

**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： CS2006**

**学 号： U202015471**

**姓 名： 杨释钧**

**指导教师： 纪俊文**

**报告日期： 2021年10月7日**

**计算机科学与技术学院**

**目录**

[**任务书** I](#_Toc84784217)

[1. 设计内容 I](#_Toc84784218)

[2. 设计要求 I](#_Toc84784219)

[3. 参考文献 II](#_Toc84784220)

[1 **引言** 3](#_Toc84784221)

[1.1 课题背景与意义 3](#_Toc84784222)

[1.1.1 sat问题简介 3](#_Toc84784223)

[1.1.2 数独问题简介 3](#_Toc84784224)

[1.2国内外研究现状 3](#_Toc84784225)

[1.3课程设计的主要研究工作 4](#_Toc84784226)

[2**系统需求分析与总体设计** 6](#_Toc84784227)

[2.1系统需求分析 6](#_Toc84784228)

[2.2系统总体设计 6](#_Toc84784229)

[3**系统详细设计** 8](#_Toc84784230)

[3.1有关数据结构的定义 8](#_Toc84784231)

[3.2主要算法设计 9](#_Toc84784232)

[3.2.1 sat问题求解模块 9](#_Toc84784233)

[3.2.2 数独游戏模块 11](#_Toc84784234)

[4**系统实现与测试** 16](#_Toc84784235)

[4.1系统实现 16](#_Toc84784236)

[4.1.1 实现与测试环境 16](#_Toc84784237)

[4.1.2 系统功能说明 16](#_Toc84784238)

[4.1.3 常量声明 17](#_Toc84784239)

[4.1.4 函数声明 17](#_Toc84784240)

[4.1.5 main函数 24](#_Toc84784241)

[4.2系统测试 24](#_Toc84784242)

[4.2.1 sat模块运行测试 24](#_Toc84784243)

[4.2.2 数独游戏模块运行测试 34](#_Toc84784244)

[5总结与展望 41](#_Toc84784245)

[5.1全文总结 41](#_Toc84784246)

[5.2工作展望 41](#_Toc84784247)

[6**体会** 43](#_Toc84784248)

[**参考文献** 44](#_Toc84784249)

[**附录** 45](#_Toc84784250)

# 

# 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

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

# 1 引言

## 1.1 课题背景与意义

### 1.1.1 sat问题简介

Sat问题全称是布尔可满足性问题（Boolean Satisfiability problem）,是第一个被证明的NP完全类问题。

在计算机科学中，布尔可满足性问题（有时称为命题可满足性问题，缩写为SATISFIABILITY或SAT）是确定是否存在满足给定布尔公式的解释的问题。换句话说，它询问给定布尔公式的变量是否可以一致地用值TRUE或FALSE替换，公式计算结果为TRUE。如果是这种情况，公式称为可满足。另一方面，如果不存在这样的赋值，则对于所有可能的变量赋值，公式表示的函数为FALSE，并且公式不可满足。例如，公式“a AND NOT b”是可以满足的，因为可以找到值a = TRUE且b = FALSE，这使得（a AND NOT b）= TRUE。相反，“a AND NOT a”是不可满足的。

这个决策问题在计算机科学的各个领域都至关重要，包括理论计算机科学，复杂性理论，算法，密码学和人工智能。

### 1.1.2 数独问题简介

数独是源自18世纪瑞士的一种数学游戏。是一种运用纸、笔进行演算的逻辑游戏。玩家需要根据9×9盘面上的已知数字，推理出所有剩余空格的数字，并满足每一行、每一列、每一个粗线宫（3\*3）内的数字均含1-9，不重复。

数独盘面是个九宫，每一宫又分为九个小格。在这八十一格中给出一定的已知数字和解题条件，利用逻辑和推理，在其他的空格上填入1-9的数字。使1-9每个数字在每一行、每一列和每一宫中都只出现一次，所以又称“九宫格”。

## 1.2国内外研究现状

目前，SAT问题按照可满足性可划分为可满足的SAT类和不可满足的UNSAT类；按照问题产生的来源可划分为应用类(application)、随机生成类 (random)和难以满足的组合类(hardcombination)。不同的SAT求解算法在相同问题上或同一求解算法在不同问题上的表现会有很大不同。典型的SAT求解算法包括完备算法和不完备算法两大类。完备性算法，由于其完备性的搜索技术，能判定一个 SAT 实例是可满足的还是不可满足的，但有可能不能在合理的时间对 SAT 实例进行判定。非完备性算法，通常指局部搜索算法，仅能判定一个 SAT 实例是可满足的，但其能非常高效地对可满足的 SAT 实例进行判定。目前很多研究人员尝试成功混合不同求解策略或并行多个求解器，使其逐渐发展为一个新的研究热点，有部分学者将其称之为组合算法。

目前常用的完备求解算法主要有DPLL算法，CDCL算法，LA算法。DPLL的重要思想在于穷举、分支回溯和布尔约束传播(BooleanConstraintPropagation，BCP)，对所有变量进行深度搜索以形成一棵完全二叉树的遍历过程，如果问题是可满足的，则给出所有解；如果问题是不可满足的，则给出完备性证明。若公式中所有变量都被赋值，且没有发现一个冲突子句，那么该布尔公式是可满足的，否则是不可满足的，由于算法本身效率较低并且受到计算机硬件性能的限制，只能用来求解小规模的SAT问题。CDCL算法是目前完备求解算法中最重要的一类，它基于DPLL算法的框架，但是在预处理、分支冲突、数据结构等方面做了相应的改进。LA算法同样基于DPLL算法的框架，但是不同于前两种暴力求解算法，LA算法侧重于通过大量的附加推理以及选择有效决策变量构建一个小巧平衡的搜索树来求解问题，在某些问题上优于前两种算法。

目前绝大多数的不完备求解算法都不能判断SAT问题的不可满足性，但是在处理大部分随机生成的可满足性问题时性能更好，常用的有SLS算法，MP算法，EA算法，后两种算法均基于SLS算法。SLS算法在求解过程中首先为一个布尔变元进行赋值，然后进行搜索，查看是否满足，是则停止搜索，否则重复上述过程。MP算法是一种启发式算法，通过计算全局信息来引导搜索过程。EA算法的基本思想是将SAT问题转化成一个求相应目标函数最值的优化问题。

目前的组合求解法主要有混合求解和并行求解。

## 1.3课程设计的主要研究工作

本次课程设计主要要求依据DPLL算法设计SAT问题求解器，能够求解相应的中小型算例，输出求解结果，计算求解时间，要求功能如下：

1. 设计合适的数据结构对CNF范式进行存储，能够成功读取CNF文件，基于一定的物理结构建立公式的内在表示，并且能够逐行输出读取结果。

2. 基于DPLL算法框架，对SAT问题进行求解。

3. 基于相应的时间处理函数，衡量DPLL算法求解所花费的时间。

4. 对基本DPLL的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现，提供明确的性能优化率结果。

5. 将数独问题转化成SAT求解问题，将其集成到SAT求解器中，要求游戏可玩，并且具有一定的交互性。

2系统需求分析与总体设计

## 2.1系统需求分析

对用户而言，本系统可以进行sat问题求解，也可以玩数独游戏。对于sat问题求解这一方面，需要根据已有的CNF文件，读取CNF范式，并且通过合适的数据结构实现高效的存储运算，然后用户可以选择输出保存后的CNF范式，也可以选择使用系统给出的DPLL算法对sat问题进行求解，并能够将求解结果保存在res文件中，同时给出相应的求解时间。对于数独游戏这一方面，首先需要给出一个数独游戏的棋盘，然后在游戏过程中需要给出一定的交互性，用户需要能够选择填入或删除棋盘中的某些数据，最后用户也应当可以要求给出数独游戏的终盘答案。

根据以上分析，我们首先需要设计一个合适的数据结构，用于读入CNF文件，并且需要能够设计对其进行增加、删除、修改、查找等操作，然后需要依托于该数据结构使用DPLL算法进行求解。另外需要设计一套编码体系，将数独游戏的整个棋盘转换为CNF范式，使用DPLL算法进行求解。最后我们的系统应该有一定的交互性，能够让用户在不知道整体的设计思路的情况下使用最终的程序。

## 2.2系统总体设计

根据之前的分析，整个系统应该包含两个大模块，分别是sat问题求解模块和数独游戏模块，在整个系统最开始的时候应该由用户选择进入sat问题求解模块还是数独游戏模块。

sat问题求解模块主要包括以下子模块：

1. ReadCNF模块，主要用来根据输入的文件路径读取CNF文件，并将其保存到设计好的数据结构中。

2. ShowCNF模块，主要用来逐行打印读取到的CNF范式。

3. BetterDPLL模块，主要用来使用优化过后的DPLL算法来求解sat问题并给出求解时间。

4. DPLL模块，主要用来使用优化之前的DPLL算法来求解sat问题并给出求解时间。

5. exit模块，主要用来退出当前的sat问题求解模块。

数独游戏模块主要包含以下子模块：

1. CreateSudoku模块，主要用来使用挖洞法生成数独游戏棋盘。

2. PlaySudoku模块，主要用来玩数独游戏，其中仍然包含一定的子模块，即用户可以选择在某位置加数，可以选择在某位置删除数，也可以选择输出正确答案。

3. exit模块，主要用来退出当前的数独游戏模块。

整个系统的逻辑如图2-1所示。

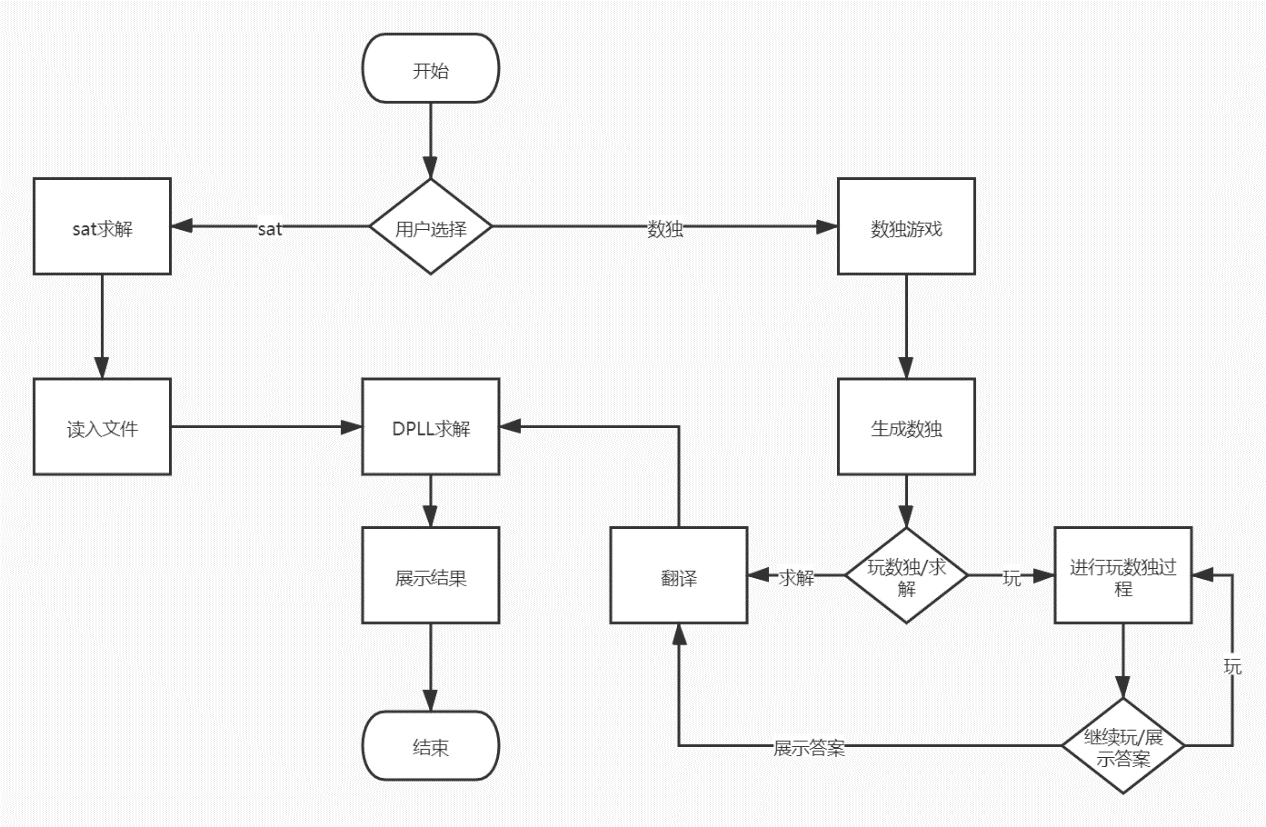


图2-1 总体设计

# 3系统详细设计

## 3.1有关数据结构的定义

整个系统中要处理的核心数据就是CNF公式，我们容易看出，在每个CNF文件的开始，由‘c’开头的是若干注释说明行；‘p’开头的行说明公式的总体信息，包括：范式为CNF；公式有200个布尔变元，由1到200的整数表示；320个子句。之后每行对应一个子句，0为结束标记。46表示第46号变元，且为正文字；-46则是对应的负文字，文字之间以空格分隔。这不难让我们想到，我们可以利用邻接表储存CNF范式。

typedef struct \_charanode

{

    int data;

    struct \_charanode \*next;

} charanode;

typedef struct \_cnf

{

*//int length;*

    charanode \*head;

    struct \_cnf \*next;

} cnf;

cnf的结构实际上就是子句的结构，由于每个子句由若干个文字组成，所以子句中包含了一个文字指针，或者说指向该子句的第一个文字的指针，然后包含了指向下一个子句的指针，而charanode结构保存了文字的信息，包括当前文字的数值以及指向下一个文字的指针，数据在系统之间的关联如图3-1所示。

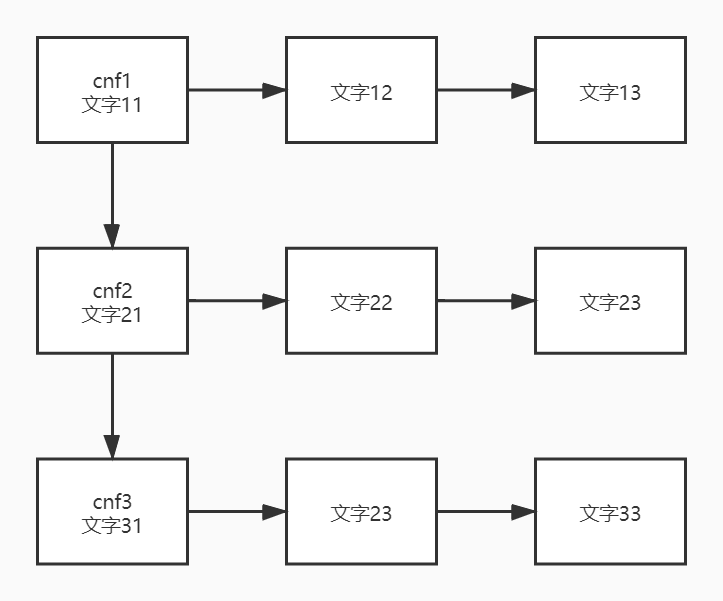


图3-1 数据结构示意图

## 3.2主要算法设计

### 3.2.1 sat问题求解模块

该模块的实现主要采用了以下函数：

int readCNF(cnf \*&*CNF*, char *Filename*[80]);

void deleteCNF(cnf \*&*CNF*);

void showCNF(cnf \**CNF*);

int isOneClause(charanode \**chara*);

int deleteClause(cnf \*&*CNF*, cnf \*&*deleted*);

int deletechara(charanode \*&*head*, charanode \*&*CNF*);

int iswithEmptyClause(cnf \**CNF*);

int betterDPLL(cnf \*&*CNF*, int *value*[]);

void addClause(cnf \**CNF*, cnf \*&*root*);

void copyclause(cnf \*&*CNF*, cnf \**CNF2*);

int savedata(int *value*[], char *Filename*[], double *time*, int *result*);

sat问题求解主要采用DPLL算法。

DPLL算法基于二叉树的回溯搜索算法，主要有两种基本策略：

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S1，如果它是空集，则S可满足。否则对S1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S2。S可满足当且仅当S2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S2；若L取假值（即¬L成立）时，S可化成S1。

事实上在实现的过程中尝试采用了重言子句策略：如果一个子句是重言子句（一般是子句中存在互补的文字），那么可以将他从合取范式中删除。但是对DPLL算法效率提升并没有太大帮助。

我是这样考虑进行DPLL算法的，首先开始是一个大循环，如果遍历位置非空，就持续进行，然后在大循环中，利用单子句规则和分裂策略来简化整个邻接表，调用函数isOneClause来判断目前的子句是否是单子句，进而找到第一个单子句，然后利用分裂规则，调用函数delelteClause和deletechara来删除不必要的子句和文字，接下来进行相关的判定，其中调用了函数iswithEmptyClause来判断是否有空子句，如果有的话说明该函数一定不能被满足，这个时候程序终止，直接返回0即可，如果整个CNF范式为空，说明被化简完了，这个时候它一定是可满足的，直接返回1即可，其中的答案使用value数组存储。化简结束后需要进行回溯，这里的变元选取策略是选择出现次数最多的正文字，如果没有正文字的话，选择出现次数最多的负文字，当然这是优化后的DPLL算法，在优化前，我仅仅选用了比较简单的遍历统计出现次数最多的变元，而这样的效率在大多数情况下都是比较低的，另外，这里的统计思路是开一个数组在遍历邻接表的过程中记录所有变元出现的次数，然后再进行循环统计出现的最大次数即可。在选取完变元之后，直接进行回溯即可。该算法流程可参考图3-2。

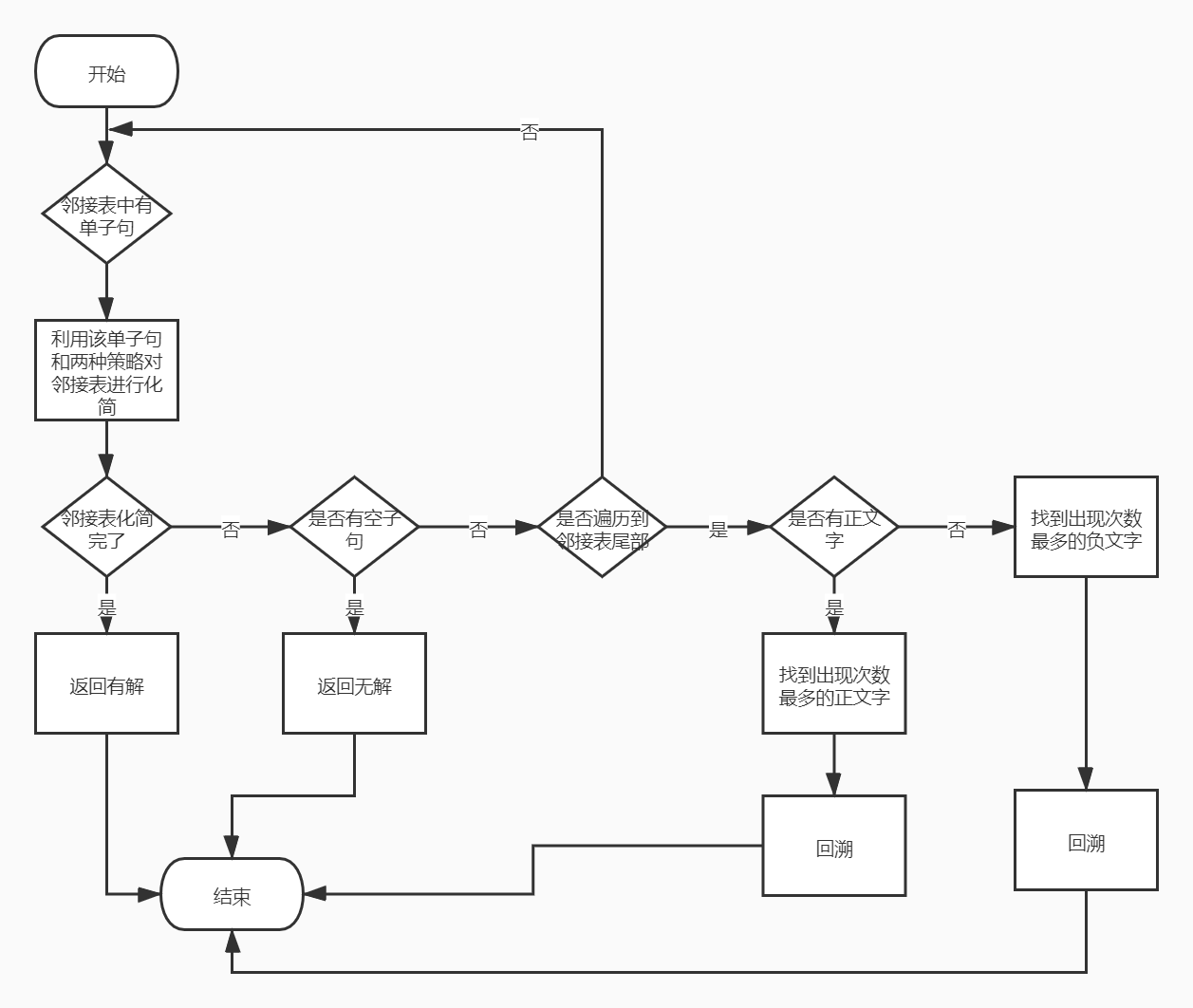


图3-2 DPLL算法流程图

### 3.2.2 数独游戏模块

该模块的实现主要运用了以下函数：

void createsudoku(int *sudoku*[][9]);

void dfs();

int solveSudoku(int *sudoku*[][9]);

void dighole(int *sudoku*[][9], int *n*);

int check(int *sudoku*[][9], int *x*, int *y*);

void addClause(cnf \**CNF*, cnf \*&*root*);

对于数独游戏模块，最重要的部分应该就是生成数独棋盘以及将数独游戏转化成CNF范式进而求解。

对于将数独游戏转化成CNF范式这一问题，这里主要参考了课程设计任务书中给出的方式。变元可按语义编码为1～9之间数字构成的三位整数*ijk*，*i*, *j*, *k*∈{1,2,…,9}，其中*i*表示单元格的行号，*j*表示单元格的列号，*k*表示单元格<*i*, *j*>填入的数字为*k*。如163变元表示第1行6列填入3；负文字 -452表示第4行5列不填入2。这样编码共有729个变元。

数独游戏的基本要求是：每个单元格只能填入唯一一个数字。以单元格<1,1>例，这可以表示为如下子句：

111 112 113 114 115 116 117 118 119 0

-111 -112 0

-111 -113 0

……

-118 -119 0

上述表示中，每个子句的末尾的0表示结束标记；第一个子句的含义是单元格<1,1>可填入至少一个数字；后面的子句集共同表示只能填入一个数字，子句-111 -112 0表示不能同时填1与2；其它类推。按这种方式需要对81个单元格进行类似表示，得到对应的子句集。

行约束要求每行需要填入1～9中的每个数字，且每个数字只出现一次。以第1行为例可表示为（此处在每个子句后加入注释，说明子句的含义）：

111 121 131 141 151 161 171 181 191 0 第1行含有1

112 122 132 142 152 162 172 182 192 0 第1行含有2

… …

119 129 139 149 159 169 179 189 199 0 第1行含有9

-111 -121 0 前两格不同时为1

-111 -131 0 第1与第3格不同时为1

… …

-111 -191 0 第1与第9格不同时为1

… …

列约束仿照行约束易于表示为对应子句集，同学们可自行写出。

对于3×3的盒子约束，以左上角的盒子为例进行说明，其子句集可表示如下：

111 121 131 211 221 231 311 321 331 0 包含1

… …

119 129 139 219 229 239 319 329 339 0 包含9

-111 -211 0 11格与21格不同时为1

-111 -121 0 11格与12格不同时为1

… …

最后，对于每个具体的数独游戏，已经填入了部分提示数，如图2.3中的左图，每个提示数可表示为一个单子句，如第2行3列填入5，对应单子句如下：

235 0

SAT公式CNF文件中，一般变元是从1进行连续编码的，可以将上述语义编码转换为自然顺序编码，公式为：*ijn* → (*i*-1)\*81+(*j*-1)\*9+*n*；根据上述分析，整个数独游戏可以转换成一个大的CNF范式，整个转换方式应当以多重循环的方式实现，流程图见图3-3。

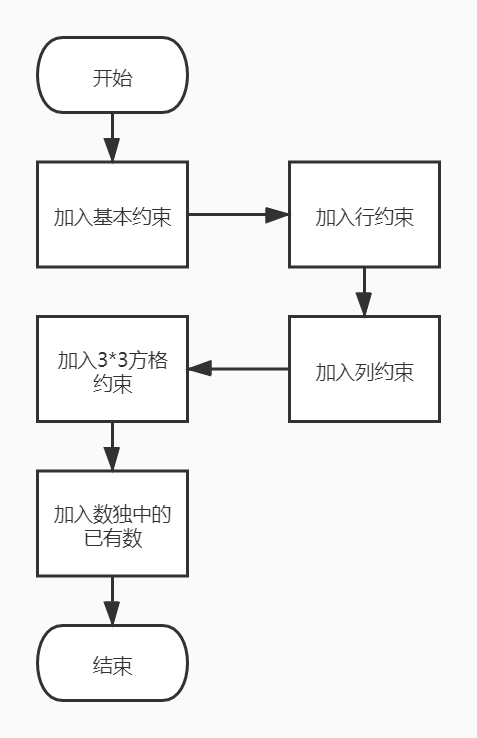


图3-3 数独转换

对于生成数独棋盘，本系统采用了挖洞法，即先生成数独终盘，然后从其中随机挖洞，每次挖一个洞就判断一次，判断是否有唯一解，其判断方法也很简单，即假设该洞填了别的数，使用DPLL算法查看其是否有解，如果有解，则说明它不是唯一解，该洞不可挖，否则该洞可挖。另外值得一提的就是，在最开始挖洞的时候，我们需要首先随意挖出第一个洞，因为第一个洞不管挖到哪肯定都是具有唯一解的。此处挖洞的数量由用户输入。终盘是这样生成的，即首先随机生成1-9的全排列，然后对其进行相应的移动，进而得到一个数独终盘。生成数独棋盘算法流程图见图3-4，判断是否有唯一解流程图见图3-5。

