

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级： CS1703**

**学 号： U201714668**

**姓 名：**

**指导教师： 袁凌**

**报告日期： 2019/4/7**

**计算机科学与技术学院**

# 任 务 书

1. **设计内容**

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

1. **设计要求**

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)
2. **公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)
4. **时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
5. **程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)
6. **SAT应用：**将数独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)
7. **参考文献**

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

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

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

[4]Carsten Sinz.Visualizing SAT Instances and Runs of the DPLL Algorithm.J Autom Reasoning (2007) 39:219–243

[5] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

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

[7]Ins Lynce and Jol Ouaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

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

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

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

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

**目录**

[任 务 书 I](#_Toc5589588)

[1引言 4](#_Toc5589589)

[1.1课题背景与意义 4](#_Toc5589590)

[1.2 国内外研究现状 5](#_Toc5589591)

[1.3课程设计的主要研究工作 5](#_Toc5589592)

[2系统需求分析与总体设计 6](#_Toc5589593)

[2.1 系统需求分析 6](#_Toc5589594)

[2.2系统总体设计 6](#_Toc5589595)

[3系统详细设计 8](#_Toc5589596)

[3.1有关数据结构的定义 8](#_Toc5589597)

[3.2主要算法设计 10](#_Toc5589598)

[4系统实现与测试 14](#_Toc5589599)

[4.1系统实现 14](#_Toc5589600)

[4.2系统测试 15](#_Toc5589601)

[5总结与展望 20](#_Toc5589602)

[5.1全文总结 20](#_Toc5589603)

[5.2工作展望 20](#_Toc5589604)

[6 体会 21](#_Toc5589605)

# 1引言

### 1.1课题背景与意义

对于计算机科学与技术、信息安全与物联网专业大二学生，在前三个学期已经学习了C语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及C语言程序设计实验、数据结构实验两门编程实践课程，不仅具有较为系统性的C语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“综合程序设计”课程设计正是使大家从简单验证到综合应用，甚至在编程中实现智慧与风格升华的重要实践环节，为后续学习与进行计算机系统编程打下坚实的基础，让综合编程技能成为大家的固有能力与通向未来专业之门的钥匙。

**1.1.1 课程目标**

基于“综合程序设计”课程设计实践课程规划原则及其在计算机相关专业人才培养中的地位，其应该体现与达到如下目标：

⑴综合性训练目标：在该课程中涉及C语言的主要编程要素，如典型的数据类型与控制结构；覆盖多种典型的数据结构如线性结构、二叉树与树结构、图结构及查找表结构等。从先前实验课的单要素或单一结构训练向多要素，多结构综合应用训练转变。

⑵培养应用问题的求解能力：程序设计是为问题求解服务的，提高对应用问题进行分析，数据抽象与建模，及问题定义与功能划分等综合分析与表示能力。

⑶程序编写向程序设计转化：在实验课程中，老师基本描述了相关数据结构，程序框架及主要算法，基于此进行程序编写训练，其属于验证与复现性编程实践。“综合程序设计”要求同学们基于对应用问题的分析，建立求解模型，设计数据结构与主要算法，从而进行程序设计，更多地体现“设计”的内涵与份量。

⑷进一步培养编程规范性与工程化素养：通过“综合程序设计”实践进一步培养良好的规范性编程习惯，以及一定的程序设计与软件开发的工程化素养，按照问题定义、必要的需求分析、系统设计、编程实现、程序测试分析及编制综合程序设计课程设计报告的流程组织本实践课程的开展与进行，形成初步的工程化程序设计素养。

**1.1.2 课程任务**

在选择与确定了“综合程序设计”课程设计问题之后，按工程化的基本流程分别完成如下任务：

⑴阅读“综合程序设计”课程设计任务书，熟悉问题，查阅文献，了解问题背景及相关知识。

⑵对设计问题进行需求分析，分析问题中所涉及的数据对象，划分功能，人机交互需求与数据文件读写等，并对问题进行形式化表示。

⑶基于上述需求分析，进行系统设计，明确程序的模块结构；设计数据结构（逻辑结构及其物理结构），参考并设计主要子问题的求解算法。

⑷程序实现，基于系统设计，制定相应的实现方案，编写各程序模块，完成程序编写与调试任务。

⑸程序测试，设计测试用例对程序进行功能测试，性能测量及理论分析。

⑹程序优化，对设计方案中的结构，算法进行一定优化，测试与分析性能改善结果。

⑺设计总结，按规范化要求撰写“综合程序设计”课程设计报告。

⑻成果提交：将程序源代码/工程文件、可独立运行的可执行程序、简要操作手册及“综合程序设计”课程设计报告电子版打包，文件夹名称格式为“专业班级-学号姓名”,如：CS1702-U201714999李某某。并将设计报告打印为纸质版（A4双面打印），然后以班为单位在指定时间（一般在设计课结束后两周内）集体提交到指导老师。

后面将对一个或多个候选设计问题进行问题与要求描述及设计指导，每个同学选择其中一题作为自己的综合程序设计课程设计问题。

### 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[1]，并在 2001 年和 2007 年先后 对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出 对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。 SAT 解决器的实现是我们关心的主要问题，目前的 SAT 算法大致可以归结为 两大类：完备算法（也称回溯搜索算法）和局部搜索算法。其中，完备算法大都 是基于回溯搜索的，局部搜索算法是基于局部随机搜索的。完备算法基于穷举法 思想，它的优点是能保证找到对应 SAT 问题的解或证明公式不可满足，但是效率 极低，它的平均时间复杂度虽是多项式级的，但是最坏情况下的时间复杂度却是 指数级的。一些完备算法采用了精巧的技术来减小搜索空间和问题规模，提高了 算法的时间效率，这一类算法将是本文研究的重点。局部搜索算法相对于完备算 法而言，由于采用了启发式策略来指导搜索，使得求解速度相对较快，但是在某 些实例上可能得不到解，它不保证一定能够找到对应 SAT 问题的解，即它不能证 明 SAT 问题的不可满足性。局部搜索算法的研究热潮是在最近几年才兴起的，本 文没有重点研究这类算法，而只是将其作为一个必要步骤应用到了完备算法的求 解过程中。 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 人在 1960 年提出[3]，其它的完备算法大都是在 DPLL 算法的基础上衍生出来的， 是对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂 度是指数级别，最初这使得许多的研究者望而却步。而后，S.A.Cook 在 1971 年证 明了 SAT 问题是 NP 完全问题，这更加削弱了许多学者研究 SAT 问题的兴趣，从 而导致了 SAT 问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢， 研究成果较少。但是 1996 年以后，很多国家都相继举办了一些 SAT 竞赛和研讨会，

这使得越来越多的人开始关注并研究 SAT 问题，所以这段时间也涌现出了众多新 的高效的 SAT 算法如 MINISAT[2]、SATO[4]、CHAFF[5]、POSIT[6]和 GRASP[7]等， SAT 算法的研究成果显著，求解算法也越来越多地应用到了实际问题领域。这些 新兴的算法大都是基于 DPLL 算法的改进算法，改进的方面包括：采用新的数据 结构、新的变量决策策略或者新的快速的算法实现方案。国内也涌现出了许多高 效的求解算法，如 1998 年作者梁东敏提出了改进的子句加权 WSAT 算法[8]，2000 年金人超和黄文奇提出的并行 Solar 算法[9]，2002 年作者张德富在文献[10]中，提 出模拟退火算法。

### 1.3课程设计的主要研究工作

本设计要求精心设计问题中变元、文字、子句、公式等有效的物理存储结构，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。在此基础上，通过改进算法、使用不同的选择策略和物理存储结构来优化求解过程，使求解器能更快地求解。完成基于此算法的数独游戏，通过将数独归约成SAT问题进行求解。

