

**课程设计报告**

**题目：基于SAT的二进制数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS1903**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2021/3/16**

**计算机科学与技术学院**

# 任 务 书

# 设计内容

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于DPLL算法实现一个完备SAT求解器，对输入的CNF范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

# 设计要求

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)
2. **公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)
4. **时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
5. **程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)
6. **SAT应用：**将二进制数独游戏[5，6]问题转化为SAT问题[6]，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-9]。(15%)

# 参考文献

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

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

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

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

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

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

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

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

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

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

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

目 录

[**任 务 书** I](#_Toc67781521)

[设计内容 I](#_Toc67781522)

[设计要求 I](#_Toc67781523)

[参考文献 II](#_Toc67781524)

[**1 引言** 1](#_Toc67781525)

[1.1 课题背景与意义 1](#_Toc67781526)

[1.1.1数独游戏发展背景 1](#_Toc67781527)

[1.1.2 SAT问题简介 1](#_Toc67781528)

[1.2 国内外研究现状 1](#_Toc67781529)

[1.3 课程设计的主要研究工作 2](#_Toc67781530)

[**2系统需求分析与总体设计** 3](#_Toc67781531)

[2.1 系统需求分析 3](#_Toc67781532)

[2.2系统总体设计 3](#_Toc67781533)

[**3系统详细设计** 5](#_Toc67781534)

[3.1有关数据结构的定义 5](#_Toc67781535)

[3.2函数定义及其功能 6](#_Toc67781536)

[3.3模块算法设计 10](#_Toc67781537)

[**4系统实现与测试** 12](#_Toc67781538)

[4.1系统实现 12](#_Toc67781539)

[4.1.1 程序实现环境 12](#_Toc67781540)

[4.1.2 头文件及预定义说明 12](#_Toc67781541)

[4.2系统测试 13](#_Toc67781542)

[4.2.1 系统使用说明 13](#_Toc67781543)

[4.2.2 系统测试 13](#_Toc67781544)

[**5总结与展望** 21](#_Toc67781545)

[5.1全文总结 21](#_Toc67781546)

[5.2工作展望 21](#_Toc67781547)

[**6实验体会** 23](#_Toc67781548)

[参考文献 24](#_Toc67781549)

[附录 课程设计源程序 25](#_Toc67781550)

[main.cpp 25](#_Toc67781551)

[func.cpp 28](#_Toc67781552)

[sudoku.cpp 47](#_Toc67781553)

[func.h 66](#_Toc67781554)

[def.h 69](#_Toc67781555)

# 1 引言

## 1.1 课题背景与意义

命题逻辑公式的可满足性问题（SAT）是数理逻辑、计算机科学、集成电路设计和人工智能等领域中的核心问题，并且是第一个被证明出来的NP问题。SAT问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法意义重大。但目前并不存在一种求解算法在最坏情况下的时间复杂度是多项式级别，其求解速度仍是制约SAT算法发展的一大难题。

### 1.1.1数独游戏发展背景

从1960年至今，SAT问题一直广受关注，世界各国研究人员为了这个问题做出了许多努力，提出了许多求解算法。每年都有国际会议组织一次SAT竞赛以求找到一组最快的SAT求解器。尽管命题逻辑的可满足性问题理论研究已趋于成熟，但在SAT求解器应用逐渐广泛的今天，探寻SAT高效求解算法仍然是一个极具挑战的研究方向。

### 1.1.2 SAT问题简介

SAT的应用领域非常广泛，能用来解决旅行商（TSP）、约束满足问题（CSP）、语义信息的处理等问题；SAT亦可以解决许多实际问题如人工智能、数据库检索、图着色等问题。

## 1.2 国内外研究现状

Bart Selman和Henry Kautz分别于1997年和2003年在人工智能第五届国际合作会议上提出了SAT问题面临的十大挑战性问题，并在2001年和2007年先后对当时的可满足行问题现状进行了全面的阐述和总结。

最经典的求解SAT问题的完备算法是DPLL算法，它是由Davis和Putnam等人再1960年提出的。而后，S.A.Cook在1971年证明了SAT问题是NP完全问题。在1996年后，很多国家都相继举办了一些SAT竞赛和研讨会，使得SAT问题更受关注，这段时间内出现了诸如MINISAT、SATO等众多新颖高效的算法。

国内也涌现了许多高效的求解算法，如1998年梁东敏提出的改进的子句加权WAST算法，还有2000年金人超和黄文奇提出的并行Solar 算法，以及2002 年张德富提出的模拟退火算法。

## 1.3 课程设计的主要研究工作

核心在于编写出一个基于SAT的二进制数独游戏求解程序。首先设计基于DPLL的算法与程序框架，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；然后在此算法的基础上进行程序的改进优化，使得SAT求解器的效率更高；再实现二进制数独游戏归约为SAT的功能，将游戏转化为CNF以应用SAT求解器；最后组装函数功能，完善程序界面，实现一个有可玩性的功能模块。

# 2系统需求分析与总体设计

## 2.1 系统需求分析

本系统需要精心设计变元、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，以完成功能：1）输入输出功能；2）公式解析与验证；3）DPLL求解SAT算例；4）时间性能的测量；5）程序优化；6）SAT问题在二进制数独的应用。

# 2.2系统总体设计

求解系统有一共七个选项：

1. 初始化存储结构，将用于存储CNF的邻接链表、用于存储结果的链表、用于存储分裂策略选择变元的链表初始化，并且显示是否初始化成功；
2. 读取四阶数独并输出CNF文件，输入数独存储文件名，进行读取，存储在二维数组中，并且根据存储的二维数组创建CNF文件，以同名CNF保存文件；
3. 读取六阶数独并输出CNF文件，功能与（2）相同；
4. 读取CNF文件，用户输入文件名，根据文件名将对应CNF文件中内容存储在邻接链表中，同时创建分裂策略变元优先选择表；
5. DPLL运算，对于邻接链表中存储的CNF进行DPLL运算，若是能够成功运算，则显示运算成功并且显示结果链表。若是运算失败则显示运算失败；
6. 优化后的DPLL运算；
7. 输出res文件，将结果以要求形式输出；
8. 输出数独答案；
9. 退出系统。

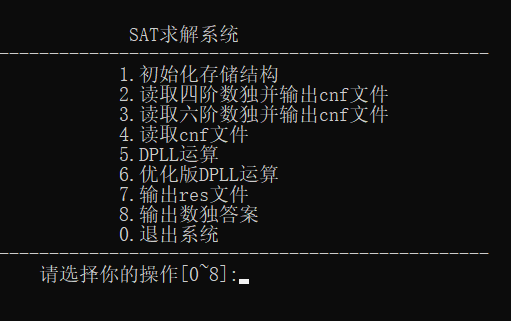


图2-2-1 系统功能模块图

# 3系统详细设计

# 3.1有关数据结构的定义

表3-1-1 文字结点结构

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | int | status | LitNode\* |
| 数据名称 | litseq | flag | next |
| 数据作用 | 文字序号 | 文字状态：是否被删除 | 文字结点指针 |

表3-1-2 子句结点结构

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | int | status | int |
| 数据名称 | data | flag | flagchange |
| 数据作用 | 子句序号 | 子句状态:是否被删除 | 记录子句因哪个文字删除 |
|  | | | |
| 数据类型 | int | LitNode\* | ClaNode\* |
| 数据名称 | lnum | firstlit | next |
| 数据作用 | 子句中文字数目 | 子句中第一个文字结点的指针 | 子句结点指针 |

表3-1-3 CNF结构

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | int | int | ClaNode\* |
| 数据名称 | clanum | litnum | CNFhead |
| 数据作用 | 子句数目 | 文字数目 | 头结点指针且其firstlit  用于存放最终赋值结果 |

表3-1-4 结果存储链表结构

|  |  |  |
| --- | --- | --- |
| 数据类型 | int | ResNode\* |
| 数据名称 | data | next |
| 数据作用 | 已消除文字序号 | 结果文字结点指针 |

表3-1-5 文字权重存储链表结点结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据类型 | int | status | double | MOMNode\* |
| 数据名称 | lit | flag | data | next |
| 数据作用 | 文字序号 | 文字状态:是否已经被使用 | 文字权重 | 结点指针 |

表3-1-6 分裂策略选取文字存储链表结点结构

|  |  |  |
| --- | --- | --- |
| 数据类型 | int | DivideNode\* |
| 数据名称 | data | next |
| 数据作用 | 分裂策略选取文字序号 | 结点指针 |

