# Software Testing

POLET Simon

29-06-2019

# Table des matières



1. a.k.a. **Validation Testing.** It's use in a paradigm of black box testing.
2. There is also **Regression Testing**

# Chapitre 1

# What is testing ?

## 1.1 Introduction

*Software testing* is a discipline of computer science. The subject she deals with is the study of technics to find and fix problems in a program. Testing a program in the whole have most of the time an insolvent complexity. So, *software testing* test and resolve sub-problems and it helps to become a better tester.

A software under test is tested by different inputs with expected outputs. If you find a bug, you have to check what kind of bug it is.

— Bug in S.U.T
— Bug in acceptability test
— Bug in specification
— Bug in OS, compiler, libraries, hardare

## 1.2 Equivalence tests

When we are testing our code, we produce a single input and receive a single output. To make our tests relevant, we have to define *equivalences classes of test cases.*

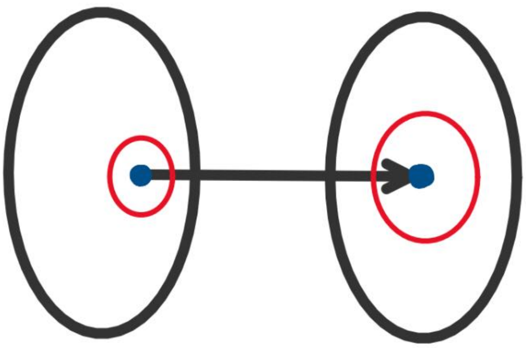

FIGURE 1.1 – Classes d'équivalence d'un input donné vers un output reçu

## 1.3 Creating Testable Software

Before testing our software, we have to produce it and be sure that it will be testable. Here are some tips to help having a testable software :

- Writing clean code
- Knowing what each module does
- Knowing interactions between modules
- No extra thread
- Avoid global variables
- Avoid pointer soup
- Having unit test for modules
- Adding support for fault injection
- Writing a large number of assertions

Rem : Specification is the most important for the developer because it defines what is acceptable.

## 1.4 Assertions

### 1.4.1 Definition

**Assertion :** Executable check for a property that must be True.

### 1.4.2 Using assertions

Assertions are used to check the domain of the error. Thanks to that we can blame the correct module. Making our code self checking with assertions lead to more effective testing because we make the code fail closer to the bug. We usually use assertion for checking *document assumptions*, *preconditions*, *postconditions* and *invariants*.

### 1.4.3 Sous section

Rem : We never use assertions for error handling and we have to avoid making assertions with side effect.

### 1.4.4 Disable assertions

| **Advantages** | **Disavantages** |
|---|---|
| The code is running faster<br>The code keep going | The code could relies to a side effecting assertion |

## 1.5 Kinds of software

**Operating System** When we are testing something like an operating system kernel, we want to test basically with all possible values of the parameters. It's because we want to have an isolation between the user and the operating system to enforce security.

**Graphical User Interface** For a GUI, we have to check every single input (clicks, keyboard, events, swipes, ...) which determine a GUI application state.

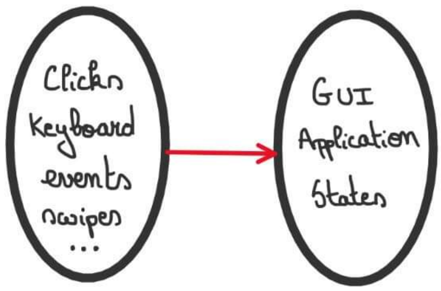


FIGURE 1.2 – GUI Application range and domain

**PRINCIPLE :** Interfaces that span trust boundaries are special and must be tested in the full range of representable values.

**Web Browser** A web browser can't control what the OS does. We have to check if it works correctly. Furthermore, a web browser have to check the time at which input arrive. [1]

## 1.6 Defensive Coding

**Defensive Coding** : It's a form of defensive design intended to ensure the continuing function of a piece of software.

Defensive programming practices are often used where high availability, safety or security is needed. [2]

## 1.7 Fault Injection

**Fault Injection** : Technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths.

1. If the data arrives at a short window of time, the data will be rendered as a web page but if the date comes from the network is scattered across to much time, it wil result a time-out.
2. You can not trust your team mates because you can't even trust yourself.

Ex : In Python, *f = open("tmp/foo", "w")* can have errors that can be difficultly determine because it will almost always succeed. So, we can create *my_open("tmp/foo", "w")* with a implementation almost identical to handle the bug.

## 1.8 Non-Functional Input

**Non-Functional Input** : Inputs that have nothing to do with the API provided by the software that we are testing and nothing to do with the API that are used by the software that we are testing.

Ex : Switching between different threads of execution in a multi-threading software under test [3]. It's difficult to solve because the code is depending of the schedule of these threads and bugs in the software under test can either be concealed or revealed. This is not under the control of the application but of the operating system.

## 1.9 Kinds Of testing

### 1.9.1 White Box Testing - Black Box Testing

**White Box Testing** : Structural tests focus on the implementation.
**Black Box Testing** : Functional tests focused on the specification.

### 1.9.2 Unit Testing - Integration Testing - System Testing [4]

**Unit Testing** : Testing a module in an isolation fashion.
**Integration Testing** : Taking multiple software modules that have already unit tested in combination with each other.
**System Testing** : Test if the system as a whole meets his goal.

The goal of unit testing is to find defects in the internal logic in the software under test as early as possible to make them as robust as possible. On the other size, we must verify in integration testing that we have define and specify tightly enough for different groups of people implementing the different modules were able to make compatible assumptions which are necessary for software modules to all work together. For system testing, we are concerned about how our system will be used. We won't test our system for all possible input but rather make sure that it performs acceptably for important use cases.

### 1.9.3 Different kinds of testing [5]

**Differential Testing** Testing the same input across different implementations of the software under test and comparing them for equality.

3. It's called context switching.
4. a.k.a. **Validation Testing**. It's use in a paradigm of black box testing.
5. There is also **Regression Testing**

**Stress Testing** Test at or beyond the normal limits of a software [6].

**Random Testing** Using the result of a random number generator to randomly create test inputs and deliver them to the software under test. [7]

6. Usually use to increase the robustness and the reliability of the software under test.
7. Usually used to find corner cases in software systems and the crashme program for Unix Kernel.

# Chapitre 2

# Coverage Testing

## 2.1 Score of the Test

When we are testing our code, we want to know how good is our testing. So, we use an automatic tool which tell us where our testing strategy is not doing a good job.

| **Advantages** | **Disavantages** |
|---|---|
| Gives an objective score | Don't find errors of omission |
| coverage $< 100\% \Rightarrow$ meaningful tasks | $100\%$ coverage $\neq$ all bugs were found |

## 2.2 Coverage

**Test Coverage** : Measure of the proportion of the program executed during testing.

Test coverage is an automatic way of partitioning the input domain with some observed features of the source code.

**Function Coverage** Every function in the code is executed during testing.

**Statement Coverage** Every statement in the code is executed during testing.

**Line Coverage** Every physical line in the code is executed during testing.

**Branch Coverage**[1] Every branch in the code is executed in both way during testing.

**Loop Coverage** Every loop in the code is executed 0 times, once, and more than once during testing.

**Modified Condition/Decision Coverage**[2] It's an hybrid which use branch coverage techniques and must takes on every possible input. Furthermore, every condition independently must affect the outcome of a decision.[3]

1. a.k.a **Decision Coverage**
2. This kind of coverage must be done if we are testing a highly critical system.
3. If we have conditionals that don't independently affect the outcome of a decision, there is a programming mistake which means that somebody didn't understand what was going out.

**Path Coverage** Every path in the code is executed during testing.

**Boundary Value Coverage**[4] Each boundary value in the code must be checked during testing.[5]

**Synchronisation Coverage** Ensure that during testing the locks actually does something.[6]

## 2.3 Domain Partitioning and not covered elements

With coverage metrics, we don't cover the whole domain. There are kinds of bugs that don't directly depend of the implementation of the code[7]. There are multiple ways of partitioning the input domain for purpose of testing. No only with an automated coverage, we can also partitioning the domain based on the specification.

When you are covering a code but a part of the code isn't covered, it can means 3 things :
- The code is infeasible[8]
- The code is not worth covering[9]
- The test suite is inadequate

## 2.4 Automated White Box Testing

Some tools are able to make automated white box test based on the code we produce. For example with path coverage, it will start with random parameters then, will check about constraints to check a paths.[10]

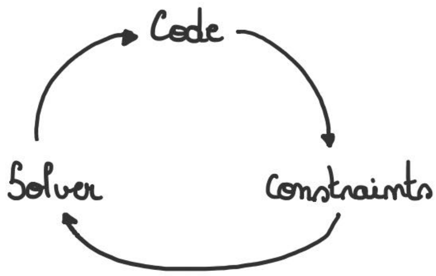


FIGURE 2.1 – Cycle of the automated test

4. Most of the programming errors are boundary errors by one.
5. It's more difficult to test variables that are in interaction because we multiply the number of boundary values to test.
6. We usually use it with **Stress Coverage** to detect bugs and **Interleaving Coverage** which is a concurrency-specific coverage metric when you recall functions which accessed shared data are called in a concurrent fashion that is by multiple threads at the same time.
7. For example, bugs can depend of the device with a full hard disk or something else. It can be resolve for example with **fault injection**
8. It's not necessary bad for example in a checkRep function pieces of code not covered means everything is right. We notify this to the tool with # ***pragma : no cover***
9. It's about things that was already tested elsewhere and the test of this would abort the program.
10. For the language C, Klee is a good automated white box testing tool.

# Chapitre 3

# Random Testing

## 3.1 Driver Script

**Oracle** : Determine whether the output of the software under test is either good or bad.

The *driver script* is the loop making random testing. It works with a random test case generator that takes inputs from a ***Psœudo-random number generator*** feeded by seed which determines the sequence of random numbers generated. Tests cases are submitted to the software under test [1] and inspected by a test oracle and if the output is not OK, we have to save the test case.

Rem : It's important to have significant domain knowledge to create good random test case generator.

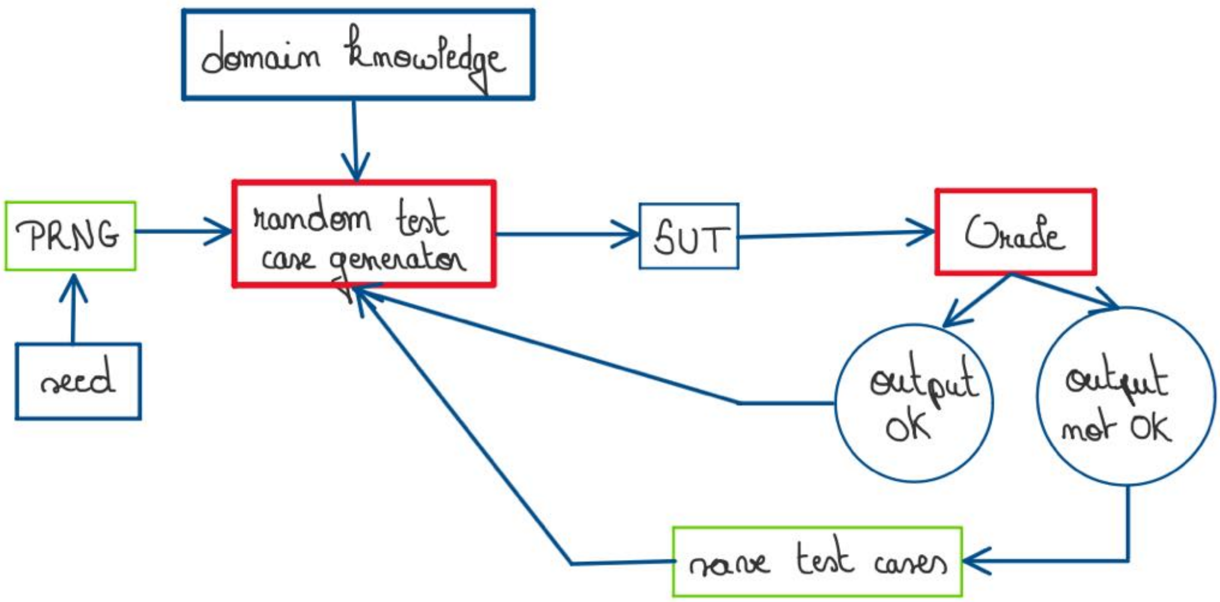


FIGURE 3.1 – Cycle of the driver script

1. The software under test will fail in an obvious way if the programmer included a lot of assertions in the S.U.T..Those assertions can be annoying as compiler uses because the compiler crashes but are actually very good for random testing and we want them to be there.

## 3.2 Input validity

A key problem of building a random test case is generating inputs that are valid. That is to say generating inputs that are part of the input domain for the software under test. If we take an invalid input, it's going to be mapped to a part of the output space that correspond to a malformed or crashed output.

Using invalid inputs in random testing is just spinning, the result is boring output because the input is rejected. If the input isn't rejected, it's because the software under test failed to contain sufficient validity checking logic and it results misbehaviours, crashes, ... [2]

Interfaces that don't spend trust boundaries often lack full *validity checking* because of performance of maintainability reasons. [3]

Rem : **There exists software for wich it is impossible to construct a *validity checker.***

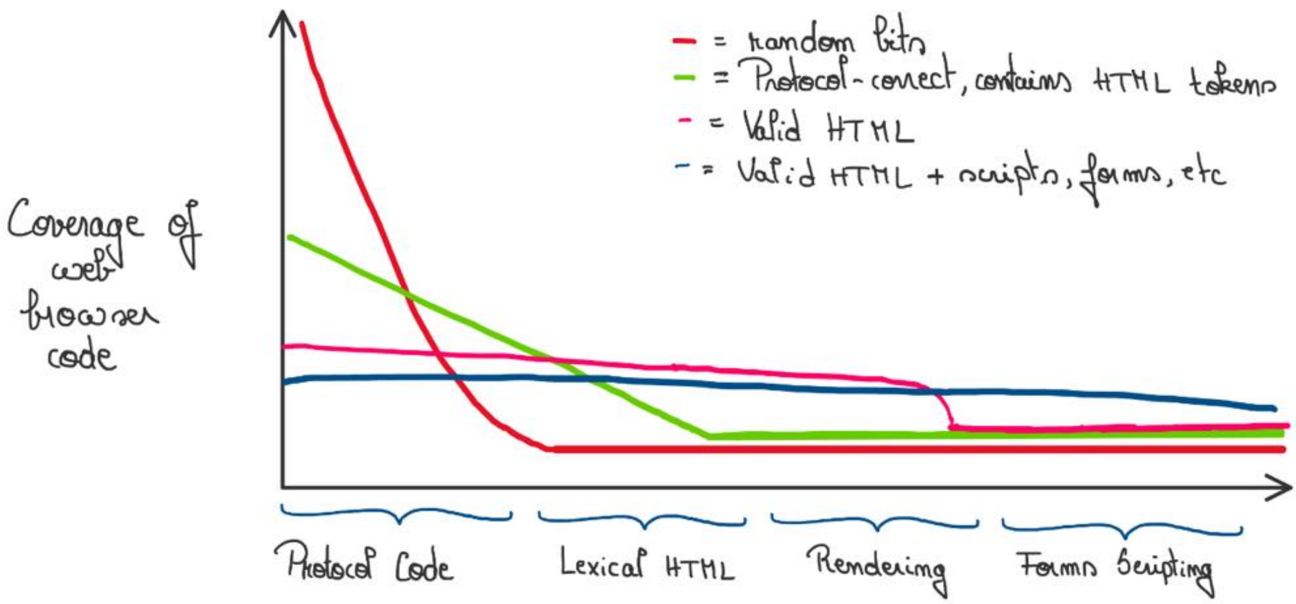


FIGURE 3.2 – Input Validity of a Web Browser Request

2. The problem is that it takes a lot of developer time for nothing.
3. Sometimes we have to trust somebody, otherwise, we can't get a new software written.

