

**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： CS1901**

**学 号： U201914909**

**姓 名： 杨鹏陈**

**指导教师： 卢萍**

**报告日期： 2021年3月28日**

**计算机科学与技术学院**

**任 务 书**

**设计内容**

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

**设计要求**

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(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%)

**目录**

[任 务 书 I](#_Toc1394)

[1 引言 1](#_Toc21422)

[1.1 课题背景与意义 1](#_Toc16656)

[1.2 国内外研究现状 1](#_Toc13913)

[1.3 课程设计的主要研究工作 1](#_Toc3363)

[2 系统需求分析与总体设计 2](#_Toc22539)

[2.1 系统需求分析 2](#_Toc28653)

[2.2 系统总体设计 2](#_Toc19238)

[3 系统详细设计 3](#_Toc27343)

[3.1 有关数据结构的定义 3](#_Toc27136)

[3.2主要算法设计 4](#_Toc15195)

[4 系统实现与测试 5](#_Toc24114)

[4.1 系统实现 5](#_Toc26085)

[4.2 系统测试 11](#_Toc8025)

[4.2.1 测试环境 11](#_Toc24007)

[4.2.2 SAT求解器测试 11](#_Toc3277)

[4.2.3 二进制数独测试 14](#_Toc187)

[5 总结与展望 16](#_Toc8731)

[5.1 全文总结 16](#_Toc1519)

**1 引言**

**1.1 课题背景与意义**

SAT问题即命题逻辑公式的可满足性问题（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。SAT问题也是程序设计与竞赛的经典问题。

**1.2 国内外研究现状**

国内外对SAT的研究量非常大。自此问题诞生以来，诞生了许多优秀的算法和求解器。在过去的二十多年中，许多主流SAT求解器被开发出来,这些求解器主要基于DPLL (Davis Put-nam Logemann Loveland)。而目前影响最大的则是在DPLL的基础上形成的目前最为流行的冲突驱动子句学习CDCL (Conflict-Driven Clause Learning)求解框架，基于CDCL的SAT求解器在近年也取得了巨大的发展。以此为框架，各种求解器在决策方面也有着多样的方法，也有类似猜测重启等等优化的方法。

**1.3 课程设计的主要研究工作**

本次课程设计中，我主要通过比较基础的DPLL方法和一种较为简单的化简策略实现了初级SAT求解器。为了将SAT求解器用于二进制数独的求解，我实现了基于挖洞法的二进制数独随机生成函数，再通过数独的读取解构函数生成cnf算例，以此通用SAT求解器实现求解。

**2** **系统需求分析与总体设计**

**2.1 系统需求分析**

此程序主要用于求解一些简单的SAT问题和cnf算例，能给出判定（是否可满足）、输出变元取值、输出时间。实现了求解8\*8以内的二进制数独问题，实现简单的交互二进制数独游戏。

**2.2 系统总体设计**

系统主要实现Sudoku二进制数独游戏一个大模块。为了实现此功能，设计了生成，转化，求解等子部分，其中随机生成唯一解的数独过程应用了SAT求解器，以下为系统示意图。



图2-1 系统模块结构图

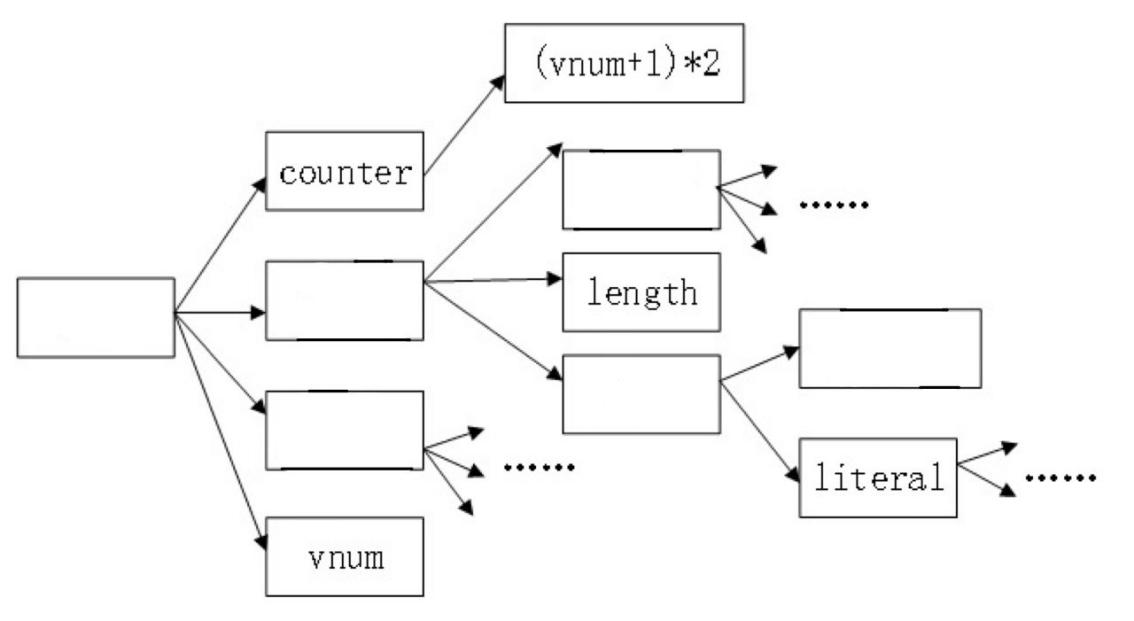
**3 系统详细设计**

**3.1 有关数据结构的定义**

系统主要处理的数据有子句集Clset，记录表counter，单文字句子链表One，多文字句子链表Others，变元数vnum，文字链表VarNode，句子长度length，文字literal，已确定文字，变元偏量。具体数据类型如表3-1，数据之间的关联如图3-1。

表3-1 数据类型

|  |  |  |
| --- | --- | --- |
| 数据 | 数据名 | 数据类型 |
| 记录表 | counter | int\*\* |
| 句子链表 | ClsNode(One/Other) | Struct |
| 变元数 | vnum | Int |
| 文字链表 | VarNode | Struct |
| 句子长度 | length | Int |
| 文字 | literal | Int |
| 已确定变元 | counter[][1] | Int |
| 变元偏量 | counter[][0] | Int |
| 子句集 | Clset | struct |



Clset

One

Others

Next

NextAdj

NextAdj

图3-1 数据之间的联系

**3.2主要算法设计**

这主要包含二进制棋局puzzle模块和SAT求解器两个模块。两个模块已经写成函数，具体过程和算法见4.1系统实现中的14（puzzle）和15（SAT求解器）。

首先是DPLL中的假设策略，即4.1中的6（FindValue）,这里我首先为了让程序跑起来，使用了最简单的选取第一个未确定真值的变元的策略，效果不是很好；然后我尝试了在子句集中定义counter数组用于记录每个变元出现的次数，出现次数越多则越应该选取该变元，但是发现对于极少数数据集提升较大，对于一些数据集还有反作用，甚至导致一些原本可以跑出来的数据集无法在5分钟内跑出来；后来我询问了一些学长，找到了一个编写难度适中，效果也不错的算法，即定义一个变元偏量，用于显示一个变元的独立程度，考虑到子句越长，所含的变元越多，该子句能满足的可能性越大。所以子句长度越短，对其中变量的正负性影响越大，所以越不应该选择该变元的策略，在偏量的计算公式中考虑了这一点，而是自增自减1000/length。这是学长多次测试后得出的简单且好的公式，有效的加入了子句长度的条件，也能避免溢出和小数归零导致的误差。直接遍历循环找到第一个没有确定的变元按照其活性给出决策。

最后就是通过将递归换成非递归的方式优化了程序的效率。

**4 系统实现与测试**

**4.1 系统实现**

根据3.1的设计，用C语言定义各种数据类型；

1、删除文字链表

函数名称：ClearVarNode

输入参数：文字链表头结点的引用head

返回值： status（typedef int，表示状态）

函数功能：删除head指向的文字链表。

调用函数：无

设计：pfront指向head，pfront不为空时循环，pfront、pbehind一前一后两个指针，前一个删除释放空间，后一个记录位置用于循环。循环结束后head指向NULL。

