



南開大學
Nankai University

计算机学院和网络空间安全学院
软件工程实验课

《数独游戏生成和求解》

程序设计 & 质量检测
& 单元测试

姓名：贾宇航 运开

学号：2013628 2012619

专业：计算机科学与技术 信息安全

2023 年 6 月 20 日

目录

1	质量分析与警告消除	2
1.1	编程规范与警告消除（静态分析）	2
1.2	程序性能剖析（动态分析）	4
1.3	补充	5
2	用例测试	6
2.1	Google Test 安装	6
2.2	测试样例设计	7
2.2.1	对主函数进行功能测试	7
2.2.2	对主函数进行边界测试	8
2.3	测试单个功能函数	9
2.4	测试结果	13
2.5	测试覆盖率统计	13
2.6	BUG 修复	14
3	代码性能改进	15

1 质量分析与警告消除

1.1 编程规范与警告消除（静态分析）

在本次实验中，我使用了 CPPCHECK 来对我编写的 C++ 程序进行编程规范与警告消除的代码剖析。

CPPCHECK 是一个广泛使用的开源静态代码分析工具，专门用于检测 C 和 C++ 程序中的潜在错误、代码风格问题和性能瓶颈。它通过对源代码进行静态分析，识别出可能导致程序错误、内存泄漏、未定义行为等问题的代码片段，并生成相应的报告。

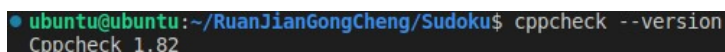
使用 CPPCHECK 对本次结对编程代码进行剖析主要有以下好处：

- 错误检测能力：CPPCHECK 能够检测出代码中的潜在错误，如空指针引用、内存泄漏、未初始化变量等。这些问题可能导致程序崩溃或产生不可预料的结果，因此通过使用 CPPCHECK 可以帮助我发现并修复这些潜在问题，提高代码的稳定性和可靠性。
- 代码风格检查：编写符合规范的代码是保持代码可读性和可维护性的重要因素之一。CPPCHECK 可以检查代码的风格和规范性，包括缩进、命名规范、变量作用域等方面。通过 CPPCHECK 的代码风格检查功能，我可以确保我的代码符合良好的编码规范，提高代码的可读性和可维护性。
- 性能优化提示：优化程序的性能是每个开发人员都关注的重点。CPPCHECK 能够检测出一些潜在的性能瓶颈，例如不必要的拷贝、低效的循环等。通过使用 CPPCHECK 分析我的代码，我可以找出这些可能影响程序性能的问题，并进行相应的优化，提升程序的执行效率。

使用如下指令安装 CPPCHECK：

```
1 sudo apt-get install cppcheck
```

安装后查看版本：



```
● ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --version
Cppcheck 1.82
```

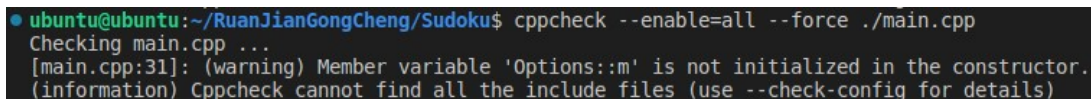
图 1.1: 查看 CPPCHECK 版本

接着使用如下指令对编写的程序进行 profiling：

```
1 cppcheck --enable=all --force ./board.h
```

得到结果如下：

- main.cpp：



```
● ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./main.cpp
Checking main.cpp ...
[main.cpp:31]: (warning) Member variable 'Options::m' is not initialized in the constructor.
(information) Cppcheck cannot find all the include files (use --check-config for details)
```

图 1.2: main.cpp 剖析结果

可以看到，剖析结果提示，在 main.cpp 的 31 行，Option 类的 m 成员变量被定义后未被初始化，检查后发现，该变量在原始版本中作为游戏难度参数的表示，后来为了更符合编程规范，将其改成了 difficulty 这一具有实际意义的变量名，但原来的 m 没有去掉，导致该 warning 产生；

- board.h/sudoku.h:

```
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./board.h
Checking board.h ...
[board.h:9]: (error) Code 'classBoard{' is invalid C code. Use --std or --language to configure the language.
(information) Cppcheck cannot find all the include files (use --check-config for details)
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./sudoku.h
Checking sudoku.h ...
[board.h:9]: (error) Code 'classBoard{' is invalid C code. Use --std or --language to configure the language.
(information) Cppcheck cannot find all the include files (use --check-config for details)
```

图 1.3: board.h/sudoku.h 剖析结果

查阅相关文档后得知, CPPCHECK 试图将.h 文件检查为 C 语言而不是 C++, 因为它认为有效的 C++ 关键字 “namespace” 无效。此处我们可以忽略这个问题。

- board.cpp:

```
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./board.cpp
Checking board.cpp ...
[board.cpp:24]: (style) The function 'generateGame' is never used.
[board.cpp:51]: (style) The function 'hasUniqueSolution' is never used.
[board.cpp:19]: (style) The function 'solve' is never used.
[board.cpp:60]: (style) The function 'toString' is never used.
(information) Cppcheck cannot find all the include files (use --check-config for details)
```