# 2系统需求分析与总体设计

### 2.1 系统需求分析

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

1. 能够读取cnf文件，输出执行结果，并保存执行结果

2. 能够对计算出的结果进行正确性的验证

3. 给出求解时间

（2） 数独游戏。经数独游戏转化为SAT问题进行求解，并具有一定的可玩性和交互性。有以下要求

1. 能够生成数独格局

2. 能够将数独问题转化为SAT问题并导出cnf文件

3. 能够将SAT的求解结果转化为数独的解

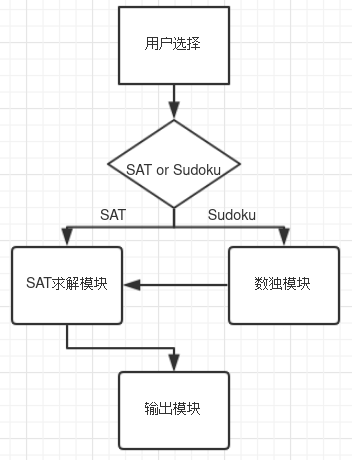
这部分应该写的是用户需求，明确你做的系统要实现的目标，能处理一些什么样的事务、事务处理流程等。

### 2.2系统总体设计

系统主要有三个模块

1. SAT求解模块
   1. 读取cnf文件并将其转化为设计的数据结构。
   2. 可以查看查看读入的cnf文件以及相关信息(如字句数，变元数等等)。
   3. 根据DPLL求解以循环而不是递归的形式求解出结果。
2. 输出模块
   1. 将SAT求解模块所求得的内容输出到控制台（可满足性、时间、解）
   2. 将最终的结果以文件的形式保存。
3. 数独模块
   1. 生成数独终局
   2. 根据“挖洞法”生成数独
   3. 将数独转化为SAT问题进行求解
   4. 将求解结果反向转化为数独的解
   5. 由于此模块使用Qt完成，所以具有一定的交互性和可玩性

系统模块结构图如图2-1所示:



**图2-1 系统模块结构图**

# 3系统详细设计

### 3.1有关数据结构的定义

* + 1. **数据定义**

1. 变元单元 Var

Int 型positiveCount用于储存该变元在剩余的自居中所出现为正的总次数

Int 型nagativeCount用于储存该变元在剩余的自居中所出现为负的总次数

intListHead \* 型 positive 用于储存该变元所在为正的字句的序号

intListHead \* 型 negative 用于储存该变元所在为负的字句的序号

bool 型 hanled 用于储存该变量是否已经被赋值

（intListHead 为自己定义的整型链表头）

1. 子句单元 Clause

Int 型 length 用于储存子句的长度

intListHead \* 型 variables 用于储存子句所包含的变元

int 型 hidden 用于标记该子句是否被删除，以及是由于哪一个变元的取值引起的

1. 整个cnf结构 CNF

Int 型 varNum 储存CNF结构中的变元数目

Int 型 clauseNum 储存CNF结构中的子句数目

Var \* 类型 variables 作为变元数组保存了CNF结构中的所有变元

Clause \* 类型 clauses 作为子句数组保存了CNF结构中的所有子句

1. 决策树节点 DecisionTreeNode

Int 型 assumption 当前节点所作出的假设，表示某一个变元的某一个取值为真

Bool 型 valid 用于标记当前节点的假设是通过单子句规则所“推理”出来的还是单纯的“选取变元”得出的，1表示前者，0表示后者

DecisionTreeNode \* 型 next 与 back 分别表示后一个决策树节点以及前一个决策树节点

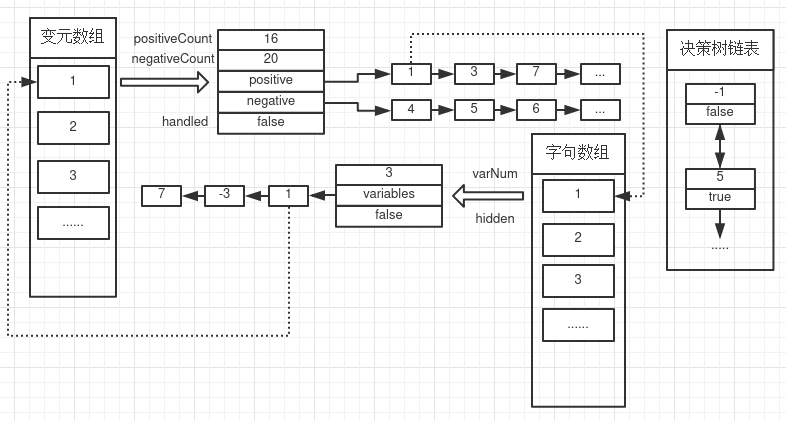
以上所有数据以及数据类型等信息如表3-1 所示

**表3-1**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据 | 数据类型 | 数据项名称 | 作用 |
| 变元单元 Var | Int | positiveCount | 取正的个数 |
| nagativeCount | 取负的个数 |
| intListHead \* | positive | 正变元所在子句序号 |
| negative | 负变元所在子句序号 |
| bool | handled | 变元是否被处理 |
| 子句单元 Clause | int | length | 子句长度 |
| intListHead \* | variables | 所含变元以及正负 |
| int | hidden | 是否被删除以及原因 |
| 整个cnf结构 CNF | int | varNum | 变元个数 |
| clauseNum | 子句个数 |
| Var \* | variables | 变元数组 |
| Clause \* | clauses | 子句数组 |
| 决策树节点 DecisionTreeNode | int | assumption | 假设 |
| bool | valid | 是“推理”还是“假设” |
| DecisionTreeNode \* | next | 下一个节点 |
| back | 上一个节点 |

* + 1. **数据关联**

以上数据之间的关联关系如图3-1所示



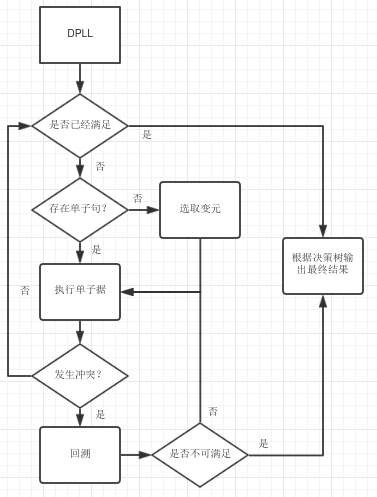
**图3-1 数据关联图**

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

### 3.2主要算法设计

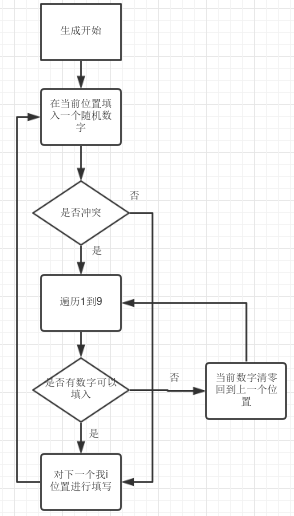
* + 1. **SAT求解模块主要算法设计**

1. 读取cnf 文件，利用 fgetc 获得开始每一行的第一个字符，判断是否是注释，并在到达数据区后进行数据的读取和保存，返回一个指向CNF结构体的指针
2. DPLL算法使用循环而非递归结构，每一次循环判断判断是否有单子句，如果没有则根绝选取策略选取变元。如果有单子句则执行单子句规则，直到没有单子句或CNF满足为止，进行下一轮循环。在执行的过程中可以同时判断当前假设是否造成了冲突(由于是循环，所以冲突必然是由当前假设所造成的，只需要检测执行单子句规则的时候是否出现了长度为0的子句)。注意在执行单子句过程之中以及回溯过程之中并没有对CNF结构体做出真正的删除操作，而是通过标记的方式来记录变元以及子句之间的关系，回溯的时候可以通过决策树DecisionTree所保留的数据来进行恢复。同时子句和变元都采用了数组进行储存，使得可以通过直接寻址快速找到变元所在的子句以及在子句之中的正负情况。流程图如图3-2所示