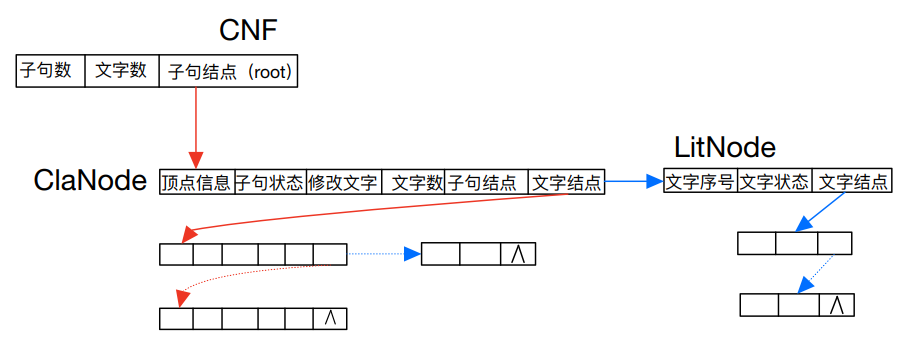


图3-1-1 CNF存储数据结构

## 3.2函数定义及其功能

（1）初始化操作：

1. status InitCNF(CNF &S)；

2. status InitResultList(ResList &L)；

3. status InitDivideList(DivideList &DL)；

4. status InitMOMList(MOMList &ML,int litnum)；

（2）CNF操作函数：

1. status ShowCNF(CNF S)：循环输出CNF中存储数据，在调试工程时用于检测文件读写是否成功；

2. status LoadCNF(CNF &S, char FileName[])：先循环读取字符串，直到读取到p和cnf，以此消去文件开头注释；然后读取文件中的文字数和子句数；接着根据子句数，循环读入每行文字内容，以此达到CNF文件到存储结构的转换。

（3）MOM链表操作函数：

1. MOMList LocateMOMNode(MOMList ML,int lit)：利用while循环，找到链表中存储文字lit的结点，并返回指针；

2.status MOMListAssign(MOMList &ML,CNF S)：读取CNF结构，每次读取到文字x，就利用利用LocateMOMNode 找到该文字结点，利用MOM算法进行文字加权赋值；

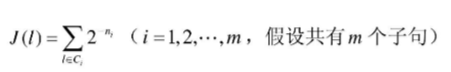


图3-2-1 MOM加权算法

3. status LiteraltoMOM(MOMList &ML,int lit)：在消除文字时，将MOM链表中目标文字设为已消除状态；

4. status LiteraloffMOM(MOMList &ML,int lit)：在回溯过程时，将MOM链表中目标文字设为未清除状态；

5. status MOMListTraverse(MOMList ML)：遍历MOM链表，在调试时测试是否有误；

（4）ResList结果链表操作函数：

1. status InsertResNode(ResList &L,int lit)：将需消除文字入栈；

2. int RecallResList(ResList &L)：回溯过程中将文字出栈，并返回出栈的文字序号；

3. status ResListTraverse(ResList L)：遍历结果链表，调试时检测算法是否正确；

（5）DivideList分裂策略选取链表操作函数：（用于非递归算法）

1. status PushDivideNode(DivideList &L,int lit)：将分裂策略选取的文字入栈；

2. int PopDivideList(DivideList &L)：将链表末尾文字出栈；

3. status DivideListTraverse(DivideList L)：遍历链表，调试时检测算法是否正确；

（6）CNF子句文字操作函数：

1. ClaLink LocateClause(CNF S,int cla)：定位目标子句序号的子句结点，返回其指针；

2. status IsUnitClause(ClaLink C)：判断目标子句是否为单子句，是则返回TRUE，否则返回FALSE；

3. int LocateUnitClause(CNF S)：查找并定位单子句,查找成功便返回子句序号，失败则返回ERROR；

4. status DestroyClause(CNF &S,int cla,int lit)：目标子句flag设为0，并标记该子句是因为文字lit而消除；

5. status ExistNonClause(CNF S)：判断是否存在空子句；

6. status AddClausetoLast(CNF &S,int lit)：对文字lit创建单子句，加入CNF；

7. status RemoveLastClause(CNF &S)：回溯过程中，将CNF中最末尾子句删除；

8. int FindFirstLiteral(ClaLink L)：查找子句L结点中第一个未删除文字；

9.status ExistClause(CNF S)：判断CNF是否存在子句；

10. status ClearLiteral(CNF &S,int lit)：单子句规则实现，传入文字lit，对包含lit的子句调用DestroyClause；将-lit文字状态置0，且将该子句文字数目减一；

（7）分裂策略函数：

1. int SimpleDivideStrategy(ResList L,CNF S)：选取第一个未在ResList中出现的文字作为分裂策略；

2. int MOMDivideStrategy(ResList L,CNF S,MOMList ML)：选取MOMList中第一个ResList中未出现过的文字作为分裂策略；

（8）回溯操作函数：

1. status RecallCNF(CNF &S,int lit)：将目标文字lit恢复，因为lit而被删除的子句状态置1，-lit文字状态置1且带有-lit文字的子句的文字数加一；

2. status Backtracking(CNF &S,ResList &L,int lit)：对ResList中文字lit及之后所有文字调用RecallResList、RecallCNF，并移除加入的单子句；

3. int BacktrackingbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL)：利用PopDivideList找到上一个分裂点lit，对ResList中文字lit及之后所有文字调用RecallResList、RecallCNF、LiteraloffMOM，并且返回lit；

（9）DPLL求解函数：

1. status DPLL(CNF &S,ResList &L)：详见3.3（3）中介绍；

2. status DPLLbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL)：详见3.3（4）中介绍；

（10）排序算法函数：

1. void BubbleSortR(ResList &List)：ResList升序冒泡排序，以便输出结果；

2. void BubbleSortM(MOMList &List)：MOMList升序冒泡排序；

（11）结果输出函数：

1. status ResultPrint(ResList L,int satisfy)；

2. int ResWrite(int satisfy,double time,ResList L,char filename[])：详见3.3（5）；

（12）数独函数：

1. status LoadSudoku4(int sudoku4[][4], char FileName[])：将文件中四阶数独存储进二维数组；

2. status LoadSudoku6(int sudoku6[][6], char FileName[])：六阶数组存储；

3. status Sudoku4toCNF(int sudoku4[][4],char FileName[])：四维数独的二维数组形式输出CNF文件，详见3.3（2）；

4. status Sudoku6toCNF(int sudoku6[][6],char FileName[])：六维数独输出CNF文件；

5. status PuzzleResult(ResList L,int kind)：数独结果输出函数，详见3.3（6）。

## 3.3模块算法设计

（1）读取CNF文件：使用LoadCNF函数将目标文件存储至CNF结构中。

（2）读取数独并输出CNF文件：Ⅰ）统计已存在的数独值，因为约束一二三条件增加的子句数是一定的，所以可以得出一共有多少子句；Ⅱ）按照CNF文件格式，输出第一排内容；Ⅲ）根据已存在的数独值，加入单子句约束条件；Ⅳ）分别循环加入约束一、二、三形成的子句，完成数独形式到CNF文件的转换。

（3）DPLL运算：Ⅰ）调用LocateUnitClause查找是否有单子句存在，若有则进行单子句化简，将化简变元x入栈于ResList中，将CNF结构中带有变元x的子句状态置0，并且标记因为变元x而改变，将CNF结构中带有变元-x的文字状态置0；Ⅱ）判断是否还存在子句以及是否存在空子句；Ⅲ）若无单子句存在，则运用分裂策略选择出一个变元m（按照文字序号从小到大，选取第一个没有在ResList中出现过的变元）；Ⅳ）用变元m创造出单子句加入CNF，进行第一分支递归运算；Ⅴ）若第一分支运算失败，则进行回溯，将ResList中m及之后的变元出栈，逐个恢复标记；Ⅵ）反转变元再生成单子句加入CNF，进行第二分支的递归运算；Ⅶ）最终返回FALSE，则运算失败，无解，返回OK，则生成解，存储于ResList中。



