



McGill
UNIVERSITY

ECSE 429 Software Validation
Fall 2017
Project Part 3

Group 10

Michael MAATOUK	260554267
Lucien GEORGE	260571775
Alexander BRATYSHKIN	260684228
Nathan LAFRANCE-BERGER	260686487
Benjamin WILLMS	260690005

Table of Contents

Table of Contents	2
Description of Testing Order	3
Environment Setup	4
Installation of Project Dependencies	4
Running the Tests	4
Accessing Code Coverage Reports	4
Discussion of Chosen Whitebox Technique & Coverage	5
Coverage Report Discussion	7
ChainedClosure	8
BagUtils	8
TreeBidiMap	8
Defects	10
Appendix	12
Test Cases	12
ChainedClosure Tests	12
TreeBidiMap Tests	13
BagUtils Tests	27
Control Flow Graphs	29
Graph 1 - rotateLeft	29
Graph 2 - insertValue	30
Graph 3 - doRedBlackInsert	31
Graph 4 - rotateRight	32
Graph 5 - doPut	33
Graph 6 - checkNonNullComparable	34
Graph 7 - lookup	35
Graph 8 - doEquals	36
Graph 9 - nextSmaller	37
Graph 10 - nextGreater	38
Graph 11 - doRedBlackDeleteFixup	39
Graph 12 - doRedBlackDelete	40
Graph 13 - swapPosition	41

Description of Testing Order

For the third part of the project we were tasked to test the three following classes: BagUtils, ChainedClosure, and TreeBidiMap. We divided the aforementioned classes in three suites: BagUtilsTest, ChainedClosureTest, and TreeBidiMapTest. All other classes in the Apache Commons Collections are assumed to have already been tested as mentioned in the project's description.

As explained in project part 2, BagUtils depends on Bag and SortedBag. As SortedBag extends Bag, we can use a SortedBag instance to test all the methods in BagUtils. TreeBag "is the standard implementation of a sorted bag" according to the apache commons collections documentation, and so it was chosen as the single class used to replace the necessary stub/mock to test BagUtils (we can assume TreeBag is already tested). Additionally, regarding tests for ChainedClosure, we had to use a stub for CatchAndRethrowClosure. As we showed in the previous deliverable of the project, ChainedClosure, SwitchClosure and CatchAndRethrowClosure should all be tested on the same level. Finally, our tests for TreeBidiMap did not require the use of any stubs or drivers.

Running JUnit tests is convenient in a sense that test drivers do not need to be implemented. The only requirement is to write the tests. JUnit provides two test drivers: one character-based and one GUI-based. The GUI-based test driver allows you to specify the test class to run. It would then test all the methods whose names begin with "test".

In our project we decided to use Maven which is a Java tool providing a uniform build system. Maven allowed us to test and integrate all of our test suites using a single command, "maven test". The java tool takes care of builds, dependency management, and documentation creation to name a few.

Environment Setup

In order to run the tests, the user must have Apache Maven installed in her system. Maven is a software project management and comprehension tool similar to Gradle and SBT. Please refer to this comprehensive and official guide for instructions on how to get the tool setup on a system across multiple operating systems. Prof. McIntosh said that we can assume that the grader will have Maven installed in her system, and thus we should not specifically state the steps required for the installation of the tool.

Installation of Project Dependencies

Once you have cloned our tests' repository on your local machine (which should ideally be found by following [this link](#)), navigate to the root folder of the project through your command prompt and run the command **mvn install**. In case this step doesn't work, try running **mvn clean && mvn install** instead. This should install all dependencies that are required in order to run the tests correctly. This step is absolutely necessary if you want to run the tests of our project.

Running the Tests

After installing the project dependencies, to run the tests, simply execute the **mvn test** command inside of the root folder of this project from your command prompt. The tests' results should appear promptly in your command line.

Accessing Code Coverage Reports

To access the code coverage reports generated by Jacoco, you must first run the tests (see above). Once the tests have been run, navigate, from the root of the project's directory, to the path "target/site/jacoco." From there, you can open the file "index.html" to access the coverage report for the whole of the Apache Commons Collection. You can then navigate to the appropriate classes just like you would inside of a Javadoc to find the respective coverages for ChainedClosure, TreeBidiMap and BagUtils.

