

**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： 计算机类2003班**

**学 号： U202015371**

**姓 名： 范辰睿**

**指导教师： 卢萍**

**报告日期： 2021.8.25**

**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：**将数独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

* **参考文献**

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

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

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

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

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

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

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

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

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

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

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

目录

[任务书 I](#_Toc80781573)

[1 引言 5](#_Toc80781574)

[1.1 课题背景与意义 5](#_Toc80781575)

[1.2国内外研究现状 5](#_Toc80781578)

[1.3课程设计的主要研究工作 6](#_Toc80781579)

[2 系统需求分析与总体设计 7](#_Toc80781580)

[2.1 系统需求分析 7](#_Toc80781581)

[2.2 系统总体设计 7](#_Toc80781582)

[3 系统详细设计 9](#_Toc80781587)

[3.1 有关数据结构的定义 9](#_Toc80781588)

[3.2 主要算法设计 11](#_Toc80781594)

[4 系统实现与测试 16](#_Toc80781598)

[4.1系统实现 16](#_Toc80781599)

[4.2 系统测试 18](#_Toc80781604)

[5 总结与展望 32](#_Toc80781610)

[5.1 全文总结 32](#_Toc80781611)

[5.1 工作展望 32](#_Toc80781612)

[6 体会 33](#_Toc80781613)

[参考文献 34](#_Toc80781614)

[附录一：源程序 35](#_Toc80781615)

[未优化版 35](#_Toc80781616)

[优化版 54](#_Toc80781622)

[附录二：程序使用说明 72](#_Toc80781628)

# 1 引言

## 1.1 课题背景与意义

### 1.1.1 布尔可满足性问题

布尔可满足性问题（SAT）是计算机科学与人工智能领域的基本问题，也是第一个被证明为NP难的问题。在计算复杂性理论中具有非常重要的地位，设计并实现解决该问题的高效算法非常重要。目前，可满足性理论和应用方面的国际会有每年都会进行竞赛寻找高效的SAT求解器。尽管针对命题的可满足性问题的理论研究已经趋于成熟，但其求解速度仍是制约SAT算法的一大难题。

### 1.1.2 DPLL算法

DPLL算法是一种求解SAT问题的完备算法，于1962年由马丁·戴维斯、希拉里·普特南、乔治·洛吉曼和多纳·洛夫兰德共同提出。DPLL算法以二叉搜索与回溯为基础，在发展过程中经过不同学者的不断优化于改进，至今仍是求解SAT问题的最有效算法之一，是很多一阶逻辑的自动定理证明的基础。

## 1.2国内外研究现状

自SAT问题被证明为NP难度问题后，很多学者因此望而却步，因此SAT问题的求解研究在很长一段时间内停滞不前。直到上世纪九十年代，各国家与机构开始组织SAT问题的求解竞赛，涌现出了一批对于SAT问题的高效求解办法，它们多是基于DPLL算法的框架，如MINISAT、SATO、CHAFF等。它们从新型数据结构、新型变量决策策略、新型快速算法实现等方面提升了DPLL算法的求解性能。

国内也有学者贡献了高效的求解方案，如1998年梁东敏提出的WSAT加权算法，2000年金人超与黄文奇提出的并行Solar算法，2002年张富德提出的模拟退火算法。

## 1.3课程设计的主要研究工作

（1）查阅参考文献，了解SAT问题与DPLL算法的理论背景、基础知识与研究现状。

（2）设计数据结构与算法流程，实现基于DPLL算法的高效SAT求解器。

（2）研究数独游戏向SAT问题的转化与规约，应用SAT求解器实现数独问题的求解与自动生成，设计具有可玩性的数独游戏。

（4）参考文献，研究DPLL的优化方案，改进数据结构或算法流程。

（5）基于对程序求解时间的测量，比较优化前后的算法性能，量化优化结果，做出理论分析。

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

(1) 实现cnf文件的解析与求解，能够验证读取结果、打印求解结果并将答案生成固定格式的res文件。

(2) 进行SAT求解时间性能的测量，作为输出的一部分，

(3)实现已知格局的数独游戏的求解并输出答案。

(4)实现数独格局的自动生成和挖洞，保证答案存在且唯一。

(5)实现数独游戏的交互功能，提供难度选择、填写、提示与答案显示功能。

(6)对系统的都一方面进行优化，量化优化结果。

## 2.2 系统总体设计

### 2.2.1 控制模块

模块功能：总体控制程序流程；具体功能包括系统主菜单的打印，根据用户选项控制系统转至相应的功能模块，并在相应功能完成后控制系统返回主菜单，提供程序退出选项。

### 2.2.2 求解器模块

模块功能：求解基于cnf范式的SAT问题；具体功能包括读取并解析cnf文件，验证读入结果，使用DPLL算法求解SAT问题，测量求解时间性能，生成固定格式的res文件。

### 2.2.3 数独模块

模块功能：实现含空数独的求解和交互性数独游戏系统；具体功能包括数独问题向SAT问题的转化，SAT答案向数独答案的转化，已知格局数独的求解，数独格局的自动生成与挖洞，可交互性的游戏系统。

### 2.2.4 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

### 3.1.1 problem结构体

problem结构体能独立完整的表示一个SAT问题，含有SAT问题中的关键信息，包括变元个数、子句个数、子句集根节点指针、变元真值表、变元数量统计。（如表3-1所示）

在程序优化时因为算法流程的变化对其进行了化简，删除了变元数量统计数组并将真值表设置为全局变量。（如表3-2所示）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| varnum | int | 变元个数 |
| statenum | int | 子句个数 |
| root | statement\* | 子句集头指针（不存数据） |
| ans | int[] | 变量真值表 |
| cnt | int[] | 变元数量统计数组 |

表3-1 problem结构体（优化前）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| varnum | int | 变元个数 |
| statenum | int | 子句个数 |
| root | statement\* | 子句集头结点（不存数据） |

表3-2 problem结构体（优化后）

### 3.1.2 statement结构体

statement结构体(st)表示子句集中一条子句，数据包括子句有效长度，首变元指针，标记变量，下一子句指针。（如表3-3所示）

在程序优化时对结构体进行了化简，删除了标记变量。（如表3-4所示）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| mark | int | 为0时表示子句有效，否则为删除时递归深度 |
| num | int | 有效变元个数 |
| firstnode | node\* | 首变元结点指针 |
| next | statement\* | 下一子句指针 |

表3-3 statement结构体（优化前）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| num | int | 变元个数 |
| firstnode | node\* | 首变元结点指针 |
| next | statement\* | 下一子句指针 |

表3-4 statement结构体（优化后）

### 3.1.3 node结构体

node结构体表示子句中的一个变元结点，数据包括变元的值，标记变量和下一变元指针。（如表3-5所示）

在程序优化时对结构体进行了化简，删除了标记变量。（如表3-6所示）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| mark | int | 为0时表示变元有效，否则为删除时递归深度 |
| val | int | 变元的值 |
| next | node\* | 下一变元指针 |

表3-5 node结构体（优化前）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| val | int | 变元的值 |
| next | node\* | 下一变元指针 |

表3-6 node结构体（优化后）

### 3.1.4 sud结构体

sud结构体表示一个数独，处理数据包括数独中待填空的个数，数独的原始格局，数独的显示格局。（如表3-7所示）

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据类型 | 说明 |
| num | int | 待填空个数 |
| ori | int[][] | 数独原始格局 |
| a | int[][] | 数独显示格局 |

表3-7 sud结构体

### 3.1.5 数据结构关联

如图3-1所示



图3-1 数据结构关联图

## 3.2 主要算法设计

### 3.2.1 求解器模块

#### 3.2.1.1 CnfParser子模块

模块功能：实现cnf文件的读取与解析

主要函数：void read(FILE\* fp, problem\* P); //由cnf文件读入问题

程序通过首字母判定跳过注释，读取子句数与变元数。以子句数控制循环变量创建子句链表，在每个子句节点上建立变元链表直至读入结束符，以空指针作为链表的结尾。（如图3-2所示）

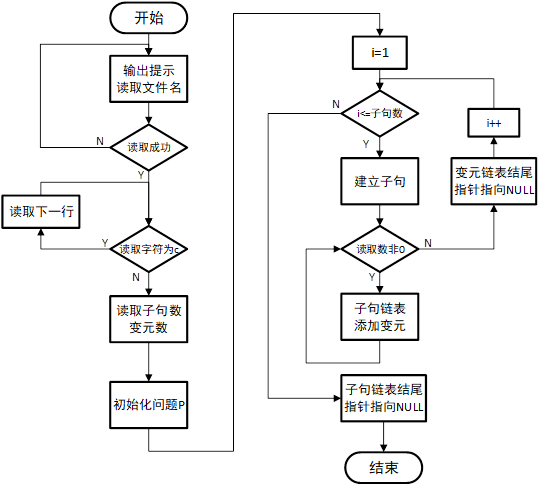


图3-2 CnfParser模块流程图

#### 3.2.1.2 show子模块

模块功能：遍历读取的cnf文件，以供用户验证读取正确性

主要函数：void show(problem\* P); //打印P中有效子句

程序输出已装填problem的变元数与子句数，并遍历输出子句链表和变元链表。（如图3-3所示）



图3-3 show模块流程图

#### 3.2.1.3 solver子模块

模块功能：根据读入的问题P，利用DPLL算法求解SAT问题。

主要函数：

int solve(problem\* P, int depth, int clue); //递归求解函数

int FindUnit(problem\* p); //寻找单子句

int DeleteUnit(problem\* P, int unit, int depth); //利用单子句规则化简P

void recovery(problem\* P, int depth); //还原递归深度为depth时的修改操作

void AddUnit(problem\* P, int unit); //首插法增加单子句

void deletelast(problem\* P, int sup); //删除首插的单子句

int select(problem\* P); //最多变量选择策略

程序不断寻找单子句，并利用单子句规则化简子句集，将所有待删除的子句与结点标记为当前递归深度，直至子句集中无单子句存在。然后，程序根据变量选择策略选择决策变量sup，分别假设sup为真和假并使用首插法添加相应的单子句，再在depth+1递归深度递归求解变化后的问题P。求解完成之后恢复在depth+1深度对问题P做出的修改并删除添加的假设单子句。若任意一种假设求解成功则返回OK，若两种假设均求解失败则返回失败。（如图3-4所示）