**图3-2 DPLL主要部分流程图**

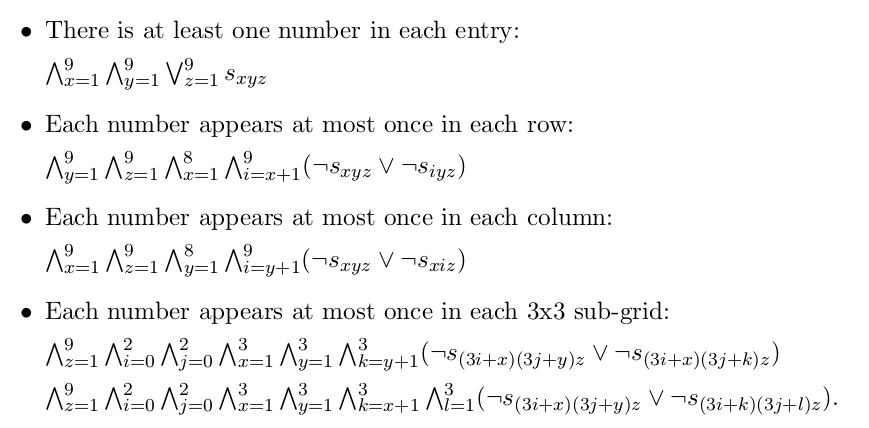
1. 将最终的决策树头节点返回，并导出结果，保存为文件。而运算的时间则在主函数中通过调用clock()函数(Windows)或gettimeofday()函数(Linux)来获得
   * 1. **数独模块主要算法设计**
2. 生成终局，从左到右从上到下，依次使用srand以及rand函数生成1到9的某一个随机数字，并检查是否存在冲突，若存在，则递增2(若超过9则从头开始，递增量不一定非要是2，只要能够使其遍历1到9即可)知道遍历1到9的每一个数字，如果依然存在冲突，则进行回溯，如果有数字可以填入，那么就继续递归。流程图如图3-3所示



**图3-3 生成数独流程图**

1. 使用挖洞法生成数独，随机生成坐标位置x、y，并挖掉该位置的数，如果该位置已经为空，则继续寻找，直到满足要求的数目。在挖的过程中需要考虑解的唯一性。
2. 将数独规约成SAT问题，在位置x,y填入z记为一个变元S，需要一个从三元组到整数的映射，此处选择了九进制的办法即

S = x\*9^0+y\*9^1+z\*9^2 + 1，再根据数独的基本规则，生成子句，如图3-4所示



**图3-4 数独生成子句规则**

1. 人机交互以及用户操作，这一部分使用Qt完成，具体请参考系统测试部分

这部分主要描述系统中的模块实现的流程，可采用文字配合流程图的方式表示各模块的算法思想及流程。

# 4系统实现与测试

### 4.1系统实现

系统环境：arch Linux 5.0.5 gcc version 8.2.1 Qt5.12.1 内存16G

* + 1. 函数定义以及其说明

1. SAT求解器

void addElem(int target, intListHead \* head);初始导入文件时添加元素到整形链表

DecisionTreeNode \* addAssumption(int assumption, bool valid, DecisionTreeHead \* head); 用于添加决策树节点

void InitIntListHead(CNF \* cnf);初始化

int LoadFile(const char \* filename, CNF \* cnf);导入文件生成CNF

void CheckFinalResult(int \* resultArray, CNF \* cnf);检测最终是否成立

void DestroyList(intListHead \* head);

void DestroyCNF(CNF \* cnf);

void DisplayData(CNF \* cnf);用于销毁

void writeFile(int \* resultArray, int length, float runtime, const char \*filename);用于写入文件

具体函数之间的调用关系不再赘述。

1. 数独部分

inline int HASH(int i, int j, int n); 用于将三元素转化为变元

void fillIn(int \* resultArray, int(\*Sudoku)[SUDOKU\_LENGTH]);将最终的结果填入数独

void printSudoku(int(\*Sudoku)[SUDOKU\_LENGTH]);打印数独

bool checkDesk(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y);用于在生成数独的时候检查当前填入的数字是否引起了冲突

bool generate(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y);用于生成数独终局

void Reduction(int(\*Sudoku)[SUDOKU\_LENGTH], const char \* filename);将数独转化为SAT，也即cnf文件

void Dig(int(\*Sudoku)[SUDOKU\_LENGTH], int level, const char \* tempfileName);挖洞

这部分应该写的是用户需求，明确你做的系统要实现的目标，能处理一些什么样的事务、事务处理流程等。

这部分可首先叙述一下你的系统实现的软硬件环境；

根据3.1的设计，用C语言定义各种数据类型；

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

### 4.2系统测试

由于数独与SAT完全分离，故分别测试

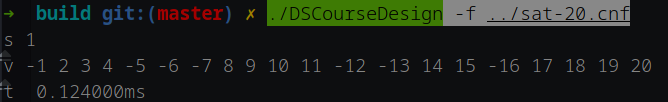
* + 1. SAT测试

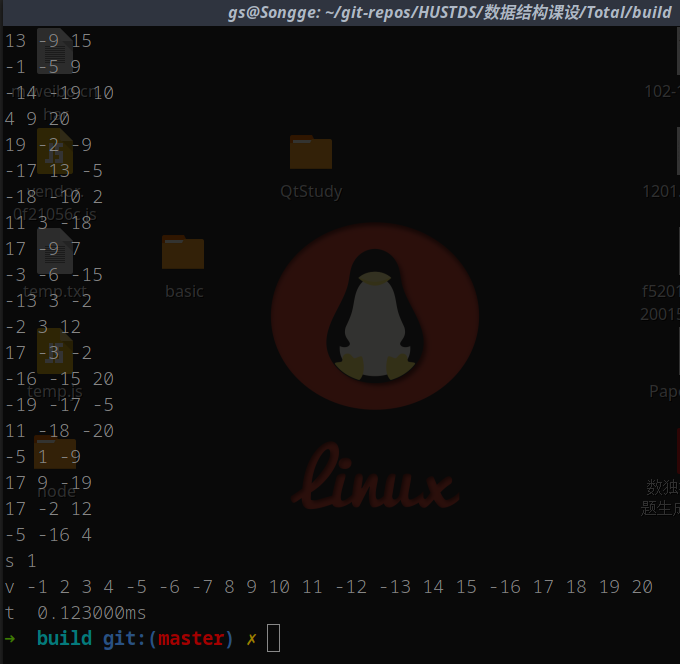
功能简述 命令行参数 -s 表示保存结果为文件，-p 表示打印出内部的结构 -f 后接待求解的文件路径

|  |  |  |
| --- | --- | --- |
| 输入 | 预期输出 | 实际输出 |
| ./DSCourseDesign | 提示需要输入文件名 | 如图4-1所示 |
| ./DSCourseDesign -f | 提示需要输入文件名 | 如图4-2所示 |
| ./DSCourseDesign -f dontexist.cnf | 提示文件不存在 | 如图4-3所示 |
| ./DSCourseDesign -f ../sat-20.cnf | 输入求解结果 | 如图4-4所示 |
| ./DSCourseDesign -p -f ../sat-20.cnf | 输出求解结果，并显示内部储存结构 | 如图4-5所示 |
| ./DSCourseDesign -s -f ../sat-20.cnf | 输出求解结果，并保存结果到文件 | 如图4-6所示 |