## 3.3 Fuzzing

**Fuzzing**[4] : Automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.[5]

It's used for finding vulnerabilities in applications. The goal is to found bugs in the service that are going to let us mount an attack such as a denial of service or some sort of an exploitable vulnerability that will let us mount an intrusion on that service.[6]

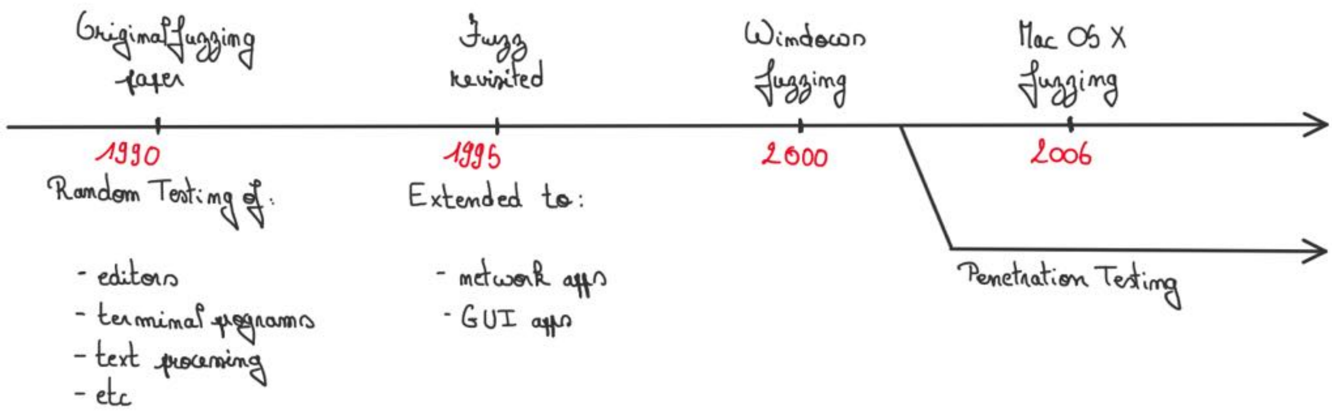


FIGURE 3.3 – Fuzzing Timeline

## 3.4 Random Testing of API's - Fuzzing File Systems

The major thing creating difficulties in random testing is the structure requires inputs. With API's it's just a collection of functions that can be invoked.[7] It's more complicated because there are dependencies among API calls. A second issue is that our test are going to become quite large. We have to find how to represent them.

Ex : The file system has to efficiently respond to all sorts of calls that perform file system operations and it's really important that it works well.

### 3.4.1 First Issue

If we want to do effective random testing of a file system, we need to track dependencies at least in some rudimentary fashion in order to issue a sequence of API calls that's going to do reasonably effective random testing of a file system.

4. It's practically the same thing as random testing.
5. The methodology is basically generating random garbage and not really worrying about the input validity problem.
5. **We can also use our human expertise or human knowledge of the domain interest in order to generate manual test cases that aren't boring.**
6. a.k.a. **Penetration Testing.**
7. They are called with Strings that are sequences of instructions.

### 3.4.2 Second Issue

Ex : The test cases exist as ephemeral objects in memory. The driver code creates the test cases, passes them to the software under test, looks at the results and keep going until it finishes.

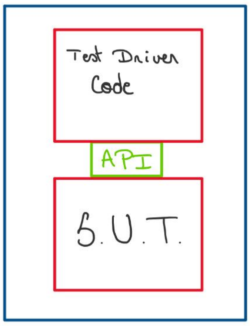


FIGURE 3.4 – Representation of Random Testing in a Python Program

Rem : There are couple of cases in which that's no so good where we'd like to save off that test case for later use in regression testing. Then we can parse files loaded and saved to create test case in memory or print them to disk in order to save test case. It facilitate regression testing and it performs test case reduction.

**Testcase Reduction** : Piece of technology that's very often combined with *random testing*. It takes a large test case that make the software under test fail and turns into smaller test case that still make the software under test fail.

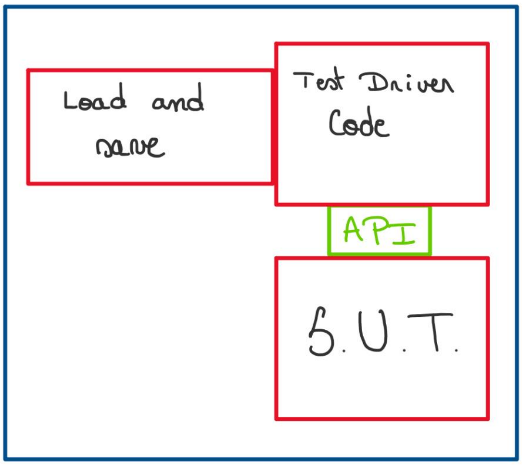


FIGURE 3.5 – Representation of Random Testing in a Python Program

## 3.5 Ways of Generating Random Inputs

**Generative Random Testing** Inputs are created from scratch based on the domain knowledge.[8]

**Mutation-Based Random Testing** Inputs are created by randomly modifying non-randomly created inputs to the software under test. It cluster the domain of input and will start with some known input and randomly modify it in a kind of in the same neighbourhood as the original input.[9]

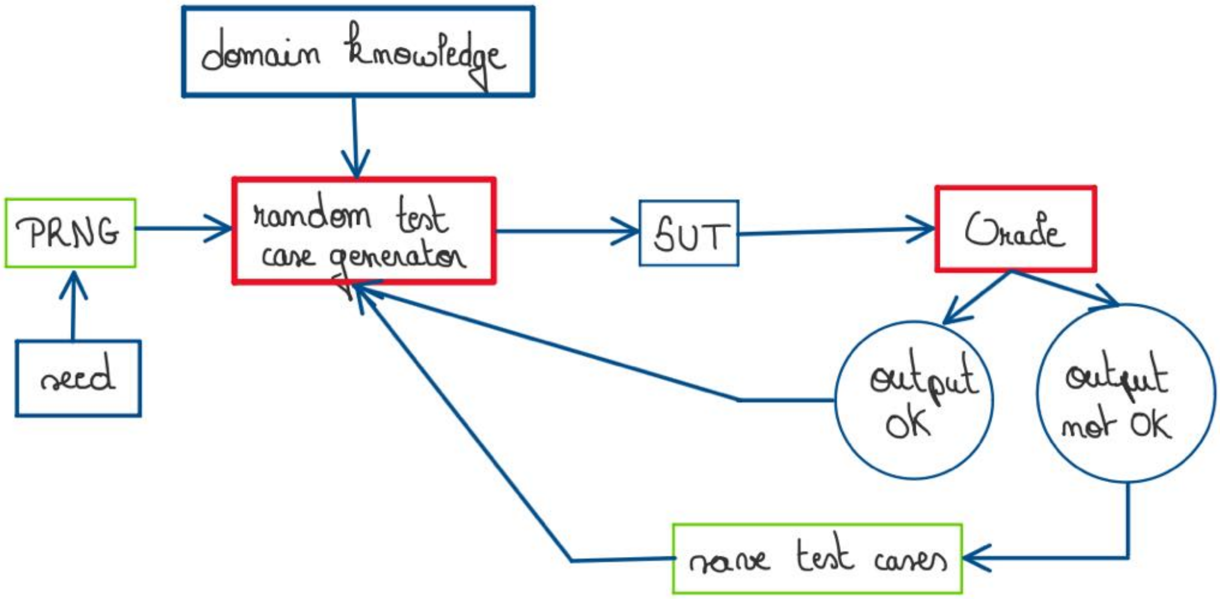


FIGURE 3.6 – Cycle of the driver script

8. It's better in ferreting out weird errors but it takes more time to implement.
9. It's limited to exploring sort of region of the input space that's close to the starting point.

## 3.6 Oracles

Oracles are extremely important for random testing because if we don't have an automated oracle that tells if a test case did something interesting, we've got nothing.

**Weak Oracles** Detect whether or not an application crashed. Which means that the software under test violated some rule that the hardware, a programming language or an enhanced execution environment imposed.

**Medium Oracles** Assertion checks that the programmer has put into the software. It provide a more specific application kind of checking than does the weak oracles.

**Strong Oracles** *Alternate implementation of the same specification* [10], *function inverse pair* [11] and *null space transformation.* [12]

10. Ex : Queue checker/Differential testing of compilers/ old version of S.U.T./reference implementation.

11. Ex : Assembler-Disassembler / Encryption-Decript / Compression-Decompression / Save-Load / transmit-Receive / encode-decode

12. We take a random test case and we make a change that shouldn't affect how its treated by the software under test.

# Chapitre 4

# Random Testing in Practice

## 4.1 Random Testing in the Bigger picture

Random testing testing is effective because of 3 reasons. First, It's based on weak bug hypothesis rather than particular failures. Every test case is a little experiment and the outcome is a bit of *information*. Second, people tend to make the same mistakes while coding and testing and there is a huge asymmetry between speed of computer and people. Even if random tester is mostly generating stupid test, if it can generate a clever test case one in a million time it still more effective than testing resources by hand.

## 4.2 How Random Testing Should Work

First, we have to randomly test our *modules independently* in the early development to have solid foundations. Second, when we *introduce a module in a more sophisticate piece of software*, we have to use another kind of random tester such as those that come in at the top level and those that perform system-level fault injection. Third, in the *complete product*, we have to focus on the external interfaces provided.

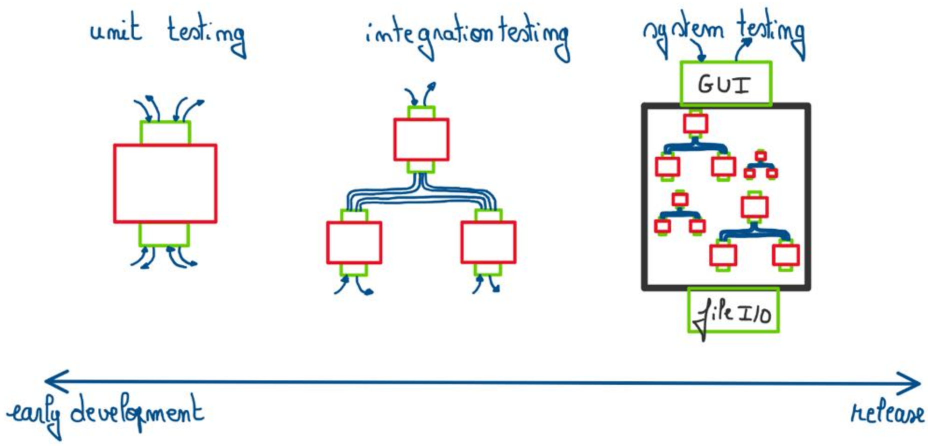


FIGURE 4.1 – Random System Tester

Rem : We have to build the third system-level random tester as early as possible in the development process because it will find some flaws that are not going to flood developers with huge number of bugs. Thanks to that our software will evolve to be more robust.

## 4.3 Tuning Rules and Probabilities

A question we must ask ourself when we are making random testing is : ***Does this random walk have a reasonable probability of executing all the cases ?"***. If we don't have one, we might have to adjust the probabilities. [1]

**Exemple : Fuzzing the bounded Queue**

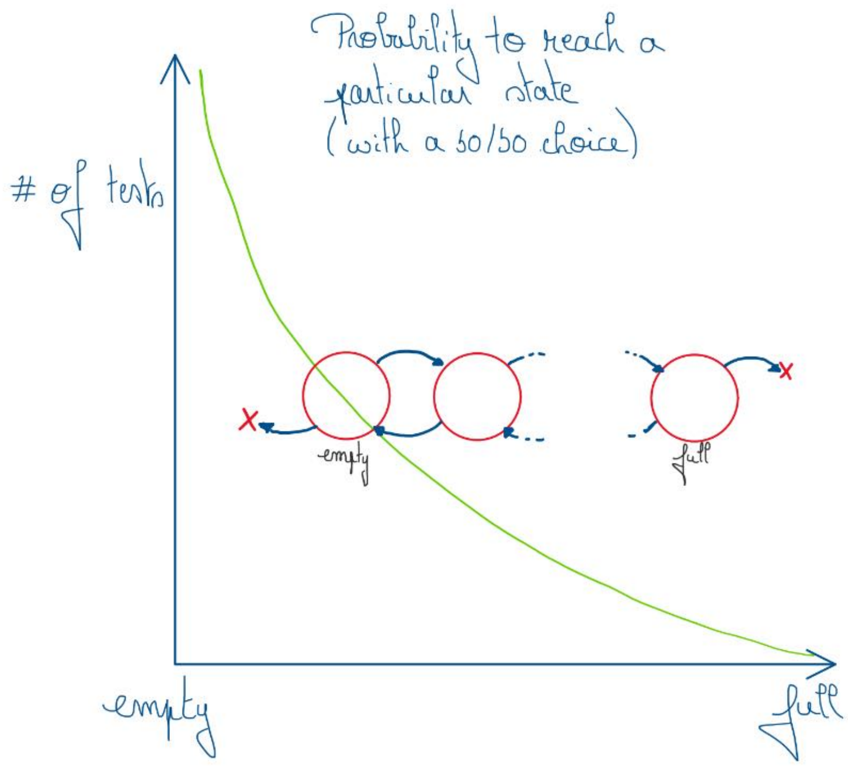


FIGURE 4.2 – Tuning Probabilities

## 4.4 Fuzzing Implicit Inputs

To fuzz implicit inputs there are different techniques [2] :

— Perturbing the schedule
  - Generating load by running apps or other applications
  - Generating network activities
  - Using thread stress testing tool

— Inserting delays near synchronisation and access to shared variables

— Using "unfriendly emulators"[3]

1. In random testing, if we want to do really good job, we need to think about what is that we're testing, how the code is structured and how we're going to execute all the way through it.
2. Ex : The timing at which different threads are on different processors. In that case, the thread scheduler provides a very important form of implicit input to multi-threaded software under test.
3. Emulators especially designed to stress test applications by doing things like invalid cache line, invoking thread schedules in odd ways, ...

## 4.5 Can Random Testing Inspire Confidence ?

If we have all of the following conditions, random testing can inspire confidence. Otherwise we mustn't only use random testing.