程序时间复杂度为（n为元素个数）

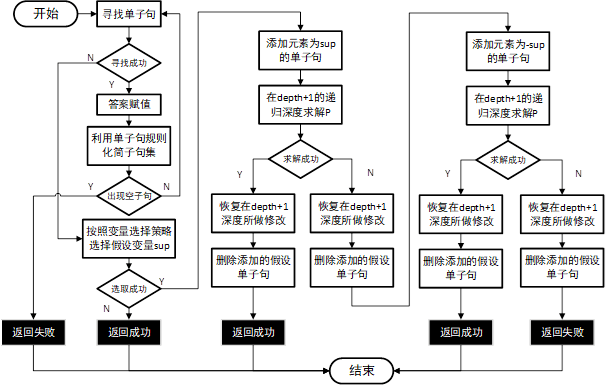


图3-4 solver模块流程图

### 3.2.2 选择策略

模块功能：在搜索树分叉时选择选取一个文字进行假设赋值，对于DPLL性能影响较大

1. **直接选择**

**算法思想：**直接选择第一子句的第一个元素

**算法特点：**节省了选择函数运行的时间，对于少数数据效果较好

1. **选择出现频率最多的元素**

**算法思想**：在链表增删过程中用数组维护每个元素出现的次数，在选择过程中遍历计数数组，选择出现次数最多的元素

**算法依据：**选择出现最多的元素能较多地删除子句和元素，较快地化简子句集，快速降低链表的复杂度

**算法特点：**对于元素个数出现差异较大的算例运行性能高

1. **选择长度最短的子句**

**算法思想：**在选择过程中遍历子句链表，寻找长度最短的子句，选择其中第一个元素

**算法依据：**由于CNF范式中一个子句只需满足一个元素，所以子句越长越容易满足，越短越难满足。优先选择难满足的元素（关键元素）可以降低搜索树规模

**算法特点：**对于子句长度差异较大的子句集性能较强，对于子句长度相近的子句集性能较弱

1. **MOM策略（最短子句中最多出现的元素）**

**原始算法思想：**加权函数(i=1,2,3,…m,假设有m个子句) ，其中为包含文字l的子句的长度。选取使加权函数值最大的变元。

**算法实现思想：**选取最短子句中出现次数最多的元素。遍历子句集，找到长度最短的子句长度，在所有长度为最短长度的子句中选择出现次数最多的元素。

**算法依据：**兼具算法2）、3）的优点

**算法特点：**对于耦合度较小的高难度算例具有优势，但是其决策过程的时间开销较大。

### 3.2.3 数独模块

#### 3.2.2.1 SudSolve子模块

模块功能：求解已知格局的数独（txt文件输入）

主要函数：

void Sudsolve(); //求解已知格局的数独

void SudToSat(char FileName[]); //数独向sat问题的转化

int VarTrans(int i, int j, int n); //数独自然编码向cnf编码转化

void BasicPrint(FILE\* fp); //数独基本规则的cnf输出

void ShowSud(FILE\* fp); //由res文件解码读出数独答案

程序读入数独文件到sud结构体，创建同名cnf后缀文件，写入经过cnf编码后的数独基本规则与已知信息。调用SAT求解器求解cnf文件，得到res答案输出，再将res文件解码为数独自然编码，得到数独答案。（如图3-5所示）



图3-5 sudsolve模块流程图

#### 3.2.2.2 SudPlay子模块

模块功能：实现可玩性数独游戏，能够自动生成指定难度的数独游戏，具有填空、提示、显示答案功能。

主要函数：

void Sudplay(); //数独游玩

sud\* Sudcreate(); //数独格局生成

void DigHole(sud\* k); //数独挖洞

void Candig(sud\* k, int x, int y); //单个点的挖洞

void ShowAns(sud\* k); //数独答案输出

void fShowSud(FILE\* fp1, FILE\* fp2); //res文件向数独文件转化

void hint(sud\* k); //随机提示

程序为空数独随机无矛盾赋值一处（改进版为3处），随后调用sudsolve模块求解数独，作为数独的已知格局。调用挖洞函数对数独进行S型挖洞，对每空进行判定，将其他八个数带入求解，若无解说明此空答案唯一，可以挖洞。再根据用户的难度选择进行特定数量的随机提示，每次提示时随机选取未填空将显示数独的值改为原数独的值。

数独生成成功后打印游玩菜单，根据用户的选项转至相应功能。当用户选择退出时提供再次游玩的选项。（如图3-6所示）

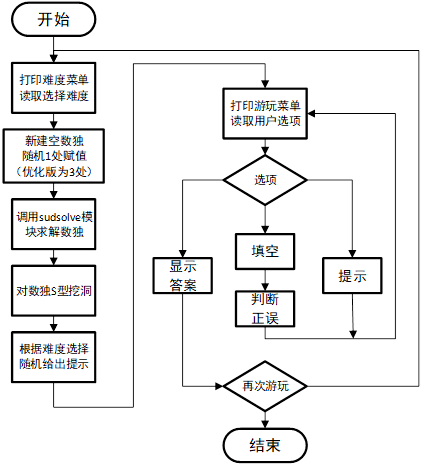


图3-6 sudplay模块流程图

### 3.2.4 系统优化

对于二叉搜索分叉时的算法进行了优化，在变量决策后将原问题复制为两个副本，分别令决策变量为真和假进行递归求解。这样虽然增加了空间的开销，但是一方面节省了回溯时对子句集进行恢复的时间，另一方面可以真正删除和释放结点从而降低了链表的规模从而降低了问题复杂度。（如图3-7所示）

与之对应，在数据结构设计上可以进行化简，删除了子句节点和变元结点的标记变量。



图3-7 系统优化示意图

# 4 系统实现与测试

## 4.1系统实现

### 4.1.1 用户需求

（1）实现高效的SAT求解器（基于CNF范式），能够生成固定格式res文件

（2）实现数独游戏的求解

（2）实现可交互性的数独游戏

### 4.1.2 系统环境

开发环境：Visual Studio 2019 Community

编译器：cl编译器

CPU处理器：Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz

内存：16GB

系统类型：基于x64的处理器

### 4.1.3 数据类型

typedef struct statement {

int mark;（优化时删除） //mark=0表示子句有效,其他情况表示删除时的递归深度

int num; //标记单句的个数

struct node\* firstnode; //第一个变量

struct statement\* next; //下一子句

}st;

typedef struct node {

int mark; （优化时删除） //mark=0表示变量有效,其他情况表示删除时的递归深度

int val;

struct node\* next;

}node;

struct problem {

int varnum, statenum; //变量数量、语句数量

struct statement\* root; //root结点中不存放数据

int cnt[MAX\_VARNUM + 1]; //记录每个变量出现的次数

bool ans[MAX\_VARNUM + 1]; //记录答案

};

struct sud {

int ori[10][10]; //记录原始数值

int a[10][10]; //0表示挖出的洞，其他情况同ori

int num; //非0个数

};