图3-3-1 DPLL算法流程图

（4）优化版DPLL运算（非递归算法以及更优的分裂策略）：Ⅰ）在还有子句存在情况下一直循环；Ⅱ）判断是否有单子句，有则循环进行单子句规则化简；Ⅲ）无单子句时则根据分裂策略选取变元（策略采用MOM算法-最短子句出现频率最大优先进行加权排序，在CNF中出现次数越多权重越大），带入CNF进行消除，再度循环；Ⅳ）存在空子句时，判断结果链表是否已经回溯至无文字存储，若链表非空，则反转变元带入CNF消除，再度循环；Ⅴ）若是结果链表已空，则返回FALSE，说明CNF无解；Ⅵ）若是循环至无子句存在，则函数返回OK，说明SAT已求解完毕，结果存入RseList中。



图3-3-2 优化后DPLL算法流程图

（5）输出res文件：Ⅰ）读入CNF文件名称，修改后缀改为res；Ⅱ）将ResList进行升序冒泡排序，然后循环输出结果，根据题目要求，以正确形式输出。

（6）输出数独答案：根据ResList中前16或者36个文字正负，循环输出1或0，显示出数独的一个解。

# 4系统实现与测试

## 4.1系统实现

## 4.1.1 程序实现环境

WIN10系统下使用CodeBlocks编辑，编程语言为C语言。

## 4.1.2 头文件及预定义说明

（1）预定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status; //状态返回值

typedef struct LitNode{ //文字结点类型定义

int litseq; //顶点位置文字序号

status flag; //文字状态

LitNode \*next; //下一个文字结点指针

} LitNode,\*LitLink;

typedef struct ClaNode{ //子句结点类型定义

int data; //顶点信息

status flag; //子句状态

int flagchange; //用于记录子句状态被哪个文字修改

int lnum; //统计子句中文字数目

LitNode \*firstlit; //指向第一个文字

ClaNode \*next; //指向下一个子句

} ClaNode,\*ClaLink; //CNF头结点以及结点指针

typedef struct { //CNF头结点的类型定义

ClaLink CNFhead; //头结点指针且其firstlit用于存放最终赋值结果

int clanum,litnum; //子句数、文字数

} CNF;

typedef struct ResNode{ //结果存储链表

int data;

ResNode \*next;

}ResNode,\*ResList;

typedef struct MOMNode{ //MOM计算链表

int lit;

status flag;

double data;

MOMNode \*next;

}MOMNode,\*MOMList;

typedef struct DivideNode{ //分裂策略存储链表栈

int data;

DivideNode \*next;

}DivideNode,\*DivideList;

（2）头文件：

1.func.h：包含所需函数的定义，在main.c中被调用；

2.def.h：包含所有预定义，在func.h中被调用。

## 4.2系统测试

## 4.2.1 系统使用说明

（1）输入1，进行初始化操作；

（2）若要进行SAT求解，则直接输入4并回车，再输入CNF文件名称；

（3）若要进行数独求解操作，则先输入2或者3，再输入四阶或者六阶数独文件名，系统将自动输出同名cnf文件；

（4）输入5或者6以进行DPLL或者优化后的DPLL运算，运算完成后将会输出是否成功运算；

（5）输入7以输出符合规范的res文件；

（6）若求解成功，可以输入8以获得已求解的数独答案；

（7）输入0以退出系统。

## 4.2.2 系统测试

选择测试一个文字数50的SAT算例和一个文字数265的SAT算例，以及一个简单难度的二进制数独和一个中等难度的二进制数独。

表4-2-1 SAT测试1

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 输入1初始化 | 初始化成功 |  |
| 输入4及文件名进行读取 | 创建成功 |  |
| 输入5进行DPLL运算并输入7输出答案 | 运算成功 |  |
| 输入6进行优化DPLL运算并输入7输出答案 | 运算成功 |  |
| 放入verify进行验证 | 满足条件 |  |

表4-2-2 SAT测试2

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 输入1初始化 | 初始化成功 |  |
| 输入4及文件名进行读取 | 创建成功 |  |
| 输入5进行DPLL运算并输入7输出答案 | 运算成功 |  |
| 输入6进行优化DPLL运算并输入7输出答案 | 运算成功 |  |
| 放入verify文件验证 | 满足 |  |

表4-2-3 二进制数独测试1

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 输入1初始化 | 初始化成功 |  |
| 输入2及文件名进行读取 | 输出数独 |  |
| 输入4及文件名进行读取 | 创建成功 |  |
| 输入6进行DPLL运算 | 运算成功 |  |
| 输入8以输出答案 | 输出答案 |  |

表4-2-4 二进制数独测试2

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 输入1初始化 | 初始化成功 |  |
| 输入3及文件名进行读取 | 输出数独 |  |
| 输入4及文件名进行读取 | 创建成功 |  |
| 输入6进行DPLL运算 | 运算成功 |  |
| 输入8以输出答案 | 输出答案 |  |

# 5总结与展望

## 5.1全文总结

本次完成程序设计后，对自己的工作做如下总结：

1. 查阅相关资料，初步了解SAT问题，学习SAT求解的算法策略；
2. 设计CNF文件读取功能，选择合适的数据结构，实现CNF到存储结构的转换功能；
3. 基于最基础的DPLL算法，设计SAT求解器并且利用DPLL算法求解基础的SAT算例，进行调试；
4. 实现最基础的DPLL算法之后进行算法的优化，针对分裂策略进行设计并且改用非递归算法进行实现；
5. 设计数独问题到CNF文件的转换函数，将二进制数独问题转化为SAT问题并且能够利用所设计的DPLL算法进行求解，再设计函数输出得到的数独解；
6. 搭建交互式系统，将函数模块化，使系统能够执行用户输入的功能。

## 5.2工作展望

虽然优化后DPLL算法性能有所改善，但是仍然有很大的提高空间，在今后的研究中，可以围绕着如下几个方面进行改进：

1. 添加冲突分析、子句学习过程，可以找出需要回溯的原因并进行规避，在冲突产生的时候，记录下产生冲突的子句，以此，在冲突分析之后可以得知精确的回溯目标层级，在决策树中可以回溯多个层级，而非单纯的回溯到父节点；
2. 分裂策略也可以根据不同的SAT问题进行选择，不难发现本次实验中采用的最短子句出现频率最大优先策略，在给到的算例中区分度并不大，或许尝试资料中所给的独立变量状态衰减和策略或者最短正子句优先策略，能够有更好的决策效果；
3. 在本次实验中尝试了利用非递归实现DPLL算法，但是发现计算时间能够减小与是否使用非递归关系并不大，更关键的在于分裂策略的选取，非递归只是在空间上进行了节省；
4. 或许可以利用分治算法，将计算拆分，以此加快计算速度，使得求解器能够计算更大的算例。

# 6实验体会

本次课程设计实验完成的较为仓促，若是假期内花更多时间来研究学习，或许完成度会更高。

在这次程序设计实验之前，对于程序设计的理解只停留在模块上，所学的不同数据结构的知识是分立的，但是在这次设计之后，提升了能够将不同数据结构结合起来的能力，对于数据结构的理解也更深了一层。

其次，面对一个崭新的课题，当看到设计书时，有些不知所措，没有经验，但是通过仔细研读，查阅相关资料，慢慢理解了设计要点，一点点完成了课程设计。以后在面对一个大型的课题时，也有了相关的经验，不至于无从下手了。

在进行求解器实现时，有几个主要的问题，在初期造成了较大的影响，在解决后也使得自己的编程能力有所提升。

首先是DPLL算法的核心的理解，一开始并没有充分理解DPLL的递归的实现方式，回溯过程总是会有错误产生。在反复debug的过程中，产生了更深的理解，也最终完成了递归函数的编写。虽然花了很多的时间，但是完成了编写后也获得了极大的鼓舞，面对后面的错误也能静下心来慢慢试错。

其次，在进行非递归算法编写的时候，判断条件的错误编写导致出现了很多bug，对心态产生了极大的影响，但是最后也通过慢慢的试错，对照结果检查，发现了问题所在，并且实现了功能。

最后，在数独部分，如何将数独转化为CNF文件这个问题也困扰了自己很久，关键在于文字数和子句数的计算和需要新增的文字序号如何选择，在反复学习设计书上的相关内容、草稿纸上仔细演算后才终于理解，也是对逻辑思维的一次考验。

完成整个系统的编写后，初识了数据结构在实际问题中的应用，提升了自己对整体化的项目的处理能力，受益匪浅。

# 参考文献

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

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

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

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

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

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

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

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

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

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

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

# 附录 课程设计源程序

# main.cpp

#include"func.h"

int main()

{

int op=1;

char filename[20]; //cnf文件名存储

char sudokuname[20]="sudoku2.txt"; //数读文件名存储

char sudokucnf[20]="sudoku2.cnf";