图 1.4: board.cpp 剖析结果

可以看到, 由于我们是使用 CPPCHECK 单独对每一个文件进行剖析, 因此, CPPCHECK 将 board 类的成员函数 generateGame、hasUniqueSolution、solve 以及 toString 等判定为了没有使用的函数, 而对于它们的使用实际上是在 sudoku.cpp 中, 在后面我们将会对整个项目进行整体的剖析, 该报错便会消失;

- sudoku.cpp:

```
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./sudoku.cpp
Checking sudoku.cpp ...
[sudoku.cpp:176] -> [sudoku.cpp:179]: (warning) Variable 'blankMin' is reassigned a value before the old one has been used. 'break;' missing?
[sudoku.cpp:177] -> [sudoku.cpp:180]: (warning) Variable 'blankMax' is reassigned a value before the old one has been used. 'break;' missing?
[sudoku.cpp:179] -> [sudoku.cpp:182]: (warning) Variable 'blankMin' is reassigned a value before the old one has been used. 'break;' missing?
[sudoku.cpp:180] -> [sudoku.cpp:183]: (warning) Variable 'blankMax' is reassigned a value before the old one has been used. 'break;' missing?
[sudoku.cpp:150]: (style) The function 'generateAnswer' is never used.
[sudoku.cpp:113]: (style) The function 'generateFinal' is never used.
(information) Cppcheck cannot find all the include files (use --check-config for details)
```

图 1.5: sudoku.cpp 剖析结果

可以看到, CPPCHECK 报错显示, 存在变量 blankMin 和 blankMax 被连续赋值了两次, 且在连续两次赋值之间该变量并未被使用, 系统提示可能是 “break” 语句缺少导致, 查看报错代码如下:

```
1  .....
2  // 根据难度调整块数
3  switch (difficulty) {
4      case 1:
5          blankMin = 20;
6          blankMax = 30;
7      case 2:
8          blankMin = 30;
```

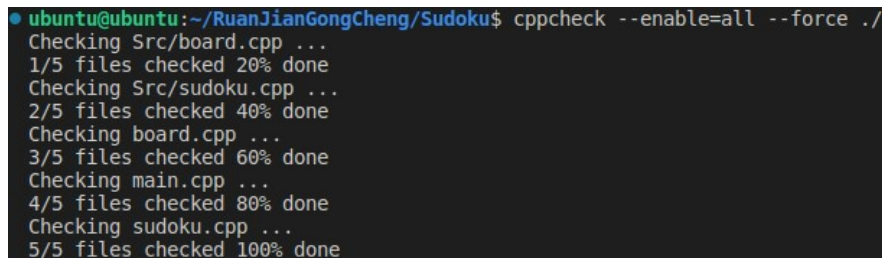
```

9         blankMax = 40;
10     case 3:
11         blankMin = 40;
12         blankMax = 50;
13         break;
14     default:
15         break;
16 }
17 .....

```

果然，前两个 case 分支的结尾均缺失了 break 语句，导致变量 blankMin 和 blankMax 被重复赋值，补上 break 语句后报错消失；

- 所有 warning 修复后，再次测试结果如下：



```

ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ cppcheck --enable=all --force ./
Checking Src/board.cpp ...
1/5 files checked 20% done
Checking Src/sudoku.cpp ...
2/5 files checked 40% done
Checking board.cpp ...
3/5 files checked 60% done
Checking main.cpp ...
4/5 files checked 80% done
Checking sudoku.cpp ...
5/5 files checked 100% done

```

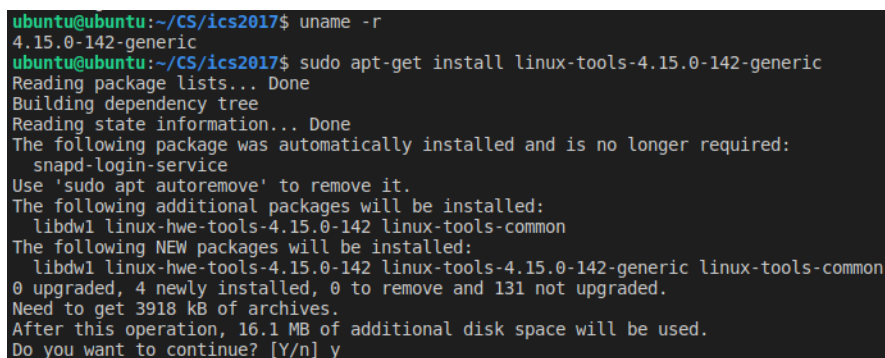
图 1.6: 修复后剖析结果

测试显示不再有 warning。

1.2 程序性能剖析（动态分析）

GNU/Linux 内核提供了性能剖析工具 perf, 可以方便地收集程序运行的信息。首先使用如下语句查看操作系统内核版本：uname -r（发现内核版本为 4.15.0-142-generic）。