### 4.1.4 系统函数

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <time.h>

void DPLL(); //DPLL问题求解

void init(problem\* P); //初始化root结点

void show(problem\* P); //打印P中有效子句

void read(FILE\* fp, problem\* P); //由cnf文件读入问题

int FindUnit(problem\* p); //寻找单子句

int DeleteUnit(problem\* P, int unit, int depth); //利用单子句规则化简P

void recovery(problem\* P, int depth); //还原递归深度为depth时的修改操作

void AddUnit(problem\* P, int unit); //首插法增加单子句

void deletelast(problem\* P, int sup); //删除首插的单子句

int select(problem\* P); //最多变量选择策略

void sfree(st\* s); //释放子句

void pfree(problem\* P); //销毁问题

int solve(problem\* P, int depth, int clue); //递归求解函数

void Ansprint(FILE\* fp, problem\* P, int succ, int t); //输出答案

int momselect(problem\* P); //MOM策略

int minselect(problem\* P); //最短子句策略

int VarTrans(int i, int j, int n); //数独自然编码与cnf编码转化

void BasicPrint(FILE\* fp); //数独基本规则的cnf输出

void ShowSud(FILE\* fp); //由res文件解码读出数独答案

void SudToSat(char FileName[]); //数独向sat问题的转化

void Sudsolve(); //求解已知格局的数独

void Sudprint(FILE\* fp, sud\* k); //数独的文件写入

void Sudread(FILE\* fp, sud\* k); //数独的文件读取

void Sudplay(); //数独游玩

void Candig(sud\* k, int x, int y); //单个点的挖洞

void DigHole(sud\* k); //数独挖洞

void ShowAns(sud\* k); //数独答案输出

sud\* Sudcreate(); //数独格局生成

void fShowSud(FILE\* fp1, FILE\* fp2); //res文件向数独文件转化

void hint(sud\* k); //随机提示

## 4.2 系统测试

### 4.2.1测试方法

（1）按程序执行状态分类

①静态测试

不运行程序，只依靠分析程序的语句、结构、过程等检查程序是否有误。可参照程序设计说明书、流程图等找出错误。

②动态测试

运行程序，通过对比运行结果和预期结果进行分析，同时分析运行效率和程序健壮性能。

（2）以具体实现算法细节和系统内部结构的相关情况分类

①黑盒测试

不考虑程序内部结构和特性，对软件界面和软件功能进行测试

②白盒测试

借助程序内部结构和相关信息，通过检测内部动作进行测试

③灰盒测试

介于黑盒测试和白盒测试之间，及考虑程序的输入输出也考虑其内部运行情况。

（3）以不同阶段划分

①单元测试 ②集成测试 ③系统测试 ④验收测试

### 4.2.2 控制模块测试

测试对象：控制模块

测试功能：主菜单打印，功能模块索引，系统返回，程序退出

（主菜单打印测试情况如图4-1所示）

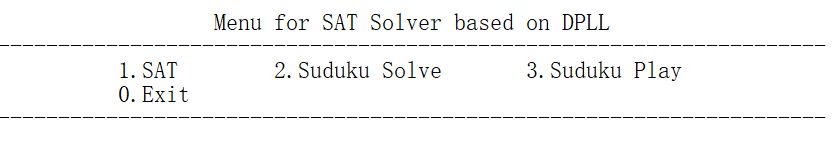


图4-1 主菜单打印测试结果

结果分析：主菜单打印正常，模块读取选项后成功转至相应功能模块，相应功能完成后成功返回主菜单，输入退出选项后程序窗口自动退出。

### 4.2.2 求解器测试

测试对象：求解器模块

测试功能：求解SAT问题（CNF文件）

测试数据：从测试集中选取共20组算例，其中小型算例7组，中型算例7组，大型算例6组；满足算例17组，不满足算例3组；且包含4组基准案例。

测试说明：分别运行优化前后的程序，使用优化率对比程序性能。

测试结果如表4-1所示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试算例 | 算例类型 | 是否优化 | 运行结果 | 优化率 |
| sat-20.cnf | 小型  基准算例 | 否 |  | 算例过小 |
| 变元数  子句数 | 20  91 | 是 |  | 无法统计 |
| unsat-5cnf-30.cnf | 小型  基准算例 | 否 |  | 0% |
| 变元数  子句数 | 30  420 | 是 |  |  |
| ais10.cnf | 小型  基准算例 | 否 |  | 优化率为负 |
| 变元数  子句数 | 181  3151 | 是 |  |  |
| sud00009.cnf | 中型  基准算例 | 否 |  | 99.3% |
| 变元数  子句数 | 303  2851 | 是 |  |  |
| tst\_v25\_c100.cnf | 小型  满足算例 | 否 |  | 算例过小 |
| 变元数  子句数 | 25  100 | 是 |  | 无法统计 |
| problem3-100.cnf | 小型  满足算例 | 否 |  | 89.6% |
| 变元数  子句数 | 100  340 | 是 |  |  |
| problem11-100.cnf | 小型  满足算例 | 否 |  | 76.9% |
| 变元数  子句数 | 100  600 | 是 |  |  |
| u-problem7-50.cnf | 小型  不满足算例 | 否 |  | 86.9% |
| 变元数  子句数 | 50  100 | 是 |  |  |
| problem12-200.cnf | 中型满足算例 | 否 |  | 99.9% |
| 变元数  子句数 | 200  1200 | 是 |  |  |
| sud00001.cnf | 中型满足算例 | 否 |  | 99.9% |
| 变元数  子句数 | 301  2780 | 是 |  |  |
| sud00012.cnf | 中型满足算例 | 否 |  | 99.2% |
| 变元数  子句数 | 232  1901 | 是 |  |  |
| sud00021.cnf | 中型满足算例 | 否 |  | 99.1% |
| 变元数  子句数 | 308  2911 | 是 |  |  |
| sud00079.cnf | 中型满足算例 | 否 |  | 97.5% |
| 变元数  子句数 | 301  2810 | 是 |  |  |
| bart17.shuffled-231.cnf | 中型满足算例 | 否 |  | 100% |
| 变元数  子句数 | 231  1166 | 是 |  |  |
| eh-dp04s04.shuffled-1075 | 大型满足算例 | 否 |  | 优化率为负 |
| 变元数  子句数 | 1075  3152 | 是 |  |  |
| qg4-08.cnf | 大型不满足算例 | 否 | 求解时间过长 | 100% |
| 变元数  子句数 | 512  9685 | 是 |  |  |
| qg7-09.cnf | 大型满足算例 | 否 |  | 98.5% |
| 变元数  子句数 | 729  22060 | 是 |  |  |
| sw100-1.cnf | 大型满足算例 | 否 |  | 优化率为负 |
| 变元数  子句数 | 500  3100 | 是 |  |  |
| sw100-70.cnf | 大型满足算例 | 否 |  | 优化率位负 |
| 变元数  子句数 | 500  3100 | 是 |  |  |
| bw\_large.a.cnf | 大型满足算例 | 否 |  | 优化率为负 |
| 变元数  子句数 | 459  4675 | 是 |  |  |

表4-1 求解器模块测试结果

运行结果分析：

程序能有效求解小型算例、中型算例和部分大型算例。求解时间对于变元的变化比对于子句数的变化更为敏感。

优化后的程序在求解中型算例和多数小型算例时较优化前有显著提升，在求解大型算例时性能有所下降，原因有待阐明。

### 4.2.3 数独求解测试

测试对象：数度求解模块

测试功能：求解已知格局的数独（txt文件输入）

测试数据：从互联网（http://www.puzl.be/en/puzl\_list.html）随机读取10组数独游戏进行求解

测试结果如表4-2所示

|  |  |  |
| --- | --- | --- |
| 文件名 | 初始格局 | 求解答案 |
| Sud01 |  |  |
| Sud02 |  |  |
| Sud03 |  |  |
| Sud04 |  |  |
| Sud05 |  |  |
| Sud06 |  |  |
| Sud07 |  |  |
| Sud08 |  |  |
| Sud09 |  |  |
| Sud10 |  |  |

表4-2 数独求解模块测试结果

运行结果分析：

模块能成功读取并求解已知格局的数独，平均求解时间为45.7ms

### 4.2.4 数独游戏测试

测试对象：数独游戏模块

测试功能：挖洞法自动生成数独，难度选择菜单打印，游戏菜单打印，数独填空，提示，显示答案功能。

测试数据：系统自动生成

测试情况如表4-3所示

|  |  |
| --- | --- |
| 测试项目 | 测试结果 |
| 难度菜单打印 |  |
| 数独自动生成  （简单/中等） |  |
| 数独自动生成  （困难/噩梦） |  |
| 填写答案  （错误/正确） |  |
| 更多提示  （前/后） |  |
| 显示答案 |  |

表4-3 数独游戏模块测试结果

运行结果分析：

模块能成功挖洞生成数独，但挖空的空间分布不太均匀，集中分布在数独的第一行和右上角。算法优化后，程序自动生成数独的时间为10-20s。

数独游戏交互功能运行正常，填空、提示、显示答案模块均运行正常。

# 5 总结与展望

## 5.1 全文总结

课程设计主要工作如下：

（1）查阅参考文献，了解SAT问题的课题背景、理论基础与研究现状。

（2）查阅资料，掌握DPLL算法的算法思想与实现思路。

（3）设计数据结构，实现DPLL算法算法流程，完成SAT求解器编写。

（4）研究数独问题向SAT问题的规约转化，实现SAT求解器的应用。

（5）再次查阅参考文献，了解并实现DPLL算法优化方法。

（6）测试程序，测量算法时间性能，对比优化前后的运行速度。

## 5.1 工作展望

在今后的研究中，围绕着如下几个方面开展工作

（1）深入分析优化后程序为何在中型算例求解中优势明显而在大型算例求解释性能下降的原因。

（2）研究并实现膨胀数据结构、智能回溯算法等DPLL算法优化技术，进一步提升SAT求解器效率。

（3）改进数独S型挖洞算法，解决挖洞空间分布不均、数独自动生成相应时间较长等问题。

（4）设计数独游戏UI与胜利画面，提高数独游戏用户体验与可玩性。

（5）了解SAT求解器在其他领域的应用，拓宽求解器的应用场景，开发基于求解器的更多功能模块。

6 体会

此课程设计是笔者为数不多的如此规模的程序编写经历。虽然大一上学期也开发过成绩管理系统，但是数据结构和算法的复杂度远不及此；虽然下学期的数据结构代码量也比较大，但是都是在给定的数据结构设计框架下完成的。这次从数据结构到算法流程的设计与实现给了笔者很大的锻炼。

在开发过程中经历多次也最令笔者烦扰的是代码重构的过程。当开发过程中需要进行后续功能的实现时，发现原先设计的数据结构不能有效支持后续的算法，就只能重新设计数据结构，将一部分代码推翻重写。这样的经历无疑是令人沮丧的，在此次程序编写的过程中出现了三四次之多，也引起了笔者的思考。

一方面，代码重构次数偏多来源于初期对项目的整体认知不足，在开发过程中只注重如何简洁、高效地实现当前功能，而忽略了后续需求的实现。如果在程序编写之处对项目整体有更深入的理解，做好计划，就可以在一定程度上降低代码重构的次数，从而提升开发的效率。

