文 华 学 院

0学号 200501050026 专业 软件工程 班级 软件2001 学生姓名 梁艺潇 **0**

2021～2022学年度第二学期《数据结构课程设计》考查试卷（A卷）

装订线内请勿答题

课程性质：必修 使用范围：本科

考核时间：2022年 月 日

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 考试类型 | 论文 | 大作业 | 调研报告 | 其他 | 总分 |  |
|  |  |  | √ | 阅卷老师签名 |  |

试 题：

主要功能

1. 交通咨询系统的设计并实现

2. 平衡二叉排序树的设计实现

3. 内部排序算法研究

4. 数据结构实验系统

5. 自拟题目

要 求：

1. 每人任选一题，独立完成；
2. 各题须说明课程设计目的、内容和要求
3. 实验数据和实验结果均以计算机截图方式表示；
4. 按设计报告格式完成课程设计报告。



**数据结构**

**课程设计报告**

**课题名称： 基于SAT的数独游戏求解程序**

**姓 名： 梁艺潇**

**学 号： 200501050026**

**院 系： 信息科学与技术**

**专业班级： 软件 2001**

**指导教师： 祝建华**

**完成日期： 2022 年 4 月 6 日**

**目 录**

[**1．课程设计目的 1**](#_Toc102316804)

[**2．课程设计内容和要求 1**](#_Toc102316805)

[**2.1问题描述 1**](#_Toc102316806)

[**2.2设计要求 2**](#_Toc102316807)

[**3．课程设计总体方案及分析 2**](#_Toc102316808)

[**3.1问题分析 2**](#_Toc102316809)

[**3.2系统功能模块设计 5**](#_Toc102316810)

[**3.3测试数据与结果 10**](#_Toc102316811)

[**4．结束语 13**](#_Toc102316812)

[**附录（源代码） 13**](#_Toc102316813)

[**SAT.h: 13**](#_Toc102316814)

[**SAT.cpp： 20**](#_Toc102316815)

[**参考文献 39**](#_Toc102316816)

# 1．课程设计目的

基于“程序设计”综合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“程序设计”综合课程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程序设计综合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。

# 2．课程设计内容和要求

## 2.1问题描述

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

## 2.2设计要求

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。
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]。
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。
4. **时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。
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。
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]。

# 3．课程设计总体方案及分析

## 3.1问题分析

本次课程设计的实现功能重点在于解决SAT问题，其中DPLL算法是经典的SAT完备型求解算法，对给定的一个SAT问题实例，理论上可判定其是否满足，满足时可给出对应的一组解。所以本程序设计的算法与框架基于DPLL算法进行设计与改进。

**DPLL算法思想：**

DPLL算法是一种基于树的回溯算法，主要使用两种基本处理策略：

单子句规则。如果子句集*S*中有一个单子句*L*,那么*L*一定取真值，于是可以从*S*中删除所有包含*L*的子句（包括单子句本身），得到子句集*S*1，如果它是空集，则*S*可满足。否则对*S*1中的每个子句，如果它包含文字*¬L*,则从该子句中去掉这个文字，这样可得到子句集合*S*2。*S*可满足当且仅当*S*2可满足。单子句传播策略就是反复利用单子句规则化简*S*的过程。

分裂策略。按某种策略选取一个文字*L*.如果*L*取真值，则根据单子句传播策略，可将*S*化成*S*2；若*L*取假值（即*¬L*成立）时，*S*可化成*S*1.

根据上述规则可不断对公式化简，并最终达到终止状态，其执行过程可表示为一棵二叉搜索树,如图3.1所示。

**图 3.1 DPLL算法搜索树**

基于单子句传播与分裂策略的DPLL算法可以描述为一个如后所示的递归过程DPLL( *S* ),为了优化执行效率，一般用非递归实现。

DPLL( *S*) :

/\* *S*为公式对应的子句集。若其满足，返回TURE；否则返回FALSE. \*/

{

while(*S*中存在单子句) {//单子句传播

在*S*中选一个单子句*L*；

依据单子句规则，利用*L*化简*S*；

if *S* = Φ return(TRUE);

else if (*S*中有空子句 ) return（FALSE）；

}//while

基于某种策略选取变元*v*；//策略对DPLL性能影响很大

if DPLL（*S* ∪*v* ）return(TURE);

return DPLL(*S* ∪¬*v*);

}

**DPLL改进思路：**

对于DPLL算法可以分为三个部分：

1.选取变元：变元优先选取对当前子句集影响因素较大的变元，从而减少分支选择数,既优先选择单子句或当前子句集中出现次数最多的变元，然后交由第二部分处理。如果没有所选变元则SAT问题求解完成DPLL算法结束。

2.变元赋值与判断：对于非单子句变元则进行赋值操作，赋值为假。单子句则不进行赋值操作。赋值完成则进行判断操作，将选取的变元带入子句集中进行判断，变元的值是否正确，正确则执行第一部分，错误则执行第三部分。

3.回溯：由于变元赋值错误，既子句集中出现无解状态，则开始还原状态。首先清空单子句队列，然后读取栈顶变元，如果该变元为单子句变元，则重复出栈，直到栈顶为非单子句变元，并还原状态。对于非单子句变元先判断变元的值，如果为一次赋值则给变元赋一个相反的值，再次判断没有错误则执行第一部分，否则执行第三部分。如果为二次赋值则还原状态，执行第三部分。

**物理结构设计：**

十字链表节点：

struct crossListNode

{

int id //文字（子句被消除的原因）

bool falseTrue //节点是否消除(子句是否消除)

crossListNode\* up //上节点

crossListNode\* down //下节点

crossListNode\* left //左节点

crossListNode\* right //右节点

};