使用如下命令安装对应版本的 perf: sudo apt-get install linux-tools-4.15.0-142-generic:



```

ubuntu@ubuntu:~/CS/ics2017$ uname -r
4.15.0-142-generic
ubuntu@ubuntu:~/CS/ics2017$ sudo apt-get install linux-tools-4.15.0-142-generic
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  snapd-login-service
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libdw1 linux-hwe-tools-4.15.0-142 linux-tools-common
The following NEW packages will be installed:
  libdw1 linux-hwe-tools-4.15.0-142 linux-tools-4.15.0-142-generic linux-tools-common
0 upgraded, 4 newly installed, 0 to remove and 131 not upgraded.
Need to get 3918 kB of archives.
After this operation, 16.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

图 1.7: 查看内核版本并安装对应的 perf

之后使用 perf 对程序进行性能剖析，得到的结果如下：

- 剖析生成终局过程：sudo perf record sudoku -c10000:

Overhead	Command	Shared Object	Symbol
11.64%	sudoku	sudoku	[.] __gnu_cxx::to_xstring<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>,
9.43%	sudoku	libc-2.27.so	[.] _IO_vfprintf
5.66%	sudoku	sudoku	[.] Board::toString[abi:cxx11]
5.03%	sudoku	libc-2.27.so	[.] _IO_default_xsputn
4.72%	sudoku	libstdc++.so.6.0.25	[.] std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append
4.72%	sudoku	sudoku	[.] std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_construct<char*>
4.40%	sudoku	libc-2.27.so	[.] __strchrnul_avx2
4.09%	sudoku	libc-2.27.so	[.] __vsnprintf
3.46%	sudoku	libc-2.27.so	[.] _IO_old_init

图 1.8: 针对生成终局进行 perf

可以看到，在生成终局的过程中，程序的大部分性能开销都花费到了二维数组的 alloca 操作、IO 操作、以及数组向 String 转换的操作上了，这部分属于必要开销，无法优化。

- 剖析生成游戏过程：sudo perf record ./sudoku -n10000 -m3 -r50 55 -u:

Overhead	Command	Shared Object	Symbol
47.26%	sudoku	sudoku	[.] Board::isValid
14.88%	sudoku	sudoku	[.] Board::solveDFS
10.23%	sudoku	sudoku	[.] std::subtract_with_carry_engine<unsigned long, 48ul, 5ul, 12ul>::operator()
9.01%	sudoku	sudoku	[.] Board::generateGame
3.47%	sudoku	[kernel.kallsyms]	[k] __softirqentry_text_start
2.50%	sudoku	libc-2.27.so	[.] random
1.96%	sudoku	libc-2.27.so	[.] random_r
1.50%	sudoku	libc-2.27.so	[.] __memcpy_avx_unaligned_erms
1.27%	sudoku	libc-2.27.so	[.] rand
1.07%	sudoku	sudoku	[.] generateGame

图 1.9: 针对生成终局进行 perf

生成游戏时，由于有最终解是否唯一的判断，因而程序有近一半的性能开销都花到了生成游戏解是否唯一的判断上，这部分我们采用的检测算法是做深度遍历，如果有两组解均成立，说明生成的游戏解不唯一，此方法复杂度较高，后续该部分进行了优化。

- 剖析求解游戏过程：sudo perf record ./sudoku -n10000 -m3 -r50 55 -u:

Overhead	Command	Shared Object	Symbol
33.68%	sudoku	sudoku	[.] Board::isValid
16.03%	sudoku	sudoku	[.] Board::solveDFS
4.61%	sudoku	libc-2.27.so	[.] _IO_vfprintf
4.50%	sudoku	[kernel.kallsyms]	[k] __softirqentry_text_start
4.04%	sudoku	sudoku	[.] __gnu_cxx::to_xstring<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>,
2.77%	sudoku	sudoku	[.] std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_construct<char*>
2.19%	sudoku	libstdc++.so.6.0.25	[.] std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append
1.73%	sudoku	sudoku	[.] std::replace__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>,
1.61%	sudoku	libc-2.27.so	[.] __vsnprintf
1.27%	sudoku	libc-2.27.so	[.] _IO_default_xsputn

图 1.10: 针对生成终局进行 perf

求解游戏时与生成游戏相似，大部分性能开销都花到了求出的解是否正确的判断以及做深度优先遍历的过程中。

1.3 补充

考虑到 CPPCHECK 检查到的 warning 非常有限，因此我们使用 clang 对审查过的代码再进行一次检查，编译指令如下：

```
1 clang-tidy ./* --checks=*,readability-*,bugprone-*,clang-diagnostic-*
```

得到如下报错信息：

```
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/board.cpp:32:16: warning: implicit conversion 'int' -> bool [readability-implicit-bool-conversion]
while (num-- ) {
      ^~~~~~
      ( ) != 0
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/board.cpp:43:12: warning: implicit conversion 'int' -> bool [readability-implicit-bool-conversion]
while (num-- ) {
      ^~~~~~
      ( ) != 0
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/board.cpp:113:15: warning: do not use 'else' after 'return' [readability-else-after-return]
} else if (count == temp + 1) {
      ^~~~~~
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/board.h:28:10: warning: function 'Board::generateGame' has a definition with different parameter names
stent-declaration-parameter-name]
void generateGame(int blockNum);
      ^~~~~~
      blankNum
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/board.cpp:25:13: note: the definition seen here
void Board::generateGame(int blankNum) {
      ^~~~~~
```