另一方面，代码重构也是项目开发中不可避免的一部分。在实际开发中，很多时候一开始无法预料到后续算法的细节，甚至可能面临开发中途项目需求的更改或增添；这都要求开发人员具备一定的代码重构能力。在敏捷开发理论中，代码重构也是提升代码质量的一大利器，产品生命周期中重要的环节。辩证地看，在程序编写之初既要做足准备，在编写过程中也要利用好代码重构的机会，整理思路，优化代码，提升代码质量。

另外，在优化的过程中，简化的思路和数据结构反而优化了性能。这启示笔者有时简洁也代表着效率。

最后，在报告攥写阶段由于文件误删和回收站误清空而导致了重大失误，导致报告文件丢失。经数据恢复无果，不得不重写报告，白白多花费了三天的时间，实则是一个教训。

# 参考文献

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

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

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

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

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

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

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

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

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

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

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

[11] 百度百科：软件测试方法https://baike.baidu.com/item/软件测试方法/1850037

[12] 百度百科：程序说明书 https://baike.baidu.com/item/程序说明书/22299741

# 附录一：源程序

## 未优化版

### datast.h

#pragma once

#define MAX\_STATENUM 25000

#define MAX\_VARNUM 3000

#define ERROR 0

#define OK 1

typedef struct statement {

int mark; //mark=0表示子句有效,其他情况表示删除时的递归深度

int num; //标记单句的个数

struct node\* firstnode; //第一个变量

struct statement\* next; //下一子句

}st;

typedef struct node {

int mark; //mark=0表示变量有效,其他情况表示删除时的递归深度

int val;

struct node\* next;

}node;

struct problem {

int varnum, statenum; //变量数量、语句数量

struct statement\* root; //root结点中不存放数据

int cnt[MAX\_VARNUM + 1]; //记录每个变量出现的次数

bool ans[MAX\_VARNUM + 1]; //记录答案

};

struct sud {

int ori[10][10]; //记录原始数值

int a[10][10]; //0表示挖出的洞，其他情况同ori

int num; //非0个数

};

### func.h

#pragma once

#include "datast.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <time.h>

void DPLL(); //DPLL问题求解

void init(problem\* P); //初始化root结点

void show(problem\* P); //打印P中有效子句

void read(FILE\* fp, problem\* P); //由cnf文件读入问题

int FindUnit(problem\* p); //寻找单子句

int DeleteUnit(problem\* P, int unit, int depth);//利用单子句规则化简P

void recovery(problem\* P, int depth); //还原递归深度为depth时的修改操作

void AddUnit(problem\* P, int unit); //首插法增加单子句

void deletelast(problem\* P, int sup); //删除首插的单子句

int select(problem\* P); //最多变量选择策略

void sfree(st\* s); //释放子句

void pfree(problem\* P); //销毁问题

int solve(problem\* P, int depth, int clue); //递归求解函数

void Ansprint(FILE\* fp, problem\* P, int succ, int t); //输出答案

int momselect(problem\* P); //MOM策略

int minselect(problem\* P); //最短子句策略

int VarTrans(int i, int j, int n); //数独自然编码与cnf编码转化

void BasicPrint(FILE\* fp); //数独基本规则的cnf输出

void ShowSud(FILE\* fp); //由res文件解码读出数独答案

void SudToSat(char FileName[]); //数独向sat问题的转化

void Sudsolve(); //求解已知格局的数独

void Sudprint(FILE\* fp, sud\* k); //数独的文件写入

void Sudread(FILE\* fp, sud\* k); //数独的文件读取

void Sudplay(); //数独游玩

void Candig(sud\* k, int x, int y); //单个点的挖洞

void DigHole(sud\* k); //数独挖洞

void ShowAns(sud\* k); //数独答案输出

sud\* Sudcreate(); //数独格局生成

void fShowSud(FILE\* fp1, FILE\* fp2); //res文件向数独文件转化

void hint(sud\* k); //随机提示

### control.cpp

#include "func.h"

#include "datast.h"

int main() {

int choice=1;

while (choice) {//打印菜单

system("cls");

printf(" Menu for SAT Solver based on DPLL \n");

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

printf(" 1.SAT 2.Suduku Solve 3.Suduku Play \n");

printf(" 0.Exit \n");

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

scanf("%d",&choice);//读取选项

switch (choice){

case 1:

DPLL();

break;

case 2:

Sudsolve();

getchar(); getchar();

break;

case 3:

Sudplay();

getchar(); getchar();

default:

break;

}

}

}

### solver.cpp

#include "datast.h"

#include "func.h"

void DPLL() {

problem \*P = (problem\*)malloc(sizeof(problem));

char FileName[20];

printf("请输入测试文件名\n");

FILE\* fp=NULL;

while (!fp) {//读取文件

scanf("%s",FileName);

fp = fopen(FileName, "r");

if (!fp) printf("打开失败，请检查文件并重新输入\n");

}

read(fp,P);

fclose(fp);

int choice,state,flag,len;

char c;

flag = 1;

while (flag) {

system("cls");

printf("文件读入成功！\n");

printf("1.验证读取结果 2.cnf求解\n");

printf("0.退出\n");

scanf("%d", &choice);

switch (choice) {

case 1:

system("cls");

show(P);//验证读入

c = getchar(); c = getchar();

break;

case 2:

system("cls");

time\_t t1, t2;

t1 = clock();

state = solve(P, 1, 0);//求解问题

t2 = clock();

recovery(P, 1);

if (state == OK) {

printf("可满足!\n");

for (int i = 1; i <= P->varnum; i++) {//stdin显示答案

if (P->ans[i]) printf("%5d ", i);

else printf("%5d ", (-1) \* i);

if (!(i % 15)) printf("\n");

}

}

else printf("不可满足!");

printf("\n用时: %d ms", t2 - t1);

len = strlen(FileName);

FileName[len - 3] = 'r', FileName[len - 2] = 'e', FileName[len - 1] = 's';

fp = fopen(FileName,"w+");

Ansprint(fp, P, state, t2 - t1);//res文件输出

fclose(fp);

c = getchar(); c = getchar();

break;

default:

flag = 0;

}

}

pfree(P);

}

void init(problem \*P) {//初始化问题

P->root = (st\*)malloc(sizeof(st));

P->root->firstnode = NULL;

P->root->next = NULL;

P->root->mark = 0;

P->root->num = 0;

memset(P->ans, 0, sizeof(P->ans));//清空数组

memset(P->cnt, 0, sizeof(P->cnt));

return;

}

void show(problem \*P) {

st\* s=P->root;

node\* p;

system("cls");

printf("读取成功\n共有%d个变元，%d个子句\n",P->varnum,P->statenum);

while (s->next) { //遍历子句集

s = s->next;

if (s->mark) continue;

p = s->firstnode;

while (p) {

if(!p->mark) printf("%d ",p->val);

p = p->next;

}

printf("0\n");

}

return;

}

void read(FILE\* fp,problem \*P) {

char c,r[500];

int var;

while ((c = fgetc(fp)) == 'c') {//读取注释

ungetc(c,fp);

fscanf(fp, "%[^\n]", r);

c=fgetc(fp);

}

fscanf(fp,"%s%d%d",r,&P->varnum,&P->statenum);

init(P);

st\* s = P->root;

node\* p;

for (int i = 1; i <= P->statenum; i++) {//建立邻接表

s->next = (st\*)malloc(sizeof(st));

s = s->next;

s->num = 0;

s->mark = 0;

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

fscanf(fp,"%d",&p->val);

p->mark = 0;

s->num++;

P->cnt[abs(p->val)]++;

s->firstnode = p;

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

while (var) {//非结束符

p->next = (node\*)malloc(sizeof(node));

p = p->next;

p->mark = 0;

p->val = var;

s->num++;

P->cnt[abs(var)]++;

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

}

p->next = NULL;

}

s->next = NULL;

}

int FindUnit(problem \*P) {//寻找单子句

int res = 0;

st\* s=P->root;

node\* p;

while (s->next) {//遍历句子

s = s->next;

if (s->mark) continue;

if (s->num==1) {//找到s->num==1的句子中的有效变量

p = s->firstnode;

while (p && p->mark) {

p = p->next;

}

if (p) {

res = p->val;

break;

}

}

}

return res;

}

int DeleteUnit(problem\* P, int unit,int depth) {

st\* s=P->root;

node\* p;

while (s->next) {//遍历句子

s = s->next;

if (s->mark) continue;

p = s->firstnode;

while (p) {//遍历句子中的变元

if (p->mark) {

p = p->next;

continue;

}

if (p->val == unit) {//删除句子

s->mark = depth;//用递归深度标记结点以便回溯

p = s->firstnode;

while (p) {//遍历当前句子中的变元

if (p->mark) {

p = p->next;

continue;

}

P->cnt[abs(p->val)]--;//修改变元出现次数

p = p->next;

}

break;

}

if (p->val + unit == 0) {//删除变元

p->mark = depth;

s->num--;//修改当前句子的变元数

P->cnt[abs(p->val)]--;//修改变元出现次数

if (!s->num) //若修改后当前句子为空，则说明冲突

return ERROR;

}

p = p->next;

}

}

return OK;

