华中科技大学

课程设计报告

题目：基于 SAT 的对角线数独游戏求解程序

课程名称：程序设计综合课程设计

专业班级：本硕博 2301

学　　号：U202315752

姓　　名：陈宇航

指导教师：向文

报告日期：2024 年 9 月 4 日

计算机科学与技术学院

任务书

□ 设计内容

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

□ 设计要求

要求具有如下功能：

（1）输入输出功能：包括程序执行参数的输入，SAT 算例 cnf 文件的读取，执行结果的输出与文件保存等。(15%)

（2）公式解析与验证：读取 cnf 算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

（3）DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。(35%)

（4）时间性能的测量：基于相应的时间处理函数（参考 time.h），记录 DPLL 过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

（5）程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：$[(t-t_o)/t]*100\%$,其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)

（6）SAT 应用：将对角线数独游戏[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%)

口 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

Twodoku： https://en.grandgames.net/multisudoku/twodoku

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

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

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

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

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

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

目录

1引言

1.1 课题背景与意义

计算机科学与技术、大数据与智能科学专业大二学生，在前两个学期已经学习了C语言程序设计、数据结构两门面向编程知识与技术的基础理论课，以及C语言程序设计实验、数据结构实验两门编程实践课程。学生不仅具有较为系统性的C语言、常用数据结构基本知识，而且具有初步的程序设计、数据抽象与建模、问题求解与算法设计的能力，奠定了进行复杂程序设计的知识基础。但两门实验课仍属于对基本编程模型与技术的验证性训练，而“程序设计”综合课程设计正是使大家从简单验证到综合应用，甚至在编程中实现智慧与风格升华的重要实践环节，为后续学习与进行计算机系统编程打下坚实的基础，让综合编程技能成为大家的固有能力与通向未来专业之门的钥匙。

命题逻辑公式的可满足性问题（SAT）是数理逻辑、计算机科学、集成电路设计与验证和人工智能等领域中的核心问题，并且是第一个被证明出来的NP问题。SAT问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法意义重大

1.2 国内外研究现状

自 1960 年以来，SAT 问题一直受到全球的广泛关注。世界各地的研究人员为此付出了大量的努力，提出了多种解决算法。每年在可满足性理论和应用方面的国际会议上都会举办一场 SAT 竞赛，旨在寻找一组最快的SAT 求解器，同时还会详细展示一系列高效求解器的性能。在 2003 年的SAT竞赛中，超过三十种的解决方案针对从成千上万的基准问题中挑选出的SAT问题实例进行同台竞技。与此同时，国内也经常组织 SAT 竞赛和研讨会，这些活动极大地推动了SAT算法的发展。

DPLL 算法被认为是最经典的完备算法，它最早由 Davis 和 Putnam 等人于1960 年提出。后续的完备算法大多是在 DPLL 算法的基础上进行改进的。然而，由于 SAT 问题的最坏情况时间复杂度是指数级别，一度使得许多研究者望而却步。 直到 1971 年，S.A.Cook 证明了SAT问题是 NP 完全问题，这才重新点燃了研究兴趣。从此，问题吸引了越来越多的研究者，引发了一系列高效的SAT算法的诞生，如 MINISAT、

SATO、CHAFF、POSIT 和 GRASP 等。这些算法的发展主要基于对 DPLL 算法的改进，包括新的数据结构、变量决策策略和快速算法实现方案等。

国内也涌现出了一系列高效的 SAT 求解算法，包括梁东敏提出的改进的子句加权 WSAT 算法（1998 年）、金人超和黄文奇提出的并行 Solar 算法（2000 年），及模拟退火算法等。此外，中科院的蔡少伟团队、华中科技大学的吕志鹏团队和西南交通大学的徐扬团队等在SAT问题求解领域也积极贡献了许多突破性工作，并在国际SAT竞赛中表现出色。

尽管SAT问题的理论研究已相对成熟，但随着SAT求解器在各种实际问题领域广泛应用，寻找更有效的算法仍然是一个引人注目且充满挑战的研究方向。这个领域持续发展，为解决复杂问题提供了强有力的工具。

1.3 课程设计的主要研究工作

学习命题逻辑可满足性问题的相关理论知识，并对基于 DPLL 的求解器所采用的关键技术及其算法框架进行了研究与实现，并将其应用于对角线数独的求解。

（1）了解 SAT 问题的研究背景和国内外研究现状，总结 SAT 问题的基本解决方法，解决命题逻辑可满足性问题的一般策略。

（2）读入 cnf 文件，设计数据结构来存储 CNF 范式的文字和子句。

（3）利用代码实现 DPLL 运算器，实现 DPLL 算法的递归过程，并对分支变元的选取策略进行优化。求解出 SAT 问题的答案，生成.res 文件记录，并验证答案。

（4）利用挖洞法将数独游戏转化为 cnf 文件，将 SAT 求解器用于对角线数独自的求解中，实现数独游戏的求解与可玩性的交互。

2 系统需求分析与总体设计

2.1系统需求分析

精心设计问题中变元、文字、子句、公式等有效的物理存储结构，基于 DPLL 过程实现一个高效 SAT 求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。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。

（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，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。优化率的计算公式为：$[(t-t_o)/t]*100\%$, 其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 则为优化DPLL 实现时求解同一算例的执行时间。

（6）SAT 应用：将对角线数独游戏问题转化为 SAT 问题, 并集成到上面的求解器进行数独游戏的求解，游戏可玩，具有一定的、简单的交互性。

2.2系统总体设计

系统设计分为两个模块，分别为SAT问题求解程序模块和蜂窝数独游戏交互程序模块。在初始界面中根据用户的输入进入相应的模块。

(1) SAT求解模块

1. cnf文件读取与存储；
2. 解已指定的cnf;
3. DPLL算法求解SAT问题；
4. DPLL所求解写入文件。

(2)X-Sudoku模块

1. 生成数独；

2. 进行数独填充；

3. 求解当前数独，验证结果正确性；

4. 判断数独的完成度。

系统模块结构图如图2.2-1所示：

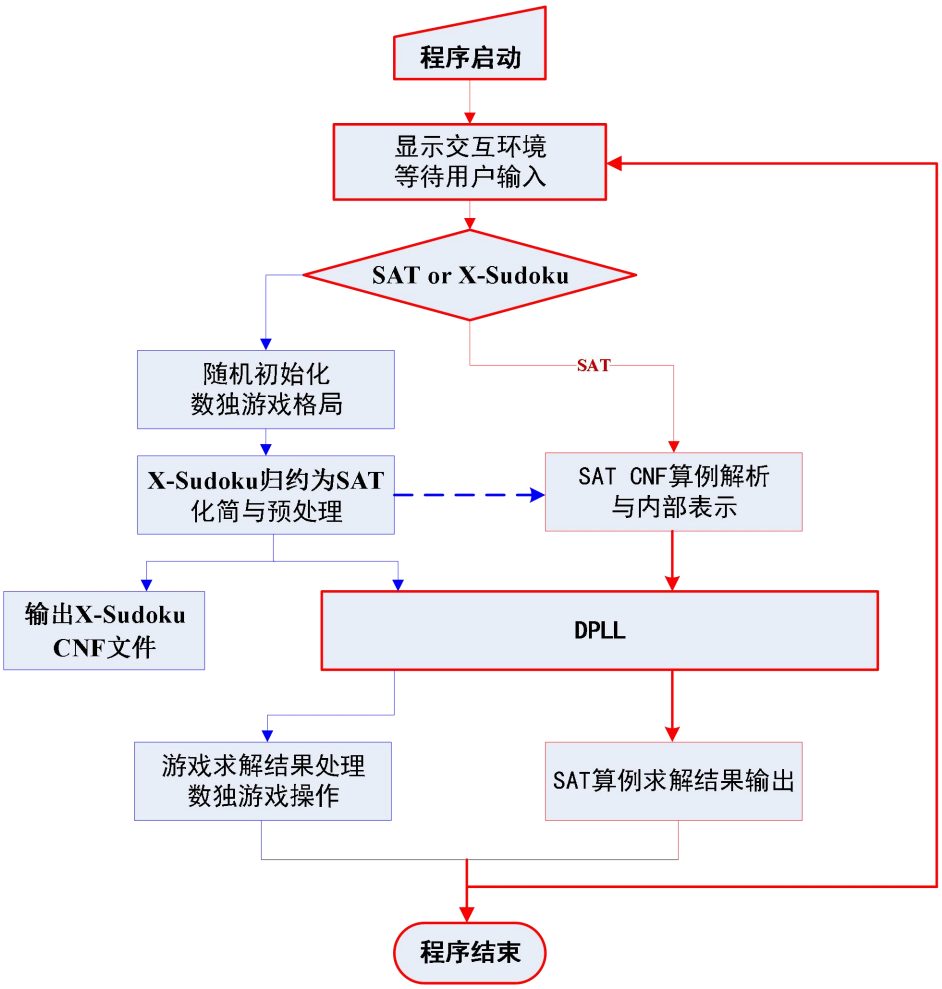

图 2-1 系统模块结构图

2 系统详细设计

3.1有关数据结构的定义

在SAT求解模块中，需要对每个子句和子句包含的文字进行存储。由于不同的cnf文件中子句数目及子句中文字的数目并不相同，所以采用类似十字链表的结构进行存储：以crossCNF为起始，crossNode为节点进行连接。如图3-1所示。

