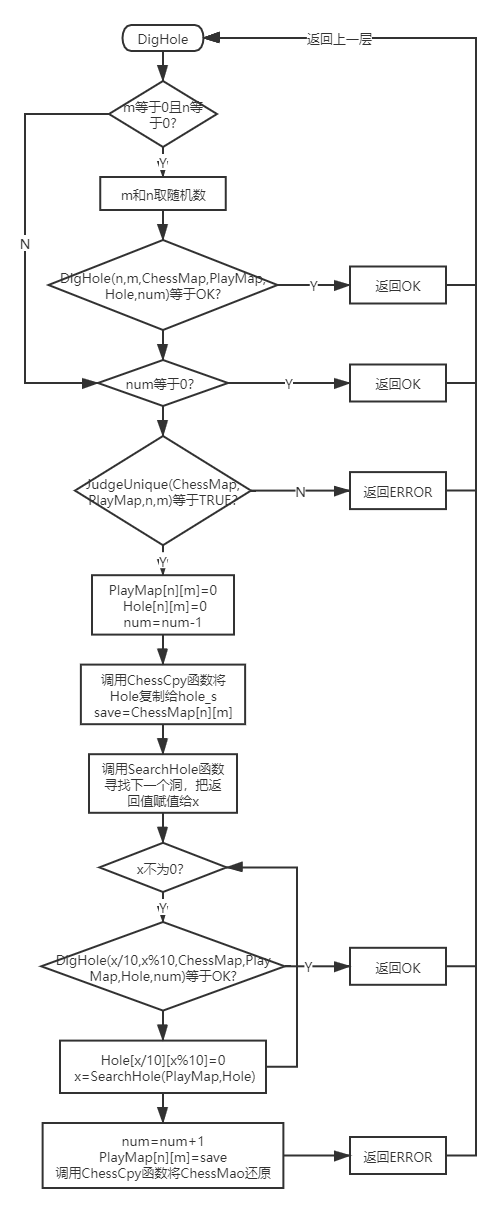


图19 DigHole函数流程图

⑤创造唯一解的棋盘

函数声明：status CreateUniqueMap(ALGraph &G,formula &Formula,int PlayMap[][10],int ChessMap[][10] ,int Hole[][10])

函数功能：先创建一个已经填完且符合数独规则的棋盘，然后由用户输入挖洞个数，之后进行挖洞且确保只有一个唯一解。

函数思想：定义一个状态量flag，初始值为1。用while循环（跳出循环条件是flag等于0，每层循环初始化PlayMap、ChessMap、Value和Hole（Hole数组初始化为1），然后调用CreateMap函数创建一个只有17个数的棋盘，调用Transform函数构建邻接表和索引表后，用DPLL求解。如果求解结果为OK，则让flag为0，跳出循环；否则继续循环（当前棋盘无解）。跳出循环后，用AnswerMatch函数将求解结果反映到棋盘上，得到完整的棋盘，然后调用ChessMapCpy函数将ChessMap复制给PlayMap。定义一个int类型变量num，用用户输入num的值。调用DigHole函数挖洞。

⑥读取棋盘

函数声明：void ReadMap(int ChessMap[][10],char SAVE[][90])

函数功能：从suduku.txt文件中读取棋盘，并将其赋值给ChessMap。

⑦开始游戏

函数声明：void GamePlay(int PlayMap[][10],int ChessMap[][10])

函数功能：数独答案记录在ChessMap中，数独游戏棋盘记录在PlayMap中。本函数实现和玩家的交互功能，玩家可以每次可以选择填入数字、删除自己填入的数字、取得提示和退出游戏。退出游戏时，判定玩家是否正确求解并给出正确结果。

⑧产生cnf文件

函数声明：void MakeCnfFile(ALGraph G,formula Formula)

函数功能：根据邻接表和索引表生成一个cnf文件

## 4.2 系统测试

首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程，（1）先明确模块的功能、设计目标等。（2）分析、叙述如何选取测试数据，要求有完整的测试大纲。（3）运行结果（这时可用截图）。（4）分析运行结果、确认程序满足该模块的设计目标。

1. SAT问题求解模块测试
2. 测试说明

本模块的测试主要以助教检查时的测试集为主，再加上“SAT任务测试备选算例”中的部分算例作为补充。测试的程序为优化后的程序。

1. 测试样例

表1 SAT问题求解模块测试样例

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试样例 | 文件名称 | 测试结果 | 变元数 | 子句数 | 运行时间 | 优化率 |
| 样例1 | 1.cnf | Satisfied | 200 | 1200 | 21078ms | 12.92% |
| 样例2 | 2.cnf | Satisfied | 1075 | 3152 | 3138ms | 92.21% |
| 样例3 | 3.cnf | Satisfied | 301 | 2780 | 116ms | 98.42% |
| 样例4 | 4.(unsatisifed).cnf | Unsatisfied | 512 | 9685 | 2653ms | 100% |
| 样例5 | 5.cnf | Satisfied | 20 | 1532 | 48ms | 100% |
| 样例6 | 6.cnf | Satisfied | 265 | 5666 | 57768ms | 100% |
| 样例7 | sud00012.cnf | Satisfied | 232 | 1901 | 100ms | 94.72% |
| 样例8 | sud00021.cnf | Satisfied | 308 | 2911 | 2825ms | 79.67% |
| 样例9 | sud00079.cnf | Satisfied | 301 | 2810 | 257ms | 84.25% |
| 样例10 | eh-dp04s04.shuffled-1075.cnf | Satisfied | 1075 | 3152 | 3124ms | 100% |
| 样例11 | sud00082.cnf | Satisfied | 224 | 1762 | 71ms | 100% |
| 样例12 | sud00861.cnf | Satisfied | 297 | 2721 | 174ms | 92.77% |
| 样例13 | tst\_v10\_c100.cnf | Unsatisfied | 10 | 100 | 0ms | 100% |
| 样例14 | u-5cnf\_3900\_3900\_060.shuffled-60.cnf | Unsatisfied | 60 | 936 | 317372ms | 74.21% |
| 样例15 | problem11-100.cnf | Satisfied | 100 | 600 | 84 | 84.32% |
| 样例16 | 7cnf20\_90000\_90000\_7.shuffled-20.cnf | Satisfied | 20 | 1532 | 47 | 85.22% |
| 样例17 | sud00001.cnf | Satisfied | 301 | 2780 | 125 | 94.77% |
| 样例18 | problem9-100.cnf | Satisfied | 100 | 200 | 30569 | 100% |