}

void recovery(problem\* P, int depth) {

st\* s = P->root;

node\* p;

while (s->next) {//遍历句子

s = s->next;

if (s->mark == depth) {//若句子在depth深度被删除

s->mark = 0;//恢复有效性

p = s->firstnode;

while (p) {//遍历句子中的变元

if (p->mark) {

if (p->mark == depth) {

p->mark = 0;

s->num++;

P->cnt[abs(p->val)]++;//修改出现次数

}

p = p->next;

continue;

}

P->cnt[abs(p->val)]++;

p = p->next;

}

continue;

}

p = s->firstnode;

while (p) {//遍历变量

if (p->mark != depth) {

p = p->next;

continue;

}

p->mark = 0;

s->num++;

P->cnt[abs(p->val)]++;

p = p->next;

}

}

}

void sfree(st\* s) {//释放句子

node\* p1 = s->firstnode, \* p2;

while (p1) {

p2 = p1->next;

free(p1);

p1 = p2;

}

free(s);

}

void pfree(problem\* P) {//释放问题

st\* s1 = P->root,\*s2;

while (s1) {//释放元素链表

s2 = s1->next;

sfree(s1);

s1 = s2;

}

free(P);//释放子句结点

}

int select(problem \*P) {//选择出现次数最多的变量

int max = 0, res = 0;

for (int i = 1; i <= MAX\_VARNUM; i++) {

if (max < P->cnt[i]) {

max = P->cnt[i];

res = i;

}

}

return res;

}

void AddUnit(problem \*P, int unit) {//首插法插入只含一个变量unit的句子

st\* s = P->root;

st\* s1 = s->next,\*s2;

s2 = (st\*)malloc(sizeof(st));

s2->next = s1;

s->next = s2;

s2->mark = 0;

s2->num = 1;

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

p->val = unit;

P->cnt[abs(unit)]++;

p->next = NULL;

p->mark = 0;

s2->firstnode = p;

P->statenum++;

}

void deletelast(problem\* P, int sup) {//删除之前首插的句子

st\* s1 = P->root,\*s2;

s2 = s1->next;

s1->next = s2->next;

P->cnt[abs(sup)]--;

sfree(s2);

P->statenum--;

}

int solve(problem\* P,int depth,int clue) {

int sig;

if (!clue) sig = FindUnit(P);//若无线索，寻找单子句

else sig = clue;//线索为单子句的元素

while (sig) {//若能找到单子句

if (sig > 0) P->ans[sig] = true;//答案赋值

else P->ans[(-1)\*sig] = false;

int state = DeleteUnit(P, sig, depth);//利用单子句规则化简问题

if (state==ERROR) {

return ERROR;

}

sig = FindUnit(P);//继续寻找单子句

}

int sup = select(P);//选取出现次数最多的变元

if (!sup) return OK;//若无变元，说明求解完毕

AddUnit(P, sup);//假设选取的变元为真

if (solve(P, depth + 1,sup)) {//新增的假设必为单子句，将其作为线索

recovery(P, depth + 1);//恢复depth+1递归深度所做的修改

deletelast(P, sup);//删除假设

return OK;

}

recovery(P, depth + 1);//恢复depth+1递归深度所做修改

deletelast(P,sup);//删除假设

AddUnit(P, sup \* (-1));//假设选取的变元为假

if (solve(P, depth + 1,(-1)\*sup)) {

recovery(P, depth + 1);

deletelast(P, sup);

return OK;

}

recovery(P, depth + 1);

deletelast(P,sup);

return ERROR;//若两分支都为假，则结果为假

}

void Ansprint(FILE\* fp, problem \*P, int succ, int t) {

fprintf(fp,"s %d",succ);//成功与否

if (succ != 1) return;

for (int i = 1; i <= P->varnum; i++) {

if (!((i - 1) % 9)) fprintf(fp,"\nv");

if (P->ans[i]) fprintf(fp,"%5d ", i);

else fprintf(fp,"%5d ", (-1) \* i);

}

fprintf(fp,"\nt %d",t);

}

int momselect(problem\* P) {//选取最短子句中出现最多的元素

st\* s = P->root;

node\* p;

int min,fg,res;

fg = 1;

min = 0xFFFFFF, p = NULL;

while (s->next) {//找到最短子句

s = s->next;

if (s->mark) continue;

if (s->num && s->num < min) {

min = s->num;

fg = 0;

}

}

if (fg) return 0;

memset(P->cnt, 0, sizeof(P->cnt));

s = P->root;

while (s->next) {//找到最短子句中出现次数最多的元素

s = s->next;

if (s->num != min||s->mark) continue;

p = s->firstnode;

while (p) {

if (p->mark) {

p = p->next;

continue;

}

P->cnt[abs(p->val)]++;

p = p->next;

}

}

int max = 0;

res = 0;

for (int i = 1; i <= P->varnum; i++) {

if (max < P->cnt[i]) {

max = P->cnt[i];

res = i;

}

}

return res;

}

int minselect(problem\* P) {//选择最短子句

st\* s = P->root;

node\* p;

int min;

min = 0xFFFFFF, p = NULL;

while (s->next) {//寻找最短子句

s = s->next;

if (s->mark) continue;

if (s->num && s->num < min) {

min = s->num;

p = s->firstnode;

}

}

if (!p) return 0;

min = 0xFFFFFF;

while (p) {//寻找最短子句中的有效元素

if (p->mark) {

p = p->next;

continue;

}

return p->val;

}

return 0;

}

### sud.cpp

#include "datast.h"

#include "func.h"

void Sudsolve() {

system("cls");

char FileName[20];

printf("请输入数独文件:\n");

scanf("%s",FileName);

int len = strlen(FileName);

while (strcmp(FileName + len - 3, "txt")) {

printf("格式错误，请重新输入\n");

scanf("%s", FileName);

len = strlen(FileName);

}

SudToSat(FileName);//将数独转化为SAT问题

FileName[len - 3] = 'c', FileName[len - 2] = 'n', FileName[len - 1] = 'f';

problem\* P = (problem\*)malloc(sizeof(problem));

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

read(fp, P);

fclose(fp);

time\_t t1, t2;

t1 = clock();

int state=solve(P,1,0);//求解SAT问题

t2 = clock();

if (state == 1) printf("求解成功！\n");

if (state == 0) printf("数独无解\n");

if (state == -1) printf("求解超时\n");

len = strlen(FileName);

FileName[len - 3] = 'r', FileName[len - 2] = 'e', FileName[len - 1] = 's';

fp = fopen(FileName, "w+");

Ansprint(fp, P, state, t2-t1);//输出res文件

fclose(fp);

fp = fopen(FileName, "r");

ShowSud(fp);//解码res文件，得到数独答案

printf("\n用时: %d ms",t2-t1);

pfree(P);

}

int VarTrans(int i, int j, int n) {

return (i - 1) \* 81 + (j - 1) \* 9 + n;

}

void BasicPrint(FILE\* fp) {

for (int i = 1; i <= 9; i++) { //每空至少一个数字

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++) {//每行每个数字都出现

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

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

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

}

fprintf(fp,"0\n");

}

}

for (int j = 1; j <= 9; j++) {//每列每个数字都出现

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int p = 0; p < 3; p++) {//每3x3区域每个数字都出现

for (int q = 0; q < 3; q++) {

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

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

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

fprintf(fp,"%d ",VarTrans(3\*p+i,3\*q+j,n));

}

}

}

fprintf(fp,"0\n");

}

}

for (int i = 1; i <= 9; i++) {//一行数字不重复

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

for (int j1 = 1; j1 < 9; j1++) {

for (int j2 = j1 + 1; j2 <= 9; j2++) {

fprintf(fp, "%d %d 0\n", (-1) \* VarTrans(i, j1, n), (-1) \* VarTrans(i, j2, n));

}

}

}

}

for (int j = 1; j <= 9; j++) {//一列数字不重复

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

for (int i1 = 1; i1 < 9; i1++) {

for (int i2 = i1 + 1; i2 <= 9; i2++) {

fprintf(fp, "%d %d 0\n", (-1) \* VarTrans(i1, j, n), (-1) \* VarTrans(i2, j, n));

}

}

}

}

for (int p = 0; p < 3; p++) {//一3x3区域数字不重复

for (int q = 0; q < 3; q++) {

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

for (int k1 = 0; k1 <= 7; k1++) {

for (int k2 = k1 + 1; k2 <= 8; k2++) {

fprintf(fp, "%d %d 0\n",

(-1) \* VarTrans(3 \* p + k1 / 3 + 1, 3 \* q + k1 % 3 + 1, n),

(-1) \* VarTrans(3 \* p + k2 / 3 + 1, 3 \* q + k2 % 3 + 1, n));

}

}

}

}

}

}

void ShowSud(FILE\* fp){//解码res文件，stdout输出答案

int n;

char c;

fscanf(fp,"%c%d",&c,&n);

if (n != 1) return;

for (int k = 1; k <= 81; k++) {

if (!((k-1) % 9)) printf("\n");

fscanf(fp," %c",&c);

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

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

if (n > 0) printf("%3d",i);

}

}

}

void SudToSat(char FileName[]) {

FILE\* fp1, \* fp2;

fp1 = fopen(FileName,"r");

char Output[20];

strcpy(Output, FileName);

int len = strlen(FileName);

Output[len - 3] = 'c', Output[len - 2] = 'n', Output[len - 1] = 'f';

fp2 = fopen(Output, "w");

sud\* k;

k = (sud\*)malloc(sizeof(sud));