2、删除句子链表

函数名称：ClearClsNode

输入参数：句子链表头结点的引用head

返回值： status（typedef int，表示状态）

函数功能：删除head指向的句子链表。

调用函数：ClearVarNode

设计：p指向head，p不为空时循环，q、p一前一后两个指针，前一个调用ClearVarNode删除所有文字，然后删除结点释放空间，后一个记录位置用于循环。循环结束后head指向NULL。

3、遍历消解

函数名称：TraverseDispose

输入参数：子句集Clset

返回值： status（typedef int，表示状态）

函数功能：按照规则，根据当前已确定变元消解化简子句集

调用函数：ClearVarNode，ClearClsNode

设计：见流程图。由于忘记使用引用导致过程分类讨论很多，写的有些混乱。



图3-2 遍历消解流程图

4、文件读取

函数名称：ReadClset

输入参数：待写入子句集Clset，cnf文件名filename[100]

返回值： status（typedef int，表示状态）

函数功能：按照cnf文件内容生成子句集Clset

调用函数：无

设计：见流程图



图3-3 文件读取流程图

5、深拷贝

函数名称：DeepCopy

输入参数：模板子句集Clset，等待复制的空子句集的引用record

返回值： status（typedef int，表示状态）

函数功能：把当前Clset的内容复制到record，并按照此时状态计算变元偏量

调用函数：无

设计：给record的counter分配好内存空间并初始化，counter[][0]置零，counter[][1]和Clset的一样，然后遍历Clset的两个句子链表复制子句，每复制一个子句都计算该子句中出现的文字的偏量，累加到counter[][1]。偏量的绝对值为10000/length，正负与变量在该子句中的文字一致。

6、假设决策

函数名称：ChooseVar

输入参数：当前消解完的子句集record

返回值： 做出的决策int（含正负）

函数功能：给出一个猜测的变元值

调用函数：无

设计：遍历record中的变元，找到的第一个还没有确定的变元，即counter[][1]为0的值，按照该变元累计偏量的值确定正负并返回。

7、添加结点

函数名称：InitClsNode

输入参数：待添加结点的子句集引用Clset，决策完待添加的变元值value

返回值：status（typedef int，表示状态）

函数功能：把只有value的单子句添加到Clset->One字句链表中

调用函数：无

设计：分配好内存空间并赋值，首插法直接添加结点。

8、DLPP模块

函数名称：DPLL

输入参数：子句集的引用Clset

返回值：status（typedef int，表示状态）

函数功能：DPLL循环求解。

调用函数：ClearClsNode，TraverseDispose，Deepcopy，FindValue，InitClsNode

设计：见流程图



图3-4 DPLL模块流程图

9、SAT求解器

函数名称：SATsolver

输入参数：无

函数功能：求解cnf文件中的SAT问题并输出res答案文件

调用函数：ReadClset，DPLL

设计：让用户输入cnf文件名，用读取到Clset之后用DPLL求解，根据求解结果输出答案文件，在DPLL过程中记录时间一并写入答案文件中。

10、约束二排列组合

函数名称：Combination2

输入参数：处于正确位置的文件指针fp，排列组合元素数组array，结果数组result，排列组合个数size，排列组合递归循环变量index，递归层数deep

返回值：status（typedef int，表示状态）

函数功能：把约束二要求的子句按格式写入fp指向的文件位置。

调用函数：递归实现

设计：见流程图。



图3-5 约束二排列组合流程图

11、约束三排列组合

函数名称：CombinationR/ CombinationL

输入参数：处于正确位置的文件指针fp，排列组合元素数组array，结果数组result，排列组合个数size，排列组合递归循环变量index，递归层数deep，变元计数器引用vnumcount

返回值：status（typedef int，表示状态）

函数功能：把约束三要求的子句按格式写入fp指向的文件位置。

调用函数：递归实现

设计：类似约束三排列组合，加了vnumcount用于记录当前变元个数，用于约束三中添加变元。

12、文件转换

函数名称： PuzzleTransform

输入参数：二维指针puzzle，puzzle大小size，待写入文件名filename[100]

返回值：status（typedef int，表示状态）

函数功能：把二进制数独题面转换为cnf文件写入filename中。

调用函数：Combination2，CombinationR，CombinationL

设计：根据size和当前数独体面中的空位计算出变元数和子句数，按照格式接入第一行，再根据三个约束写好子句，约束一直接按规律写，约束二、三给好实参调用上面的函数排列组合实现。

13、数独生成

函数名称： PuzzleInitAuto

输入参数： puzzle大小size

函数功能：随机生成一个有唯一解的size\*size的二进制数独。

调用函数：PuzzleTransform，ReadClset，DPLL

设计：见流程图。



图3-6数独生成流程图

14、字句展示

函数名称：Display

输入参数：需要展示子句的子句集Clset

函数功能：把Clset中的子句一一显示，可与cnf文件人工对比确认准确性

调用函数：无

设计：根据用户输入，选择是否输出。若要，则循环输出每一子句内容。

15、二进制数独

函数名称：puzzle

输入参数：无

函数功能：二进制数独游戏，自动生成等待用户输入并根据输入返回结果

调用函数：PuzzleInitAuto

设计：提示用户输入4/6/8，代表数独大小，调用PuzzleInitAuto生成棋局，并得到答案数组，提示用户按格式输入，输入后与答案比较，给出结果，如果错误用户可以选择再来一次。

**4.2 系统测试**

**4.2.1 测试环境**

Windows10，64位，8G内存，i7-8750H，2.21GHZ

Vscode配置的C/C++环境

**4.2.2 SAT求解器测试**

表4-1 SAT求解器三个版本性能测试对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 算例名 | 版本1/ms | 版本2/ms | 版本三/ms |
| 1 | sat-20.cnf | 121 | 104 | 31 |
| 3 | ais10.cnf | 7704 | 3908 | 3102 |

可见优化效率大约为46%.以下为助教给出测试算例结果（可跑出1-6,7\_1）

|  |  |
| --- | --- |
| 算例名 | 测试结果 |
| 1.cnf |  |
| 2.cnf |  |
| 3.cnf |  |
| 1. cnf   (unsatisfied) |  |
| 5.cnf |  |
| 6.cnf |  |
| 7\_1.cnf |  |

**4.2.3 二进制数独测试**

程序比较好的实现了二进制数独游戏的功能，数独为随机生成的唯一解的数独。演示过程截图如下：

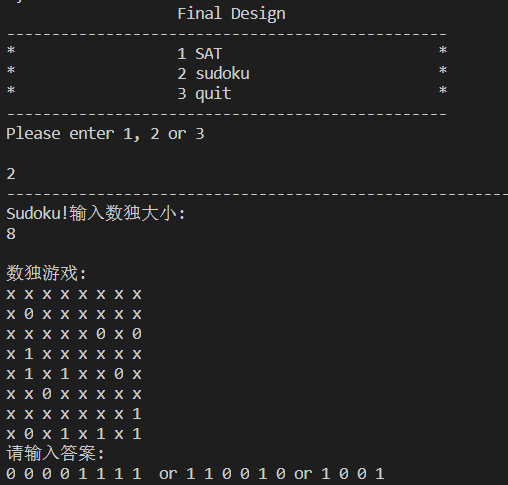


图4-1 输入8后随机生成的唯一解数独及格式提示

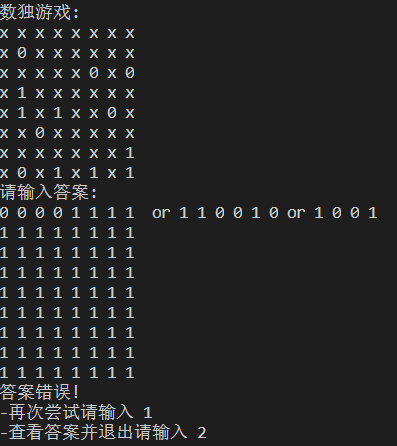


图4-2 答案错误提示

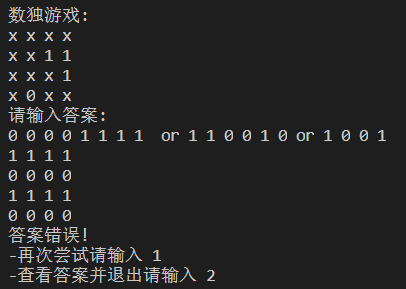


图4-3 答错后查看答案

**5 总结与展望**

**5.1 全文总结**

本次课程设计实验任务对我来说受益匪浅，收获了很多，在上学期老师就发了关于课设实验的任务书和资料包，寒假的时候我就开始了解和构思实验任务以及初步的定义了一些数据结构和简单的函数，开学上课以后每次上课都能有新的收获，有空闲的时间也会通过查阅资料、询问同学的方式来改进算法，也因为在这期间改动较大，定义的新的数据结构、对原有数据结构的改动较多，导致代码内部注释较少且某些函数逻辑不清晰，比较遗憾。