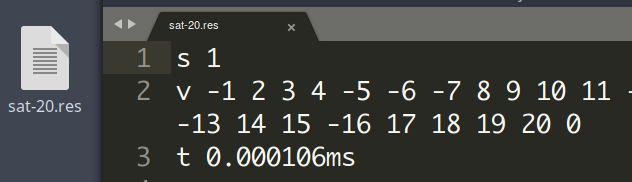
**图4-1测试1**

**图4-2 测试2**

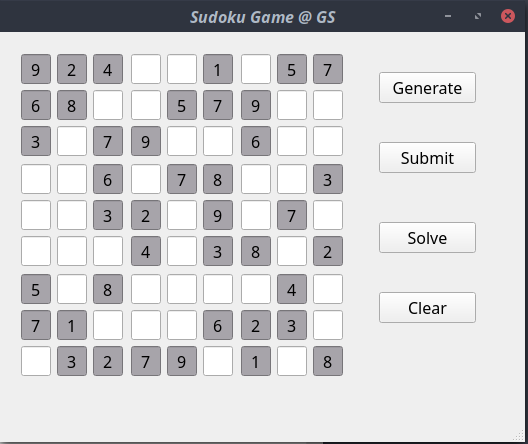
**图4-3 测试3**

**图4-4 测试4**

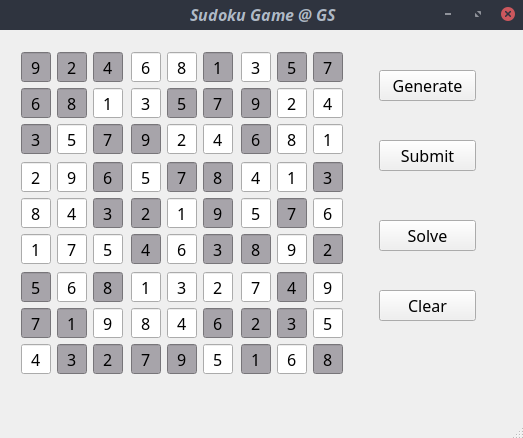
**图4-5 测试5**

**图4-6 测试6**

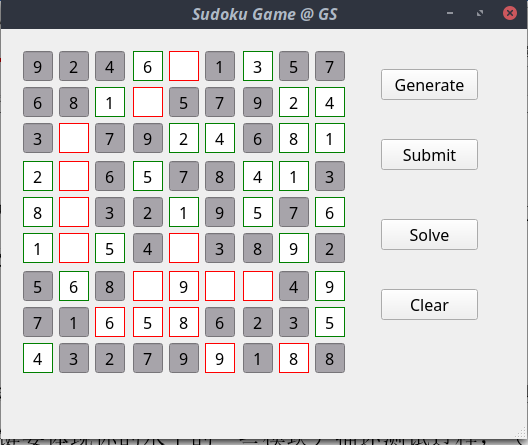
* + 1. 数独测试

生成终局:如图4-7所示

**图4-7 生成终局**

求解数独：如图4-8所示

**图4-8 生成终局**

用户交互：用户可以填写，提交，以及清除所有已填位置，如图4-9所示

**图4-9 用户交互测试**

* + 1. DPLL优化对比

优化主要有两个部分，一个是数据结构的优化，另一个是选取变元的优化，其中数据结构的并没有采用原任务书中所说的那种数据结构。另一方面，选取变元策略主要有以下4种，出现次数最多取正，出现次数最多取负，顺序选取和逆序选取。各种选取方式的对比，如图4-10所示



**图4-10 不同选取策略的对比**

可以看出其实并没有哪一种可以称之为普适的好方法，同时我们也可以发现顺序选取和逆序选取非常不稳定，对于数独类的CNF求解起来非常快，但是对于其他类型的就很慢。另一方面出现次数最多取正和出现次数最多取负是两种不同的解决角度，前者是想要尽快求解出正确的解，每一次选取变元都会使尽可能多的子句被消除，但是后者是想要尽快发生冲突，进而剪除搜索分支，每一次选取变元都会使尽可能多的子句长度减少。从表中可以看到，后者在大部分情况要比前者快一些。

# 5总结与展望

### 5.1全文总结

主要工作如下：

1. 在DPLL的基础上，实现了对于简单SAT问题的求解，其中仔细斟酌并尝试使用了多种数据结构以及选取变元的策略以及背后的原因。
2. 完成了数独的生成、转化以及求解并有了一个可以简单交互的界面

### 5.2工作展望

1. 整个SAT求解器可以说还是处于一种非常初级的状态，对于大型算例基本上无能为力，而本人在网络上找到的专业的求解器连满足算例中最大的CNF都可以跑出来。所以DPLL的过程可能还需要进一步改善，比如回溯的时候采用非时间回溯，或者加入子句学习策略以及多线程的使用，都可以进一步加快运行速度。
2. 数独游戏还无法非常快的生成唯一解的盘局，也没有显示分数等等功能，还需要进一步完善。

# 6 体会

这一次的课设可以说是用C写过的最大的项目了，整体上还是花了不少时间和经历的，尤其是操作系统的迁移带来了不少麻烦。最大的体会有以下几点：

1. 程序的模块化真的非常重要，将不同的函数功能分开实现有助于代码之间的解耦，尤其是切换变元选取策略的时候非常方便
2. 在写比较大的项目的时候要做好前期准备工作，比如多查阅相关资料以及论文，能够大大减少走弯路的可能性

**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

**附录**

basic.h

#ifndef BASIC\_H

#define BASIC\_H

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <time.h>

#include <string.h>

#include <sys/time.h>

#define DISCARD\_BUFFER\_SIZE 100

struct intListNode {

int value;

intListNode \* next;

};

struct intListHead

{

intListNode \* first;

int length = 0;

};

struct Var

{

//int index; // ??????????????

int positiveCount; // ????????

int negativeCount; // ????????

intListHead \* positive;

intListHead \* negative;

bool handled; // ?????????????????????????FindMax????????

};

struct Clause

{

int length;

intListHead \* variables; // ??????????????��????��??

int hidden; // ???????????????????????

};

struct CNF {

int varNum;

int clauseNum;

Var \* variables;

Clause \* clauses;

};

struct DecisionTreeNode

{

int assumption;

bool valid;

DecisionTreeNode \* next;

DecisionTreeNode \* back;

};

struct DecisionTreeHead

{

DecisionTreeNode \* firstNode;

};

inline void deleteElem(int target, intListHead \* head) {

--head->length;

}

inline void recoverElem(int target, intListHead \* head) {

++head->length;

}

void addElem(int target, intListHead \* head);

void printAssumptions(DecisionTreeHead \* head);

DecisionTreeNode \* addAssumption(int assumption, bool valid, DecisionTreeHead \* head);

void InitIntListHead(CNF \* cnf);

int LoadFile(const char \* filename, CNF \* cnf);

void DisplayResult(DecisionTreeHead \* Head);

int \* TurnToArray(DecisionTreeHead \* result, CNF \* cnf);

void PrintArray(int \* resultArray, int length);

void CheckFinalResult(int \* resultArray, CNF \* cnf);

void DestroyList(intListHead \* head);

void DestroyCNF(CNF \* cnf);

void DisplayData(CNF \* cnf);

void writeFile(int \* resultArray, int length, float runtime, const char \*filename);

#endif

dpll.h

#ifndef DPLL\_H

#define DPLL\_H

#include "basic.h"

#include "selectvar.h"

bool Process(DecisionTreeNode \* node, CNF \* cnf);

void Recover(DecisionTreeNode \* node, CNF \* cnf);

DecisionTreeNode \* BackTrack(DecisionTreeNode \* node, CNF \* cnf);

DecisionTreeHead \* DPLL(CNF \* cnf);

bool Satisfied(CNF \* cnf);

int SingleRule(CNF \* cnf);

#endif // !DPLL\_H

selectVar.h

#ifndef SELECTVAR\_H

#define SELECTVAR\_H

#include "basic.h"

void updateCountNum(CNF \* cnf);

void updateCountNum\_2(CNF \* cnf);

int FindMax\_2(CNF \* cnf);

int FindMax(CNF \*cnf);

int Findmax\_new\_2(CNF \* cnf);

int ReverseOrder(CNF \* cnf);

int InOrder(CNF \* cnf);

int SelectVar(CNF \* cnf);

int LeastLength(CNF \* cnf);