**Well-understood API**
+
**Small Code**
+
**Strong Assertions**
+
**Mature, Tuned Random Tester**
+
**Good Coverage Result**
=
**CONFIDENCE**

## 4.6 Tradeoffs in Random Testing

| **Advantages** | **Drawbacks** |
|---|---|
| Less test bias and weaker hypothesis about where bugs are | Input validity can be hard |
| Human cost of testing goes to nearly zero once testing is automated | Oracles can be hard |
| Often surprise us | No stopping criterion |
| Every fuzzer finds different bugs | Every fuzzer finds different bugs |
| | May spend all testing time on boring tests |
| | May find the same bugs many times |
| | Can be hard to debug when test case is large and/or make no sense |
| | May find unimportant bugs |

# Chapitre 5

# Testing in Practice

## 5.1 Overwhelmed by a Good Random Tester

If we made a good random tester, we possibly can be overwhelmed by too many bugs. In this case, we have to find if all these bugs are caused by the same thing or if they are different kind of bugs.

A first solution is to **pick a bug and report** it and maybe some other failures will go away too. On another side, it's irrelevant which bug it is. Maybe some other failures will go away too. Second, we can use **bug triage**.

**Bug Triage** : Process by which the severity of different bugs is determined and we start to disambiguate between different bugs in order to basically try to get a handle on which bugs we can report in parallel.[1] **Core Dump** : Recorded state of the working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally.

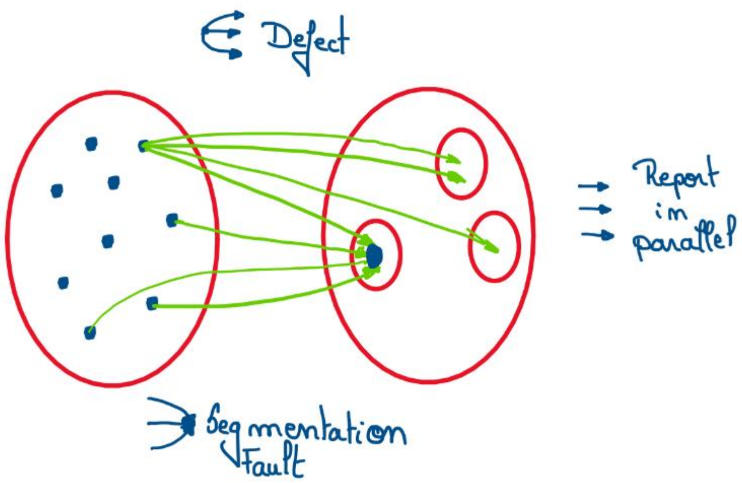


FIGURE 5.1 – Disambiguate by Violation Messages

1. We can disambiguate based on *assertion violation messages*, by *core dump* or *stack trace*, searching over *version history* of the software under test (In a git, use ***git_bisect*** command which do a binary search over revision history in an automated way) or by examine *bug-triggering* test cases.

## 5.2 Test Case Reduction/Minimization

Looking over a large, randomly-generated test cases is painful, so we have to practice *test case reduction*. We have to eliminate part of the input space [1] and see if the smaller input triggers the test case. If it doesn't we go back to the original test case and try again and if it does we do it again. [2]

**Delta Debugging** : Framework that takes a script and takes the test input and automates the process of reduction in a loop which terminates when the delta debugger, which has a bunch of heuristics built in for eliminating parts of the input, can't reduce the input any more.

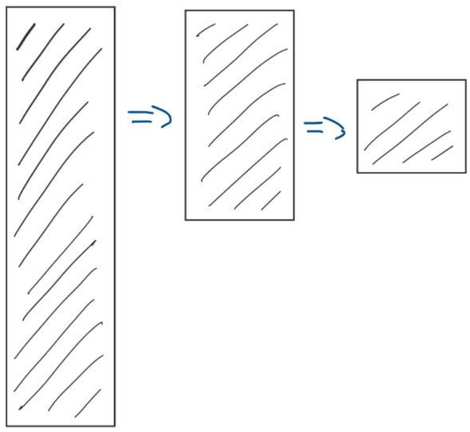

FIGURE 5.2 – Test Case Reduction

## 5.3 Reporting Bugs (to an open source project)

There are tips to respect when reporting bugs :

- Don't report duplicates
- Respect the conventions
- Include a small, stand-alone test case
- Only report valid test cases
- Indicate what output was expected and what the actual output was
- Give instruction for reproduction
  - Platform details
  - Version S.U.T.

1. Sometimes in a smart way or might be just chop some of it out blindly.
2. We can do that by *manual reduction* or also by *delta debugging*.

## 5.4 Building a Test Suite

| |
|---|
| **Test Suite** : Collection of test which often can be test automatically and that we run periodically. |

The goal of test suite is to show that S.U.T. have desired properties [3], that is to say passing over the test[4].

| |
|---|
| **Regression Test** : Any input that cased some previous version to fail.[5] |

Rem : It must be a totally separate activity with random testing.

## 5.5 Hard Testing Problem

There are tips that make testing very hard :
- Lack of / bad specification
- No comparable implementation
- Big S.U.T
- large, highly structured input space
- Non-determinism
- lots of hidden state

3. Ex : Small, feature-specific tests or large, realistic inputs or ***regression test.***
4. Especially proving that the S.U.T. don't have severe bugs.
5. A previous bug which occurred in a previous version could come back due to a lack of gotten rid of all the instances of that defect in the source code, an accidental come back to an old version of a file before we fix the bug or it can comes from errors in people's thinking who didn't correct the error in their thinking.