1. 测试截图

1）样例1

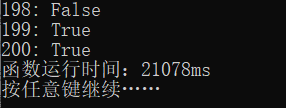


图20 样例1测试图

2）样例2

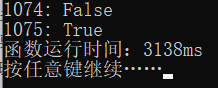


图21 样例2测试图

3）样例3

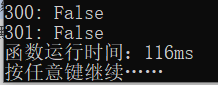


图22 样例3测试图

4）样例4

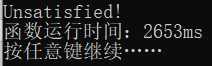


图23 样例4测试图

1. 样例5



图24 样例5测试图

6）样例6



图25 样例6测试图

7）样例7（sud00012.cnf）



图25 样例7测试图

8）样例8（sud00021.cnf）



图26 样例8测试图

9）样例9（sud00079.cnf）



图27 样例9测试图

10）样例10（eh-dp04s04.shuffled-1075.cnf）



图28 样例10测试图

11）样例11（sud00082.cnf）

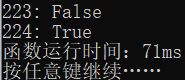


图29 样例11测试图

12）样例12（sud00861.cnf）

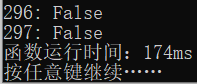


图30 样例12测试图

13）样例13（tst\_v10\_c100.cnf）

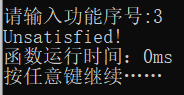


图31 样例13测试图

14)样例14（u-5cnf\_3900\_3900\_060.shuffled-60.cnf）

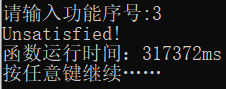


图32 样例14测试图

15）样例15（problem11-100.cnf）

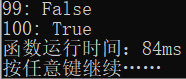


图33 样例15测试图

16）样例16（7cnf20\_90000\_90000\_7.shuffled-20.cnf）

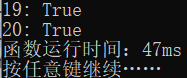


图34 样例16测试图

17）样例17（sud00001.cnf）

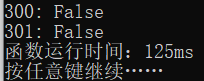


图34 样例16测试图

18）样例18（problem9-100.cnf）

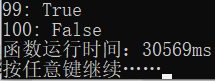


图35 样例17测试图

1. 数独模块测试
2. 测试说明

该部分测试主要测试能否正确产生棋盘并对其进行求解。测试样例按照一定的顺序连续操作。

1. 测试样例

表2 数独模块测试样例

|  |  |  |  |
| --- | --- | --- | --- |
| 次序 | 功能名称 | 功能说明 | 测试结果 |
| 1 | CreateMap | 产生一个有唯一解的棋盘 | 正确，如图29 |
| 2 | PrintMap | 打印填充前的棋盘 | 正确，如图30 |
| 3 | FillMap | 填充当前棋盘 | 正确，如图31 |
| 4 | PrintMap | 打印填充后的棋盘 | 正确，如图32 |
| 5 | PlayGame | 开始游戏 | 正确，如图33 |
| 6 | ReadMap | 从文件中读取棋盘 | 正确，如图34 |
| 7 | PresentAnswer | 显示正确答案 | 正确，如图35 |
| 8 | CreateMap | 产生一个有唯一解的棋盘 | 正确，如图36 |
| 9 | PresentAnswer | 显示正确答案 | 正确，如图37 |
| 10 | GameStart | 玩家开始完成数独 | 正确，如图38 |

1. 测试截图
2. 样例序号1（创建棋盘）

如图，正确创建一个有60个空位的棋盘。

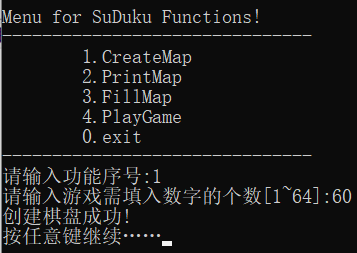


图36 样例1测试图

1. 样例序号2（打印棋盘）

将创建好的棋盘打印出来。

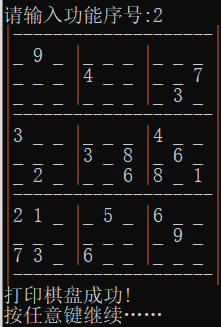


图37 样例2测试图

1. 样例序号3（填充棋盘）

将棋盘转化为SAT问题后，调用DPLL问题进行求解。



图38 样例3测试图

1. 样例序号4（打印棋盘）

将求解完的棋盘打印出来。

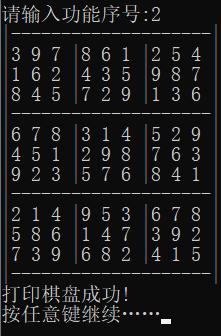


图39 样例4测试图

1. 样例序号5（开始游戏）

进入游戏的初始界面。

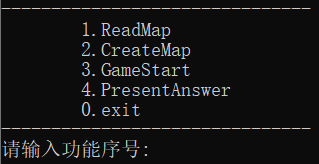
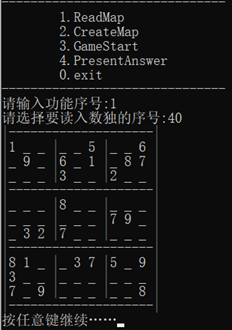


图40 样例5测试图

1. 样例序号6（读取棋盘）

从Sudoku.txt文件中读取棋盘。





（文件中棋盘的显示形式）

图41 样例6测试图

1. 样例序号7（显示答案）

对读取的棋盘进行求解，并将正确结果显示出来。

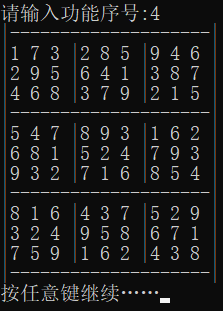


图42 样例7测试图

1. 样例序号8（创建棋盘）

自己创建一个游戏棋盘。

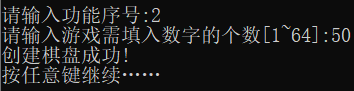


图43样例8测试图

1. 样例序号9（显示答案）

对创建的棋盘进行求解，并将正确结果显示出来。

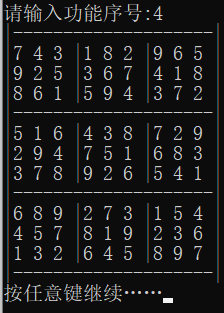


图44 样例9测试图

10）样例序号10

读取/创建好棋盘后，便可以开始游戏了。

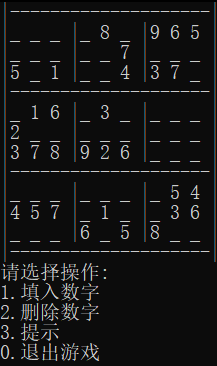


图45 样例10测试图

# 5 优化策略与方案

## 5.1 优化策略

在对DPLL算法进行优化时主要是对变元的选择策略以及部分基本功能的函数进行了修改。

1. 修改前

修改之前变元选择策略是首先检查邻接表中有无单子句，如果有单子句则直接选择单子句中的文字；如果没有单子句则选择索引表中frequence数组中出现频率最高的文字。

这种方案的明显缺点是每次删除单子句时要对整个子句中所有文字的出现频率在索引表中做相应的修改，即每次删除单子句多出一个while循环，导致时间复杂度的偏高。

1. 修改后

修改之后变元的选择策略在单子句的部分相同，区别时如果邻接表中没有单子句，则从索引表中的sequence数组中选择还未选择过的文字中出现频率最高的文字。

索引表的sequence数组中的元素是在索引表构建时就已经确定的，为索引表构建时元素出现频率从高到低的排序。因此不用在删除子句时修改其中文字的频率。经验证修改后对于大部分SAT问题求解效率显著提高。