变元头节点：

struct nodeUphead

{

int leng //影响力

bool falseTrue //变元真假

bool type //是否处理

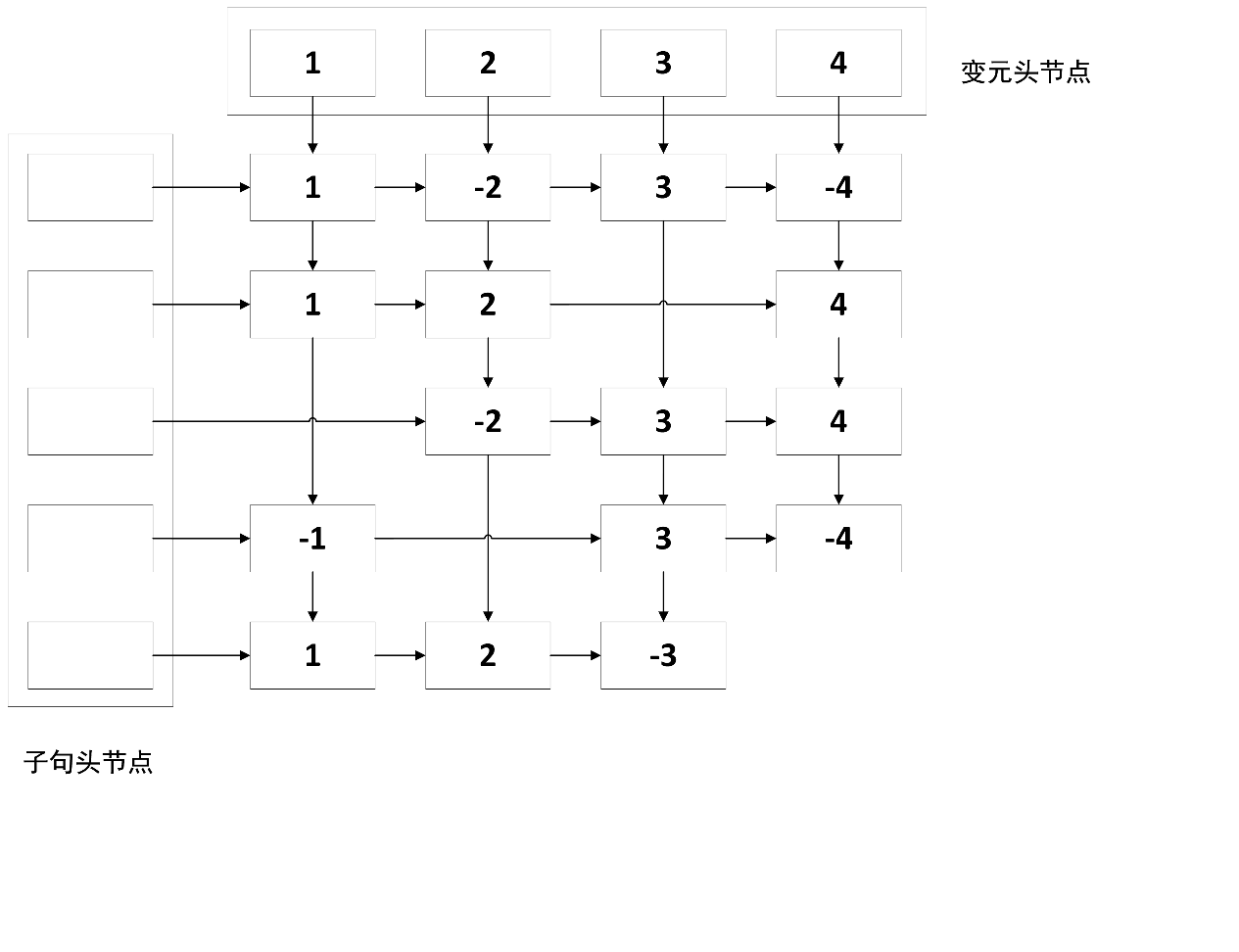
bool danzj //是否作为单子句处理

crossListNode\* begin //变元列的起始地址

crossListNode\* fail //变元列的结束地址

};

创建完成的十字链表的物理结构如图3.2所示。



**图3.2十字链表物理结构**

## 3.2系统功能模块设计

1. 类名：CnfParser

成员变量如表3.3所示：

|  |  |  |
| --- | --- | --- |
| 变量名 | 变量类型 | 作用 |
| boolsize | int | 储存变元的数量 |
| clausesize | int | 储存子句的数量 |
| root | crossListNode\* | 储存十字链表的起始节点 |
| vmap | unordered\_map<int,nodeUphead\*> | 储存变元编号与变元头节点的映射关系 |
| clause | Deque<int>\* | 临时储存单子句的队列 |
| back | Deque<int>\* | 储存处理过的变元用于回溯 |

**表3.3 CnfParser类成员变量表**

成员方法如表3.4所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 方法名 | 输入类型 | 输出类型 | 方法功能 |
| readfile | string s | bool | 将输入的地址s所指向的文件，转化为十字链表,地址错误返回false。 |
| SAT | void | bool | 对转化好的十字链表进行DPLL求解，有解返回true,无解返回false。 |
| SATzj | int wid, bool youhua | bool | 根据输入的变元编号wid在子句集中分析判断。未出现无解状态返回true,出现无解状态返回false。 |
| back\_pop | void | void | 将栈顶元素还原，并出栈。 |

**表3.4 CnfParser类成员方法表**

方法readfile的流程图如图3.5所示，方法SAT的流程图如图3.6所示。

2. 方法名：createSudokuToFile（int holes）

方法功能：根据输入的holes产生数独的cnf文件，并返回文件名。

设计思想：先创建空白棋盘，然后调用createSudoku方法生成数独终盘，然后调用createStartinggrid方法挖洞产生数独初始棋盘，然后调用toCnf方法将初始棋盘输出为cnf文件。

3. 方法名：randomFirstRow(int row[9])

方法功能：将输入的数组随机填充，范围1~9,不重复。

设计思想：通过随机函数发生器对数组进行填充，出现重复则重新填充直到不重复。通过给数独的第一行进行赋值，可以大幅减少后续生成所耗时间。

4. 方法名：Digit(int a[][9], int i, int j)

方法功能：对输入数组的第i行j列进行填充，并调用自身填充。

设计思想：先对填充位置所在的行、列、宫所使用过的数字进行记录，然后通过随机数发生器产生未使用的随机数，并填充到所在位置，然后调用自身填充继续填充，优先填充列，每列填充完成接着填充下一列。

5. 方法名：createSudoku(int cnfSolution[9][9])

方法功能：产生数独终局。

设计思路：调用方式randomFirstRow和Digit生成数独终局。

6. 方法名：createStartinggrid(const int a[][9], int b[][9], int numDigits)