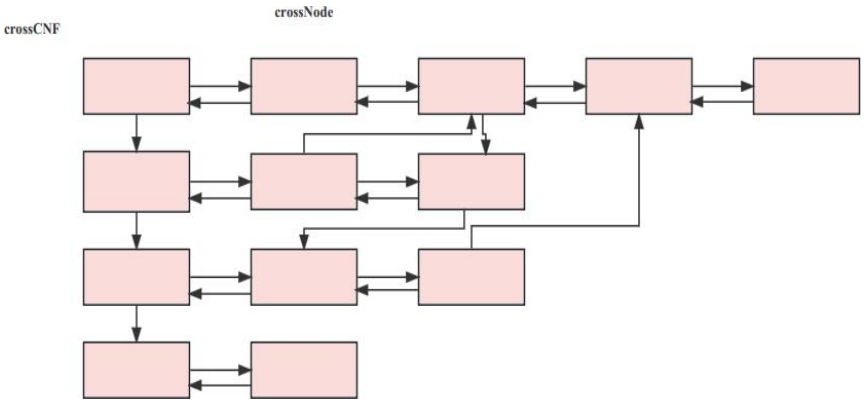

图 3.1 cnf 公式存储结构图

crossNode	*up	*down	*next	*left
节点	指向上一行 相同文字	指向下一行 相同文字	指向 下一个节点	指向 上一个节点

图 3.2 十字链表节点存储结构图

文字节点中，设置有 up,down,next,left 指针。Up指针用来指向上一行相同的文字，down用来指向下一行相同的文字，next指针用来指向下一个节点，left指针用来指向上一个节点，并且设置了一个标记的布尔变量，负责标记这个文字是否被删除，且设置一个整形变量负责记录这个文字当前是真还是假。在子句节点中，设计boolNum来记录包含的变元的个数，设clauseNum来记录子句个数，remainClauseNum来记录剩下的子句个数，定义sum来记录每个子句剩余文字个数。

在X-Sudoku模块中，定义了一个二维数组mymap来记录数独中的信息。

3.2 主要算法设计

3.2.1 cnf文件读取

cnf文件操作时，文件名构造一个crosscnf的类，并利用其中的构造函数，传

入文件的地址，将文件中的信息存储到十字链表中。

（1）读取cnf文件

利用new申请空间，将cnf文件中数据读入十字链表，各子句和文字的标志flag设置为0，同时在问题规模结构体中记录每个文字的出现次数。

（2）输出cnf文件

遍历整个十字链表，并将关键信息在屏幕上输出。

3.2.2 DPLL算法

（1） 单子句规则

在子句集中寻找单子句，假设找到子句集S中单子句为L，L取真值，更新答案数组。再根据单子句规则，屏蔽子句集中所有包含L的句子（即将结点的标志赋值为递归深度，初始深度为1），同时屏蔽所有¬L 的文字，更新子句中文字的个数。执行屏蔽操作时，更新各个变元出现的次数。

（2） 分裂策略

首先基于不同的变元选择策略选取一个文字L。假设L取真值，则将其作为下一次递归中的目标单子句。根据单子句传播策略递归往复，直到无法找到可赋值的变元时函数执行完毕；如果在执行过程中出现空子句，则表示出现冲突。此时回溯至上一级，恢复在上一级中删除的文字和子句，修改子句中文字的个数。接着假设¬L 取真值，再次进入递归，直到求解成功。若所有情况都无法求解成功，则原范式是不可满足的。

（3） 答案保存与检验

求解完成后，显示范式是否满足和求解所用时间，并将问题结构体中数组所保存的答案录入同名但以.res结尾的文件中，文件中同时保存求解结果，求解答案与求解时间。采用“检查要求”包中的verify.exe进行结果的检验。

3.2.3 对角线数独游戏

（1） 主要的思路在于生成满的数独盘，以及将数独存储到cnf文件里面，如何利用挖洞法生成符合要求提示个数的数独盘，并将其在命令行显示出来，还有就是怎么验算所填写的答案是否正确。

（2） cnf范式归约

Cnf约束重要性在于可以利用一个写着cnf范式归约的文件来求解并以此来生成一个数独解。对角线数独游戏是由72个位置组成，分布于9行，每行分别为有9个格。每个位置可以填1~9中任意数字，故文字数为81×9=729个

在cnf文件中布尔变元采用逐行从左到右、从上到下由1至729存储，其中每个位置含有编号为9n+1~9n+9的九个布尔变元。位置中数字要满足所在行，列，盒子，对角线对应的四种约束和格约束，即每条线上的格与格之间应填不同数字，且所填数字是连续的。

（3） 数独格局生成

调用dpll算法去求解事先准备的写好cnf范式的cnf文件，求解出一组解，并将这组解存储到一个res文件里面，再利用一个mymap数组读取这组解，并按照玩家需求，显示相应个数的数独盘。

（4） 数独盘的填写

填写数独的时候需要输入想要填写位置的横坐标和纵坐标以及想要填入的数字，填写的数独内容先存储到mymap数组里面，再通过一个负责展示的函数，实时将命令行中的数独盘更新。

（5） 数独的验证

将mymap数组中填写的内容和res文件中存好的答案进行对比，如果一样的话说明答案正确，不过不能保证只有唯一解的情况。

3.3 程序优化

相较于普通链表的存储结构和朴素DPLL算法，本程序对结构和算法层面做了以下优化。

3.3.1 结构优化

在cnf文件的存储与DPLL算法操作中，采用十字链表存储归约，以一种更直观和紧凑的方式表示布尔公式，通过 up、down、left 和 right 指针，可以轻松地访问和遍历与相同布尔变量、相同子句或相同文字相关的节点，而无需额外的复杂操作；设置flag标记是否删除和记录递归深度。这样可以在子句和文字的删除操作时，通过改变标志使删除对象在后续不进行遍历，从而避免了删除和修改指针带来的时间消耗与复杂操作，也减少了出现不明段错误的几率；在读取cnf文件并存储其中信息的时候，**记录每个文字是正还是负出现的次数，方便在后续进行假设的时候优先假设（如果一个文字是正数出现的次数多，就优先假设正为真值，反之亦然）**且在统计次数时也有技巧，将负数通过公式转换成成一种较大的正数，分开了正负文字的统计，使其他函数处理负数文字更加方便。

在X-Sudoku模块中，用 一个二维数组来存储数独信息，通过三元坐标和绝

对坐标的转换实现存储转换。

3.3.2 算法优化

在X-Sudoku模块的DPLL求解中，借助了倾向性策略来选择首要尝试的布尔变量赋值。该倾向性策略会基于CNF公式中正负文字的频率来决定优先探索哪个变量的赋值，这有助于增加找到解的概率，进而提高了算法的效率。此外，还引入了随机化策略选项，以随机选择要尝试的变量赋值。这种随机化策略在一定程度上有助于避免算法陷入固定的模式，从而提高了算法的优越性。