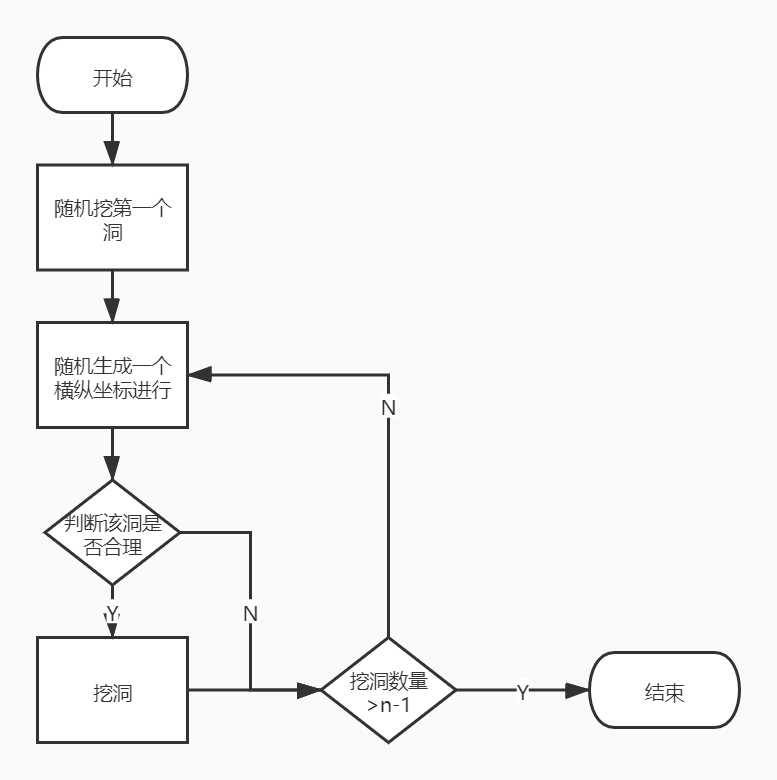


图3-4 生成棋盘

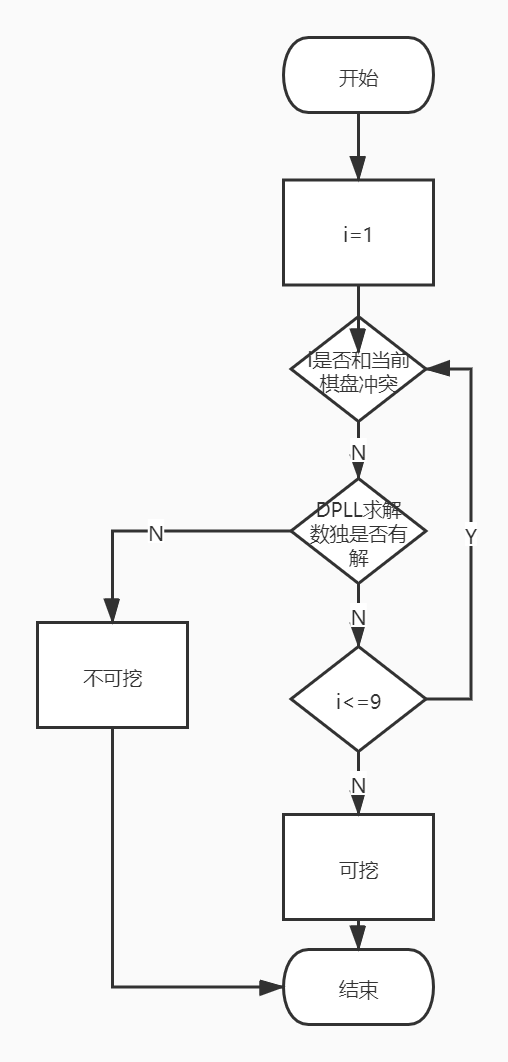


图3-5 挖洞法流程图

# 4系统实现与测试

## 4.1系统实现

### 4.1.1 实现与测试环境

硬件环境：

* 处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
* 机带RAM：16.0GB(15.4GB可用)
* 系统类型：64位操作系统，基于x64的处理器

编译器：gcc

代码编辑器：vscode

### 4.1.2 系统功能说明

系统包括两个部分，即SAT模块和Sudoku模块。

SAT模块是求解命题逻辑公式可满足性问题的模块，用户需要根据提示输入.cnf文件的路径，然后根据相应的功能选择实现sat问题求解。

当用户输入1选择SAT模块时，系统将会出现SAT模块所具有的功能编号，其中包括：

1. ReadCNF，该功能根据用户输入的路径名读取相应的CNF文件，将其存储到设计好的数据结构中；

2. ShowCNF，该功能可以逐行打印读入的CNF文件，便于用户判断导入的CNF文件是否正确；

3. BetterDPLL，该功能用来调用优化后的DPLL算法来进行求解sat问题，该功能将最后求解完毕得到的答案保存在.res文件中，并且给出了求解所需要的时间，用户可以使用verify.exe文件来验证求解的正确性；

4. DPLL，该功能用来调用优化前的DPLL算法来进行求解sat问题，该功能将最后求解完毕得到的答案保存在.res文件中，并且给出了求解所需要的时间，用户可以使用verify.exe文件来验证求解的正确性；0. exit，该功能用来退出当前模块。

用户应当根据提示一步一步进行相应操作，否则有可能出现错误。

Sudoku模块是对于数独游戏求解的模块，用户需要根据提示输入初始化棋盘的信息，系统会自动随机生成终盘，后根据用户输入进入相应功能。

当用户输入2选择Sudoku模块时，系统将会出现Sudoku模块所具有的功能编号。其中包括：

1. CreateSudoku，该功能用来提示用户输入挖洞数量，根据该数据创建棋盘，随后给出生成完毕的棋盘；

2. PlaySudoku，该功能可以让用户进行玩数独游戏时进行选择，包括在某一点加数据，在某一点删除自己加的数据，或者给出最终的答案；

3. SaveAns，该功能可以保存最终的终盘；

0. exit，该功能可以退出当前的模块。

用户应当根据提示一步一步进行相应操作，否则有可能出现错误。

### 4.1.3 常量声明

所使用的常量如下。

#define TRUE 1*//判断为真*

#define FALSE 0*//判断为假*

#define OK 1*//执行操作成功*

#define ERROR 0*//执行操作失败*

#define INFEASTABLE -1*//结构体未分配存储空间*

#define OVERFLOW -2*//发生溢出，分配存储空间失败*

### 4.1.4 函数声明

首先本系统模块所用到的函数的调用关系如图4-1所示。



图4-1 函数调用关系

各函数如下详细介绍如下：

（1）int readCNF(cnf \*&CNF, char Filename[80]);//读入CNF文件，存入数据结构

函数形参：cnf指针引用，char字符串。

主要功能：读入CNF文件，将文件中的CNF范式保存到系统数据结构中，这也就实现了SAT模块的第一个功能。

实现思路：首先注意到CNF文件均具有以下特征，即在每个CNF文件的开始，由‘c’开头的是若干注释说明行；‘p’开头的行说明公式的总体信息，包括：范式为CNF；公式有n个布尔变元，由1到n的整数表示；m个子句。之后每行对应一个子句，0为结束标记。46表示第46号变元，且为正文字；-46则是对应的负文字，文字之间以空格分隔。那么我们首先需要为CNF指针分配存储空间，然后为其中的文字头指针分配空间，然后创建文件指针，打开需要读取的文件。接下来逐步处理CNF文件中的内容。

首先处理注释部分，如果最开始读入的字符为c，那么该行为注释行，只需要一直读取下一个字符直到换行符即可，这样初始部分就读取完毕。

然后继续读取，当读到p字符的时候，说明来到了说明公式总体信息的一行，然后读取布尔变元数和子句数即可。

接下来读入的每一行都是一个子句，只需要最开始创建一个新的子句指针，然后创建它对应的文字头指针，接下来按照尾插法或者首插法创建当前的子句即可。

时间复杂度：O（mn）

（2）void showCNF(cnf \*CNF);//遍历整个邻接表，逐行打印CNF范式

函数参数：cnf指针。

函数功能：遍历整个邻接表，逐行打印CNF范式。这也就实现了SAT模块的第二个功能。

实现思路：其实就是遍历整个邻接表，将邻接表中的所有数据打印出来

时间复杂度：O（mn）

（3）int isOneClause(charanode \*chara) //判断该子句是否为单子句

函数参数：charanode指针。

函数功能：判断该子句是否为单子句，是则返回1，否则返回0。

实现思路：只需要判断当前的头指针是否有后继元素即可。

时间复杂度：O（1）

（4）int deleteClause(cnf \*&CNF, cnf \*&deleted) //删除邻接表中的deleted子句

函数参数：cnf指针引用，cnf指针引用。

函数功能：删除邻接表中的deleted子句，并且释放deleted所分配的空间，删除成功返回1，否则返回0。

实现思路：如果被删除的子句是邻接表中的第一个子句，只需要让当前的头指针指向下一个子句，并释放当前子句所占有的空间，否则需要遍历整个邻接表，并删除该子句，释放所有空间。

时间复杂度：O（m）

（5）int deletechara(charanode \*&head, charanode \*&deleted) //删除当前子句中的某个文字

函数参数：charanode指针引用，charanode指针引用。

函数功能：删除当前子句中的deleted文字，并且释放deleted所分配的空间，删除成功返回1，否则返回0。

实现思路：整体和deleteClause函数类似，如果被删除的文字是当前子句中的第一个文字，只需要让当前的头指针指向下一个文字，并释放当前文字所占有的空间，否则需要遍历整个子句，找到该文字并删除该文字，释放所有空间。

时间复杂度：O（n）

（6）int iswithEmptyClause(cnf \*CNF) //判断其是否含有空子句

函数参数：cnf指针。

函数功能：判断当前邻接表中是否含有空子句，是则返回1，否则返回0。

实现思路：其实就是需要对当前的邻接表进行一个遍历，如果某个子句的头文字指针为空，说明该子句是一个空子句，返回1即可。如果没有，返回0即可。

时间复杂度：O（n）

（7）void copyclause(cnf \*&CNF, cnf \*CNF2) //该函数将CNF2的值赋到CNF

函数参数：cnf指针引用，cnf指针。

函数功能：复制当前的邻接表，进而进行回溯。

实现思路：首先初始化CNF，为其分配存储空间，然后为其文字指针分配空间，接下来用两重循环遍历整个邻接表，在内重循环中为拷贝的文字指针分配存储空间，拷贝当前的文字，然后特判，如果到了当前子句的最后一个，需要将其后继设置为NULL。然后在内重循环结束时需要为下一个子句分配存储空间，为其文字分配存储空间，然后特判，如果到了当前子句的最后一个，需要将其后继设置为NULL。

时间复杂度：O（mn）

（8）void deleteCNF(cnf \*&CNF)

函数参数：cnf指针引用。

函数功能：释放整个邻接表的内存，删除邻接表。

实现思路：需要用到两个文字指针，需要进行两重循环对整个邻接表进行遍历，才能够删除整个邻接表并释放所有空间，需要注意在释放后要将所有指针赋为空。

时间复杂度：O（n）

（9）int savedata(int value[], char Filename[], double time, int result)

函数参数：整型数组，字符型数组，浮点型，整型。

函数功能：根据value数组得到最终的可满足性问题的解，然后将其保存到与原问题同名的res文件中，并且给出求解时间。

实现思路：首先根据形参Filename得到更改后的文件名，然后根据value数组，若为0说明为正文字，否则为负文字，这就是最终需要的答案，然后根据time，计算得到整个DPLL算法运行的时间。

时间复杂度：O（n）

（10）int DPLL(cnf \*&CNF, int value[])

函数参数：cnf指针引用，整型数组。

函数功能：利用DPLL算法求解传入的CNF范式，给出是否有解，并且将最终的解传入到value数组中，这也就实现了SAT模块的第三个功能。

实现思路：首先我们在最开始需要利用单子句规则和分裂策略对CNF范式进行简化。这里需要用到一个三指针p、q、r的策略，p指针用来对整个邻接表进行遍历，主要用来查找其中的单子句，然后q指针和r指针需要根据由p指针得出的单子句的数据来遍历整个邻接表，对邻接表表进行化简。在外层循环的最开始，我们需要调用isOneClause函数来判断当前的p是否为单子句，进而可以找到第一个单子句，如果找不到，说明目前无法化简，直接退出大循环，如果能找到，则根据单子句规则，当前单子句的文字必须为真，我们对value数组此处的位置进行标记，然后利用分裂规则，利用q指针和r指针来遍历整个邻接表，如果单子句的文字出现在了某个子句中，直接调用deleteClause函数删除当前的子句，如果单子句文字对应的负文字出现了某个子句中，则直接调用deletechara函数删除当前子句中的这个文字。在q指针和r指针遍历了整个邻接表后，我们需要进行相应的特判，如果整个邻接表被完全删除，那么就直接可满足，退出函数，返回1，如果调用iswithEmptyClause函数后发现当前邻接表里有空子句，则调用deleteCNF函数，释放整个邻接表的空间，返回0，因为这个时候就说明该CNF范式一定是不可满足的。这样操作之后，在将p指针指向表头位置，继续遍历即可。在化简完毕之后，我们需要选取合适的变元进行回溯。这里的选择方式是选取出现次数最多的文字，方法是开一个二倍于文字数量的数组，统计所有文字出现的次数，然后找到最大值和最大值出现的位置。接下来我们需要调用copy clause函数，得到两个新的邻接表，进而进行回溯。将选取的变元加入到其中的一个邻接表中，重新调用DPLL函数，有界就返回1，销毁这个邻接表，否则将该文字的负文字加入到另一个邻接表中，调用DPLL算法，直接销毁该邻接表并返回这次调用的值即可。

（11）int betterDPLL(cnf \*&CNF, int value[])

函数参数：cnf指针引用，整型数组。

函数功能：优化后的DPLL算法，更高效的处理部分sat问题，同时也完成了sat模块的第四个功能。

实现思路：考虑对基本DPLL的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现。在存储结构方面最初考虑在每个子句方面加入子句长度这一数据，然后对整个邻接表进行预处理，即依据子句长度对整个邻接表种的子句进行一次快速排序，这样可以更快的找到CNF范式中的单子句，然而其实性能并没有太大的优化，在面对有的数据的时候性能反而下降。另外就是在简化CNF范式的时候加入了重言子句策略，但是对性能也没有太大的提升，最后考虑在分支变元选取策略上进行优化。最开始的选取策略是选择出现次数最多的变元，但是其实回溯的时候正负变元都会用到，所以可以考虑先选取出现次数最多的正变元，如果没有正变元，再选取出现次数最多的负变元即可。

（12）void addClause(cnf \*CNF, cnf \*&root)

函数参数：cnf指针，cnf指针引用。

函数功能：将CNF子句添加到当前的root范式中。

实现思路：只需要让CNF子句的后继为root，然后让root指针指向CNF即可。

时间复杂度：O（1）

（13）void dfs()

函数参数：无。

函数功能：随机生成数独游戏的终盘的第一行，也就是生成了1-9的随机一个的全排列。

实现思路：首先定义一个整型数组a并且初始化，使其按顺序存储了1-9，接下来将第一行定义成一个栈，在循环中，首先依次得到1-9之间的一个随机数，然后将该位置上的数组a的值入栈，把该值和a数组中的最后一个数交换，然后将数组a的长度可看作减一，因为当前最后的一个数已经被使用过了。

时间复杂度：O（n）

（14）int solveSudoku(int sudoku[][9])

函数参数：二维整型数组。

函数功能：将目前的数独棋盘转换成CNF范式，并且将其整合到DPLL算法中进行求解，给出最终的终盘。同时也完成了Sudoku模块中的求解功能。

实现思路：其实主要的难点就是将数独棋盘转换成CNF范式，这里参考了课程设计任务书中给出的方法，利用合适的编码以及多重循环将行约束、列约束、3\*3约束以及已有的数字整合成相应的CNF范式，其中坐标转换公式为ijn → (i-1)\*81+(j-1)\*9+n。然后整合到DPLL算法中进行求解即可，最后还需要一个解码的过程，其实将整个编码的过程倒过来就可以了。过程中需要调用better DPLL函数和AddClause函数

（15）void dighole(int sudoku[][9], int n)

函数参数：二维整型数组，整形变量。

函数功能：利用挖洞法，在已经生成好的数独终盘上挖出n个洞，得到一个随机的数独棋盘，并且保证生成的这个数独棋盘具有唯一解。

主要思路：首先，我们需要利用随机数生成器在棋盘上随意得到一个坐标，然后挖掉这里的洞，因为仅仅挖掉了一个数，所以我们完全可以保证此时的数独是具有唯一解的，然后进入循环，每次循环的时候随机生成一个随机坐标，然后调用check函数检查如果挖掉当前的洞，是否还具有唯一解，如果具有的话，挖掉即可，另外需要使用一个count来记录被删掉的有效的数据。

（16）int check(int sudoku[][9], int x, int y)

函数参数：二维整型数组，整型变量，整形变量。

函数功能：利用DPLL算法检查传入的坐标是否可以删除，其实就是检查删除了之后是否还具有唯一解。

实现思路：其实本身的思路是非常简单的，要想验证是否具有唯一解，只需要将这个位置的数替换成其他的数，然后用DPLL算法验证是否具有解即可，因为如果有唯一解的话，将这个位置的数换成别的数，是必然得不出正解的，如果没有唯一解的话，说明在这个位置更换成别的数可以得到正解，也就是说这里的这个数是不能被更换的，返回0即可。另外就是我们要对这个位置的数进行编辑，所以我们需要再拷贝一个新的数组进行编辑。最后就是如果我们仅仅利用暴力枚举而不进行优化的话，这样挖洞可能挖20个效率都会非常低，因此我们需要检查一些明显不合理的数据，比如和当前所在的行产生了冲突，或者和当前所在的列产生了冲突，或者和当前所在的小方形产生了冲突，都可以直接continue。

（17）void createsudoku(int sudoku[][9])

函数参数：二维整型数组。

函数功能：生成一个数独游戏棋盘，同时完成了sat问题的第一个模块的功能。

实现思路：其实这里的关键部分在于生成一个数独的终盘，我们考虑这样生成一个终盘，即首先调用dfs函数，生成第一行的一个全排列，然后定义一个shift数组，利用二重循环对第一行进行轮换拷贝，即可得到一个数独终盘。然后系统会提示用户输入想要挖洞的数量，然后调用函数dighole进行挖洞。

### 4.1.5 main函数

main函数的设计其实非常简单，主要就是利用switch语句进行分类、分模块，另外就是需要给出一定的提示和错误输入的反馈。

## 4.2系统测试

### 4.2.1 sat模块运行测试

该模块要求功能如下：

1. 整个模块可以流畅运行。

2. 能够正确读取CNF文件，并将其正确存储到相应的数据结构中。

3. 能够逐行打印读取的CNF范式。

4. 能够使用DPLL算法正确求解sat问题，并将其保存到文件中。

5. 对DPLL算法应当有一定的优化。

6. 能够正常退出模块。

测试大纲：

1. 由于本系统要求能够正确求解sat问题，所以选取两个小样例分别测试本系统对可满足算例与不可满足算例的求解，这里直接选取优化后的DPLL算法即可。

2. 由于本系统对求解性能有一定要求，因此选取两个较大样例来测试优化前和优化后的DPLL算法的性能，并且计算相应的优化率。

本组测试如表4-1所示。

表4-1 sat模块测试步骤

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 输入 | 预计输出 | 实际输出 |
| 1 | 1（进入sat模块） | sat模块内容 | 如图4-2 |
| 2 | 1（读取简单满足CNF文件） | “请输入文件路径” | 如图4-3 |
| 3 | C:\Users\yx7\Desktop\sat-20.cnf | “读取成功” | 如图4-4 |
| 4 | 2（逐行打印CNF范式） | CNF范式 | 如图4-5 |
| 5 | 3（使用优化后的算法求解） | 输出结果并成功保存 | 如图4-6 |
| 6 | 1（读取简单不满足CNF文件） | “请输入文件路径” | 如图4-9 |
| 7 | C:\Users\yx7\Desktop\unsat-5cnf-30.cnf | “读取成功” | 如图4-10 |
| 8 | 2（逐行打印CNF范式） | CNF范式 | 如图4-11 |
| 9 | 3（使用优化后的算法求解） | 输出结果并成功保存 | 如图4-12 |
| 10 | 1（读取复杂满足CNF文件） | “请输入文件路径” | 如图4-14 |
| 11 | C:\Users\yx7\Desktop\  ais10.cnf | “读取成功” | 如图4-15 |
| 12 | 2（逐行打印CNF范式） | CNF范式 | 如图4-16 |
| 13 | 3（使用优化后DPLL算法求解） | 输出结果并成功保存 | 如图4-17 |
| 14 | 1、C:\Users\yx7\Desktop\  ais10.cnf、4（优化前求解） | 输出结果并成功保存 | 如图4-18 |
| 15 | 1（读取复杂满足样例） | “请输入文件路径” | 如图4-19 |
| 16 | C:\Users\yx7\Desktop\6.cnf | “读取成功” | 如图4-20 |
| 17 | 2（逐行打印CNF范式） | CNF范式 | 如图4-21 |
| 18 | 3（使用优化后DPLL算法求解） | 输出结果并成功保存 | 如图4-22 |
| 19 | 1、C:\Users\yx7\Desktop\  ais10.cnf、4（优化前求解） | 输出结果并成功保存 | 如图4-23 |
| 19 | 0（退出sat模块） | 主系统模块 | 如图4-24 |