CNF S; //储存cnf

DivideList DL; //分裂结点链表栈

ResList RL; //结果储存

MOMList ML; //MOM链表

int sudoku4[4][4]; //记录四阶数独

int sudoku6[6][6]; //记录六阶数独

int sudokind=0; //记录选择数独种类

int satisfy; //DPLL是否运算成功

clock\_t start\_t, end\_t;

double total\_t;

while(op){

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

printf(" SAT求解系统\n");

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

printf(" 1.初始化存储结构\n");

printf(" 2.读取四阶数独并输出cnf文件\n");

printf(" 3.读取六阶数独并输出cnf文件\n");

printf(" 4.读取cnf文件\n");

printf(" 5.DPLL运算\n");

printf(" 6.优化版DPLL运算\n");

printf(" 7.输出res文件\n");

printf(" 8.输出数独答案\n");

printf(" 0.退出系统\n");

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

printf(" 请选择你的操作[0~8]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitCNF(S)&&InitResultList(RL)&&InitDivideList(DL))

printf("初始化成功\n");

else

printf("初始化失败\n");

getchar();getchar();

break;

case 2:

sudokind=4;

printf("请输入数独文件名\n");

scanf("%s",sudokuname);

LoadSudoku4(sudoku4,sudokuname);

PrintSudodu4(sudoku4);

Sudoku4toCNF(sudoku4,sudokuname);

getchar();getchar();

break;

case 3:

sudokind=6;

printf("请输入数独文件名\n");

scanf("%s",sudokuname);

LoadSudoku6(sudoku6,sudokuname);

PrintSudodu6(sudoku6);

Sudoku6toCNF(sudoku6,sudokuname);

getchar();getchar();

break;

case 4:

printf("请输入文件名\n");

scanf("%s",filename);

LoadCNF(S,filename);

InitMOMList(ML,S.litnum);

MOMListAssign(ML,S);

BubbleSortM(ML);

printf("创建成功\n");

getchar();getchar();

break;

case 5:

start\_t = clock();

satisfy=DPLL(S,RL);

end\_t = clock();

if(satisfy)

{

printf("运算成功\n");

ResultPrint(RL,satisfy);

}

else

printf("运算失败\n");

total\_t = ((double)(end\_t - start\_t))/ CLOCKS\_PER\_SEC\*1000;

printf("%f ms",total\_t);

getchar();getchar();

break;

case 6:

start\_t = clock();

//satisfy=DPLLbyLoop(S,RL,ML,DL);

satisfy=MOMDPLL(S,RL,ML);

end\_t = clock();

if(satisfy)

{

printf("运算成功\n");

ResultPrint(RL,satisfy);

}

else

printf("运算失败\n");

total\_t = ((double)(end\_t - start\_t))/ CLOCKS\_PER\_SEC\*1000;

printf("%f ms",total\_t);

getchar();getchar();

break;

case 7:

if(ResWrite(satisfy,total\_t,RL,filename)==OK)

printf("结果写入成功\n");

else printf("结果写入失败\n");

getchar();getchar();

break;

case 8:

PuzzleResult(RL,sudokind);

getchar();getchar();

break;

case 0:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

# func.cpp

#include "func.h"

/////////////////////////初始化操作//////////////////////////////

status InitCNF(CNF &S)

{

S.clanum=0;

S.litnum=0;

S.CNFhead=(ClaNode\*)malloc(sizeof(ClaNode));

return OK;

}

status InitResultList(ResList &L)

{

L=(ResNode\*)malloc(sizeof(ResNode));

L->data=0;

L->next=NULL;

return OK;

}

status InitDivideList(DivideList &DL)

{

DL=(DivideNode\*)malloc(sizeof(DivideNode));

DL->data=0;

DL->next=NULL;

return OK;

}

status InitMOMList(MOMList &ML,int litnum) //MOM初始化

{

ML=(MOMNode\*)malloc(sizeof(MOMNode));

ML->lit=0;

ML->data=0; //头结点初始化

MOMList p=ML,q;

int i;

for(i=1;i<=litnum;i++)

{

q=(MOMNode\*)malloc(sizeof(MOMNode));

q->lit=i;

q->flag=1;

q->data=0;

p->next=q;

p=q;

}

p->next=NULL;

return OK;

}

///////////////////////CNF操作函数///////////////////////////////

status ShowCNF(CNF S) //测试文件读写是否成功函数

{

if(S.CNFhead==NULL)

return ERROR;

int i;

ClaLink q=S.CNFhead;

LitLink p;

printf("%d %d \n",S.litnum,S.clanum);

for(i=0;i<S.clanum;i++)

{

q=q->next;

printf("%d%d %d %d ",q->data,q->flag,q->lnum,q->flagchange);

p=q->firstlit;

while((p=p->next)!=NULL)

printf("%d%d ",p->litseq,p->flag);

printf("\n");

}

return OK;

}

status LoadCNF(CNF &S, char FileName[])

//读入文件FileName的子句以及文字数据，创建邻接链表

{

FILE \*fp;

char words1[20],words2[20]; //用于存放注释的数组

int lit; //临时存放数字的变量

int counter; //记录子句中文字数目变量

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

return ERROR;

fscanf(fp,"%s",words2);

do

{

strcpy(words1,words2);

fscanf(fp,"%s",words2);

} while(strcmp(words1,"p")&&strcmp(words2,"cnf")); //将c注释全部忽略直至p cnf

fscanf(fp,"%d %d",&S.litnum,&S.clanum); //将文字、子句数读入

int i; //循环变量

LitLink lp,ltemp; //用于新建的两个文字指针

ClaLink cp,ctemp; //用于新建的两个子句指针

cp=S.CNFhead;

for(i=0;i<S.clanum;i++) //子句读取控制

{

lp=(LitNode\*)malloc(sizeof(LitNode));

ctemp=(ClaNode\*)malloc(sizeof(ClaNode));

ctemp->data=i+1;

ctemp->flag=1;

ctemp->flagchange=0;

ctemp->firstlit=lp;

cp->next=ctemp;

counter=0;

while(fscanf(fp,"%d",&lit)&&(lit!=0)) //文字读取控制

{

counter++;

ltemp=(LitNode\*)malloc(sizeof(LitNode));

ltemp->litseq=lit;

ltemp->flag=1;

lp->next=ltemp;

lp=ltemp;

}

lp->next=NULL;

cp=ctemp;

cp->lnum=counter;

}

cp->next=NULL;

fclose(fp);

return OK;

}

//////////////////////////////////MOM链表操作/////////////////////////

MOMList LocateMOMNode(MOMList ML,int lit)

{

MOMList p=ML->next;

while(p)

{

if(p->lit==abs(lit))

return p;

p=p->next;

}

return NULL;

}

status MOMListAssign(MOMList &ML,CNF S)

{

int i,j;

double temp;

ClaLink q=S.CNFhead;

LitLink p;

MOMList mp;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

mp=LocateMOMNode(ML,abs(p->litseq));

temp=1;

for(j=0;j<q->lnum;j++)

temp=temp/2;

mp->data+=temp;

}

}

return OK;

/\*int i; //出现最多次原则

ClaLink q=S.CNFhead;

LitLink p;

MOMList mp;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

mp=LocateMOMNode(ML,abs(p->litseq));

mp->data++;

}

}

return OK;\*/

}

status LiteraltoMOM(MOMList &ML,int lit) //将MOM链表中文字设为已清除

{

MOMList mp=LocateMOMNode(ML,abs(lit));

mp->flag=0; //表示该文字已被清除

return 0;

}

status LiteraloffMOM(MOMList &ML,int lit) //将MOM链表中文字设为未清除

{

MOMList mp=LocateMOMNode(ML,abs(lit));

mp->flag=1; //表示该文字未被清除

return 0;

}

status MOMListTraverse(MOMList ML)

{

if(ML==NULL)

return INFEASIBLE;

MOMList p=ML->next;

if(p==NULL)

return OK;

while(p)

{

printf("%d %f %d ",p->lit,p->data,p->flag);

p=p->next;

}

printf("\n");

return OK;

}

///////////////////////////////////////结果链表操作函数//////////////////////

status InsertResNode(ResList &L,int lit)

{

ResList p=L,q;

while((p->next!=NULL))

p=p->next;

q=(ResNode\*)malloc(sizeof(ResNode));

q->data=lit;

q->next=NULL;

p->next=q;

L->data++;

return OK;

}

int RecallResList(ResList &L)

{

int ret;