## 5.2 优化结果

1．测试说明

本次测试主要选取基准算例中性能测试的两个样例ais10.cnf和sud00009.cnf进行测试。

2．测试样例

表3 DPLL算法优化测试样例

|  |  |  |  |
| --- | --- | --- | --- |
| 测试文件名 | 优化前求解时间 | 优化后求解时间 | 优化率 |
| ais10.cnf | 4172ms | 1371ms | 67.13% |
| sud00009.cnf | 5606ms | 300ms | 94.65% |

3．测试截图

1）ais10.cnf

a.优化前



图46 ais10.cnf优化前

b.优化后

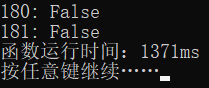


图47 ais10.cnf优化后

2）sud00009.cnf

a.优化前

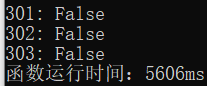


图48 sud00009.cnf优化前

b.优化后

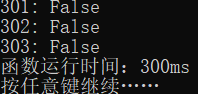


图49 sud00009.cnf优化后

# 6 体会

本次程序设计课程对我来说是一个比较大的考验。在此之前，我从来没有写过一个相对完整的程序，一般都是对某一个问题进行编程求解。从最开始研究这个课题时，我没过多久就大致明白了此次课设的任务要求，但是到正式开始写程序却花了很久，因为我怕最开始的思路就有问题导致写到中途就要重新来过。可这样的事情最后还是发生了。在第一版DPLL算法中，我严格采用了任务书中流程图的算法过程，即每一层DPLL进行多次单子句规则的操作，再选择出现频率最高的变元进入下一层DPLL。我的删除子句和文字全部采用了直接删除结点的方式，每一层DPLL的回溯的方式是在DPLL最初新建一个邻接表和索引表记录删除操作前的状态，而且索引表也只是记录了文字和子句的数目情况并没有指向邻接表结点的指针。可想而知，最初的DPLL算法完全不行。每一层新建图的开销太大，即使用free函数在一层结束时释放空间也无济于事（因为总是会在还没开始回溯的时候内存就不够了）。每次删除文字和子句都要依次遍历，而且删除用的是“真删除”导致时间复杂度过高（单就删除而言，最初的方案是O(n+m)，最后的方案是O(1)）。这一版DPLL连S里面的样例也跑不了几个。

后来在室友的启发下，我把索引表里面新增了save\_node用来存储指向文字所在子句结点的各类指针，同时修改了DPLL的结构，每一层只对一个文字进行删除操作，这样一来很多问题迎刃而解，依照这个思路不断优化后便得到了最终版。在此之后其实还有一些别的尝试，例如在某一个节点后选择文字数目3以下的子句中数目最少的变元等，最后要么实现起来过于复杂，要么优化后效率还不如优化前导致方案的失败，因此还是选用了本报告中的方案作为最后结果。

在我写程序的过程中，实现数独问题的求解遇到的问题比写DPLL算法遇到的问题更多。我一开始想了很久也没有想到如何将数独问题转化为DPLL问题，后面查阅许多资料以及向同学老师请教才知道了大致的思路。在实现挖洞法时也是绞尽脑汁，有几次都想放弃了采用室友的方案，不过最后还是用自己随机挖洞的思路实现了挖洞法，完成之后非常有成就感。唯一可惜的一点是最后只能挖到60个，可能需要改进或者换掉DPLL算法才能挖更多吧。

总的来说从这次课设中我收获颇丰，大大提高了我的编程能力，让我对一个完成的项目有了更深的认识，更加深了我对数据结构的理解和掌握。

# 参考文献

[1] 曹计昌，卢萍，李开. C语言与程序设计. 电子工业出版社，2013

[2]严蔚敏等.数据结构（C语言版）.清华大学出版社，

[3] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition,[Calvin College](http://cs.calvin.edu/),2005

[4] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197

[5] 严蔚敏等.数据结构题集（C语言版）.清华大学出版社

[6]知乎用户白龙鱼服算法学习笔记.网站：[基于DPLL算法的SAT工具箱 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/348334847)

# 附录

工程源代码

1.define.h

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

typedef int status;

2.Graph&Formula.h

#include"define.h"

typedef struct ArcNode {

int literal;

int mark;

struct ArcNode \*nextarc;

} ArcNode;

typedef struct VNode{

ArcNode \*firstarc;

int literal\_Gcount;

int mark;

int sequence;//子句序号

} VNode,\*AdjList;

typedef struct {

AdjList vertices;

int clause\_Gcount; //邻接表中当前的子句数目

int clause\_max;//子句最大数目

int clause\_ini;//分配空间时用，主要针对数独

} ALGraph;

typedef struct save\_node{//文字对应的每个子句

ArcNode\* node;//查找该文字在子句中是否被删除

VNode\* mark\_clause;//查找该子句是否被删除

int way;//记录该记录删除的方式,0:整个子句删除 1:单个文字删除

save\_node\* next;

}save\_node;

typedef struct save\_form{//每个文字对应一个save

save\_node\* p;//子句的指针

}save\_form;

typedef struct formula{//记录邻接表中各种信息

int ans;//结果

int time;//运行时间

int clause\_count;//子句的数目

int literal\_count;//文字的数目

int\* literal\_frequency;//记录文字出现的频率（正负分别记录）

int\* literal\_tof;//记录文字目前的取值（1表示真，0表示假，-1表示未完成）

int\* literal\_sequence;//记录数据提取顺序

save\_form \*save\_true;//正文字对应子句

save\_form \*save\_false;//负文字对应子句

}formula;

//功能函数

void GraphPrint(ALGraph G);

void SequenceSort(formula &Formula);

void reset(int \*s,int n,int x)//初始化数组

{

int max=n/4;

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

s[i]=x;

}

void InitialGraph(ALGraph &G)//初始化邻接表

{

G.vertices=(VNode \*)malloc((sizeof(VNode)+5)\*G.clause\_ini);

for(int i=1;i<=G.clause\_ini;i++)

{

G.vertices[i].firstarc=NULL;

G.vertices[i].literal\_Gcount=0;

G.vertices[i].mark=1;

G.vertices[i].sequence=i;

}

}

void DestroyGraph(ALGraph &G)//销毁邻接表

{

if(G.clause\_ini==0) return;

if(G.vertices){

free(G.vertices);

G.vertices=NULL;

}

G.clause\_ini=0;

G.clause\_max=0;

G.clause\_Gcount=0;

}

void AddVertex(ALGraph &G,int n,int literal)//向第n个子句添加文字

{

ArcNode \*p,\*q;

p=(ArcNode \*)malloc(sizeof(ArcNode));

p->literal=literal,p->nextarc=NULL,p->mark=1;

if(!G.vertices[n].literal\_Gcount){

G.vertices[n].firstarc=p;