下面是相应的运行截图。

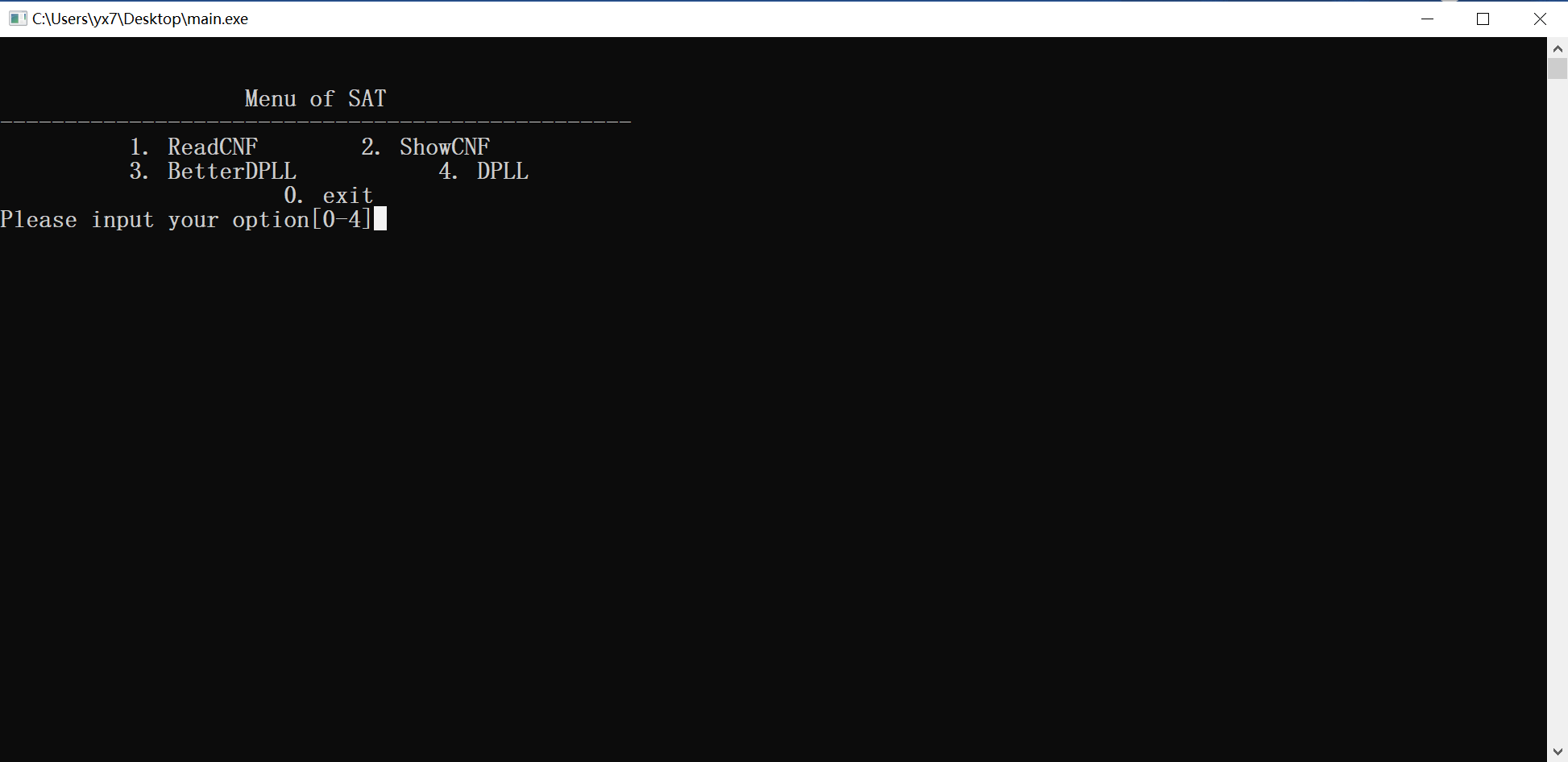


图4-2 步骤一运行结果

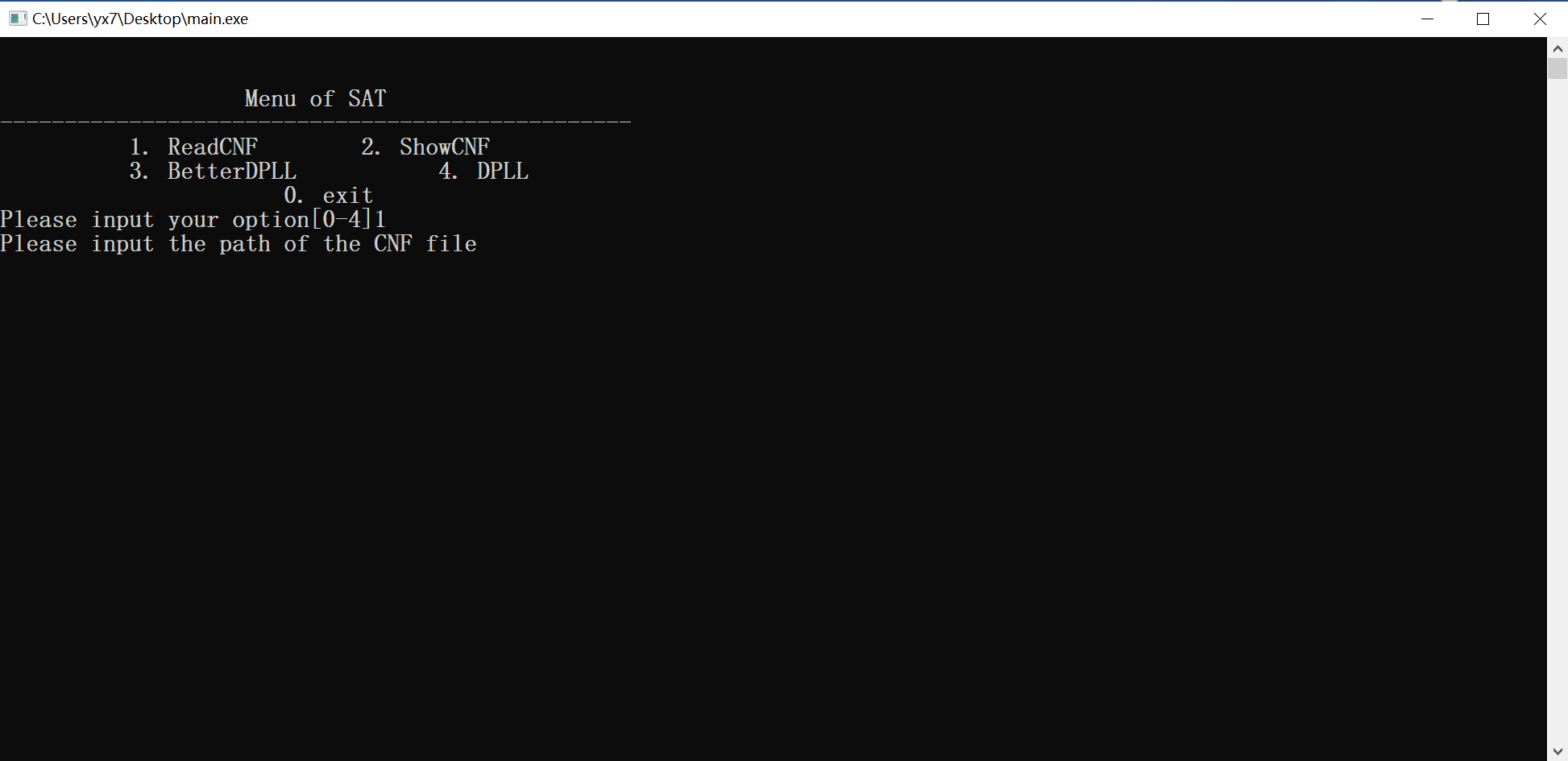


图4-3 步骤二运行结果

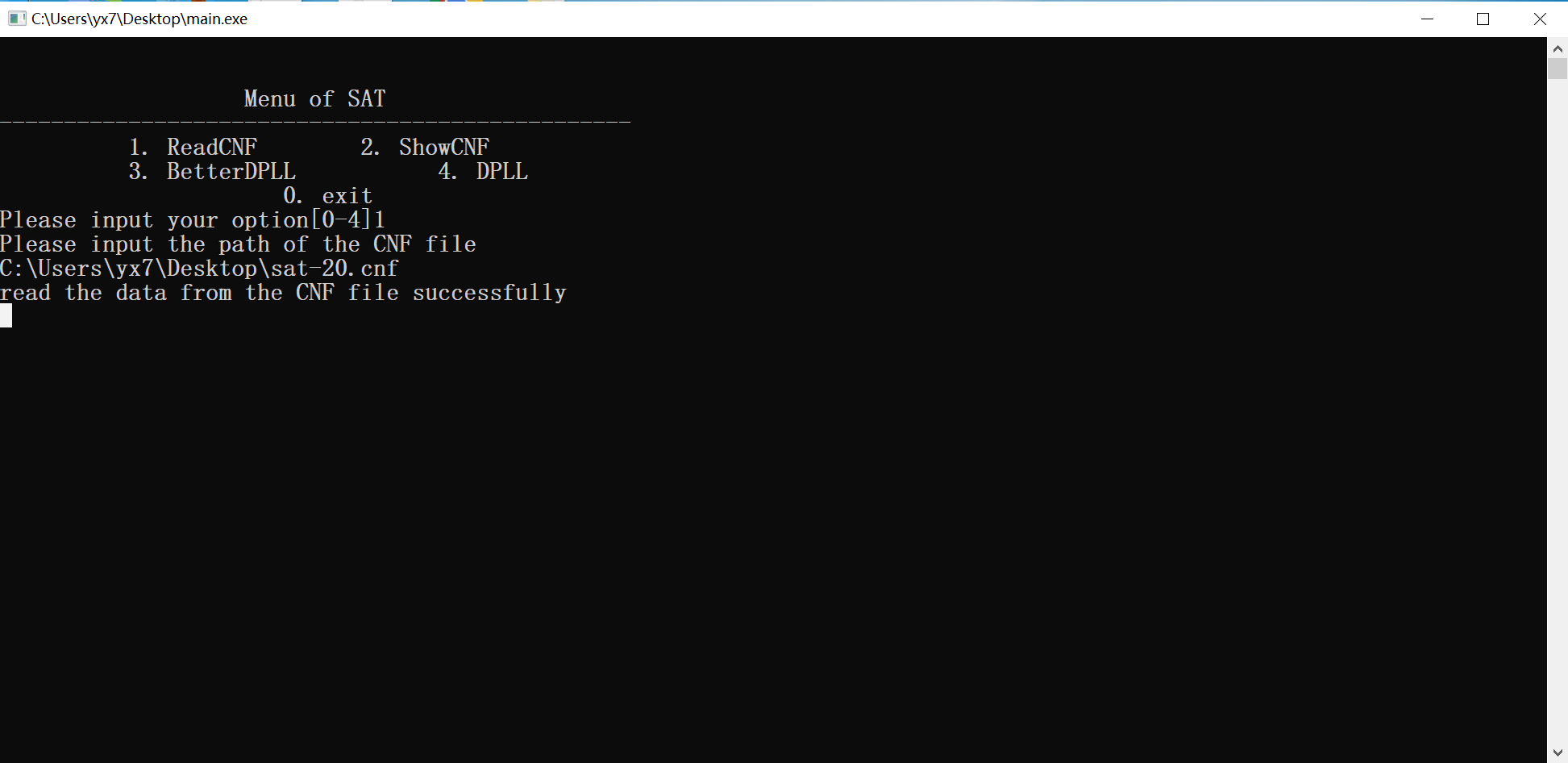


图4-4 步骤三运行结果

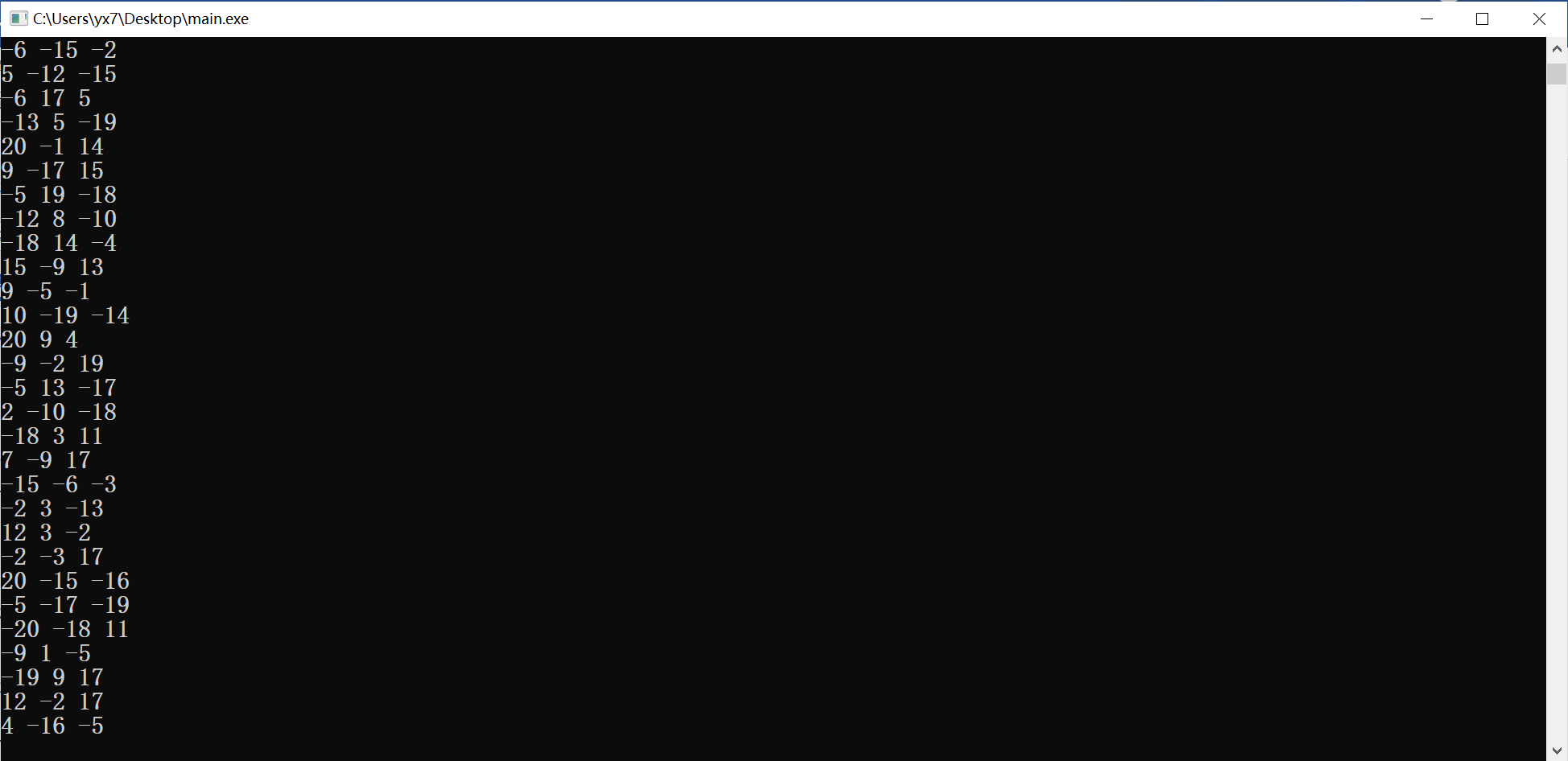


图4-5 步骤四运行结果

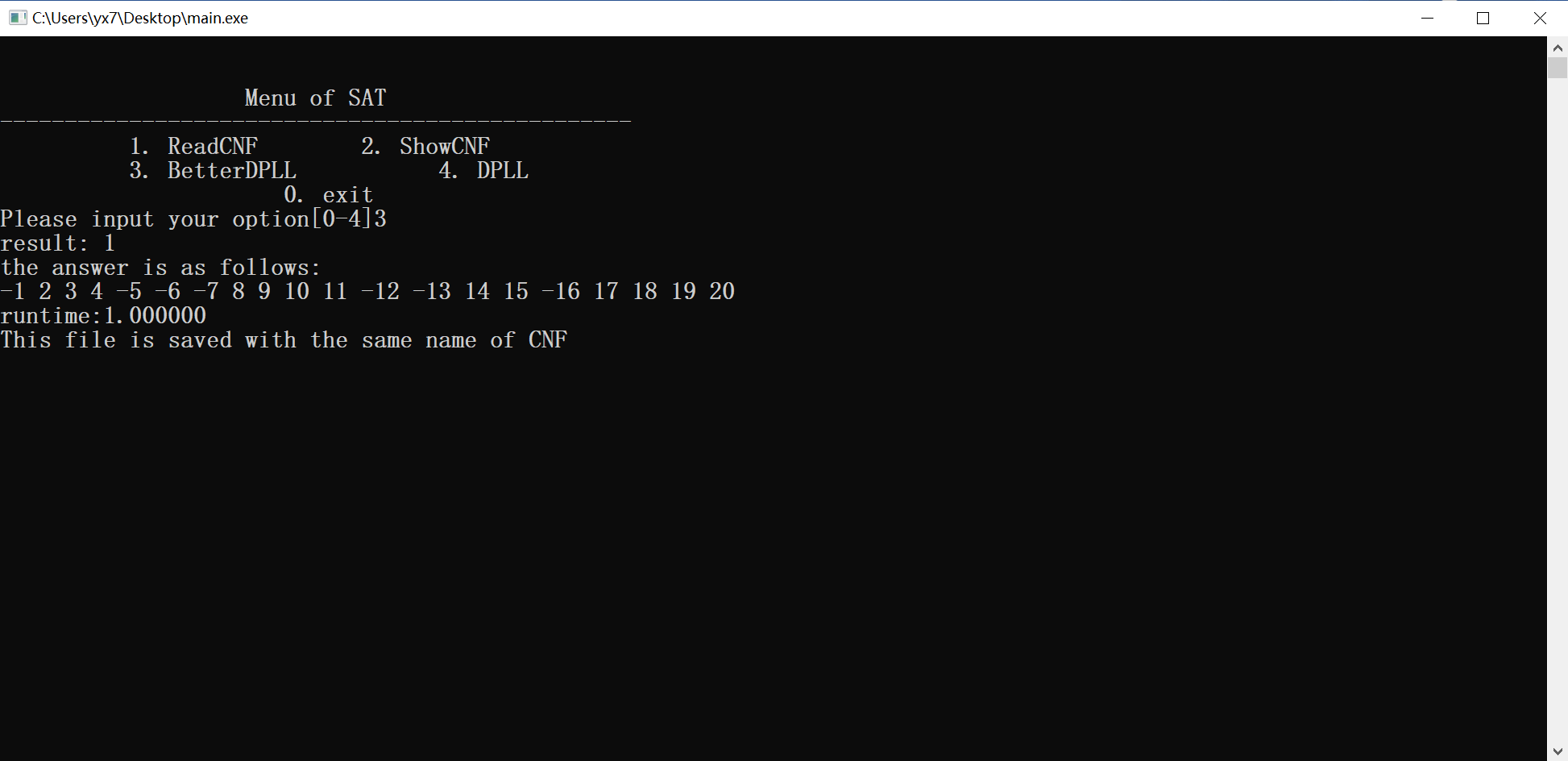


图4-6 步骤五运行结果

下面是res文件中的内容，见图4-7。

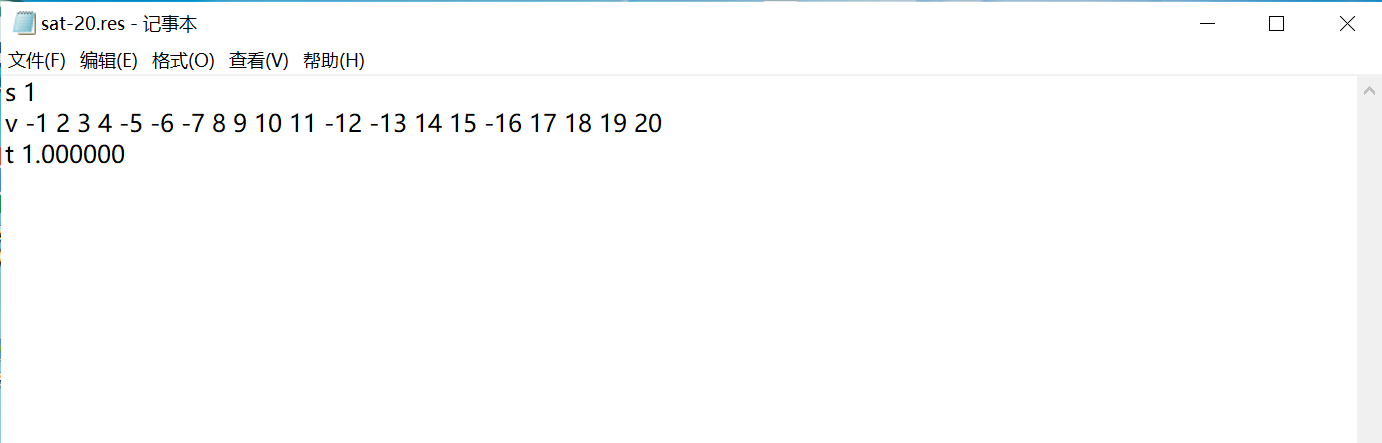


图4-7 res文件内容

下面是用verify.exe文件验证的结果，见图4-8。



图4-8 verify.exe文件验证结果

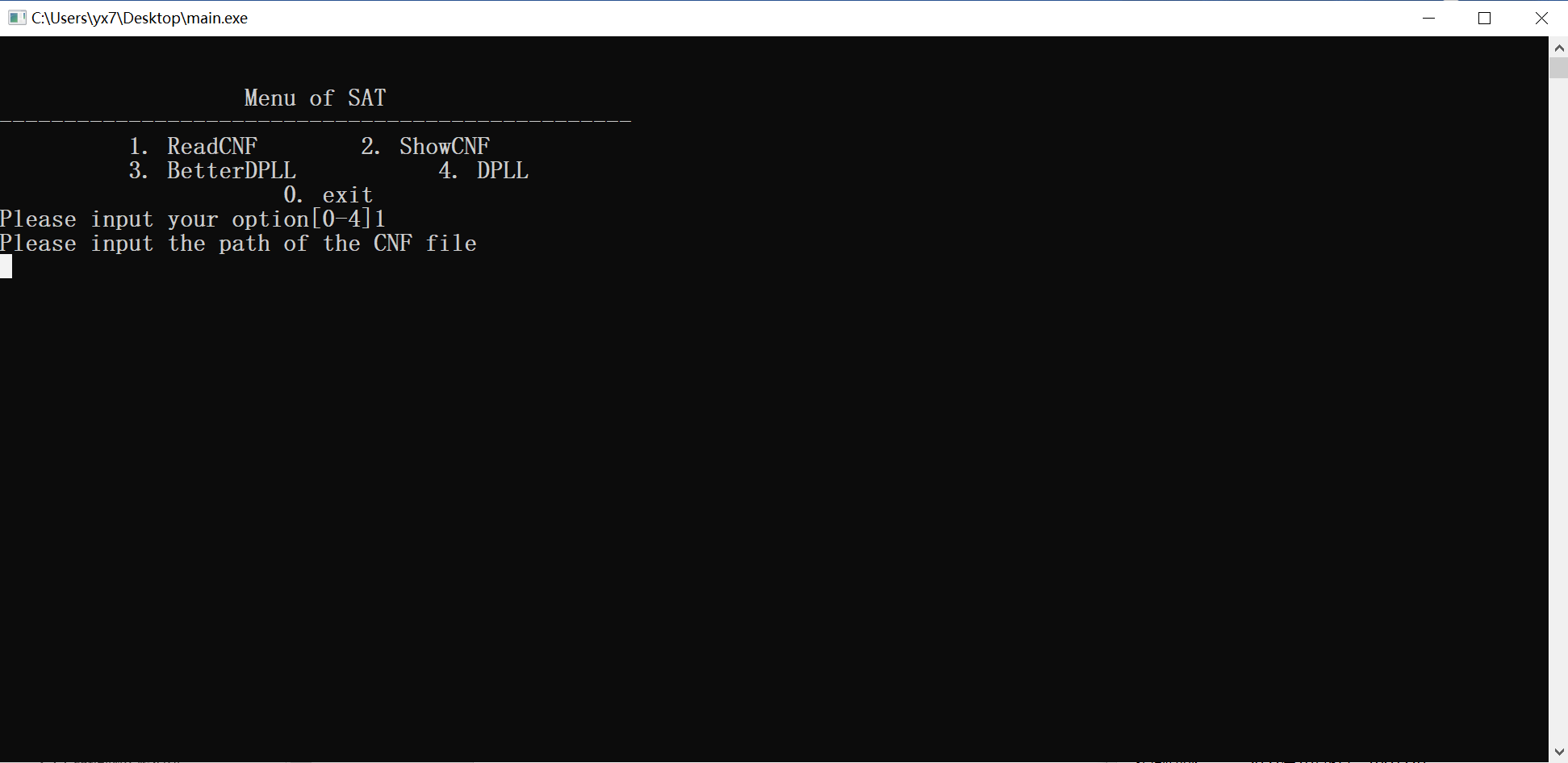


图4-9 步骤六运行结果

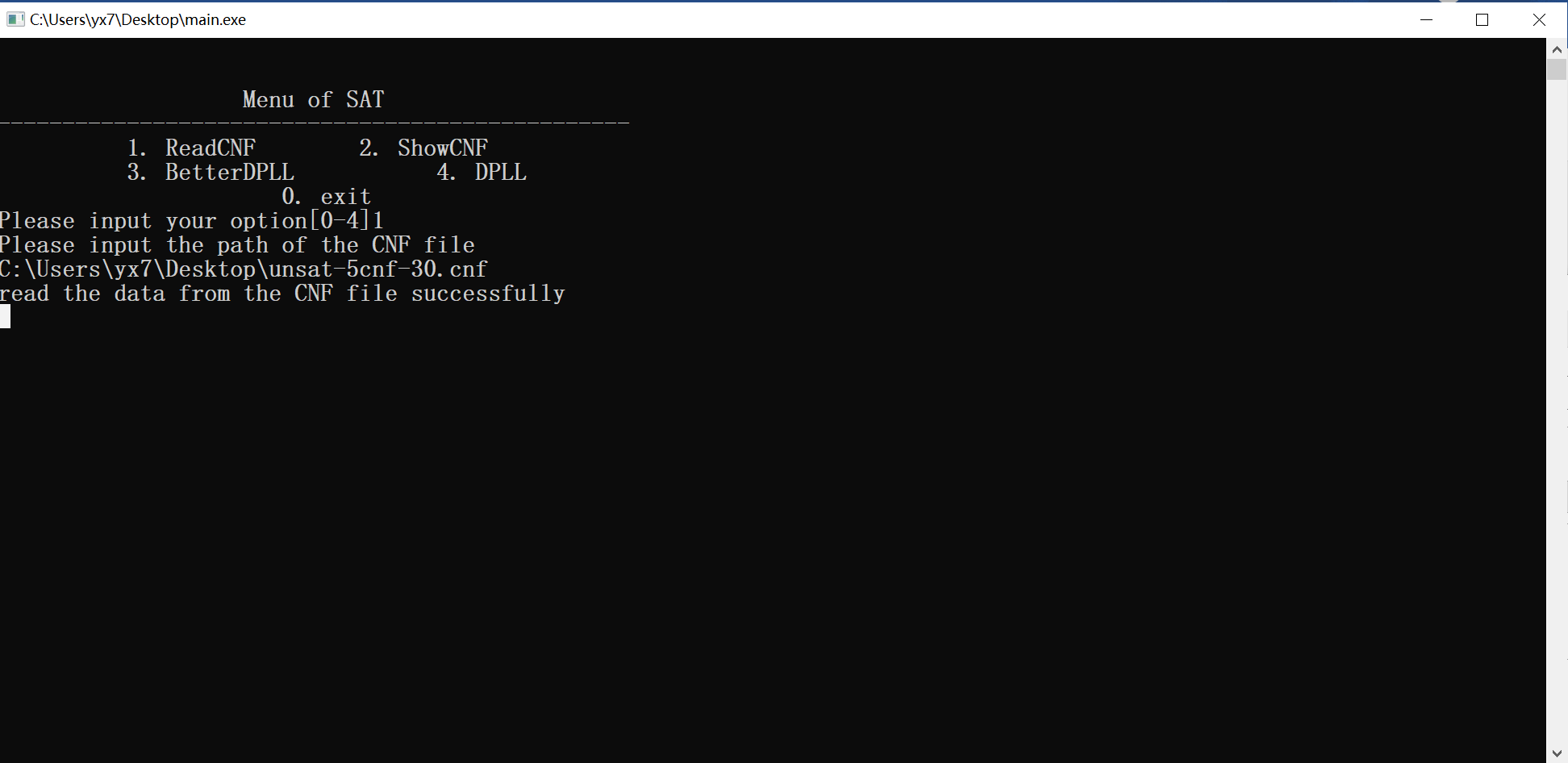


图4-10 步骤七运行结果

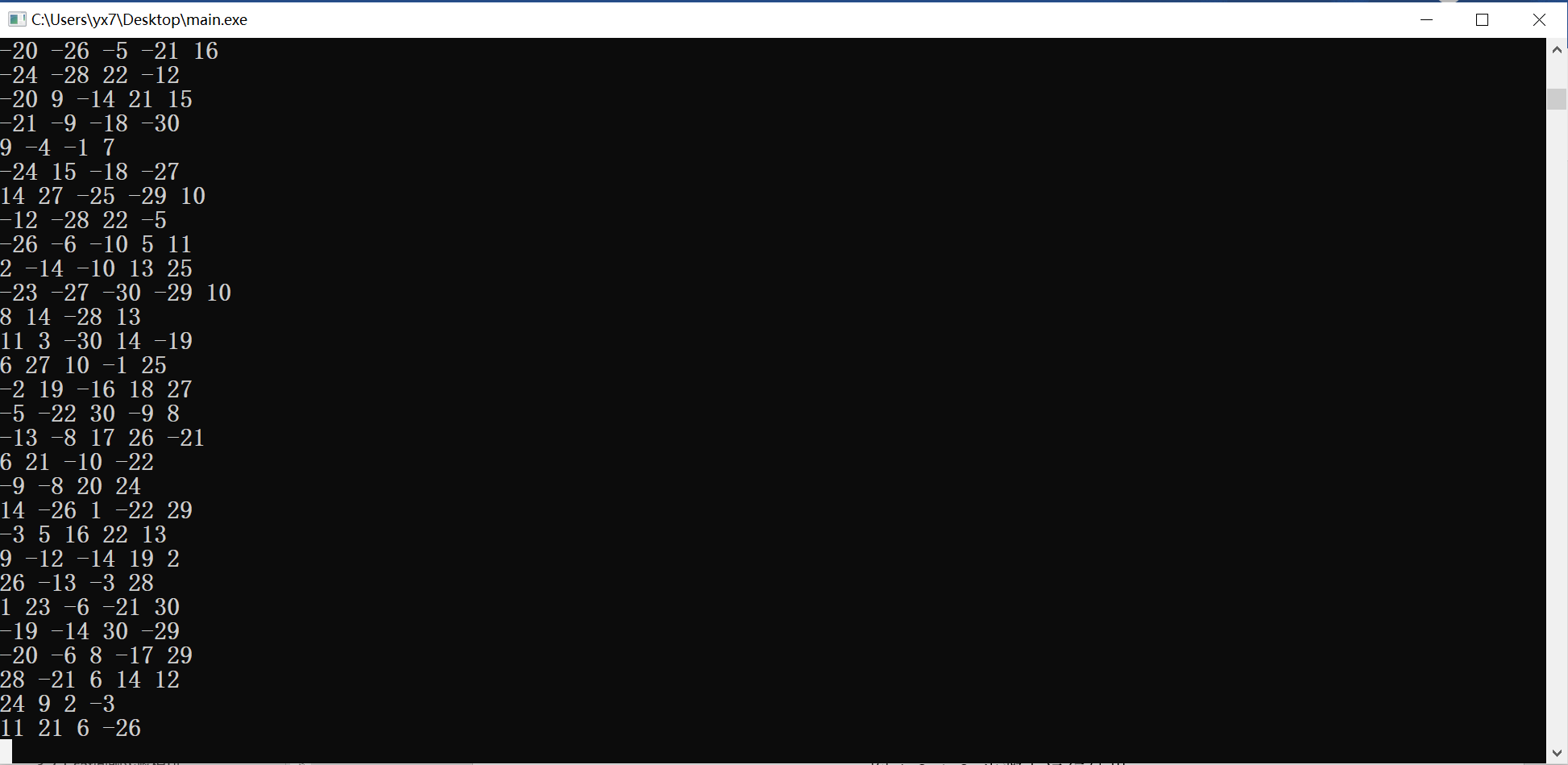


图4-11 步骤八运行结果

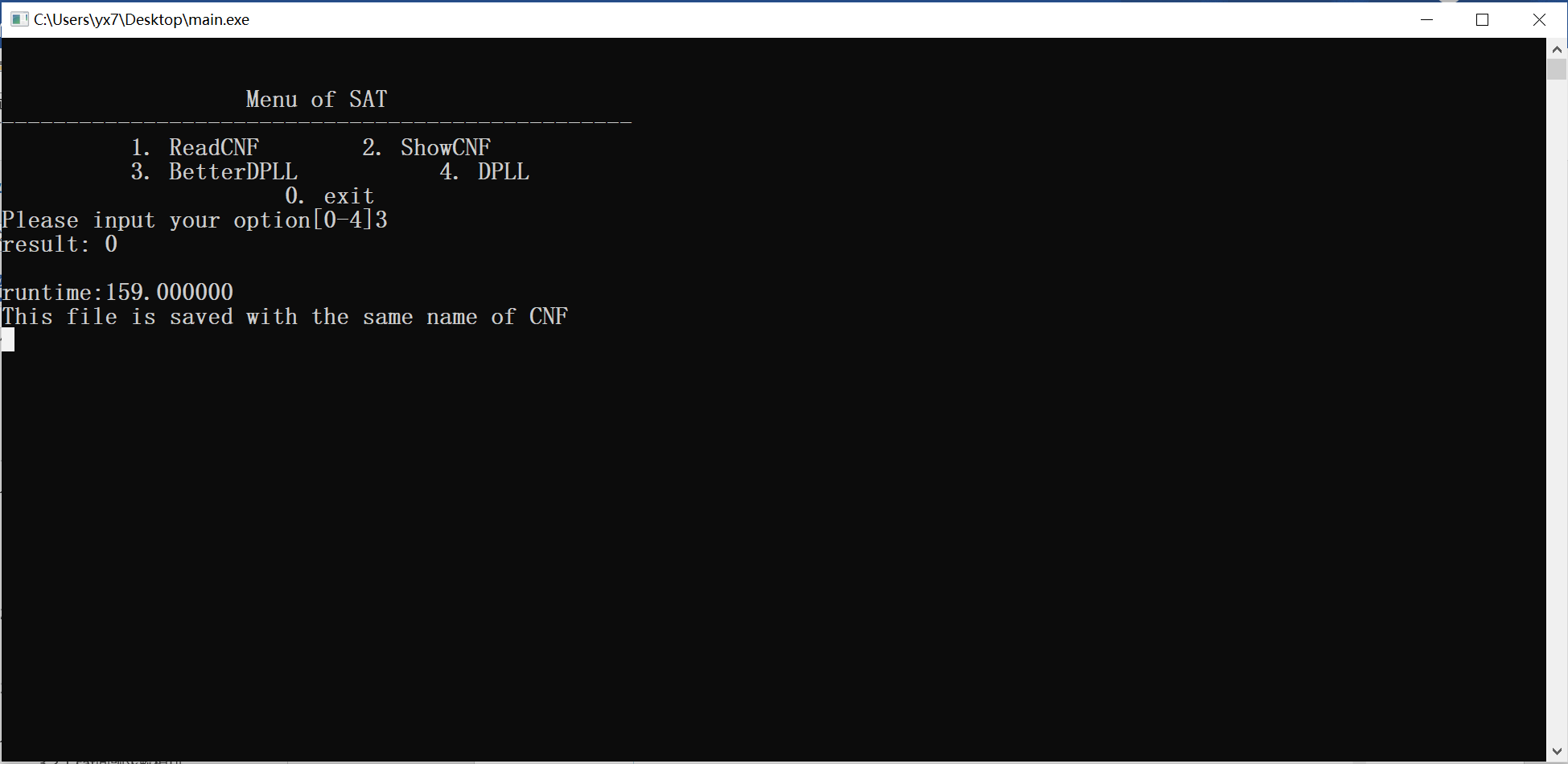


图4-12 步骤九运行结果

下面是res文件中的内容，见图4-13。

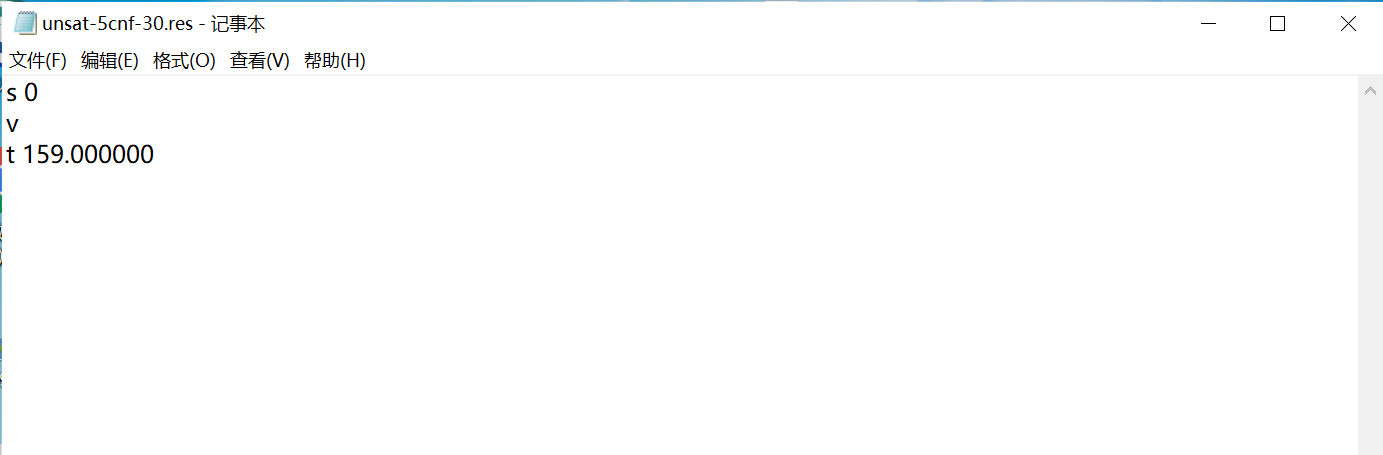


图4-13 res文件内容

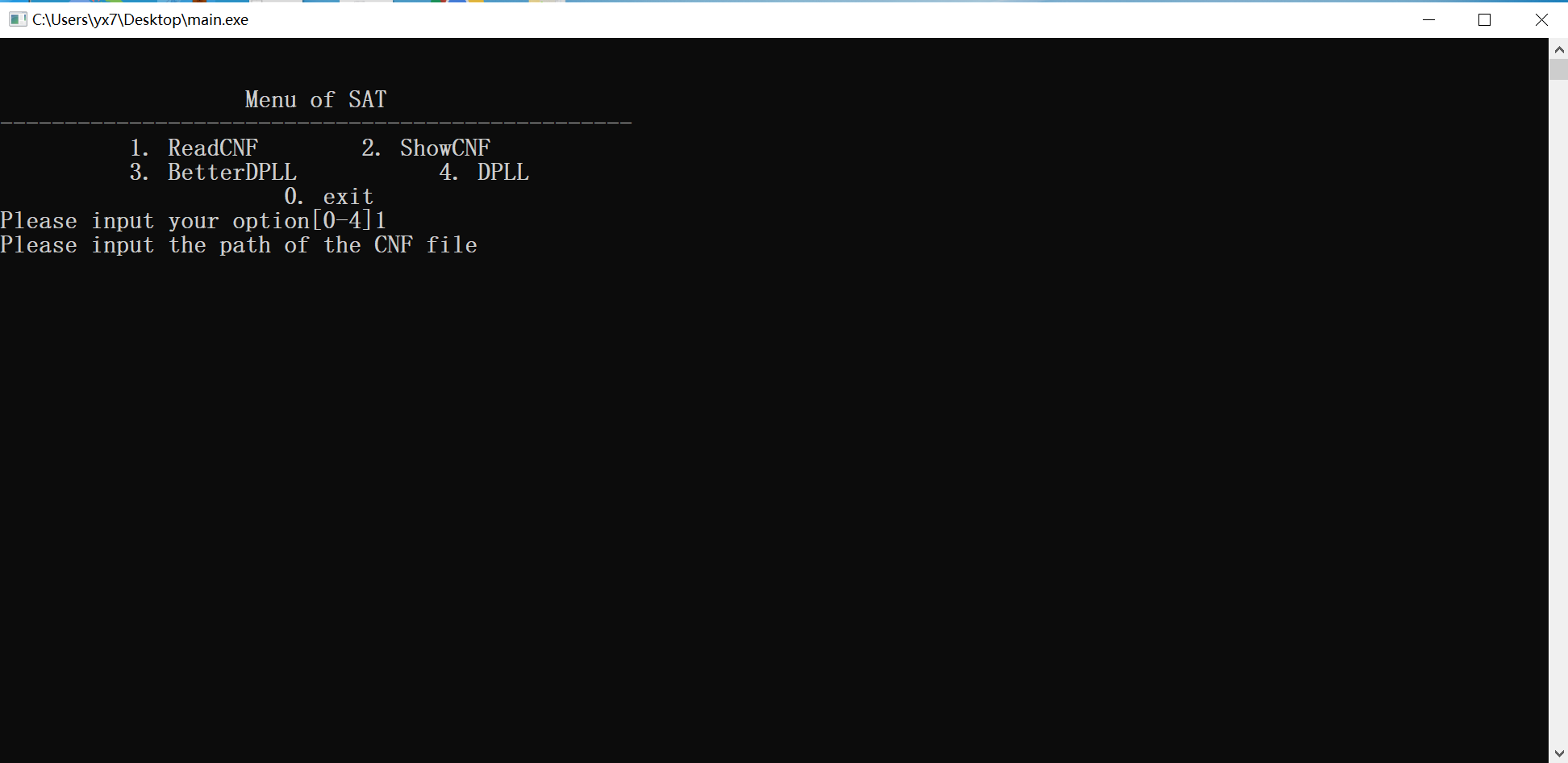


图4-14 步骤十运行结果

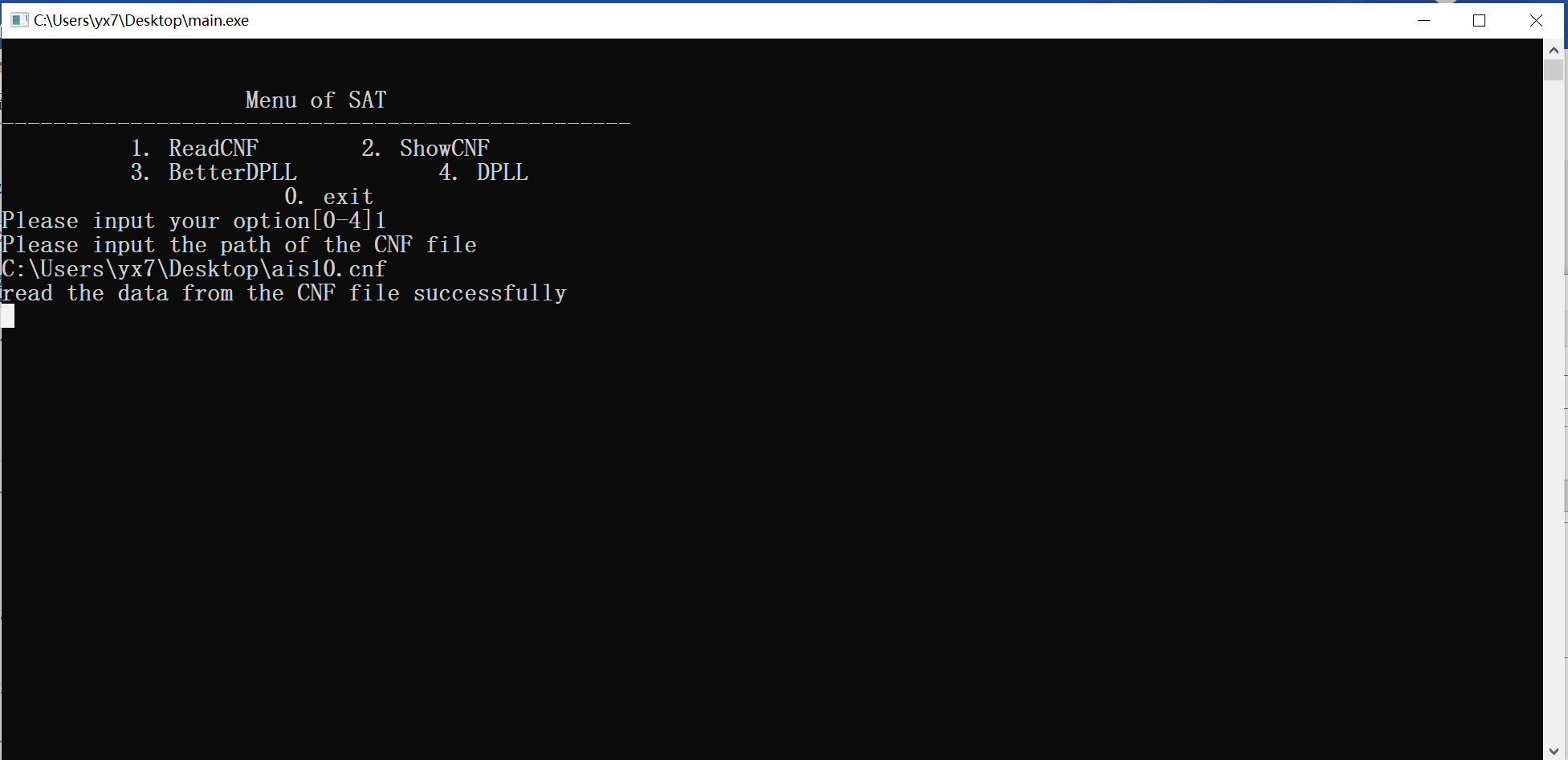


图4-15 步骤十一运行结果

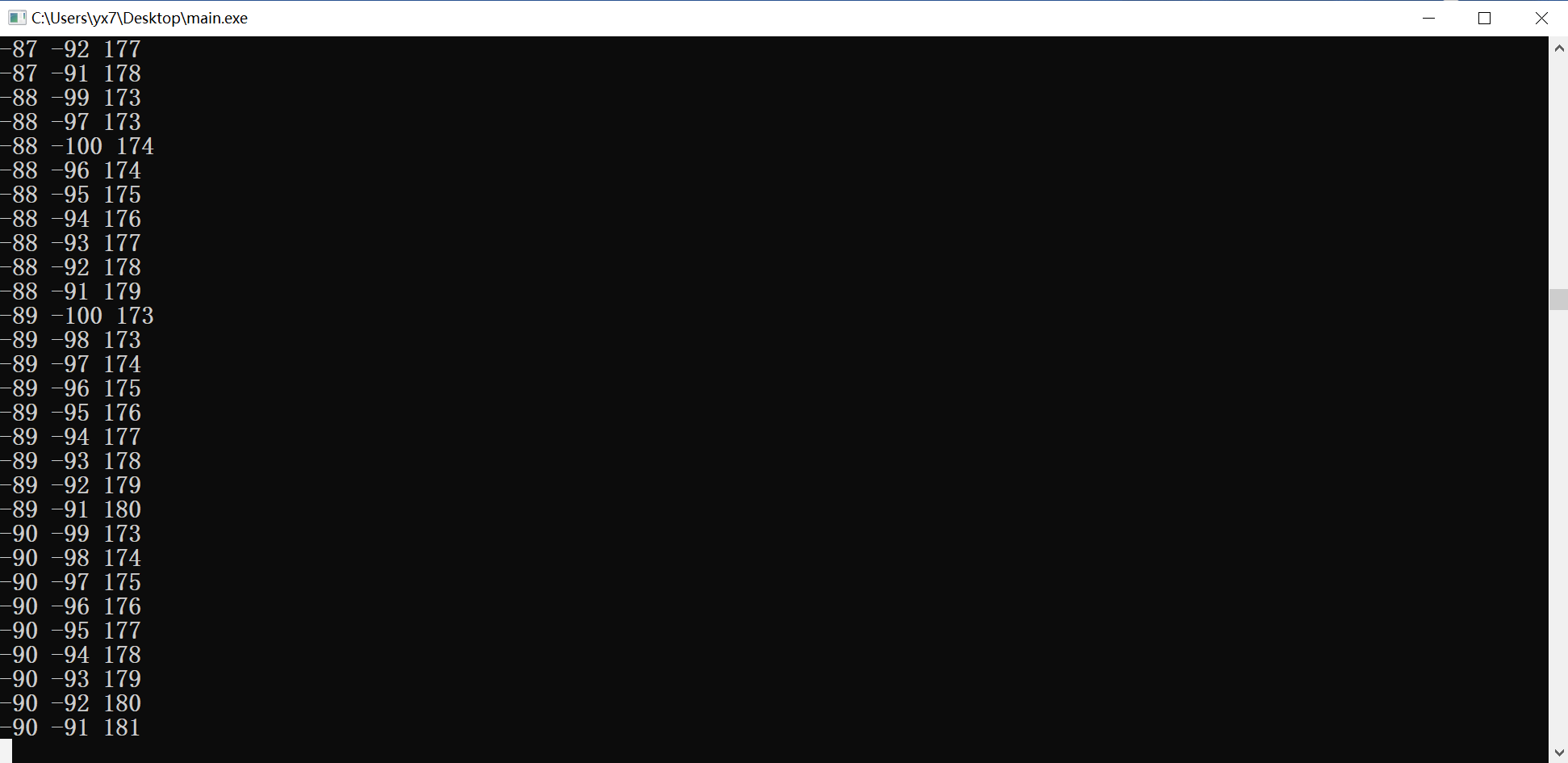