4 系统实现与测试

4.1系统实现

本实验在Windows11系统下采用Dev C++开发环境进行编译与调试，编程语言为C++。

4.1.1 数据类型声明

```
1.  struct crossNode
2.  //十字链表节点
3.  {
4.
5.      int Bool;//表示自己的布尔变元
6.      int Clause;//表示自己的子句
7.      bool del;
8.      crossNode *up,*down,*right,*left,*next;
9.      crossNode()
10.     {
11.         Bool=0;
12.         Clause=0;
13.         up=NULL;
14.         down=NULL;
15.         right=NULL;
16.         left=NULL;
17.         next=NULL;
18.         del=false;
19.     }
20.
21. };
22.
23.
24. class crossCNF
25. //十字链表 cnf
26. {
27.     public:
28.
29.         crossCNF(const char * const filename);//构造函数
30.         crossCNF(FILE *fp);//构造函数（文件指针版）
31.         ~crossCNF();//析构函数
32.         bool calculate(const char * const filename);//检验
```

```
33.         void print(FILE *);//打印
34.         bool solve(const char* const filename,bool display);//名义解
35.
36.     private:
37.         int boolNum;//布尔变元个数
38.         int clauseNum;//子句个数（原来的）
39.         int remainClauseNum;//删了一些之后剩下的子句个数
40.         int *sum;//每个子句的剩余文字个数
41.         int *tendency;//每个文字的“倾向”
42.         crossNode *bools;//编号从 1 开始
43.         crossNode *clauses;//编号从 1 开始
44.         BOOLVALUE *hypo;//搜索时使用的“假设”
45.         intStack single;//存单子句的栈
46.
47.         void addNode(int,int);//添加结点，生成 cnf 对象时要用
48.         int changeBool(int);//把[-boolNum,-1]U[1,boolNum]翻译成[1,2*boolNum]
    的函数
49.         int innerSolve();//实际解
50.         int believe(int);//“相信”，具体含义请看 solver.cpp
51.         void restore(int L,int startNum,crossNode *Head);//恢复被 believe 函数
    修改过的 cnf
52.
53. };
54.
55.
56. class Hanidoku//表示一套数独布局
57. {
58.     public:
59.         //以下每个方法有数独和对角线两种情况
60.         Hanidoku();//生成空数独
61.         void load(const char* const filename); //从 res 文件中读取
62.         int& cell(int x,int y);//自己的第 x 行第 y 个这个格子，其实很冗余
63.         bool legal(int x,int y);//一个坐标是否合法
64.         void print();//打印
65.         void generate_cnf(const char* const filename);//输出为 cnf
66.         int content_num();//自己有多少个格子被填了
67.         void randomGenerate(int n);// 随机生成数独题目，n 是提示数的数量
68.     private:
69.         int map[9][9];//9*9 的格子，足以存下两种数独
70. };
```

4.1.2 SAT求解模块主要函数

Class crossCNF这个类中负责cnf文件的存储，DPLL求解和验算。

（1） crossCNF(const char * const filename);

构造函数，将构造根据传入的文件名称，读取cnf信息并存储到十字链表。

（2） bool solve(const char* const filename,bool display);

负责调用 DPLL 进行求解存储的cnf范式归约。

（3） void print(FILE *);

将存储的cnf输出到命令行。

（4） bool calculate(const char * const filename);

负责检验所求解的答案是否正确，先将res文件中的答案读取到数组里面，再对每一个子句进行遍历，当遇到第一个为真的文字时候，这个子句为真，进行下一个子句的遍历，当全部遍历完的时候，即所求答案为正确的。

4.1.2 X-Sudoku模块主要函数

class X_ sudoku这个类中的成员变量和成员函数负责主要的数据的存储和功能实现

（1） X_sudoku();

这是一个构造函数，负责初始化装着数独信息的二维数组，都初始化为空。

（2） void load(const char* const filename);

这是一个参数为字符串指针的函数，接收传进来的参数，读取指定 res 文件里面以及填好的答案，并将其存储到二维数组里面。

（3） int& cell(int x,int y);

这个函数负责调用 legal 函数判断所要填写的位置的坐标是否合法，如果合法就返回对应的存储位置的引用，方便直接赋值。

（4） bool legal(int x,int y);

负责判断索要填写的位置的坐标是否超出

（5） void print();

通过一些转换规则来将存储好了数独信息的数组显示到命令行窗口，形成一个 ui 界面，供交互功能使用。

（6） void generate_cnf(const char* const filename);

将数独中已经填写的内容存储到本地的 cnf 文件里面，其实就是在原来的 cnf范式的约束上加上数独盘中已经填写的单子句的规则。

（7） int content_num();

计算数独盘中已经填写的位置的个数，并将其返回。

（8） void randomGenerate(int n);

这是一个负责随机生成数独题目的成员函数，先将负责存储数独的二维数组初始化为空，然后调用 dpll 求解器求解指定的 cnf 文件中的 cnf 范式归约，并将完整的数独盘的信息存储到我们自己定义的一个二维数组里面，最后再通过挖洞法得到最后的数独题。

4.2 系统测试

4.2.1 菜单界面测试

系统包括两个部分：SAT和X-Sudoku。SAT是求解可满足性问题，需要用户输入给定的cnf文件，通过SAT求解器来求解。X_sudoku是求解对角线数独游戏，实现与用户的简单交互。主菜单如图4-1所示：

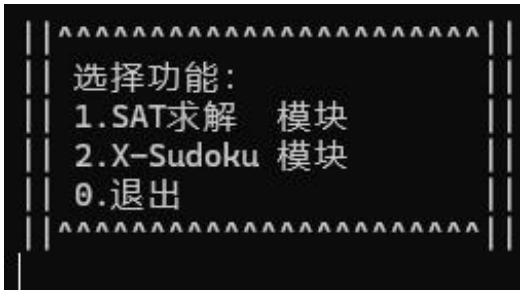

图 4-1 主菜单界面图

输入1进入SAT界面，如图4-2所示：

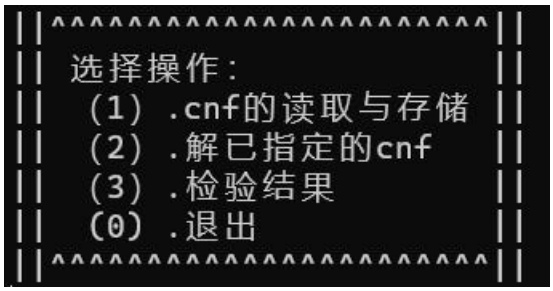

图 4-2 SAT 界面图

输入2进入X_sudoku界面，如图4-3所示：

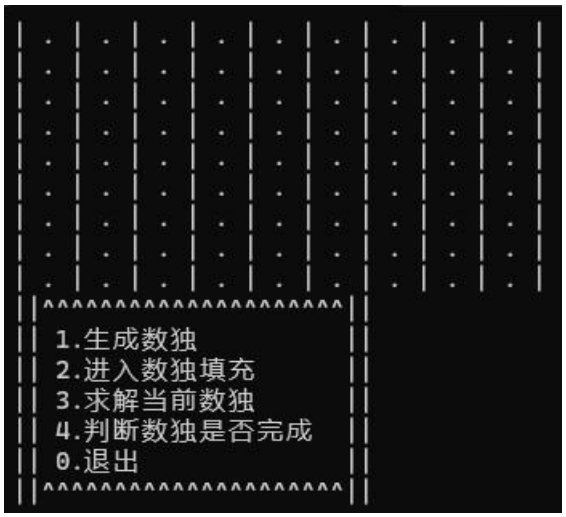

图 4-3 X_sudoku 界面图

4.2.2 SAT求解测试

（1） cnf 文件读取功能测试

此功能可读取 cnf 文件，将数据信息保存在问题规模结构体中。本测试将测度程序是否能成功读入文件并显示子句数和文字数。测试 1 为正确文件名，测试 2 为错误文件名，可测试程序能否对异常输入做出相应的提示。测试结果如下表格及图片所示：

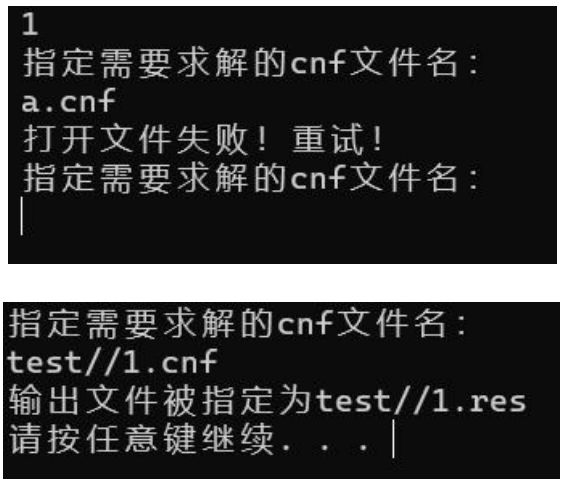

图4-4 cnf 文件读取功能测试

测试输入	目标文件	预期结果	实际效果
1	test\1.cnf	打印子句数和文字数，输出读取成功	如图 4-4
1	a.cnf	输出读取失败	如图 4-4

（2） cnf 文件输出功能测试

此功能可将 cnf 文件中子句的信息打印出来。本测试将测度程序是否能成功打印已读入文件的子句信息，并且是否能对未读入文件时的情况做出相应反应。除未读入文件时的情况外，其他文件名均正确。测试结果如下表格及图片所示：

表 4-2 cnf 文件输出功能测试

测试输入	目标文件	预期结果	实际效果
2	未读取文件	输出尚未读取文件	如图 4-6
2	test\SAT 测试备选算例\sat-20.cnf	打印各子句文字数和文字	部分如图 4-7