G.vertices[n].literal\_Gcount++;

}

else{

q=G.vertices[n].firstarc;

while(q->nextarc){

q=q->nextarc;

}

q->nextarc=p;

G.vertices[n].literal\_Gcount++;

}

}

void DeleteClause(ALGraph &G,formula &Formula,int n)//删除子句

{

if(!G.vertices[n].mark) return;

G.vertices[n].mark=0;

G.vertices[n].literal\_Gcount=0;

G.clause\_Gcount--;

}

void DeleteLiteral(ALGraph &G,formula &Formula,int literal)//删除文字

{

if(literal>0){

save\_node \*p=Formula.save\_true[literal].p;

if(!p) return;

do{

if(p->mark\_clause->mark){

p->way=1;

p->node->mark=0;

p->mark\_clause->literal\_Gcount--;

}

p=p->next;

}while(p);

}

else{

save\_node \*p=Formula.save\_false[-literal].p;

if(!p) return;

do{

if(p->mark\_clause->mark){

p->way=1;

p->node->mark=0;

p->mark\_clause->literal\_Gcount--;

}

p=p->next;

}while(p);

}

}

void GraphPrint(ALGraph G)//打印邻接表

{

for(int i=1;i<=G.clause\_max;i++)

{

if(!G.vertices[i].mark) continue;

printf("%d: ",i);

ArcNode \*p=G.vertices[i].firstarc;

if(!p) continue;

do{

if(p->mark)

printf("%d ",p->literal);

p=p->nextarc;

}while(p);

printf("\n");

}

}

int SearchLiteral\_M(formula Formula)//寻找出现频率最高的文字

{

int max=0,x;

for(int i=1;i<=2\*Formula.literal\_count;i++)

{

if(Formula.literal\_frequency[i]>max){

max=Formula.literal\_frequency[i];

if(i%2) x=(i+1)/2;

else x=-i/2;

}

}

return x;

}

//初始化Formula

void FormulaInitial(formula &Formula)

{

Formula.ans=-1;

Formula.time=0;

Formula.literal\_frequency=(int \*)malloc(sizeof(int)\*(2\*Formula.literal\_count+10));

Formula.literal\_sequence=(int \*)malloc(sizeof(int)\*(2\*Formula.literal\_count+10));

Formula.literal\_tof=(int \*)malloc(sizeof(int)\*(Formula.literal\_count+10));

Formula.save\_true=(save\_form \*)malloc(sizeof(save\_form)\*(Formula.literal\_count+10));

Formula.save\_false=(save\_form \*)malloc(sizeof(save\_form)\*(Formula.literal\_count+10));

for(int i=0;i<=Formula.literal\_count;i++)

{

Formula.save\_true[i].p=NULL;

Formula.save\_false[i].p=NULL;

}

reset(Formula.literal\_sequence,sizeof(int)\*(2\*Formula.literal\_count+10),0);

reset(Formula.literal\_frequency,sizeof(int)\*(2\*Formula.literal\_count+10),0);

reset(Formula.literal\_tof,sizeof(int)\*(Formula.literal\_count+10),-1);

}

void DestroyFormula(formula &Formula)

{

if(Formula.literal\_count == 0) return;

if(Formula.literal\_frequency){

free(Formula.literal\_frequency);

Formula.literal\_frequency=NULL;

}

if(Formula.literal\_sequence){

free(Formula.literal\_sequence);

Formula.literal\_sequence=NULL;

}

if(Formula.literal\_tof){

free(Formula.literal\_tof);

Formula.literal\_tof=NULL;

}

if(Formula.save\_false){

free(Formula.save\_false);

Formula.save\_false=NULL;

}

if(Formula.save\_true){

free(Formula.save\_true);

Formula.save\_true=NULL;

}

}

void MatchFormula(ALGraph G,formula &Formula)

{

for(int i=1;i<=G.clause\_max;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

do{

save\_node \*ps1=(save\_node \*)malloc(sizeof(save\_node));

ps1->next=NULL,ps1->way=-1,ps1->mark\_clause=&(G.vertices[i]);

ps1->node=p;

int literal=p->literal;

if(literal>0){

save\_node \*ps2=Formula.save\_true[literal].p;

if(!ps2) Formula.save\_true[literal].p=ps1;

else{

while(ps2->next){

ps2=ps2->next;

}

ps2->next=ps1;

}

}

else{

save\_node \*ps2=Formula.save\_false[-literal].p;

if(!ps2) Formula.save\_false[-literal].p=ps1;

else{

while(ps2->next){

ps2=ps2->next;

}

ps2->next=ps1;

}

}

p=p->nextarc;

}while(p);

}

SequenceSort(Formula);

}

void SequenceSort(formula &Formula)

{

for(int i=1;i<=2\*Formula.literal\_count;i++)

{

int x=SearchLiteral\_M(Formula);

Formula.literal\_sequence[i]=x;

if(x>0){

Formula.literal\_frequency[2\*x-1]=0;

Formula.literal\_frequency[2\*x]=0;

}

else{

Formula.literal\_frequency[-2\*x]=0;

Formula.literal\_frequency[-2\*x-1]=0;

}

}

}

//显示每个文字的取值情况

void Literal\_Print(formula Formula)

{

for(int i=1;i<=Formula.literal\_count;i++)

{

if(Formula.literal\_tof[i]==1) printf("%d: True\n",i);

else if(Formula.literal\_tof[i]==0) printf("%d: False\n",i);

else printf("%d: True\n",i);

}

}

//显示当前数据的记录情况

void Formula\_Print(formula Formula)

{

for(int i=1;i<=Formula.literal\_count;i++)

{

printf("%d: %d\n",i,Formula.literal\_tof[i]);

printf(" ");

save\_node \*p1=Formula.save\_true[i].p;

save\_node \*p2=Formula.save\_false[i].p;

do{

if(p1 && p1->mark\_clause->mark && p1->node->mark){

printf("%d ",p1->mark\_clause->sequence);

}

if(p1)

p1=p1->next;

}while(p1);

printf("\n ");

do{

if(p2 &&p2->mark\_clause->mark && p2->node->mark){

printf("%d ",p2->mark\_clause->sequence);

}

if(p2)

p2=p2->next;

}while(p2);

printf("\n");

}

}

3.SAT problem.h

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<time.h>

#include"Graph & Formula.h"

//读取文件

status readfile(char \*FileName,formula &Formula,ALGraph &G)

{

FILE \*fp=fopen(FileName,"r");

char c,s[500];

while(1){

c=fgetc(fp);

if(c=='c')// 如果是注释

fgets(s,1000,fp);

else if(c == 'p'){

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

if(strcmp(s,"cnf")==0)

break;

else return ERROR;

}

else return ERROR;

}

fscanf(fp,"%d", &Formula.literal\_count);

fscanf(fp,"%d", &Formula.clause\_count);

//分配空间并初始化

FormulaInitial(Formula);