方法功能：根据终局a 产生初始棋盘b。

设计思路：先将棋盘a 复制给b，然后对b进行挖洞，挖numDigits次，每次挖洞通过随机数生成器产生挖洞位置，并储存，如果已经挖过，则重新选择位置。最终产生初始棋盘b。

7. 方法名：showsudoku(int a[][9])

方法功能：将数独棋盘a输出到屏幕。

设计思想：遍历数组a将a中的数字输出。

8. 方法名：toCnf(int a[][9], int holes)

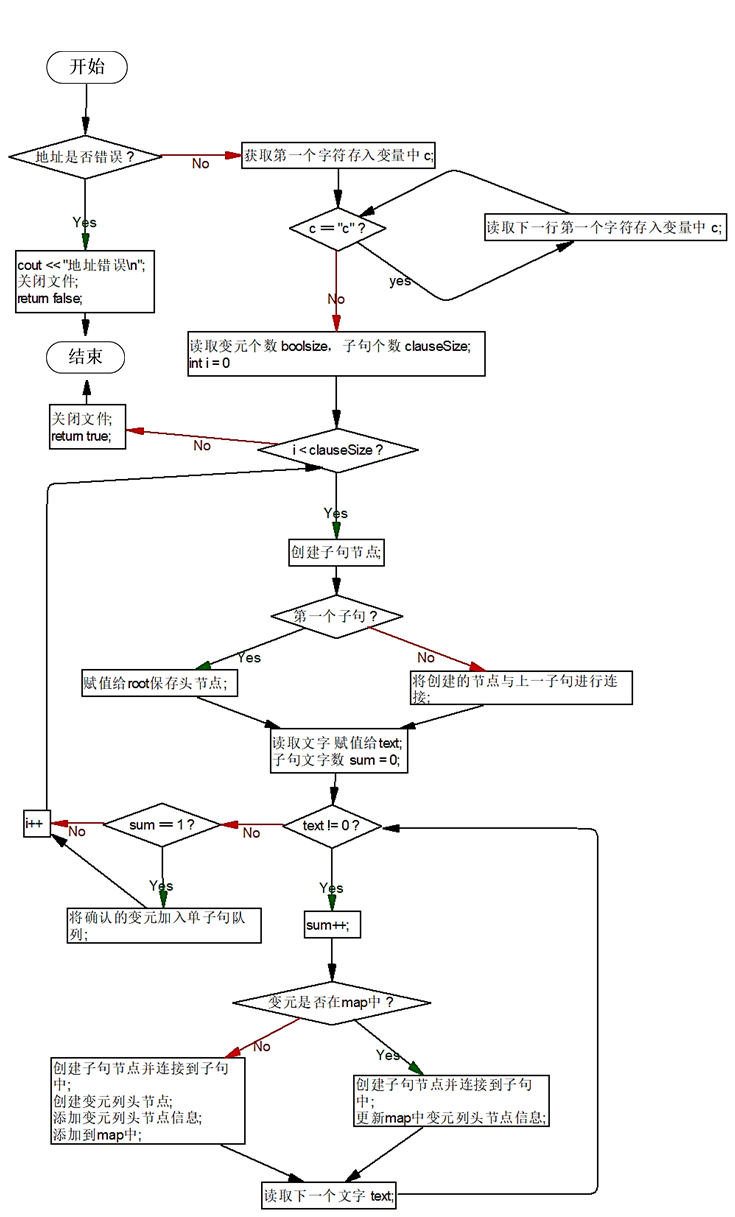
方法功能：将棋盘a转化为cnf文件输出。

设计思想：通过数独转化公式将棋盘转化为cnf文件。

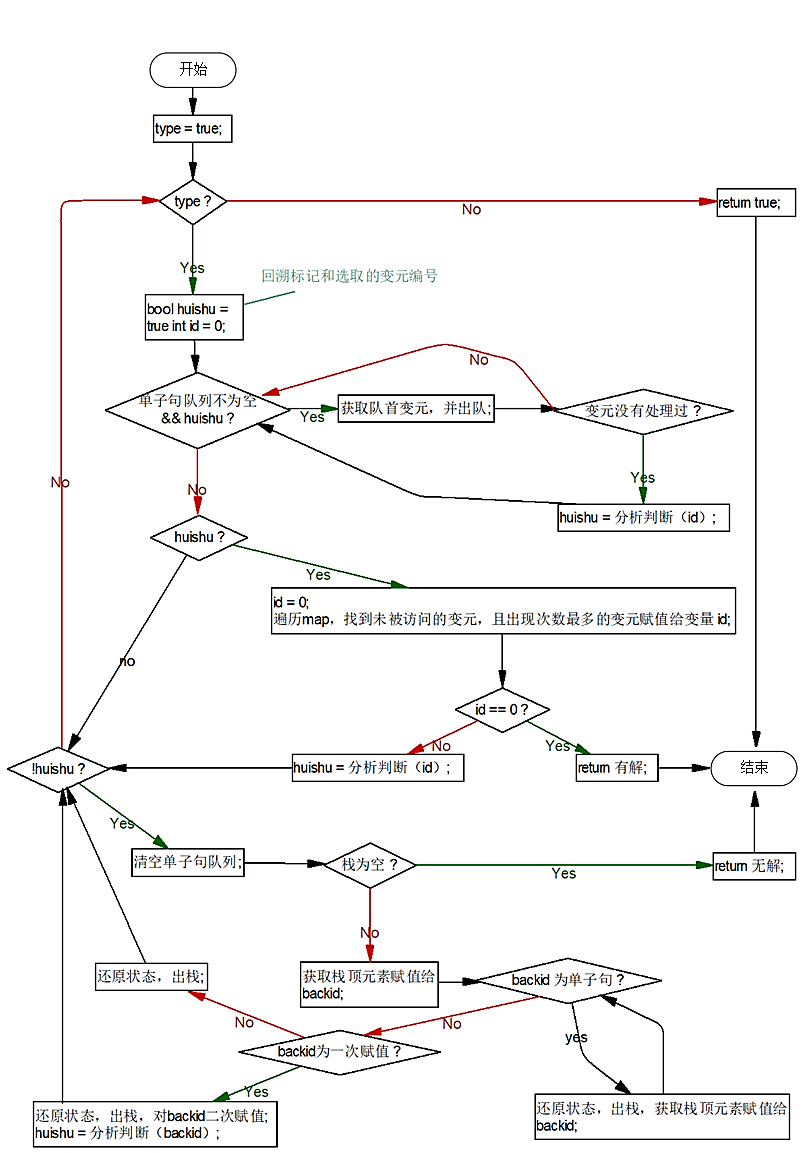
9. 方法名：void showResults(string ptr)