图 4-6 文件输出功能测试 1 图

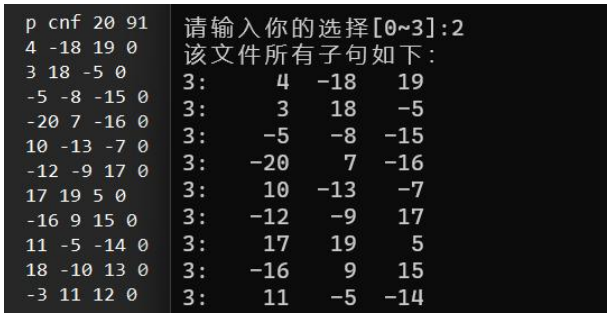

图4-7 文件输出功能测试 2 图

测试 2 中，图片左侧为原 cnf 文件内容，右侧为程序输出的内容。两者是一样的，即 cnf 文件输出功能可以正常使用。

（3） DPLL求解功能测试

此功能用以求解 SAT 问题。在测试中，均采用随机变元选取策略进行 DPLL 求解。测试结果如下表格及图片所示：

表 4-3 DPLL 求解功能测试

测试输入	目标文件	文件规模	实际效果
2	SAT 测试备选算例\满足算例\S\problem11-100.cnf	小型，100 个变元，600 个子句	如图 4-8
2	SAT 测试备选算例\满足算例\M\sud00082.cnf	中型，224 个变元，1762 个子句	如图 4-9
2	test\SAT 测试备选算例\eh-dp04s04.shuffled-1075.cnf	大型，1075 个变元，3152 个子句	如图 4-10

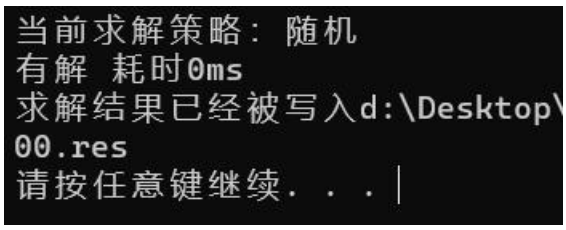

图 4-8 DPLL 求解功能测试

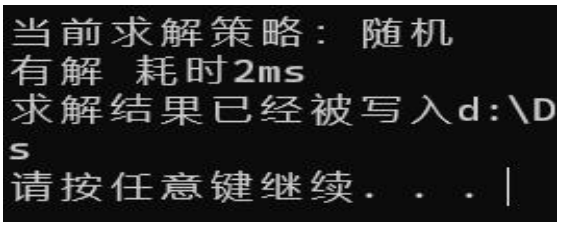

图 4-9 DPLL 求解功能测试

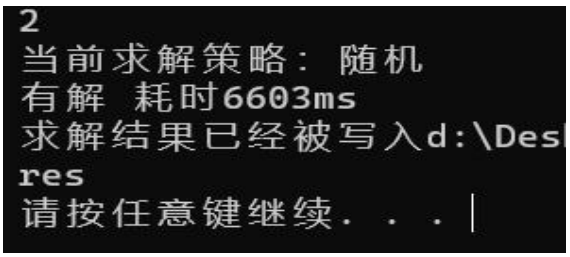

图 4-10 DPLL 求解功能测试

采用 verify.exe 验证，上述求解均正确。

（4） DPLL优化性能测试

测试DPLL算法的优化性能，主要是针对变元选取策略进行优化。我设置了随机和采取

不同的算例寻求减少DPLL的求解时间时，采用的变元选取策略未必相同。

随着算例规模的增大，这种区分会愈发显著。因此，在不改变DPLL算法的递归结构下，并不存在一种普适的算法能够实现绝对的优化效果。即使采用寻找子句集中的首个变元这一策略，在处理某些特定算例时也可能耗费极短的时间。故在测试过程中可能出现负优化的情况，属于正常现象。

算例名称后带有（*）的算例为不满足算例，其他为满足算例。测试结果如下表所示：

表 4-4 DPLL 优化性能测试

算例名	文字数	子句数	优化前时间	优化后时间	优化率
7cnf20_90000_90000_7.shuffled-20.cnf	20	1532	40ms	10ms	75%
st_v25_c100.cnf	25	100	0ms	0ms	0%
problem8-50.cnf	50	300	0ms	0ms	0%
problem3-100.cnf	100	340	0ms	0ms	0%
problem11-100.cnf	100	600	9ms	0ms	100%
tst_v200_c210.cnf	200	210	0ms	0ms	0%
bart17.shuffled-231.cnf	231	1166	0ms	0ms	0%
sud00009.cnf	303	2851	70ms	5953ms	-8404%
sud00012.cnf	232	1901	20ms	10ms	50%
sud00021.cnf	308	2911	380ms	120ms	68%
sud00079.cnf	301	2810	50ms	20ms	60%
sud00082.cnf	224	1762	10ms	10ms	0%
sud00861.cnf	297	2721	10ms	40ms	-300%
eh-dp04s04.shuffled-1075.cnf	1075	3152	699100ms	2010ms	99%
qg7-09.cnf	729	22060	330ms	130ms	61%
tst_v10_c100.cnf (*)	10	100	0ms	0ms	0%
u-problem7-50.cnf (*)	50	100	0ms	0ms	0%
u-problem10-100 (*)	100	200	10ms	0ms	100%

4.3.3 对角线数独游戏测试

对角线数独游戏实现了挖洞法生成数独，操作者自己进行求解，以及求解结果的检验四项功能。

（1） 生成数独测试

本测试将测验程序能否正常生成空数独和有提示数的数独。生成空数独的目的是生成对角线数独的cnf归约，以便输入初始合法格局进行求解。测试结果如表格及图片所示：

下

表 4-5 生成数独功能测试

序号	测试输入	预期结果	实际效果
1	1；1	生成空数独	如图 4-9
2	1；2，25	生成 25 个提示数的数独	如图 4-10
3	1；2，40	生成 40 个提示数的数独	如图 4-11