图4-16 步骤十二运行结果

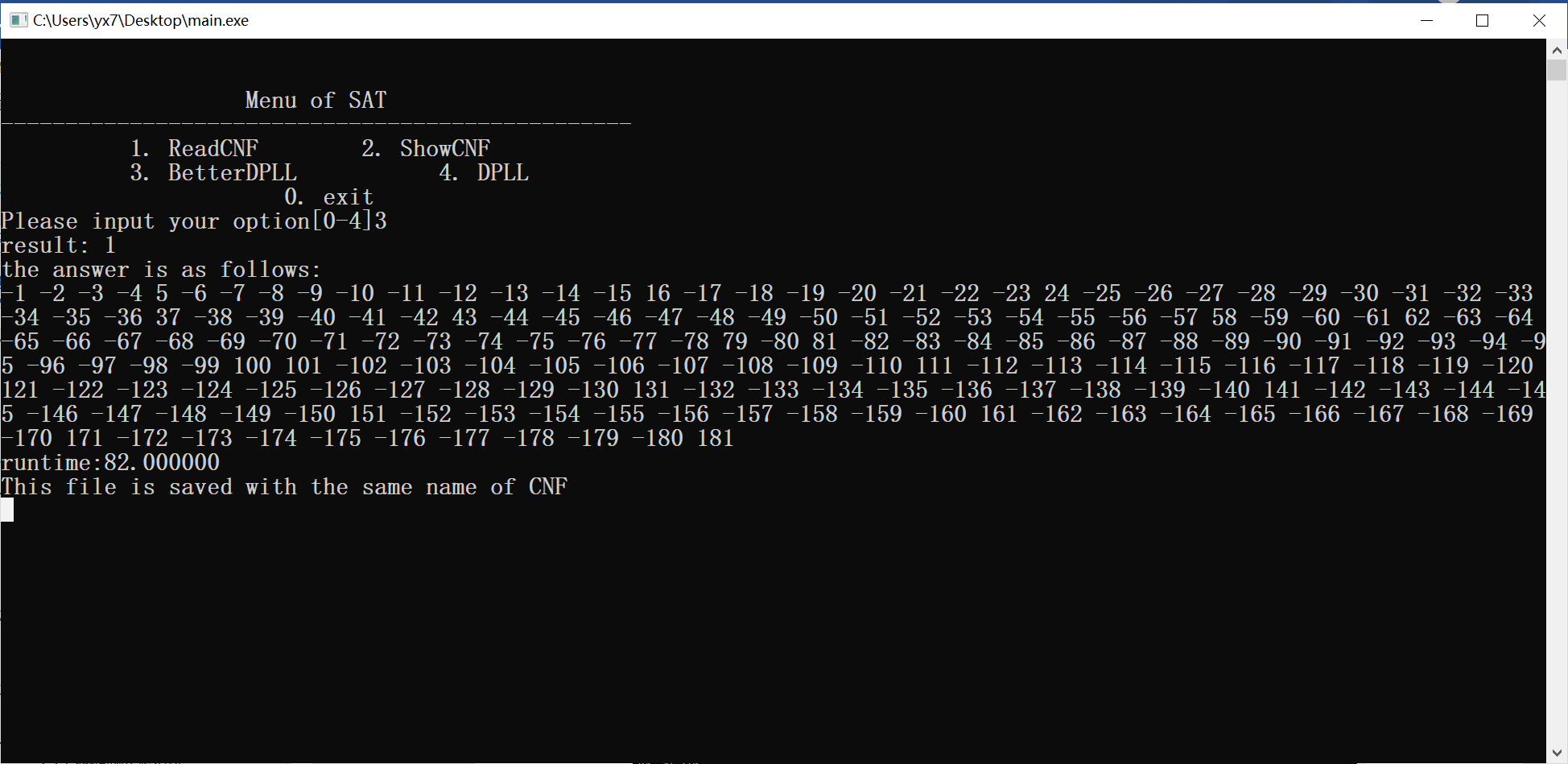


图4-17 步骤十三运行结果

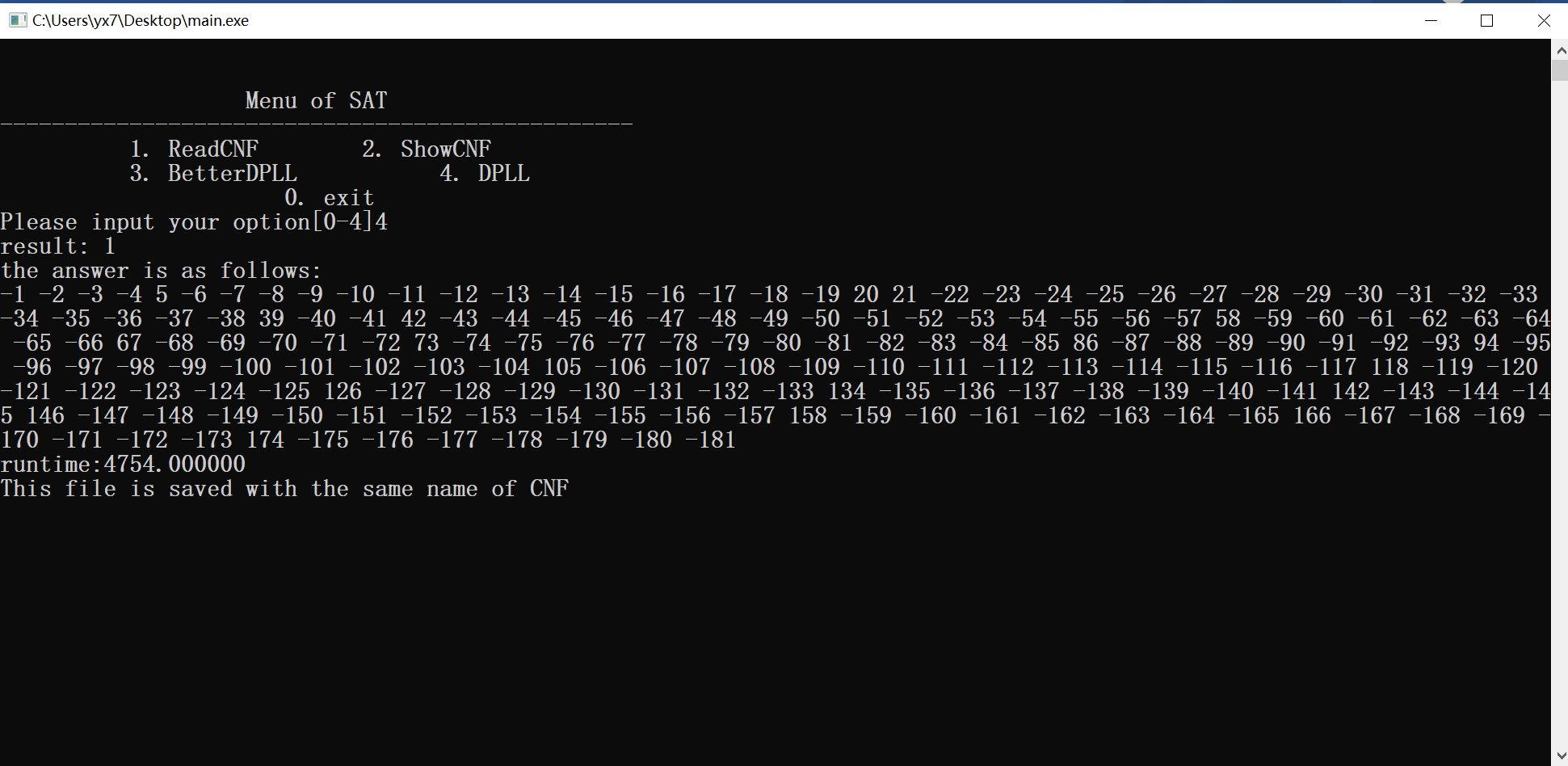


图4-18 步骤十四运行结果

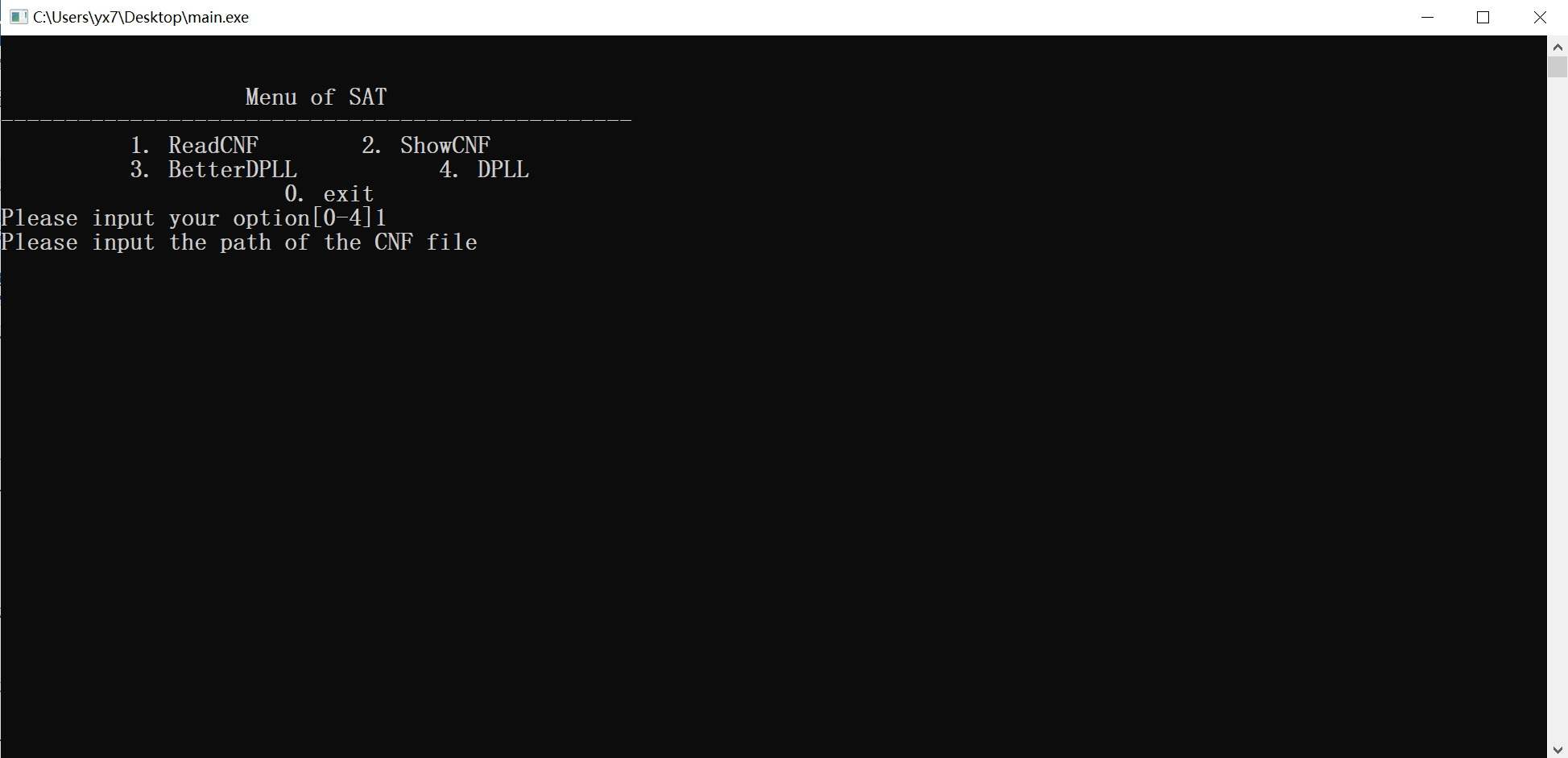


图4-19 步骤十五运行结果

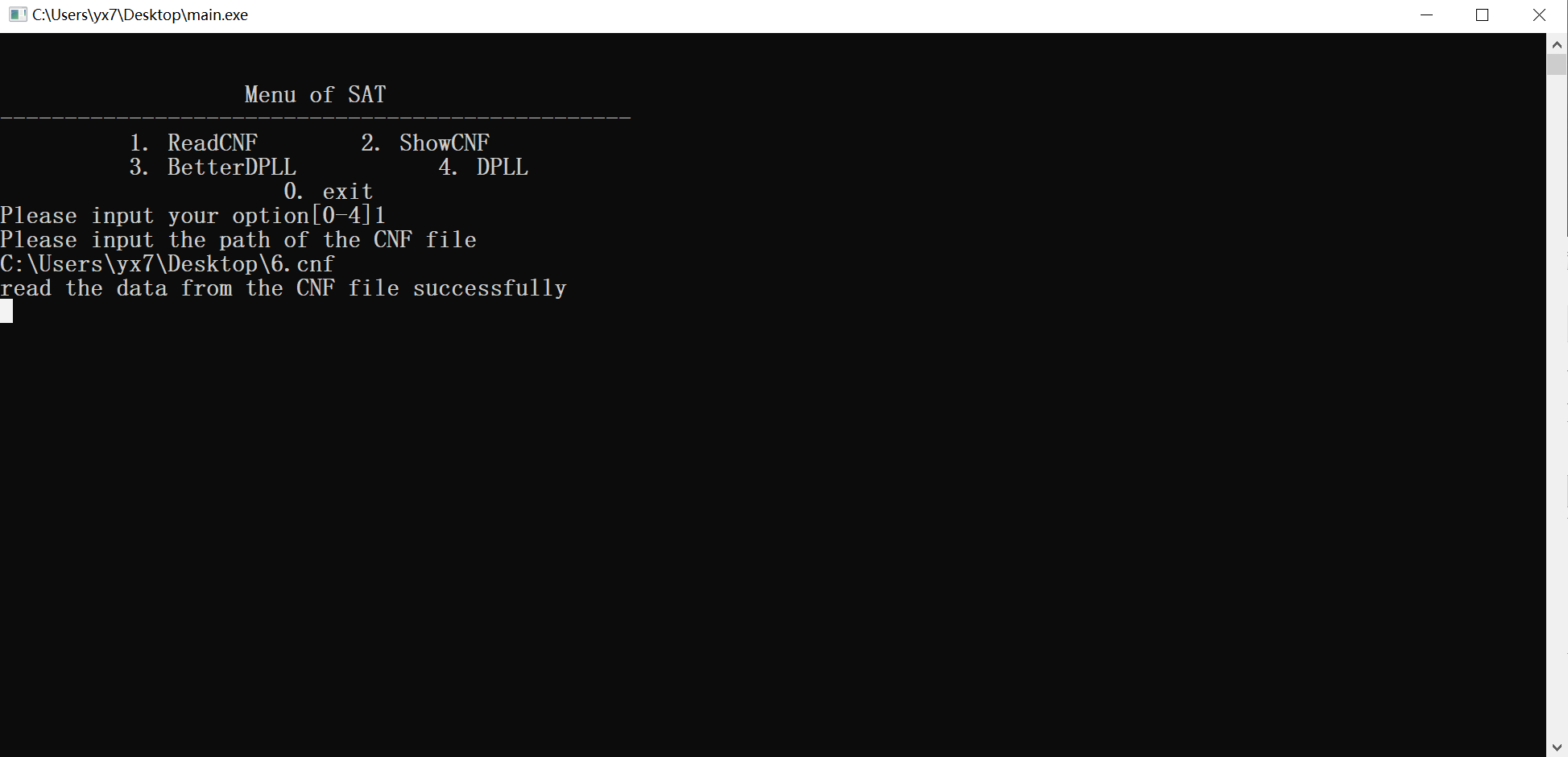


图4-20 步骤十六运行结果

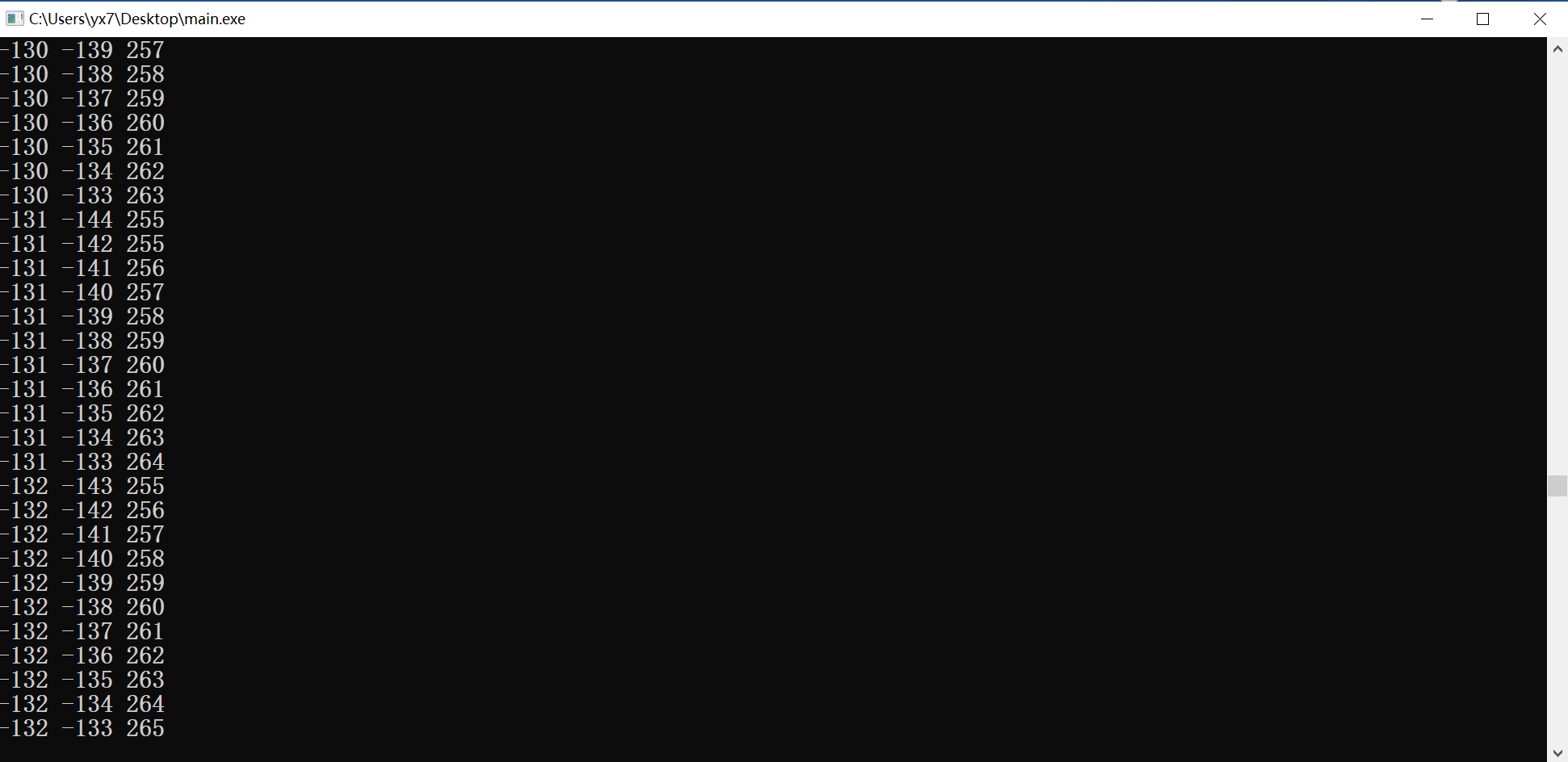


图4-21 步骤十七运行结果

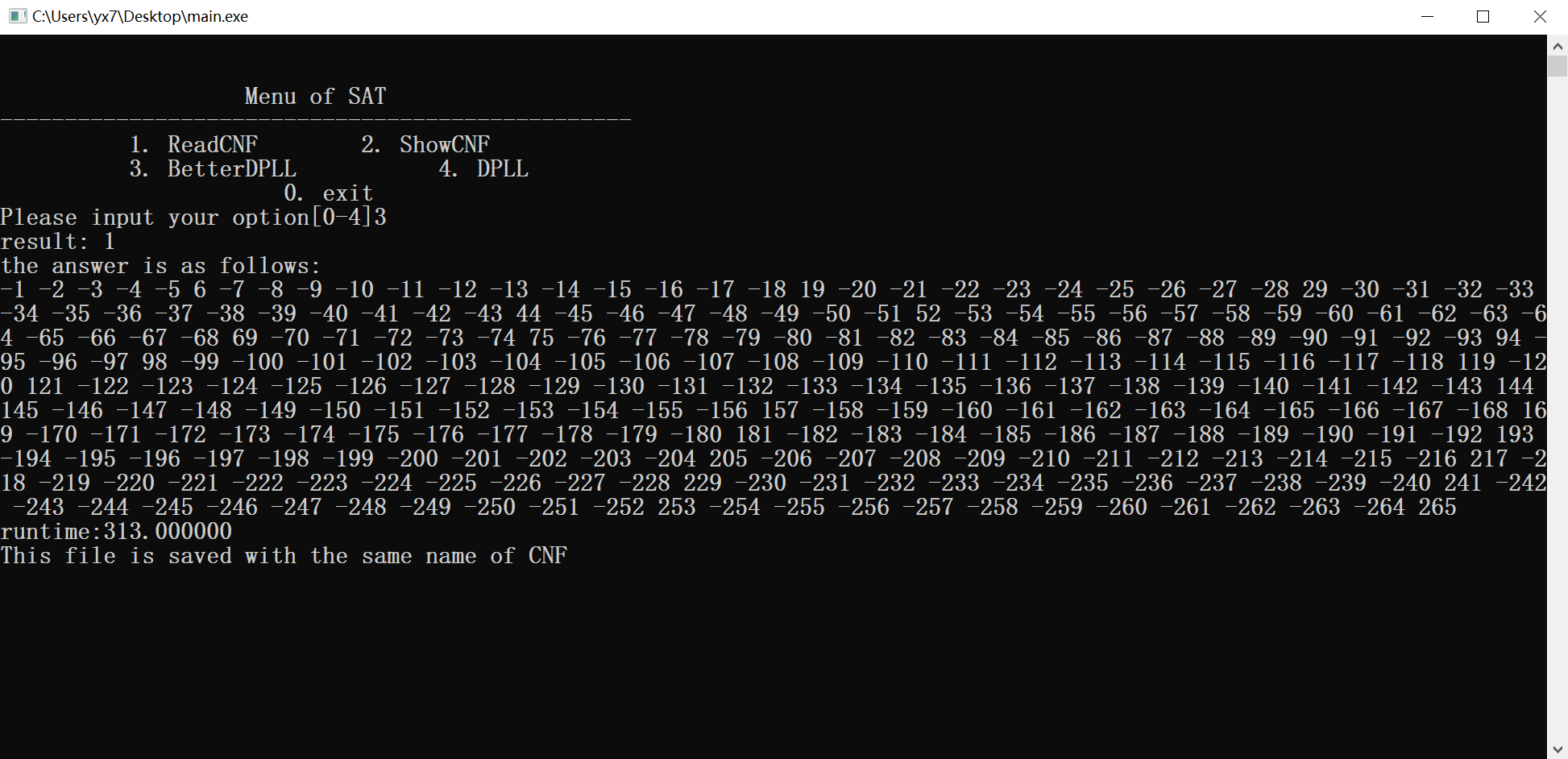


图4-22 步骤十八运行结果

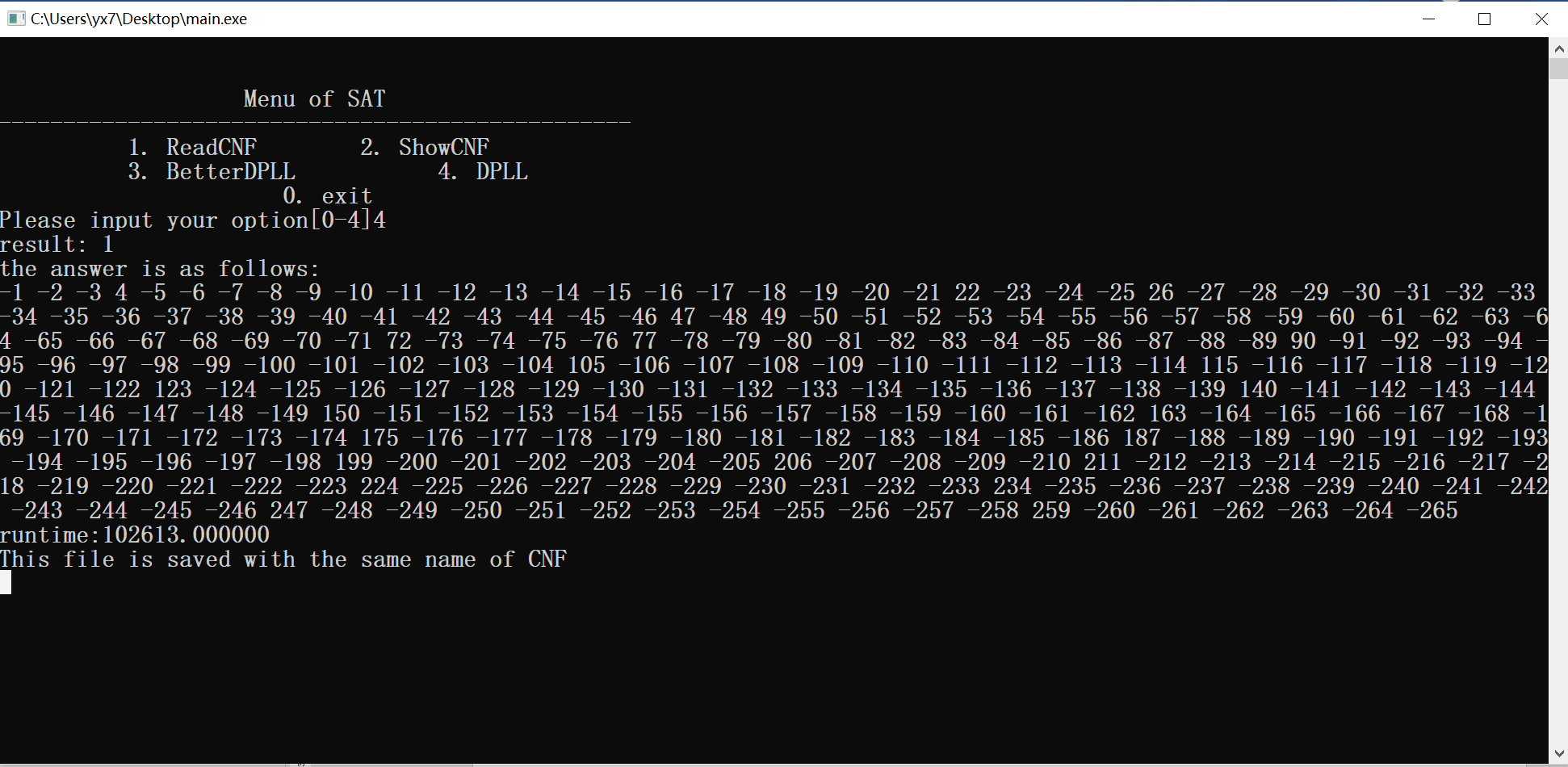


图4-23 步骤十九运行结果

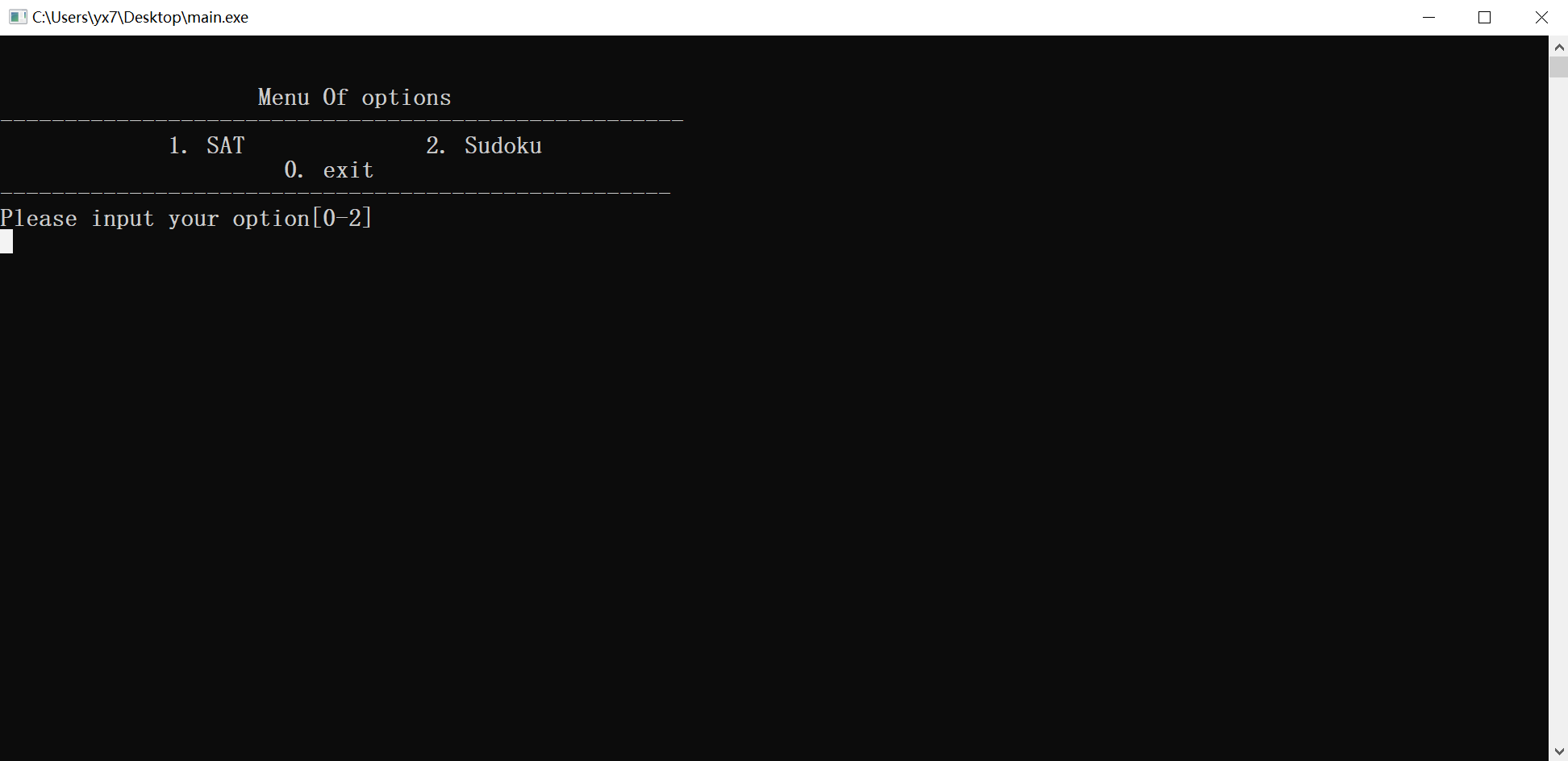


图4-24 步骤二十运行结果

就实际运行结果来看，首先根据测试来看，整个系统运行结果较为流畅，然后经过多次运行功能一，证实将CNF文件读入到程序中并存储到相应的数据结构中这一过程是可以的，另外打印整个CNF范式也是没有问题的。另外利用优化后的DPLL算法可以正确求解sat问题，并且能够将最终的运行时间和答案全部保存在res文件中，通过verify.exe文件对res结果进行验证，是没有问题的。

最后就是关于算法优化这一部分，可以看出来最终优化率是相当高的，对于6.cnf文件，优化前，运行时间接近十万毫秒，优化后只需要三百毫秒左右就可以计算完成，并且计算结果是完全正确的，对于ais10.cnf文件，优化前运行时间接近五千毫秒，优化后运行时间仅需要八十余毫秒。

下面给出优化前后的部分测试数据的对比表。

表4-2 优化算法对比

|  |  |  |  |
| --- | --- | --- | --- |