SAT求解器设计完成度一般，在助教给出的算例中能跑出1-6、7\_1。SAT求解器本身就是一个处于科技前沿的问题，我通过读论文、查阅资料、问同学和学长的方式等见识了许多优秀的算法思想和新颖、简洁、有效的数据结构，也尽量的把一些优秀的想法融入到我的SAT中。我的策略在写程序的过程中修改了无数次，最初的版本是选择第一个出现的变元（主要是为了先让程序跑起来试试看）；第二个版本加入了记录变元出现次数的数组，出现次数越多越先选择；最后效果最好的是学长的想法，即定义“偏量”。最后把递归实现改成了栈循环实现，减少了占用内存。另外，在助教检查的过程中，我发现我的代码对于一些异常情况的处理不好，以及代码注释太少，这些在撰写实验报告的期间都有改进。

二进制数独的完成度是比较好的。通过看论文我了解到了如何随机生成一个有唯一解的数独，然后稍作修改用于二进制数独上。凭借我最终的选取策略在解决数独问题时的优越性，我甚至可以完成8\*8的数独，算是超额完成了任务。唯一的遗憾是我的交互界面对用户输入的格式是有严格限制的，没有另外输入读取的规范化函数。但总体不影响使用。

本次课设可能是我对C语言和数据结构掌握程度的一次最好检验，整个过程中精心设计框架、算法、数据结构、函数，学到了不少技巧。

**参考文献**

[1]□王静康,张凤宝,夏淑倩等.论化工本科专业国际认证与国内认证的“实质性”.高等工程教育研究,2014,5:1-4

[2]□Stone J A, Howard L P. A simple technique for observing periodic nonlinearities in Michelson interferometers. Precision Engineering,1998,22(4):220-232

[3]□朱印红,袁衍明.Dreamweaver完美网页设计——技术入门篇.(第一版).北京:中国电力出版社,2006:19～20

[4]□Lewis S L. Physics and chemistry of the solar system.北京:北京大学出版社,2014.1～2

[5]□陈剑.上博简《民之父母》“而得既塞於四海矣”句解释[EB/OL］.简帛研究网站，<http://www.bamboosilk.org/Wssf/2003/chenjian03.htm．2003-01-18>

[6] 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

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

[8] 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

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

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

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

[12] 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.

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

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

<http://zhangroup.aporc.org/images/files/Paper_3485.pdf>

**附录**

problem5-200.cnf测试结果：

s 1

v 1 -2 -3 -4 -5 -6 7 -8 9 10 11 -12 -13 14 -15 -16 -17 18 -19 20 21 -22 -23 -24 -25 -26 27 -28 -29 -30 -31 32 33 -34 35 -36 37 -38 -39 40 41 -42 -43 -44 45 -46 47 48 -49 50 -51 -52 53 54 -55 -56 -57 58 59 60 61 62 63 64 -65 66 67 -68 -69 -70 -71 72 73 -74 -75 -76 -77 -78 -79 80 -81 -82 -83 84 -85 -86 -87 -88 -89 90 91 -92 -93 94 95 -96 97 98 99 -100 101 -102 -103 -104 105 106 107 108 109 -110 -111 -112 -113 -114 115 -116 -117 118 119 -120 -121 122 -123 -124 125 126 -127 -128 -129 -130 131 -132 -133 -134 135 136 137 -138 -139 -140 -141 -142 -143 144 -145 146 147 148 149 -150 -151 -152 153 -154 -155 -156 -157 158 -159 -160 -161 -162 163 164 165 166 -167 168 169 170 -171 -172 173 -174 -175 -176 -177 -178 -179 180 181 -182 -183 -184 -185 186 -187 -188 -189 -190 -191 -192 -193 194 -195 196 -197 -198 199 200

t 1565038.000000

eh-vmpc\_25.renamed-as.sat05-1913-625.cnf测试结果：

s 1

v -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20 -21 -22 -23 -24 25 -26 -27 -28 -29 -30 -31 -32 -33 -34 -35 -36 -37 -38 -39 -40 -41 -42 -43 -44 45 -46 -47 -48 -49 -50 -51 -52 -53 -54 -55 -56 -57 -58 -59 -60 -61 62 -63 -64 -65 -66 -67 -68 -69 -70 -71 -72 -73 -74 -75 -76 -77 -78 -79 -80 -81 -82 -83 -84 -85 -86 -87 -88 89 -90 -91 -92 -93 -94 -95 -96 -97 -98 -99 -100 -101 -102 -103 104 -105 -106 -107 -108 -109 -110 -111 -112 -113 -114 -115 -116 -117 -118 -119 -120 -121 -122 -123 -124 -125 -126 -127 -128 -129 -130 -131 -132 -133 -134 -135 -136 -137 -138 -139 -140 -141 -142 -143 -144 -145 -146 -147 -148 -149 -150 -151 -152 -153 -154 -155 -156 -157 -158 -159 160 -161 -162 -163 -164 -165 -166 -167 -168 -169 -170 -171 -172 -173 -174 -175 -176 -177 -178 -179 -180 -181 -182 -183 -184 -185 -186 -187 -188 -189 -190 -191 -192 -193 -194 -195 -196 -197 0 -199 -200 -201 -202 -203 -204 -205 -206 -207 -208 -209 -210 -211 -212 -213 -214 -215 -216 -217 218 -219 -220 -221 -222 -223 -224 -225 -226 -227 -228 -229 -230 -231 -232 -233 -234 -235 -236 -237 -238 -239 -240 -241 -242 -243 -244 -245 -246 247 -248 -249 -250 -251 252 -253 -254 -255 -256 -257 -258 -259 -260 -261 -262 -263 -264 -265 -266 -267 -268 -269 -270 -271 -272 -273 -274 -275 -276 -277 -278 -279 -280 -281 -282 -283 -284 -285 -286 -287 -288 -289 290 -291 -292 -293 -294 -295 -296 -297 -298 -299 -300 -301 -302 -303 -304 -305 -306 -307 -308 309 -310 -311 -312 -313 -314 -315 -316 -317 -318 -319 -320 -321 -322 -323 -324 -325 -326 -327 -328 -329 -330 331 -332 -333 -334 -335 -336 -337 -338 -339 -340 -341 -342 -343 -344 -345 -346 -347 -348 -349 -350 -351 -352 -353 -354 -355 -356 -357 -358 -359 -360 -361 -362 -363 -364 -365 -366 367 -368 -369 -370 -371 -372 -373 -374 -375 376 -377 -378 -379 -380 -381 -382 -383 -384 -385 -386 -387 -388 -389 -390 -391 -392 -393 -394 -395 -396 -397 -398 -399 -400 -401 -402 -403 -404 -405 -406 -407 -408 -409 -410 -411 -412 -413 -414 -415 -416 -417 -418 -419 -420 -421 -422 423 -424 -425 -426 -427 428 -429 -430 -431 -432 -433 -434 -435 -436 -437 -438 -439 -440 -441 -442 -443 -444 -445 -446 -447 -448 -449 -450 -451 -452 -453 -454 -455 -456 -457 -458 -459 -460 -461 -462 -463 -464 -465 466 -467 -468 -469 -470 -471 -472 -473 -474 -475 -476 -477 -478 -479 -480 -481 -482 -483 -484 -485 -486 -487 -488 -489 -490 -491 -492 -493 -494 -495 -496 -497 498 -499 -500 -501 -502 -503 -504 -505 -506 507 -508 -509 -510 -511 -512 -513 -514 -515 -516 -517 -518 -519 -520 -521 -522 -523 -524 -525 -526 -527 -528 -529 -530 -531 -532 -533 -534 -535 -536 -537 -538 -539 -540 -541 -542 -543 -544 -545 -546 -547 548 -549 -550 -551 -552 -553 -554 -555 -556 -557 -558 -559 -560 -561 -562 563 -564 -565 -566 -567 -568 569 -570 571 -572 -573 574 -575 -576 -577 -578 -579 580 -581 -582 -583 -584 -585 -586 -587 -588 -589 -590 -591 -592 -593 -594 -595 -596 -597 -598 -599 -600 -601 -602 -603 -604 -605 -606 -607 -608 -609 -610 611 -612 -613 -614 -615 -616 -617 -618 -619 -620 -621 -622 -623 -624 -625