```
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/main.cpp:83:50: warning: implicit conversion 'int' -> bool [readability-implicit-bool-conversion]
if ((options.mode == Mode::GenerateGame) && !options.count) {
      ^~~~~~
      ( ) == 0
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/main.cpp:87:49: warning: implicit conversion 'std::__cxx11::basic_string<char, std::char_traits<char>, >::size_type' (aka 'unsigned long') -> bool [readability-implicit-bool-conversion]
if ((options.mode == Mode::SolveSudoku) && !options.filename.length()) {
      ^~~~~~
      ( ) == 0u
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/sudoku.cpp:38:16: warning: implicit conversion 'int' -> bool [readability-implicit-bool-conversion]
while (k-- && getline(file, temp))
      ^~~~~~
      (( ) != 0)
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/sudoku.cpp:38:43: warning: statement should be inside braces [readability-braces-around-statements]
while (k-- && getline(file, temp))
      ^~~~~~
      {
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/sudoku.cpp:71:38: warning: statement should be inside braces [readability-braces-around-statements]
while (getline(file, temp, '\n'))
      ^~~~~~
      {
/home/ubuntu/RuanJianGongCheng/Sudoku/clang/sudoku.cpp:156:38: warning: statement should be inside braces [readability-braces-around-statements]
for (int i = 0; i < gameNum; ++i)
      ^~~~~~
      {
```

依次分析并修改如下：

- while(statement) 中的表达式 statement 应当为 bool 类型，我们在编程时往往习惯用 int 类型来替代，这样虽然编译器会做隐式类型转换将 int 类型转换为 bool 类型，但是这种写法并不符合编程规范；
- 我们在分支语句中常常使用 return 语句来快速结束此分支并返回函数，但是此做法不符合编程规范，我们可以在所有分支结束后再根据计算的结果进行返回；
- 函数声明和实现时形参的变量名不一致，这可能会导致错误，同时也不符合编程规范；
- 当循环体或分支语句只有一条语句时，常常会省略，而是直接写此语句，这样容易导致错误出现；

2 用例测试

2.1 Google Test 安装

在本次实验中，我选择使用 Google Test 来进行用例测试，以评估和验证我编写的 C++ 代码的正确性和可靠性。Google Test 是一个广泛应用的开源单元测试框架，它提供了丰富的功能和工具，用于编写、组织和执行测试用例，以确保程序在各种情况下都能按预期工作。

我选择使用 Google Test 进行用例测试的主要原因如下：

- 自动化测试：Google Test 允许我编写自动化测试用例，自动运行这些测试用例，并生成相应的

测试结果报告。相比手动测试，自动化测试可以提高测试效率，并且可以在开发过程中持续运行测试，及早发现和修复代码中的问题。

- 组织和管理测试用例：Google Test 提供了一套灵活的框架和语法，可以帮助我组织和管理大量的测试用例。我可以将测试用例按功能或模块进行分组，并使用 Google Test 的断言宏来编写测试代码，验证程序在不同情况下的行为是否符合预期。
- 断言和测试覆盖率：Google Test 提供了丰富的断言宏，可以方便地对程序的输出结果进行验证。通过编写断言语句，我可以检查函数的返回值、异常情况、预期输出等，并确保代码的逻辑正确性。此外，Google Test 还提供测试覆盖率分析工具，可以帮助我评估测试覆盖的程度，确保尽可能覆盖代码的各个分支和路径。

首先使用如下语句序列完成 GoogleTest 的安装：

```
1 sudo apt update
2 sudo apt install cmake libgtest-dev
3 cd /usr/src/gtest
4 sudo cmake CMakeLists.txt
5 sudo make
6 sudo cp *.a /usr/lib
```

2.2 测试样例设计

2.2.1 对主函数进行功能测试

按照生成终局、生成游戏（是否唯一解）、生成解答的顺序依次对整个程序的执行情况进行测试：

编号	测试用例	正确结果	说明
1	./sudoku -c10000	输出正确提示，产生正确的 final.txt 文件	测试程序对生成终局请求的响应
2	./sudoku -n100 -m3	输出正确提示，产生正确的 game.txt 文件	测试程序对生成指定难度游戏请求的响应
3	./sudoku -n100 -r20 45 -u	输出正确提示，产生正确的 game.txt 文件	测试程序对生成指定挖空数目、具有唯一解游戏请求的响应
4	./sudoku -s ./game.txt	输出正确提示，产生正确的 result.txt 文件	测试程序对求解游戏请求的响应

表 1: 测试用例表（一）

针对上述测试样例编写测试文件如下（部分，完整版见仓库 gTest.cpp）：

```
1 TEST(MainTest, GenerateFinalMode) {
2     char* argv[] = { (char*)"./sudoku", (char*)" -c10000" }; //生成 10000 个终局;
```



```

3     int argc = sizeof(argv) / sizeof(argv[0]);
4
5     // 运行主函数
6     ::testing::internal::CaptureStdout(); // 捕获标准输出
7     MainTest(argc, argv);
8     std::string output = ::testing::internal::GetCapturedStdout(); // 获取标准输出
9     std::ifstream file("./final.txt");
10    bool fileExists = file.good();
11    file.close();
12
13    // 进行断言，验证结果是否符合预期
14    EXPECT_TRUE(output.find("success.") != std::string::npos);
15    EXPECT_TRUE(fileExists);
16 }

```

2.2.2 对主函数进行边界测试

针对主函数，对输入的参数进行边界测试，主要检测程序对非法输入参数的检测：

编号	测试用例	正确结果	说明
5	./sudoku -c0	输出对应报错提示	生成 0 个终局
6	./sudoku -n100 -m0	输出对应报错提示	生成 100 局难度为 0 的游戏
7	./sudoku -n100 -r56 57 -u	输出对应报错提示	生成 100 局挖空在 56 57 之间的游戏
8	./sudoku -s ./none.txt	输出对应报错提示	解决不存在的游戏

表 2: 测试用例表（二）

针对上述测试样例编写测试文件如下（部分，完整版见仓库 gTest.cpp）：