//从文件中读取

G.clause\_Gcount=Formula.clause\_count;

G.clause\_max=G.clause\_Gcount;

G.clause\_ini=G.clause\_Gcount;

InitialGraph(G);

for(int i=1;i<=Formula.clause\_count;i++){

int literal=1;

while(literal){

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

if(literal>0){

Formula.literal\_frequency[2\*literal-1]++;

AddVertex(G,i,literal);

}

else if(literal<0){

Formula.literal\_frequency[-literal\*2]++;

AddVertex(G,i,literal);

}

}

}

MatchFormula(G,Formula);

fclose(fp);

return OK;

}

//生成res文件

status makefile(formula Formula,char \*FileName)

{

int l=strlen(FileName),i,flag=0;

for(i=0;i<l;i++)

{

if(FileName[i]=='.' && FileName[i+1]=='c' && FileName[i+2]=='n' && FileName[i+3]=='f'){

flag=1;

break;

}

}

if(!flag) return ERROR;

FileName[i+1]='r',FileName[i+2]='e',FileName[i+3]='s';

FILE \*fp=fopen(FileName,"w");

if(Formula.ans==1){

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

for(int i=1;i<=Formula.literal\_count;i++)

if(Formula.literal\_tof[i]==1) fprintf(fp,"%d ",i);

else if(Formula.literal\_tof[i]==0) fprintf(fp,"%d ",-i);

fprintf(fp,"\nt %dms",Formula.time);

}

else if(Formula.ans==0){

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

fprintf(fp,"v\n");

fprintf(fp,"t %dms",Formula.time);

}

else fprintf(fp,"s -1\nv ");

fclose(fp);

return OK;

}

//删除子句中的永真式

void Delete\_Tautologies(ALGraph &G,formula &Formula)

{

for(int i=1;i<=Formula.literal\_count;i++)

{

save\_node \*p1,\*p2;

p1=Formula.save\_true[i].p;

if(!p1) continue;

do{

p2=Formula.save\_false[i].p;

if(!p2) break;

do{

if(p1->mark\_clause->sequence == p2->mark\_clause->sequence){

DeleteClause(G,Formula,p1->mark\_clause->sequence);

p1->way=0;

break;

}

p2=p2->next;

}while(p2);

p1=p1->next;

}while(p1);

}

}

//删除某一文字

void DeleteUnitartLiteral(ALGraph &G,formula &Formula,int literal)

{

save\_node \*p;

if(literal>0){

Formula.literal\_tof[literal]=1;

p=Formula.save\_true[literal].p;

do{

if(p->mark\_clause->mark){

DeleteClause(G,Formula,p->mark\_clause->sequence);

p->way=0;

}

p=p->next;

}while(p);

}

else{

Formula.literal\_tof[-literal]=0;

p=Formula.save\_false[-literal].p;

do{

if(p->mark\_clause->mark){

DeleteClause(G,Formula,p->mark\_clause->sequence);

p->way=0;

}

p=p->next;

}while(p);

}

DeleteLiteral(G,Formula,-literal);

}

//恢复某一文字

void RestoreLiteral(ALGraph &G,formula &Formula,int literal)//literal表示被删除的文字

{

save\_node \*p;

if(literal>0){

//恢复被删除的子句

Formula.literal\_tof[literal]=-1;

p=Formula.save\_true[literal].p;

do{

if(p->way==0){

int n=p->mark\_clause->sequence;//取该子句的序号

p->mark\_clause->mark=1;

p->way=-1;

G.clause\_Gcount++;

ArcNode \*p1=G.vertices[n].firstarc;

do{

if(p1->mark){//该文字本身未被删除

G.vertices[n].literal\_Gcount++;

}

p1=p1->nextarc;

}while(p1);

}

p=p->next;

}while(p);

//恢复被删除的文字的反

p=Formula.save\_false[literal].p;

if(!p) return;

do{

if(p->mark\_clause->mark && p->way==1){

p->mark\_clause->literal\_Gcount++;

p->way=-1;

p->node->mark=1;

}

p=p->next;

}while(p);

}

else{

//恢复被删除的子句

Formula.literal\_tof[-literal]=-1;

p=Formula.save\_false[-literal].p;

do{

if(p->way==0){

int n=p->mark\_clause->sequence;//取该子句的序号

p->mark\_clause->mark=1;

p->way=-1;

G.clause\_Gcount++;

ArcNode \*p1=G.vertices[n].firstarc;

do{

if(p1->mark){

G.vertices[n].literal\_Gcount++;

}

p1=p1->nextarc;

}while(p1);

}

p=p->next;

}while(p);

//恢复被删除的文字的反

p=Formula.save\_true[-literal].p;

if(!p) return;

do{

if(p->mark\_clause->mark && p->way==1){

p->mark\_clause->literal\_Gcount++;

p->way=-1;

p->node->mark=1;

}

p=p->next;

}while(p);

}

}

//判断是否有空子句

status JudgeEmptyClause(ALGraph G)

{

for(int i=1;i<=G.clause\_max;i++)

if(G.vertices[i].mark && !G.vertices[i].literal\_Gcount)

return TRUE;

return FALSE;

}

//寻找本次操作的文字

int SearchLiteral(ALGraph G,formula &Formula,int &flag)

{

int x;

for(int i=1;i<=G.clause\_max;i++)

{

if(G.vertices[i].mark && G.vertices[i].literal\_Gcount==1){

ArcNode \*p=G.vertices[i].firstarc;

do{

if(p->mark){

flag=1;

return p->literal;

}

p=p->nextarc;

}while(p);

}

}

for(int i=1;i<=2\*Formula.literal\_count;i++)

{

int y=Formula.literal\_sequence[i];

y=y>0?y:-y;

if(Formula.literal\_tof[y]==-1)

return Formula.literal\_sequence[i];

}

}

//DPLL

status DPLL(ALGraph &G,formula &Formula,int literal)

{

int flag=0;

if(literal){

DeleteUnitartLiteral(G,Formula,literal);

}

//GraphPrint(G);

//Formula\_Print(Formula);

if(JudgeEmptyClause(G)==TRUE) return FALSE;

if(!G.clause\_Gcount) return TRUE;

int x=SearchLiteral(G,Formula,flag);

if(DPLL(G,Formula,x)==TRUE) return TRUE;

else{

RestoreLiteral(G,Formula,x);

if(flag) return FALSE;

//GraphPrint(G);

//Formula\_Print(Formula);

if(DPLL(G,Formula,-x)==TRUE) return TRUE;

else{

RestoreLiteral(G,Formula,-x);

//GraphPrint(G);

//Formula\_Print(Formula);

return FALSE;

}

}

}

//解SAT问题

void SAT\_Solve(ALGraph &G,formula &Formula)

{

//删除重言式

clock\_t start,finish;

start=clock();//开始运行

Delete\_Tautologies(G,Formula);

if(DPLL(G,Formula,0)==TRUE){

printf("Satisfied!\n");

Formula.ans=1;

Literal\_Print(Formula);