Sudread(fp1, k);

fprintf(fp2,"p cnf %d %d\n",729,9000+k->num);//共729个变元，原始规则包含9000条语句

BasicPrint(fp2);//打印原始规则

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

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

if(k->a[i][j]) fprintf(fp2,"%d 0\n",(i-1)\*81+(j-1)\*9+k->a[i][j]);

}

}

fclose(fp1), fclose(fp2);

}

void Sudprint(FILE\* fp, sud \*k) {

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

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

fprintf(fp,"%d ",k->a[i][j]);

}

fprintf(fp,"\n");

}

}

void Sudread(FILE\* fp, sud\* k) {

k->num = 0;

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

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

fscanf(fp, "%d", &k->ori[i][j]);

k->a[i][j] = k->ori[i][j];

if (k->a[i][j]) k->num++;

}

}

}

void Candig(sud\* k, int x,int y) {

char temp[10] = "temp.txt";

FILE\* fp1,\*fp2;

int state;

for (int i = 1; i <= 9; i++) {//试填除原来数字之外的8个数字，若都不能满足说明可以挖洞

if (i == k->ori[x][y]) continue;

fp1 = fopen(temp, "w");

fp2 = fopen("temp.cnf", "r");

k->a[x][y] = i;

Sudprint(fp1, k);

fclose(fp1);

SudToSat(temp);

problem\* P = (problem\*)malloc(sizeof(problem));

read(fp2, P);

fclose(fp2);

state = solve(P, 1, 0);

if (state == 1) {//若能满足，说明不能挖洞

k->a[x][y] = k->ori[x][y];

pfree(P);

return;

}

pfree(P);

}

k->a[x][y] = 0;

return;

}

void DigHole(sud\* k) {

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

fclose(fp);

for (int i = 1; i <= 9; i++) {//尝试另外9种情况

if (i % 2) {//S型行进

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

Candig(k, i, j);

}

}

else {

for (int j = 9; j >= 1; j--) {

Candig(k, i, j);

}

}

}

}

void ShowAns(sud\* k) {

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

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

printf("%3d",k->ori[i][j]);

}

printf("\n");

}

}

sud\* Sudcreate() {

int x, y, v;

srand(time(NULL));

x = rand() % 9 + 1;//随机确定数独一个位置的数值

y = rand() % 9 + 1;

v = rand() % 9 + 1;

sud\* k = (sud\*)malloc(sizeof(sud));

k->num = 1;

memset(k->a, 0, sizeof(k->a));

k->a[x][y] = v;

FILE\* fp = fopen("sud.txt","w");

Sudprint(fp, k);

fclose(fp);

free(k);

char temp[10] = "sud.txt";

SudToSat(temp);

problem\* P = (problem\*)malloc(sizeof(problem));

fp = fopen("sud.cnf","r");

read(fp, P);

time\_t t1, t2;

t1 = clock();

solve(P, 1, 0);//求解只含有一个数字的数独

t2 = clock();

fclose(fp);

fp = fopen("sud.res","w");

Ansprint(fp,P,1,t2-t1);

fclose(fp);

pfree(P);

FILE\* fp1 = fopen("sud.res", "r");

FILE\* fp2 = fopen("sud.txt", "w");

fShowSud(fp1, fp2);//生成未挖洞的数独格局

fclose(fp1), fclose(fp2);

fp = fopen("sud.txt", "r");

k = (sud\*)malloc(sizeof(sud));

Sudread(fp, k);

fclose(fp);

DigHole(k);//挖洞

return k;

}

void fShowSud(FILE\* fp1,FILE\* fp2) {

int n;

char c;

fscanf(fp1, "%c%d", &c, &n);

if (n != 1) return;

for (int k = 1; k <= 81; k++) {

if (!((k - 1) % 9)) fprintf(fp2,"\n");

fscanf(fp1, " %c", &c);

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

fscanf(fp1, "%d", &n);

if (n > 0) fprintf(fp2,"%3d", i);

}

}

}

void Sudplay() {

int choice = 1,diff,op,x,y,v;

while (choice) {

system("cls");

printf("请选择难度：\n");

printf(" 1.简单 2.中等 3.困难 4.噩梦\n");

scanf("%d",&diff);

printf("正在生成数独，请稍等");

sud\* k;

k = Sudcreate();

for (int i = 1; i <= (4 - diff) \* 10; i++) {

hint(k);

}

op = 1;

while (op) {

system("cls");

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

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

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

printf("%3d", k->a[i][j]);

}

printf("\n");

}

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

printf("\n1.填写 2.更多提示\n0.显示答案\n");

scanf("%d",&op);

if (op==2) hint(k);

if (op == 1) {

printf("\n输入填写的坐标与待填值:");

scanf("%d%d%d",&x,&y,&v);

if (!k->a[x][y] && v == k->ori[x][y]) {

printf("\n填写正确!\n");

k->a[x][y] = k->ori[x][y];

getchar(); getchar();

}

else {

printf("\n答案错误\n");

getchar(); getchar();

}

}

}

ShowAns(k);

printf("1.再来一局 0.退出\n");

free(k);

scanf("%d",&choice);

}

}

void hint(sud\* k) {

int x, y, fg;

fg = 1;

srand(time(NULL));

while (fg) {

x = rand() % 9 + 1;

y = rand() % 9 + 1;

if (!k->a[x][y]) {//为空

k->a[x][y] = k->ori[x][y];

fg = 0;

}

}

}

## 优化版

### datast.h

#pragma once

#define MAX\_STATENUM 25000

#define MAX\_VARNUM 3000

#define ERROR 0

#define OK 1

typedef struct statement {

int num; //标记单句的个数

struct node\* firstnode; //第一个变量

struct statement\* next; //下一子句

}st;

typedef struct node {

int val;

struct node\* next;

}node;

struct problem {

int varnum, statenum; //变量数量、语句数量

struct statement\* root; //root结点中不存放数据

};

struct count {

int num = 0;

int pnum = 0;

int nnum = 0;

};

struct sud {

int ori[10][10]; //记录原始数值

int a[10][10]; //0表示挖出的洞，其他情况同ori

int num; //非0个数

};

### func.h

#pragma once

#include "datast.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <time.h>

void DPLL(); //DPLL问题求解

void init(problem\* P); //初始化root结点

void show(problem\* P); //打印P中有效子句

void read(FILE\* fp, problem\* P); //由cnf文件读入问题

int FindUnit(problem\* p); //寻找单子句

int DeleteUnit(problem\* P, int unit); //利用单子句规则化简P

void AddUnit(problem\* P, int unit); //首插法增加单子句

int select(problem\* P); //最多变量选择策略

void sfree(st\* s); //释放子句

void pfree(problem\* P); //销毁问题

int solve(problem\* P); //递归求解函数

void Ansprint(FILE\* fp, problem\* P, int succ, int t); //输出答案

int minselect(problem\* P); //最短子句策略

problem\* PCopy(problem\* P1);

int VarTrans(int i, int j, int n); //数独自然编码与cnf编码转化

void BasicPrint(FILE\* fp); //数独基本规则的cnf输出

void ShowSud(FILE\* fp); //由res文件解码读出数独答案

void SudToSat(char FileName[]); //数独向sat问题的转化

void Sudsolve(); //求解已知格局的数独

void Sudprint(FILE\* fp, sud\* k); //数独的文件写入

void Sudread(FILE\* fp, sud\* k); //数独的文件读取

void Sudplay(); //数独游玩

void Candig(sud\* k, int x, int y); //单个点的挖洞

void DigHole(sud\* k); //数独挖洞

void ShowAns(sud\* k); //数独答案输出

sud\* Sudcreate(); //数独格局生成

void fShowSud(FILE\* fp1, FILE\* fp2); //res文件向数独文件转化

void hint(sud\* k); //随机提示

### control.cpp

#include "func.h"

#include "datast.h"

int main() {

int choice=1;

while (choice) {//打印菜单

system("cls");

printf(" Menu for SAT Solver based on DPLL \n");

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

printf(" 1.SAT 2.Suduku Solve 3.Suduku Play \n");

printf(" 0.Exit \n");

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

scanf("%d",&choice);//读取选项

switch (choice){

case 1:

DPLL();

break;

case 2:

Sudsolve();

getchar(); getchar();

break;

case 3:

Sudplay();

getchar(); getchar();

default:

break;

}

}

}

### solver.cpp

#include "datast.h"

#include "func.h"

bool ans[MAX\_VARNUM];

void DPLL() {

problem\* P = (problem\*)malloc(sizeof(problem));

char FileName[25];

printf("请输入测试文件名\n");

FILE\* fp = NULL;

while (!fp) {//读取文件名

scanf("%s", FileName);

fp = fopen(FileName, "r");

if (!fp) printf("打开失败，请检查文件并重新输入\n");

}

read(fp, P);

fclose(fp);

int choice, state, flag, len;

char c;

flag = 1;

while (flag) {

system("cls");

printf("文件读入成功！\n");

printf("1.验证读取结果 2.cnf求解\n");

printf("0.退出\n");

scanf("%d", &choice);

switch (choice) {

case 1:

system("cls");

show(P);//验证读入

c = getchar(); c = getchar();

break;

case 2:

system("cls");

time\_t t1, t2;

t1 = clock();

state = solve(P);//求解问题

t2 = clock();

if (state == OK) {

printf("可满足!\n");

for (int i = 1; i <= P->varnum; i++) {//stdout显示答案

if (ans[i]) printf("%5d ", i);

else printf("%5d ", (-1) \* i);

if (!(i % 15)) printf("\n");