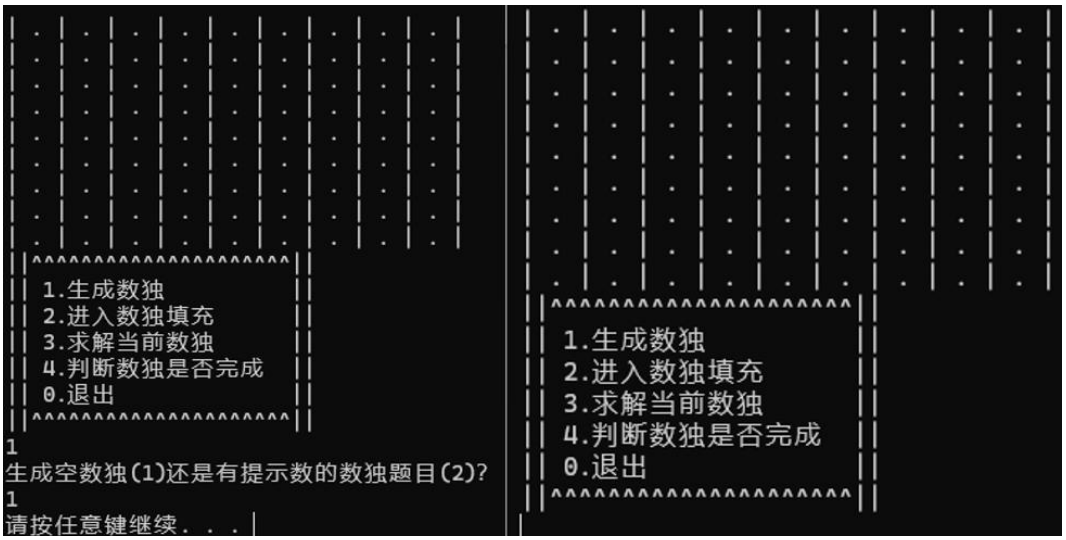

图 4-9 生成数独测试 1 图

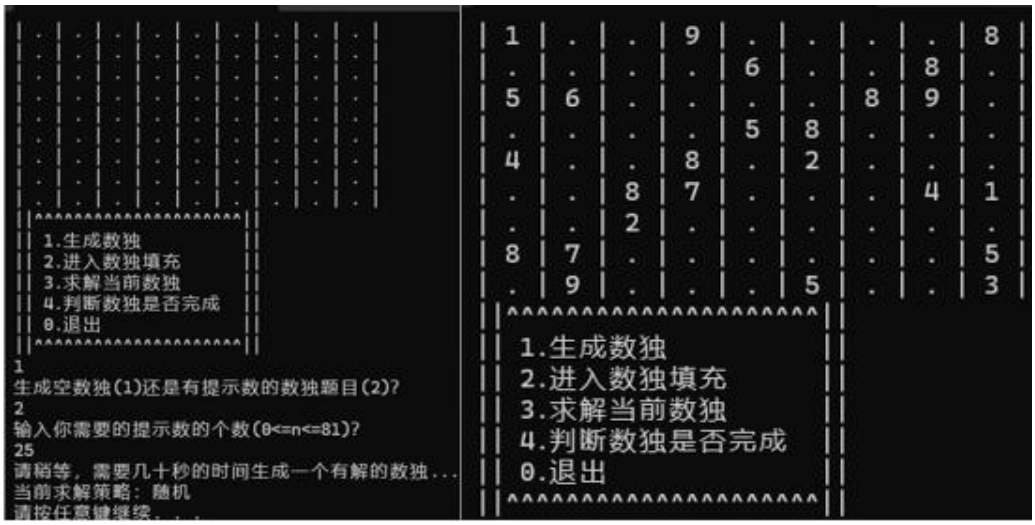

图 4-10 生成数独测试 2 图

图 4-11 生成数独测试 3 图

（2） 自动求解功能测试

本测试将测验程序能否对蜂窝数独进行自动求解。采用上述生成数独测试2的数独格局求解，测试结果如下图所示：

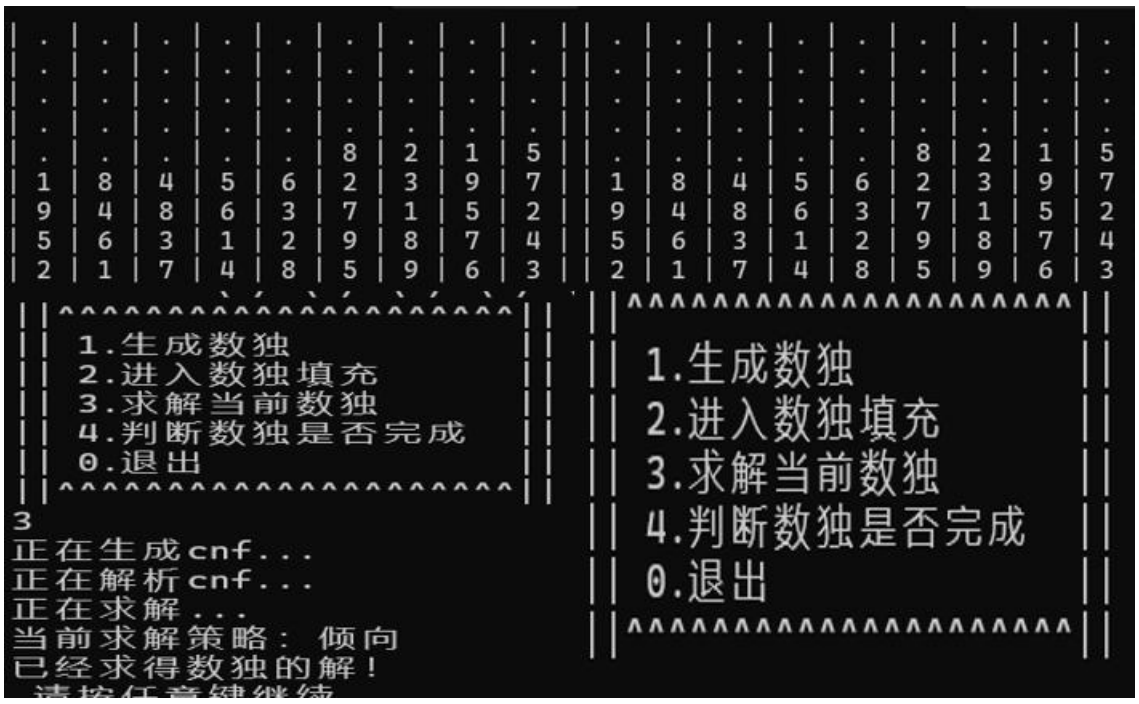

图 4-12 自动求解数独测试图

（3） 用户交互功能测试

此功能使用户通过输入行列数和填充内容在蜂窝数独中填入数字，实现数独的手动求解。测试将测验程序能否由用户输入手动求解对角线数独，以及能否对用户不合理输入进行纠正。

（4） 数独检验测试

此功能用于检验数独的完成性与结果的正确性。本测试主要测验程序对数独是否填完及数独求解是否正确的判断。测试结果如下表格及图片所示：

表 4-7 数独检验功能测试

序号	初始条件	测试输入	预期结果	实际效果
1	数独未完成	4	输出场上未填满	如图 4-13
2	数独已完成，但结果不正确	4	输出数独不合理	如图 4-14
3	数独已完成，且结果正确	4	输出数独已完成	如图 4-15

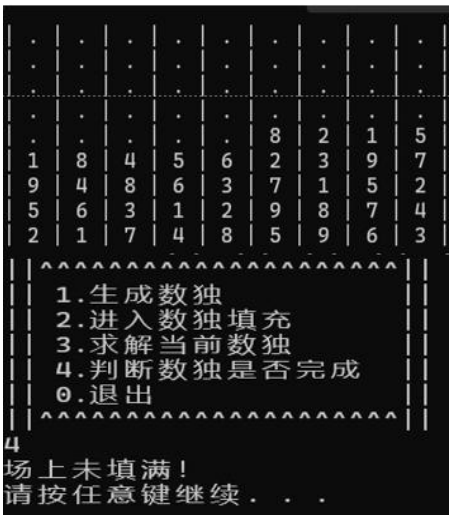

图 4-13 数独检验测试图

图 4-14 数独检验测试图

图 4-15 数独检验测试图

5 总结与展望

5.1 全文总结

整个课程设计流程中，主要工作如下：

（1）研究与理解可满足性问题： 首先，需要深入学习和理解可满足性问题的概念、研究背景以及其在计算机科学中的重要性。通过文献资料的查阅，握可满足性问题的历史发展、国内外研究现状以及我校在这一领域的研究成果。这也包括了对可满足性问题的研究前景有清晰的认识。

（2）构建 CNF 数据结构和 DPLL 算法实现：自行设计和建立用于存储 CNF 公式的数据结构，并实现 DPLL 算法以解决可满足性问题。

（3）模块化算法开发： 将算法的整体框架进行模块化处理，分阶段编写和调试各个模块的代码。确保每个模块的功能正常运作。

（4）整合核心算法和 SAT 问题转化代码： 将编写的模块整合成 DPLL 算法的主体部分，并根据提供的 SAT 测试案例对 DPLL 算法进行调试。同时，编写代码将两类数独问题转化为 SAT 问题的形式。

（5）构建完整工程并设计界面： 组织已编写的模块和代码，构建一个完整的工程。此外，设计用户界面，以便用户能够使用和测试算法。

（6）代码注释与文档编写： 对所有代码进行分类和组织，编写注释以解释每个头文件的作用以及包含的代码块。同时，编写文档，记录课程设计的整个流程、算法细节和界面说明。

以上工作的完善和执行将有助于顺利完成课程设计，同时确保代码的可维护性和可理解性，以及对可满足性问题的深入理解和应用。

5.2 工作展望

在今后的研究中，围绕着如下几个方面开展工作。

（1）多线程优化：我将深入学习多线程编程，以提高 DPLL 算法的时间性能。通过并行化处理，将尝试减少求解时间，使算法更高效。

（2）CDCL 算法探索：我将持续研究冲突驱动子句学习（CDCL）算法，以进一步优化 DPLL 算法的回溯过程。这将有助于提高算法在复杂问题上的解决能力。

（3）处理大型算例：我将致力于解决计算机设备的限制，以克服 DPLL 算法在处理大型算例时的局限性。这包括优化内存使用和算法设计，以处理更大规模的问题。

（4）Qt 界面设计学习：我将学习 Qt 界面设计技巧，以改进算法的用户界面。通过提高菜单和数独游戏的交互性，使用户更容易使用和理解算法。

（5）代码简化和美化：我将进行代码审查，去除冗余设计，以使代码更简洁和易于维护。这将有助于提高代码的可读性和可维护性。

通过这些改进和研究方向，我希望能够进一步提升 DPLL 算法的性能、功能和用户体验，以更好地应对现实世界中的可满足性问题。

6 体会

通过这次课程设计，我获得了许多宝贵的经验和体会：

综合运用编程知识： 这次课程设计要求将之前学到的 C 语言和数据结构知识综合运用到一个复杂的项目中。这锻炼了我将理论知识转化为实际应用的能力。

自学能力： 面对之前未接触过的 DPLL 算法，我积极主动地查阅相关文献和资料，从零开始学习并理解了算法思想。这锻炼了我自主学习和解决问题的能力。

编程乐趣： 虽然整个过程中遇到了各种问题和挑战，但当程序最终成功运行时，我体验到了巨大的成就感和编程的乐趣。这激发了我对编程的兴趣和动力。

编程规范和风格： 在项目中，我注意到了编程规范和良好的代码风格的重要性。注释函数功能和使用预处理指令等良好的编程实践有助于提高代码的可读性和维护性。

问题解决能力： 面对编程过程中的错误和挑战，我学会了耐心地一步一步调试和解决问题，同时也学会了更有效地避免一些常见的编程错误。

算法理解与设计： 通过研究和实现 DPLL 算法，我深化了对算法设计和分析的理解。我也明白了算法在解决复杂问题时的重要性。

工程项目管理： 这次课程设计要求将不同模块整合成一个完整的工程，这锻炼了我项目管理和组织的能力。

总的来说，这次课程设计是一个重要的学习经历，让我在编程和计算机科学领域迈出了一大步。我将继续努力练习实战，不断提高自己的编程技能，并积累更多的实际经验。同时，我也明白了持续学习和解决问题的重要性，这将成为我未来学术和职业生涯中的宝贵财富。

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] 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.