```

1  TEST(MainTest, generateAnswerBoarderTest) {
2      char* argv[] = { (char*)"./sudoku", (char*)"-s ./none.txt" }; //解决不存在的游戏;
3      int argc = sizeof(argv) / sizeof(argv[0]);
4
5      ::testing::internal::CaptureStdout();
6      MainTest(argc, argv);
7      std::string output = ::testing::internal::GetCapturedStdout();
8      std::ifstream file("./answer.txt");
9      bool fileExists = file.good();
10     file.close();
11
12     EXPECT_TRUE(output.find("failed.") != std::string::npos);

```

```

13     EXPECT_FALSE(fileExists);
14 }

```

2.3 测试单个功能函数

编号	测试用例	正确结果	说明
9	board.generateFinal(10);	输出 10 个合法的终局	测试 board 的成员函数 generateFinal
10	board.generateGame(40);	输出满足数独规则的要求同时具有 40 个挖空的棋盘	测试 board 的成员函数 generateGame
11	<pre> int input[9][9] = { {5, 3, 0, 0, 7, 0, 0, 0, 0}, {6, 0, 0, 1, 9, 5, 0, 0, 0}, {0, 9, 8, 0, 0, 0, 0, 6, 0}, {8, 0, 0, 0, 6, 0, 0, 0, 3}, {4, 0, 0, 8, 0, 3, 0, 0, 1}, {7, 0, 0, 0, 2, 0, 0, 0, 6}, {0, 6, 0, 0, 0, 0, 2, 8, 0}, {0, 0, 0, 4, 1, 9, 0, 0, 5}, {0, 0, 0, 0, 8, 0, 0, 7, 9} }; Board board(input); board.solveDFS(0, 0, count); </pre>	<pre> EXPECT_EQ(board.toString(), "534678912" "672195348" "198342567" "859761423" "426853791" "713924856" "961537284" "287419635" "345286179"); </pre>	测试 board 的成员函数 solveDFS

表 3: 测试用例表 (三)

针对上述测试样例编写测试文件如下 (部分, 完整版见仓库 gTest.cpp):

```

1  TEST(BoardTest, GenerateGame) {
2      Board board;
3      board.generateGame(40); // 生成一个有 40 个空格的数独游戏
4
5      //进一步检查生成的游戏是否满足数独规则的要求
6      std::string game = board.toString();
7
8      // 检查生成的游戏是否满足数独规则的要求
9      for (int i = 0; i < Board::MaxRow; i++) {
10         for (int j = 0; j < Board::MaxCol; j++) {
11             int num = board.grid[i][j];
12             if (num != 0) {
13                 // 检查同行是否有重复数字
14                 for (int col = 0; col < Board::MaxCol; col++) {

```

```

15         if (col != j && board.grid[i][col] == num) {
16             FAIL() << "Duplicate number " << num
17             << " in the same row " << i;
18         }
19     }
20     // 检查同列是否有重复数字
21     .....//与同行检查类似;
22
23     // 检查同宫格是否有重复数字
24     int startRow = i / 3 * 3;
25     int startCol = j / 3 * 3;
26     for (int row = startRow; row < startRow + 3; row++) {
27         for (int col = startCol; col < startCol + 3; col++) {
28             if (row != i && col != j && board.grid[row][col] == num) {
29                 FAIL() << "Duplicate number " << num
30                 << " in the same block (" << startRow / 3 << ", " << startCol
31             }
32         }
33     }
34     .....
35     // 检查生成的游戏空格数目:
36     int blankCount = 0;
37     for (int i = 0; i < Board::MaxRow; i++) {
38         for (int j = 0; j < Board::MaxCol; j++) {
39             if (board.grid[i][j] == 0) blankCount++;
40         }
41     }
42     EXPECT_EQ(blankCount, 40); // 检查生成的游戏中的空格数量是否正确
43 }
44
45 TEST(BoardTest, SolveDFS) {
46     int input[9][9] = {
47         {5, 3, 0, 0, 7, 0, 0, 0, 0},
48         {6, 0, 0, 1, 9, 5, 0, 0, 0},
49         {0, 9, 8, 0, 0, 0, 0, 6, 0},
50         {8, 0, 0, 0, 6, 0, 0, 0, 3},
51         {4, 0, 0, 8, 0, 3, 0, 0, 1},
52         {7, 0, 0, 0, 2, 0, 0, 0, 6},
53         {0, 6, 0, 0, 0, 0, 2, 8, 0},
54         {0, 0, 0, 4, 1, 9, 0, 0, 5},
55         {0, 0, 0, 0, 8, 0, 0, 7, 9}};
56

```