方法功能：将输入的数独解文件进行输出，输出到屏幕。

设计思想：将解出的数独cnf文件读取，根据数独转化规则，把文件转化棋盘输出。



**图3.5 CnfParser类readfile方法流程图**



**图3.6 CnfParser类SAT方法流程图**

## 3.3测试数据与结果

1. CNF算例求解

CNF基准算例如表3.7所示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例属性 | 算例名称 | 消耗时间 |
| 20 | 91 | 功能测试 | sat-20.cnf | 1ms |
| 30 | 420 | 功能测试 | unsat-5cnf-30.cnf | 32ms(无解) |
| 181 | 3151 | 性能测试 | ais10.cnf | 362ms |
| 303 | 2851 | 性能测试 | sud00009.cnf | 337ms |

**表3.7 CNF基准算例**

CNF算例如表3.8所示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例属性 | 算例名称 | 消耗时间 |
| 20 | 91 | S | problem1-20.cnf | 1ms |
| 50 | 80 | S | problem2-50.cnf | 4ms |
| 100 | 340 | S | problem3-100.cnf | 162ms |
| 50 | 100 | S | problem6-50.cnf | 0ms |
| 50 | 300 | S | problem8-50.cnf | 0ms |
| 100 | 200 | S | problem9-100.cnf | 20ms |
| 100 | 600 | S | problem11-100.cnf | 3ms |
| 25 | 100 | S | tst\_v25\_c100.cnf | 0ms |
| 20 | 1532 | S | 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 22ms |

**表3.8 CNF算例**

CNF算例如表3.9所示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例属性 | 算例名称 | 消耗时间 |
| 200 | 320 | M | problem5-200.cnf | 超时 |
| 200 | 1200 | M | problem12-200.cnf | 637ms |
| 301 | 2780 | M | sud00001.cnf | 3273ms |
| 303 | 2851 | M | sud00009.cnf | 326ms |
| 232 | 1901 | M | sud00012.cnf | 996ms |
| 308 | 2911 | M | sud00021.cnf | 1568ms |
| 301 | 2810 | M | sud00079.cnf | 557ms |
| 224 | 1762 | M | sud00082.cnf | 83ms |
| 297 | 2721 | M | sud00861.cnf | 259ms |
| 200 | 210 | M | tst\_v200\_c210.cnf | 1ms |
| 231 | 1166 | M | bart17.shuffled-231.cnf | 72142ms |

**表3.9 CNF算例**

2. 数独游戏测试如表3.10所示

文本

描述已自动生成

**图3.10数独的生成与求解**

3. 哈希表与表的效率测试如表3.11所示

|  |  |  |
| --- | --- | --- |
| 算例 | map花费时间 | HashMap花费时间 |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 45 | 121 |
| problem1-20.cnf | 1 | 1 |
| problem11-100.cnf | 360 | 99 |
| problem2-50.cnf | 362 | 97 |
| problem3-100.cnf | 8519 | 3646 |
| problem6-50.cnf | 13 | 1 |
| problem8-50.cnf | 18 | 4 |
| problem9-100.cnf | 80922 | 591 |
| problem11-100.cnf | 143 | 64 |
| tst\_v25\_c100.cnf | 2 | 0 |
| bart17.shuffled-231.cnf | 149028 | 83308 |
| problem12-200.cnf | 8509 | 631 |
| sud00001.cnf | 6233 | 3179 |
| sud00009.cnf | 794 | 339 |
| sud00012.cnf | 148 | 61 |

**表3.11哈希表与表的效率测试**

4.分析：

（1）对于SAT问题：

通过vs2019的cpu运行统计工具所得数据如图3.12所示，可以得到对于变元的分析判断所站时间为50%，根据变元编号求对应头节点的hash地址所消耗的时间是较多的，又由表3.9可知Hashmap效率是高于map，那么使用数组则可以减少28%的时间消耗，但是由于数独转化为CNF文件过程中存在变元不连续的情况，所以才使用Hashmap作为变元编号与变元头节点的映射。

蓝色屏幕的截图

描述已自动生成

**图3.12 程序中各函数代码在CPU中执行的时间**

（2）对于数独求解：

对于数独求解重点在于生成棋盘，对于生成终盘与挖洞来说消耗时间较为良好，但是对于将数独格局转化为CNF文件存在一定的弊端，选取的转化公式不能生成连续的变元，如所有变元遵从（x，y，z）模式，x，y为坐标，z为坐标的数值，如果将公式改为生成连续的变元那么可以将SAT问题求解器中的Hashmap改为数组，可减少时间消耗，存在优化空间。

# 4．结束语

通过课程设计教学与实践环节，我进一步正确理解与应用了专业知识，增强和提高了分析问题与解决问题的综合能力，加深了对于求解实际问题的基本科研步骤的体会与理解，增强和提升了信息搜索和分析技能，培养了技术总结的基本技能，锻炼了课程设计报告的撰写的能力。

虽然这次课程设计的花了较多的时间，但是程序还是存在较大的优化空间，对于程序的编写也存在一定的不足，如没有完成的模块化，如对将C++/C语言的多个cpp文件的调用不理解，使得数独求解、SAT问题与界面显示求解共同写在了一个C++文件中。

通过这次的实践我也体会到了封装对程序的重要性，将一些繁琐且的方法进行封装，如本次设计中我将十字链表的多种构造方法进行了重载，使得我创建十字链表时，只要对构造函数传入不同的参数，就能完成节点构造与上下左右的节点连接的工作，减少了程序的编写工作，也使得代码更加简洁易读。

在这次课程设计中我也遇到了许多问题，通过问老师和同学最终将这些问题解决，对于学习来说最重要一点那就是勤于请教，不懂就要问，我也非常感谢给我提供帮助的人。希望在后面的学习中我能学到更多，做的更好。