附录

display.cpp

```
 #include<iostream>
#include<stdexcept>
#include<cstdio>
#include<cstdlib>
#include "solver.h"
#include "X_sudoku.h"
#include <ctime>
using namespace std;
extern int random;

int inputOrder(int stt,int end,const char* const text=NULL)
//这个函数用来输入选项，选择范围是[stt,end]
//将此单独设置成函数，可以避免重复的检查输入是否正确
//以返回值的形式输出结果
//会把字符串text输出在屏幕上，以提示用户输入
{

	if(end<stt)throw runtime_error("in inputOrder():end<stt\n");//遇到输入问题时报错。如真报错，肯定是代码问题而非输入问题
	int i,ans=-1;
	/*
	i:循环变量
	ans:待返回的值

	*/
	char s[1000];//输入的一行，应该没有人会一行打1000个字符吧
	bool ok=false;//用于标记输入是否合法。
	if(text)printf("%s",text);//输出提示文字
	while(1)//主循环。如输入合法，返回输入，否则再次循环
	{
		fgets(s,sizeof(s),stdin);//“安全地”从屏幕上读入一行字符串
		ok=true;

		for(i=0;s[i]!='\n';i++)//检查输入是否含有非数字字符
		{
			if(s[i]<'0' || s[i]>'9')
			{
				printf("输入含有非数字字符(%c,%d)，请重试\n",s[i],s[i]);
				ok=false;
				break;//跳出for循环
			}
		}
		s[i]='\0';//标记字符串的末尾
		if(ok==false)continue;
		if(strlen(s)==0)continue;//这是为了应对用户输入一个空行的情况

		ans=atoi(s);// atoi:把字符串形式的整数转化为int
		if(ans<stt || ans>end)// 判断是否超出范围
		{
```

```
                printf("输入超出范围，请重试\n");
                ok=false;
                continue;
            }
            return ans;//如未超出范围，返回
        }



}

void useDPLL()
//此函数用于演示dpll模块
{
        srand(clock());

        char filename[500];//cnf文件名
        char filename2[500];//res文件名
        crossCNF *cnf=NULL;//待定的cnf对象
        int choice;//选项
        FILE *fp=NULL;//待定文件指针
        bool solved=false;//表示当前cnf是否被解过。被解过的cnf对象数据结构会被破坏，不能再解，再解需要重新从文件中读取
        while(1)//主循环
        {
            system("cls");//清屏
            choice=inputOrder(0,4,
            "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
            "|| 选择操作：           ||\n"
            "|| （1）.cnf的读取与存储 || \n"
            "|| （2）.解已指定的cnf   ||\n"
            "|| （3）.检验结果       ||\n"
            "||  (0) .退出           ||\n"
            "||^^^^^^^^^^^^^^^^^^^^^^^^||\n");

            switch(choice)
            {
                case 0:{//退出
                    if(cnf!=NULL)delete cnf;//释放cnf
                    return;
                }
                case 1:{//指定cnf
                    //除了指定cnf文件名之外，这个块还需要重新初始化其他内容
                    if(cnf!=NULL)
                    {
                        printf("cnf已经定义，正在释放旧cnf\n");
                        delete cnf;//释放cnf
                        cnf=NULL;
                        fclose(fp);
                        fp=NULL;
                    }


                    while(fp==NULL)
                    {
```

```
                        printf("指定需要求解的cnf文件名：\n");
                        scanf("%s",filename);

                        fp=fopen(filename,"r");
                        if(fp==NULL)printf("打开文件失败！重试！\n");
                }
                //现在已经打开文件成功
                cnf=new crossCNF(fp);//重新初始化
                solved=false;
                //然后还要重新赋值filename2
                strcpy(filename2,filename);
                int i=0;
                while(filename2[i]!='.')i++;
                filename2[i+1]='r';
                filename2[i+2]='e';
                filename2[i+3]='s';
                filename2[i+4]='\0';

                printf("输出文件被指定为%s\n",filename2);
                break;
        }

        case 2:{//求解已指定的cnf
                if(cnf==NULL)
                {
                        printf("还未指定cnf!\n");
                        break;
                }
                if(solved)
                {
                        printf("这个cnf已经被解过， 数据结构被破坏，请重新初
始化！\n");

                        break;
                }

                cnf->solve(filename2,true);
                solved=true;
                printf("求解结果已经被写入%s\n",filename2);
                break;
        }

        case 3:{//验证并显示
                if(cnf==NULL)
                {
                        printf("还未指定cnf!\n");
                        break;
                }
                if(solved==false)
                {
                        printf("这个cnf还未被求解过！\n");
                        break;
                }
                cnf->calculate(filename2);
                break;
        }
```

```
		}
		system("pause");
	}



}


void useHanidoku()
//此函数用于使用数独模块
{
	Hanidoku hanidoku,*final,*temp;
	int choice;
	hanidoku.generate_cnf("temp\\empty.cnf");



	while(1)//主循环
	{
		system("cls");
		hanidoku.print();
		choice=inputOrder(0,4,
		"||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
		"|| 1.生成数独          ||\n"
		"|| 2.进入数独填充      ||\n"
		"|| 3.求解当前数独      ||\n"
		"|| 4.判断数独是否完成  ||\n"
		"|| 0.退出              ||\n"
		"||^^^^^^^^^^^^^^^^^^^^^^^^||\n");

		switch(choice)
		{
			case 0:{//退出

				return;
			}
			case 1:{//生成数独
				int subchoice=inputOrder(1,2,"生成空数独(1)还是有提示数的数独
题目(2)？\n");
				if(subchoice==1)new (&hanidoku)Hanidoku;//重置数独对象
				else
				{
					int ssc=inputOrder(0,81,"输入你需要的提示数的个数
(0<=n<=81)？\n");
					if(ssc)printf("请稍等，需要几十秒的时间生成一个有解的
数独...\n");
					random=1;
					hanidoku.randomGenerate(ssc);
					random=0;
				}
				break;
			}

			case 2:{//进入操作模式
```

```
                int line,row,content;
                while(1)
                {
                        system("cls");
                        hanidoku.print();
                        printf("输入行数，列数，和填充内容，或者输入0退出\n
（填充0即为移除格中内容）\n");

                        scanf("%d",&line);
                        if(line==0)break;
                        scanf("%d%d",&row,&content);
                        if(hanidoku.legal(line,row) && 0<=content && content<=9)
                        {
                                hanidoku.cell(line,row)=content;
                        }
                        else
                        {
                                printf("数据超出范围，请重试\n");
                                system("pause");
                        }

                }
                break;
        }

        case 3:{//求解场上格局
                printf("正在生成cnf...\n");
                hanidoku.generate_cnf("temp\\tempCNF.cnf");
                printf("正在解析cnf...\n");
                crossCNF tempCNF("temp\\tempCNF.cnf");
                printf("正在求解...\n");
                bool ans=tempCNF.solve("temp\\tempRES.res",false);
                if(ans)
                {
                        printf("已经求得数独的解！\n ");
                        hanidoku.load("temp\\tempRES.res");
                }
                else printf("没有求得数独的解！也许之前的模板是错误的！\n");
                break;
        }

        case 4:{//判断游戏能否结束
                if(hanidoku.content_num()<61)
                {
                        printf("场上未填满！\n");
                        break;
                }
                printf("正在生成cnf...\n");
                hanidoku.generate_cnf("temp\\tempCNF.cnf");
                printf("正在解析cnf...\n");
                crossCNF tempCNF("temp\\tempCNF.cnf");
                printf("正在求解...\n");
                bool ans=tempCNF.solve("temp\\tempRES.res",false);
                if(ans)printf("数独已完成！\n");
                else printf("数独不合理！\n");
```

```
                                        break;
                                }
                        }
                        system("pause");
                }
}



int main()
{
        int choice;
        while(1)
        {
                system("cls");
                choice=inputOrder(0,2,
                        "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
                        "|| 选择功能：            ||\n"
                        "|| 1.SAT求解  模块       ||\n"
                        "|| 2.X-Sudoku 模块       ||\n"
                        "|| 0.退出                ||\n"
                        "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
                        );

                switch(choice)
                {
                        case 0:{
                                return 0;//此时退出就是结束整个程序
                        }
                        case 1:{
                                useDPLL();
                                break;
                        }

                        case 2:{
                                useHanidoku();
                                break;
                        }
                }

        }


}
```