#endif // !SELECTVAR\_H

sudoku.h

#ifndef SUDOKU\_H

#define SUDOKU\_H

#include "basic.h"

#define SUDOKU\_LENGTH 9

#define SUDOKU\_BOX 729

inline int HASH(int i, int j, int n);

void fillIn(int \* resultArray, int(\*Sudoku)[SUDOKU\_LENGTH]);

void printSudoku(int(\*Sudoku)[SUDOKU\_LENGTH]);

bool checkDesk(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y);

bool generate(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y);

void Reduction(int(\*Sudoku)[SUDOKU\_LENGTH], const char \* filename);

void display(int(\*Sudoku)[SUDOKU\_LENGTH]);

void Dig(int(\*Sudoku)[SUDOKU\_LENGTH], int level, const char \* tempfileName);

void Dig2(int(\*Sudoku)[SUDOKU\_LENGTH], int level, const char \* tempfileName);

#endif // !SUDOKU\_H

basic.cpp

#include "basic.h"

void addElem(int target, intListHead \* head) {

intListNode \* newNode = (intListNode \*)malloc(sizeof(intListNode));

assert(newNode != nullptr);

// old

newNode->value = target;

newNode->next = head->first;

head->first = newNode;

++head->length;

}

void InitIntListHead(CNF \* cnf) {

cnf->clauses = (Clause \*)malloc((cnf->clauseNum + 1) \* sizeof(Clause));

cnf->variables = (Var \*)malloc((cnf->varNum + 1) \* sizeof(Var));

assert(cnf->clauses != nullptr && cnf->variables != nullptr);

for (int i = cnf->clauseNum; i > 0; --i) {

cnf->clauses[i].length = 0;

cnf->clauses[i].variables = (intListHead \*)malloc(sizeof(intListHead));

assert(cnf->clauses[i].variables != nullptr);

cnf->clauses[i].variables->length = 0;

cnf->clauses[i].hidden = 0;

cnf->clauses[i].variables->first = nullptr;

}

for (int i = cnf->varNum; i > 0; --i) {

cnf->variables[i].negative = (intListHead \*)malloc(sizeof(intListHead));

cnf->variables[i].positive = (intListHead \*)malloc(sizeof(intListHead));

assert(cnf->variables[i].negative != nullptr);

assert(cnf->variables[i].positive != nullptr);

cnf->variables[i].negative->length = 0;

cnf->variables[i].positive->length = 0;

cnf->variables[i].negative->first = nullptr;

cnf->variables[i].positive->first = nullptr;

cnf->variables[i].handled = false;

}

}

void printAssumptions(DecisionTreeHead \* head) {

system("cls");

DecisionTreeNode \* tempNode = head->firstNode;

while (tempNode)

{

printf("%d", tempNode->assumption);

if (!tempNode->valid)

printf(" false");

else

printf(" true ");

printf("\t");

tempNode = tempNode->next;

}

}

DecisionTreeNode \* addAssumption(int assumption, bool valid, DecisionTreeHead \* head) {

DecisionTreeNode \* tempNode = head->firstNode;

if (tempNode == nullptr) {

// ��ʱ��û���κ�assumption

head->firstNode = (DecisionTreeNode \*)malloc(sizeof(DecisionTreeNode));

assert(head->firstNode != nullptr);

head->firstNode->back = nullptr;

head->firstNode->next = nullptr;

head->firstNode->assumption = assumption;

head->firstNode->valid = valid;

return head->firstNode;

}

while (tempNode->next)

tempNode = tempNode->next;

// �ҵ����һ���ڵ�

tempNode->next = (DecisionTreeNode \*)malloc(sizeof(DecisionTreeNode));

assert(tempNode->next != nullptr);

tempNode->next->back = tempNode;

tempNode->next->next = nullptr;

tempNode->next->assumption = assumption;

tempNode->next->valid = valid;

return tempNode->next;

}

int LoadFile(const char \* filename, CNF \* cnf) {

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

if (nullptr == fp) {

return -1;

}

else {

char firstChar = fgetc(fp);

while ('c' == firstChar) {

// ����� c ˵����ע�ͣ�ֱ��������һ��

char \* discardBuffer = (char \*)malloc(DISCARD\_BUFFER\_SIZE \* sizeof(char));

fgets(discardBuffer, DISCARD\_BUFFER\_SIZE, fp);

firstChar = fgetc(fp);

free(discardBuffer);

discardBuffer = nullptr;

}

if ('p' == firstChar) {

char \* discardBuffer = (char \*)malloc(DISCARD\_BUFFER\_SIZE \* sizeof(char));

fscanf(fp, "%s", discardBuffer); // ����cnf �ַ���

fscanf(fp, "%d", &cnf->varNum); // ��ȡ��Ԫ��

fscanf(fp, "%d", &cnf->clauseNum); // ��ȡ�Ӿ���

free(discardBuffer);

discardBuffer = nullptr;

}

// �����һ���ռ䣬�������

InitIntListHead(cnf);

// �滻

int clauseLength;

int totalLength = cnf->clauseNum;

for (int i = 1; i <= totalLength; ++i) {

clauseLength = 0;

int temp; // �����ȡ��������

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

while (temp)

{

if (temp > 0) {

addElem(i, cnf->variables[temp].positive);

addElem(temp, cnf->clauses[i].variables);

}

else {

addElem(i, cnf->variables[-temp].negative);

addElem(temp, cnf->clauses[i].variables);

}

++clauseLength;

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

}

cnf->clauses[i].length = clauseLength;

assert(cnf->clauses[i].length >= 0);

}

}

//fclose(fp);

}

void DisplayResult(DecisionTreeHead \* Head) {

DecisionTreeNode \* tempNode = Head->firstNode;

while (tempNode)

{

printf("%d ", tempNode->assumption);

tempNode = tempNode->next;

}

}

int \* TurnToArray(DecisionTreeHead \* result, CNF \* cnf) {

int \* resultArray = (int \*)calloc((cnf->varNum + 1), sizeof(int));

assert(resultArray != nullptr);

DecisionTreeNode \* tempNode = result->firstNode;

while (tempNode)

{

resultArray[abs(tempNode->assumption)] = tempNode->assumption;

tempNode = tempNode->next;

}

return resultArray;

}

void PrintArray(int \* resultArray, int length) {

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

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

}

void CheckFinalResult(int \* resultArray, CNF \* cnf) {

intListNode \* tempNode = nullptr;

for (int i = 1; i <= cnf->clauseNum; ++i) {

bool flag = false;

tempNode = cnf->clauses[i].variables->first;

while (tempNode)

{

if (resultArray[abs(tempNode->value)] \* tempNode->value > 0)

flag = true;

tempNode = tempNode->next;

}

if (!flag)

printf("Clause %d gg\n", i);

}

}

void DestroyList(intListHead \* head) {

if (!head) {

intListNode \* tempNode = head->first;

intListNode \* tempNodeFree = nullptr;

while (tempNode) {

tempNodeFree = tempNode;

tempNode = tempNode->next;

free(tempNodeFree);

}

head->first = nullptr;

free(head);

head = nullptr;

tempNode = tempNodeFree = nullptr;

}

}

void DestroyCNF(CNF \* cnf) {

for (int i = cnf->varNum - 1; i >= 0; --i) {

DestroyList(cnf->variables[i].positive);

DestroyList(cnf->variables[i].negative);

cnf->variables[i].positive = cnf->variables[i].negative = nullptr;