# 附录（源代码）

## SAT.h:

#include<iostream>

#include<string>

#include<fstream>

using namespace std;

template<typename T> //节点模板

struct listnode

{

public:

listnode() :up(nullptr), down(nullptr) {};

listnode(T data) :up(nullptr), data(data), down(nullptr) {};

listnode(listnode\* pup, T data) :up(pup), data(data), down(nullptr) {};

listnode(T data, listnode\* pdoen) :up(nullptr), data(data), down(pdoen) {};

listnode(listnode\* pup, T data, listnode\* pdoen) :up(pup), data(data), down(pdoen) {};

listnode\* up;

listnode\* down;

T data;

};

template<class T>

class Deque //双向链表

{

public:

Deque()

{

head = new listnode<T>();

tail = new listnode<T>();

head->down = tail;

tail->up = head;

size = 0;

};

Deque(const Deque& d)

{

d.cop(this);

};

~Deque();

int getsize();

void push\_back(T data); //在结尾添加

void push\_front(T data); //在开头添加

void pop\_back(); //删除结尾

void pop\_front(); //删除开头

void cop(Deque<T>& cop); //复制函数

void delarr(); //清空

bool empty(); //判空

T back(); //返回结尾

T front(); //返回开头

//遍历器

class iterator

{

private:

listnode<T>\* ptr;

public:

iterator(listnode<T>\* p = nullptr) : ptr(p) {};

listnode<T>\* getptr()

{

return ptr;

}

T& operator\*() const {

return \*ptr;

}

listnode<T>\* operator->() const {

return ptr;

}

iterator& operator++() {

ptr = ptr->down;

return \*this;

}

iterator operator++(int) {

listnode<T>\* tmp = ptr;

// this 是指向list\_iterator的常量指针，因此\*this就是list\_iterator对象，前置++已经被重载过

++(\*this);

return iterator(tmp);

}

bool operator==(const iterator& t) const {

return t.ptr == this->ptr;

}

bool operator!=(const iterator& t) const {

return t.ptr != this->ptr;

}

};

iterator begin()

{

return iterator(head->down);

};

iterator end()

{

return iterator(tail);

};

void emplace(iterator& prt, T data); //添加节点到指定位置

void del(iterator& prt); //删除指定位置的节点

//查找节点

iterator find(T data)

{

for (iterator i = this->begin(); i != this->end(); i++)

{

if (i->data == data)

return i;

}

return this->end();

}

private:

listnode<T>\* head; //队首

listnode<T>\* tail; //队尾

int size;

};

template<class T>

Deque<T>::~Deque()

{

iterator en = end();

iterator beg = begin();

for (iterator it = beg; it != en;)

{

iterator itt = it;

it++;

}

delete(head);

delete(tail);

}

template<class T>

int Deque<T>::getsize()

{

return size;

}

template<class T>

void Deque<T>::push\_back(T data)

{

listnode<T>\* add = new listnode<T>(tail->up, data, tail);

listnode<T>\* pup = tail->up;

pup->down = add;

tail->up = add;

size++;

}

template<class T>

void Deque<T>::push\_front(T data)

{

listnode<T>\* add = new listnode<T>(head, data, head->down);

listnode<T>\* dow = head->down;

dow->up = add;

head->down = add;

size++;

}

template<class T>

void Deque<T>::pop\_back()

{

listnode<T>\* del = tail->up;

listnode<T>\* upn = del->up;

upn->down = tail;

tail->up = upn;

delete(del);

size--;

}

template<class T>

void Deque<T>::pop\_front()

{

listnode<T>\* del = head->down;

listnode<T>\* dow = del->down;

dow->up = head;

head->down = dow;

delete(del);

size--;

}

/\*

复制链表

\*/

template<class T>

void Deque<T>::cop(Deque<T>& cop)

{

for (Deque<T>::iterator i = this->begin(); i != this->end(); i++)

{

cop.push\_back(i->data);

}

cop.size = this->size;

}

/\*

清空链表

\*/

template<class T>

void Deque<T>::delarr()

{

int a = size;

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

{

this->pop\_front();

}

}

/\*

判空

\*/

template<class T>

bool Deque<T>::empty()

{

if (size == 0)

return true;

else

return false;

}

template<class T>

T Deque<T>::back()

{

return T(tail->up->data);

}

template<class T>

T Deque<T>::front()

{

return T(head->down->data);

}

/\*

添加元素

\*/

template<class T>

void Deque<T>::emplace(iterator& prt, T data)

{

listnode<T>\* add = new listnode<T>(prt->up, data, prt.getptr());

listnode<T>\* pup = prt->up;

pup->down = add;

prt->up = add;

size++;

prt = iterator(add);

}

/\*

根据遍历器删除对应节点

\*/

template<class T>

void Deque<T>::del(iterator& prt)

{

if (prt != this->begin() && prt != this->end())

{

prt->up->down = prt->down;

prt->down->up = prt->up;

iterator a = prt;

prt++;

size--;

delete(a.getptr());

}

}

struct crossListNode

{

crossListNode() {};

crossListNode(crossListNode\* up)

{

this->up = up;

this->up->down = this;

};

crossListNode(crossListNode\* list, int data)

{

this->left = list;

this->left->right = this;

id = data;

};

crossListNode(crossListNode\* up, crossListNode\* list, int data)

{

this->up = up;

this->left = list;

id = data;

this->up->down = this;

this->left->right = this;

};

int id = 1e9; //文字

bool falseTrue = false; //节点是否消除(子句是否消除)

crossListNode\* up = nullptr; //上一个节点

crossListNode\* down = nullptr; //下一个节点

crossListNode\* left = nullptr; //左节点

crossListNode\* right = nullptr; //右节点

};

struct nodeUphead

{

nodeUphead() {};

int leng = 0;//包含变元的长度

bool falseTrue = false;//变元真假

bool type = false;//是否处理

bool danzj = false;

crossListNode\* begin = nullptr;//变元列的起始地址

crossListNode\* fail = nullptr;//变元列的结束地址

};

## SAT.cpp：

#include "SAT.h"