t 274870.000000

源代码

def.h

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <math.h>
4. #include <string.h>
5. #include <time.h>
6. #define OK 1
7. #define ERROR -1
8. #define UNFEASIBLE -2
9. #define TRUE 1
10. #define FALSE 0
11. #define MAX 50
12. #ifndef \_TEST\_H\_
13. #define \_TEST\_H\_
14. typedef struct VarNode
15. {
16. int literal;
17. struct VarNode \*next;
18. } VarNode; *//句子的表结点，文字*
19. typedef struct ClsNode
20. {
21. int length;
22. struct ClsNode \*NextNode;
23. VarNode \*NextAdj;
24. } ClsNode; *//句子链表*
25. typedef struct Clset
26. {
27. int \*\*counter; *//记录表*
28. ClsNode \*One;    *//只有一个文字的句子*
29. ClsNode \*Others; *//其他句子*
30. int vnum;
31. } Clset;
32. int DPLL(Clset \*&cset);
33. int ClearVarNode(VarNode \*&head);
34. int ClearClsNode(ClsNode \*&head);
35. int TraverseDispose(Clset \*cset); *//根据确定变元简化sat*
36. ClsNode \*InitClsNode();   *//初始化句子结点*
37. int ReadCset(Clset \*cset, char filename[100]);  *//从cnf文件读入*
38. int DeepCopy(Clset \*cset, Clset \*&record);   *//深拷贝*
39. int ChooseVar(Clset \*record);    *//搜索最佳值*
40. int InitClsNode(Clset \*&cset, int value);    *//初始化新策略结点*
41. int PuzzleTransform(char \*\*puzzle, int size, char filename[100]);                        *//生成二进制数组的cnf文件*
42. int Combination2(FILE \*fp, int array[], int result[], int size, int index, int deep);    *//针对第二个条件排列组合构建子句*
43. int CombinationR(FILE \*fp, int result[], int size, int index, int deep, int &vnumcount); *//针对第三个条件行排列组合构建子句*
44. int CombinationL(FILE \*fp, int result[], int size, int index, int deep, int &vnumcount); *//针对第三个条件列排列组合构建子句*
45. char \*\*PuzzleInitAuto(int size);        *//自动随机生成数组*
46. int puzzle(void);          *//数独游戏*
47. int SATsolver(void);                   *//SAT求解*
48. int Display(Clset \*cset);           *//展示子句*
49. #endif