}

else{

Formula.ans=0;

printf("Unsatisfied!\n");

}

finish=clock();//结束运行

Formula.time=(int)(1000\*(finish-start)/CLOCKS\_PER\_SEC);

for(int i=1;i<=Formula.literal\_count;i++)

if(Formula.literal\_tof[i]==-1) Formula.literal\_tof[i]=1;

printf("函数运行时间：%dms\n",Formula.time);

}

4.Suduku.h

#include"SAT problem.h"

int ChessMap[10][10],PlayMap[10][10],Hole[10][10],Value[10][10][10];

char SAVE[1005][90];

void PrintChessMap(int ChessMap[][10])

{

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

{

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

printf("%d ",ChessMap[i][j]);

printf("\n");

}

}

void ValuePrint(int Value[][10][10])

{

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

{

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

{

printf("%d %d:",i,j);

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

if(Value[i][j][x])

printf("%d ",x);

printf("\n");

}

}

}

status JudgeCorrect(int ChessMap[][10],int x,int n,int m)//n:行 m:列

{

//判断第n行

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

if(ChessMap[n][j] == x) return ERROR;

//判断第m列

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

if(ChessMap[i][m] == x) return ERROR;

//判断目标位置所在大方块

int p=(n-1)/3\*3+1,q=(m-1)/3\*3+1;

for(int i=p;i<p+3;i++)

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

if(ChessMap[i][j] == x) return ERROR;

return OK;

}

void CreateMap(int ChessMap[][10])

{

memset(ChessMap,0,sizeof(int)\*100);

int i,j,x;

srand((unsigned int)(time(NULL)));

for(int k=0;k<17;k++)

{

i=rand()%9+1;

j=rand()%9+1;

x=rand()%9+1;

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

ChessMap[i][j]=x;

else k--;

}

}

void PrintMap(int ChessMap[][10])

{

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

{

if(i==1 || i==4 || i==7){

printf("|");

for(int j=1;j<=20;j++) printf("-");

printf("|\n");

}

printf("|");

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

if(ChessMap[i][j])

printf("%d ",ChessMap[i][j]);

else printf("\_ ");

if(j==3 || j==6 || j==9) printf("|");

}

printf("\n");

}

printf("|");

for(int j=1;j<=20;j++) printf("-");

printf("|\n");

}

void ChessMapCpy(int ChessMap[][10],int ChessMap\_S[][10])

{

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

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

ChessMap\_S[i][j]=ChessMap[i][j];

}

void ReadLiteral(ALGraph &G,formula &Formula,int x)

{

int n=G.clause\_Gcount;

if(x){

AddVertex(G,n,x);

if(x>0)

Formula.literal\_frequency[2\*x-1]++;

else

Formula.literal\_frequency[-2\*x]++;

}

else{

G.clause\_Gcount++;

G.clause\_max++;

}

}

void LiteralMatch(formula &Formula,int ChessMap[][10],int Value[][10][10])//取值转化为文字

{

int sq=1;//文字的序号

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

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

if(!ChessMap[i][j])

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

if(JudgeCorrect(ChessMap,x,i,j)==OK)

Value[i][j][x]=sq++;

Formula.literal\_count=sq-1;

}

void UnitRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//单格取值范围

{

int sign[10];//用来记录某个数字在本格是否能填

memset(sign,0,sizeof(sign));

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

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

{

if(!ChessMap[i][j]){

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

if(JudgeCorrect(ChessMap,x,i,j))

sign[x]=1;

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

if(sign[x]){

for(int y=x+1;y<=9;y++)

if(sign[y]){

ReadLiteral(G,Formula,-Value[i][j][x]);

ReadLiteral(G,Formula,-Value[i][j][y]);

ReadLiteral(G,Formula,0);

}

sign[x]=0;

}

}

}

}

void RowRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//行取值范围

{

int sign[10];//记录数字在本行某列是否能填

int arr[10];//记录本行需要补的数

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

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

{

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;//标0表示本行不能补这个数

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

{

if(arr[x]){//x需要补

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))//x能够补进去

sign[j]=1;

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

if(sign[j]){

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

if(sign[h]){

ReadLiteral(G,Formula,-Value[i][j][x]);

ReadLiteral(G,Formula,-Value[i][h][x]);

ReadLiteral(G,Formula,0);

}

sign[j]=0;//复原

}

}

arr[x]=1;//复原

}

}

}

void LineRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//列取值范围

{

int sign[10];

int arr[10];

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

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

{

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;

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

{

if(arr[x]){

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

sign[i]=1;

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

{

if(sign[i]){

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

if(sign[h]){

ReadLiteral(G,Formula,-Value[i][j][x]);

ReadLiteral(G,Formula,-Value[h][j][x]);

ReadLiteral(G,Formula,0);

}

}

sign[i]=0;

}

}

arr[x]=1;

}

}

}

void BlockRange(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//块取值范围

{

int sign[10][10];

int arr[10];

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

for(int n=1;n<=9;n+=3)

for(int m=1;m<=9;m+=3)

{

for(int i=n;i<n+3;i++)

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;

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

{

if(arr[x]){

for(int i=n;i<n+3;i++)

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

sign[i][j]=1;

for(int i=n;i<n+3;i++)

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

if(sign[i][j]){

int i0=i,j0=j;

if(j0+1>m+2){

j0=m;i0++;

}

else j0++;

for(;i0<n+3;i0++)

{

for(;j0<m+3;j0++)

if(sign[i0][j0]){

ReadLiteral(G,Formula,-Value[i][j][x]);

ReadLiteral(G,Formula,-Value[i0][j0][x]);

ReadLiteral(G,Formula,0);

}

j0=m;

}

sign[i][j]=0;

}

}

arr[x]=1;

}

}

}

void UnitEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//单格可能取值

{

int sign[10];

memset(sign,0,sizeof(sign));

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

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

{

if(!ChessMap[i][j]){

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

if(JudgeCorrect(ChessMap,x,i,j))

sign[x]=1;

int flag=0;

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

{

if(sign[x]){

ReadLiteral(G,Formula,Value[i][j][x]);

flag=1;

}

sign[x]=0;

}

if(flag)

ReadLiteral(G,Formula,0);

}

}

}

void RowEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//行可能取值

{

int sign[10];

int arr[10];

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

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

{

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;

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

{

if(arr[x]){//x是本行要填的数

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

sign[j]=1;

int flag=0;

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

if(sign[j]){

ReadLiteral(G,Formula,Value[i][j][x]);

flag=1;

sign[j]=0;

}

if(flag)

ReadLiteral(G,Formula,0);

}

arr[x]=1;

}

}

}

void LineEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//列可能取值

{

int sign[10];

int arr[10];

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

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

{

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;

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

{

if(arr[x]){

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

sign[i]=1;

int flag=0;

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

{

if(sign[i]){

ReadLiteral(G,Formula,Value[i][j][x]);

flag=1;

sign[i]=0;

}

}

if(flag)

ReadLiteral(G,Formula,0);