| 测试集 | 优化前时间 | 优化后时间 | 优化率 |
| ais10.cnf | 4928ms | 92ms | 98.13% |
| 6.cnf | 102613ms | 313ms | 99.69% |
| Sud00009.cnf | 6667ms | 31ms | 99.53% |
| 3.cnf | 跑不出 | 93ms | 100% |
| 2.cnf | 22300ms | 2203ms | 90.01% |

### 4.2.2 数独游戏模块运行测试

该模块有如下要求：

1. 整个系统能够流畅正确的运行。

2. 能够生成一个具有唯一解的数独棋盘。

3. 具有简单地交互性，游戏可玩。

4. 能够将生成的数独保存到文件中。

测试大纲：

1. 测试生成具有四十个洞以上的数独棋盘。

2. 对游戏可玩性进行简单测试。

3. 测试文件保存功能的完备性。

表4-3 数独游戏模块测试步骤

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 输入 | 预计输出 | 实际输出 |
| 1 | 2（进入Sudoku模块） | Sudoku界面 | 如图4-25 |
| 2 | 1（生成数独棋盘） | “请输入挖洞数量” | 如图4-26 |
| 3 | 50（挖五十个洞） | 数独棋盘 | 如图4-27 |
| 4 | 2（玩数独游戏） | 玩数独游戏界面 | 如图4-28 |
| 5 | 1（添加一个数） | 当前数独游戏界面 | 如图4-29 |
| 6 | 5 1 5（添加数的位置） | 删除后数独界面 | 如图4-30 |
| 7 | 2（删除一个数） | 当前数独游戏界面 | 如图4-31 |
| 8 | 5 1（删除数的位置） | 删除后数独界面 | 如图4-32 |
| 9 | 0（生成答案） | 最终的数独终盘 | 如图4-33 |
| 10 | 3（保存答案） | “保存成功” | 如图4-34 |
| 11 | 0（退出当前模块） | “欢迎下次来玩” | 如图4-36 |