# Chapitre 6

# Summary

## 6.1 Chapter 1

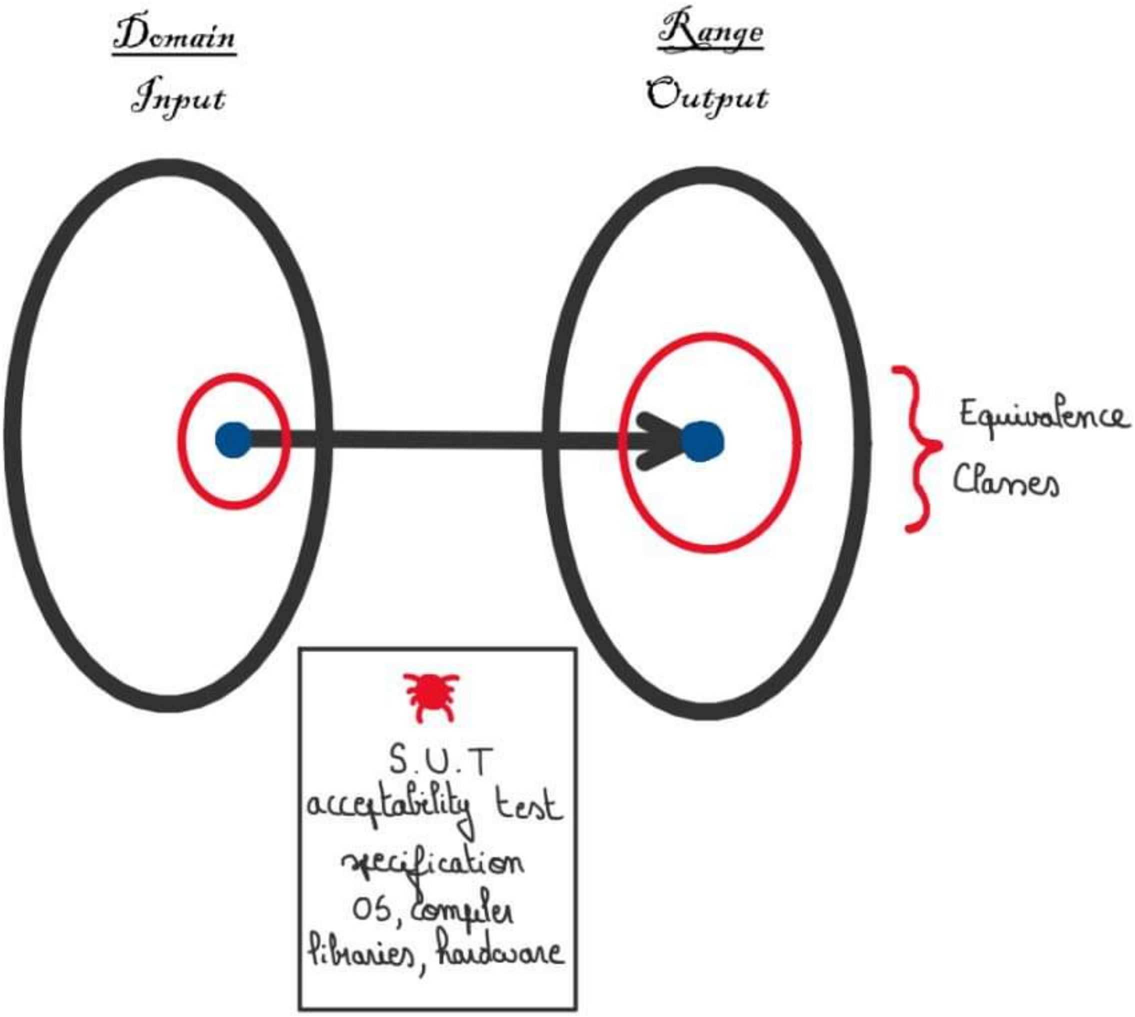


Figure 6.1 – Domain and Range of testing

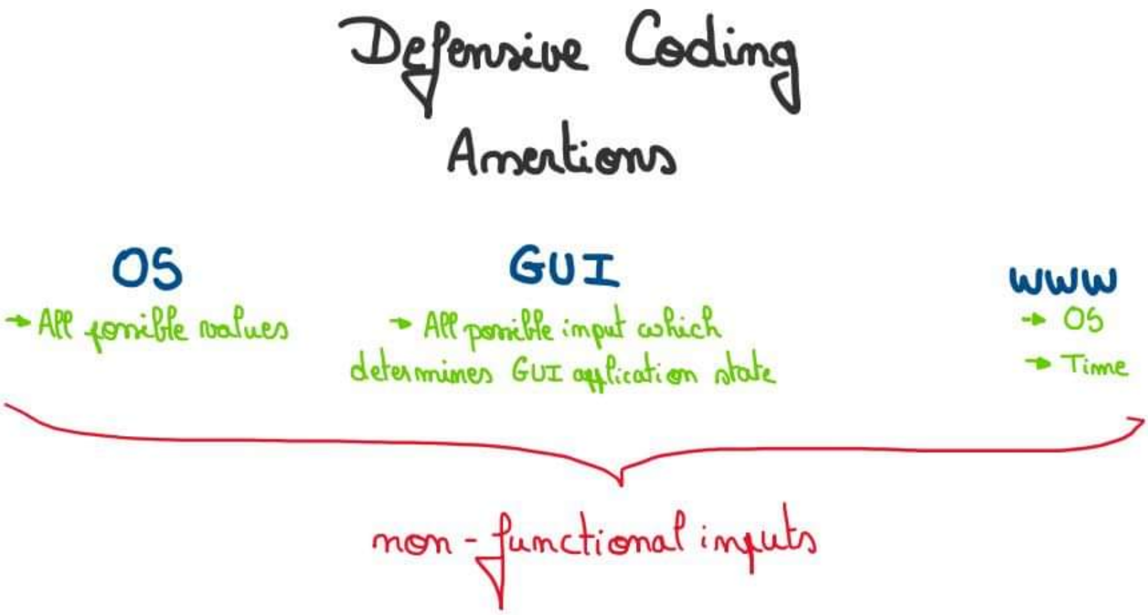


FIGURE 6.2 – Inputs to verify

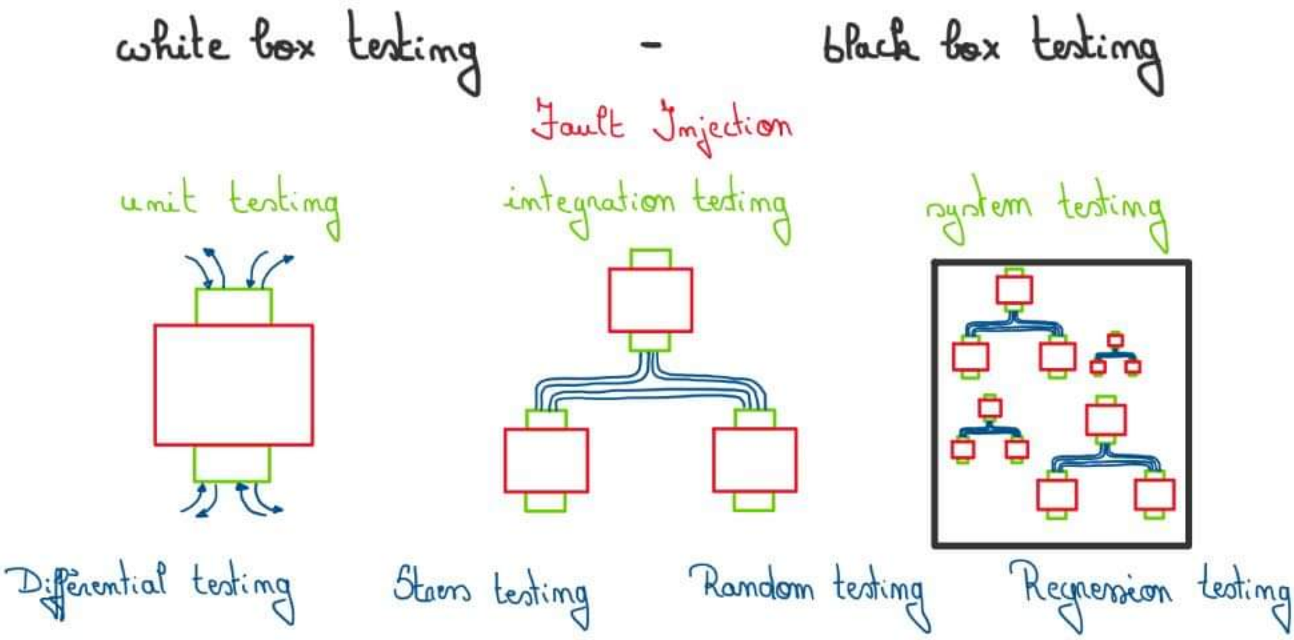


FIGURE 6.3 – Kinds of testing

## 6.2 Chapter 2

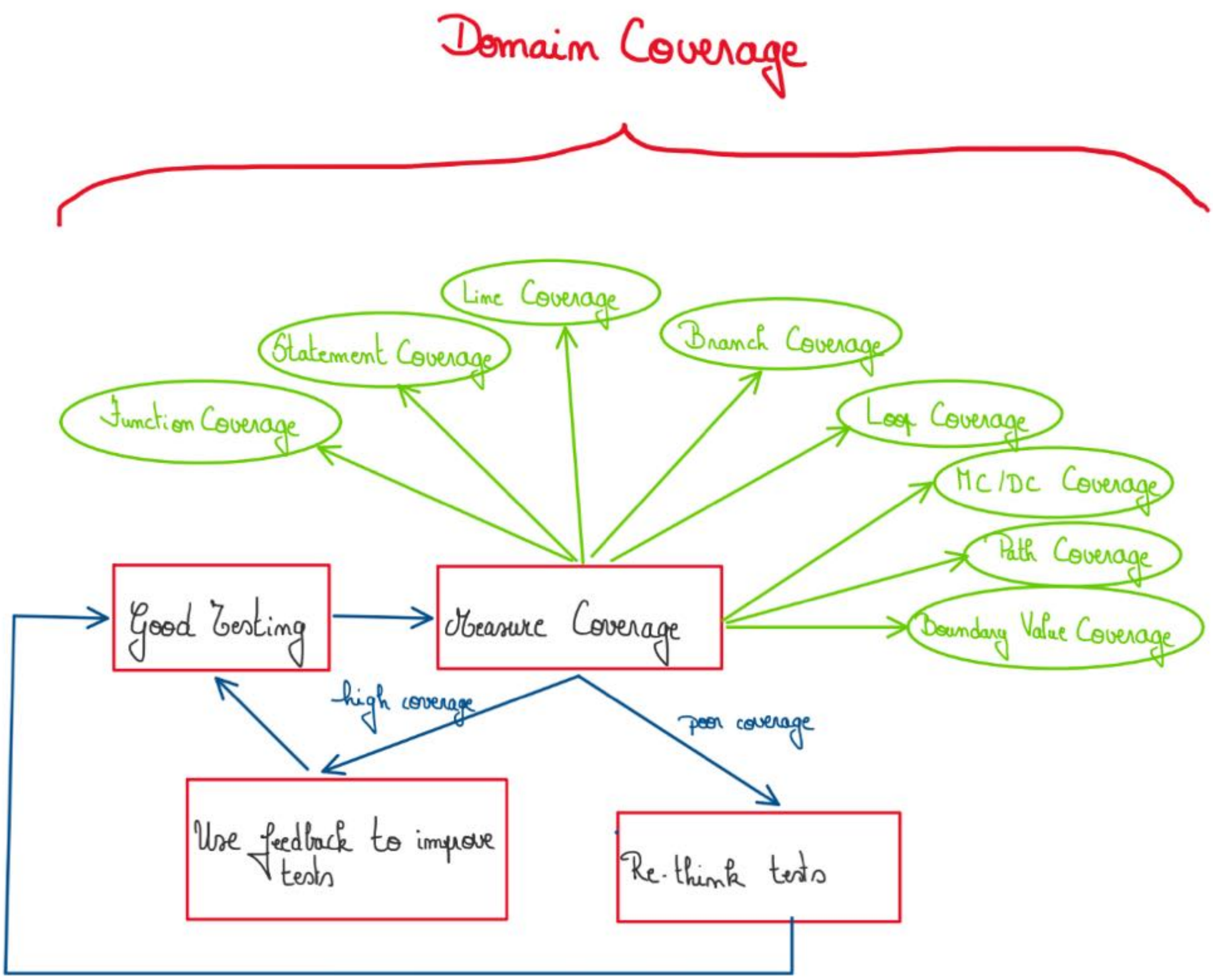


Figure 6.4 – Coverage Testing

## 6.3 Chapter 3

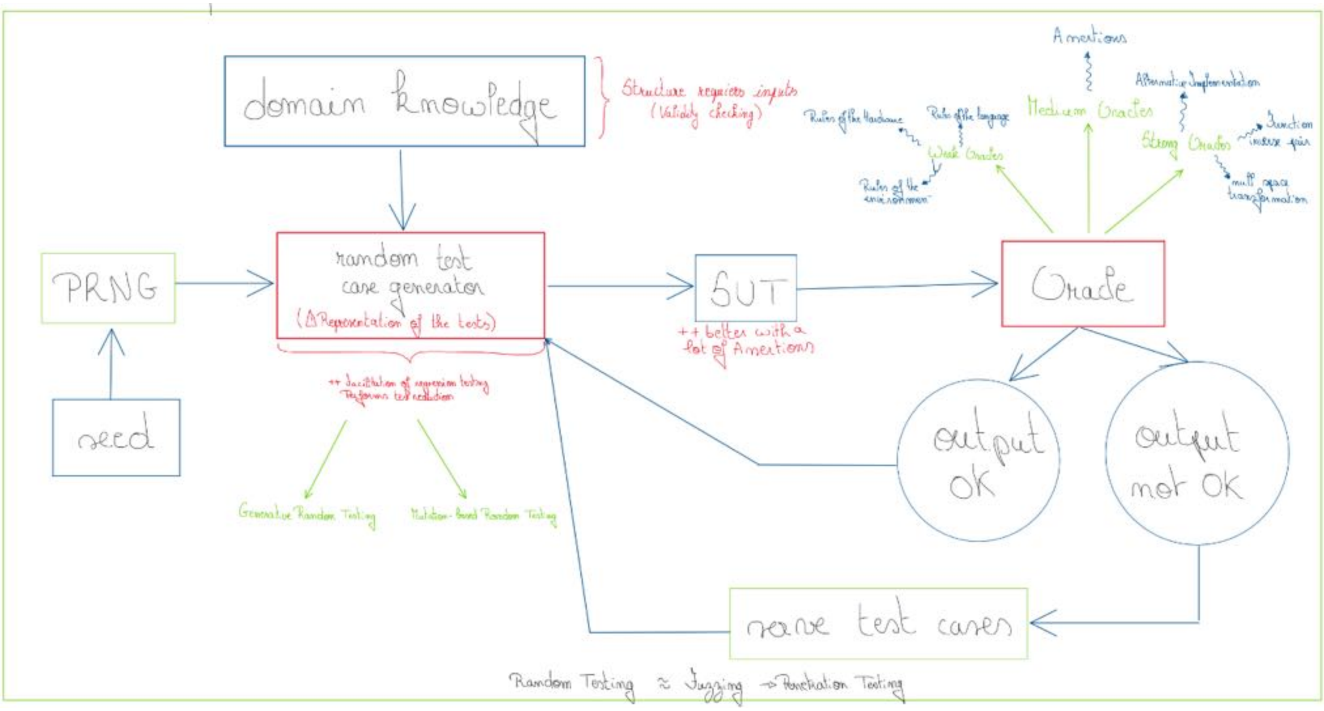


FIGURE 6.5 – Random Testing

## 6.4 Chapter 4

**Does this random walk have a resonable probbility of executing all the cases ?**

**Well-understood API**
+
**Small Code**
+
**Strong Assertions**
+
**Mature, Tuned Random Tester**
+
**Good Coverage Result**
=
**CONFIDENCE**

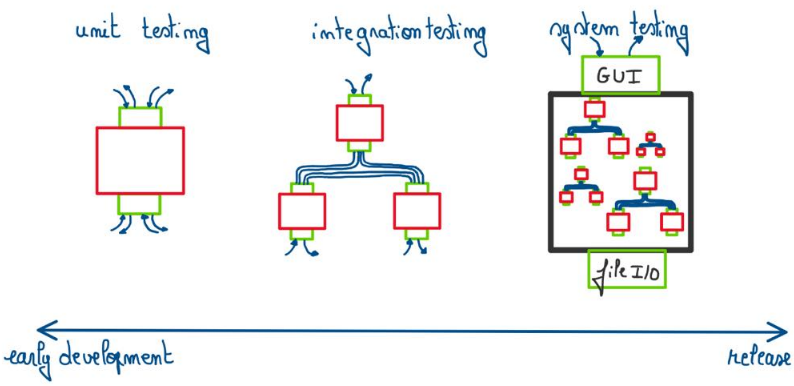


FIGURE 6.6 – Random System Tester

## 6.5 Chapter 5

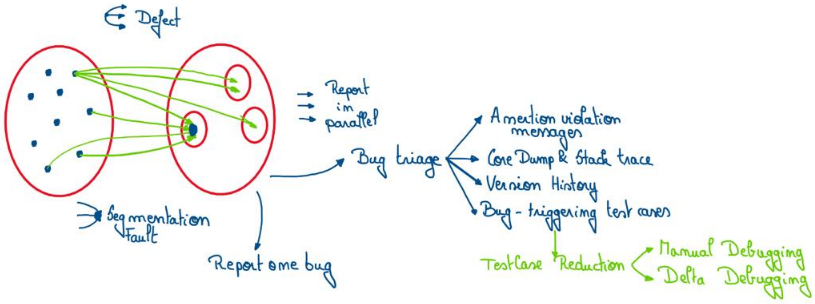


FIGURE 6.7 – Testing in Practice

# Annexe A

# Fixed-Size Queue

```
# -*- coding: utf-8 -*-
import array
import random


"""
@overview the Queue class provides a fixed-size FIFO queue of integers.
A Queue is mutable.
A Queue is a set a values <max, head, tail, size, data> where :
@specfield max integer : the maximum size of the Queue
@specfield head integer : head of the Queue
@specfield tail integer : tail of the Queue
@specfield size integer : size of the Queue
@specfield data
@invariant size >= 0
@invariant max >= 0
@invariant max >= size
"""
class Queue:

    # FA(c) is a fixed-sizze FIFO queue of integers

    # IR(c) : size >= 0 &&
    #         max >= 0 &&
    #         max >= size &&

    # Constructors
    """
    @effects initialize self to a empty of a size_max lenght
    Queue si size_max >= 0
    @throws AssertionError sinon
    """
    def __init__(self, size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max)) # new array of integer

    # Observers
    """
    @return returns true if self is empty
```

```
    """
    def empty(self):
        return self.size == 0

    """
    @return returns true if self is full
    """
    def full(self):
        return self.size == self.max

    # Mutators
    """
    @requires x is an integer
    @modifies self
    @effects we add x to the end of self
    @return False if self is full
            else, return True
    """
    def enqueue(self, x):
        if self.size == self.max:
            return False # Queue is full
        # Update meta-data
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0 # More efficient to move indice than data in the array
        return True

    """
    @modifies self
    @effects we delete the first element of self
    @return None if self is empty
            else, return the first element of self
    """
    def dequeue(self):
        if self.size == 0:
            return None # Queue is empty
        # Update meta-data
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0 # More efficient to move indice than data in the array
        return x

    def checkRep(self):
        assert self.size >= 0 and self.size <= self.max
        if self.tail > self.head:
            assert(self.tail - self.head) == self.size
        if self.tail < self.head:
            assert(self.head - self.tail) == (self.max - self.size)
        if self.tail == self.head:
            assert(self.size == 0) or (self.size == self.max)
        return
```

./Code/FixedSizeQueue.py

## A.1 Ex : Test cases on Fixed-Size Queue

```
def test1():
    q = Queue(3)
    res = q.empty()

    if not res:
        print "test1 NOT OK"
        return
    res = q.enqueue(10)
    if not res:
        print "test1 NOT OK"
    res = q.enqueue(11)
    if not res:
        print "test1 NOT OK"
    x = q.dequeue()
    if x != 10:
        print "test1 NOT OK"
    if x != 11:
        print "test1 NOT OK"
    res = q.empty()
    if not res:
        print "test1 NOT OK"
    print "test1 IS OK"

def test2():
    q = Queue(2)
    res = q.empty()
    if not res:
        print "test2 NOT OK"
        return
    res = q.enqueue(1)
    if not res:
        print "test2 NOT OK"
        return
    res = q.enqueue(2)
    if not res:
        print "test2 NOT OK"
        return
    res = q.enqueue(3)
    if q.tail != 0:
        print "test2 NOT OK"
        return
    print "test2 IS OK"

def test3():
    q = Queue(1)
    res = q.empty()
    if not res:
        print "test3 NOT OK"
        return
    x = q.dequeue()
    if not x is None:
        print "test3 NOT OK"
        return
    res = q.enqueue(1)
    if not res:
        print "test3 NOT OK"
```

```
            return
        x = q.dequeue()
        if x != 1 or q.head != 0:
            print "test3 NOT OK"
            return
        print "test3 IS OK"
```

./Code/TestCases_FixedSizeQueue.py

# Annexe B

# SplayTree

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.left = self.right = None

    def equals(self, node):
        return self.key == node.key

class SplayTree:
    def __init__(self):
        self.root = None
        self.header = Node(None) # For splay()

    def insert(self, key):
        if(self.root == None):
            self.root = Node(key)
            return

        self.splay(key)
        if self.root.key == key:
            # If the key is already there in the tree, don't do anything.
            return

        n = Node(key)
        if key < self.root.key:
            n.left = self.root.left
            n.right = self.root
            self.root.left = None
        else:
            n.right = self.root.right
            n.left = self.root
            self.root.right = None
        self.root = n

    def remove(self, key):
        self.splay(key)
        if key != self.root.key:
            raise "key not found in tree"

        # Now delete the root
        if self.root.left == None:
            self.root = self.root.right
        else:
```

```
                x = self.root.right
                self.root = self.root.left
                self.splay(key)
                self.root.right = x

    def findMin(self):
        if self.root == None:
            return None
        x = self.root
        while x.left != None:
            x = x.left
        self.slay(x.key)
        return x.key

    def findMax(self):
        if self.root == None:
            return None
        x = self.root
        while(x.right != None):
            x = x.right
        self.splay(x.key)
        return x.key

    def find(self, key):
        if self.root == None:
            return None
        self.slay(key)
        if self.root.key != key:
            return None
        return self.root.key

    def isEmpty(self):
        return self.root == None

    # Error if a key is not in our SplayTree
    def splay(self, key):
        l = r = self.header
        t = self.root
        self.header.left = self.header.right = None
        while True:
            if key < t.key:
                if t.left == None:
                    break
                if key < t.left.key:
                    # Right rotate
                    y = t.left
                    t.left = y.right
                    y.right = t
                    t = y
                    if t.left == None:
                        break
                r.left = t
                r = t
                t = t.left
            elif key > t.key:
                if t.right == None:
                    break
                if key < t.right.key:
                    # Left rotate
```

```
                    y = t.right
                    t.right = y.left
                    y.left = t
                    t = y
                    if t.right == None:
                        break
                l.right = t
                l = t
                t = t.right
            else:
                break

        l.right = t.left
        r.left = t.right
        t.left = self.header.right
        t.right = self.header.left
        self.root = t
```

./Code/SplayTree.py

## B.1 Ex : Black Box Testing

```
import unittest
from splay import SplayTree

class TestCase(unittest.TestCase):
    def setUp(self):
        self.keys = [0,1,2,3,4,5,6,7,8,9]
        self.t =SplayTree()
        for key in self.keys:
            self.t.insert(key)

    def testInsert(self):
        for key in self.keys:
            self.assertEquals(key, self.t.find(key))

    def testRemove(self):
        for key in self.key:
            self.t.remove(key)
            self.assertEquals(self.t.find(key), None)
        self.t.remove(-999)

    # Stress testing
    def testLargeInserts(self):
        t = SplayTree()
        nums = 40000
        gap = 307
        i = gap
        while i != 0:
            t.insert(i)
            i = (i + gap) % nums

    def testIsEmpty(self):
        self.assertFalse(self.t.isEmpty())
        t = SplayTree()
        self.assertTrue(t.isEmpty())

    def testMinMax(self):
        self.assertEquals(self.t.findMin(), 0)
        self.assertEquals(self.t.findMax(), 9)

if __name__ == "__main__":
    unittest.main()
```

./Code/TestCases_SplayTree.py

# Annexe C

# Luhn Algorithm