ResList q=L,p=q->next;

while(p->next)

{

q=p;

p=p->next;

}

ret=p->data;

free(p);

q->next=NULL;

L->data--;

return ret;

}

status ResListTraverse(ResList L)

// 如果结果表存在，依次显示元素，每个元素间空一格，返回OK；如果表L不存在，返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

ResList p=L->next;

if(p==NULL)

return OK;

while(p)

{

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

p=p->next;

}

printf("\n");

return OK;

}

////////////////////////DivideList操作函数/////////////////////

status PushDivideNode(DivideList &L,int lit)

{

DivideList p=L,q;

while((p->next!=NULL))

p=p->next;

q=(DivideNode\*)malloc(sizeof(DivideNode));

q->data=lit;

q->next=NULL;

p->next=q;

return OK;

}

int PopDivideList(DivideList &L)

{

int ret;

DivideList q=L,p=q->next;

while(p->next)

{

q=p;

p=p->next;

}

ret=p->data;

free(p);

q->next=NULL;

return ret;

}

status DivideListTraverse(DivideList L)

// 如果表存在，依次显示元素，每个元素间空一格，返回OK；如果表L不存在，返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

DivideList p=L->next;

if(p==NULL)

return OK;

printf("dividelist:");

while(p)

{

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

p=p->next;

}

printf("\n");

return OK;

}

///////////////////////////////CNF子句文字操作函数////////////////////////

ClaLink LocateClause(CNF S,int cla)

{

ClaLink cp=S.CNFhead;

int i;

for(i=1;i<=S.clanum;i++)

{

cp=cp->next;

if(i==cla)

return cp;

}

return NULL;

}

status IsUnitClause(ClaLink C) //判断目标子句是否为单子句，是则返回TRUE，否则返回FALSE

{

if(C->data==0)

return FALSE;

if(C->lnum==1)

return OK;

return FALSE;

}

int LocateUnitClause(CNF S) //查找并定位单子句,查找成功便返回子句序号，失败则返回ERROR

{

ClaLink p=S.CNFhead;

int i;

for(i=1;i<=S.clanum;i++)

{

p=p->next;

if(p->flag==0)

continue;

if(IsUnitClause(p))

return i;

}

return ERROR;

}

status DestroyClause(CNF &S,int cla,int lit) //目标子句flag设为0

{

if(cla<1||cla>S.clanum)

return INFEASIBLE;

ClaLink cp;

cp=LocateClause(S,cla);

if(cp&&cp->flag!=0)

{

cp->flag=0;

cp->flagchange=lit;

return OK;

}

return ERROR;

}

status ExistNonClause(CNF S) //是否存在空子句

{

ClaLink p=S.CNFhead;

int i;

for(i=0;i<S.clanum;i++)

{

p=p->next;

if(p->lnum==0&&p->flag==1)

return i+1;

}

return FALSE;

}

status AddClausetoLast(CNF &S,int lit)

{

ClaLink cp1=LocateClause(S,S.clanum);

ClaLink cp2;

LitLink lp;

S.clanum++;

cp2=(ClaNode\*)malloc(sizeof(ClaNode));

cp2->data=S.clanum;

cp2->flag=1;

cp2->lnum=1;

lp=(LitNode\*)malloc(sizeof(LitNode));

lp->next=NULL;

lp->litseq=lit;

lp->flag=1;

cp2->firstlit=(LitNode\*)malloc(sizeof(LitNode));

cp2->firstlit->next=lp;

cp1->next=cp2;

return OK;

}

status RemoveLastClause(CNF &S)

{

ClaLink cp1=LocateClause(S,S.clanum-1);

free(cp1->next);

cp1->next=NULL;

S.clanum--;

return OK;

}

int FindFirstLiteral(ClaLink L)

{

LitLink p=L->firstlit->next;

while(p)

{

if(p->flag==1)

return p->litseq;

p=p->next;

}

return ERROR;

}

status ExistClause(CNF S)

{

int i;

ClaLink cp;

for(i=1;i<=S.clanum;i++)

{

cp=LocateClause(S,i);

if(cp->flag==1&&cp->lnum!=0)

return TRUE;

}

return FALSE;

}

status ClearLiteral(CNF &S,int lit) //单子句规则实现，传入lit为文字值，执行完后返回OK

{

int i;

LitLink lp1;

ClaLink cp1=S.CNFhead;

for(i=1;i<=S.clanum;i++)

{

cp1=LocateClause(S,i);

if(cp1==NULL)

break;

lp1=cp1->firstlit;

while((lp1=lp1->next)!=NULL)

{

if(lp1->litseq==-lit) //包含文字非L的状况

{

lp1->flag=0;

cp1->lnum--;

break;

}

if(lp1->litseq==lit)

{

DestroyClause(S,cp1->data,lit);

break;

}

}

}

return OK;

}

/////////////////////////////分裂策略函数/////////////////////////////////

int SimpleDivideStrategy(ResList L,CNF S)

//简单的分裂策略：选取第一个未在结果链表中出现的文字作为分裂策略

{

ResList p;

int i;

for(i=1;i<=S.litnum;i++)

{

p=L->next;

while(p)

{

if(p->data==i||p->data==-i)

break;

p=p->next;

}

if(p==NULL)

return -i;

}

return 0;

}

int MOMDivideStrategy(ResList L,CNF S,MOMList ML)

//最短子句出现频率最大优先

{

MOMList mp=ML->next;

while(mp)

{

if(mp->flag)

return mp->lit;

mp=mp->next;

}

return 0;

}

/////////////////////////////////回溯操作函数///////////////////////

status MOMBacktracking(CNF &S,ResList &L,MOMList &ML,int lit) //总回溯流程

{

int temp=0;

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

LiteraloffMOM(ML,temp);

}

return OK;

}

int BacktrackingbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) //总回溯流程

{

int temp=0;

int lit=PopDivideList(DL);

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

LiteraloffMOM(ML,temp);

}

return lit;

}

status Backtracking(CNF &S,ResList &L,int lit) //总回溯流程（改进前

{

int temp=0;

int i;

ClaLink cp;

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

}

for(i=S.clanum;i>0;i--)

{

cp=LocateClause(S,i);

if(FindFirstLiteral(cp)==lit)

{

RemoveLastClause(S);

break;

}

RemoveLastClause(S);

}

return OK;

}

status RecallCNF(CNF &S,int lit)

{

int i;

ClaLink q=S.CNFhead;

LitLink p;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

if(p->litseq==-lit)

{

p->flag=1;

q->lnum++;

break;

}

if(p->litseq==lit&&q->flagchange==lit)

{

q->flag=1;

break;

}

}

}

return OK;

}

///////////////////////DPLL求解部分//////////////////////////

status DPLLbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) //非递归实现DPLL

{

int cla,lit;

ClaLink cp;

int temp;

int flag;

while(ExistClause(S))

{

flag=1;

while(cla=LocateUnitClause(S))

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

if(ExistNonClause(S))

{

lit=BacktrackingbyLoop(S,L,ML,DL);

if(L->next==NULL)

return FALSE;

ClearLiteral(S,-lit);

InsertResNode(L,-lit);

LiteraltoMOM(ML,lit);

flag=0;

break;

}

if(ExistClause(S)==0)

return OK;

}

if(flag==1)

{

lit=MOMDivideStrategy(L,S,ML);

PushDivideNode(DL,lit);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

}

}

return OK;

}

status MOMDPLL(CNF &S,ResList &L,MOMList &ML) //升级DPLL算法

//传入CNF，结果链表L，MOM排序表ML，分裂策略传递文字literal

{

int cla=LocateUnitClause(S),lit;

int i;

ClaLink cp;

int temp; //状态储存变量

while(cla)

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit); //清除文字

InsertResNode(L,lit); //结果链表中插入已清除文字

LiteraltoMOM(ML,lit);

if(ExistClause(S)==0)

return TRUE;

else if(i=ExistNonClause(S))

return FALSE;

cla=LocateUnitClause(S);

}

lit=MOMDivideStrategy(L,S,ML);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

temp=MOMDPLL(S,L,ML);

if(temp)

return TRUE;

MOMBacktracking(S,L,ML,lit);

ClearLiteral(S,-lit);

InsertResNode(L,-lit);

LiteraltoMOM(ML,-lit);

return MOMDPLL(S,L,ML);

}

status DPLL(CNF &S,ResList &L) //初级DPLL算法

{

int cla=LocateUnitClause(S),lit;