```

57     Board board(input);
58     int count = 0;
59     board.solveDFS(0, 0, count);
60
61     //检查解决数独的结果是否正确
62     EXPECT_EQ(count, 1); // 只能有一个解
63     EXPECT_EQ(board.toString(), "534678912\n"
64                                   "672195348\n"
65                                   "198342567\n"
66                                   "859761423\n"
67                                   "426853791\n"
68                                   "713924856\n"
69                                   "961537284\n"
70                                   "287419635\n"
71                                   "345286179\n");
72 }

```

编号	测试用例	正确结果	说明
12	<pre>int array[9][9] = { {5, 3, 0, 0, 7, 0, 0, 0, 0}, {6, 0, 0, 1, 9, 5, 0, 0, 0}, {0, 9, 8, 0, 0, 0, 0, 6, 0}, {8, 0, 0, 0, 6, 0, 0, 0, 3}, {4, 0, 0, 8, 0, 3, 0, 0, 1}, {7, 0, 0, 0, 2, 0, 0, 0, 6}, {0, 6, 0, 0, 0, 0, 2, 8, 0}, {0, 0, 0, 4, 1, 9, 0, 0, 5}, {0, 0, 0, 0, 8, 0, 0, 7, 9} }; Board board(array);</pre>	<pre>bool hasUnique = board.hasUniqueSolution(); EXPECT_TRUE(hasUnique);</pre>	测试一个具有唯一解的游戏
13	<pre>int array[9][9] = { {5, 3, 0, 0, 7, 0, 0, 0, 0}, {6, 0, 0, 1, 9, 5, 0, 0, 0}, {0, 9, 8, 0, 0, 0, 0, 6, 0}, {8, 0, 0, 0, 6, 0, 0, 0, 3}, {4, 0, 0, 8, 0, 3, 0, 0, 1}, {7, 0, 0, 0, 2, 0, 0, 0, 6}, {0, 6, 0, 0, 0, 0, 2, 8, 0}, {0, 0, 0, 4, 1, 9, 0, 0, 5}, {0, 0, 0, 0, 8, 0, 0, 0, 0} }; Board board(array);</pre>	<pre>bool hasUnique = board.hasUniqueSolution(); EXPECT_FALSE(hasUnique);</pre>	测试一个具有多个解的游戏

表 4: 测试用例表 (四)

针对上述测试样例编写测试文件如下 (部分, 完整版见仓库 gTest.cpp):

```
1 TEST(BoardTest, HasUniqueSolution) {
```

```
2      // 创建一个数独游戏，该游戏有唯一解
3      int array[9][9] = {
4          {5, 3, 0, 0, 7, 0, 0, 0, 0},
5          {6, 0, 0, 1, 9, 5, 0, 0, 0},
6          {0, 9, 8, 0, 0, 0, 0, 6, 0},
7          {8, 0, 0, 0, 6, 0, 0, 0, 3},
8          {4, 0, 0, 8, 0, 3, 0, 0, 1},
9          {7, 0, 0, 0, 2, 0, 0, 0, 6},
10         {0, 6, 0, 0, 0, 0, 2, 8, 0},
11         {0, 0, 0, 4, 1, 9, 0, 0, 5},
12         {0, 0, 0, 0, 8, 0, 0, 7, 9}
13     };
14     Board board(array);
15
16     // 检查数独游戏是否有唯一解
17     bool hasUnique = board.hasUniqueSolution();
18
19     EXPECT_TRUE(hasUnique);
20
21     // 创建一个数独游戏，该游戏有多个解
22     int array2[9][9] = {
23         {5, 3, 0, 0, 7, 0, 0, 0, 0},
24         {6, 0, 0, 1, 9, 5, 0, 0, 0},
25         {0, 9, 8, 0, 0, 0, 0, 6, 0},
26         {8, 0, 0, 0, 6, 0, 0, 0, 3},
27         {4, 0, 0, 8, 0, 3, 0, 0, 1},
28         {7, 0, 0, 0, 2, 0, 0, 0, 6},
29         {0, 6, 0, 0, 0, 0, 2, 8, 0},
30         {0, 0, 0, 4, 1, 9, 0, 0, 5},
31         {0, 0, 0, 0, 8, 0, 0, 0, 0}
32     };
33     Board board2(array2);
34
35     // 检查数独游戏是否有唯一解
36     hasUnique = board2.hasUniqueSolution();
37
38     EXPECT_FALSE(hasUnique);
39 }
40
```

2.4 测试结果

编译 gTest.cpp 生成可执行文件 test_executable，并使用 test_executable 对待测试文件进行单元测试，得到如下结果：

```

ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ g++ -o test_executable gTest.cpp -lgtest -lgtest_main -pthread
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ ./test_executable
[=====] Running 12 tests from 2 test cases.
[-----] Global test environment set-up.
[-----] 8 tests from MainTest
[ RUN      ] MainTest.GenerateFinalMode
[ OK       ] MainTest.GenerateFinalMode (123 ms)
[ RUN      ] MainTest.GenerateGameMode1
[ OK       ] MainTest.GenerateGameMode1 (268 ms)
[ RUN      ] MainTest.GenerateGameMode2
[ OK       ] MainTest.GenerateGameMode2 (534 ms)
[ RUN      ] MainTest.generateAnswerMode
[ OK       ] MainTest.generateAnswerMode (2675 ms)
[ RUN      ] MainTest.GenerateFinalBoarderTest
gTest.cpp:114: Failure
Value of: output.find("failed.") != std::string::npos
Actual: false
Expected: true
[ FAILED   ] MainTest.GenerateFinalBoarderTest (0 ms)
[ RUN      ] MainTest.GenerateGameBoarderTest1
[ OK       ] MainTest.GenerateGameBoarderTest1 (1 ms)
[ RUN      ] MainTest.GenerateGameBoarderTest2
[ OK       ] MainTest.GenerateGameBoarderTest2 (0 ms)
[ RUN      ] MainTest.generateAnswerBoarderTest
[ OK       ] MainTest.generateAnswerBoarderTest (1 ms)
[-----] 8 tests from MainTest (3602 ms total)