Discussion of Chosen Whitebox Technique & Coverage

First of all, we decided to start off the project by opting for control flow-oriented techniques, as opposed to data flow or mutation-based approaches. The reason for this is quite simple: the control flow graph struck a perfect balance between usefulness and complexity. Analyzing how control flowed within the three respective classes was, albeit sometimes challenging, rather doable until a certain point (more on the limitations below). It also provided us with an efficient way of tracking down dependencies and calls between intra-class sub-classes and methods, which eventually helped us devise test cases that could achieve an optimal coverage. In contrast, data flow-oriented approaches would've practically been infeasible, since we would have to trace definitions and uses of a method from as far as its first call in different contexts for the case of a class like `TreeBidiMap`, making the whole strategy seem very inefficient and tedious in exchange of adding very little if any rigorousness to our tests in comparison to control flows. Although mutation testing might have seemed appealing at first, we realized that it would be too cumbersome and very time consuming to create a large number of mutants for a codebase as involved as the Apache Commons Collection within the time frame that was allocated for this project. We elaborate on how we drew our control flow graphs a few paragraphs below.

For the unit testing part of this project, we chose several different criteria that we would use depending on the structure and nature of the unit under test. For simple, straightforward methods that do not contain many branches such as `get` and `set` methods, we decided to use statement coverage. The reasoning behind this decision stems from the fact that the method functions very similarly for all range of inputs and so testing every statement seems a reliable way to determine in few test cases if the unit functions as described. For methods with more possible paths we opted for branch coverage in order to once again test as many statements as possible. With branch coverage, we aim to ensure that each possible branch is from each condition or decision point is executed at least once to therefore ensure that we cover as much of the reachable code as possible. Every branch was evaluated at its true and false values, which helped us ensure that no branch entailed any form of abnormal behaviour in the application. An interesting obstacle we encountered was the fact that many of the public methods of this type executed code from inaccessible private methods in which most of the branching occurred. In the instances where the public methods only role was passing the inputs directly to the private methods, extensive branch coverage could be achieved with relative ease. However, when the call to private methods was itself reliant on other branching within the public method, the units' structure complexity would become troublesome with regard to achieving total branch coverage.

We opted for not directly testing private methods because it is widely considered a bad practice. Private methods are non-existent to the rest of the application and, unless they're never used within the application (e.g. deprecated/dead code), they must be called by one of its respective class' public methods somewhere along the program's execution stack. Therefore, if we want to properly consider interactions between methods within our unit tests in addition to achieving complete coverage, we must choose a set of inputs for the public caller such that the private callee is fully executed so that we cover as many branches as we can. If we are to write explicit tests for the private tests, at best, we obtain redundancy in our tests (because a proper testing of the public methods implies full coverage of the private method it calls) and, at worst, we do not map our method interactions properly. If we are to exclusively write concrete tests for private methods, we will end up obfuscating the implementation details of the program and specific interactions / intra-class dependencies that occur within it.

In addition, we used the black box technique known as Equivalent Class testing in order to cover the behaviour found in the methods which segregated null inputs from normal inputs. This latest behaviour would often raise an exception as null inputs would correspond to an invalid input. In the pursuit of our coverage goals, we aimed to raise every exception once per public method. Even though the exception would often be raised by a

private method called from several different sources, this allowed us to verify that the handling of errors was done appropriately throughout the calls.

We decided on a bottom-up approach to avoid reliance on untested material. Therefore, we developed our testing order to first cover the more basic methods that do not depend on any other code to be covered. Following this we would test the methods that relied only on this simpler, now tested material, so on and so forth until our coverage goals would be met. However, as mentioned above, we were restricted in some cases by the accessors of some of the methods and, therefore, couldn't trace a full hierarchy of inter-class dependencies by only accounting for publicly accessible functions. Evidently, for some classes (e.g. `TreeBidiMap`) most of the low-level methods (i.e. lowest reliability on other components) were not accessible outside of the class, and thus we did not test them directly. Nonetheless, we tried to keep a sense of consistency by applying this logic to the testing public methods as thoroughly as we could, i.e. we would first start by testing the public methods relying on the fewest dependencies, and then move our way up to the more complex and interactive public methods. For instance, we would make sure to test the method `put` in `TreeBidiMap` before testing `putAll`, since the latter relies on the former for its functionality. Also, there were cases where it would be simply impossible to follow any sort of modular approach, because even the constructor would contain a call to another method, which we wouldn't even be able to test without first instantiating the class! To compensate for this, we mentioned in the comments for the relevant tests that these tests' results should be cross referenced with other ones (also specified) to determine the location of errors, if any exist.

Some testing methods such as path coverage and modified condition weren't opted. This is due to the complexity that such tests would incur for a relatively straight forward piece of code. In the case of path coverage, the tests would lead to a large number of paths that would take too much time to implement with not enough coverage to compensate for the effort. Moreover the latter reasoning can also be applied to the modified condition method. By applying each condition to be at least once true and at least once false would require $n+1$ test cases. However the conditions present in the tested code were plain and does not require such as extensive test method.