ClaLink cp;

int temp; //状态储存变量

while(cla)

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit);

InsertResNode(L,lit);

if(ExistClause(S)==0)

return TRUE;

else if(ExistNonClause(S))

return FALSE;

cla=LocateUnitClause(S);

}

lit=SimpleDivideStrategy(L,S);

AddClausetoLast(S,lit);

temp=DPLL(S,L);

if(temp)

return TRUE;

Backtracking(S,L,lit);

AddClausetoLast(S,-lit);

return DPLL(S,L);

}

/////////////////////////////////排序算法///////////////////////////////////

void BubbleSortR(ResList &List)

{

ResList p,q,tail;

tail = NULL;

while((List->next->next) != tail)

{

p = List;

q = List->next;

while(q->next != tail)

{

if(abs(q->data) > abs(q->next->data))

{

p->next = q->next;

q->next = q->next->next;

p->next->next = q;

q = p->next;

}

q = q->next;

p = p->next;

}

tail = q;

}

}

void BubbleSortM(MOMList &List)

{

MOMList p,q,tail;

tail = NULL;

while((List->next->next) != tail)

{

p = List;

q = List->next;

while(q->next != tail)

{

if(q->data < q->next->data)

{

p->next = q->next;

q->next = q->next->next;

p->next->next = q;

q = p->next;

}

q = q->next;

p = p->next;

}

tail = q;

}

}

//////////////////////////////////////结果输出函数//////////////////////////

status ResultPrint(ResList L,int satisfy)

{

printf("s %d\n",satisfy);

printf("v ");

BubbleSortR(L);

printf("t ");

ResListTraverse(L);

return OK;

}

int ResWrite(int satisfy,double time,ResList L,char filename[])

{

ResList p=L->next;

int i=1;

while(filename[i]!='\0') i++;

filename[i-3]='r'; //只改变文件的扩展名

filename[i-2]='e';

filename[i-1]='s';

FILE \*fp;

fp=fopen(filename,"w");

if(fp==NULL){

printf("打开文件失败\n");

return ERROR;

}

fprintf(fp,"s %d\n",satisfy); //res是求解结果，1表示满足，0表示不满足，-1未定

fprintf(fp,"v\n");

while(p)

{

i++;

if(i%10==0) fprintf(fp,"\n");

fprintf(fp,"%5d",p->data);

p=p->next;

}

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

fclose(fp);

return OK;

}

# sudoku.cpp

#include "func.h"

/////////////////////////初始化操作//////////////////////////////

status InitCNF(CNF &S)

{

S.clanum=0;

S.litnum=0;

S.CNFhead=(ClaNode\*)malloc(sizeof(ClaNode));

return OK;

}

status InitResultList(ResList &L)

{

L=(ResNode\*)malloc(sizeof(ResNode));

L->data=0;

L->next=NULL;

return OK;

}

status InitDivideList(DivideList &DL)

{

DL=(DivideNode\*)malloc(sizeof(DivideNode));

DL->data=0;

DL->next=NULL;

return OK;

}

status InitMOMList(MOMList &ML,int litnum) //MOM初始化

{

ML=(MOMNode\*)malloc(sizeof(MOMNode));

ML->lit=0;

ML->data=0; //头结点初始化

MOMList p=ML,q;

int i;

for(i=1;i<=litnum;i++)

{

q=(MOMNode\*)malloc(sizeof(MOMNode));

q->lit=i;

q->flag=1;

q->data=0;

p->next=q;

p=q;

}

p->next=NULL;

return OK;

}

///////////////////////CNF操作函数///////////////////////////////

status ShowCNF(CNF S) //测试文件读写是否成功函数

{

if(S.CNFhead==NULL)

return ERROR;

int i;

ClaLink q=S.CNFhead;

LitLink p;

printf("%d %d \n",S.litnum,S.clanum);

for(i=0;i<S.clanum;i++)

{

q=q->next;

printf("%d%d %d %d ",q->data,q->flag,q->lnum,q->flagchange);

p=q->firstlit;

while((p=p->next)!=NULL)

printf("%d%d ",p->litseq,p->flag);

printf("\n");

}

return OK;

}

status LoadCNF(CNF &S, char FileName[])

//读入文件FileName的子句以及文字数据，创建邻接链表

{

FILE \*fp;

char words1[20],words2[20]; //用于存放注释的数组

int lit; //临时存放数字的变量

int counter; //记录子句中文字数目变量

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

return ERROR;

fscanf(fp,"%s",words2);

do

{

strcpy(words1,words2);

fscanf(fp,"%s",words2);

} while(strcmp(words1,"p")&&strcmp(words2,"cnf")); //将c注释全部忽略直至p cnf

fscanf(fp,"%d %d",&S.litnum,&S.clanum); //将文字、子句数读入

int i; //循环变量

LitLink lp,ltemp; //用于新建的两个文字指针

ClaLink cp,ctemp; //用于新建的两个子句指针

cp=S.CNFhead;

for(i=0;i<S.clanum;i++) //子句读取控制

{

lp=(LitNode\*)malloc(sizeof(LitNode));

ctemp=(ClaNode\*)malloc(sizeof(ClaNode));

ctemp->data=i+1;

ctemp->flag=1;

ctemp->flagchange=0;

ctemp->firstlit=lp;

cp->next=ctemp;

counter=0;

while(fscanf(fp,"%d",&lit)&&(lit!=0)) //文字读取控制

{

counter++;

ltemp=(LitNode\*)malloc(sizeof(LitNode));

ltemp->litseq=lit;

ltemp->flag=1;

lp->next=ltemp;

lp=ltemp;

}

lp->next=NULL;

cp=ctemp;

cp->lnum=counter;

}

cp->next=NULL;

fclose(fp);

return OK;

}

//////////////////////////////////MOM链表操作/////////////////////////

MOMList LocateMOMNode(MOMList ML,int lit)

{

MOMList p=ML->next;

while(p)

{

if(p->lit==abs(lit))

return p;

p=p->next;

}

return NULL;

}

status MOMListAssign(MOMList &ML,CNF S)

{

int i,j;

double temp;

ClaLink q=S.CNFhead;

LitLink p;

MOMList mp;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

mp=LocateMOMNode(ML,abs(p->litseq));

temp=1;

for(j=0;j<q->lnum;j++)

temp=temp/2;

mp->data+=temp;

}

}

return OK;

/\*int i; //出现最多次原则

ClaLink q=S.CNFhead;

LitLink p;

MOMList mp;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

mp=LocateMOMNode(ML,abs(p->litseq));

mp->data++;

}

}

return OK;\*/

}

status LiteraltoMOM(MOMList &ML,int lit) //将MOM链表中文字设为已清除

{

MOMList mp=LocateMOMNode(ML,abs(lit));

mp->flag=0; //表示该文字已被清除

return 0;

}

status LiteraloffMOM(MOMList &ML,int lit) //将MOM链表中文字设为未清除

{

MOMList mp=LocateMOMNode(ML,abs(lit));

mp->flag=1; //表示该文字未被清除

return 0;

}

status MOMListTraverse(MOMList ML)

{

if(ML==NULL)

return INFEASIBLE;

MOMList p=ML->next;

if(p==NULL)

return OK;

while(p)

{

printf("%d %f %d ",p->lit,p->data,p->flag);

p=p->next;

}

printf("\n");

return OK;

}

///////////////////////////////////////结果链表操作函数//////////////////////

status InsertResNode(ResList &L,int lit)

{

ResList p=L,q;

while((p->next!=NULL))

p=p->next;

q=(ResNode\*)malloc(sizeof(ResNode));

q->data=lit;

q->next=NULL;

p->next=q;

L->data++;

return OK;

}

int RecallResList(ResList &L)

{

int ret;

ResList q=L,p=q->next;

while(p->next)

{

q=p;

p=p->next;

}

ret=p->data;

free(p);

q->next=NULL;

L->data--;

return ret;

}

status ResListTraverse(ResList L)

// 如果结果表存在，依次显示元素，每个元素间空一格，返回OK；如果表L不存在，返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

ResList p=L->next;

if(p==NULL)

return OK;

while(p)

{

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

p=p->next;

}

printf("\n");

return OK;

}

////////////////////////DivideList操作函数/////////////////////

status PushDivideNode(DivideList &L,int lit)

{

DivideList p=L,q;

while((p->next!=NULL))

p=p->next;

q=(DivideNode\*)malloc(sizeof(DivideNode));

q->data=lit;

q->next=NULL;

p->next=q;

return OK;