#include <time.h>

#include <unordered\_map>

#include <map>

using namespace std;

string createSudokuToFile(int holes); //根据holes来挖洞

void createSudoku(int cnfSolution[9][9]); //生成数独终盘

void randomFirstRow(int row[9]); //随机生成第一行

void createStartinggrid(const int a[][9], int b[][9], int numDigits); //随机生成初盘

bool Digit(int a[][9], int i, int j); //递归生成后i行

void showsudoku(int a[][9]); //打印数独数组

string toCnf(int a[][9], int holes); //转换成CNF文件

void showResults(string ptr,double time); //将数独棋盘的解res文件转化为数独棋盘

class CnfParser

{

public:

CnfParser(){ clause = new Deque<int>();};

~CnfParser();

bool readfile(string s); //读文件

bool SAT();

void back\_pop(); //还原状态

bool SATzj(int wid, bool youhua); //解决SAT问题

void outfileHash(string s, bool type, double time); //输出结果（哈希表）

private:

int boolsize = 0; //变元数量

int clauseSize = 0; //子句数量

crossListNode \*root = nullptr;

unordered\_map<int, nodeUphead \*> vmap = {};

bool jie = true;

Deque<int> \*clause = nullptr; //单子句队列

Deque<int> \*back = new Deque<int>(); //栈(用来回溯)

};

CnfParser::~CnfParser()

{

if (clause != nullptr)

{

delete clause;

clause = nullptr;

}

if (back != nullptr)

{

delete back;

back = nullptr;

}

while (root)

{

crossListNode \*zjdel = root;

root = root->down;

while (zjdel)

{

crossListNode \*del = zjdel;

zjdel = zjdel->right;

delete del;

}

}

}

bool CnfParser::readfile(string s)

{

string c, cnf;

int nodesize = 0, zjsize = 0;

char ccc[161];

ifstream inFile(s, ios::in);

if (!inFile)

{

cout << "地址错误\n";

inFile.close();

return false;

}

inFile >> c;

while (c == "c")

{

inFile.getline(ccc, 160);

inFile >> c;

}

inFile >> cnf >> nodesize >> zjsize;

boolsize = nodesize;

clauseSize = zjsize;

crossListNode \*addlist = nullptr; //每个子句头节点(上一个)

crossListNode \*addnode = nullptr; //子句添加上的节点

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

{

if (i == 0)

root = addnode = addlist = new crossListNode();

else

addnode = new crossListNode(addlist);

int text;

addlist = addnode;

inFile >> text;

int sum = 0;

int dz = 0;

while (text != 0)

{

sum++;

int d = text;

inFile >> text;

int id = d > 0 ? d : -d;

dz = d;

if (vmap.find(id) == vmap.end()) //判断变元是否在map中

{ //不在map中，则添加

new crossListNode(addnode, d); //添加行文字节点

addnode = addnode->right; //右移到新添加的节点

nodeUphead \*add = new nodeUphead(); //添加变元列头节点

add->begin = add->fail = addnode; //设置变元列头节点的头节点和尾节点指针

add->leng++; //变元列的长度加1

vmap[id] = add; //添加到map中

}

else

{ //在map中

new crossListNode(vmap[id]->fail, addnode, d); //添加行文字节点

addnode = addnode->right; //右移到新添加的节点

vmap[id]->fail = addnode; //设置变元列尾节点

vmap[id]->leng++; //变元列的长度加1

}

}

if (sum == 1)

{

if (dz > 0)

{

vmap[dz]->danzj = true;

clause->push\_back(dz);

vmap[dz]->falseTrue = true;

}

else

{

vmap[-dz]->danzj = true;

clause->push\_back(-dz);

vmap[-dz]->falseTrue = false;

}

}

}

inFile.close();

return true;

}

bool CnfParser::SAT()

{

int chulizj = 0;

crossListNode \*nodeptr = root;

Deque<int> dui;

bool type = true;

while (!clause->empty())

{ //对文件原有的单子句处理,不需要回溯(不入栈)

bool a = false;

int id = clause->front();

clause->pop\_front();

if (vmap[id]->type)

{

continue;

}

a = SATzj(id, true);

if (!a)

{

return false;

}

}

while (type)

{

bool huishu = true; //回溯标记

int id = 0;

while (!clause->empty() && huishu)

{

id = clause->front();

clause->pop\_front();

if (vmap[id]->type)

continue;

vmap[id]->danzj = true;

huishu = SATzj(id, false);

}

if (huishu)

{

id = 0;

//遍历map，找到未被访问的变元，且最大的变元

for (auto it = vmap.begin(); it != vmap.end(); it++)

{

if (!it->second->type && !it->second->danzj)

{

if (id == 0)

{

id = it->first;

}

else

{

if (it->second->leng > vmap[id]->leng)

{

id = it->first;

}

}

}

}

if (id == 0)

{ //处理完,结束处理

return true;

}

vmap[id]->falseTrue = false;

huishu = SATzj(id, false);

}

else

{

while (!huishu)

{ //回溯

while (!clause->empty())

{

clause->pop\_front();

}

if (back->empty())

return false;

int backid = back->back();

while (vmap[backid]->danzj)

{ //单子句

back\_pop();

backid = back->back();

}

if (vmap[backid]->falseTrue)

{

vmap[backid]->falseTrue = false;

back\_pop();

}

else

{

vmap[backid]->falseTrue = true;

back\_pop();

huishu = SATzj(backid, false);

}

}

}

}

return true;

}

void CnfParser::back\_pop()

{ //还原状态

int back\_id = back->back(); //回溯对象的编号

vmap[back\_id]->type = false;

crossListNode \*clauseSize = vmap[back\_id]->begin;

if (vmap[back\_id]->danzj)

{

vmap[back\_id]->danzj = false;

vmap[back\_id]->falseTrue = false;

}

while (clauseSize)

{

crossListNode \*zjHead = clauseSize;

while (zjHead->left != nullptr)

{

zjHead = zjHead->left;

}

if (zjHead->id == back\_id)

{ //子句因为当前文字而消除

zjHead->falseTrue = false;

zjHead->id = 1e9;

zjHead = zjHead->right;

while (zjHead)

{

int id = zjHead->id > 0 ? zjHead->id : -(zjHead->id);

vmap[id]->leng++;

zjHead = zjHead->right;

}

}

clauseSize->falseTrue = false;

clauseSize = clauseSize->down;