Solver.cpp

```
//这个文件是对 solver.h 中 CNF 求解器部分的实现

#include<cstdlib>
#include<cstdio>
#include<ctime>
#include<new>
#include<cassert>
#include "solver.h"
using namespace std;

int random=1;//如果 random==0，会使用“倾向”求解策略，否则会使用“随机”求解策略
//倾向：在分类讨论布尔变元 L 为真还是为假时，如果一开始的 cnf 中+L 的出现次数比-L 多则先讨论真，否则先讨论假
//随机：在分类讨论布尔变元 L 为真还是为假时，随机先讨论真还是先讨论假


bool crossCNF::calculate(const char * const filename)
//验证 cnf 和 res 文件是否匹配
{

    int *ans=new int[boolNum+1];//res 文件的内容会被写入 ans 再与 cnf 进行比较
    int i,n;

    FILE *fp=fopen(filename,"r");
    if(fp==NULL)
    {
        printf("文件打开失败，不能检验\n");
        delete[] ans;
```

```
        return 0;
    }
    fscanf(fp,"%*s%d%*s",&n);
    if(n==0)
    {
        printf("无解，不可验证\n");
        return 0;
    }

    printf("res=[");
    for(i=0;i<boolNum;i++)
    {
        fscanf(fp,"%d",&n);
        if(n<0)ans[-n]=0;
        else ans[n]=1;
        printf("%d ",n);
    }
    bool ok;
    crossNode *p;
    printf("]\n 检验开始\n");
    for(i=1;i<=clauseNum;i++)//在每个子句中找到第一个为 True 的文字，找到了，这个子句
就算 True
    {
        ok=false;
        p=clauses[i].right;
        while(p)
        {
            n=p->Bool;
            printf("%d ",n);
            if(n<0 && ans[-n]==0    || n>0 && ans[n]==1)
            {
                printf("True");
                ok=true; break;
            }


            p=p->right;
        }
        printf("\n");
        if(!ok)
        {
            printf("clause:False");//如果发现哪个子句为 False，则整个 cnf 也就为 False
            fclose(fp);
```

```
                delete[] ans;
                return false;
            }
        }


        delete[] ans;
        fclose(fp);
        return true;
}

int crossCNF::changeBool(int x)
//把负数翻译成一种比较大的正数的函数，以便其他函数处理负数文字
{
        return (x>0)?x:(2*boolNum+1+x);
}


bool crossCNF::solve(const char* const filename,bool display)
//名义解，主要工作是为 innerSolve 函数创造环境，把结果写成文件什么的
{

        FILE *fp=NULL;
        if(filename)fp=fopen(filename,"w");
        if(random)printf("当前求解策略：随机\n");
        else printf("当前求解策略：倾向\n");


        int start_time=clock();//开始计时
        bool ans=innerSolve();
        int delta_time=clock()-start_time;//结束计时

        if(ans)
        {
            //有解
            if(display)printf("有解  耗时%dms\n",delta_time);
            hypo[0]=TRUE;
            if(fp)
            {
                fprintf(fp,"s 1\nv ");

                for(int i=1;i<=boolNum;i++)
                {
                    if(hypo[i]==TRUE)fprintf(fp,"%d ",i);
```

```
                    else if(hypo[i]==FALSE)fprintf(fp,"-%d ",i);
                    else fprintf(fp,"-%d ",i);
                }

                fprintf(fp,"\nt %d",delta_time);
            }


        }
        else
        {
            //无解
            if(display)printf("无解  耗时%d\n",delta_time);hypo[0]=FALSE;
            if(fp)fprintf(fp,"s 0\n");

        }
        fclose(fp);



        return ans;

}

int crossCNF::believe(int L)
//相信 L==true，这个函数会和 innerSolve 相互调用
{

        //在 hypothesis 中注册结果
        if(L>0)hypo[L]=TRUE;
        else hypo[-L]=FALSE;
        //设置“被删除”栈
        crossNode *DeleteNodesHead=NULL;
        //设置临时指针变量
        crossNode *p,*q;
        int i,target;
        int startNum=remainClauseNum;

        //删去所有含 L 的子句
        for(q=bools[changeBool(L)].down;q;q=q->down)
        {

            if(q->del)continue;
```

```
            target=q->Clause;
            sum[target]++;
            remainClauseNum--;
            for(p=&clauses[target];p;p=p->right)
            {
                if(p->del)continue;
                p->del=true;          // 已经被删除了

                //把 p 加入 DeleteNodes 栈
                p->next=DeleteNodesHead;
                DeleteNodesHead=p;
                sum[target]--;
            }
        }

        //如果导致剩余子句数量为 0 ,就认为这是真解
        if(remainClauseNum==0)return 1;
        //需要删去所有子句中的-L
        for(p=bools[changeBool(-L)].down;p;p=p->down)
        {
            if(p->del)continue;
            p->del=true;
            p->next=DeleteNodesHead;
            DeleteNodesHead=p;
            sum[p->Clause]--;
            // 删除多了怎么办
            //可以在这里检查空子句
            if(sum[p->Clause]==0)
            {
                restore(L,startNum,DeleteNodesHead);
                return 0;
            }
            if(sum[p->Clause]==1)single.push(p->Clause);//如果删出了单子句，就把它加到“单子句”栈中去
        }
        if(innerSolve())return 1;
        restore(L,startNum,DeleteNodesHead);
        return 0;
    }


    void crossCNF::restore(int L,int startNum,crossNode *Head)
    //把 believe 中做出的修改复原
    {
```

```
    hypo[abs(L)]=UNSURE;
    while(Head)//复原一些结点
    {
        Head->del=false;
        sum[Head->Clause]++;
        Head=Head->next;
    }
    remainClauseNum=startNum;
}

int crossCNF::innerSolve()
//实际解，会和 believe 相互调用
{
    int i,L;
    crossNode *p;
    while(!single.empty())//试图从“单子句”栈中获得单子句
    {
        L=single.pop();
        if(!clauses[L].del && sum[L]==1)
        {
            p=clauses[L].right;
            while(p->del)p=p->right;
            return believe(p->Bool);
        }
    }

    //选取“第一行第一个” 文字进行分类讨论 ， 这个策略 一般是最好的

    for(i=1;i<=clauseNum;i++)
    {
        if(sum[i]==0) continue;     // 还剩余的文字为 0
        p=clauses[i].right;
        while(p && p->del)p=p->right;
        break;
    }
    L=abs(p->Bool);       // 转化为正数

    //开始分类讨论，首先要决定先考虑正的还是先考虑负的
    if(random)L*=2*(rand()%2)-1;//随机化选取策略
    else L*=tendency[L];//倾向选取策略

    if(believe(L))return 1;
    return believe(-L);
```

```
}
```