[-----] 4 tests from BoardTest
[ RUN      ] BoardTest.GenerateFinal
[ OK       ] BoardTest.GenerateFinal (540 ms)
[ RUN      ] BoardTest.GenerateGame
[ OK       ] BoardTest.GenerateGame (2666 ms)
[ RUN      ] BoardTest.SolveDFS
[ OK       ] BoardTest.SolveDFS (786 ms)
[ RUN      ] BoardTest.HasUniqueSolution
[ OK       ] BoardTest.HasUniqueSolution (267 ms)
[-----] 4 tests from BoardTest (4259 ms total)

[-----] Global test environment tear-down
[=====] 12 tests from 2 test cases ran. (7861 ms total)
[ PASSED   ] 11 tests.
[ FAILED   ] 1 test, listed below:
[ FAILED   ] MainTest.GenerateFinalBoarderTest

1 FAILED TEST

```

图 2.11: gTest 测试结果

可以看到，12 组测试样例，通过了 11 组，测试通过率为 91.67%。

2.5 测试覆盖率统计

由于 GoogleTest 提供了测试覆盖率的统计工具，可以分别根据代码的行数和函数的个数进行覆盖率的统计，因此我们也使用该工具来进行覆盖率统计，统计结果如下：

LCOV - code coverage report

Current view: top level		Hit	Total	Coverage
Test: coverage.info	Lines:	986	1344	73.3 %
Date: 2023-06-30 01:49:24	Functions:	398	511	77.8 %

Directory	Line Coverage	Functions
/home/ubuntu/RuanJianGongCheng/Sudoku	63.1 % 268 / 425	87.7 % 64 / 73
/usr/include/gtest	90.0 % 9 / 10	87.5 % 7 / 8
/usr/include/gtest/internal	100.0 % 14 / 14	98.3 % 57 / 58
1	100.0 % 6 / 6	100.0 % 4 / 4
7/bits	77.7 % 655 / 842	74.8 % 247 / 330
7/ext	72.3 % 34 / 47	50.0 % 19 / 38

Generated by: [LCOV version 1.13](#)

图 2.12: 覆盖率统计

可以看到，代码行数覆盖率为 73.3%，函数个数覆盖率为 77.8%，说明测试样例设计的较为合理，覆盖范围比较充分。

2.6 BUG 修复

针对之前检测出的错误，我们在 main.cpp 中关于参数范围检测的判断条件加上 <1 的判断，对于数量小于或等于 0 的情形进行报错处理，修改后的代码如下：

```

1  void printErrorMessage(const string& message) {
2      printf("%s failed.\n", message.c_str());
3      exit(-1);
4  }
5      .....
6  Options parseOptions(int argc, char* argv[]) {
7      .....
8      while ((opt = getopt(argc, argv, "c:s:n:m:r:u")) != -1) {
9          switch (opt) {
10             case 'c':
11                 options.mode = Mode::GenerateFinal;
12                 options.count = stoi(optarg);
13                 if (options.count < 1 || options.count > 1000000) { //检测参数范围是否小于 1;
14                     printErrorMessage("Parameters should range from 1 to 1000000");
15                 }
16                 break;
17             .....
18             .....
19
20

```

修改后，再次进行测试：

```

ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ g++ -o test_executable gTest.cpp -lgtest -lgtest_main -pthread
ubuntu@ubuntu:~/RuanJianGongCheng/Sudoku$ ./test_executable
[=====] Running 12 tests from 2 test cases.
[-----] Global test environment set-up.
[-----] 8 tests from MainTest
[ RUN      ] MainTest.GenerateFinalMode
[ OK       ] MainTest.GenerateFinalMode (132 ms)
[ RUN      ] MainTest.GenerateGameMode1
[ OK       ] MainTest.GenerateGameMode1 (265 ms)
[ RUN      ] MainTest.GenerateGameMode2
[ OK       ] MainTest.GenerateGameMode2 (524 ms)
[ RUN      ] MainTest.generateAnswerMode
[ OK       ] MainTest.generateAnswerMode (2603 ms)
[ RUN      ] MainTest.GenerateFinalBoarderTest
[ OK       ] MainTest.GenerateFinalBoarderTest (0 ms)
[ RUN      ] MainTest.GenerateGameBoarderTest1
[ OK       ] MainTest.GenerateGameBoarderTest1 (1 ms)
[ RUN      ] MainTest.GenerateGameBoarderTest2
[ OK       ] MainTest.GenerateGameBoarderTest2 (1 ms)
[ RUN      ] MainTest.generateAnswerBoarderTest
[ OK       ] MainTest.generateAnswerBoarderTest (0 ms)
[-----] 8 tests from MainTest (3526 ms total)