}

}

else printf("不可满足!");

printf("\n用时: %d ms", t2 - t1);

len = strlen(FileName);

FileName[len - 3] = 'r', FileName[len - 2] = 'e', FileName[len - 1] = 's';

fp = fopen(FileName, "w+");

Ansprint(fp, P, state, t2 - t1);//res文件输出

fclose(fp);

c = getchar(); c = getchar();

break;

default:

flag = 0;

}

}

pfree(P);

}

void init(problem\* P) {//初始化问题

P->root = (st\*)malloc(sizeof(st));

P->root->firstnode = NULL;

P->root->next = NULL;

P->root->num = 0;

return;

}

void show(problem\* P) {

st\* s = P->root;

node\* p;

system("cls");

printf("读取成功\n共有%d个变元，%d个子句\n", P->varnum, P->statenum);

while (s->next) {遍历子句集

s = s->next;

p = s->firstnode;

while (p) {

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

p = p->next;

}

printf("0\n");

}

return;

}

void read(FILE\* fp, problem\* P) {

char c, r[500];

int var;

while ((c = fgetc(fp)) == 'c') {//读取注释

ungetc(c, fp);

fscanf(fp, "%[^\n]", r);

c = fgetc(fp);

}

fscanf(fp, "%s%d%d", r, &P->varnum, &P->statenum);

init(P);

st\* s = P->root;

node\* p;

for (int i = 1; i <= P->statenum; i++) {//建立邻接表

s->next = (st\*)malloc(sizeof(st));

s = s->next;

s->num = 0;

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

fscanf(fp, "%d", &p->val);

s->num++;

s->firstnode = p;

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

while (var) {//非结束符

p->next = (node\*)malloc(sizeof(node));

p = p->next;

p->val = var;

s->num++;

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

}

p->next = NULL;

}

s->next = NULL;

}

int FindUnit(problem\* P) {//寻找单子句

int res = 0;

st\* s = P->root;

while (s->next) {//遍历句子

s = s->next;

if (s->num == 1) {//找到s->num==1的句子中的有效变量

res = s->firstnode->val;

break;

}

}

return res;

}

int DeleteUnit(problem\* P, int unit) {

st\* s1 = P->root,\*s2;

node\* p1,\*p2;

while (s1->next) {//遍历句子

s2 = s1->next;

p1 = s2->firstnode;

if (p1->val == unit) {

s1->next = s2->next;

p1 = s2->firstnode;

while (p1) {

p1 = p1->next;

}

sfree(s2);

continue;

}

while (p1->val + unit == 0) {//特判首结点

s2->firstnode = p1->next;

free(p1);

p1 = s2->firstnode;

s2->num--;

if (!p1) return ERROR;

}

while (p1->next) {//判定后续结点

p2 = p1->next;

if (p2->val == unit) {//删除子句

s1->next = s2->next;

p1 = s2->firstnode;

sfree(s2);

s2 = s1;

break;

}

if (p2->val + unit == 0) {//删除结点

p1->next = p2->next;

free(p2);

s2->num--;

continue;

}

p1 = p2;

}

s1 = s2;

}

return OK;

}

void sfree(st\* s) {//释放句子

node\* p1 = s->firstnode, \* p2;

while (p1) {

p2 = p1->next;

free(p1);

p1 = p2;

}

free(s);

}

void pfree(problem\* P) {//释放问题

st\* s1 = P->root, \* s2;

while (s1) {

s2 = s1->next;

sfree(s1);

s1 = s2;

}

free(P);

}

void AddUnit(problem\* P, int unit) {//首插法插入只含一个变量unit的句子

st\* s = P->root;

st\* s1 = s->next, \* s2;

s2 = (st\*)malloc(sizeof(st));

s2->next = s1;

s->next = s2;

s2->num = 1;

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

p->val = unit;

p->next = NULL;

s2->firstnode = p;

P->statenum++;

}

int solve(problem\* P) {

int sig;

sig = FindUnit(P);//寻找单子句

while (sig) {//若能找到单子句

if (sig > 0) ans[sig] = true;//答案赋值

else ans[(-1) \* sig] = false;

int state = DeleteUnit(P, sig);//利用单子句规则化简问题

if (state == ERROR) {

return ERROR;

}

sig = FindUnit(P);//继续寻找单子句

}

//int sup = select(P);//选取出现次数最多的变元

//if (!sup) return OK;//若无变元，说明求解完毕

if (!P->root->next) return OK;

int sup = P->root->next->firstnode->val;

problem\* P1,\*P2;

P1 = PCopy(P);

AddUnit(P1, sup);

if (solve(P1)) {

return OK;

}

pfree(P1);

P2 = PCopy(P);

AddUnit(P2, -1 \* sup);

if (solve(P2)) {

return OK;

}

pfree(P2);

return ERROR;//若两分支都为假，则结果为假

}

problem\* PCopy(problem\* P1) {

problem\* P2 = (problem\*)malloc(sizeof(problem));//新建副本

P2->varnum = P1->varnum;//复制变元数和子句数

P2->statenum = P1->statenum;

st\* s1, \* s2;

node\* p1, \* p2;

s1 = P1->root;

s2 = (st\*)malloc(sizeof(st));

s2->firstnode = NULL;

s2->num = 0;

P2->root = s2;

while (s1->next) {//复制子句链表

s1 = s1->next;

s2->next = (st\*)malloc(sizeof(st));

s2 = s2->next;

s2->num = s1->num;

p1 = s1->firstnode;

s2->firstnode = (node\*)malloc(sizeof(node));//首结点特判

p2 = s2->firstnode;

p2->val = p1->val;

while (p1->next) {//复制元素链表

p1 = p1->next;

p2->next = (node\*)malloc(sizeof(node));

p2 = p2->next;

p2->val = p1->val;

}

p2->next = NULL;

}

s2->next = NULL;

return P2;

}

void Ansprint(FILE\* fp, problem\* P, int succ, int t) {

fprintf(fp, "s %d", succ);

if (succ != 1) return;

for (int i = 1; i <= P->varnum; i++) {

if (!((i - 1) % 9)) fprintf(fp, "\nv");

if (ans[i]) fprintf(fp, "%5d ", i);

else fprintf(fp, "%5d ", (-1) \* i);

}

fprintf(fp, "\nt %d", t);

}

int select(problem\* P) {//选择出现次数最多的变量

count\* cnt = new count[P->varnum+1];

st\* s = P->root;

node\* p;

while (s->next) {

s = s->next;

p = s->firstnode;

while (p) {

cnt[abs(p->val)].num++;

if (p->val > 0) cnt[abs(p->val)].pnum++;

else cnt[abs(p->val)].nnum++;

p = p->next;

}

}

int fg = 0, res = 0, max = 0;

for (int i = 1; i <= P->varnum; i++) {

if (max < cnt[i].num) {

max = cnt[i].num;

res = i;

if (cnt[i].pnum > cnt[i].nnum) fg = 1;

else fg = -1;

}

}

delete[] cnt;

return fg \* res;

}

int minselect(problem\* P) {

st\* s = P->root;

node\* p;

int min;

min = 0xFFFFFF, p = NULL;

while (s->next) {

s = s->next;

if (s->num < min) {

min = s->num;

p = s->firstnode;

}

}

if (!p) return 0;

return p->val;

}

### sud.cpp

#include "datast.h"

#include "func.h"

void Sudsolve() {

system("cls");

char FileName[20];

printf("请输入数独文件:\n");

scanf("%s", FileName);

int len = strlen(FileName);

while (strcmp(FileName + len - 3, "txt")) {

printf("格式错误，请重新输入\n");

scanf("%s", FileName);

len = strlen(FileName);

}

SudToSat(FileName);

FileName[len - 3] = 'c', FileName[len - 2] = 'n', FileName[len - 1] = 'f';

problem\* P = (problem\*)malloc(sizeof(problem));

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

read(fp, P);

fclose(fp);

time\_t t1, t2;

t1 = clock();

int state = solve(P);

t2 = clock();

if (state == 1) printf("求解成功！\n");

if (state == 0) printf("数独无解\n");

if (state == -1) printf("求解超时\n");

len = strlen(FileName);

FileName[len - 3] = 'r', FileName[len - 2] = 'e', FileName[len - 1] = 's';

fp = fopen(FileName, "w+");

Ansprint(fp, P, state, t2 - t1);

fclose(fp);

fp = fopen(FileName, "r");

ShowSud(fp);

printf("\n用时: %d ms", t2 - t1);

pfree(P);

}

int VarTrans(int i, int j, int n) {

return (i - 1) \* 81 + (j - 1) \* 9 + n;

}

void BasicPrint(FILE\* fp) {//每空至少一个数字

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

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++) {//每行每个数字都出现

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int j = 1; j <= 9; j++) {//每列每个数字都出现

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

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

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

}

fprintf(fp, "0\n");

}

}

for (int p = 0; p < 3; p++) {//每区域每个数字都出现

for (int q = 0; q < 3; q++) {

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

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

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

fprintf(fp, "%d ", VarTrans(3 \* p + i, 3 \* q + j, n));

}

}

}

fprintf(fp, "0\n");

}

}

for (int i = 1; i <= 9; i++) {//每行每个数字出现一次

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

for (int j1 = 1; j1 < 9; j1++) {

for (int j2 = j1 + 1; j2 <= 9; j2++) {

fprintf(fp, "%d %d 0\n", (-1) \* VarTrans(i, j1, n), (-1) \* VarTrans(i, j2, n));