}

back->pop\_back();

}

bool CnfParser::SATzj(int wid, bool youhua)

{

int id = wid;

crossListNode \*beg = vmap[id]->begin;

bool falseTrue = vmap[id]->falseTrue;

vmap[id]->type = true;

bool type = true;

while (beg)

{

crossListNode \*head = beg;

while (head->left != nullptr)

{ //子句的头节点

head = head->left;

}

beg->falseTrue = true;

if (head->falseTrue)

{ //子句已经被消除的情况

beg = beg->down;

continue;

}

else

{

if (beg->id > 0 && !falseTrue || beg->id < 0 && falseTrue)

{ //消去文字的情况

int leng = 0, danid = 0;

head = head->right;

while (head)

{

if (!head->falseTrue)

{

leng++;

danid = head->id;

}

head = head->right;

}

if (leng == 0)

type = false; //全部消去(回溯)

else if (leng == 1)

{ //单子句

if (danid > 0)

{ //正文字

vmap[danid]->falseTrue = true;

clause->push\_back(danid);

}

else

{

vmap[-danid]->falseTrue = false;

clause->push\_back(-danid);

}

}

}

else if (beg->id < 0 && !falseTrue || beg->id > 0 && falseTrue)

{

head->id = id;

head->falseTrue = true;

head = head->right;

while (head)

{

int vid = head->id > 0 ? head->id : -head->id;

vmap[vid]->leng--;

head = head->right;

}

}

beg = beg->down;

}

}

if (!youhua)

{

back->push\_back(id);

}

return type;

}

void CnfParser::outfileHash(string s, bool type, double time)

{ //哈希表转化为表（k从无序变有序）

map<int, nodeUphead \*> mp;

for (auto it = vmap.begin(); it != vmap.end(); it++)

{

mp[it->first] = it->second;

}

string ss = s;

ss.erase(s.length() - 4);

ss = ss + ".res";

ofstream inFile(ss, ios::out);

if (inFile)

{

inFile << "s ";

if (type)

{

inFile << "1\n" << "v";

//遍历vmap输出fasleTrue,false输出负文字，true输出正文字

for (auto it = mp.begin(); it != mp.end(); it++)

{

if (it->second->falseTrue)

{

inFile << " " << it->first;

}

else

{

inFile << " -" << it->first;

}

}

inFile << "\n";

}

else

{

inFile << "0\n" << "v\n";

}

inFile << "t " << time;

}

inFile.close();

}

int main()

{

CnfParser \*cnfSolution = nullptr;

CnfParser \*sudokuGame = nullptr;

string filePtr;

string ss;

bool type = false;

double time;

int op = 1;

while (op)

{

system("cls");

cout << endl << endl;

cout << "\t\t CnfParser or SudokuGame\n";

cout << "\t++++++++++++++++++++++++++++++++++++++++++++++++++\n";

cout << "\t\t1.CnfParser\t\t2.SudokuGame\n";

cout << "\t\t\t\t0.Exit\n";

cout << "\t++++++++++++++++++++++++++++++++++++++++++++++++++\n";

cin >> op;

system("cls");

switch (op)

{

case 1:

type = false;

time = 0;

cnfSolution = new CnfParser();

cout << "输入文件地址:";

cin >> filePtr;

if (cnfSolution->readfile(filePtr))

{

clock\_t start, finish; //开始,结束时间

start = clock();

type = cnfSolution->SAT();

finish = clock();

time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

cnfSolution->outfileHash(filePtr, type, time);

cout << "花费时间:" << time << "ms" << endl;

}

else

{

cout << "地址错误\n" << endl;

}

system("pause");

delete cnfSolution;

break;

case 2:

type = false;

time = 0;

sudokuGame = new CnfParser();

cout << "输入初始棋盘的数字个数（17~81）：";

int num, holes;

cin >> num;

holes = 81 - num;

if (num < 17 || num > 81)

{

cout << "输入错误，请重新输入\n";

system("pause");

break;

}

clock\_t start, finish; //开始,结束时间

start = clock();

filePtr = createSudokuToFile(holes);

finish = clock();

time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "生成初期棋盘花费时间：" << time << "ms\n";

if (sudokuGame->readfile(filePtr))

{

start = clock();

type = sudokuGame->SAT();

finish = clock();

time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

sudokuGame->outfileHash(filePtr, type, time);

showResults(filePtr,time);

}

else

{

cout << "文件打开失败\n";

}

system("pause");

delete sudokuGame;

break;

case 0:

exit(0);

break;

default:

cout << "\t\t\t输入错误，请重新输入(1~2)\n";

}

}

return 0;

}

string createSudokuToFile(int holes)

{

int cnfSolution[9][9] = {0};

int starting\_grid[9][9] = {0};

//记录所花的时间

clock\_t start, finish;

start = clock();

createSudoku(cnfSolution); //生成数独终盘

finish = clock();

double time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "生成数独终盘花费时间：" << time << "ms\n";

start = clock();

createStartinggrid(cnfSolution, starting\_grid, holes); //随机生成初盘

finish = clock();

time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "生成初盘花费时间：" << time << "ms\n";

cout << "初始棋盘为:" << endl;

showsudoku(starting\_grid);

start = clock();

string filename = toCnf(starting\_grid, holes); //转化为cnf文件

finish = clock();

time = ((double)(finish - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "转化为cnf文件花费时间：" << time << "ms\n";

return filename;

}

void createSudoku(int cnfSolution[9][9])

{

randomFirstRow(cnfSolution[0]);

Digit(cnfSolution, 1, 0);

}

void randomFirstRow(int row[9])

{

int i;

bool flag[10] = {false};

srand((unsigned)time(nullptr));

for (i = 0; i < 9; i++)

{

int j = rand() % 9 + 1;

if (!flag[j])

{

row[i] = j;

flag[j] = true;

}

else

{

j = 1;

while (flag[j])

{

j++;

}

row[i] = j;

flag[j] = true;

}

}

}

/\*

\* 递归生成后i行

\* 参数：int a[][COL]，数独数组；int i，行数；int j，列数

\* 返回值：true 成功 false 失败

\* 由左上开始，按照列填充，每列填充完毕，再填充下一列

\*/

bool Digit(int a[][9], int i, int j)