func\_SAT.h

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. \*文件名称：func\_SAT.h
3. \*文件描述：关于求解SAT问题的函数。
4. \*日    期：2020.03.01
5. \*作    者：YPC
6. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
7. *#include "def.h"*
8. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
9. \*函数名称：ClearVarNode
10. \*函数功能：删除head指向的文字链表。
11. \*输入参数：表头指针head
12. \*返回参数：OK -> 删除成功
13. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
14. int ClearVarNode(VarNode \*&head)
15. {
16. VarNode \*pfront, \*pbehind;
17. pbehind = head;
18. while (pbehind)
19. {
20. pfront = pbehind;
21. pbehind = pbehind->next;
22. free(pfront);
23. }
24. head = NULL;
25. return OK;
26. }
27. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
28. \*函数名称：ClearClsNode
29. \*函数功能：删除head指向的句子链表。
30. \*输入参数：表头指针head
31. \*返回参数：OK -> 删除成功
32. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
33. int ClearClsNode(ClsNode \*&head)
34. {
35. ClsNode \*pfront, \*pbehind;
36. pbehind = head;
37. while (pbehind)
38. {
39. pfront = pbehind;
40. pbehind = pbehind->NextNode;
41. ClearVarNode(pfront->NextAdj);
42. free(pfront);
43. }
44. head = NULL;
45. return OK;
46. }
47. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
48. \*函数名称：TraverseDispose
49. \*函数功能：按照规则，根据当前已确定变元消解化简子句集
50. \*输入参数：子句集Clset
51. \*返回参数：OK -> 消解成功
52. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
53. int TraverseDispose(Clset \*cset)
54. {
55. ClsNode \*pforcl;
56. ClsNode \*qforcl;
57. ClsNode \*del;
58. VarNode \*pforlit;
59. VarNode \*qforlit;
60. if (cset->Others)
61. {
62. pforcl = cset->Others;
63. qforlit = pforcl->NextAdj;
64. if (!pforcl->NextNode)
65. {
66. if (qforlit->literal == cset->counter[abs(qforlit->literal)][1])
67. {
68. ClearClsNode(pforcl);
69. }
70. else if (qforlit->literal == -(cset->counter[abs(qforlit->literal)][1]))
71. {
72. pforcl->NextAdj = qforlit->next;
73. free(qforlit);
74. qforlit = pforcl->NextAdj;
75. pforcl->length--;
76. if (pforcl->length == 1)
77. {
78. pforcl->NextNode = cset->One;
79. cset->One = pforcl;
80. cset->Others = NULL;
81. }
82. }
83. else
84. {
85. pforlit = qforlit->next;
86. while (pforlit)
87. {
88. if (pforlit->literal == cset->counter[abs(pforlit->literal)][1])
89. {
90. ClearClsNode(pforcl);
91. break;
92. }
93. else if (pforlit->literal == -(cset->counter[abs(pforlit->literal)][1]))
94. {
95. qforlit->next = pforlit->next;
96. free(pforlit);
97. pforlit = qforlit->next;
98. pforcl->length--;
99. if (pforcl->length == 1)
100. {
101. pforcl->NextNode = cset->One;
102. cset->One = pforcl;
103. cset->Others = NULL;
104. break;
105. }
106. continue;
107. }
108. qforlit = qforlit->next;
109. pforlit = pforlit->next;
110. }*//while*
111. }*//else*
112. }*//if*
113. else
114. {
115. qforcl = cset->Others;
116. pforcl = qforcl->NextNode;
117. label:             *//标签：不用递增*
118. while (pforcl) *//第一个先不管*
119. {
120. qforlit = pforcl->NextAdj;
121. pforlit = qforlit->next; *//q在前p在后*
122. *//第一个文字不管,后续循环*
123. while (pforlit)
124. {
125. if (pforlit->literal == cset->counter[abs(pforlit->literal)][1])
126. {
127. del = pforcl;
128. pforcl = pforcl->NextNode;
129. qforcl->NextNode = pforcl;
130. ClearVarNode(del->NextAdj);
131. free(del);
132. goto label;
133. }
134. else if (pforlit->literal == -(cset->counter[abs(pforlit->literal)][1]))
135. {
136. qforlit->next = pforlit->next;
137. free(pforlit);
138. pforlit = qforlit->next;
139. pforcl->length--;
140. if (pforcl->length == 1)
141. {
142. del = pforcl;
143. pforcl = pforcl->NextNode;
144. qforcl->NextNode = pforcl;
145. del->NextNode = cset->One;
146. cset->One = del;
147. goto label;
148. }
149. continue;
150. }
151. qforlit = qforlit->next;
152. pforlit = pforlit->next;
153. } *//end of inner while*
154. *//处理第一个文字*
155. pforlit = pforcl->NextAdj;
156. if (pforlit->literal == cset->counter[abs(pforlit->literal)][1])
157. {
158. del = pforcl;
159. pforcl = pforcl->NextNode;
160. qforcl->NextNode = pforcl;
161. ClearVarNode(del->NextAdj);
162. free(del);
163. goto label;
164. }
165. else if (pforlit->literal == -(cset->counter[abs(pforlit->literal)][1]))
166. {
167. pforcl->NextAdj = pforlit->next;
168. free(pforlit);
169. pforcl->length--;
170. pforlit = pforcl->NextAdj;
171. if (pforcl->length == 1)
172. {
173. del = pforcl;
174. pforcl = pforcl->NextNode;
175. qforcl->NextNode = pforcl;
176. del->NextNode = cset->One;
177. cset->One = del;
178. goto label;
179. }
180. }
181. pforcl = pforcl->NextNode;
182. qforcl = qforcl->NextNode;
183. }*//while*
184. } *//end of while*
185. pforcl = cset->Others; *//处理第一个句子*
186. if (pforcl)
187. {
188. qforlit = pforcl->NextAdj;
189. pforlit = qforlit->next;
190. int flagA = 0;
191. while (pforlit)
192. {
193. if (pforlit->literal == cset->counter[abs(pforlit->literal)][1])
194. {
195. cset->Others = pforcl->NextNode;
196. ClearVarNode(pforcl->NextAdj);
197. free(pforcl);
198. flagA = 1;
199. break;
200. }
201. else if (pforlit->literal == -(cset->counter[abs(pforlit->literal)][1]))
202. {
203. qforlit->next = pforlit->next;
204. free(pforlit);
205. pforlit = qforlit->next;
206. pforcl->length--;
207. if (pforcl->length == 1)
208. {
209. cset->Others = pforcl->NextNode;
210. pforcl->NextNode = cset->One;
211. cset->One = pforcl;
212. flagA = 1;
213. break;
214. }
215. continue;
216. }
217. qforlit = qforlit->next;
218. pforlit = pforlit->next;
219. }
220. if (!flagA)
221. {
222. pforlit = pforcl->NextAdj;
223. if (pforlit->literal == cset->counter[abs(pforlit->literal)][1])
224. {
225. cset->Others = pforcl->NextNode;
226. ClearVarNode(pforcl->NextAdj);
227. free(pforcl);
228. }
229. else if (pforlit->literal == -(cset->counter[abs(pforlit->literal)][1]))
230. {
231. pforcl->NextAdj = pforlit->next;
232. free(pforlit);
233. pforlit = pforcl->NextAdj;
234. pforcl->length--;
235. if (pforcl->length == 1)
236. {
237. cset->Others = pforcl->NextNode;
238. pforcl->NextNode = cset->One;
239. cset->One = pforcl;
240. }
241. }
242. }*//if*
243. }*//if pforcl*
244. }*//if(cset->Others)*
245. return OK;
246. }
247. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
248. \*函数名称：ReadCset
249. \*函数功能：按照cnf文件内容生成子句集Clset
250. \*输入参数：待写入子句集Clset，cnf文件名filename[100]
251. \*返回参数：OK -> 删除成功
252. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
253. int ReadCset(Clset \*cset, char filename[100])
254. {
255. FILE \*fp;
256. char temp[10];
257. int i, lnum;    *//i循环*
258. char a;         *//a单个储存*
259. char line[256]; *//解决前面的文字*
260. int feasible = 1;
261. if ((fp = fopen(filename, "r")) == NULL)
262. {
263. printf("open %s failed\n", filename);
264. return ERROR;
265. }
266. while (fgets(line, sizeof(line), fp))
267. {
268. if (line[0] == 'c')
269. continue;
270. if (line[0] == 'p')
271. {
272. sscanf(line, "p cnf %d %d", &cset->vnum, &lnum);
273. break;
274. }
275. }
276. memset(temp, '\0', sizeof(temp));
277. int k, l;
278. cset->counter = (int \*\*)malloc(sizeof(int \*) \* (cset->vnum + 1)); *//为二维数组分配MAX行*
279. for (k = 0; k < (cset->vnum + 1); k++)
280. { *//为每列分配2个大小空间，0记录正负权值，1记录总出现，*
281. cset->counter[k] = (int \*)malloc(sizeof(int) \* 2);
282. }
283. *//初始化*
284. for (k = 0; k < (cset->vnum + 1); k++)
285. {
286. for (l = 0; l < 2; l++)
287. {
288. cset->counter[k][l] = 0;
289. }
290. }
291. i = 1;
292. ClsNode \*cl = (ClsNode \*)malloc(sizeof(ClsNode));
293. cl->length = 0;
294. cl->NextAdj = NULL;
295. cl->NextNode = NULL; *//给第一个句子准备*
296. ClsNode \*one = NULL;
297. ClsNode \*other = NULL;
298. int SingleRecord[40] = {0}; *//记录单个句子的出现次数，处理重复*
299. while ((fgets(&a, 2, fp)) != NULL)
300. {
301. if ((a <= '9' && a > '0') || (a == '0' && i != 1) || a == '-'){
302. temp[i] = a;
303. i++;
304. }
305. else if (a == ' ')
306. {
307. int x = atoi(temp + 1);
308. int j, flag;
309. flag = 1;
310. for (j = 0; j < 40; j++)
311. {
312. if (SingleRecord[j] == x)
313. {
314. flag = 0;
315. break;
316. }
317. else if (SingleRecord[j] == 0)
318. {
319. SingleRecord[j] = x;
320. break;
321. }
322. }
323. if (flag == 0)
324. {
325. i = 1;
326. memset(temp, '\0', sizeof(temp));
327. continue;
328. }
329. cl->length++;
330. VarNode \*lit = (VarNode \*)malloc(sizeof(VarNode));
331. lit->literal = x;
332. lit->next = cl->NextAdj;
333. cl->NextAdj = lit;
334. i = 1;
335. memset(temp, '\0', sizeof(temp));
336. } *//end if ' '*
337. else if (a == '0' && i == 1)
338. {
339. char f;
340. fgets(&f, 2, fp);