}

int PopDivideList(DivideList &L)

{

int ret;

DivideList q=L,p=q->next;

while(p->next)

{

q=p;

p=p->next;

}

ret=p->data;

free(p);

q->next=NULL;

return ret;

}

status DivideListTraverse(DivideList L)

// 如果表存在，依次显示元素，每个元素间空一格，返回OK；如果表L不存在，返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

DivideList p=L->next;

if(p==NULL)

return OK;

printf("dividelist:");

while(p)

{

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

p=p->next;

}

printf("\n");

return OK;

}

///////////////////////////////CNF子句文字操作函数////////////////////////

ClaLink LocateClause(CNF S,int cla)

{

ClaLink cp=S.CNFhead;

int i;

for(i=1;i<=S.clanum;i++)

{

cp=cp->next;

if(i==cla)

return cp;

}

return NULL;

}

status IsUnitClause(ClaLink C) //判断目标子句是否为单子句，是则返回TRUE，否则返回FALSE

{

if(C->data==0)

return FALSE;

if(C->lnum==1)

return OK;

return FALSE;

}

int LocateUnitClause(CNF S) //查找并定位单子句,查找成功便返回子句序号，失败则返回ERROR

{

ClaLink p=S.CNFhead;

int i;

for(i=1;i<=S.clanum;i++)

{

p=p->next;

if(p->flag==0)

continue;

if(IsUnitClause(p))

return i;

}

return ERROR;

}

status DestroyClause(CNF &S,int cla,int lit) //目标子句flag设为0

{

if(cla<1||cla>S.clanum)

return INFEASIBLE;

ClaLink cp;

cp=LocateClause(S,cla);

if(cp&&cp->flag!=0)

{

cp->flag=0;

cp->flagchange=lit;

return OK;

}

return ERROR;

}

status ExistNonClause(CNF S) //是否存在空子句

{

ClaLink p=S.CNFhead;

int i;

for(i=0;i<S.clanum;i++)

{

p=p->next;

if(p->lnum==0&&p->flag==1)

return i+1;

}

return FALSE;

}

status AddClausetoLast(CNF &S,int lit)

{

ClaLink cp1=LocateClause(S,S.clanum);

ClaLink cp2;

LitLink lp;

S.clanum++;

cp2=(ClaNode\*)malloc(sizeof(ClaNode));

cp2->data=S.clanum;

cp2->flag=1;

cp2->lnum=1;

lp=(LitNode\*)malloc(sizeof(LitNode));

lp->next=NULL;

lp->litseq=lit;

lp->flag=1;

cp2->firstlit=(LitNode\*)malloc(sizeof(LitNode));

cp2->firstlit->next=lp;

cp1->next=cp2;

return OK;

}

status RemoveLastClause(CNF &S)

{

ClaLink cp1=LocateClause(S,S.clanum-1);

free(cp1->next);

cp1->next=NULL;

S.clanum--;

return OK;

}

int FindFirstLiteral(ClaLink L)

{

LitLink p=L->firstlit->next;

while(p)

{

if(p->flag==1)

return p->litseq;

p=p->next;

}

return ERROR;

}

status ExistClause(CNF S)

{

int i;

ClaLink cp;

for(i=1;i<=S.clanum;i++)

{

cp=LocateClause(S,i);

if(cp->flag==1&&cp->lnum!=0)

return TRUE;

}

return FALSE;

}

status ClearLiteral(CNF &S,int lit) //单子句规则实现，传入lit为文字值，执行完后返回OK

{

int i;

LitLink lp1;

ClaLink cp1=S.CNFhead;

for(i=1;i<=S.clanum;i++)

{

cp1=LocateClause(S,i);

if(cp1==NULL)

break;

lp1=cp1->firstlit;

while((lp1=lp1->next)!=NULL)

{

if(lp1->litseq==-lit) //包含文字非L的状况

{

lp1->flag=0;

cp1->lnum--;

break;

}

if(lp1->litseq==lit)

{

DestroyClause(S,cp1->data,lit);

break;

}

}

}

return OK;

}

/////////////////////////////分裂策略函数/////////////////////////////////

int SimpleDivideStrategy(ResList L,CNF S)

//简单的分裂策略：选取第一个未在结果链表中出现的文字作为分裂策略

{

ResList p;

int i;

for(i=1;i<=S.litnum;i++)

{

p=L->next;

while(p)

{

if(p->data==i||p->data==-i)

break;

p=p->next;

}

if(p==NULL)

return -i;

}

return 0;

}

int MOMDivideStrategy(ResList L,CNF S,MOMList ML)

//最短子句出现频率最大优先

{

MOMList mp=ML->next;

while(mp)

{

if(mp->flag)

return mp->lit;

mp=mp->next;

}

return 0;

}

/////////////////////////////////回溯操作函数///////////////////////

status MOMBacktracking(CNF &S,ResList &L,MOMList &ML,int lit) //总回溯流程

{

int temp=0;

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

LiteraloffMOM(ML,temp);

}

return OK;

}

int BacktrackingbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) //总回溯流程

{

int temp=0;

int lit=PopDivideList(DL);

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

LiteraloffMOM(ML,temp);

}

return lit;

}

status Backtracking(CNF &S,ResList &L,int lit) //总回溯流程（改进前

{

int temp=0;

int i;

ClaLink cp;

while(temp!=lit)

{

temp=RecallResList(L);

RecallCNF(S,temp);

}

for(i=S.clanum;i>0;i--)

{

cp=LocateClause(S,i);

if(FindFirstLiteral(cp)==lit)

{

RemoveLastClause(S);

break;

}

RemoveLastClause(S);

}

return OK;

}

status RecallCNF(CNF &S,int lit)

{

int i;

ClaLink q=S.CNFhead;

LitLink p;

for(i=0;i<S.clanum;i++)

{

q=q->next;

p=q->firstlit;

while((p=p->next)!=NULL)

{

if(p->litseq==-lit)

{

p->flag=1;

q->lnum++;

break;

}

if(p->litseq==lit&&q->flagchange==lit)

{

q->flag=1;

break;

}

}

}

return OK;

}

///////////////////////DPLL求解部分//////////////////////////

status DPLLbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) //非递归实现DPLL

{

int cla,lit;

ClaLink cp;

int temp;

int flag;

while(ExistClause(S))

{

flag=1;

while(cla=LocateUnitClause(S))

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

if(ExistNonClause(S))

{

lit=BacktrackingbyLoop(S,L,ML,DL);

if(L->next==NULL)

return FALSE;

ClearLiteral(S,-lit);

InsertResNode(L,-lit);

LiteraltoMOM(ML,lit);

flag=0;

break;

}

if(ExistClause(S)==0)

return OK;

}

if(flag==1)

{

lit=MOMDivideStrategy(L,S,ML);

PushDivideNode(DL,lit);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

}

}

return OK;

}

status MOMDPLL(CNF &S,ResList &L,MOMList &ML) //升级DPLL算法

//传入CNF，结果链表L，MOM排序表ML，分裂策略传递文字literal

{

int cla=LocateUnitClause(S),lit;

int i;

ClaLink cp;

int temp; //状态储存变量

while(cla)

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit); //清除文字

InsertResNode(L,lit); //结果链表中插入已清除文字

LiteraltoMOM(ML,lit);

if(ExistClause(S)==0)

return TRUE;

else if(i=ExistNonClause(S))

return FALSE;

cla=LocateUnitClause(S);

}

lit=MOMDivideStrategy(L,S,ML);

ClearLiteral(S,lit);

InsertResNode(L,lit);

LiteraltoMOM(ML,lit);

temp=MOMDPLL(S,L,ML);

if(temp)

return TRUE;

MOMBacktracking(S,L,ML,lit);

ClearLiteral(S,-lit);

InsertResNode(L,-lit);

LiteraltoMOM(ML,-lit);

return MOMDPLL(S,L,ML);

}

status DPLL(CNF &S,ResList &L) //初级DPLL算法

{

int cla=LocateUnitClause(S),lit;

ClaLink cp;

int temp; //状态储存变量

while(cla)

{

cp=LocateClause(S,cla);

lit=FindFirstLiteral(cp);

ClearLiteral(S,lit);

InsertResNode(L,lit);

if(ExistClause(S)==0)

return TRUE;

else if(ExistNonClause(S))

return FALSE;

cla=LocateUnitClause(S);

}

lit=SimpleDivideStrategy(L,S);