}

arr[x]=1;

}

}

}

void BlockEvalue(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])//块可能取值

{

int sign[10][10];

int arr[10];

memset(sign,0,sizeof(sign));

for(int i=0;i<=9;i++) arr[i]=1;

for(int n=1;n<=9;n+=3)

{

for(int m=1;m<=9;m+=3)

{

for(int i=n;i<n+3;i++)

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

if(ChessMap[i][j])

arr[ChessMap[i][j]]=0;

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

{

if(arr[x]){

for(int i=n;i<n+3;i++)

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

if(!ChessMap[i][j] && JudgeCorrect(ChessMap,x,i,j))

sign[i][j]=1;

int flag=0;

for(int i=n;i<n+3;i++)

{

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

if(sign[i][j]){

ReadLiteral(G,Formula,Value[i][j][x]);

flag=1;

sign[i][j]=0;

}

}

if(flag)

ReadLiteral(G,Formula,0);

}

arr[x]=1;

}

}

}

}

void AnswerMatch(formula &Formula,int ChessMap[][10],int Value[][10][10])//将答案反映到棋盘

{

for(int k=1;k<=Formula.literal\_count;k++)

{

if(Formula.literal\_tof[k]==1)

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

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

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

if(Value[i][j][x]==k)

ChessMap[i][j]=x;

}

}

void Transform(ALGraph &G,formula &Formula,int ChessMap[][10],int Value[][10][10])

{

memset(Value,0,sizeof(int)\*1000);

LiteralMatch(Formula,ChessMap,Value);

G.clause\_Gcount=G.clause\_max=1;

G.clause\_ini=6000;

InitialGraph(G);

FormulaInitial(Formula);

UnitRange(G,Formula,ChessMap,Value);

RowRange(G,Formula,ChessMap,Value);

LineRange(G,Formula,ChessMap,Value);

BlockRange(G,Formula,ChessMap,Value);

UnitEvalue(G,Formula,ChessMap,Value);

RowEvalue(G,Formula,ChessMap,Value);

LineEvalue(G,Formula,ChessMap,Value);

BlockEvalue(G,Formula,ChessMap,Value);

G.clause\_Gcount--;

G.clause\_max--;

Formula.clause\_count=G.clause\_Gcount;

MatchFormula(G,Formula);

}

status JudgeUnique(int ChessMap[][10],int PlayMap[][10],int n,int m)//判断n行m列处是否有多解

{

ALGraph G;

formula Formula;

int value[10][10][10],num=0;

memset(value,0,sizeof(value));

LiteralMatch(Formula,PlayMap,value);

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

if(x!=ChessMap[n][m] && JudgeCorrect(PlayMap,x,n,m)){

int save=PlayMap[n][m];

PlayMap[n][m]=x;

Transform(G,Formula,PlayMap,value);

if(DPLL(G,Formula,0)==OK)

num++;

if(num>0){

PlayMap[n][m]=save;

DestroyGraph(G);

DestroyFormula(Formula);

return FALSE;

}

}

return TRUE;

}

status JudgeEmpty(int Hole[][10])

{

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

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

if(Hole[i][j]) return FALSE;

return TRUE;

}

int SearchHole(int PlayMap[][10],int Hole[][10])

{

int n,m;

srand((unsigned int)(time(NULL)));

n=rand()%9+1;

m=rand()%9+1;

if(JudgeEmpty(Hole)==TRUE) return 0;

while(!Hole[n][m]){

n=rand()%9+1;

m=rand()%9+1;

}

return n\*10+m;

}

status DigHole(int n,int m,int ChessMap[][10],int PlayMap[][10],int Hole[][10],int &num)

{

int hole\_s[10][10],save=PlayMap[n][m];

if(n==0 && m==0){

srand((unsigned int)(time(NULL)));

n=rand()%9+1;

m=rand()%9+1;

if(DigHole(n,m,ChessMap,PlayMap,Hole,num)==OK)//初始位置

return OK;

}

if(num==0) return OK;//全部挖完

if(JudgeUnique(ChessMap,PlayMap,n,m)==TRUE){

PlayMap[n][m]=0;

Hole[n][m]=0;

num--;

}

else return ERROR;

ChessMapCpy(Hole,hole\_s);

int x=SearchHole(PlayMap,Hole);//寻找下一个洞

while(x){

if(DigHole(x/10,x%10,ChessMap,PlayMap,Hole,num)==OK) return OK;

else{

Hole[x/10][x%10]=0;

x=SearchHole(PlayMap,Hole);

}

}

num++;

PlayMap[n][m]=save;

ChessMapCpy(hole\_s,Hole);

return ERROR;

}

status CreateUniqueMap(ALGraph &G,formula &Formula,int PlayMap[][10],int ChessMap[][10],int Hole[][10])

{

int flag=1;

while(flag){

DestroyGraph(G);

DestroyFormula(Formula);

memset(PlayMap,0,sizeof(int)\*100);

memset(ChessMap,0,sizeof(int)\*100);

memset(Value,0,sizeof(int)\*1000);

memset(Hole,0,sizeof(int)\*100);

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

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

Hole[i][j]=1;

CreateMap(ChessMap);

Transform(G,Formula,ChessMap,Value);

if(DPLL(G,Formula,0)==OK) flag=0;

}

AnswerMatch(Formula,ChessMap,Value);

ChessMapCpy(ChessMap,PlayMap);

int num;

printf("请输入游戏需填入数字的个数[1~64]:");

scanf("%d", &num);

while(num<1 || num>64){

printf("输入不符合规范，请重新输入!\n");

scanf("%d", &num);

}

DigHole(0,0,ChessMap,PlayMap,Hole,num);

DestroyGraph(G);

DestroyFormula(Formula);

return TRUE;

}

void ReadMap(int ChessMap[][10],char SAVE[][90])

{

int n,count=0;

FILE \*fp=fopen("suduku.txt","r");

for(int i=1;i<=1000;i++)

fscanf(fp,"%s",SAVE[i]);

fclose(fp);

printf("请选择要读入数独的序号:");

scanf("%d", &n);

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

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

{

if(SAVE[n][count]=='.')