Cnfparser.cpp

```
#include<new>
#include "solver.h"
crossCNF::crossCNF(FILE *fp)//构造函数，可根据 fp 构造CNF 对象
{
	char get;

	//跳过注释：
	while(1)
	{
		get=fgetc(fp);
		if(get=='c')while(fgetc(fp)!='\n'); // 跳过注释
		else break;
	}

	//理论上，现在 get=='p'
	//取元数据
	fscanf(fp,"%*s%d%d",&boolNum,&clauseNum);

	remainClauseNum=clauseNum;


	//开辟数据空间
	clauses = new crossNode[clauseNum+1];
	bools = new crossNode[2*boolNum+2];
	hypo = new BOOLVALUE[boolNum+1];
	sum = new int[clauseNum+1];


	int i,nowClause=1,temp;
	for(i=0;i<=clauseNum;i++)clauses[i].Bool=0;
	for(i=0;i<=boolNum;i++)hypo[i]=UNSURE;
	for(i=0;i<2*boolNum+2;i++)bools[i].Bool=0;
	for(i=0;i<=clauseNum;i++)sum[i]=0;
	//逐个存储
	while(nowClause<=clauseNum)
	{
		fscanf(fp,"%d",&temp);
		if(temp==0)nowClause++;
		else addNode(nowClause,temp);// 在此处添加十字链表结点
```

```
    }
    fclose(fp);



    //初始化“单子句”栈
    for(i=1;i<=clauseNum;i++)if(sum[i]==1)single.push(i);

    //初始化“倾向”
    tendency=new int[boolNum+1];
    int posi,nega;
    crossNode *p;
    for(i=1;i<=boolNum;i++)
    {
        posi=0;
        nega=0;
        // 记录正数出现多还是负数出现多
        for(p=bools[i].down;p;p=p->down)posi++;
        for(p=bools[changeBool(-i)].down;p;p=p->down)nega++;
        if(posi>=nega)tendency[i]=1;
        else tendency[i]=-1;
    }

    return ;
}


crossCNF::crossCNF(const char * const filename)//构造函数，根据文件名构造CNF 对象
{
    FILE *fp=fopen(filename,"r");
    new (this)crossCNF(fp);
    fclose(fp);
}

crossCNF::~crossCNF()//析构函数
{
    crossNode *p,*q;
    int i;
    //逐行删除
    for(i=1;i<=clauseNum;i++)
    {
        p=clauses[i].right;
        while(p)
```

```
            {
                q=p->right;
                delete p;
                p=q;
            }
        }
        delete[] bools,clauses,hypo,sum,tendency;//删去手动分配的数组空间
}


void crossCNF::print(FILE *fp=stdout)
{
        crossNode *p;
        for(int i=1;i<=clauseNum;i++)
        {
            if(clauses[i].del)continue; fprintf(fp,"%d:",i);
            for(p=clauses[i].right;p!=NULL;p=p->right)
            {
                if(p->del)continue;
                fprintf(fp,"%d ",p->Bool);
            }
            fprintf(fp,"\n");
        }
}



void crossCNF::addNode(int Clause,int Bool)
{
        crossNode *p=new crossNode(),*p1,*p2;
        p->Bool=Bool;
        p->Clause=Clause;
        Bool=changeBool(Bool);//正负数坐标转化为物理坐标
        p1=&clauses[Clause];
        p2=&bools[Bool];

        while(p1->right!=NULL && changeBool(p1->right->Bool)<Bool)p1=p1->right;
        while(p2->down!=NULL && p2->down->Clause<Clause)p2=p2->down;
        if(p1->right)p1->right->left=p;
        p->right=p1->right;
        p1->right=p;
        p->left=p1;
```

```
        if(p2->down)p2->down->up=p;
        p->down=p2->down;
        p2->down=p;
        p->up=p2;

        sum[Clause]++;

}
```

X_sudoku.cpp

```
 #include <cstdio>
#include <cstring>
#include <stdexcept>
#include <iostream>
#include <cmath>
#include <ctime>
#include "X_sudoku.h"
#include "solver.h"
using namespace std;

int translate_hive(int i, int j, int k) // 三元坐标转化为绝对坐标
{
   return (i-1)*81+(j-1)*9+k;
}

void head(FILE* fp)
{
   int i, j, k, k1, k2;

   // 每个格子约束
   for (i = 1; i <= 9; i++)
   {
      for (j = 1; j <= 9; j++)
      {
         // 每个格子必须填一个数字
         for (k = 1; k <= 9; k++)
         {
            fprintf(fp, "%d ", translate_hive(i, j, k));
         }
         fprintf(fp, "0\n");

         // 每个格子不能填两个不同的数字
         for (k1 = 1; k1 <= 8; k1++)
         {
            for (k2 = k1 + 1; k2 <= 9; k2++)
            {
               fprintf(fp, "-%d -%d 0\n", translate_hive(i, j, k1), translate_hive(i, j, k2));
            }
         }
      }
```

```
    }

    // 行、列查重约束
    for (k = 1; k <= 9; k++)
    {
        for (i = 1; i <= 9; i++)
        {
            for (int j1 = 1; j1 <= 8; j1++)
            {
                for (int j2 = j1 + 1; j2 <= 9; j2++)
                {
                    // 行查重
                    fprintf(fp, "-%d -%d 0\n", translate_hive(i, j1, k), translate_hive(i, j2, k));
                    // 列查重
                    fprintf(fp, "-%d -%d 0\n", translate_hive(j1, i, k), translate_hive(j2, i, k));
                }
            }
        }
    }

    // 对角线查重约束
    for (k = 1; k <= 9; k++)
    {
        // 主对角线 (从左上到右下)
        for (i = 1; i <= 8; i++)
        {
            for (j = i + 1; j <= 9; j++)
            {
                fprintf(fp, "-%d -%d 0\n", translate_hive(i, i, k), translate_hive(j, j, k));
            }
        }

        // 副对角线 (从右上到左下)
        for (i = 1; i <= 8; i++)
        {
            for (j = i + 1; j <= 9; j++)
            {
                fprintf(fp, "-%d -%d 0\n", translate_hive(i, 10 - i, k), translate_hive(j, 10 - j, k));
            }
        }
    }
}
```

```
void reduction(int &i, int &j, int &k, int n) // 绝对坐标翻译为三元坐标
{
   k = n % 9;
   if (k == 0)
   {
      k = 9;
      n -= 9;
   }
   n /= 9;
   j = n % 9 + 1;
   i = n / 9 + 1;
}

Hanidoku::Hanidoku() // 空数独
{
   for (int i = 0; i < 9; i++) for (int j = 0; j < 9; j++) mymap[i][j] = 0;
}

void Hanidoku::load(const char* const filename) // 从res文件中读取
{
   FILE* fp = fopen(filename, "r");
   while (fgetc(fp) != 'v'); // 一直读取到v为止
   int i, j, k, n, t;
   for (i = 0; i < 9; i++) for (j = 0; j < 9; j++) mymap[i][j] = 0;

   for (t = 0; t < 729; t++)
   {
      fscanf(fp, "%d", &n);
      if (n > 0)
      {
         reduction(i, j, k, n);
         cell(i, j) = k;
      }
   }
   fclose(fp);
   return;
}

int& Hanidoku::cell(int x, int y) // 取第x行第y个格子，xy从1开始
{
```

```
    if (!legal(x, y)) throw runtime_error("Sudoku.cell不合法访问");
    return mymap[x - 1][y - 1];
}

bool Hanidoku::legal(int x, int y) // 一个坐标是否合法
{
    if (x < 1 || x > 9 || y < 1 || y > 9)
        return false;
    return true;
}

void Hanidoku::print()
{
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
            printf("| %c ", (mymap[i][j]) ? (mymap[i][j] + '0') : '.');
        printf("|\n");
    }
}

void Hanidoku::generate_cnf(const char* const filename) // 根据Sudoku生成对应的cnf
{
    FILE* fp = fopen(filename, "w"); // 新建文件
    fprintf(fp, "c generated by cnf_developer.generate_cnf()\np cnf ");

    fprintf(fp, "729 %d\n", 8829 + content_num()); // 729变量，基础8829行
    head(fp);

    for (int i = 1; i <= 9; i++)
    {
        for (int j = 1; j <= 9; j++)
        {
            if (cell(i, j))
            {
                fprintf(fp, "%d 0\n", translate_hive(i, j, cell(i, j)));
            }
        }
    }

    fclose(fp);
}
```

```
int Hanidoku::content_num() // 返回自己已经填了的格子的数量
{
   int n = 0, i, j;
   for (i = 0; i < 9; i++) for (j = 0; j < 9; j++) if (mymap[i][j]) n++;
   return n;
}

void Hanidoku::randomGenerate(int n) // 生成随机数独题目，n为提示数
{
   int i, j, m;
   bool broke;
   for (i = 0; i < 9; i++) for (j = 0; j < 9; j++) mymap[i][j] = 0; // 初始化

   if (n == 0) return;

   generate_cnf("temp\\tempCNF.cnf");
   crossCNF cnf("temp\\tempCNF.cnf");

   cnf.solve("temp\\tempRES.res", false); // 主要耗时

   load("temp\\tempRES.res");

   int del = 0;
   while (81 - del > n)
   {
      // 随机挑选一个元素进行删除
      m = rand() % (81 - del);
      broke = false;
      for (i = 1; i <= 9 && !broke; i++)
      {
         for (j = 1; j <= 9; j++)
         {
            if (cell(i, j))
            {
               m--;
               if (m == 0)
               {
                  cell(i, j) = 0;
                  del++;
                  broke = true;
                  break;
```

```
            }
          }
        }
      }
    }
}
```