下面是相应的运行结果截图。

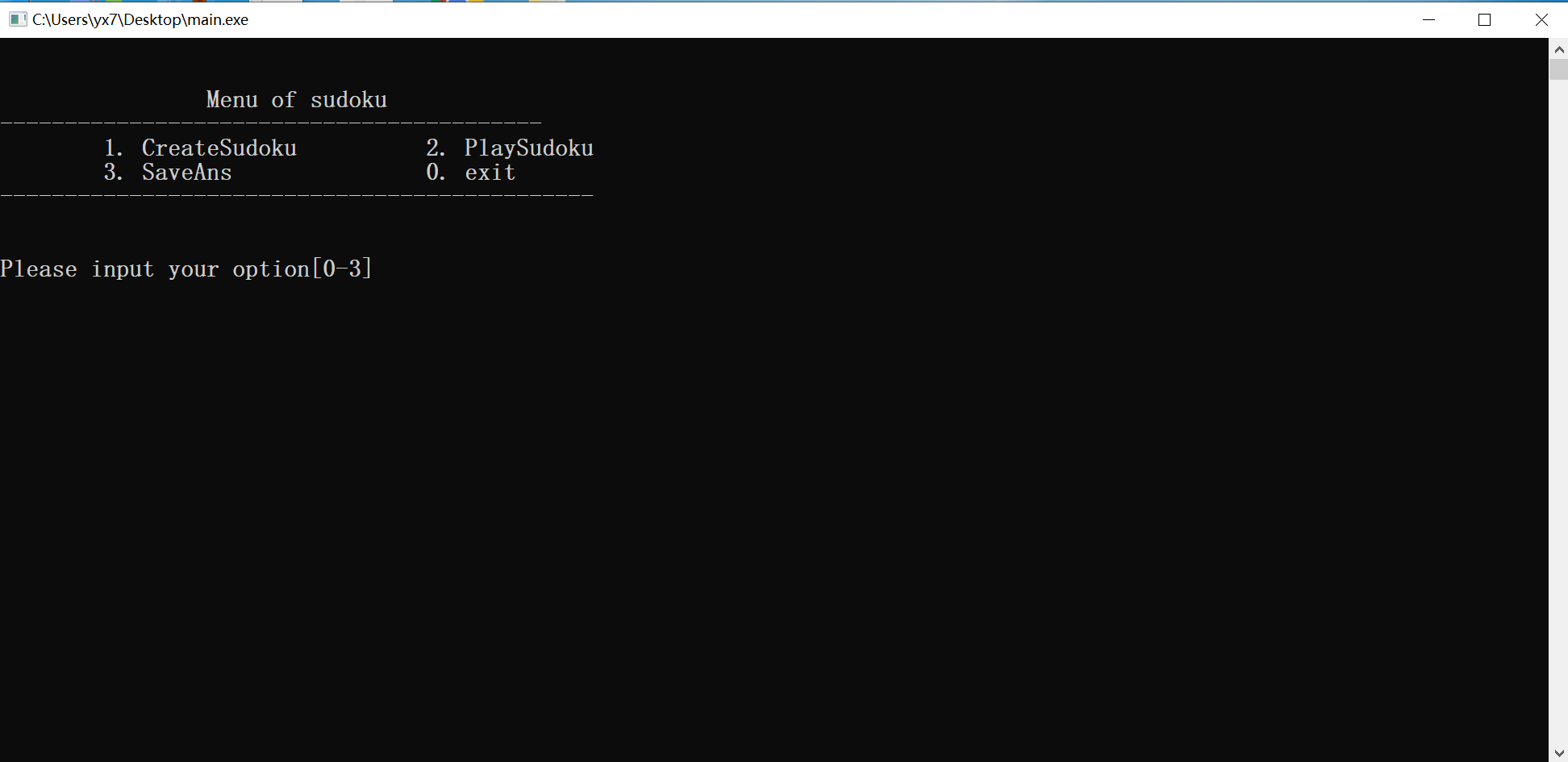


图4-25 步骤一运行结果

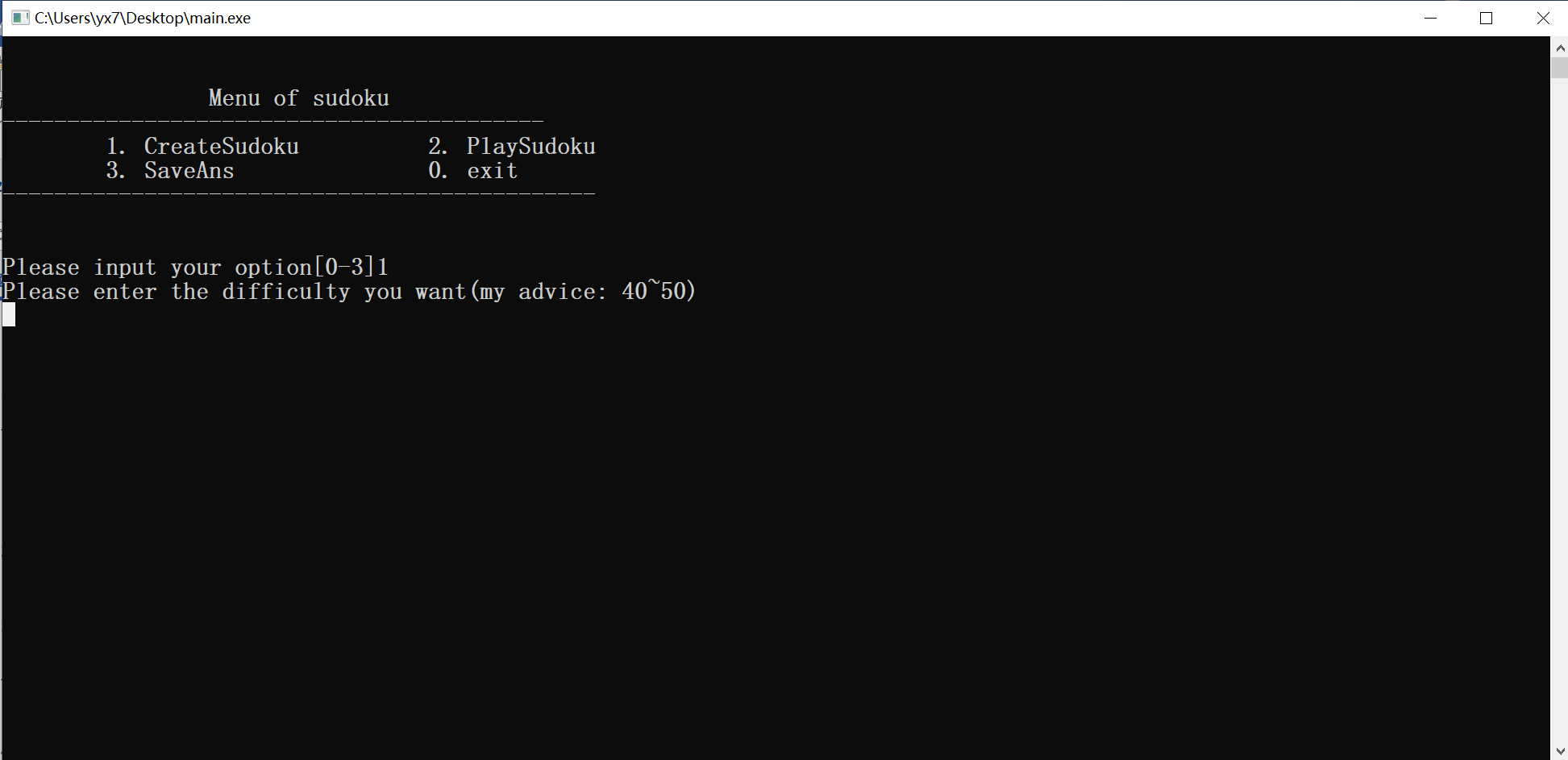


图4-26 步骤二运行结果

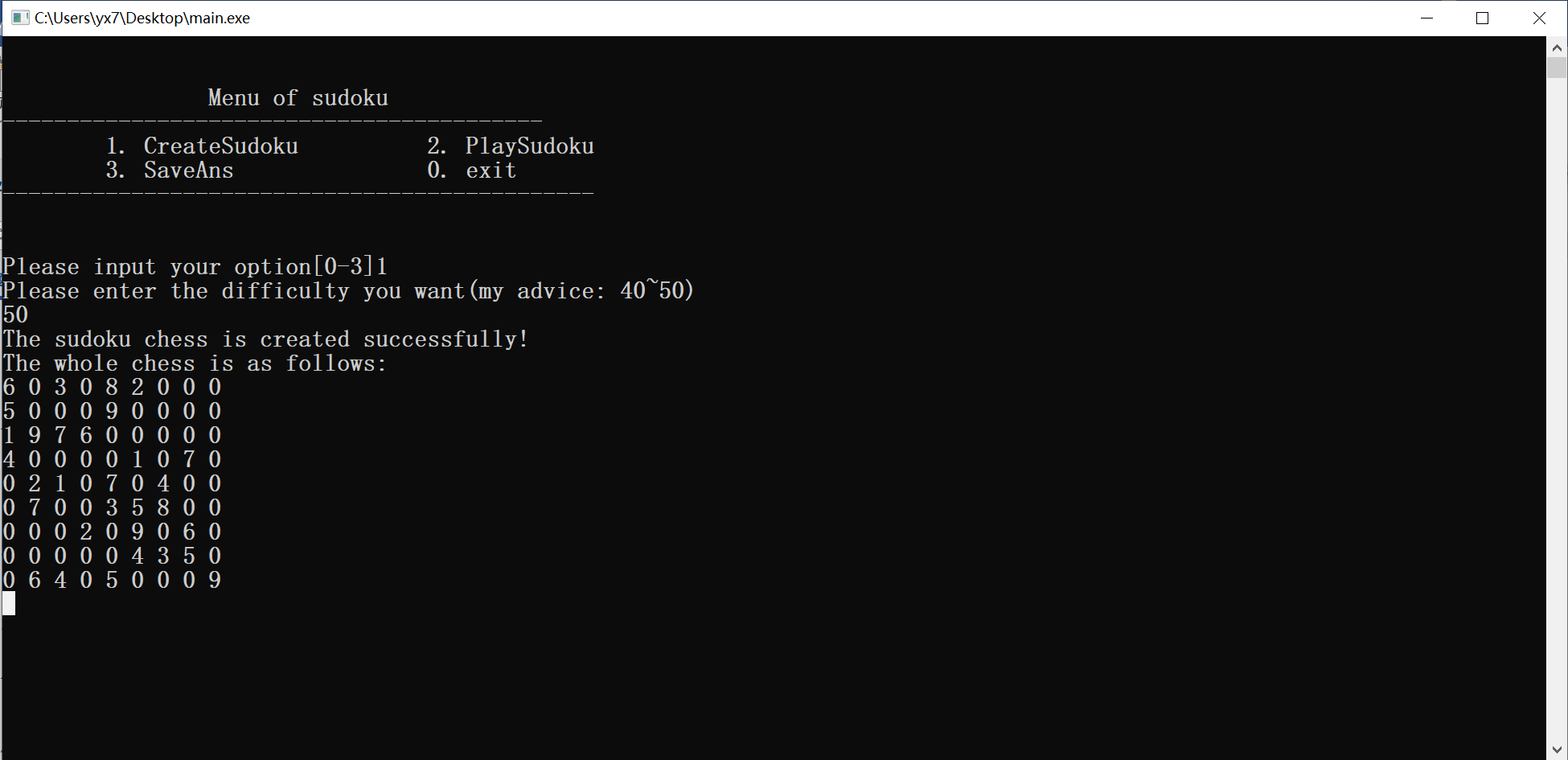


图4-27 步骤三运行结果

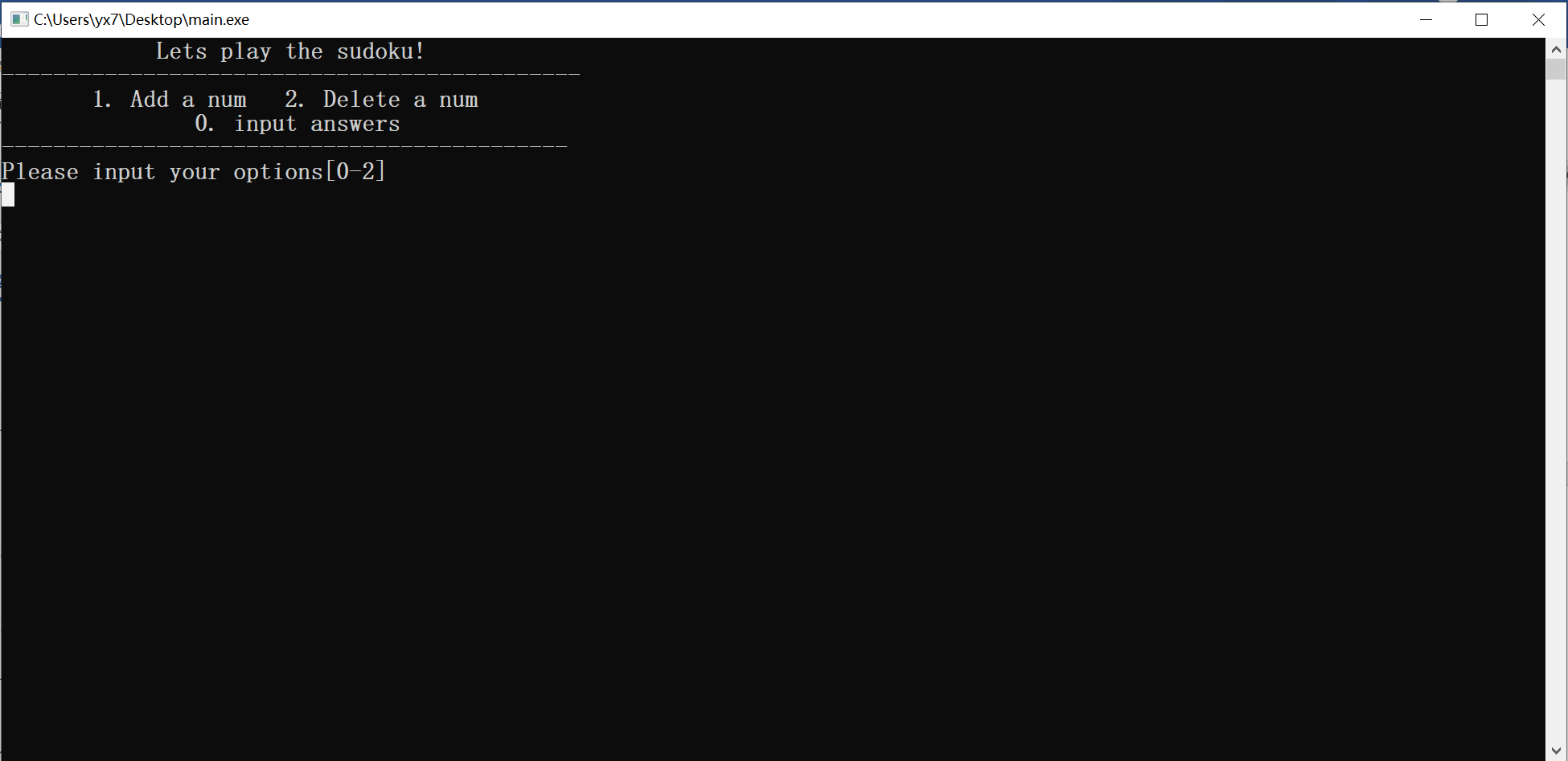


图4-28 步骤四运行结果

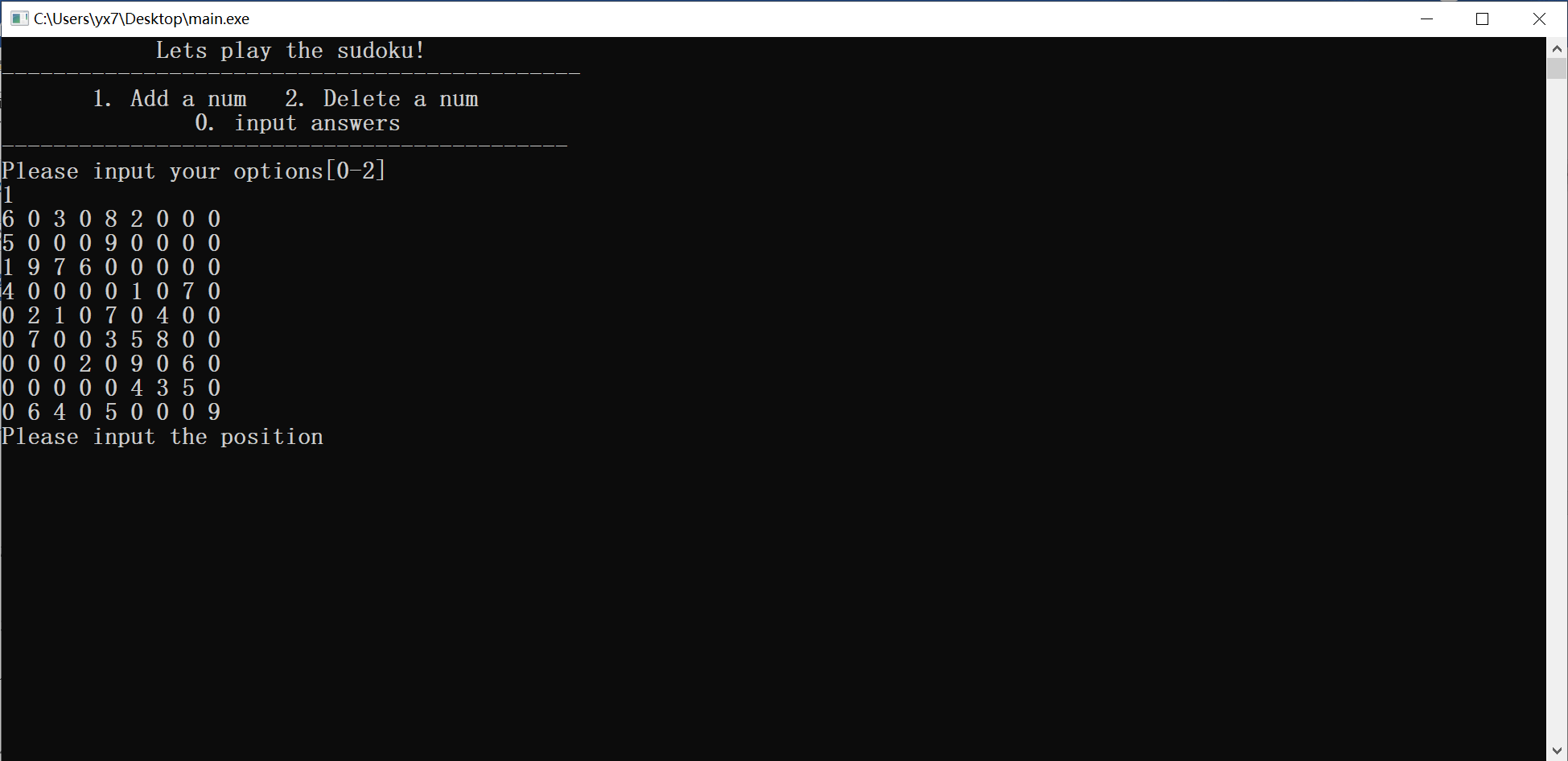


图4-29 步骤五运行结果

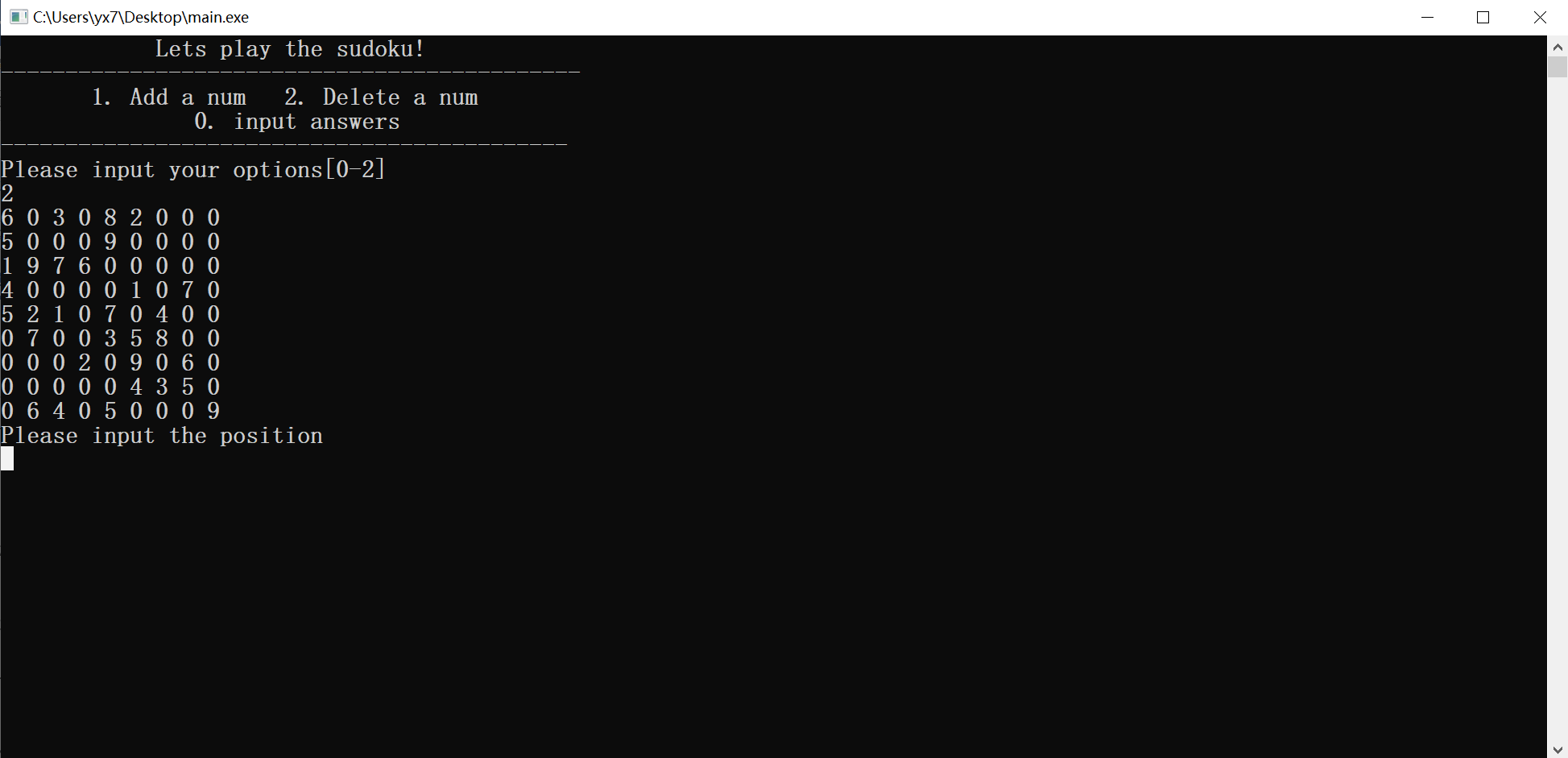


图4-30 步骤六运行结果

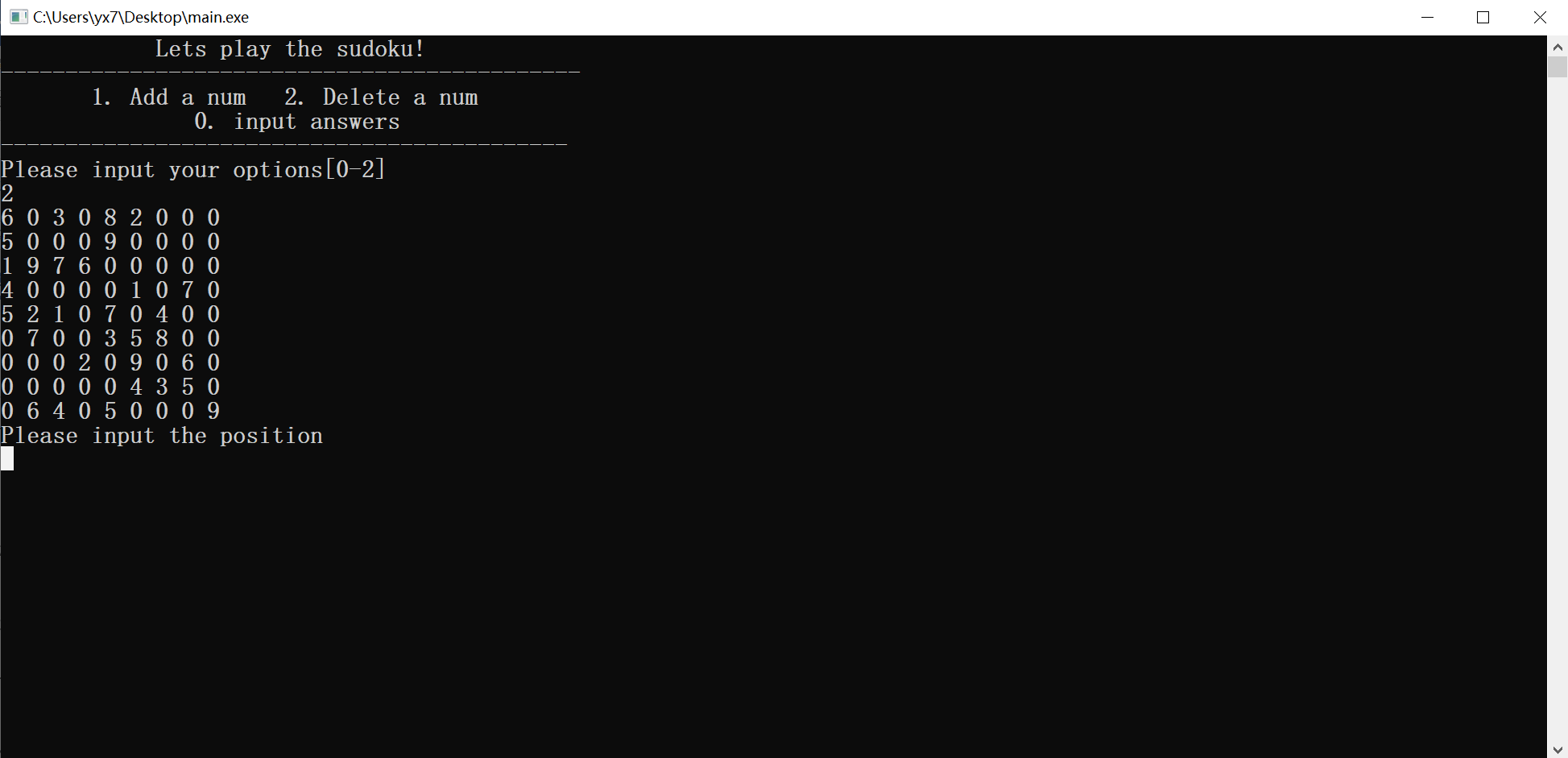


图4-31 步骤七运行结果

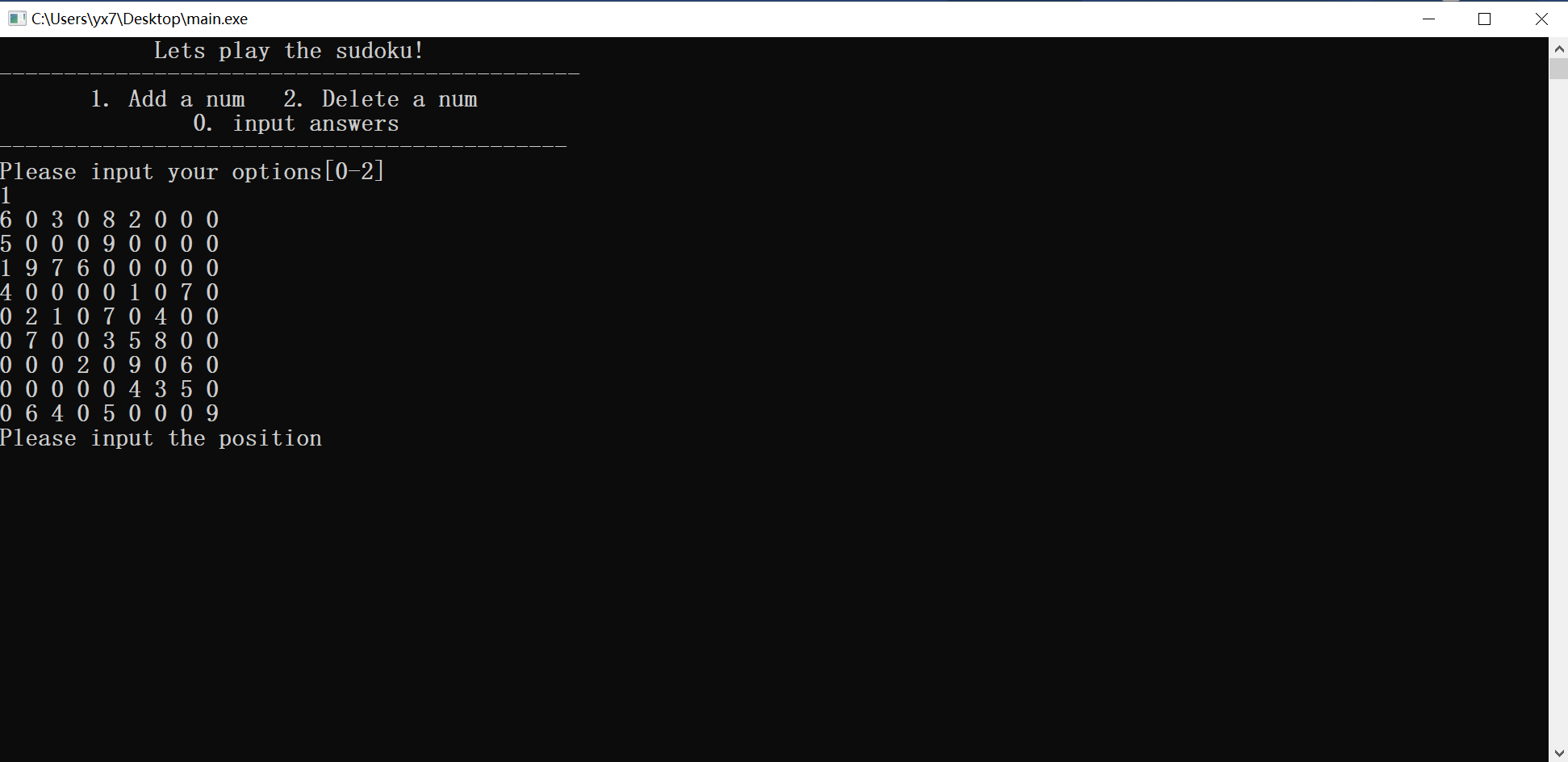


图4-32 步骤八运行结果

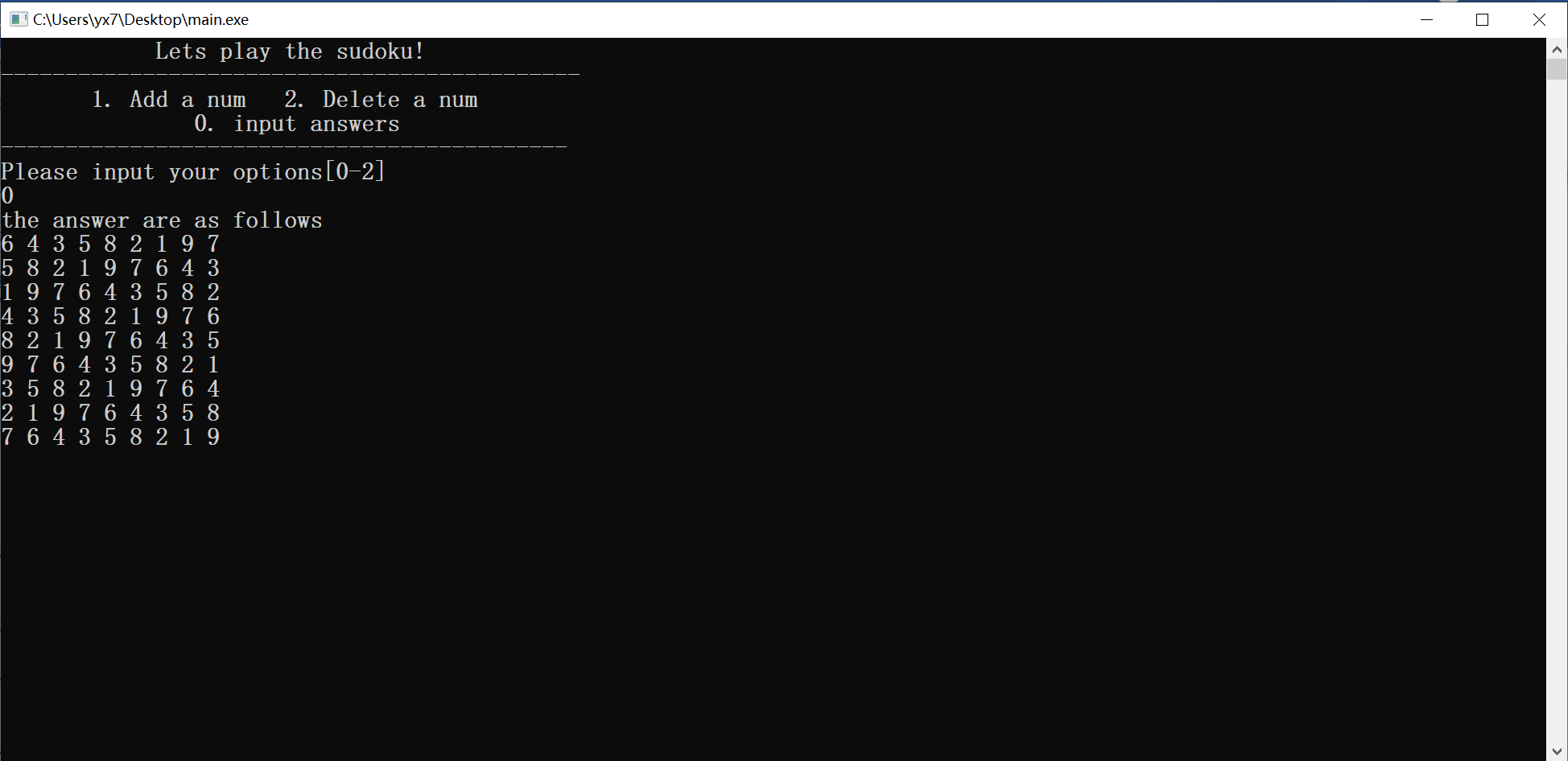


图4-33 步骤九运行结果

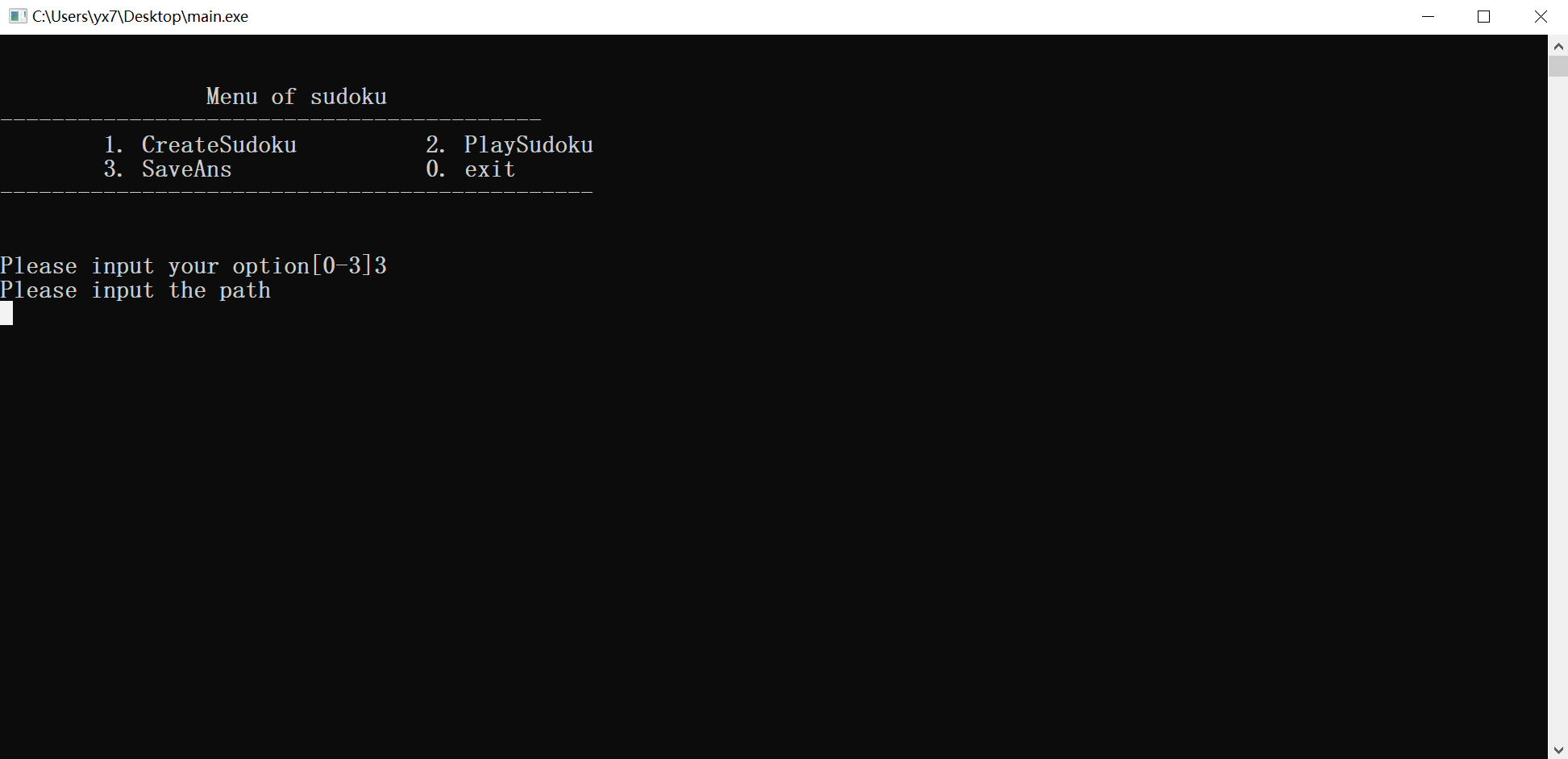


图4-34 步骤十运行结果

保存的文件内容如图4-35所示。

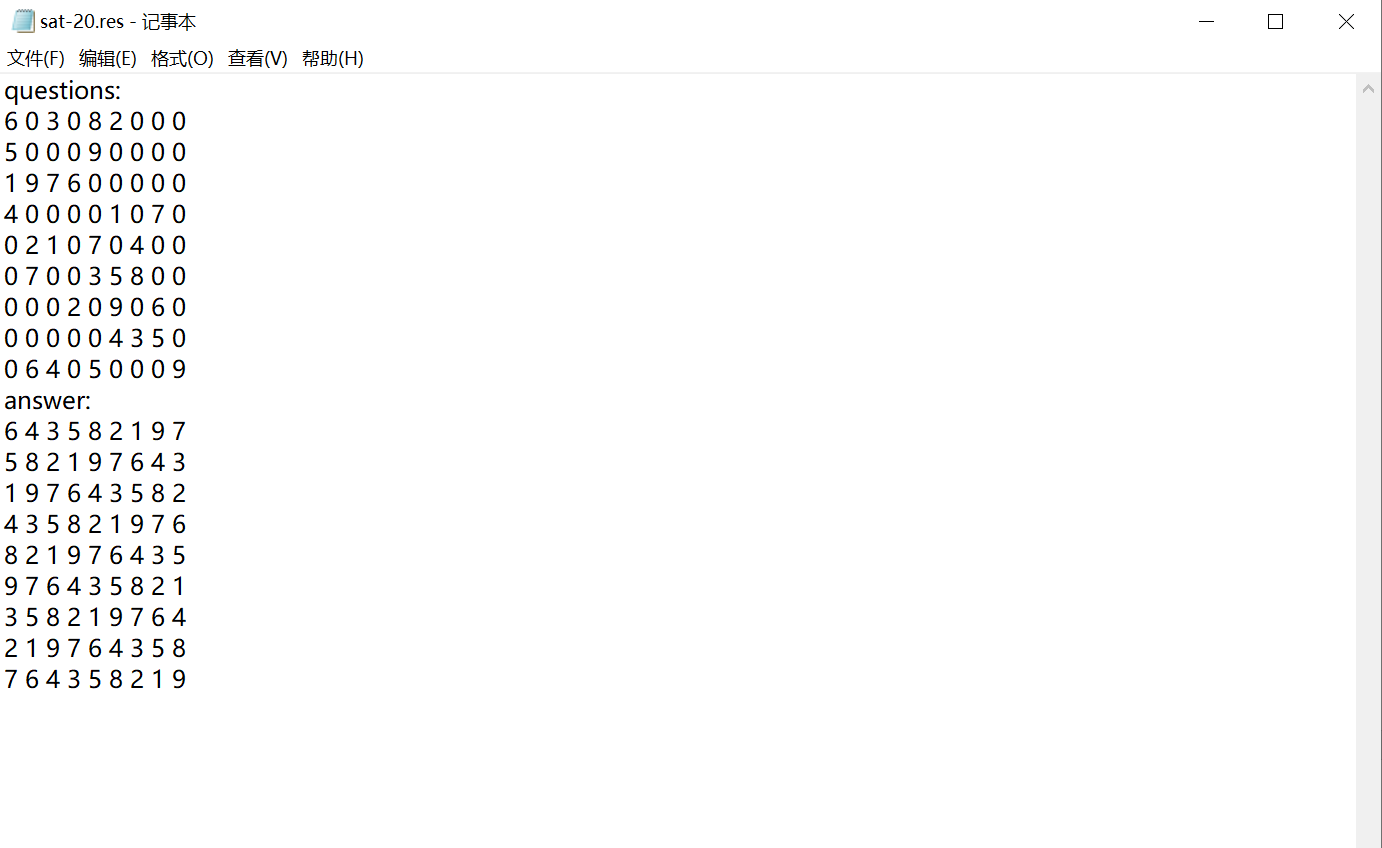
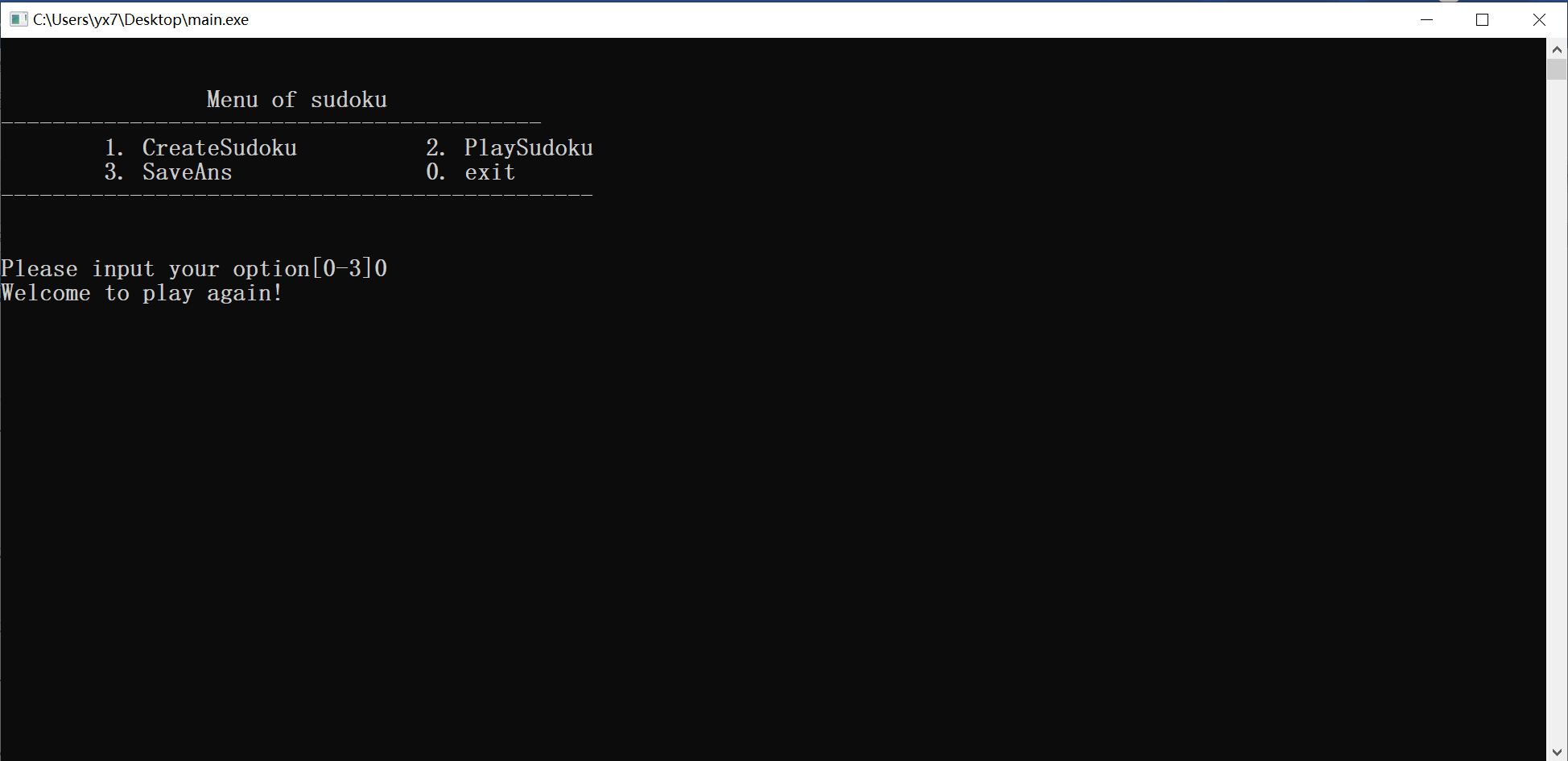


图4-35 数独结果文件



4-36 步骤十二运行结果

就实际运行结果来看，数独模块方面的完成了课程设计的要求。本系统可以利用挖洞法生成具有唯一解的随机数独游戏棋盘，在挖洞的过程中利用DPLL算法对当前挖洞点的可行性进行判断，因此需要调用多次DPLL算法，这就对DPLL算法的性能有了较高的要求，事实上，经过多次实验，在使用优化前的DPLL算法时，挖洞数量超过二十的时候就需要运行较长的一段时间，而优化了DPLL算法之后，挖洞数量可以达到接近六十。

另外就是用户玩数独游戏这一方面的内容，用户主要有三种操作，即在某个位置上添加数字，删除自己填的某个地方的数字，另外就是在用户不想玩的时候可以输出整个数独游戏的终盘。

最后就是用户可以选择将本次游戏的棋盘和终盘保存到给定路径下的文件中。

# 5总结与展望

## 5.1全文总结

本次课程设计主要完成了关于求解sat问题的DPLL算法的研究和实现，对最初的DPLL算法做了相应的优化，同时将数独游戏转化成sat问题进行求解，主要完成了以下工作：

1.了解学习sat问题基本含义：了解了什么叫做布尔可满足性问题，复习了离散数学中命题逻辑相关的知识，并为接下来学习数字逻辑课程打下了一定基础。同时阅读了相关文献，了解了当前国内外对布尔可满足性问题的研究以及该问题的前景。

2.在课设开始初期要求的简单功能：通过阅读课设相关文档了解到了课程设计的基本要求，选取邻接表作为基本的存储结构，并且通过对CNF文件内容的分析成功实现将CNF文件存储到邻接表中的功能，并且能够对文件进行逐行打印功能。

3.完成DPLL算法并成功求解sat问题：通过阅读和DPLL算法相关的文档，更深切的理解到了DPLL算法中包含的思想，并且通过在变元选取这一方面对DPLL算法做了优化。

4.在求解sat问题的基础上完成了求解数独游戏模块：通过阅读相关的文章，将9\*9数独棋盘转化成CNF范式，并将其整合到DPLL算法中，参考相关文章，使用挖洞法生成一个具有唯一解的数独游戏棋盘。

## 5.2工作展望

本次课设完成度尚可，但是由于时间仓促，仍然存在着一些遗憾，也有很多可以补充的东西。

1.DPLL算法可以继续优化。我在DPLL算法优化这一部分主要是考虑在变元选取方面进行优化，最开始是选择出现次数最多的变元，后来更改为先选择出现次数最多的正变元，在没有正变元的情况下，选择出现次数最多的负变元。虽然相比于之前已经有了比较好的优化，但是有相当一部分样例仍然跑不出来。另外就是在查阅文献的过程中了解到了目前在求解sat问题上常用的CDCL算法和LA算法，还有一些不完备求解算法和组合算法，用这些算法应该有更高的求解效率，但是由于时间关系，没能有更深一步的研究。目前我个人是在学习分布式计算的一些入门知识，希望以后在接触到这方面更多的知识之后能够采用并行算法对DPLL算法有更好的优化。

2.可以做出更好的交互性。由于本次实验的时间较紧，最终做的交互性较差，一些可能存在的用户的错误输入没能检查到位，另外在玩数独游戏这一部分，也仅仅只有加数、删数、生成答案这三个功能，还是非常遗憾的。

3.在测试的时候，发现会有一些特殊的数据使用特定的算法会更加简单。比如之前学习数据结构课程中的排序算法时，快速排序在绝大多数情况下运行速度更快，但是对于某些特殊的数据，可以考虑直接使用插入排序，性能会比直接使用快速排序更优。C++中STL的sort函数也正是基于这一点进行设计的。这也启发我们在未来的算法设计中可以多设计集中不同的算法，针对各种不同的数据来进行求解。

# 6体会

本次课程设计相比于之前的C语言实验和数据结构实验，工程量更大，难度也更高，本人体会如下：

1. 首先在课程设计初期，需要有明确的规划，明确课题要求我们做什么，明确问题的真实含义，需要根据问题的具体要求设计好所需的数据结构，然后明确在设计中需要实现哪些函数，各函数之间的调用关系是怎样的，设计好整个程序的基本框架，这样才能够顺利的完成整个程序。

2. 在整个课程设计过程中，需要善于在网上查询相关的资料。课设任务书中给出的相关的算法细节等信息还是较少，而网上的资源是非常丰富的，从DPLL算法的设计和各种优化到用挖洞法生成具有唯一解的数独棋盘，都有较为详细的论文和资料，值得参考。

3. 对于一个较大的设计，需要有一个良好的时间规划，需要明确在哪个时间要做好哪些事情，另外需要有一定的弹性时间安排，否则最后很可能因为时间不够而导致没能很好的完成课程设计。

对于本次程序设计，我有如下收获：

1. 巩固了C语言和数据结构课程的知识，对相关知识有了更深的了解。

2. 提升了自己的编程能力，整个课程设计的总工程量还是比较大的，这更加锻炼了我的全局思考能力。

3. 锻炼了自己上网搜集资料的能力。由于本次课程设计所需要的知识和全新的算法难度较高，而课设任务书上所给的知识又太少，所以很多知识都需要自己上网去搜集解决方案，去搜集更详细的算法讲解，这进一步提升了我上网搜集资料的能力。

4. 让我在编程方面有了更好的习惯。之前编写的大多数算法对空间和时间性能要求并不高，所以就经常直接开大数组，但是在这次课程设计中，要合理的使用内存，尽可能提升程序运行效率，几乎所有的数组都要用malloc函数为其动态分配空间，然后用完后及时free以释放空间，而在之前编写的一些程序中我甚至都没有free的习惯。

5. 培养了自己的统筹能力，要求我合理规划时间完成任务。

# 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

本次课程设计的所有代码如下。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define TRUE 1 *//真值设为1*

#define FALSE 0 *//假值设为0*

#define OK 1 *//预处理定义*

#define ERROR 0 *//定义ERROR*

#define INFEASTABLE -1 *//定义报错*

#define OVERFLOW -2 *//定义溢出*

int boolnum;

int clausnum, firstdelx, firstdely;

int \*firstline;

int shift[10] = {0, 3, 6, 1, 4, 7, 2, 5, 8};

typedef struct \_charanode

{

    int data;

    struct \_charanode \*next;