The control flow graphs' main purpose is to keep track of the different branches of the tested code. The main criteria that decided which methods would be included in the flow graphs' is that it would be helpful in obtaining the most graph coverage possible. However, due to the complexity of the interdependencies only the methods that were reachable before the code became too complex were drawn. The complexities that were faced were mainly from the nested code within other classes. This would lead to large graphs that take a lot of time to deduce the proper test cases. Therefore, the graphs were limited to a maximum amount of four nested function calls to minimize the cognitive complexity of the handled segments. Moreover, due to the simplicity of some other branches it was decided to follow a minimum of four conditions within a method so as to not waste time in drawing graphs that would be of little use to our testing endeavours. This procedure excluded all methods from `BagUtils` and `ChainedClosures` but heavily focuses on `TreeBidiMap`. You can find the specific graphs we have drawn in the appendix section of this report.

Coverage Report Discussion

We aimed to achieve at least full branch coverage on all public methods in each of the three classes under test. In the case of BagUtils, this included testing all methods except the private constructor. The comments in the Apache Commons Collections say that initialization of BagUtils is unnecessary and unintended, so testing the constructor was deemed unnecessary. To reach full branch coverage for these methods, statement coverage was enough, as they contain no conditions, and only one branch each. ChainedClosure was slightly more complex than BagUtils, as the methods in ChainedClosure contained some conditions/branches that needed to be tested. To test these methods, condition/decision coverage was used. This ensured that at least coverage of all branches was achieved (as per our goal), while offering the possibility of more comprehensive testing. The tests for TreeBidiMap used a range of techniques to reach full branch coverage. The methods with single branches used statement coverage, while the rest used either branch or condition/decision coverage. All techniques resulted in tests fulfilling full branch coverage for the methods tested.

It must be noted that the coverage tool used points out some missed methods and branches. This is because not all methods and branches can be easily reached through the use of the public methods provided. For example, on and near line 522 of TreeBidiMap, it says:

```
if (cmp == 0) {
    // shouldn't happen
    throw new IllegalArgumentException("Cannot store a duplicate key (\\"" + key + "\") in
this Map");
...

```

This shows that some branches can't or shouldn't be reached, and so they were ignored in our test design. These branches still exist, and so the coverage tool points out that they were missed. There are other conditions in TreeBidiMap that are marked as "dead code?" which further solidifies our reasons for missing some branches. Further, there were many pieces of code lying within inner classes that we didn't test. This was because it was very difficult to trace how to specifically access these pieces of code, as they lie beneath large layers of convoluted code. It was difficult to determine precise inputs to public methods that would allow us to reach these inner pieces.

Additionally, one has to take into consideration other types of branches that will never be called because some parts of the code already take into account the same conditions somewhere previously within the stack trace. For example, take a look at the method copyColor on line 735: we check for the case of a node whose color we're copying isn't null. However, the only usages of the method is found inside of the doRedBlackDeleteFixup method on lines 1063 and 1095, in which we already have ensured that we're dealing with a node whose parent isn't null! Therefore, there is a redundancy in the checks, which unfortunately further draws down our branch coverage metric. It is understandable nonetheless why these pieces of code are necessary -- someone might decide to reuse the method later on from within another context in which the "dormant" branch finally can be utilized.

Another point that should be noted is that the private methods which are accessible through public ones were considered as parts of the public methods calling them. That is, they were treated as being "copy-pasted" into the public methods, and then analyzed using the coverage criteria chosen for the public method. In this way, we ensured that all accessible private methods were tested, and reached full branch coverage.

Further, some methods, like the ones found in the Inverse inner class, were simply wraps of the methods tested in TreeBidiMap (all they do is call the method with a different input than is used in TreeBidiMap).

As such, we found that testing these methods would be repetitive, so they were omitted. The method “removeValue” in the inner class Inverse is an example of this. Due to our decision to ignore these methods, our reported coverage was negatively impacted.

The reports below were gathered using Jacoco. To view how to access these reports directly after running tests, please refer to the first section of the report The green bars represent tested parts, while the red represents missed parts.

ChainedClosure

100% branch coverage was met, fulfilling our goal.

org.apache.commons.collections4.functors

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
NotNullPredicate		78%		100%	2	6	2	6	2	5	0	1
ChainedClosure		100%		100%	0	12	0	24	0	6	0	1

BagUtils

The missed lines, methods, and instructions correspond to the private constructor mentioned earlier. It is likely that the “n/a” under branches is due to the single branch nature of everything in this class. This means that 100% branch coverage was achieved for all tested methods.

org.apache.commons.collections4







Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BagUtils		94%	n/a	n/a	1	13	1	15	1	13	0	1

TreeBidiMap

As seen in our coverage reports, the majority of methods were tested with full branch coverage, however there are those that don’t reach that goal. The missing branches all correspond to the aforementioned notable points.