AddClausetoLast(S,lit);

temp=DPLL(S,L);

if(temp)

return TRUE;

Backtracking(S,L,lit);

AddClausetoLast(S,-lit);

return DPLL(S,L);

}

/////////////////////////////////排序算法///////////////////////////////////

void BubbleSortR(ResList &List)

{

ResList p,q,tail;

tail = NULL;

while((List->next->next) != tail)

{

p = List;

q = List->next;

while(q->next != tail)

{

if(abs(q->data) > abs(q->next->data))

{

p->next = q->next;

q->next = q->next->next;

p->next->next = q;

q = p->next;

}

q = q->next;

p = p->next;

}

tail = q;

}

}

void BubbleSortM(MOMList &List)

{

MOMList p,q,tail;

tail = NULL;

while((List->next->next) != tail)

{

p = List;

q = List->next;

while(q->next != tail)

{

if(q->data < q->next->data)

{

p->next = q->next;

q->next = q->next->next;

p->next->next = q;

q = p->next;

}

q = q->next;

p = p->next;

}

tail = q;

}

}

//////////////////////////////////////结果输出函数//////////////////////////

status ResultPrint(ResList L,int satisfy)

{

printf("s %d\n",satisfy);

printf("v ");

BubbleSortR(L);

printf("t ");

ResListTraverse(L);

return OK;

}

int ResWrite(int satisfy,double time,ResList L,char filename[])

{

ResList p=L->next;

int i=1;

while(filename[i]!='\0') i++;

filename[i-3]='r'; //只改变文件的扩展名

filename[i-2]='e';

filename[i-1]='s';

FILE \*fp;

fp=fopen(filename,"w");

if(fp==NULL){

printf("打开文件失败\n");

return ERROR;

}

fprintf(fp,"s %d\n",satisfy); //res是求解结果，1表示满足，0表示不满足，-1未定

fprintf(fp,"v\n");

while(p)

{

i++;

if(i%10==0) fprintf(fp,"\n");

fprintf(fp,"%5d",p->data);

p=p->next;

}

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

fclose(fp);

return OK;

}

# func.h

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"def.h"

#include "string.h"

#include <time.h>

/////////////////////////初始化操作//////////////////////////////

status InitCNF(CNF &S);

status InitResultList(ResList &L);

status InitDivideList(DivideList &DL);

status InitMOMList(MOMList &ML,int litnum);

///////////////////////CNF操作函数///////////////////////////////

status ShowCNF(CNF S);

status LoadCNF(CNF &S, char FileName[]);

//////////////////////////////////MOM链表操作/////////////////////////

MOMList LocateMOMNode(MOMList ML,int lit);

status MOMListAssign(MOMList &ML,CNF S);

status LiteraltoMOM(MOMList &ML,int lit) ;

status LiteraloffMOM(MOMList &ML,int lit) ;

status MOMListTraverse(MOMList ML);

///////////////////////////////////////结果链表操作函数//////////////////////

status InsertResNode(ResList &L,int lit);

int RecallResList(ResList &L);

status ResListTraverse(ResList L);

////////////////////////DivideList操作函数/////////////////////

status PushDivideNode(DivideList &L,int lit);

int PopDivideList(DivideList &L);

status DivideListTraverse(DivideList L);

///////////////////////////////CNF子句文字操作函数////////////////////////

ClaLink LocateClause(CNF S,int cla);

status IsUnitClause(ClaLink C) ;

int LocateUnitClause(CNF S);

status DestroyClause(CNF &S,int cla,int lit);

status ExistNonClause(CNF S) ;

status AddClausetoLast(CNF &S,int lit);

status RemoveLastClause(CNF &S);

int FindFirstLiteral(ClaLink L);

status ExistClause(CNF S);

status ClearLiteral(CNF &S,int lit);

/////////////////////////////分裂策略函数/////////////////////////////////

int SimpleDivideStrategy(ResList L,CNF S);

int MOMDivideStrategy(ResList L,CNF S,MOMList ML);

/////////////////////////////////回溯操作函数///////////////////////

status MOMBacktracking(CNF &S,ResList &L,MOMList &ML,int lit);

int BacktrackingbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) ;

status Backtracking(CNF &S,ResList &L,int lit);

status RecallCNF(CNF &S,int lit);

///////////////////////DPLL求解部分//////////////////////////

status DPLLbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL);

status MOMDPLL(CNF &S,ResList &L,MOMList &ML) ;

status DPLL(CNF &S,ResList &L) ;

/////////////////////////////////排序算法///////////////////////////////////

void BubbleSortR(ResList &List);

void BubbleSortM(MOMList &List);

//////////////////////////////////////结果输出函数//////////////////////////

status ResultPrint(ResList L,int satisfy);

int ResWrite(int satisfy,double time,ResList L,char filename[]);

/////////////////////////////////////数独部分函数/////////////

status LoadSudoku4(int sudoku4[][4], char FileName[]);

status PrintSudodu4(int sudoku4[][4]);

status Sudoku4toCNF(int sudoku4[][4],char FileName[]);

status LoadSudoku6(int sudoku6[][6], char FileName[]);

status PrintSudodu6(int sudoku6[][6]);

status Sudoku6toCNF(int sudoku6[][6],char FileName[]);

status PuzzleResult(ResList L,int kind);

# def.h

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"def.h"

#include "string.h"

#include <time.h>

/////////////////////////初始化操作//////////////////////////////

status InitCNF(CNF &S);

status InitResultList(ResList &L);

status InitDivideList(DivideList &DL);

status InitMOMList(MOMList &ML,int litnum);

///////////////////////CNF操作函数///////////////////////////////

status ShowCNF(CNF S);

status LoadCNF(CNF &S, char FileName[]);

//////////////////////////////////MOM链表操作/////////////////////////

MOMList LocateMOMNode(MOMList ML,int lit);

status MOMListAssign(MOMList &ML,CNF S);

status LiteraltoMOM(MOMList &ML,int lit) ;

status LiteraloffMOM(MOMList &ML,int lit) ;

status MOMListTraverse(MOMList ML);

///////////////////////////////////////结果链表操作函数//////////////////////

status InsertResNode(ResList &L,int lit);

int RecallResList(ResList &L);

status ResListTraverse(ResList L);

////////////////////////DivideList操作函数/////////////////////

status PushDivideNode(DivideList &L,int lit);

int PopDivideList(DivideList &L);

status DivideListTraverse(DivideList L);

///////////////////////////////CNF子句文字操作函数////////////////////////

ClaLink LocateClause(CNF S,int cla);

status IsUnitClause(ClaLink C) ;

int LocateUnitClause(CNF S);

status DestroyClause(CNF &S,int cla,int lit);

status ExistNonClause(CNF S) ;

status AddClausetoLast(CNF &S,int lit);

status RemoveLastClause(CNF &S);

int FindFirstLiteral(ClaLink L);

status ExistClause(CNF S);

status ClearLiteral(CNF &S,int lit);

/////////////////////////////分裂策略函数/////////////////////////////////

int SimpleDivideStrategy(ResList L,CNF S);

int MOMDivideStrategy(ResList L,CNF S,MOMList ML);

/////////////////////////////////回溯操作函数///////////////////////

status MOMBacktracking(CNF &S,ResList &L,MOMList &ML,int lit);

int BacktrackingbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL) ;

status Backtracking(CNF &S,ResList &L,int lit);

status RecallCNF(CNF &S,int lit);

///////////////////////DPLL求解部分//////////////////////////

status DPLLbyLoop(CNF &S,ResList &L,MOMList &ML,DivideList &DL);

status MOMDPLL(CNF &S,ResList &L,MOMList &ML) ;

status DPLL(CNF &S,ResList &L) ;

/////////////////////////////////排序算法///////////////////////////////////

void BubbleSortR(ResList &List);

void BubbleSortM(MOMList &List);

//////////////////////////////////////结果输出函数//////////////////////////

status ResultPrint(ResList L,int satisfy);

int ResWrite(int satisfy,double time,ResList L,char filename[]);

/////////////////////////////////////数独部分函数/////////////

status LoadSudoku4(int sudoku4[][4], char FileName[]);

status PrintSudodu4(int sudoku4[][4]);

status Sudoku4toCNF(int sudoku4[][4],char FileName[]);

status LoadSudoku6(int sudoku6[][6], char FileName[]);

status PrintSudodu6(int sudoku6[][6]);

status Sudoku6toCNF(int sudoku6[][6],char FileName[]);

status PuzzleResult(ResList L,int kind);