} charanode;

typedef struct \_cnf

{

*//int length;*

    charanode \*head;

    struct \_cnf \*next;

} cnf;

int readCNF(cnf \*&*CNF*, char *Filename*[80]);*//读取CNF文件*

void deleteCNF(cnf \*&*CNF*);

void showCNF(cnf \**CNF*);*//逐行打印CNF范式*

int isOneClause(charanode \**chara*);

int deleteClause(cnf \*&*CNF*, cnf \*&*deleted*);

int deletechara(charanode \*&*head*, charanode \*&*CNF*);

int iswithEmptyClause(cnf \**CNF*);

int betterDPLL(cnf \*&*CNF*, int *value*[]);

void addClause(cnf \**CNF*, cnf \*&*root*);

void copyclause(cnf \*&*CNF*, cnf \**CNF2*);

int savedata(int *value*[], char *Filename*[], double *time*, int *result*);

void createsudoku(int *sudoku*[][9]);

void dfs();

int solveSudoku(int *sudoku*[][9]);

void dighole(int *sudoku*[][9], int *n*);

int check(int *sudoku*[][9], int *x*, int *y*);

int DPLL(cnf \*&*CNF*, int *value*[]);

int main()

{

    cnf \*CNF = NULL;

     int sudoku[9][9];

                int sudoku1[9][9];

    int opt = 1, opt1 = 1, opt2 = 1;

    int \*value;

    clock\_t start, finish;

    char Filename[80];

    while (opt)

    {

        system("cls");

        opt1 = 1, opt2 = 1;

        printf("\n\n");

        printf("                    Menu Of options   \n");

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

        printf("             1. SAT              2. Sudoku           \n");

        printf("                      0. exit\n");

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

        printf("Please input your option[0-2]\n");

        scanf("%d", &opt);

        switch (opt)

        {

        case 1:

        {

            while (opt1)

            {

                system("cls");

                printf("\n\n");

                printf("                   Menu of SAT      \n");

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

                printf("          1. ReadCNF        2. ShowCNF           \n");

                printf("          3. BetterDPLL           4. DPLL        \n");

                printf("                      0. exit    \n");

                printf("Please input your option[0-4]");

                scanf("%d", &opt1);

                switch (opt1)

                {

                case 1:

                {

                    printf("Please input the path of the CNF file\n");

                    scanf("%s", Filename);

                    int ret = readCNF(CNF, Filename);

                    if (ret == 0)

                        printf("the path of file is error!\n");

                    else

                        printf("read the data from the CNF file successfully\n");

                    getchar();

                    getchar();

                    break;

                } *// end of case1 in sat*

                case 2:

                {

                    if (CNF == NULL)

                    {

                        printf("Please read the data first!\n");

                    }

                    else

                    {

                        printf("the clause in this file is as follows:\n");

                        showCNF(CNF);

                    }

                    getchar();

                    getchar();

                    break;

                } *// end of case2 in sat*

                case 3:

                {

                    if (CNF == NULL)

                    {

                        printf("Please read the data first!\n");