341. if (f != '\n')
342. return ERROR;
343. else
344. {
345. if (cl->length == 0)
346. {
347. feasible = 0;
348. return UNFEASIBLE; *//有空句子*
349. }
350. else if (cl->length == 1)
351. {
352. if (one == NULL)
353. {
354. one = cl;
355. cl->NextNode = NULL;
356. }
357. else
358. {
359. cl->NextNode = one;
360. one = cl;
361. }
362. }
363. else if (cl->length > 1)
364. {
365. if (other == NULL)
366. {
367. other = cl;
368. cl->NextNode = NULL;
369. }
370. else
371. {
372. cl->NextNode = other;
373. other = cl;
374. }
375. }
376. else
377. return ERROR;
378. }
379. i = 1;
380. ClsNode \*t = (ClsNode \*)malloc(sizeof(ClsNode));
381. t->length = 0;
382. t->NextAdj = NULL;
383. t->NextNode = NULL;
384. cl = t;
385. memset(temp, '\0', sizeof(temp));
386. memset(SingleRecord, 0, sizeof(SingleRecord));
387. }
388. }
389. cset->One = one;
390. cset->Others = other;
391. fclose(fp);
392. return OK;
393. }
394. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
395. \*函数名称：DeepCopy
396. \*函数功能：把当前Clset的内容复制到record，并按照此时状态计算变元偏量
397. \*输入参数：模板子句集Clset，等待复制的空子句集的引用record
398. \*返回参数：OK -> 删除成功
399. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
400. int DeepCopy(Clset \*cset, Clset \*&record) *//深拷贝Cset*
401. {
402. *//复制cset*
403. record = (Clset \*)malloc(sizeof(Clset));
404. record->vnum = cset->vnum;
405. record->One = NULL;
406. record->Others = NULL;
407. int k;
408. record->counter = (int \*\*)malloc(sizeof(int \*) \* (record->vnum + 1)); *//为二维数组分配MAX行*
409. for (k = 0; k < (record->vnum + 1); k++)
410. { *//为每列分配2个大小空间，0记录出现，1记录答案*
411. record->counter[k] = (int \*)malloc(sizeof(int) \* 2);
412. }
413. *//初始化*
414. for (k = 0; k < (record->vnum + 1); k++)
415. {
416. record->counter[k][0] = 0;
417. record->counter[k][1] = cset->counter[k][1];
418. }
419. ClsNode \*pfrc = cset->Others;
420. VarNode \*pfrl = pfrc->NextAdj;
421. while (pfrc)
422. {
423. ClsNode \*add = (ClsNode \*)malloc(sizeof(ClsNode));
424. add->length = pfrc->length;
425. add->NextAdj = NULL;
426. pfrl = pfrc->NextAdj;
427. while (pfrl)
428. {
429. VarNode \*ad = (VarNode \*)malloc(sizeof(VarNode));
430. ad->literal = pfrl->literal;
431. if (pfrl->literal > 0)
432. {
433. record->counter[pfrl->literal][0] = record->counter[pfrl->literal][0] + 1000 / pfrc->length;
434. }
435. else if (pfrl->literal < 0)
436. {
437. record->counter[-pfrl->literal][0] = record->counter[-pfrl->literal][0] - 1000 / pfrc->length;
438. }
439. else
440. return ERROR;
441. ad->next = add->NextAdj;
442. add->NextAdj = ad;
443. pfrl = pfrl->next;
444. }
445. add->NextNode = record->Others;
446. record->Others = add;
447. pfrc = pfrc->NextNode;
448. } *//复制结束*
449. return OK;
450. }
451. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
452. \*函数名称：ChooseVar
453. \*函数功能：给出一个猜测的变元值
454. \*输入参数：当前消解完的子句集record
455. \*返回参数：决策int（含正负）
456. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
457. int ChooseVar(Clset \*record) *//做出下一个假设变量的策略*
458. {
459. int i;
460. for (i = 1; i < (record->vnum + 1); i++)
461. {
462. if (!record->counter[i][1])
463. break;
464. }
465. if (record->counter[i][0] < 0)
466. i = -i;
467. return i;
468. }
469. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
470. \*函数名称：InitClsNode
471. \*函数功能：把只有value的单子句添加到Clset->One字句链表中
472. \*输入参数：待添加结点的子句集引用Clset，决策完待添加的变元值value
473. \*返回参数：OK -> 添加成功
474. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
475. int InitClsNode(Clset \*&cset, int value)
476. {
477. ClsNode \*add = (ClsNode \*)malloc(sizeof(ClsNode));
478. add->NextAdj = (VarNode \*)malloc(sizeof(VarNode));
479. add->length = 1;
480. add->NextAdj->next = NULL;
481. add->NextAdj->literal = value;
482. add->NextNode = cset->One;
483. cset->One = add;
484. return OK;
485. }
486. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
487. \*函数名称：Display
488. \*函数功能：展示读入成功的子句集中的所有子句
489. \*输入参数：子句集Clset
490. \*返回参数：OK -> 输出成功
491. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
492. int Display(Clset \*cset) *//展示子句*
493. {
494. int i;
495. int counter = 1;
496. printf("display the Clauses? -Yes enter 1  -No enter 0\n");
497. scanf("%d", &i);
498. getchar();
499. if (!i)
500. return OK;
501. printf("----------------------------------------------------\n");
502. printf("number of variety:%d\n", cset->vnum);
503. ClsNode \*p = cset->One;
504. while (p)
505. {
506. printf("%d clause: %d\n", counter, p->NextAdj->literal);
507. counter++;
508. p = p->NextNode;
509. }
510. p = cset->Others;
511. while (p)
512. {
513. printf("%d clause: ", counter);
514. VarNode \*pa = p->NextAdj;
515. while (pa)
516. {
517. printf("%d ", pa->literal);
518. pa = pa->next;
519. }
520. printf("\n");
521. counter++;
522. p = p->NextNode;
523. }
524. printf("display finished!\n");
525. printf("----------------------------------------------------\n");
526. return OK;
527. }
528. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
529. \*函数名称：DPLL
530. \*函数功能：DPLL循环求解
531. \*输入参数：子句集的引用Clset
532. \*返回参数：int（按文档中规定输出状态）
533. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
534. int DPLL(Clset \*&cset)
535. {
536. Clset \*stack[200];
537. int top = 0;
538. stack[top] = cset;
539. int flag = 0;
540. while (1)
541. {
542. while (stack[top]->One)
543. {
544. flag = 0;
545. ClsNode \*pforcl = stack[top]->One;
546. int literal, f;
547. while (pforcl)
548. {
549. literal = pforcl->NextAdj->literal;
550. if (stack[top]->counter[abs(literal)][1] == literal || stack[top]->counter[abs(literal)][1] == 0)
551. stack[top]->counter[abs(literal)][1] = literal;
552. else if (stack[top]->counter[abs(literal)][1] == -literal)
553. {
554. flag = 1;
555. break;
556. }
557. else
558. return ERROR;
559. pforcl = pforcl->NextNode;
560. }
561. if (flag == 1)
562. break;
563. ClearClsNode(stack[top]->One);
564. stack[top]->One = NULL;
565. f = TraverseDispose(stack[top]); *//所有含这个文字的句子处理*
566. if (f == UNFEASIBLE)
567. return UNFEASIBLE;
568. }
569. if (flag == 1)
570. {
571. stack[top]->vnum = 0;
572. ClearClsNode(stack[top]->Others);
573. ClearClsNode(stack[top]->One);
574. top--;
575. if (top < 0)
576. return UNFEASIBLE;
577. }
578. else if (!stack[top]->Others)
579. {
580. cset->counter = stack[top]->counter;
581. return OK;
582. }
583. else
584. {
585. DeepCopy(stack[top], stack[top + 1]);
586. top++;
587. int value = ChooseVar(stack[top]);
588. InitClsNode(stack[top], value);
589. InitClsNode(stack[top - 1], -value);
590. }
591. }
592. }
593. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
594. \*函数名称：SATsolver
595. \*函数功能：简单的用户界面，以及输出结果
596. \*输入参数：无
597. \*返回参数：OK -> 输出成功
598. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
599. int SATsolver(void) *//SAT求解器*
600. {
601. char question[100];
602. Clset \*cset = (Clset \*)malloc(sizeof(Clset));
603. printf("Enter the question filename:\n");
604. scanf("%s", question);
605. int f = ReadCset(cset, question);
606. if (f == ERROR)
607. {
608. printf("error");
609. getchar();
610. return ERROR;
611. }
612. Display(cset);
613. char answerfilename[100];
614. int i;
615. for (i = 0; i < 96; i++)
616. {
617. answerfilename[i] = question[i];
618. if (answerfilename[i] == '.')
619. {
620. answerfilename[i + 1] = 'r';
621. answerfilename[i + 2] = 'e';
622. answerfilename[i + 3] = 's';
623. answerfilename[i + 4] = '\0';
624. break;
625. }
626. }
627. if (i == 96)
628. {
629. printf("ERROR!");
630. return ERROR;
631. }
632. if (f == UNFEASIBLE)
633. {
634. FILE \*fp;
635. if ((fp = fopen(answerfilename, "wb")) == NULL)
636. {
637. printf("save error!\n");
638. return ERROR;
639. }
640. fprintf(fp, "s 0\n");
641. fprintf(fp, "t 0\n");
642. getchar();
643. }
644. else
645. {
646. clock\_t start, finish;
647. double totaltime;
648. start = clock();
649. f = DPLL(cset);
650. finish = clock();
651. totaltime = (double)(finish - start);
652. FILE \*fp;
653. if ((fp = fopen(answerfilename, "wb")) == NULL)
654. {
655. printf("save error!\n");
656. return ERROR;
657. }
658. if (f == OK)
659. {
660. fprintf(fp, "s 1\n");
661. fprintf(fp, "v ");
662. int i;
663. for (i = 1; i <= cset->vnum; i++)
664. {
665. fprintf(fp, "%d ", cset->counter[i][1]);
666. }
667. fprintf(fp, "\n");
668. fprintf(fp, "t %f", totaltime);
669. }
670. else
671. {
672. fprintf(fp, "s 0\n");
673. fprintf(fp, "t %f", totaltime);
674. }
675. fclose(fp);
676. }
677. printf("finished! Enter spcae please!");
678. getchar();
679. return OK;
680. }