}

free(cnf->variables);

cnf->variables = nullptr;

for (int i = cnf->clauseNum - 1; i >= 0; --i) {

DestroyList(cnf->clauses[i].variables);

cnf->clauses[i].variables = nullptr;

}

free(cnf->clauses);

cnf->clauses = nullptr;

free(cnf);

}

void writeFile(int \* resultArray, int length, float runtime, const char \*filename){

char resultfilename[100];

int i = 0;

while('\0' != (resultfilename[i] = filename[i])){

printf("%c", resultfilename[i]);

i++;

}

resultfilename[i-1] = 's';

resultfilename[i-2] = 'e';

resultfilename[i-3] = 'r';

/\*i = 0;

while('\0' != resultfilename[i]){

printf("%c", resultfilename[i]);

i++;

}\*/

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

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

for (int i = 1; i <= length; ++i){

fprintf(fp, "%d ", resultArray[i]);

}

fprintf(fp, "\nt %fms\n", runtime);

fclose(fp);

}

void DisplayData(CNF \*cnf) {

for (int i = 1; i <= cnf->clauseNum; ++i){

intListNode \* temp = cnf->clauses[i].variables->first;

while(temp){

printf("%d ", temp->value);

temp = temp->next;

}

printf("\n");

}

}

dpll.cpp

#include "dpll.h"

#include "selectvar.h"

bool Process(DecisionTreeNode \* node, CNF \* cnf) {

bool flag = true; //��־�Ƿ���ֿ��Ӿ�,trueΪû�г��֣�falseΪ������

int assumption = node->assumption;

int currentClauseIndex = 0;

intListNode \* positiveClauseIndexPtr = nullptr;

intListNode \* negativeClauseIndexPtr = nullptr;

if (assumption > 0) {

positiveClauseIndexPtr = cnf->variables[assumption].positive->first;

negativeClauseIndexPtr = cnf->variables[assumption].negative->first;

cnf->variables[assumption].handled = true;

}

else {

negativeClauseIndexPtr = cnf->variables[-assumption].positive->first;

positiveClauseIndexPtr = cnf->variables[-assumption].negative->first;

cnf->variables[-assumption].handled = true;

}

// ��ɾ���Ӿ䡢��ɾ�����֣����ܳ�������ͬʱ������ͬһ���Ӿ�����

while (positiveClauseIndexPtr)

{

currentClauseIndex = positiveClauseIndexPtr->value;

// ȡ�������Ӿ�ɾ��

if (!cnf->clauses[currentClauseIndex].hidden) {

cnf->clauses[currentClauseIndex].hidden = assumption;

}

positiveClauseIndexPtr = positiveClauseIndexPtr->next;

}

currentClauseIndex = 0;

while (negativeClauseIndexPtr)

{

currentClauseIndex = negativeClauseIndexPtr->value;

// ȡ�������Ӿ��еĶ�Ӧ����ɾ��

if (!cnf->clauses[currentClauseIndex].hidden) {

assert(cnf->clauses[currentClauseIndex].variables->length >= 0);

deleteElem(-assumption, cnf->clauses[currentClauseIndex].variables);

--cnf->clauses[currentClauseIndex].length; // �Ӿ䳤�ȼ���

if (0 == cnf->clauses[currentClauseIndex].length) {

// ˵�������˿��Ӿ�

flag = false;

}

}

negativeClauseIndexPtr = negativeClauseIndexPtr->next;

}

return flag;

}

void Recover(DecisionTreeNode \* node, CNF \* cnf) {

int assumption = node->assumption;

int currentClauseIndex = 0;

intListNode \* positiveClauseIndexPtr = nullptr;

intListNode \* negativeClauseIndexPtr = nullptr;

if (assumption > 0) {

positiveClauseIndexPtr = cnf->variables[assumption].positive->first;

negativeClauseIndexPtr = cnf->variables[assumption].negative->first;

cnf->variables[assumption].handled = false; // �ָ�״̬

}

else {

negativeClauseIndexPtr = cnf->variables[-assumption].positive->first;

positiveClauseIndexPtr = cnf->variables[-assumption].negative->first;

cnf->variables[-assumption].handled = false;

}

// �� Process�෴���Ȼָ����֣��ٻָ�ɾ��

while (negativeClauseIndexPtr)

{

currentClauseIndex = negativeClauseIndexPtr->value;

// ȡ�������Ӿ��еĶ�Ӧ���ָֻ�

if (!cnf->clauses[currentClauseIndex].hidden) {

recoverElem(-assumption, cnf->clauses[currentClauseIndex].variables);

++cnf->clauses[currentClauseIndex].length;// �Ӿ䳤�Ȼָ�

}

negativeClauseIndexPtr = negativeClauseIndexPtr->next;

}

while (positiveClauseIndexPtr)

{

// ȡ��,����Ӿ��ɾ�����ɵ�ǰassumption���µģ���ָ�

currentClauseIndex = positiveClauseIndexPtr->value;

if (cnf->clauses[currentClauseIndex].hidden == assumption)

cnf->clauses[currentClauseIndex].hidden = 0;

positiveClauseIndexPtr = positiveClauseIndexPtr->next;

}

}

DecisionTreeNode \* BackTrack(DecisionTreeNode \* node, CNF \* cnf) {

while (node->valid && node->back)

{

DecisionTreeNode \* tempPtr = node;

Recover(node, cnf);

node = node->back;

node->next = nullptr;

free(tempPtr);

tempPtr = nullptr;

}

if (!node->valid) {

// ��ʱ����ΪvalidΪfalse������ѭ����˵���ҵ��˼����

Recover(node, cnf);

node->assumption = -node->assumption;

node->valid = true;

return node;

}

else {

// ��ʱ����ʹ���ݵ��ʼҲ�޷�ʹCNFΪ�棬����NULL

return nullptr;

}

}

bool Satisfied(CNF \* cnf) {

// ���CNF�Ƿ��Ѿ�����

int flag = true;

// ������ڻ�δ��ɾ�����Ӿ���˵��û������

for (int i = cnf->clauseNum; i > 0; --i) {

if (!cnf->clauses[i].hidden) {

flag = false;

break;

}

}

return flag;

}

int SingleRule(CNF \* cnf) {

int result = 0;

intListNode \* tempNode = nullptr;

for (int i = cnf->clauseNum; i > 0; --i) {

if (1 == cnf->clauses[i].length && !cnf->clauses[i].hidden) {

tempNode = cnf->clauses[i].variables->first;

while (cnf->variables[abs(tempNode->value)].handled)

tempNode = tempNode->next;

result = tempNode->value;

break;

}

}

return result;

}

DecisionTreeHead \* DPLL(CNF \* cnf) {

DecisionTreeHead \* Head = (DecisionTreeHead \*)malloc(sizeof(DecisionTreeHead));

assert(Head != nullptr);

DecisionTreeNode \* currentTreeNode = nullptr;

Head->firstNode = nullptr;

bool flag = true;

int singleRuleResult = 0;

int assumption = 0;

while (!Satisfied(cnf))

{

while (flag && (singleRuleResult = SingleRule(cnf))) {

currentTreeNode = addAssumption(singleRuleResult, true, Head);

flag = Process(currentTreeNode, cnf);

if (!flag)

break;

}

if (flag) {

// ���ڲ����е��Ӿ���˳�

assumption = SelectVar(cnf);

if (assumption == 0)

return Head;

currentTreeNode = addAssumption(assumption, false, Head);

// ����

flag = Process(currentTreeNode, cnf);

}

else {

// ���ڴ��ڳ�ͻ���˳�

currentTreeNode = BackTrack(currentTreeNode, cnf);

if (nullptr == currentTreeNode)

return nullptr;

flag = Process(currentTreeNode, cnf);

}

}

return Head;

}

CourseDesign.cpp

#include "dpll.h"

#include "sudoku.h"