                    }

                    else

                    {

                        value = (int \*)malloc(sizeof(int) \* (boolnum + 1));

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

                            value[i] = 1;

                        start = clock();

                        int result = betterDPLL(CNF, value);

                        finish = clock();

                        printf("result: %d\n", result);

                        if (result)

                        {

                            printf("the answer is as follows:\n");

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

                            {

                                if (value[i])

                                    printf("%d ", i);

                                else

                                    printf("-%d ", i);

                            }

                        }

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

                        printf("\nruntime:%lf\n", time \* 1000);

                        savedata(value, Filename, time, result);

                        printf("This file is saved with the same name of CNF\n");

                    }

                    getchar();

                    getchar();

                    break;

                } *// end of case3 in sat*

                case 4:

                {

                    value = (int \*)malloc(sizeof(int) \* (boolnum + 1));

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

                        value[i] = 1;

                    int result;

                    start = clock();

                    result = DPLL(CNF, value);

                    finish = clock();

                    printf("result: %d\n", result);

                    if (result)

                    {

                        printf("the answer is as follows:\n");

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

                        {

                            if (value[i])

                                printf("%d ", i);

                            else

                                printf("-%d ", i);

                        }

                    }

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

                    printf("\nruntime:%lf\n", time \* 1000);

                    savedata(value, Filename, time, result);

                    printf("This file is saved with the same name of CNF\n");

                    getchar();

                    getchar();

                    break;

                } *// end of case4 in sat*

                case 0:

                {

                    break;

                } *// end of case0 in sat*

                } *// end of the sat switch*

            } *// end of the sat while*

            break;

        } *// end of case 1*

        case 2:

        {

            while (opt2)

            {

                system("cls");

                printf("\n\n");

                printf("                Menu of sudoku\n");

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

                printf("        1. CreateSudoku          2. PlaySudoku\n");

                printf("        3. SaveAns               0. exit\n");

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

                printf("\n\n");

                printf("Please input your option[0-3]");

                scanf("%d", &opt2);

                switch (opt2)

                {

                case 1:

                {

                    createsudoku(sudoku);

                    printf("The sudoku chess is created successfully!\n");

                    printf("The whole chess is as follows:\n");

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

                    {

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

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

                        printf("\n");

                    }

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

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

                            sudoku1[i][j] = sudoku[i][j];

                    getchar();

                    getchar();

                    break;

                } *//end of case 1 in sudoku*

                case 2:

                {

                    int opt3 = 1;

                    int termsudoku[9][9];

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

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

                            termsudoku[i][j] = sudoku[i][j];

                    while (opt3)

                    {

                        system("cls");

                        printf("            Lets play the sudoku!\n");

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

                        printf("       1. Add a num   2. Delete a num\n");

                        printf("               0. input answers\n");

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

                        printf("Please input your options[0-2]\n");

                        scanf("%d", &opt3);

                        switch (opt3)

                        {

                        case 1:

                        {

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

                            {

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

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

                                printf("\n");

                            }

                            printf("Please input the position\n");

                            int x, y;

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

                            x--, y--;

                            printf("Please input the num\n");

                            int data;

                            scanf("%d", &data);

                            if (x >= 0 && x <= 8 && y >= 0 && y <= 8 && data >= 1 && data <= 9)

                            {

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

                                    continue;

                                else

                                    termsudoku[x][y] = data;

                            }

                            else

                            {

                                printf("The input is error!\n");

                                printf("Please input again!\n");

                                continue;

                            }

                            getchar();

                            getchar();

                            break;

                        }

                        case 2:

                        {

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

                            {

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

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

                                printf("\n");

                            }

                            printf("Please input the position\n");

                            int x, y;

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

                            x--, y--;

                            if (x >= 0 && x <= 8 && y >= 0 && y <= 8)

                            {

                                if (termsudoku[x][y] == 0)

                                    continue;

                                termsudoku[x][y] = 0;

                            }

                            else

                            {

                                printf("The input is error!\n");

                                printf("Please input again!\n");

                                continue;

                            }

                            getchar();

                            getchar();

                            break;

                        }

                        case 0:

                        {

                            printf("the answer are as follows\n");

                            if (solveSudoku(sudoku))

                            {

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

                                {

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

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

                                    printf("\n");

                                }

                            }

                            getchar();

                            getchar();

                            break;

                        }

                        }

                    }

                    getchar();

                    getchar();

                    break;

                }

                case 3:

                {

                    printf("Please input the path\n");

                    char filename[80];

                    FILE \*fp;

                    scanf("%s", filename);

                    fp = fopen(filename, "w");

                    if (fp == NULL)

                    {

                        printf("ERROR!\n");

                        break;

                    }

                    fprintf(fp, "questions:\n");

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

                    {

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

                            fprintf(fp, "%d ", sudoku1[i][j]);

                        fprintf(fp, "\n");

                    }

                    if(solveSudoku(sudoku)){

                    fprintf(fp, "answer:\n");

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

                    {

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

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

                        fprintf(fp, "\n");

                    }

                    fclose(fp);}

                    else

                    {

                        fprintf(fp, "No answer!\n");

                        fclose(fp);

                    }

                    break;

                }

                case 0:

                {

                    printf("Welcome to play again!\n");

                    getchar();

                    getchar();

                    break;

                }

                }

            }

            break;

        }

        case 0:

        {

            printf("welcome to this system next time!\n");

            break;

        }

        } *// end of switch*

    } *// end of while*

}

int readCNF(cnf \*&*CNF*, char *Filename*[80]) *//读入CNF文件，存入数据结构*

{

    FILE \*fp;

    fp = fopen(*Filename*, "r");

    if (fp == NULL)

    {

        return 0;

    }

    char c;

    fscanf(fp, "%c", &c);

    while (c == 'c')

    {

        while (c != '\n' && c != '\r')

            fscanf(fp, "%c", &c);

        fscanf(fp, "%c", &c);

        if (c == '\n')

            fscanf(fp, "%c", &c);

    }

    fscanf(fp, " cnf ");

    fscanf(fp, "%d", &boolnum);

    fscanf(fp, "%d", &clausnum);

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

*CNF*->next = NULL;

*CNF*->head = (charanode \*)malloc(sizeof(charanode));

*CNF*->head->next = NULL;

    cnf \*tailcnf = *CNF*;

    charanode \*tailbool = *CNF*->head;

    int key;

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

    {

        fscanf(fp, "%d", &key);

        while (key != 0)

        {

*//tailcnf->length++;*

            tailbool->data = key;

            tailbool->next = (charanode \*)malloc(sizeof(charanode));

            fscanf(fp, "%d", &key);

            if (key == 0)

                tailbool->next = NULL;

            tailbool = tailbool->next;

        }

        tailcnf->next = (cnf \*)malloc(sizeof(cnf));

*//tailcnf->next->length = 0;*

        tailcnf->next->head = (charanode \*)malloc(sizeof(charanode));

        if (i == clausnum - 1)

        {

            tailcnf->next = NULL;

            break;

        }

        tailcnf = tailcnf->next;

        tailbool = tailcnf->head;

    }

    fclose(fp);

    return 1;

}

void showCNF(cnf \**CNF*) *//遍历整个邻接表，逐行打印CNF范式*

{

    cnf \*p = *CNF*;

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

    {

        charanode \*q = p->head;

        while (q)

        {

            printf("%d ", q->data);

            q = q->next;

        }

        printf("\n");

        p = p->next;

    }

}

int isOneClause(charanode \**chara*) *//判断该子句是否为单子句*

{

    if (*chara* != NULL && *chara*->next == NULL)

        return 1;

    return 0;

}

int deleteClause(cnf \*&*CNF*, cnf \*&*deleted*) *//删除邻接表中的deleted子句*

{

    cnf \*p = *CNF*;

    if (*deleted* == p) *//是否为第一个*

    {

*CNF* = *CNF*->next; *//直接指向下一个*

        free(*deleted*); *//释放内存*

*deleted* = NULL;

        return 1;

    }

    else *//如果不是第一个*

    {

        while (p && p->next != *deleted*)

            p = p->next;

        p->next = p->next->next;

        free(*deleted*);

*deleted* = NULL;

        return 1;

    }

    return 0;

}

int deletechara(charanode \*&*head*, charanode \*&*deleted*) *//删除当前子句中的某个文字*

{

    charanode \*p = *head*;

    if (p == *deleted*)

    {

*head* = *head*->next;

        free(*deleted*);

*deleted* = NULL;

        return 1;

    }

    else

    {

        while (p != NULL && p->next != *deleted*)

            p = p->next;

        p->next = p->next->next;

        free(*deleted*);

*deleted* = NULL;

        return 1;

    }

    return 0;

}

int iswithEmptyClause(cnf \**CNF*) *//判断其是否含有空子句*

{

    cnf \*p = *CNF*;

    while (p != NULL)

    {

        if (p->head == NULL)

            return 1;

        p = p->next;

    }

    return 0;

}

void copyclause(cnf \*&*CNF*, cnf \**CNF2*) *//该函数将CNF2的值赋到CNF*

{

    cnf \*cocnf, \*cocnf2;

    charanode \*boolnode1, \*boolnode2;

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

*CNF*->head = (charanode \*)malloc(sizeof(charanode));

*CNF*->next = NULL;

*CNF*->head->next = NULL; *//初始化当前的CNF范式*

    for (cocnf = *CNF*, cocnf2 = *CNF2*; cocnf2; cocnf2 = cocnf2->next, cocnf = cocnf->next)

    { *//逐行遍历邻接表进行复制*

        for (boolnode1 = cocnf->head, boolnode2 = cocnf2->head; boolnode2; boolnode1 = boolnode1->next, boolnode2 = boolnode2->next)

        {

            boolnode1->data = boolnode2->data;

            boolnode1->next = (charanode \*)malloc(sizeof(charanode));

            boolnode1->next->next = NULL;

            if (boolnode2->next == NULL) *//特判一下，到达最后的话就进行处理*

            {

                free(boolnode1->next);

                boolnode1->next = NULL;

            }

        }

        cocnf->next = (cnf \*)malloc(sizeof(cnf)); *//处理新的CNF范式*

        cocnf->next->head = (charanode \*)malloc(sizeof(charanode));

        cocnf->next->head->next = NULL;

        cocnf->next->next = NULL;

        if (cocnf2->next == NULL) *//同样的特判*

        {

            free(cocnf->next->head);

            free(cocnf->next);

            cocnf->next = NULL;

        }

    }

}

int betterDPLL(cnf \*&*CNF*, int *value*[]) *//用优化后的DPLL算法求解sat问题*

{

    cnf \*p, \*q, \*r;

    p = *CNF*, q = *CNF*;

    charanode \*boolnode;

    int \*num, max, maxpos; *//num数组用来存储变元的出现次数*

    while (p != NULL)

    {

        while (p && isOneClause(p->head) == 0)

            p = p->next;

        if (p)

        {

            int singlekey = p->head->data;

            if (singlekey > 0)

*value*[singlekey] = 1;

            else

*value*[-singlekey] = 0;

            for (q = *CNF*; q; q = r)

            {

                r = q->next;

                for (boolnode = q->head; boolnode; boolnode = boolnode->next)

                {

                    if (boolnode->data == singlekey)

                    {

                        deleteClause(*CNF*, q);

                        break;

                    }

                    if (boolnode->data == -singlekey)

                    {

                        deletechara(q->head, boolnode);

                        break;

                    }

                }

            }

            if (*CNF* == NULL)

            {

                return 1;

            }

            else if (iswithEmptyClause(*CNF*))

            {

                deleteCNF(*CNF*);

                return 0;

            }

            p = *CNF*;

        }

    }

    num = (int \*)malloc(sizeof(int) \* (2 \* boolnum + 2));

    for (int i = 0; i <= 2 \* boolnum; i++)

        num[i] = 0;

    for (p = *CNF*; p; p = p->next)

    {

        for (boolnode = p->head; boolnode; boolnode = boolnode->next)

            if (boolnode->data > 0)

                num[boolnode->data]++;

            else

                num[boolnum - boolnode->data]++;

    }

    max = 0;

    maxpos = 0;

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

    {

        if (max < num[i])

        {

            max = num[i];

            maxpos = i;

        }

    }

    if (max == 0)

        for (int i = boolnum + 1; i <= 2 \* boolnum; i++)

        {

            if (max < num[i])

            {

                max = num[i];

                maxpos = -i;

            }

        }

    free(num);

    p = (cnf \*)malloc(sizeof(cnf));

    p->head = (charanode \*)malloc(sizeof(charanode));

    p->head->data = maxpos;

    p->head->next = NULL;

    p->next = NULL;

    q = NULL;

    copyclause(q, *CNF*);

    if (p != NULL)

    {

        p->next = q;

        q = p;

    }

    if (betterDPLL(q, *value*) == 1)

        return 1;

    deleteCNF(q);

    p = (cnf \*)malloc(sizeof(cnf));

    p->head = (charanode \*)malloc(sizeof(charanode));

    p->head->data = -maxpos;

    p->head->next = NULL;

    p->next = NULL;

    if (p != NULL)

    {

        p->next = *CNF*;

*CNF* = p;

    }

    int i = betterDPLL(*CNF*, *value*);

    deleteCNF(*CNF*);

    return i;

}

void deleteCNF(cnf \*&*CNF*)

{

    cnf \*p1, \*p2;

    charanode \*q1, \*q2;

    for (p1 = *CNF*; p1 != NULL; p1 = p2)

    {

        p2 = p1->next;

        for (q1 = p1->head; q1 != NULL; q1 = q2)

        {

            q2 = q1->next;

            free(q1);

        }

        free(p1);

    }

*CNF* = NULL;

}

int savedata(int *value*[], char *Filename*[], double *time*, int *result*)

{

    FILE \*fp;

    for (int i = 0; *Filename*[i] != '\0'; i++)

    {

        if (*Filename*[i] == '.' && *Filename*[i + 4] == '\0')

        {

*Filename*[i + 1] = 'r';

*Filename*[i + 2] = 'e';

*Filename*[i + 3] = 's';

            break;

        }

    }

    fp = fopen(*Filename*, "w");

    if (fp == NULL)

    {

        printf("can't open this file!\n");

        return 0;

    }

    fprintf(fp, "s %d\nv ", *result*);

    if (*result* == 1)

    {

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

        {

            if (*value*[i])

                fprintf(fp, "%d ", i);

            else

                fprintf(fp, "-%d ", i);

        }

    }

    fprintf(fp, "\nt %lf", *time* \* 1000);

    fclose(fp);

    return 1;

}

*//求解sat问题的函数就到此结束了*

*//下面是求解数独游戏的函数*

void createsudoku(int *sudoku*[][9])

{

    firstline = (int \*)malloc(sizeof(int) \* 10);

    dfs(); *//生成第一行的一个随机全排列*

    memcpy(*sudoku*[0], firstline, sizeof(int) \* 9);

    for (int i = 1; i < 9; i++)

    {

        for (int j = 0; j < 9; j++)

        {

*sudoku*[i][j] = firstline[(j + shift[i]) % 9];

*//通过轮换来得到一个数独*

        }

    }

    free(firstline);

    firstline = NULL;

    int n;

    printf("Please enter the difficulty you want(my advice: 40~50)\n");

    scanf("%d", &n);

    dighole(*sudoku*, n);

}

void dfs()

{

    int a[10];

    for (int i = 1; i <= 9; i++)

        a[i] = i;

    int n = 9;

    int top = 0;

    srand((int)time(0));

    for (int i = 1; i <= 9; i++)

    {

        int x = 1 + rand() % n;

        firstline[top++] = a[x];

        int temp = a[x];

        a[x] = a[n];

        a[n] = temp;

        n--;

    }

    return;

}

*//下面的函数调用DPLL算法求解数独游戏*

int solveSudoku(int *sudoku*[][9])

{

    cnf \*Sudoku = NULL, \*clausep = NULL;

    charanode \*BoolNode = NULL;

    int size = 9, size2 = 81;

    int i, j, k;

*//下面的循环加入自身约束*

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

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

            clausep->head = (charanode \*)malloc(sizeof(charanode));

            clausep->next = NULL;

            clausep->head->next = NULL;

            BoolNode = clausep->head;

            for (k = 1; k <= size; k++)

            {

                BoolNode->data = i \* size2 + j \* size + k;

                BoolNode->next = (charanode \*)malloc(sizeof(charanode));

                if (k == size)

                {

                    BoolNode->next = NULL;

                    break;

                }

                BoolNode = BoolNode->next;

            }

            addClause(clausep, Sudoku);

            for (k = 1; k <= size; k++)

            {

                for (int t = k + 1; t <= size; t++)

                {

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

                    clausep->head = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->data = -(i \* size2 + j \* size + k);

                    clausep->head->next = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->next->data = -(i \* size2 + j \* size + t);

                    clausep->next = NULL;

                    clausep->head->next->next = NULL;

                    addClause(clausep, Sudoku);

                }

            }

        }

    }

*//下面的循环加入行约束*

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

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

            clausep->head = (charanode \*)malloc(sizeof(charanode));

            clausep->head->next = NULL;

            clausep->next = NULL;

            BoolNode = clausep->head;

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

            {

                BoolNode->data = i \* size2 + k \* size + j + 1;

                BoolNode->next = (charanode \*)malloc(sizeof(charanode));

                if (k == size - 1)

                {

                    BoolNode->next = NULL;

                    break;

                }

                BoolNode = BoolNode->next;

            }

            addClause(clausep, Sudoku);

        }

    }

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

            for (int t = 0; t < size; t++)

                for (k = t + 1; k < size; k++)

                {

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

                    clausep->head = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->data = -(i \* size2 + t \* size + j + 1);

                    clausep->head->next = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->next->data = -(i \* size2 + k \* size + j + 1);

                    clausep->head->next->next = NULL;

                    clausep->next = NULL;

                    addClause(clausep, Sudoku);

                }

        }

    }

*//下面的循环加入列约束*

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

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

            clausep->head = (charanode \*)malloc(sizeof(charanode));

            clausep->head->next = NULL;

            clausep->next = NULL;

            BoolNode = clausep->head;

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

            {

                BoolNode->data = k \* size2 + i \* size + j + 1;

                BoolNode->next = (charanode \*)malloc(sizeof(charanode));

                if (k == size - 1)

                {

                    BoolNode->next = NULL;

                    break;

                }

                BoolNode = BoolNode->next;

            }

            addClause(clausep, Sudoku);

        }

    }

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

            for (int t = 0; t < size; t++)

                for (k = t + 1; k < size; k++)

                {

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

                    clausep->head = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->data = -(t \* size2 + i \* size + j + 1);

                    clausep->head->next = (charanode \*)malloc(sizeof(charanode));

                    clausep->head->next->data = -(k \* size2 + i \* size + j + 1);

                    clausep->head->next->next = NULL;

                    clausep->next = NULL;

                    addClause(clausep, Sudoku);

                }

        }

    }

*//列约束的部分到此结束*

*//下面的循环加入3\*3约束*

    for (k = 0; k < 3; k++)

        for (int t = 0; t < 3; t++)

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

            {

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

                clausep->head = (charanode \*)malloc(sizeof(charanode));

                clausep->head->next = NULL;

                clausep->next = NULL;

                BoolNode = clausep->head;

                for (int i = 3 \* k + 1; i <= 3 \* k + 3; i++)

                {

                    for (int j = 3 \* t + 1; j <= 3 \* t + 3; j++)

                    {

                        BoolNode->data = (i - 1) \* size2 + (j - 1) \* size + count;

                        BoolNode->next = (charanode \*)malloc(sizeof(charanode));

                        if (i == 3 \* k + 3 && j == 3 \* t + 3)

                        {

                            BoolNode->next = NULL;

                            break;

                        }

                        BoolNode = BoolNode->next;

                    }

                }

                addClause(clausep, Sudoku);

            }

*//继续加入3\*3约束*

    for (k = 0; k < 3; k++)

    {

        for (int t = 0; t < 3; t++)

        {

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

            {

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

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

                        for (i = 3 \* k + l; i <= 3 \* k + 3; i++)

                        {

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

                            {

                                if (i == 3 \* k + l && j == 3 \* t + y)

                                    continue;

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

                                clausep->head = (charanode \*)malloc(sizeof(charanode));

                                clausep->head->data = -((3 \* k + l - 1) \* size2 + (3 \* t + y - 1) \* size + count);

                                clausep->head->next = (charanode \*)malloc(sizeof(charanode));

                                clausep->head->next->data = -((i - 1) \* size2 + (j - 1) \* size + count);

                                clausep->next = NULL;

                                clausep->head->next->next = NULL;

                                addClause(clausep, Sudoku);

                            }

                        }

            }

        }

    }

*//下面的循环加入当前所在的数*

    for (i = 0; i < size; i++)

    {

        for (j = 0; j < size; j++)

        {

            if (*sudoku*[i][j] != 0)

            {

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

                clausep->head = (charanode \*)malloc(sizeof(charanode));

                clausep->head->next = NULL;

                clausep->next = NULL;

                clausep->head->data = i \* size2 + j \* size + *sudoku*[i][j];

                addClause(clausep, Sudoku);

            }

        }

    }

*//下面的循环调用DPLL算法来求解sat问题*

    int \*ans;

    boolnum = 729;

    ans = (int \*)malloc(sizeof(int) \* 750);

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

        ans[i] = 1;

    if (betterDPLL(Sudoku, ans))

    {

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

        {

            if (ans[i] == 1)

            {

                int data = i % 9;

                if (data == 0)

                    data = 9;

                int x, y;

                x = (i - data) / 81;

                y = (i - 81 \* x - data) / 9;

*sudoku*[x][y] = data;

            }

        }

        free(ans);

        return 1;

    }

    else

    {

        free(ans);

        deleteCNF(Sudoku);

        return 0;

    }

}

void addClause(cnf \**CNF*, cnf \*&*root*)

{

    if (*CNF* != NULL)

    {

*CNF*->next = *root*;

*root* = *CNF*;

        return;

    }

}

void dighole(int *sudoku*[][9], int *n*)

{

    int count = 0;

    int x, y;

    srand((int)time(0));

    firstdelx = rand() % 9;

    firstdely = rand() % 9;

*sudoku*[firstdelx][firstdely] = 0;

*n*--;

    while (count < *n*)

    {

        x = rand() % 9;

        y = rand() % 9;

        if (check(*sudoku*, x, y))

        {

*sudoku*[x][y] = 0;

            count++;

        }

    }

    return;

}

int check(int *sudoku*[][9], int *x*, int *y*)

{

    int termsuduku[9][9];

    for (int i = 1; i <= 9; i++)

    {

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

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

                termsuduku[ii][j] = *sudoku*[ii][j];

        int key = termsuduku[*x*][*y*], flag = 0;

        if (i != key)

        {

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

            {

                if (j == *x*)

                    continue;

                if (termsuduku[j][*y*] == i)

                {

                    flag = 1;

                    break;

                }

            }

            if (flag)

                continue;

            flag = 0;

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

            {

                if (j == *y*)

                    continue;

                if (termsuduku[*x*][j] == i)

                {

                    flag = 1;

                    break;

                }

            }

            if (flag)

                continue;

            flag = 0;

            int xx = *x* / 3, yy = *y* / 3;

            for (int ii = xx \* 3; ii < xx \* 3 + 3; ii++)

                for (int jj = yy \* 3; jj < yy \* 3 + 3; jj++)

                {

                    if (ii == *x* && jj == *y*)

                        continue;

                    if (termsuduku[ii][jj] == i)

                    {

                        flag = 1;

                        break;

                    }

                }

            if (flag)

            {

                continue;

            }

            termsuduku[*x*][*y*] = i;

            if (solveSudoku(termsuduku))

            {

                return 0;

            }

        }

    }

    return 1;

}

int DPLL(cnf \*&*CNF*, int *value*[])

{

    cnf \*p, \*q, \*r;

    p = *CNF*, q = *CNF*;

    charanode \*boolnode;

    int \*num, max, maxpos; *//num用来记录各个变元出现的次数*

    while (p != NULL)

    {

        while (p && isOneClause(p->head) == 0)

            p = p->next;

        if (p)

        {

            int singlekey = p->head->data;

            if (singlekey > 0)

*value*[singlekey] = 1;

            else

*value*[-singlekey] = 0;

            for (q = *CNF*; q; q = r)

            {

                r = q->next;

                for (boolnode = q->head; boolnode; boolnode = boolnode->next)

                {

                    if (boolnode->data == singlekey)

                    {

                        deleteClause(*CNF*, q);

                        break;

                    }

                    if (boolnode->data == -singlekey)

                    {

                        deletechara(q->head, boolnode);

                        break;

                    }

                }

            }

            if (*CNF* == NULL)

            {

                return 1;

            }

            else if (iswithEmptyClause(*CNF*))

            {

                deleteCNF(*CNF*);

                return 0;

            }

            p = *CNF*;

        }

    }

    num = (int \*)malloc(sizeof(int) \* (2 \* boolnum + 2));

    for (int i = 0; i <= 2 \* boolnum; i++)

        num[i] = 0;

    for (p = *CNF*; p; p = p->next)

    {

        for (boolnode = p->head; boolnode; boolnode = boolnode->next)

            if (boolnode->data > 0)

                num[boolnode->data]++;

            else

                num[boolnum - boolnode->data]++;

    }

    max = 0;

    maxpos = 0;

    int ii;

    int max1 = 0, MaxWord1;

*//统计正变元出现的次数*

    for (ii = 2; ii <= boolnum; ii++)

    {

        if (max1 < num[ii])

        {

            max1 = num[ii];

            MaxWord1 = ii;

        }

    }

    int max2 = 0, MaxWord2;

*//统计负变元出现的次数*

    for (ii = boolnum + 1; ii <= boolnum \* 2; ii++)

    {

        if (max2 < num[ii])

        {

            max2 = num[ii];

            MaxWord2 = -ii + boolnum;

        }

    }

    if (max1 >= max2)

    {

        max = max1;

        maxpos = MaxWord1;

    }

    else

    {

        max = max2;

        maxpos = MaxWord2;

    }

    free(num);

    p = (cnf \*)malloc(sizeof(cnf));

    p->head = (charanode \*)malloc(sizeof(charanode));

    p->head->data = maxpos;

    p->head->next = NULL;

    p->next = NULL;

    q = NULL;

    copyclause(q, *CNF*);

    if (p != NULL)

    {

        p->next = q;

        q = p;

    }

    if (DPLL(q, *value*) == 1)

        return 1;

    deleteCNF(q);

    p = (cnf \*)malloc(sizeof(cnf));

    p->head = (charanode \*)malloc(sizeof(charanode));

    p->head->data = -maxpos;

    p->head->next = NULL;

    p->next = NULL;

    if (p != NULL)

    {

        p->next = *CNF*;

*CNF* = p;

    }

    int i = DPLL(*CNF*, *value*);

    deleteCNF(*CNF*);

    return i;

}