func\_Sudoku.h

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. \*文件名称：func\_Sudoku.h
3. \*文件描述：关于求解二进制数独问题的函数。
4. \*日    期：2020.03.01
5. \*作    者：YPC
6. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
7. #include "def.h"
8. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
9. \*函数名称：Combination2
10. \*函数功能：把约束三要求的子句按格式写入fp指向的文件位置
11. \*输入参数：处于正确位置的文件指针fp，排列组合元素数组array，结果数组result，排列组合个数size，排列组合递归循环变量index，递归层数deep，变元计数器引用vnumcount
12. \*返回参数：OK -> 成功
13. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
14. int Combination2(FILE \*fp, int array[], int result[], int size, int index, int deep) *//约束二子句生成，排列组合递归*
15. {
16. int i;
17. if (deep == size / 2 + 1)
18. {
19. for (i = 0; i < size / 2 + 1; i++)
20. {
21. fprintf(fp, "%d ", result[i]);
22. }
23. fprintf(fp, "0\n");
24. for (i = 0; i < size / 2 + 1; i++)
25. {
26. fprintf(fp, "-%d ", result[i]);
27. }
28. fprintf(fp, "0\n");
29. return OK;
30. }
31. for (i = index; i < size; i++, index++)
32. {
33. result[deep] = array[i];
34. deep++;
35. Combination2(fp, array, result, size, index + 1, deep);
36. deep--;
37. }
38. return OK;
39. }
40. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
41. \*函数名称：CombinationR\L
42. \*函数功能：把约束三要求的子句按格式写入fp指向的文件位置。
43. \*输入参数：处于正确位置的文件指针fp，排列组合元素数组array，结果数组result，排列组合个数size，排列组合递归循环变量index，递归层数deep，变元计数器引用vnumcount
44. \*返回参数：OK -> 成功
45. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
46. int CombinationR(FILE \*fp, int result[], int size, int index, int deep, int &vnumcount) *//约束三行子句生成，排列组合递归*
47. {
48. int i;
49. if (deep == 2)
50. {
51. for (i = 1; i <= size; i++)
52. {
53. vnumcount++;
54. fprintf(fp, "%d -%d 0\n", result[0] \* size + i, vnumcount);
55. fprintf(fp, "%d -%d 0\n", result[1] \* size + i, vnumcount);
56. fprintf(fp, "-%d -%d %d 0\n", result[0] \* size + i, result[1] \* size + i, vnumcount);
57. vnumcount++;
58. fprintf(fp, "-%d -%d 0\n", result[0] \* size + i, vnumcount);
59. fprintf(fp, "-%d -%d 0\n", result[1] \* size + i, vnumcount);
60. fprintf(fp, "%d %d %d 0\n", result[0] \* size + i, result[1] \* size + i, vnumcount);
61. vnumcount++;
62. fprintf(fp, "-%d %d 0\n", vnumcount - 2, vnumcount);
63. fprintf(fp, "-%d %d 0\n", vnumcount - 1, vnumcount);
64. fprintf(fp, "%d %d -%d 0\n", vnumcount - 2, vnumcount - 1, vnumcount);
65. }
66. vnumcount++;
67. fprintf(fp, "-%d ", vnumcount);
68. for (i = 0; i < size; i++)
69. {
70. fprintf(fp, "-%d ", vnumcount - 1 - i \* 3);
71. }
72. fprintf(fp, "0\n");
73. for (i = 1; i <= size; i++)
74. {
75. fprintf(fp, "%d %d 0\n", vnumcount - i, vnumcount);
76. }
77. return OK;
78. }
79. for (i = index; i < size; i++, index++)
80. {
81. result[deep] = i;
82. deep++;
83. CombinationR(fp, result, size, index + 1, deep, vnumcount);
84. deep--;
85. }
86. return OK;
87. }
88. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
89. \*函数名称：ClearVarNode
90. \*函数功能：删除head指向的文字链表。
91. \*输入参数：表头指针head
92. \*返回参数：OK -> 删除成功
93. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
94. int CombinationL(FILE \*fp, int result[], int size, int index, int deep, int &vnumcount) *//约束三列子句生成，排列组合递归*
95. {
96. int i;
97. if (deep == 2)
98. {
99. for (i = 0; i < size; i++)
100. {
101. vnumcount++;
102. fprintf(fp, "%d -%d 0\n", i \* size + result[0] + 1, vnumcount);
103. fprintf(fp, "%d -%d 0\n", i \* size + result[1] + 1, vnumcount);
104. fprintf(fp, "-%d -%d %d 0\n", i \* size + result[0] + 1, i \* size + result[1] + 1, vnumcount);
105. vnumcount++;
106. fprintf(fp, "-%d -%d 0\n", i \* size + result[0] + 1, vnumcount);
107. fprintf(fp, "-%d -%d 0\n", i \* size + result[1] + 1, vnumcount);
108. fprintf(fp, "%d %d %d 0\n", i \* size + result[0] + 1, i \* size + result[1] + 1, vnumcount);
109. vnumcount++;
110. fprintf(fp, "-%d %d 0\n", vnumcount - 2, vnumcount);
111. fprintf(fp, "-%d %d 0\n", vnumcount - 1, vnumcount);
112. fprintf(fp, "%d %d -%d 0\n", vnumcount - 2, vnumcount - 1, vnumcount);
113. }
114. vnumcount++;
115. fprintf(fp, "-%d ", vnumcount);
116. for (i = 0; i < size; i++)
117. {
118. fprintf(fp, "-%d ", vnumcount - 1 - i \* 3);
119. }
120. fprintf(fp, "0\n");
121. for (i = 1; i <= size; i++)
122. {
123. fprintf(fp, "%d %d 0\n", vnumcount - i, vnumcount);
124. }
125. return OK;
126. }
127. for (i = index; i < size; i++, index++)
128. {
129. result[deep] = i;
130. deep++;
131. CombinationL(fp, result, size, index + 1, deep, vnumcount);
132. deep--;
133. }
134. return OK;
135. }
136. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
137. \*函数名称：PuzzleTransform
138. \*函数功能：把二进制数独题面转换为cnf文件写入filename中。
139. \*输入参数：二维指针puzzle，puzzle大小size，待写入文件名filename[100]
140. \*返回参数：OK -> 成功
141. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
142. int PuzzleTransform(char \*\*puzzle, int size, char filename[100]) *//棋局转换为cnf*
143. {
144. if (size != 4 && size != 6 && size != 8)
145. {
146. printf("only 4,6,8 are allowed!");
147. return ERROR;
148. }
149. FILE \*fp;
150. if ((fp = fopen(filename, "wb")) == NULL)
151. {
152. printf("save error!\n");
153. return ERROR;
154. }
155. int vnum, lnum;
156. if (size == 8)
157. {
158. vnum = 1464;
159. lnum = 4536 + 192 + 1792;
160. }
161. else if (size == 6)
162. {
163. vnum = 570 + 36;
164. lnum = 1830 + 96 + 360;
165. }
166. else if (size == 4)
167. {
168. vnum = 156 + 16;
169. lnum = 492 + 32 + 64;
170. }
171. int i, j;
172. for (i = 0; i < size; i++)
173. {
174. for (j = 0; j < size; j++)
175. {
176. if (puzzle[i][j] == '1' || puzzle[i][j] == '0')
177. lnum++;
178. }
179. }
180. fprintf(fp, "p cnf %d %d\n", vnum, lnum);
181. for (i = 0; i < size; i++)
182. {
183. for (j = 0; j < size; j++)
184. {
185. if (puzzle[i][j] == '1')
186. fprintf(fp, "%d 0\n", i \* size + j + 1);
187. else if (puzzle[i][j] == '0')
188. fprintf(fp, "-%d 0\n", i \* size + j + 1);
189. }
190. }
191. for (i = 0; i < size; i++)
192. {
193. for (j = 0; j < size - 2; j++)
194. {
195. fprintf(fp, "%d %d %d 0\n", i \* size + j + 1, i \* size + j + 2, i \* size + j + 3);
196. fprintf(fp, "-%d -%d -%d 0\n", i \* size + j + 1, i \* size + j + 2, i \* size + j + 3);
197. }
198. }
199. for (i = 0; i < size - 2; i++)
200. {
201. for (j = 0; j < size; j++)