}

}

}

}

for (int j = 1; j <= 9; j++) {//每列每个数字出现一次

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

for (int i1 = 1; i1 < 9; i1++) {

for (int i2 = i1 + 1; i2 <= 9; i2++) {

fprintf(fp, "%d %d 0\n", (-1) \* VarTrans(i1, j, n), (-1) \* VarTrans(i2, j, n));

}

}

}

}

for (int p = 0; p < 3; p++) {//每区域每个数字出现一次

for (int q = 0; q < 3; q++) {

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

for (int k1 = 0; k1 <= 7; k1++) {

for (int k2 = k1 + 1; k2 <= 8; k2++) {

fprintf(fp, "%d %d 0\n",

(-1) \* VarTrans(3 \* p + k1 / 3 + 1, 3 \* q + k1 % 3 + 1, n),

(-1) \* VarTrans(3 \* p + k2 / 3 + 1, 3 \* q + k2 % 3 + 1, n));

}

}

}

}

}

}

void ShowSud(FILE\* fp) {

int n;

char c;

fscanf(fp, "%c%d", &c, &n);

if (n != 1) return;

for (int k = 1; k <= 81; k++) {

if (!((k - 1) % 9)) printf("\n");

fscanf(fp, " %c", &c);

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

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

if (n > 0) printf("%3d", i);

}

}

}

void SudToSat(char FileName[]) {//数独向SAT问题转化

FILE\* fp1, \* fp2;

fp1 = fopen(FileName, "r");

char Output[20];

strcpy(Output, FileName);

int len = strlen(FileName);

Output[len - 3] = 'c', Output[len - 2] = 'n', Output[len - 1] = 'f';

fp2 = fopen(Output, "w");

sud\* k;

k = (sud\*)malloc(sizeof(sud));

Sudread(fp1, k);

fprintf(fp2, "p cnf %d %d\n", 729, 9000 + k->num);//共729个变元，原始规则包含9000条语句

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

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

if (k->a[i][j]) fprintf(fp2, "%d 0\n", (i - 1) \* 81 + (j - 1) \* 9 + k->a[i][j]);

}

}

BasicPrint(fp2);//打印原始规则

fclose(fp1), fclose(fp2);

}

void Sudprint(FILE\* fp, sud\* k) {

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

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

fprintf(fp, "%d ", k->a[i][j]);

}

fprintf(fp, "\n");

}

}

void Sudread(FILE\* fp, sud\* k) {

k->num = 0;

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

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

fscanf(fp, "%d", &k->ori[i][j]);

k->a[i][j] = k->ori[i][j];

if (k->a[i][j]) k->num++;

}

}

}

void Candig(sud\* k, int x, int y) {

char temp[10] = "temp.txt";

FILE\* fp1, \* fp2;

int state;

for (int i = 1; i <= 9; i++) {//试填除原来数字之外的8个数字，若都不能满足说明可以挖洞

if (i == k->ori[x][y]) continue;

fp1 = fopen(temp, "w");

fp2 = fopen("temp.cnf", "r");

k->a[x][y] = i;

Sudprint(fp1, k);

fclose(fp1);

SudToSat(temp);

problem\* P = (problem\*)malloc(sizeof(problem));

read(fp2, P);

fclose(fp2);

state = solve(P);

if (state == 1) {//若能满足，说明不能挖洞

k->a[x][y] = k->ori[x][y];

pfree(P);

return;

}

pfree(P);

}

k->a[x][y] = 0;

return;

}

void DigHole(sud\* k) {

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

fclose(fp);

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

if (i % 2) {//S型行进

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

Candig(k, i, j);

}

}

else {

for (int j = 9; j >= 1; j--) {

Candig(k, i, j);

}

}

}

}

void ShowAns(sud\* k) {

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

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

printf("%3d", k->ori[i][j]);

}

printf("\n");

}

}

sud\* Sudcreate() {

int x, y, v;

srand(time(NULL));

sud\* k = (sud\*)malloc(sizeof(sud));

k->num = 3;

memset(k->a, 0, sizeof(k->a));

for (int i = 0; i < 3; i++) {

x = rand() % 3 + 1 + i \* 3;//随机确定数独一个位置的数值

y = rand() % 3 + 1 + i \* 3;

v = rand() % 3 + 1 ;

k->a[x][y] = v;

}

FILE\* fp = fopen("sud.txt", "w");

Sudprint(fp, k);

fclose(fp);

free(k);

char temp[10] = "sud.txt";

SudToSat(temp);

problem\* P = (problem\*)malloc(sizeof(problem));

fp = fopen("sud.cnf", "r");

read(fp, P);

time\_t t1, t2;

t1 = clock();

solve(P);//求解只含有一个数字的数独

t2 = clock();

fclose(fp);

fp = fopen("sud.res", "w");

Ansprint(fp, P, 1, t2 - t1);

fclose(fp);

pfree(P);

FILE\* fp1 = fopen("sud.res", "r");

FILE\* fp2 = fopen("sud.txt", "w");

fShowSud(fp1, fp2);//生成未挖洞的数独格局

fclose(fp1), fclose(fp2);

fp = fopen("sud.txt", "r");

k = (sud\*)malloc(sizeof(sud));

Sudread(fp, k);

fclose(fp);

DigHole(k);//挖洞

return k;

}

void fShowSud(FILE\* fp1, FILE\* fp2) {

int n;

char c;

fscanf(fp1, "%c%d", &c, &n);

if (n != 1) return;

for (int k = 1; k <= 81; k++) {

if (!((k - 1) % 9)) fprintf(fp2, "\n");

fscanf(fp1, " %c", &c);

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

fscanf(fp1, "%d", &n);

if (n > 0) fprintf(fp2, "%3d", i);

}

}

}

void Sudplay() {

int choice = 1, diff, op, x, y, v;

while (choice) {

system("cls");

printf("请选择难度：\n");

printf(" 1.简单 2.中等 3.困难 4.噩梦\n");

scanf("%d", &diff);

printf("正在生成数独，请稍等");

sud\* k;

k = Sudcreate();

for (int i = 1; i <= (4 - diff) \* 10; i++) {

hint(k);

}

op = 1;

while (op) {

system("cls");

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

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

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

printf("%3d", k->a[i][j]);

}

printf("\n");

}

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

printf("\n1.填写 2.更多提示\n0.显示答案\n");

scanf("%d", &op);

if (op == 2) hint(k);

if (op == 1) {

printf("\n输入填写的坐标与待填值:");

scanf("%d%d%d", &x, &y, &v);

if (!k->a[x][y] && v == k->ori[x][y]) {

printf("\n填写正确!\n");

k->a[x][y] = k->ori[x][y];

getchar(); getchar();

}

else {

printf("\n答案错误\n");

getchar(); getchar();

}

}

}

ShowAns(k);

printf("1.再来一局 0.退出\n");

free(k);

scanf("%d", &choice);

}

}

void hint(sud\* k) {

int x, y, fg;

fg = 1;

srand(time(NULL));

while (fg) {//为空

x = rand() % 9 + 1;

y = rand() % 9 + 1;

if (!k->a[x][y]) {

k->a[x][y] = k->ori[x][y];

fg = 0;

}

}

}

# 附录二：程序使用说明

程序名称：基于SAT的数独游戏求解程序

程序模块：求解器模块、数独模块

程序编写语言：C/C++（旧MSVC/ISO C++14标准）

程序输入方法与格式：如表9-1所示

|  |  |  |
| --- | --- | --- |
| 内容 | 输入方式 | 输入格式 |
| 菜单选项 | stdin输入 | 数字 |
| 文件名 | stdin输入 | 带文件后缀名输入 |
| SAT算例 | .cnf文件 | CNF范式规范输入 |
| 数独算例 | .txt文件 | 空用数字0代替，其余为原数字矩阵 |
| 游戏交互 | stdin输入 | 数字 |

表9-1 程序输入方法与格式

程序输出方法与格式：如表9-2所示

|  |  |  |
| --- | --- | --- |
| 内容 | 输出方式 | 输出格式 |
| 菜单 | stdout输出 | 菜单格式 |
| SAT答案 | stdout输出 /.res文件 | 是否满足+变量赋值+用时/.res规范格式输出 |
| 数独算例 | stdout输出 | 答案数字矩阵 |
| 游戏交互 | stdout输出 | 游戏数独+中文提示 |

表9-2 程序输出方法与格式

程序处理过程说明：

（1）运行程序，显示程序主菜单

（2）根据需求，输入功能选项，跳转至相应功能

（3）根据表9-1所示输入方法与格式进行程序输入

（4）根据表9-2所示输出方法与格式获取程序输出

（5）在主菜单界面选择继续执行功能或退出程序

程序运行环境说明：

开发环境：Visual Studio 2019 Community

编译器：cl编译器

CPU处理器：Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz

内存：16GB

系统类型：基于x64的处理器

程序配置命令行如下

/external:env:"EXTERNAL\_INCLUDE" /permissive- /ifcOutput "x64\Release\" /GS /GL /W3 /Gy /Zc:wchar\_t /Zi /Gm- /O2 /sdl- /Fd"x64\Release\vc142.pdb" /Zc:inline /fp:precise /D "NDEBUG" /D "\_CONSOLE" /D "\_UNICODE" /D "UNICODE" /errorReport:prompt /WX- /Zc:forScope /Gd /Oi /MD /FC /Fa"x64\Release\" /EHsc /nologo /Fo"x64\Release\" /Fp"x64\Release\Task\_1.1.pch" /diagnostics:column