ChessMap[i][j]=0;

else ChessMap[i][j]=SAVE[n][count]-'0';

count++;

}

}

void GamePlay(int PlayMap[][10],int ChessMap[][10])

{

int n,m,x,flag=1;

int num=0;

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

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

if(PlayMap[i][j])

num++;

int mark[10][10];

memset(mark,0,sizeof(mark));

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

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

if(PlayMap[i][j])

mark[i][j]=1;

int op=1;

while(num<=81 && op){

system("cls");

PrintMap(PlayMap);

printf("请选择操作:\n");

printf("1.填入数字\n");

printf("2.删除数字\n");

printf("3.提示\n");

printf("0.退出游戏\n");

scanf("%d", &op);

switch(op){

case 1:

printf("请选择行数，列数和填入的数字:");

scanf("%d%d%d", &n, &m, &x);

while((n<1||n>9) || (m<1||m>9) || (x<1 || x>9) || PlayMap[n][m] || JudgeCorrect(PlayMap,x,n,m)==ERROR){

printf("输入不符合规范，请重新输入!\n");

scanf("%d%d%d", &n, &m, &x);

}

PlayMap[n][m]=x;

num++;

break;

case 2:

printf("请选择行数，列数:");

scanf("%d%d", &n, &m);

while((n<1||n>9) || (m<1||m>9) || !ChessMap[n][m] || mark[n][m])

{

printf("输入不符合规范，请重新输入!\n");

scanf("%d%d%d", &n, &m, &x);

}

PlayMap[n][m]=0;

num--;

break;

case 3:

printf("请输入行数和列数:");

int i,j;

scanf("%d%d", &i, &j);

printf("%d",ChessMap[i][j]);

break;

case 0:

break;

}

if(!op) break;

getchar();getchar();

}

PrintMap(PlayMap);

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

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

if(PlayMap[i][j]!=ChessMap[i][j]){

flag=0;

break;

}

if(flag==0){

printf("结果错误！正解如下:\n");

PrintMap(ChessMap);

}

else printf("结果正确！");

}

void Game(ALGraph &G,formula &Formula,int PlayMap[][10],int ChessMap[][10],int Value[][10][10])

{

int op=1,i,j;

int flag=0;

while(op){

memset(PlayMap,0,sizeof(PlayMap));

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

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

printf(" 1.ReadMap\n");

printf(" 2.CreateMap\n");

printf(" 3.GameStart\n");

printf(" 4.PresentAnswer\n");

printf(" 0.exit\n");

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

printf("请输入功能序号:");

scanf("%d", &op);

switch(op){

case 1:

memset(PlayMap,0,sizeof(int)\*100);

memset(ChessMap,0,sizeof(int)\*100);

memset(Value,0,sizeof(int)\*1000);

ReadMap(ChessMap,SAVE);

flag=1;

PrintMap(ChessMap);

ChessMapCpy(ChessMap,PlayMap);

Transform(G,Formula,ChessMap,Value);

if(DPLL(G,Formula,0)){

AnswerMatch(Formula,ChessMap,Value);

}

printf("按任意键继续……");

break;

case 2:

if(CreateUniqueMap(G,Formula,PlayMap,ChessMap,Hole)==OK){

flag=1;

printf("创建棋盘成功!\n");

}

else printf("创建棋盘失败!\n");

printf("按任意键继续……");

break;

case 3:

if(flag==0){

printf("还未创建棋盘!\n");

printf("按任意键继续……");

break;

}

if(JudgeEmpty(PlayMap)==FALSE)

GamePlay(PlayMap,ChessMap);

flag=0;

break;

case 4:

PrintMap(ChessMap);

printf("按任意键继续……");

break;

case 0:

break;

default:

printf("Please input number[0~5]\n");

}

if(!op) break;

getchar();getchar();

}

}

void MakeCnfFile(ALGraph G,formula Formula)

{

FILE \*fp=fopen("123.cnf","w");

fprintf(fp,"p cnf %d %d\n",Formula.literal\_count,G.clause\_Gcount);

for(int i=1;i<=G.clause\_max;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

do{

fprintf(fp,"%d ",p->literal);

p=p->nextarc;

}while(p);

fprintf(fp,"0\n");

}

fclose(fp);

}

5.Menu.h

void Menu1(void);

void Menu1\_1(void);

void Menu1(void)

{

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

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

printf(" 1.SAT\n");

printf(" 2.Sudoku\n");

printf(" 0.exit\n");

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

printf("请输入功能序号:");

}

void Menu1\_1(void)

{

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

printf("Menu for SAT Problem Functions!\n");

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

printf(" 1.ReadFile\n");

printf(" 2.DisplayClauses\n");

printf(" 3.DPLL\n");

printf(" 0.exit\n");

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

printf("请输入功能序号:");

}

void Menu1\_2(void)

{

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

printf("Menu for SuDuku Functions!\n");

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

printf(" 1.CreateMap\n");

printf(" 2.PrintMap\n");

printf(" 3.FillMap\n");

printf(" 4.PlayGame\n");

printf(" 0.exit\n");

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

printf("请输入功能序号:");

}

void Menu3(void)

{

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

printf("Menu for ReadFile Functions!\n");

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

printf(" 0.From the test set\n");

printf(" 1.From the current folder\n");

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

printf("请输入功能序号:");

}

6.SAT.cpp

#include"Suduku.h"

#include"Menu.h"

ALGraph G;

formula Formula;

char FileName[100],s[100];

int main()

{

int op=1,op1,flag=0;

while(op){

Menu1();

scanf("%d", &op);

switch(op){

case 0:

break;

case 1:

op1=1;

while(op1){

Menu1\_1();

scanf("%d", &op1);

switch(op1){

case 0:

break;

case 1:

Menu3();

scanf("%d", &flag);

if(flag){

printf("请输入文件名:");

scanf("%s",s);

strcpy(FileName,s);

}

else{

printf("请输入文件名:");

scanf("%s", s);

strcpy(FileName,"助教课设演示检查要求\\");

strcat(FileName,s);

}

if(readfile(FileName,Formula,G)==OK)

printf("读入文件成功!\n");

else printf("读入文件失败!\n");

printf("按任意键继续……");

break;

case 2:

GraphPrint(G);

printf("打印图成功!\n");

printf("按任意键继续……");

break;

case 3:

SAT\_Solve(G,Formula);

makefile(Formula,FileName);

DestroyGraph(G);

DestroyFormula(Formula);

printf("按任意键继续……");

break;

default:

printf("Please input number[0~3]\n");

}

if(!op1) break;

getchar();getchar();

}

break;

case 2:

op1=1;

while(op1){

Menu1\_2();

int flag1=0;

scanf("%d", &op1);

switch(op1){

case 1:

if(CreateUniqueMap(G,Formula,PlayMap,ChessMap,Hole)==OK){

printf("创建棋盘成功!\n");

flag1=1;

}

else printf("创建失败!\n");

printf("按任意键继续……");

break;

case 2:

PrintMap(PlayMap);

printf("打印棋盘成功!\n");

printf("按任意键继续……");

break;

case 3:

if(!flag1){

printf("还未创建棋盘!\n");

printf("按任意键继续……");

break;

}

Transform(G,Formula,ChessMap,Value);

MakeCnfFile(G,Formula);

if(DPLL(G,Formula,0)){

AnswerMatch(Formula,ChessMap,Value);

}

ChessMapCpy(ChessMap,PlayMap);

printf("按任意键继续……");

break;

case 4:

Game(G,Formula,PlayMap,ChessMap,Value);

printf("按任意键继续……");

break;

case 0:

break;

default:

printf("Please input number[0~4]\n");

}

if(!op1) break;

getchar();getchar();

}

break;

default:

printf("Please input number[0~2]\n");

}

if(!op1) continue;

if(!op) break;

getchar();getchar();

}

}