202. {
203. fprintf(fp, "%d %d %d 0\n", i \* size + j + 1, (i + 1) \* size + j + 1, (i + 2) \* size + j + 1);
204. fprintf(fp, "-%d -%d -%d 0\n", i \* size + j + 1, (i + 1) \* size + j + 1, (i + 2) \* size + j + 1);
205. }
206. }
207. for (i = 0; i < size; i++)
208. {
209. int array[size];
210. int result[size];
211. for (j = 0; j < size; j++)
212. {
213. array[j] = i \* size + j + 1;
214. }
215. Combination2(fp, array, result, size, 0, 0);
216. }
217. for (i = 0; i < size; i++)
218. {
219. int array[size];
220. int result[size];
221. for (j = 0; j < size; j++)
222. {
223. array[j] = i + size \* j + 1;
224. }
225. Combination2(fp, array, result, size, 0, 0);
226. }
227. int array[size];
228. int result[size];
229. for (i = 0; i < size; i++)
230. {
231. array[i] = i;
232. }
233. int vnumcount = size \* size;
234. CombinationR(fp, result, size, 0, 0, vnumcount);
235. CombinationL(fp, result, size, 0, 0, vnumcount);
236. fclose(fp);
237. return OK;
238. }
239. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
240. \*函数名称：PuzzleInitAuto
241. \*函数功能：随机生成一个有唯一解的size\*size的二进制数独。
242. \*输入参数：puzzle大小size
243. \*返回参数：OK -> 成功
244. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
245. char \*\*PuzzleInitAuto(int size) *//挖洞自动生成棋局*
246. {
247. int x, y, i, j;
248. int n = size \* size / 8 + 2;
249. char filename[100] = "PuzzleInitAuto.cnf";
250. char \*\*answer = (char \*\*)malloc(sizeof(char \*) \* size); *//为二维数组分配行*
251. for (i = 0; i < size; i++)
252. {
253. answer[i] = (char \*)malloc(sizeof(char) \* size);
254. }
255. while (1)
256. {
257. for (i = 0; i < size; i++)
258. {
259. for (j = 0; j < size; j++)
260. answer[i][j] = 'x';
261. }
262. srand(time(NULL));
263. for (i = 0; i < n; i++)
264. {
265. x = rand() % size;
266. y = rand() % size;
267. if (i % 2)
268. answer[x][y] = '1';
269. else
270. answer[x][y] = '0';
271. }
272. PuzzleTransform(answer, size, filename);
273. Clset \*cset = (Clset \*)malloc(sizeof(Clset));
274. ReadCset(cset, filename);
275. if (DPLL(cset) == OK)
276. {
277. for (i = 0; i < size; i++)
278. {
279. for (j = 0; j < size; j++)
280. {
281. if (cset->counter[i \* size + j + 1][1] > 0)
282. answer[i][j] = '1';
283. else if (cset->counter[i \* size + j + 1][1] < 0)
284. answer[i][j] = '0';
285. else
286. return NULL;
287. }
288. }
289. break;
290. }
291. }
292. char \*\*a = (char \*\*)malloc(sizeof(char \*) \* size); *//为二维数组分配行*
293. for (i = 0; i < size; i++)
294. {
295. a[i] = (char \*)malloc(sizeof(char) \* size);
296. }
297. for (j = 0; j < size; j++)
298. a[0][j] = 'x';
299. for (j = 0; j < size; j++)
300. a[j][0] = 'x';
301. for (i = 1; i < size; i++)
302. {
303. for (j = 1; j < size; j++)
304. {
305. a[i][j] = answer[i][j];
306. }
307. }
308. char tempfile[100] = "temp.cnf";
309. for (i = 1; i < size; i++)
310. {
311. for (j = 1; j < size; j++)
312. {
313. if (a[i][j] == '1')
314. a[i][j] = '0';
315. else if (a[i][j] == '0')
316. a[i][j] = '1';
317. PuzzleTransform(a, size, tempfile);
318. Clset \*cset = (Clset \*)malloc(sizeof(Clset));
319. ReadCset(cset, tempfile);
320. int f;
321. f = DPLL(cset);
322. if (f == OK)
323. {
324. if (a[i][j] == '1')
325. a[i][j] = '0';
326. else if (a[i][j] == '0')
327. a[i][j] = '1';
328. }
329. else
330. {
331. a[i][j] = 'x';
332. }
333. free(cset);
334. }
335. }
336. for (i = 0; i < size; i++)
337. {
338. for (j = 0; j < size; j++)
339. printf("%c ", a[i][j]);
340. printf("\n");
341. }
342. return answer;
343. }
344. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
345. \*函数名称：puzzle
346. \*函数功能：二进制数独游戏，自动生成等待用户输入并根据输入返回结果
347. \*输入参数：五
348. \*返回参数：OK -> 成功
349. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
350. int puzzle(void) *//棋局功能*
351. {
352. int size, i, j, f = 0;
353. printf("---------------------------------------------------------------------\n");
354. printf("Sudoku!");
355. printf("输入数独大小:\n");
356. scanf("%d", &size);
357. getchar();
358. while (size != 4 && size != 6 && size != 8)
359. {
360. printf("只能输入4,6，或8.请再次输入：\n");
361. scanf("%d", &size);
362. getchar();
363. }
364. printf("\n数独游戏:\n");
365. char filename[100] = "PuzzleInitAuto.cnf";
366. char \*\*answer = PuzzleInitAuto(size);
367. char \*\*a = (char \*\*)malloc(sizeof(char \*) \* size); *//为二维数组分配MAX行*
368. for (i = 0; i < size; i++)
369. {
370. a[i] = (char \*)malloc(sizeof(char) \* size);
371. }
372. while (1)
373. {
374. f = 0;
375. printf("请输入答案:\n");
376. printf("0 0 0 0 1 1 1 1  or 1 1 0 0 1 0 or 1 0 0 1");
377. printf("\n");
378. for (i = 0; i < size; i++)
379. {
380. for (j = 0; j < size; j++)
381. {
382. scanf("%c", &a[i][j]);
383. getchar();
384. }
385. }
386. for (i = 0; i < size; i++)
387. {
388. for (j = 0; j < size; j++)
389. {
390. if (a[i][j] != answer[i][j])
391. {
392. printf("答案错误!\n");
393. while (f != 1 && f != 2)
394. {
395. printf("-再次尝试请输入 1\n-查看答案并退出请输入 2\n");
396. scanf("%d", &f);
397. getchar();
398. printf("\n");
399. break;
400. }
401. }
402. if (f)
403. break;
404. }
405. if (f)
406. break;
407. }
408. if (i == size && j == size)
409. {
410. printf("答案正确!\n标准答案为:\n\n");
411. break;
412. }
413. else if (f == 2)
414. break;
415. }
416. for (i = 0; i < size; i++)
417. {
418. for (j = 0; j < size; j++)
419. printf("%c ", answer[i][j]);
420. printf("\n");
421. }
422. printf("finished! Enter space please!");
423. getchar();
424. return OK;
425. }

huge.cpp

1. #include "def.h"
2. #include "func\_SAT.h"
3. #include "func\_Sudoku.h"
4. int main(void)
5. {
6. int option;
7. printf("                   Final Design\n");
8. while (1)
9. {
10. option = 0;
11. printf("-------------------------------------------------\n");
12. printf("\*                  1 SAT                        \*\n");
13. printf("\*                  2 sudoku                     \*\n");
14. printf("\*                  3 quit                       \*\n");
15. printf("-------------------------------------------------\n");
16. while (option != 1 && option != 2 && option != 3)
17. {
18. printf("Please enter 1, 2 or 3\n");
19. scanf("%d", &option);
20. }
21. if (option == 1)
22. SATsolver();
23. else if (option == 2)
24. puzzle();
25. else
26. break;
27. }
28. return 0;
29. }