#include <unistd.h>

#include <getopt.h>

struct options

{

int p; // 命令行参数，表示打印内部表示

int s; // 命令行参数，表示是否保存文件

const char \* filename; // 命令行参数，用于输入文件名

}option;

int main(int argc, char \*\* argv) {

int opt;

option.s = 0;

option.p = 0;

option.filename = nullptr;

while ((opt = getopt(argc, argv, "spf:")) != -1) {

if (opt == 's')

option.s = 1;

else if (opt == 'p')

option.p = 1;

else if (opt == 'f')

option.filename = optarg;

}

CNF \* cnf = (CNF \*)malloc(sizeof(CNF));

assert(cnf != nullptr);

if (option.filename != nullptr){

if (-1 == LoadFile(option.filename, cnf)){

printf("文件不存在\n");

return 0;

}

}

else {

printf("请以 -f filename 输入文件路径\n");

return 0;

}

if (option.p){

DisplayData(cnf);

}

struct timeval start;

struct timeval end;

float timer;

gettimeofday(&start, nullptr);

DecisionTreeHead \* result = DPLL(cnf);

gettimeofday(&end, nullptr);

timer = end.tv\_sec - start.tv\_sec + (float)(end.tv\_usec - start.tv\_usec)/1000000; //计算start和end之间的时间差

if (result) {

printf("s 1\nv ");

int \* resultArray = TurnToArray(result, cnf); // 最终的结果数组

PrintArray(resultArray, cnf->varNum + 1);

printf("\nt %fms\n",timer \* 1000);

CheckFinalResult(resultArray, cnf);

if (option.s){

writeFile(resultArray, cnf->varNum + 1, timer, option.filename);

}

cnf = nullptr;

}

else {

printf("\nt %fms\n",timer \* 1000);

printf("0\n");

}

return 0;

}

selectVar.cpp

#include "selectvar.h"

void updateCountNum(CNF \* cnf) {

int count = 0;

intListNode \* tempNode = nullptr;

for (int i = cnf->varNum; i > 0; --i) {

count = 0;

tempNode = cnf->variables[i].negative->first;

while (tempNode)

{

if (!cnf->clauses[tempNode->value].hidden)

++count;

tempNode = tempNode->next;

}

cnf->variables[i].negativeCount = count;

tempNode = cnf->variables[i].positive->first;

count = 0;

while (tempNode)

{

if (!cnf->clauses[tempNode->value].hidden)

++count;

tempNode = tempNode->next;

}

cnf->variables[i].positiveCount = count;

}

}

void updateCountNum\_2(CNF \* cnf) {

int count = 0;

intListNode \* tempNode = nullptr;

for (int i = cnf->varNum; i > 0; --i) {

count = 0;

if (cnf->variables[i].handled) {

cnf->variables[i].negativeCount = cnf->variables[i].positiveCount = 0;

continue;

}

else {

tempNode = cnf->variables[i].negative->first;

while (tempNode)

{

if (!cnf->clauses[tempNode->value].hidden && cnf->clauses[tempNode->value].length == 2)

++count;

tempNode = tempNode->next;

}

cnf->variables[i].negativeCount = count;

tempNode = cnf->variables[i].positive->first;

count = 0;

while (tempNode)

{

if (!cnf->clauses[tempNode->value].hidden && cnf->clauses[tempNode->value].length == 2)

++count;

tempNode = tempNode->next;

}

cnf->variables[i].positiveCount = count;

}

}

}

int FindMax\_2(CNF \* cnf) {

int max = 0;

updateCountNum\_2(cnf);

int maxCount = 0;

//int maxCount = cnf->variables[max].positiveCount + cnf->variables[max].negativeCount;

for (int i = 1; i <= cnf->varNum; ++i) {

if (!cnf->variables[i].handled) {

int iCount = cnf->variables[i].positiveCount + cnf->variables[i].negativeCount;

if (iCount > maxCount) {

max = i;

maxCount = iCount;

}

}

}

if (cnf->variables[max].positiveCount > cnf->variables[max].negativeCount)

return max;

else

return -max;

}

int FindMax(CNF \*cnf) {

int max = 0;

updateCountNum(cnf);

int maxCount = 0;

//int maxCount = cnf->variables[max].positiveCount + cnf->variables[max].negativeCount;

for (int i = 1; i <= cnf->varNum; ++i) {

if (!cnf->variables[i].handled) {

int iCount = cnf->variables[i].positiveCount + cnf->variables[i].negativeCount;

if (iCount > maxCount) {

max = i;

maxCount = iCount;

}

}

}

if (cnf->variables[max].positiveCount > cnf->variables[max].negativeCount)

return max;

else

return -max;

}

int Findmax\_new\_2(CNF \* cnf) {

int max = 0; // �����Ӿ��г�������Ԫ�ص����

int max\_2 = 0; // ���г���Ϊ2���Ӿ��г�������Ԫ�ص����

int max\_count\_positive = 0;

int max\_count\_negative = 0;

int max\_count\_2\_negative = 0;

int max\_count\_2\_positive = 0;

int currentMax\_count = 0;

int currentMax\_count\_2 = 0;

intListNode \* tempNode;

for (int i = cnf->varNum; i > 0; --i) {

// ����

max\_count\_2\_negative = max\_count\_2\_positive = max\_count\_negative = max\_count\_positive = 0;

if (!cnf->variables[i].handled) {

// positive

tempNode = cnf->variables[i].positive->first;

while (tempNode) {

if (!cnf->clauses[tempNode->value].hidden) {

if (cnf->clauses[tempNode->value].length == 2)

++max\_count\_2\_positive;

++max\_count\_positive;

}

tempNode = tempNode->next;

}

// negative

tempNode = cnf->variables[i].negative->first;

while (tempNode) {

if (!cnf->clauses[tempNode->value].hidden) {

if (cnf->clauses[tempNode->value].length == 2)

++max\_count\_2\_negative;

++max\_count\_negative;

}

tempNode = tempNode->next;

}

// length >= 2

if (max\_count\_negative + max\_count\_positive > currentMax\_count) {

max = i;

currentMax\_count = max\_count\_negative + max\_count\_positive;

}

// length == 2

if (max\_count\_2\_negative + max\_count\_2\_positive > currentMax\_count) {

max\_2 = i;

currentMax\_count\_2 = max\_count\_2\_negative + max\_count\_2\_positive;

}

}

}

// ����ѡ�񳤶�Ϊ2���Ӿ��г��ִ�������Ԫ��

// ����ѡ����ִ�������ȡֵΪ��

if (currentMax\_count\_2) {

if (max\_count\_2\_negative > max\_count\_2\_positive)

return -max\_2;

else

return max\_2;

}

if (max\_count\_negative > max\_count\_positive)

return -max;

else

return max;

return 0;

}

int ReverseOrder(CNF \* cnf) {

int i = cnf->varNum;

while (cnf->variables[i].handled)

--i;

return i;

}

int InOrder(CNF \* cnf) {

int i = 1;

while (cnf->variables[i].handled)

++i;

if (i > cnf->varNum)

return 0;

else

return i;

}

int LeastLength(CNF \* cnf){

intListNode \* temp = nullptr;

int minCount = 1000;

int min = 0;

int currentCount = 0;

for (int i = 1; i <= cnf->varNum; ++i){

if (!cnf->variables[i].handled){

intListNode \* temp = cnf->variables[i].positive->first;

while(temp){

if (!cnf->clauses[abs(temp->value)].hidden)

currentCount += cnf->clauses[temp->value].length;

temp = temp->next;

}

temp = cnf->variables[i].negative->first;

while(temp){

if (!cnf->clauses[abs(temp->value)].hidden)

currentCount += cnf->clauses[temp->value].length;

}

if (currentCount <= minCount){

minCount = currentCount;

min = i;

}

currentCount = 0;

}

}

return min;

}

int SelectVar(CNF \* cnf) {

return FindMax(cnf);