[-----] 4 tests from BoardTest
[ RUN      ] BoardTest.GenerateFinal
[ OK       ] BoardTest.GenerateFinal (516 ms)
[ RUN      ] BoardTest.GenerateGame
[ OK       ] BoardTest.GenerateGame (2660 ms)
[ RUN      ] BoardTest.SolveDFS
[ OK       ] BoardTest.SolveDFS (802 ms)
[ RUN      ] BoardTest.HasUniqueSolution
[ OK       ] BoardTest.HasUniqueSolution (266 ms)
[-----] 4 tests from BoardTest (4245 ms total)

[-----] Global test environment tear-down
[=====] 12 tests from 2 test cases ran. (7771 ms total)
[ PASSED  ] 12 tests.

```

图 2.13: bug 修复后 gTest 测试结果

可以看到，所有样例全部测试通过。

3 代码性能改进

通过1.2 程序性能剖析得知求解游戏和生成游戏时大部分性能开销花到了求出的解是否正确的判断以及深度优先遍历的求解过程中，此方法的复杂度较高，下面对代码进行分析

```

1  void Board::solveDFS(int row, int col, int &count)
2  {
3      //解的个数大于 2 则退出
4      if (count >= 2){return;}
5      //递归超出最后一行，所有的空格都能被填下数字
6      if (row == MaxRow){count++;return;}
7      //递归超出最后一列，开始递归下一行的第一列
8      if (col == MaxCol){solveDFS(row + 1, 0, count);return;}
9      //当前不是空格，递归当前行下一列
10     if (grid[row][col] != 0){solveDFS(row, col + 1, count);return;}
11
12     //是空格，迭代数字 1-9 判断是否有效，有效则填入并继续递归

```

```

13     for (int num = 1; num <= 9; num++)
14     {
15         if (isValid(row, col, num))
16         {
17             grid[row][col] = num;
18             int temp = count;
19             solveDFS(row, col + 1, count);
20             if (count >= 2){return;}
21             else if (count == temp + 1){continue;}
22         }
23     }
24 }

```

我们可以发现在当前格子是空格时，需要判断数字 1-9 中的有效数字填入；这里判断有效性的原理即是数独游戏填数的规则——当前格子所在的行、列、九宫格不能被重复数字填过，应包含 1-9 每个数字一遍，代码如下：

```

1  bool Board::isValid(int row, int col, int num) const
2  {
3      for (int k = 0; k < 9; k++){
4          if (grid[row][k] == num || grid[k][col] == num || grid[(row / 3) * 3 + k / 3]
5              [(col / 3) * 3 + k % 3] == num){
6              return false;}
7      }
8      return true;
9  }

```

上述代码对于每一个空格，都需要进行繁琐的枚举调用 isValid 函数来筛查哪些数字不能使用。但仔细分析规则我们可以得知，如果能够记录下当前空格所在行、列以及九宫格已经填过哪些数字，就可以在当前空格填下还未被填过的数字。那么如何存储已经填过哪些数字的信息呢？我们考虑可以把每一行，每一列，每一个九宫格都用一个 9 位的二进制数压缩，用这个二进制数字的第 k 位上的 0 或 1 来表示 k 是否已经在对应的范围内被使用，然后我们只需要维护这三个数组的数据即可避免重复的多次运算。

因此在递归任意一点时，对它的行，列，九宫格这三个对应值进行位或运算（得到所有用过的数字），再进行取反运算（得到还可以用的数字），即可得到一个数字，这个数字上的每一个“1”都表示一种可以填的数字，最后只需要结合 lowbit 运算即可求出答案。

在 Board 类中添加二进制记录数组以及改进算法的声明：

```

1  class Board {
2      ...
3      //二进制压缩数组，用于记录哪些数字已经被使用过；行，列，每个九宫格的中心（表示这个九宫格）
4      int rowR[15], colR[15], pointR[15][15];

```

```

5      //改进算法
6      void solveImproved(int row, int col, int &count);
7      ...
8  }

```

以及改进算法的实现

```

1  void Board::solveImproved(int row, int col, int &count)
2  {
3      //解的个数大于 2 则退出
4      if (count >= 2){return;}
5      //递归超出最后一行，所有的空格都能被填下数字
6      if (row == MaxRow){count++;return;}
7      //递归超出最后一列，开始递归下一行的第一列
8      if (col == MaxCol){solveDFS(row + 1, 0, count);return;}
9      //当前不是空格，递归当前行下一列
10     if (grid[row][col] != 0){solveDFS(row, col + 1, count);return;}
11
12     //位或运算获取已经被使用过的数字
13     int b = (rowR[row] | colR[col] | pointR[row][col]);
14     b = b ^ ((1 << 10) - 1); // 取反获得可用的数字
15     if ((b & (-b)) == 1)//去除不可用的数字 0
16         b--;
17     while (b > 0)
18     {
19         int m = b & (-b); //lowbit 运算获取最低位的 1
20         rowR[row] += m;
21         colR[col] += m;
22         pointR[deal(row)][deal(col)] += m;
23         grid[row][col] = mmap[m];
24         int temp = count;
25
26         solveImproved(row, col + 1, count);
27
28         //回溯，该空格不再使用 m 对应的数字
29         rowR[row] -= m;
30         colR[col] -= m;
31         pointR[deal(row)][deal(col)] -= m;
32         b-=m;
33
34         if (count >= 2){return;}
35         else if (count == temp + 1){continue;}

```



```
36     }  
37 }
```