{ //递归填充数独元素

if (i < 9 && j < 9)

{

int x, y, k;

bool check[9 + 1]; //用于排除已经使用过的的数字

for (k = 0; k <= 9; k++)

{

check[k] = true;

}

for (x = 0; x < j; x++)

check[a[i][x]] = false; //行使用过的数字置为false

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

check[a[x][j]] = false; //列已使用的数字置为false

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

{ //宫使用过的数字置为false, x,y为宫的右下角坐标（边界）

if (x == i)

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

check[a[x][y]] = false;

else

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

check[a[x][y]] = false;

}

bool flag = false;

for (k = 1; k <= 9 && !flag; k++)

{ //从check数组中查找安全的数字

if (check[k] == true)

{

flag = true;

a[i][j] = k;

if (j == 9 - 1 && i != 9 - 1)

{ //列填充完毕，填充下一列

if (Digit(a, i + 1, 0) == true)

return true;

else

flag = false;

}

else if (j != 9 - 1)

{ //列不满，填充下一行

if (Digit(a, i, j + 1) == true)

return true;

else

flag = false;

}

}

}

if (!flag)

{

a[i][j] = 0;

return false;

}

}

return true;

}

void createStartinggrid(const int a[][9], int b[][9], int numDigits)

{ //随机生成初盘

int i, j, k;

srand((unsigned)time(nullptr));

//复制棋盘

for (i = 0; i < 9; i++)

for (j = 0; j < 9; j++)

b[i][j] = a[i][j];

// int c[numDigits][2];

// c为存放挖洞的坐标

int \*\*c = new int \*[numDigits];

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

{

c[i] = new int[2];

}

int m, flag = 0;

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

{

j = rand() % 9;

k = rand() % 9;

flag = 0;

//判断是否已经挖洞

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

if (j == c[m][0] && k == c[m][1])

flag = 1;

if (flag == 0)

{

b[j][k] = 0;

c[i][0] = j;

c[i][1] = k;

}

else

i--;

}

}

void showsudoku(int a[][9])

{

for (int i = 0; i < 9; i++)

{

if (i % 3 == 0 && i != 0)

cout << "\t━━━━━━╋━━━━━━╋━━━━━━━ " << endl;

cout << "\t";

for (int j = 0; j < 9; j++)

{

if (j % 3 == 0 && j != 0)

cout << "┃";

if (a[i][j] == 0)

cout << " .";

else

cout << " " << a[i][j];

}

cout << endl;

}

}

/\*

\* 将数独棋盘转换为cnf文件

\* 参数：int a[][9]，数独棋盘；int holes 挖洞个数

\*/

string toCnf(int a[][9], int holes)

{

ofstream outfile(".\\cnfSolution.cnf"); //输出文件

if (!outfile.is\_open())

cout << "打开文件失败" << endl;

//将棋盘以注释的形式输出

outfile << "c 数独棋盘：" << endl;

for (int i = 0; i < 9; i++)

{

if (i % 3 == 0 && i != 0)

outfile << "c\t━━━━━━━╋━━━━━━━╋━━━━━━━━ " << endl;

outfile << "c\t";

for (int j = 0; j < 9; j++)

{

if (j % 3 == 0 && j != 0)

outfile << "┃";

if (a[i][j] == 0)

outfile << " .";

else

outfile << " " << a[i][j];

}

outfile << endl;

}

outfile << "c" << endl;

//第一行，第一个数字为变量个数，第二个数字为约束个数

outfile << "p cnf 729 " << 8829 + 81 - holes << endl;

for (int x = 0; x < 9; ++x)

{//第（x，y）格有初始值，生成单子句

for (int y = 0; y < 9; ++y)

if (a[x][y] != 0)

outfile << (x + 1) \* 100 + (y + 1) \* 10 + a[x][y] << " " << 0 << endl;

}

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

{// 每格只能有一个数字总约束（1~9）

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

{

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

outfile << x \* 100 + y \* 10 + z << " ";

outfile << 0 << endl;

}

}

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

{// 行约束

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

for (int x = 1; x <= 8; ++x)

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

outfile << 0 - (x \* 100 + y \* 10 + z) << " "

<< 0 - (i \* 100 + y \* 10 + z) << " " << 0 << endl;

}

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

{// 列约束

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

for (int y = 1; y <= 8; ++y)

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

outfile << 0 - (x \* 100 + y \* 10 + z) << " "

<< 0 - (x \* 100 + i \* 10 + z) << " " << 0 << endl;

}

// 宫约束

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

outfile << 0 - ((3 \* i + x) \* 100 + (3 \* j + y) \* 10 + z) << " "<< 0 - ((3 \* i + x) \* 100 + (3 \* j + k) \* 10 + z) << " " << 0 << endl;

}

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

outfile << 0 - ((3 \* i + x) \* 100 + (3 \* j + y) \* 10 + z) << ' '<< 0 - ((3 \* i + k) \* 100 + (3 \* j + l) \* 10 + z) << ' ' << 0 << endl;

}

outfile.close();

return ".\\cnfSolution.cnf"; //返回一个string类型的对象

}

void showResults(string ptr,double time)

{

string ss = ptr;

ss.replace(ss.end() - 4, ss.end(), ".res");//将地址后缀改为.res

ifstream infile(ss);

if (!infile.is\_open())

cout << "打开文件失败" << endl;

else

{

char c;

int a[9][9];

infile >> c;

if (c == 's')

{

infile >> c;

if (c == '1')

{

cout << "\n解出数独花费时间:" << time << "ms" << endl;

cout << "解出的棋盘：\n";

string s;

infile >> c;

int i,j,k;

if (c == 'v')

{//判断s的第一位不为‘v’

infile >> s;

while (s.at(0) != 't')

{//将s中的数字转换为整数

int num = 0;

num = atoi(s.c\_str());

if (num > 0)

{

i = num / 100;

j = (num % 100) / 10;

k = num % 10;

a[i - 1][j - 1] = k;

}

infile >> s;

}

showsudoku(a);

}

}

else

cout << "求解失败！" << endl;

}

}

infile.close();

}

# 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

[11] 赵伟楠. 对可满足性（SAT）问题求全解的算法研究及实现. 硕士学位论文. 北京交通大学