```python
def checkLuhn(identifier):
    sum = identifier[len(identifier) - 1]
    nDigits = len(identifier)
    parity = nDigits % 2

    for i in range(nDitgits):
        digit = (int) identifier[i]
        if i % 2 == parity:
            digit = digit * 2
        if digit > 9:
            digit = digit - 9
        sum += digit
    return (sum % 10 == 0)
```

./Code/LuhnAlgorithm.py

## C.1 Psœdorandom Generation

```python
import random

def generate(pref, l):
    nrand = l - len(pref) - 1
    assert nrand > 0
    n = pref

    for i in range(nrand):
        n += str(random.randrange(10))
    n += "0"
    check = checkLuhn(n)
    if check != 0:
        check = 10 - check
    n = n[:-1] + str(check)
    return n
```

./Code/LuhnNumberGeneration.py

# Annexe D

# Examples

## D.1 Ex : Read_all Function

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <assert.h>
#include <sys/time.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <stdio.h>

ssize_t read_fi(int fildes, void *buf, size_t nbyte)
{
    nbyte = (rand() % nbyte) + 1;
    return read(fildes, buf, nbyte);
}

ssize_t read_all(int fildes, void *buf, size_t nbyte)
{
    assert(fildes >= 0);
    assert(buf);
    assert(nbyte >= 0);

    size_t left = nbyte;
    while(1){
    int res = read_fi(fildes, buf, left);
    printf("%d\n", res);
    if(res < 1)
        resturn res;
    buf += res;
    left -= res;
    assert(left >= 0);
    if(left == 0)
        resturn nbyte;
    }
}

int main(void)
{
    srand(time(NULL));
```

```c

    int fd = open("SplayTree.py", O_RDONLY);
    assert(fd >= 0);

    struct stat buf;
    int res = fstat(fd, &buf);
    assert(res == 0);

    off_t len = buf.st_size;
    char *definitive = (char *) malloc(len);
    assert(definitive);

    res = read(fd, definitive, len);
    assert(res == len);

    int i;
    char *test = (char *) malloc(len);
    for(i = 0; i < 100; i++){
    resr = lseek(fd, 0, SEEK_SET);
    assert(res == 0);
    int j;
    for(j = 0; j < len; j++){
        test[j] = rand();
    }
    res = read_all(fd, test, len);
    assert(res == len);
    assert(strncmp(test, definitive, len) == 0);
    }
    return;
}
```

./Code/Read_All.c

## D.2 Ex : "Babysitting an Army of Monkeys" - Charlie Miller

```python
# Load file into buffer

numwrites = random.randrange(math.ceil((float(len(buf)) / FuzzFactor))) + 1
for j in range(numwrites):
    rbyte = ranom.randrange(256)
    rn = random.randrange(len(buf))
    buf[rn] = "%c"%(rbyte)

# Save buffer
# Run process
# Look at exit code
# If it doesn't die, kill it
# Start over
```

./Code/BabysittingAnArmyOfMonkeys.py

[1]

1. Babysitting an Army of Monkeys 1
1. Babysitting an Army of Monkeys 2

## D.3 Bitwise

```python
import random


def high_common_bits(a, b):
    mask = 0x8000000000000000
    output = 0

    for i in reversed(range(64)):
        if (a&mask) == (b&mask):
            output |= a&mask
        else:
            output |= mask
            return output
        mask >>= 1
    return output

def low_common_bits(a, b):
    mask = 1
    output = 0

    for i in range(64):
        if (a&mask) == (b&mask):
            output |= mask
            return output
        mask <<= 1
    return output

def test(a, b):
    print("a = " + str(a) + " b = " + str(b))
    print(high_common_bits(a, b))
    print(low_common_bits(a, b))


for i in range(10000):
    a = random.getrandbits(64)
    b = a
    for j in range(random.randrange(63)):
        b ^= 1 << random.randrange(0,63)
    print(high_common_bits(a, b) + low_common_bits(a, b))
```

./Code/Bitwise.py

# Annexe E

# Problem Sets

## E.1 BlackBox Testing

```python
# CORRECT SPECIFICATION:
#
# the Queue class provides a fixed-size FIFO queue of integers
#
# the constructor takes a single parameter: an integer > 0 that
# is the maximum number of elements the queue can hold.
#
# empty() returns True if and only if the queue currently
# holds no elements, and False otherwise.
#
# full() returns True if and only if the queue cannot hold
# any more elements, and False otherwise.
#
# enqueue(i) attempts to put the integer i into the queue; it returns
# True if successful and False if the queue is full.
#
# dequeue() removes an integer from the queue and returns it,
# or else returns None if the queue is empty.
#
# Example:
# q = Queue(1)
# is_empty = q.empty()
# succeeded = q.enqueue(10)
# is_full = q.full()
# value = q.dequeue()
#
# 1. Should create a Queue q that can only hold 1 element
# 2. Should then check whether q is empty, which should return True
# 3. Should attempt to put 10 into the queue, and return True
# 4. Should check whether q is now full, which should return True
# 5. Should attempt to dequeue and put the result into value, which
#    should be 10
#
# Your test function should run assertion checks and throw an
# AssertionError for each of the 5 incorrect Queues. Pressing
# submit will tell you how many you successfully catch so far.

from queue_test import *

def test():
```

```
    # Queue1 silently holds only 2 byte unsigned integers, than wraps around
    q = Queue(1)
    succeeded = q.enqueue(65537) # % 2**16 + 1
    assert succeeded
    value = q.dequeue()
    assert value == 65537

    #Queue2 silently fails to hold more than 15 elements
    q = Queue(17) # If greather than 15, put to 15
    for i in range(16):
        suceeded = q.enqueue(i)
        assert succeedded

    #Queue3 implements empty() by checking if dequeue() succeeds.
    #This chenges the state of the queue unintentionally.
    q = Queue(2)
    succeeded = q.enqueue(10)
    assert succeeded
    assert not q.empty() # dequeue the element
    value = q.dequeue()
    assert value == 10

    #Queue4 dequeue() of an empty queue return False instead of None
    q = Queue(2)
    value = q.dequeue() # not faithful to specification
    assert value == None

    #Queue5 holds one less item than intended
    q = Queue(2) # size_max -= 1
    for i in range(2):
        succeeded = q.enqueue(1)
        assert succeeded

test()
```

./ProblemSets/ProblemSet1/ProblemSet1.py

## E.2 Coverage

### E.2.1 Queue Coverage

```python
# TASK:
#
# Achieve full statement coverage on the Queue class.
# You will need to:
# 1) Write your test code in the test function.
# 2) Press submit. The grader will tell you if you
#    fail to cover any specific part of the code.
# 3) Update your test function until you cover the
#    entire code base.
#
# You can also run your code through a code coverage
# tool on your local machine if you prefer. This is
# not necessary, however.
# If you have any questions, please don't hesitate
# to ask in the forums!

import array

class Queue:
    def __init__(self,size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self,x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x

    def checkRep(self):
```

```
        assert self.tail >= 0
        assert self.tail < self.max
        assert self.head >= 0
        assert self.head < self.max
        if self.tail > self.head:
            assert (self.tail-self.head) == self.size
        if self.tail < self.head:
            assert (self.head-self.tail) == (self.max-self.size)
        if self.head == self.tail:
            assert (self.size==0) or (self.size==self.max)

# Test code that archieves 100% coverage of the
# Queue class.

def test():
    q = Queue(2)
    assert q
    q.checkRep()

    empty = q.empty()
    assert empty
    q.checkRep()

    full = q.full()
    assert not full
    q.checkRep()

    result = q.dequeue()
    assert result == None
    q.checkRep()

    result = q.enqueue(10)
    assert result == True
    q.checkRep()

    result = q.enqueue(20)
    assert result == True
    q.checkRep()

    empty = q.empty()
    assert not empty
    q.checkRep()

    full = q.full()
    assert full
    q.checkRep()

    result = q.enqueue(30)
    assert result == False
    q.checkRep()

    result = q.dequeue()
    assert result == 10
    q.checkRep()

    result = q.dequeue()
    assert result == 20
    q.checkRep()
```

```
test()
```

./ProblemSets/ProblemSet2/PS2.1/ProblemSet2_1.py

## Coverage report: 100%

| *Module ↓* | *statements* | *missing* | *excluded* | *coverage* |
|---|---|---|---|---|
| ProblemSet2_1.py | 77 | 0 | 0 | 100% |
| **Total** | **77** | **0** | **0** | **100%** |

*coverage.py v4.5.4, created at 2019-08-05 12:33*

FIGURE E.1 – Coverage

### E.2.2 Splay Tree Coverage

```
# TASK:
#
# Achieve full statement coverage on the Queue class.
# You will need to:
# 1) Write your test code in the test function.
# 2) Press submit. The grader will tell you if you
#    fail to cover any specific part of the code.
# 3) Update your test function until you cover the
#    entire code base.
#
# You can also run your code through a code coverage
# tool on your local machine if you prefer. This is
# not necessary, however.
# If you have any questions, please don't hesitate
# to ask in the forums!

# Test code that archieves 100% coverage of the SplayTree class.

class Node:
    def __init__(self, key):
        self.key = key
        self.left = self.right = None

    def equals(self, node):
        return self.key == node.key

class SplayTree:
    def __init__(self):
        self.root = None
        self.header = Node(None) #For splay()

    def insert(self, key):
        if (self.root == None):
            self.root = Node(key)
            return

        self.splay(key)
        if self.root.key == key:
            # If the key is already there in the tree, don't do anything.
            return

        n = Node(key)
        if key < self.root.key:
            n.left = self.root.left
            n.right = self.root
            self.root.left = None
        else:
            n.right = self.root.right
            n.left = self.root
            self.root.right = None
        self.root = n

    def remove(self, key):
        self.splay(key)
        if self.root is None or key != self.root.key:
            return

```

```
        # Now delete the root.
        if self.root.left== None:
            self.root = self.root.right
        else:
            x = self.root.right
            self.root = self.root.left
            self.splay(key)
            self.root.right = x

    def findMin(self):
        if self.root == None:
            return None
        x = self.root
        while x.left != None:
            x = x.left
        self.splay(x.key)
        return x.key

    def findMax(self):
        if self.root == None:
            return None
        x = self.root
        while (x.right != None):
            x = x.right
        self.splay(x.key)
        return x.key

    def find(self, key):
        if self.root == None:
            return None
        self.splay(key)
        if self.root.key != key:
            return None
        return self.root.key

    def isEmpty(self):
        return self.root == None

    def splay(self, key):
        l = r = self.header
        t = self.root
        if t is None:
            return
        self.header.left = self.header.right = None
        while True:
            if key < t.key:
                if t.left == None:
                    break
                if key < t.left.key:
                    y = t.left
                    t.left = y.right
                    y.right = t
                    t = y
                    if t.left == None:
                        break
                r.left = t
                r = t
                t = t.left
            elif key > t.key:
```

```
                if t.right == None:
                    break
                if key > t.right.key:
                    y = t.right
                    t.right = y.left
                    y.left = t
                    t = y
                    if t.right == None:
                        break
                l.right = t
                l = t
                t = t.right
            else:
                break
        l.right = t.left
        r.left = t.right
        t.left = self.header.right
        t.right = self.header.left
        self.root = t

def test():
    n1 = Node(18)
    n2 = Node(18)
    assert n1.equals(n2)

    s = SplayTree()
    current_min = None
    current_max = None
    s.splay(18)

    empty = s.isEmpty()
    assert empty == True
    _min = s.findMin()
    assert _min == None
    _max = s.findMax()
    assert _max == None

    found = s.find(10)
    assert found == None

    s.insert(100)
    current_min = 100
    current_max = 100

    for i in range(10,20):
        empty = s.isEmpty()
        assert empty == False

        s.insert(i)
        s.insert(i)

        if not current_min or i < current_min:
            current_min = i

        found = s.find(i)
        assert found == i

        _min = s.findMin()
        assert _min == current_min
```

```python

        _max = s.findMax()
        assert _max == current_max

    for i in range(10,20):
        empty = s.isEmpty()
        assert empty == False

        s.remove(i)
        s.remove(i)

        found = s.find(i)
        assert found == None

    s.insert(373)
    s.remove(373)

test()
```

./ProblemSets/ProblemSet2/PS2.2/ProblemSet2_2.py

## Coverage report: 100%

| *Module ↓* | *statements* | *missing* | *excluded* | *coverage* |
|---|---|---|---|---|
| ProblemSet2_2.py | 142 | 0 | 0 | 100% |
| **Total** | **142** | **0** | **0** | **100%** |