//return InOrder(cnf);

//return LeastLength(cnf);

//return ReverseOrder(cnf);

}

sudoku.cpp

#include "sudoku.h"

#include "dpll.h"

void fillIn(int \* resultArray, int(\*Sudoku)[SUDOKU\_LENGTH]) {

int temp = 0;

int i, j, n;

i = j = n = 0;

for (int m = 1; m <= SUDOKU\_BOX; ++m) {

if (resultArray[m] > 0) {

temp = m-1;

n = temp % SUDOKU\_LENGTH;

temp = (temp - n) / SUDOKU\_LENGTH;

i = temp % SUDOKU\_LENGTH;

temp = temp - i;

j = temp / SUDOKU\_LENGTH;

++n; // n��Ҫ�����1

Sudoku[i][j] = n;

}

}

}

void printSudoku(int(\*Sudoku)[SUDOKU\_LENGTH]) {

for (int i = 0; i < SUDOKU\_LENGTH; ++i) {

for (int j = 0; j < SUDOKU\_LENGTH; ++j) {

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

}

printf("\n");

}

}

inline int HASH(int i, int j, int n) {

return n - 1 + 9 \* (i - 1) + 9 \* 9 \* (j - 1) + 1;

}

bool checkDesk(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y) {

int \* one\_to\_nine = (int \*)calloc(SUDOKU\_LENGTH + 1, sizeof(int));

// �����

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

if (one\_to\_nine[Sudoku[x][i]])

{

free(one\_to\_nine);

one\_to\_nine = nullptr;

return false;

}

else

one\_to\_nine[Sudoku[x][i]]++;

}

// ����

memset(one\_to\_nine, 0, sizeof(int)\*(SUDOKU\_LENGTH + 1));

// �����

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

if (one\_to\_nine[Sudoku[i][y]])

{

free(one\_to\_nine);

one\_to\_nine = nullptr;

return false;

}

else

one\_to\_nine[Sudoku[i][y]]++;

}

// ���Ź���

memset(one\_to\_nine, 0, sizeof(int)\*(SUDOKU\_LENGTH + 1));

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

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

if (Sudoku[i][j]) {

if (one\_to\_nine[Sudoku[i][j]])

{

free(one\_to\_nine);

one\_to\_nine = nullptr;

return false;

}

else

one\_to\_nine[Sudoku[i][j]]++;

}

}

}

free(one\_to\_nine);

one\_to\_nine = nullptr;

return true;

}

bool generate(int(\*Sudoku)[SUDOKU\_LENGTH], int x, int y) {

srand((unsigned)time(nullptr));

Sudoku[x][y] = rand() % 9 + 1; // ������һ���������

bool flag = true;

for (int i = 0; i < SUDOKU\_LENGTH; ++i) {

// ����Ƿ�����

if (flag == false) {

Sudoku[x][y] = Sudoku[x][y] % 9 + 1;

flag = true;

}

if (!checkDesk(Sudoku, x, y)) {

// ��������� �ͳ���+1

Sudoku[x][y] = (Sudoku[x][y] +1)% 9 + 1;

}

else {

if (x == SUDOKU\_LENGTH - 1 && y == SUDOKU\_LENGTH - 1) {

// ����ɹ����ҵ��������һ��,�˳�

return true;

}

else {

// ����ɹ��ˣ�����û�е����һ��Ԫ�أ�����д��һ��Ԫ��

if (generate(Sudoku, ((y == 8) ? x + 1 : x), (y + 1) % 9))

return true;

else if (i < SUDOKU\_LENGTH) {

flag = false;

}

else {

Sudoku[x][y] = 0;

return false;

}

}

}

}

Sudoku[x][y] = 0;

return false;

}

void Reduction(int(\*Sudoku)[SUDOKU\_LENGTH], const char \* filename) {

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

if (!fp) {

return;

}

else {

// дע��

int count = 0;

for (int i = 0; i < SUDOKU\_LENGTH; ++i) {

fprintf(fp, "c ");

for (int j = 0; j < SUDOKU\_LENGTH; ++j) {

fprintf(fp, "%d ", Sudoku[i][j]);

if (Sudoku[i][j])

count++;

}

fprintf(fp, "\n");

}

fprintf(fp, "p cnf %d %d\n", SUDOKU\_BOX, count + 8829);

for (int i = 1; i <= SUDOKU\_LENGTH; ++i) {

for (int j = 1; j <= SUDOKU\_LENGTH; ++j) {

if (Sudoku[i - 1][j - 1]) {

fprintf(fp, "%d 0\n", HASH(i, j, Sudoku[i - 1][j - 1]));

}

}

}

for (int i = 1; i <= SUDOKU\_LENGTH; ++i) {

for (int j = 1; j <= SUDOKU\_LENGTH; ++j) {

for (int n = 1; n <= SUDOKU\_LENGTH; ++n) {

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

}

fprintf(fp, "0\n");

}

}

for (int y = 1; y <= SUDOKU\_LENGTH; ++y) {

for (int z = 1; z <= SUDOKU\_LENGTH; ++z) {

for (int x = 1; x < SUDOKU\_LENGTH; ++x) {

for (int i = x + 1; i <= SUDOKU\_LENGTH; ++i) {

fprintf(fp, "%d %d 0\n", -HASH(x, y, z), -HASH(i, y, z));

}

}

}

}

// ÿһ��������һ��

for (int x = 1; x <= SUDOKU\_LENGTH; ++x) {

for (int z = 1; z <= SUDOKU\_LENGTH; ++z) {

for (int y = 1; y < SUDOKU\_LENGTH; ++y) {

for (int i = y + 1; i <= SUDOKU\_LENGTH; ++i) {

fprintf(fp, "%d %d 0\n", -HASH(x, y, z), -HASH(x, i, z));

}

}

}

}

//

for (int z = 1; z <= SUDOKU\_LENGTH; ++z) {

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

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

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

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

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

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

}

}

}

}

}

}

for (int z = 1; z <= SUDOKU\_LENGTH; ++z) {

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

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

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

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

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

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

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

}

}

}

}

}

}

}

}

fclose(fp);

}

void display(int(\*Sudoku)[SUDOKU\_LENGTH]) {

for (int i = 0; i < SUDOKU\_LENGTH; ++i) {

for (int j = 0; j < SUDOKU\_LENGTH; ++j) {

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

}

printf("\n");

}

}

void Dig(int(\*Sudoku)[SUDOKU\_LENGTH], int level, const char \* tempfileName) {

srand((unsigned)time(nullptr));

int x = rand() % 9;

int y = rand() % 9; // �������һ��λ��

CNF \* cnf = (CNF \*)malloc(sizeof(CNF));

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

while (!Sudoku[x][y]) {

x = rand() % 9;

y = rand() % 9;

}

int temp = Sudoku[x][y];

for (int i = 0; i < SUDOKU\_LENGTH - 1; ++i) {

cnf = (CNF \*)malloc(sizeof(CNF));

Sudoku[x][y] = (temp + i) % 9 + 1;

Reduction(Sudoku, tempfileName);

LoadFile(tempfileName, cnf);

if (DPLL(cnf)) {

++i;

Sudoku[x][y] = temp;

DestroyCNF(cnf);

cnf = nullptr;

break;

}

//DestroyCNF(cnf);

cnf = nullptr;

}

Sudoku[x][y] = 0;

}

}

void Dig2(int(\*Sudoku)[SUDOKU\_LENGTH], int level, const char \* tempfileName) {

srand((unsigned)time(nullptr));

int x = rand() % 9;

int y = rand() % 9; // �������һ��λ��

CNF \* cnf = (CNF \*)malloc(sizeof(CNF));

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

while (!Sudoku[x][y]) {

x = rand() % 9;

y = rand() % 9;

}

Sudoku[x][y] = 0;

}

}