*coverage.py v4.5.4, created at 2019-08-05 13:00*

FIGURE E.2 – Coverage

## E.3 Sudoku

### E.3.1 Sudoku Checker

```
# SPECIFICATION:
#
# check_sudoku() determines whether its argument is a valid Sudoku
# grid. It can handle grids that are completely filled in, and also
# grids that hold some empty cells where the player has not yet
# written numbers.
#
# First, your code must do some sanity checking to make sure that its
# argument:
#
# - is a 9x9 list of lists
#
# - contains, in each of its 81 elements, an integer in the range 0..9
#
# If either of these properties does not hold, check_sudoku must
# return None.
#
# If the sanity checks pass, your code should return True if all of
# the following hold, and False otherwise:
#
# - each number in the range 1..9 occurs only once in each row
#
# - each number in the range 1..9 occurs only once in each column
#
# - each number the range 1..9 occurs only once in each of the nine
#   3x3 sub-grids, or "boxes", that make up the board
#
# This diagram (which depicts a valid Sudoku grid) illustrates how the
# grid is divided into sub-grids:
#
# 5 3 4 | 6 7 8 | 9 1 2
# 6 7 2 | 1 9 5 | 3 4 8
# 1 9 8 | 3 4 2 | 5 6 7
# ---------------------
# 8 5 9 | 7 6 1 | 4 2 3
# 4 2 6 | 8 5 3 | 7 9 1
# 7 1 3 | 9 2 4 | 8 5 6
# ---------------------
# 9 6 1 | 5 3 7 | 0 0 0
# 2 8 7 | 4 1 9 | 0 0 0
# 3 4 5 | 2 8 6 | 0 0 0
#
# Please keep in mind that a valid grid (i.e., one for which your
# function returns True) may contain 0 multiple times in a row,
# column, or sub-grid. Here we are using 0 to represent an element of
# the Sudoku grid that the player has not yet filled in.

# Matrix for equivalence tests

# Not a matrix
not_matrix = """[5,3,4,6,7,8,9,1,2],
                [6,7,2,1,9,5,3,4,8],
                [1,9,8,3,4,2,5,6,7],
                [8,5,9,7,6,1,4,2,3],
```

```
                [4,2,6,8,5,3,7,9,1],
                [7,1,3,9,2,4,8,5,6],
                [9,6,1,5,3,7,2,8,4],
                [2,8,7,4,1,9,6,3,5],
                [3,4,5,2,8,6,1,7,9]"""

# Matrix of not numbers
not_number_matrix = [["5","3","4","6","7","8","9","1","2"],
                     ["6","7","2","1","9","5","3","4","8"],
                     ["1","9","8","3","4","2","5","6","7"],
                     ["8","5","9","7","6","1","4","2","3"],
                     ["4","2","6","8","5","3","7","9","1"],
                     ["7","1","3","9","2","4","8","5","6"],
                     ["9","6","1","5","3","7","2","8","4"],
                     ["2","8","7","4","1","9","6","3","5"],
                     ["3","4","5","2","8","6","1","7","9"]]

# check_sudoku should return None
# One element lack
ill_formed = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,8,3,4,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9],   # <---
              [7,1,3,9,2,4,8,5,6],
              [9,6,1,5,3,7,2,8,4],
              [2,8,7,4,1,9,6,3,5],
              [3,4,5,2,8,6,1,7,9]]

# One row lack
row_lack = [[5,3,4,6,7,8,9,1,2],
            [6,7,2,1,9,5,3,4,8],
            [1,9,8,3,4,2,5,6,7],
            [8,5,9,7,6,1,4,2,3],
            [4,2,6,8,5,3,7,9,1],
            [7,1,3,9,2,4,8,5,6],
            [9,6,1,5,3,7,2,8,4],
            [2,8,7,4,1,9,6,3,5]]

# One collumn lack
column_lack = [[5,3,4,6,7,8,9,1],
               [6,7,2,1,9,5,3,4],
               [1,9,8,3,4,2,5,6],
               [8,5,9,7,6,1,4,2],
               [4,2,6,8,5,3,7,9],
               [7,1,3,9,2,4,8,5],
               [9,6,1,5,3,7,2,8],
               [2,8,7,4,1,9,6,3],
               [3,4,5,2,8,6,1,7]]

# check_sudoku should return True
# Valid full sudoku
valid = [[5,3,4,6,7,8,9,1,2],
         [6,7,2,1,9,5,3,4,8],
         [1,9,8,3,4,2,5,6,7],
         [8,5,9,7,6,1,4,2,3],
         [4,2,6,8,5,3,7,9,1],
         [7,1,3,9,2,4,8,5,6],
         [9,6,1,5,3,7,2,8,4],
```

```
            [2,8,7,4,1,9,6,3,5],
            [3,4,5,2,8,6,1,7,9]]

# check_sudoku should return False
# Invalid but well-formed
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [8,5,9,7,6,1,4,2,3],
           [4,2,6,8,5,3,7,9,1],
           [7,1,3,9,2,4,8,5,6],
           [9,6,1,5,3,7,2,8,4],
           [2,8,7,4,1,9,6,3,5],
           [3,4,5,2,8,6,1,7,9]]

# Duplicate on collumn but not row
invalid_column = [[5,3,4,6,7,8,9,1,2],
                  [6,7,2,1,9,5,3,4,8],
                  [1,9,0,3,8,2,5,6,7],
                  [8,5,9,7,6,1,4,2,3],
                  [4,2,6,8,5,3,7,9,1],
                  [7,1,3,9,2,4,8,5,6],
                  [9,6,1,5,3,7,2,8,4],
                  [2,8,7,4,1,9,6,3,5],
                  [3,4,5,2,8,6,1,7,9]]

# Duplicate on row but not column
invalid_row = [[5,3,4,6,7,8,9,1,2],
               [6,7,2,1,9,5,3,4,8],
               [1,9,0,3,8,2,5,6,7],
               [8,5,9,7,6,1,4,2,3],
               [4,2,6,8,5,3,7,9,1],
               [7,1,3,9,2,4,8,5,6],
               [9,6,1,5,3,7,2,8,4],
               [2,8,7,4,1,9,6,3,5],
               [3,4,5,2,0,6,1,7,9]]

# Duplicate in a block
invalid_block = [[0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,1,0,0,0,0,0],
                 [0,0,0,0,1,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0]]

# check_sudoku should return True
# valid semi-empty sudoku
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]
```

```

# valid empty sudoku
empty = [[0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0]]

# check_sudoku should return True
hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]

# Unsolvable Valid Matrix
unsolvable = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,0,3,8,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9,1],
              [7,1,3,9,2,4,8,5,6],
              [9,6,1,5,3,7,2,8,4],
              [2,8,7,4,1,9,6,3,5],
              [3,4,5,2,8,6,1,7,9]]


def matrix_trans(grid):

    # Initialisation of the transposed matrix
    grid_range = len(grid[0])
    gridTrans = []

    for i in range(grid_range):
        gridTrans.append([])

    # Reverse line and columns
    for row in grid:
        for element in row:
            gridTrans[row.index(element)].append(element)

    return gridTrans

def row_validity(row):

    # Creation of temporary variable for modification
    row_tmp = [x for x in row if x != 0]
    if len(row_tmp) != len(set(row_tmp)): # Checks if no duplicates
        return False

    for element in row_tmp:
```

```
            row_tmp = row_tmp[1:]
            if element in row_tmp:
                return False

    return True

def matrix_validity(grid):

    # Creation of the transposed grid
    gridTrans = matrix_trans(grid)

    for row in grid:
        if row_validity(row) == False: # Checks rows validity
            return False

    for row in gridTrans:
        if row_validity(row) == False : # Checks columns validity
            return False

    return True


def check_sudoku(grid):

    sub_list = []

    # Checking the sanity check
    if not isinstance(grid, (list)): # Checks that grid is a list
        return None

    if len(grid) != 9: # Checks the number of columns
        return None

    for row in grid:
        if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
            return None                                   # list and the number
                                                           # of lines

        for element in row:
            if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
                return None                          # are in the domain

    if not matrix_validity(grid):
        return False

    # Boundary delimitation of the subsets
    for i in [(0,3),(3,6),(6,9)]:
        for j in [(0,3),(3,6),(6,9)]:
            # Creation of subsets bases on boundaries
            sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
            # List of elements in the sub-grid
            for row in sub_grid:
                for element in row:
                    if element != 0:
                        sub_list.append(element)

            if len(sub_list) != len(set(sub_list)):
                return False

```

```
            # Reset of the sub-list
            sub_list = []

    return True

matrix = [not_matrix, not_number_matrix, ill_formed, row_lack, column_lack, \
          valid, invalid, invalid_column, invalid_row, invalid_block, easy, \
          empty, hard, unsolvable]

for m in matrix:
    if matrix.index(m) in [0,1,2,3,4]:
        assert check_sudoku(m) == None
    elif matrix.index(m) in [6,7,8,9]:
        assert check_sudoku(m) == False
    elif matrix.index(m) in [5,10,11,12]:
        assert check_sudoku(m)
```

./ProblemSets/ProblemSet3/PS3.1/SudokuChecker.py

## Coverage report: 100%

| *Module ↓* | *statements* | *missing* | *excluded* | *coverage* |
|---|---|---|---|---|
| SudokuChecker.py | 54 | 0 | 0 | 100% |
| **Total** | **54** | **0** | **0** | **100%** |

*coverage.py v4.5.4, created at 2019-08-05 14:04*

FIGURE E.3 – Coverage

### E.3.2 Sudoku Checker Bis

```
# SPECIFICATION:
#
# check_sudoku() determines whether its argument is a valid Sudoku
# grid. It can handle grids that are completely filled in, and also
# grids that hold some empty cells where the player has not yet
# written numbers.
#
# First, your code must do some sanity checking to make sure that its
# argument:
#
# - is a 9x9 list of lists
#
# - contains, in each of its 81 elements, an integer in the range 0..9
#
# If either of these properties does not hold, check_sudoku must
# return None.
#
# If the sanity checks pass, your code should return True if all of
# the following hold, and False otherwise:
#
# - each number in the range 1..9 occurs only once in each row
#
# - each number in the range 1..9 occurs only once in each column
#
# - each number the range 1..9 occurs only once in each of the nine
#   3x3 sub-grids, or "boxes", that make up the board
#
# This diagram (which depicts a valid Sudoku grid) illustrates how the
# grid is divided into sub-grids:
#
# 5 3 4 | 6 7 8 | 9 1 2
# 6 7 2 | 1 9 5 | 3 4 8
# 1 9 8 | 3 4 2 | 5 6 7
# ---------------------
# 8 5 9 | 7 6 1 | 4 2 3
# 4 2 6 | 8 5 3 | 7 9 1
# 7 1 3 | 9 2 4 | 8 5 6
# ---------------------
# 9 6 1 | 5 3 7 | 0 0 0
# 2 8 7 | 4 1 9 | 0 0 0
# 3 4 5 | 2 8 6 | 0 0 0
#
# Please keep in mind that a valid grid (i.e., one for which your
# function returns True) may contain 0 multiple times in a row,
# column, or sub-grid. Here we are using 0 to represent an element of
# the Sudoku grid that the player has not yet filled in.

# check_sudoku should return None
ill_formed = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,8,3,4,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9],  # <---
              [7,1,3,9,2,4,8,5,6],
              [9,6,1,5,3,7,2,8,4],
              [2,8,7,4,1,9,6,3,5],
              [3,4,5,2,8,6,1,7,9]]
```

```python

# check_sudoku should return True
valid = [[5,3,4,6,7,8,9,1,2],
         [6,7,2,1,9,5,3,4,8],
         [1,9,8,3,4,2,5,6,7],
         [8,5,9,7,6,1,4,2,3],
         [4,2,6,8,5,3,7,9,1],
         [7,1,3,9,2,4,8,5,6],
         [9,6,1,5,3,7,2,8,4],
         [2,8,7,4,1,9,6,3,5],
         [3,4,5,2,8,6,1,7,9]]

# check_sudoku should return False
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [8,5,9,7,6,1,4,2,3],
           [4,2,6,8,5,3,7,9,1],
           [7,1,3,9,2,4,8,5,6],
           [9,6,1,5,3,7,2,8,4],
           [2,8,7,4,1,9,6,3,5],
           [3,4,5,2,8,6,1,7,9]]

# check_sudoku should return True
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]

# check_sudoku should return True
hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]


def check_sudoku(grid):

    # Initialisation of variables
    row_tmp = []
    gridTrans = [[],[],[],[],[],[],[],[],[]]
    sub_gridTrans = [[],[],[]]
    sub_list = []

    # Checking the sanity check
    if not isinstance(grid, (list)): # Checks that grid is a list
        return None

    if len(grid) != 9: # Checks the number of columns
```

```
            return None

    for row in grid:
        if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
            return None                                  # list and the number
                                                          # of lines

        for element in row:
            if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
                return None                           # are in the domain

            if element != 0:
                row_tmp.append(element) # Creation of the list for check row
                                        # validity

            # Creation of the transposed grid
            gridTrans[row.index(element)].append(element)

        # Checking the validity of the grid
        if len(row_tmp) != len(set(row_tmp)): # Checks that no element in rows
            return False                       # are redundant

        # Reset the variables for the next teration of the loop
        row_tmp = []

    # Creation of the transposed grid
    for row in gridTrans:
        for element in row:

            if element != 0:
                row_tmp.append(element) # Creation of the list for check column
                                        # validity

        if len(row_tmp) != len(set(row_tmp)): # Checks that no element in
            return False                       # columns are redundant

        row_tmp = []

    # Boundary delimitation of the subsets
    for i in [(0,3),(3,6),(6,9)]:
        for j in [(0,3),(3,6),(6,9)]:
            # Creation of subsets bases on boundaries
            sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
            # List of elements in the sub-grid
            for row in sub_grid:
                for element in row:
                    if element != 0:
                        sub_list.append(element)

            if len(sub_list) != len(set(sub_list)):
                return False

            # Reset of the sub-list
            sub_list = 0

    return True


print(check_sudoku(ill_formed)) # --> None
```

```
print(check_sudoku(valid))        # —> True
print(check_sudoku(invalid))      # —> False
print(check_sudoku(easy))         # —> True
print(check_sudoku(hard))         # —> True
```

./Code/SudokuChecker.py

### E.3.3 Sudoku Randomizer

```
import random

from SudokuChecker import check_sudoku

def random_sudoku():
    sudoku = []
    bias = -0.2

    for i in range(9):
        sudoku.append([])

    for row in sudoku:
        for i in range(9):
            if (random.random() < 0.5 + bias):
                row.append(random.randint(1,9))
                bias -= 0.35
            else:
                row.append(0)
                bias += 0.05

    bias = -0.2

    return sudoku

for i in range(10000):
    print(check_sudoku(random_sudoku()))
```

./ProblemSets/ProblemSet3/PS3.1/SudokuRandomizer.py

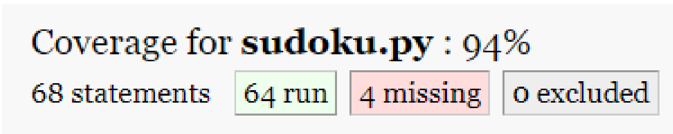


FIGURE E.4 – Coverage

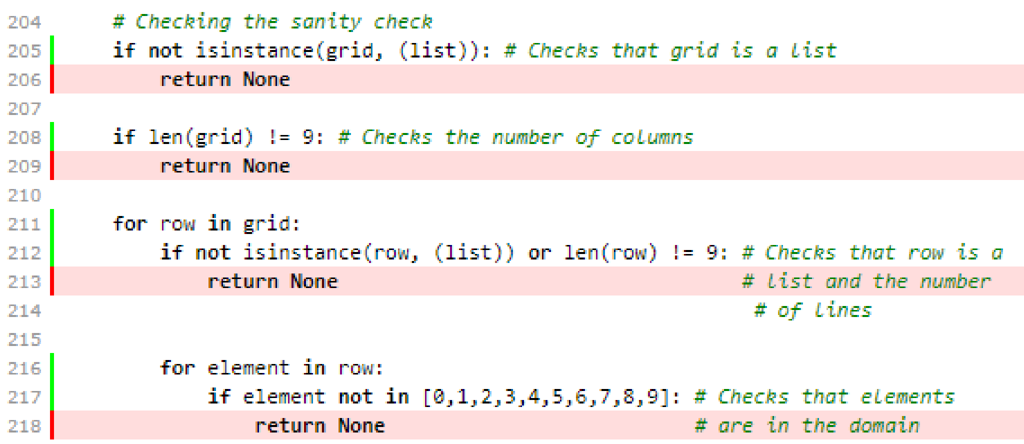


FIGURE E.5 – Coverage

### E.3.4 Sudoku Solver

```
import time


# CHALLENGE PROBLEM:
#
# Use your check_sudoku function as the basis for solve_sudoku(): a
# function that takes a partially-completed Sudoku grid and replaces
# each 0 cell with an integer in the range 1..9 in such a way that the
# final grid is valid.
#
# There are many ways to cleverly solve a partially-completed Sudoku
# puzzle, but a brute-force recursive solution with backtracking is a
# perfectly good option. The solver should return None for broken
# input, False for inputs that have no valid solutions, and a valid
# 9x9 Sudoku grid containing no 0 elements otherwise. In general, a
# partially-completed Sudoku grid does not have a unique solution. You
# should just return some member of the set of solutions.
#
# A solve_sudoku() in this style can be implemented in about 16 lines
# without making any particular effort to write concise code.

# Matrix for equivalence tests

# Not a matrix
not_matrix = """[5,3,4,6,7,8,9,1,2],
                [6,7,2,1,9,5,3,4,8],
                [1,9,8,3,4,2,5,6,7],
                [8,5,9,7,6,1,4,2,3],
                [4,2,6,8,5,3,7,9,1],
                [7,1,3,9,2,4,8,5,6],
                [9,6,1,5,3,7,2,8,4],
                [2,8,7,4,1,9,6,3,5],
                [3,4,5,2,8,6,1,7,9]"""

# Matrix of not numbers
not_number_matrix = [["5","3","4","6","7","8","9","1","2"],
                     ["6","7","2","1","9","5","3","4","8"],
                     ["1","9","8","3","4","2","5","6","7"],
                     ["8","5","9","7","6","1","4","2","3"],
                     ["4","2","6","8","5","3","7","9","1"],
                     ["7","1","3","9","2","4","8","5","6"],
                     ["9","6","1","5","3","7","2","8","4"],
                     ["2","8","7","4","1","9","6","3","5"],
                     ["3","4","5","2","8","6","1","7","9"]]

# check_sudoku should return None
# One element lack
ill_formed = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,8,3,4,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9],   # <---
              [7,1,3,9,2,4,8,5,6],
              [9,6,1,5,3,7,2,8,4],
              [2,8,7,4,1,9,6,3,5],
              [3,4,5,2,8,6,1,7,9]]

```

```
# One row lack
row_lack = [[5,3,4,6,7,8,9,1,2],
            [6,7,2,1,9,5,3,4,8],
            [1,9,8,3,4,2,5,6,7],
            [8,5,9,7,6,1,4,2,3],
            [4,2,6,8,5,3,7,9,1],
            [7,1,3,9,2,4,8,5,6],
            [9,6,1,5,3,7,2,8,4],
            [2,8,7,4,1,9,6,3,5]]

# One collumn lack
column_lack = [[5,3,4,6,7,8,9,1],
               [6,7,2,1,9,5,3,4],
               [1,9,8,3,4,2,5,6],
               [8,5,9,7,6,1,4,2],
               [4,2,6,8,5,3,7,9],
               [7,1,3,9,2,4,8,5],
               [9,6,1,5,3,7,2,8],
               [2,8,7,4,1,9,6,3],
               [3,4,5,2,8,6,1,7]]

# check_sudoku should return True
# Valid full sudoku
valid = [[5,3,4,6,7,8,9,1,2],
         [6,7,2,1,9,5,3,4,8],
         [1,9,8,3,4,2,5,6,7],
         [8,5,9,7,6,1,4,2,3],
         [4,2,6,8,5,3,7,9,1],
         [7,1,3,9,2,4,8,5,6],
         [9,6,1,5,3,7,2,8,4],
         [2,8,7,4,1,9,6,3,5],
         [3,4,5,2,8,6,1,7,9]]

# check_sudoku should return False
# Invalid but well-formed
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [8,5,9,7,6,1,4,2,3],
           [4,2,6,8,5,3,7,9,1],
           [7,1,3,9,2,4,8,5,6],
           [9,6,1,5,3,7,2,8,4],
           [2,8,7,4,1,9,6,3,5],
           [3,4,5,2,8,6,1,7,9]]

# Duplicate on collumn but not row
invalid_column = [[5,3,4,6,7,8,9,1,2],
                  [6,7,2,1,9,5,3,4,8],
                  [1,9,0,3,8,2,5,6,7],
                  [8,5,9,7,6,1,4,2,3],
                  [4,2,6,8,5,3,7,9,1],
                  [7,1,3,9,2,4,8,5,6],
                  [9,6,1,5,3,7,2,8,4],
                  [2,8,7,4,1,9,6,3,5],
                  [3,4,5,2,8,6,1,7,9]]

# Duplicate on row but not column
invalid_row = [[5,3,4,6,7,8,9,1,2],
               [6,7,2,1,9,5,3,4,8],
```

```
                    [1,9,0,3,8,2,5,6,7],
                    [8,5,9,7,6,1,4,2,3],
                    [4,2,6,8,5,3,7,9,1],
                    [7,1,3,9,2,4,8,5,6],
                    [9,6,1,5,3,7,2,8,4],
                    [2,8,7,4,1,9,6,3,5],
                    [3,4,5,2,0,6,1,7,9]]

# Duplicate in a block
invalid_block = [[0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0],
                 [0,0,0,1,0,0,0,0,0],
                 [0,0,0,0,1,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0]]

# check_sudoku should return True
# valid semi-empty sudoku
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]

# valid empty sudoku
empty = [[0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0]]

# check_sudoku should return True
hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]

# Unsolvable Valid Matrix
unsolvable = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,0,3,8,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9,1],
```

```
                [7,1,3,9,2,4,8,5,6],
                [9,6,1,5,3,7,2,8,4],
                [2,8,7,4,1,9,6,3,5],
                [3,4,5,2,8,6,1,7,9]]


def matrix_trans(grid):
    """
    @requires : grid must be an n*n matrix
    @ensures : grid transposed
    """

    # Initialisation of the transposed matrix
    grid_range = len(grid[0])
    gridTrans = []

    for i in range(grid_range):
        gridTrans.append([])

    # Reverse line and columns
    for row in grid:
        for element in row:
            gridTrans[row.index(element)].append(element)

    return gridTrans

def row_validity(row):
    """
    @requires : row instance of list
    @ensures : True if it's a valid sudoku row,
               False either.
    """

    # Creation of temporary variable for modification
    row_tmp = [x for x in row if x != 0]
    if len(row_tmp) != len(set(row_tmp)): # Checks if no duplicates
        return False

    return True

def matrix_validity(grid):
    """
    @requires : grid instance of matrix (list of list)
    @ensures : True if grid is a valid sudoku,
              False either.
    """

    # Creation of the transposed grid
    gridTrans = matrix_trans(grid)

    for row in grid:
        if row_validity(row) == False: # Checks rows validity
            return False

    for row in gridTrans:
        if row_validity(row) == False : # Checks columns validity
            return False

    return True
```

```python


def check_sudoku(grid):
    """
    @ensures : True if grid is a valid matrix,
               False if grid is an invalid matrix,
               None if grid is an invalid input.
    """

    # Checking the sanity check
    if not isinstance(grid, (list)): # Checks that grid is a list
        return None

    if len(grid) != 9: # Checks the number of columns
        return None

    for row in grid:
        if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
            return None                                 # list and the number
                                                         # of lines

        for element in row:
            if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
                return None                          # are in the domain

    if matrix_validity(grid) == False:
        return False

    # Boundary delimitation of the subsets

    for i in [(0,3),(3,6),(6,9)]:
        for j in [(0,3),(3,6),(6,9)]:
            # Creation of subsets bases on boundaries
            sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
            sub_row = [row for sub_row in sub_grid for row in sub_row]
            if row_validity(sub_row) == False:
                return False

    return True


def backTracking(grid):
    """
    @requires : grid is a valid matrix
    @modifies : grid
    @ensures : solved grid if grid is solvable,
               False either.
    """

    # Take the first blankBox
    blankBox = pickBox(grid)
    if not blankBox:
        return grid # The sudoku is solved
    else:
        # Take coordinates
        row, column = blankBox

    # Check elements in the box from 1 to 9
    for element in range(1,10):
```

```
        # Check if it's a valid element
        if element_validity(grid, row, column, element):
            # Add the element to the grid
            grid[row][column] = element

            # Recursivity
            if backTracking(grid):
                return grid # return the sudoku solved

            # If we go to a dead end, come back and try again
            grid[row][column] = 0

    return False


def element_validity(grid, row, column, element):
    """
    @requires : grid is a valid matrix &&
                0 <= row < 9 && 0 <= column < 9 &&
                0 < element <= 9
    @ensures : True if the element in this position is valid,
               False either.
    """

    # Check if element not in row
    for columns in grid[row]:
        if columns == element and column != grid[row].index(columns):
            return False

    # Check if element not in column
    for rows in grid:
        if element == grid[grid.index(rows)][column] and row != grid.index(rows):
            return False

    # Check if element not in box
    box_x = column // 3
    box_y = row // 3

    for i in range(box_y*3, box_y*3 + 3):
        for j in range(box_x * 3, box_x*3 + 3):
            if grid[i][j] == element and i != row and j != column:
                return False

    return True


def display_sudoku(grid):
    """
    @requires grid is a matrix (list of list) or None or False
    """

    # Create a GUI
    if (grid is None) or (grid is False):
        print(grid)
    else:
        for i in range(len(grid)):
            if (i % 3 == 0) and (i != 0):
                print("- - - - - - - - - - - - - - ")
```

```python

            for j in range(len(grid[0])):
                if (j % 3 == 0) and (j != 0):
                    print(" | ", end="")

                if j == 8:
                    print(grid[i][j])
                else:
                    print(str(grid[i][j]) + " ", end="")


def pickBox(grid):
    """
    @requires : grid is a valid matrix
    @returns : a tuple with the row and the column of an empty box if it exists,
               None either.
    """

    # Look for the first empty box
    for row in grid:
        for column in grid[grid.index(row)]:
            if column == 0:
                return (grid.index(row), row.index(column))

    # Return None if there is no box left
    return None


def solve_sudoku(grid):
    """
    @return : A solved sudoku if grid is solvable,
              False if the sudoku is unsolvable,
              None if the grid is an invalid input.
    """

    # Check if it's a valid sudoku
    state = check_sudoku(grid)
    if (state is None) or (state is False):
        return state

    # BackTracking Algorithm
    grid = backTracking(grid)

    return grid


matrix = [not_matrix, not_number_matrix, ill_formed, row_lack, column_lack,
          valid, invalid, invalid_column, invalid_block, invalid_row, easy,
          empty, hard, unsolvable]

for grid in matrix:
    print(matrix.index(grid))
    start = time.time()
    display_sudoku(solve_sudoku(grid))
    end = time.time()
    print("%s seconds" %(end - start))
```

./ProblemSets/ProblemSet3/PS3.2/SudokuSolver.py

Coverage report: 100%

| *Module ↓* | *statements* | *missing* | *excluded* | *coverage* |
|---|---|---|---|---|
| SudokuSolver.py | 115 | 0 | 0 | 100% |
| **Total** | **115** | **0** | **0** | **100%** |

*coverage.py v4.5.4, created at 2019-08-15 10:09*

FIGURE E.6 – Coverage

### E.3.5 Sudoku Solver Bis

```
import copy
import time

def check_row(row):
    t_row = [x for x in row if x != 0]
    if len(t_row) != len(set(t_row)):
        return False
    return True

def check_sudoku(grid):

    if len(grid) != 9 or type(grid) is not list: #checks if there are 9 columns and the grid is a list
        print("grid is not a list or number of columns != 9")
        return None

    for row in grid: #checks if there are 9 rows
        if len(row) != 9:
            print("number of rows != 9")
            return None

    for row in grid:
        for element in row:
            if not isinstance(element, (int)): #checks if numbers are intergers
                print("numbers are not intergers")
                return None
            elif not (element in [0,1,2,3,4,5,6,7,8,9]): #checks if numbers are between 0 and 9
                print("numbers not between 0 and 9")
                return None

    for row in grid:
        if check_row(row) == False: #checks if all the rows are valid
            print("some rows are invalid")
            return False

    trans_grid = zip(*grid)
    for row in trans_grid:
        if check_row(row) == False:
            return False

    for j in [(0,3),(3,6),(6,9)]:  #checks each 3x3 subgrids
```

```
        for i in [(0,3),(3,6),(6,9)]:
            cuad = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
            l = [v for sub in cuad for v in sub]
            if check_row(l) == False:
                return False

    return True

def solve_sudoku(grid):
    state = check_sudoku(grid)
    if state is None or state is False:
        return state

    new_grid = copy.deepcopy(grid)
    # new_grid = grid

    for i in range(9):
        single_row = new_grid[i]
        for j in range(9):
            single_column = [x for x in [new_grid[y][j] for y in range(9)]]
            if new_grid[i][j] == 0:
                # possible values
                ns = [x for x in range(1, 10) if x not in single_row and x not in single_column]
                for n in ns:
                    new_grid[i][j] = n
                    new = solve_sudoku(new_grid)
                    if new is not False:
                        return new
                return False
    return new_grid




# solve_sudoku should return None
ill_formed = [[5,3,4,6,7,8,9,1,2],
              [6,7,2,1,9,5,3,4,8],
              [1,9,8,3,4,2,5,6,7],
              [8,5,9,7,6,1,4,2,3],
              [4,2,6,8,5,3,7,9],   # <---
              [7,1,3,9,2,4,8,5,6],
              [9,6,1,5,3,7,2,8,4],
              [2,8,7,4,1,9,6,3,5],
              [3,4,5,2,8,6,1,7,9]]
# --> None

# solve_sudoku should return valid unchanged
valid = [[5,3,4,6,7,8,9,1,2],
         [6,7,2,1,9,5,3,4,8],
         [1,9,8,3,4,2,5,6,7],
         [8,5,9,7,6,1,4,2,3],
         [4,2,6,8,5,3,7,9,1],
         [7,1,3,9,2,4,8,5,6],
         [9,6,1,5,3,7,2,8,4],
         [2,8,7,4,1,9,6,3,5],
         [3,4,5,2,8,6,1,7,9]]
# --> True

```

```
# solve_sudoku should return False
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [8,5,9,7,6,1,4,2,3],
           [4,2,6,8,5,3,7,9,1],
           [7,1,3,9,2,4,8,5,6],
           [9,6,1,5,3,7,2,8,4],
           [2,8,7,4,1,9,6,3,5],
           [3,4,5,2,8,6,1,7,9]]
# --> False

# solve_sudoku should return a
# sudoku grid which passes a
# sudoku checker. There may be
# multiple correct grids which
# can be made from this starting
# grid.
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]
# --> True

# Note: this may timeout
# in the Udacity IDE! Try running
# it locally if you'd like to test
# your solution with it.

hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]
# --> True

hard2 = [[0,5,0,0,3,0,0,2,0],
        [1,0,0,0,0,9,0,0,6],
        [0,0,0,6,0,7,0,0,0],
        [0,9,2,0,0,0,1,0,0],
        [4,0,0,0,0,0,0,0,7],
        [0,0,5,0,0,0,3,9,0],
        [0,0,0,7,0,8,0,0,0],
        [6,0,0,2,0,0,0,0,4],
        [0,7,0,0,1,0,0,3,0]]
# --> True

hard3 = [[8,0,0,0,0,0,0,0,0],
        [0,0,3,6,0,0,0,0,0],
        [0,7,0,0,9,0,2,0,0],
```

```
        [0,5,0,0,0,7,0,0,0],
        [0,0,0,0,4,5,7,0,0],
        [0,0,0,1,0,0,0,3,0],
        [0,0,1,0,0,0,0,6,8],
        [0,0,8,5,0,0,0,1,0],
        [0,9,0,0,0,0,4,0,0]]

########## Below I've provided some extra Sudoku grids to check. ##########

valid_but_unsolved =   [[5, 3, 0, 0, 7, 0, 0, 0, 0],    # A typical unsolved Sudoku puzzle (from Wikipedia)
                        [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]]
# --> True

grid_solved =  [[5, 3, 4, 6, 7, 8, 9, 1, 2],    # The same Sudoku puzzle, but solved (from Wikipedia)
                [6, 7, 2, 1, 9, 5, 3, 4, 8],
                [1, 9, 8, 3, 4, 2, 5, 6, 7],
                [8, 5, 9, 7, 6, 1, 4, 2, 3],
                [4, 2, 6, 8, 5, 3, 7, 9, 1],
                [7, 1, 3, 9, 2, 4, 8, 5, 6],
                [9, 6, 1, 5, 3, 7, 2, 8, 4],
                [2, 8, 7, 4, 1, 9, 6, 3, 5],
                [3, 4, 5, 2, 8, 6, 1, 7, 9]]
# --> True

##### Most of the Sudoku grids below are my derivations of the above 2 Sudoku puzzles

grid_not_a_list =  ([5, 3, 4, 6, 7, 8, 9, 1, 2],    # Grid is a tuple of lists, rather than a list of lists
                    [6, 7, 2, 1, 9, 5, 3, 4, 8],
                    [1, 9, 8, 3, 4, 2, 5, 6, 7],
                    [8, 5, 9, 7, 6, 1, 4, 2, 3],
                    [4, 2, 6, 8, 5, 3, 7, 9, 1],
                    [7, 1, 3, 9, 2, 4, 8, 5, 6],
                    [9, 6, 1, 5, 3, 7, 2, 8, 4],
                    [2, 8, 7, 4, 1, 9, 6, 3, 5],
                    [3, 4, 5, 2, 8, 6, 1, 7, 9])
# --> None

too_few_rows = [[5, 3, 4, 6, 7, 8, 9, 1, 2],    # Only 8 rows
                [6, 7, 2, 1, 9, 5, 3, 4, 8],
                [1, 9, 8, 3, 4, 2, 5, 6, 7],
                [8, 5, 9, 7, 6, 1, 4, 2, 3],
                [4, 2, 6, 8, 5, 3, 7, 9, 1],
                [7, 1, 3, 9, 2, 4, 8, 5, 6],
                [9, 6, 1, 5, 3, 7, 2, 8, 4],
                [3, 4, 5, 2, 8, 6, 1, 7, 9]]
# --> None

```

```
row_not_a_list =  [(5, 3, 4, 6, 7, 8, 9, 1, 2),    # Grid is a list of tuples, rather than a lists of lists
                   (6, 7, 2, 1, 9, 5, 3, 4, 8),
                   (1, 9, 8, 3, 4, 2, 5, 6, 7),
                   (8, 5, 9, 7, 6, 1, 4, 2, 3),
                   (4, 2, 6, 8, 5, 3, 7, 9, 1),
                   (7, 1, 3, 9, 2, 4, 8, 5, 6),
                   (9, 6, 1, 5, 3, 7, 2, 8, 4),
                   (2, 8, 7, 4, 1, 9, 6, 3, 5),
                   (3, 4, 5, 2, 8, 6, 1, 7, 9)]
# --> None

row_incomplete =  [[5, 3, 4, 6, 7, 8, 9, 1],    # 1st row is incomplete (only 8 elements, rather than 9)
                   [6, 7, 2, 1, 9, 5, 3, 4, 8],
                   [1, 9, 8, 3, 4, 2, 5, 6, 7],
                   [8, 5, 9, 7, 6, 1, 4, 2, 3],
                   [4, 2, 6, 8, 5, 3, 7, 9, 1],
                   [7, 1, 3, 9, 2, 4, 8, 5, 6],
                   [9, 6, 1, 5, 3, 7, 2, 8, 4],
                   [2, 8, 7, 4, 1, 9, 6, 3, 5],
                   [3, 4, 5, 2, 8, 6, 1, 7, 9]]
# --> None

non_integer =  [[5.0, 3, 0, 0, 7, 0, 0, 0, 0],  # 1st element is a float (5.0) rather than an int (5)
                [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]]
# --> None

out_of_range = [[50, 3, 0, 0, 7, 0, 0, 0, 0],   # 1st element is '50', which is out of the range of 0..9
                [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]]
# --> None

trivial_all_zeros =   [[0, 0, 0, 0, 0, 0, 0, 0, 0],   # Trivial case of a grid consisting of lists of nothing but zeros.
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0]]
# --> True
```

```
duplicate_in_row = [[5, 3, 0, 0, 7, 0, 0, 0, 5],    # '5' duplicated in 1st row
                    [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]]
# --> False

duplicate_in_col = [[5, 3, 0, 0, 7, 0, 0, 0, 0],    # '5' duplicated in 1st
    column
                    [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],
                    [5, 0, 0, 0, 8, 0, 0, 7, 9]]
# --> False

duplicate_in_grid =    [[5, 3, 0, 0, 7, 0, 0, 0, 0],    # 5' duplicated in 1st
    sub-grid
                        [6, 0, 0, 1, 9, 5, 0, 0, 0],
                        [0, 9, 5, 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]]
# --> False

no_soln1 = [[1,2,3,4,5,6,7,8,0],
            [0,0,0,0,0,0,0,0,9],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0]]
# --> False

no_soln2 = [[1, 2, 3, 0, 0, 0, 0, 0, 0],
            [4, 5, 0, 0, 0, 0, 6, 0, 0],
            [0, 0, 0, 6, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0]]
# --> False
```

```
sudoku_grids = [ill_formed, valid, invalid, easy, hard, hard2, hard3,
    valid_but_unsolved, grid_solved,
                grid_not_a_list, too_few_rows, row_not_a_list, row_incomplete,
    non_integer,
                out_of_range, trivial_all_zeros, duplicate_in_row,
    duplicate_in_col,
                duplicate_in_grid, no_soln1, no_soln2]

for grid in sudoku_grids:
    start = time.time()
    print(solve_sudoku(grid))
    end = time.time()
    print("%s seconds" %(end - start))
```

./Code/SudokuSolverBis.py

### E.3.6 Sudoku Randomizer

```
import random

from SudokuChecker import check_sudoku

def random_sudoku():
    sudoku = []
    bias = -0.2

    for i in range(9):
        sudoku.append([])

    for row in sudoku:
        for i in range(9):
            if (random.random() < 0.5 + bias):
                row.append(random.randint(1,9))
                bias -= 0.35
            else:
                row.append(0)
                bias += 0.05

    bias = -0.2

    return sudoku

for i in range(10000):
    print(check_sudoku(random_sudoku()))
```

./ProblemSets/ProblemSet3/PS3.2/SudokuRandomizer.py

## E.4 Fuzzer

```
#!/usr/bin/python

# 5-line fuzze below is from Chalie Miller's
# "Babysitting an Army of Monkeys":
# Part 1 - http://www.youtube.com/watch?v=Xnwodi2CBws
#
# Part 2 - http://www.youtube.com/watch?v=lK5fgCvS2N4


# List of files to use as initial seed
file_list = ["C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/AutomatedWhiteBoxTesting.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/BugTriage.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS2_1.PNG",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS2_2.PNG",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS3_1.PNG",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_Sudoku_Randomizer.PNG",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/CoverageTesting.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/DefensiveCoding.png",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/DriverScript.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/EquivalenceClasses.png",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/FuzzingTimeline.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/GUIApplication.png",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/InputOutput.png",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/InputValidity.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/NotCovered_Sudoku_Randomizer.PNG",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/RandomSystemTester.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/RandomTesting.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Rep_RandomTesting_1.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Rep_RandomTesting_2.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestCase_Reduction.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestingMethodes.png",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestingPractice.jpg",
             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TuningProbability.jpg"] # "x.pdf"
```

```python

# List of applications to test
apps = "C://Program Files/GIMP 2/bin/gimp-2.10.exe" # "/path/application"

fuzz_output = "fuzz.jpg"

FuzzFactor = 100
num_tests = 2880

bugs = {}
bugsOcc = {}

########## end configuration ##########

import math
import random
import string
import subprocess
import time
import sys

log = "-" * 100
log += "\n"
log += ' '*((100 - len("----------- START FUZZING -----------") - 1)/2)
log += "----------- START FUZZING -----------"
log += "\n"
log += "-" * 100
log += "\n"

fh = open("./log.txt", 'w')

fh.close()

start = time.time()

for i in range(num_tests):
    file_choice = random.choice(file_list)

    if file_choice[-4:] == ".jpg":
        fuzz_output = fuzz_output[:-4] + ".jpg"
    elif file_choice[-4:] == ".png":
        fuzz_output = fuzz_output[:-4] + ".png"
    elif file_choice[-4:] == ".PNG":
        fuzz_output = fuzz_output[:-4] + ".PNG"

    app = apps

    buf = bytearray(open(file_choice, 'rb').read())

    # Start Charlie Miller code
    numwrites = random.randrange(math.ceil((float(len(buf)) / FuzzFactor))) + 1

    # Because of Tuning Rules and Probabilities
    fuzz_choice = random.choice(['start', 'end', 'middle', 'random'])
    begin = None
    if fuzz_choice == 'start':
        begin = 0
    if fuzz_choice == 'end':
        begin = len(buf)-numwrites-1
```

```
    if fuzz_choice == 'middle':
        begin = random.randrange(len(buf)-numwrites)

    for j in range(numwrites):
        rbyte = random.randrange(256)

        if begin is None:
            rn = random.randrange(len(buf))
        else:
            rn = begin
            begin += 1

        buf[rn] = "%c"%(rbyte)

    # End Charlie Miller code

    open(fuzz_output, 'wb').write(buf)

    print("Fuzzed App : %s \n File: %s Fuzz Type: %s #writes=%d" % (app,
   file_choice, fuzz_choice, numwrites))

    process = subprocess.Popen([app, fuzz_output],stdout=subprocess.PIPE, stderr
   =subprocess.PIPE)

    time.sleep(10)
    crashed = process.poll()

    if not crashed:
        process.terminate()

        errout = process.stderr.read()

        if not errout and FuzzFactor:
            FuzzFactor -= 100
        elif FuzzFactor > 100:
            FuzzFactor += 100

        if FuzzFactor >= 100:
            FuzzFactor = 101


        code = ""
        flag1 = -1
        flag2 = -1
        flag3 = -1
        flag4 = -1
        for i in range(len(errout)):
            code += errout[i]

            if flag1 == -1 and errout[i] in "0123456789" and code[i -15:-1] == "
   (script-fu.exe:":
                flag1 = i - 15

            if flag1 != -1 and flag2 == -1 and errout[i] == ':':
                flag2 = i

            if flag2 != -1 and flag3 == -1 and errout[i] in "0123456789":
                flag3 = i - 24

```

```
            if flag3 != -1 and flag4 == -1 and errout[i] == ':' and errout[i+1] == ' ':
                flag4 = i

        err_tmp = errout[:flag1 - 1]
        err_code = errout[flag1:flag2]
        err_time = errout[flag3:flag4]
        err_msg = errout[flag4 + 1:]

        checkList = []

        for meta in bugs.values():
            checkList.append(meta[0])

        if bugs == {} or (err_tmp not in checkList and file_choice not in bugs.keys()):
            bugs[file_choice] = [[err_tmp, app, fuzz_choice, numwrites, err_code, err_time, err_msg]]
        elif err_tmp not in checkList and file_choice in bugs.keys():
            bug_tmp = bugs[file_choice]
            bug_tmp.append([err_tmp, app, fuzz_choice, numwrites, err_code, err_time, err_msg])
            bugs[file_choice] = bug_tmp
        elif err_tmp in checkList and file_choice not in bugs.keys():
            bugs[file_choice] = [[err_tmp, app, fuzz_choice, numwrites, err_code, err_time, err_msg]]

        if err_tmp not in bugsOcc.keys():
            bugsOcc[err_tmp] = 1
        else:
            bugsOcc[err_tmp] += 1

end = time.time()

print("Number of Process : %i"%(num_tests))

log += "-"*100
log += "\n"


for key in bugsOcc:
    for value in bugs:
        for error in bugs[value]:
            if key in [error][0]:
                middleValue = int(math.ceil(len(value)/2))
                value1 = value[:middleValue - 1]
                value2 = value[middleValue:]
                log += "Fuzzed App : %s \nFile: %s\n%s Fuzz Type: %s #writes=%d\n"%(error[1], value1, value2,

        error[2], error[3])

    log += "-"*100
    log += "\n"

    log += "ERROR :\n\n %s\n\n"%(key)

    for value in bugs:
        for error in bugs[value]:
```

```
                if key in [error][0]:
                    log += "%s:%s:%s"%(error[4], error[5], error[6])


        log += "Occurs : %i"%(bugsOcc[key])

        log += "\n"
        log += "-" * 100
        log += "\n"

fh = open("./log.txt", 'a')

fh.write(log)

fh.close()

log = "-" * 100
log += "\n"
log += ' '*((100 - len("xx.xxxxxx sec - xxxxxx Tests"))/2)
log += "%f sec - %i Tests"%((end - start), num_tests)
log += "\n"
log += ' '*((100 - len("---------- END FUZZING ----------"))/2)
log += "---------- END FUZZING ----------"
log += "\n"
log += "-" * 100
log += "\n"

fh = open("./log.txt", 'a')

fh.write(log)

fh.close()
```

./ProblemSets/ProblemSet4/Fuzzer.py

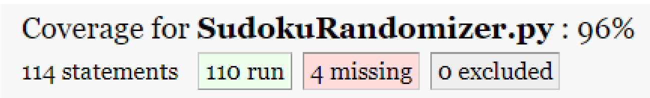


FIGURE E.7 – Coverage

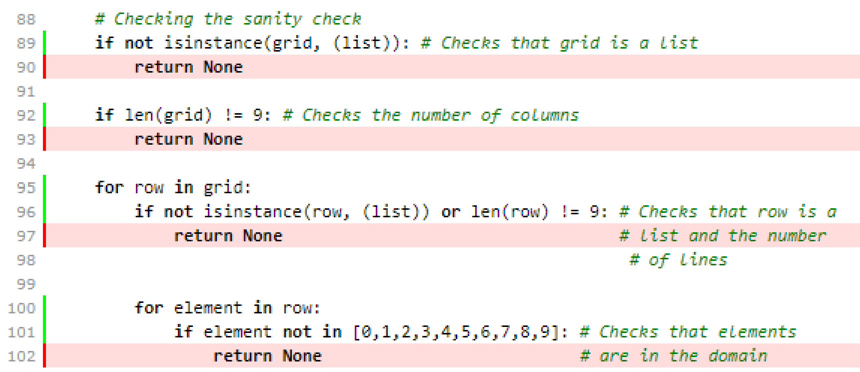


FIGURE E.8 – Coverage