In total, TreeBidiMap was tested with 78% branch coverage, with most of the missing branches corresponding to unreachable or dead code.

org.apache.commons.collections4.bidimap

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TreeBidiMap		85%		78%	52	193	63	465	2	65	0	1
AbstractDualBidiMap		0%		0%	34	34	64	64	25	25	1	1
DualTreeBidiMap		0%		0%	26	26	60	60	20	20	1	1

TreeBidiMap

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
doRedBlackDeleteFixup(TreeBidiMap.Node, TreeBidiMap.DataElement)		69%		64%	7	12	15	47	0	1
swapPosition(TreeBidiMap.Node, TreeBidiMap.Node, TreeBidiMap.DataElement)		77%		64%	12	19	12	52	0	1
readObject(ObjectInputStream)		0%		0%	2	2	8	8	1	1
writeObject(ObjectOutputStream)		0%		0%	2	2	7	7	1	1
doRedBlackInsert(TreeBidiMap.Node, TreeBidiMap.DataElement)		90%		75%	5	11	4	36	0	1
insertValue(TreeBidiMap.Node)		79%		88%	1	5	2	20	0	1
doRedBlackDelete(TreeBidiMap.Node)		91%		88%	3	13	2	28	0	1
doPut(Comparable, Comparable)		89%		90%	1	6	1	34	0	1
doEquals(Object, TreeBidiMap.DataElement)		74%		67%	4	7	8	20	0	1
nextSmaller(TreeBidiMap.Node, TreeBidiMap.DataElement)		71%		62%	2	5	2	11	0	1
doToString(TreeBidiMap.DataElement)		94%		80%	2	6	0	18	0	1
getMapIterator(TreeBidiMap.DataElement)		81%		67%	1	3	1	4	0	1
copyColor(TreeBidiMap.Node, TreeBidiMap.Node, TreeBidiMap.DataElement)		69%		50%	2	3	1	5	0	1
getParent(TreeBidiMap.Node, TreeBidiMap.DataElement)		75%		50%	1	2	0	1	0	1
getRightChild(TreeBidiMap.Node, TreeBidiMap.DataElement)		75%		50%	1	2	0	1	0	1
getLeftChild(TreeBidiMap.Node, TreeBidiMap.DataElement)		75%		50%	1	2	0	1	0	1
rotateLeft(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	4	0	13	0	1
rotateRight(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	4	0	13	0	1
nextGreater(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	5	0	11	0	1
lookup(Object, TreeBidiMap.DataElement)		100%		100%	0	4	0	10	0	1
checkNonNullComparable(Object, TreeBidiMap.DataElement)		100%		100%	0	3	0	5	0	1
doHashCode(TreeBidiMap.DataElement)		100%		100%	0	3	0	8	0	1
putAll(Map)		100%		100%	0	2	0	4	0	1
clear()		100%		n/a	0	1	0	5	0	1
firstKey()		100%		100%	0	2	0	3	0	1
lastKey()		100%		100%	0	2	0	3	0	1
TreeBidiMap()		100%		n/a	0	1	0	6	0	1
nextKey(Comparable)		100%		100%	0	2	0	3	0	1
previousKey(Comparable)		100%		100%	0	2	0	3	0	1
leastNode(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	3	0	5	0	1
greatestNode(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		75%	1	3	0	5	0	1
doRemoveKey(Object)		100%		100%	0	2	0	5	0	1
doRemoveValue(Object)		100%		100%	0	2	0	5	0	1
get(Object)		100%		100%	0	2	0	3	0	1
getKey(Object)		100%		100%	0	2	0	3	0	1
keySet()		100%		100%	0	2	0	3	0	1
values()		100%		100%	0	2	0	3	0	1
entrySet()		100%		50%	1	2	0	3	0	1
inverseBidiMap()		100%		50%	1	2	0	3	0	1
mapIterator()		100%		100%	0	2	0	3	0	1
containsKey(Object)		100%		100%	0	2	0	2	0	1
containsValue(Object)		100%		100%	0	2	0	2	0	1
put(Comparable, Comparable)		100%		n/a	0	1	0	3	0	1
isRed(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	3	0	1	0	1
isBlack(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		100%	0	3	0	1	0	1
grow()		100%		n/a	0	1	0	3	0	1
shrink()		100%		n/a	0	1	0	3	0	1
getGrandParent(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		n/a	0	1	0	1	0	1
isEmpty()		100%		100%	0	2	0	1	0	1
modify()		100%		n/a	0	1	0	2	0	1
TreeBidiMap(Map)		100%		n/a	0	1	0	3	0	1
makeRed(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		50%	1	2	0	3	0	1
makeBlack(TreeBidiMap.Node, TreeBidiMap.DataElement)		100%		50%	1	2	0	3	0	1
equals(Object)		100%		n/a	0	1	0	1	0	1
lookupKey(Object)		100%		n/a	0	1	0	1	0	1
lookupValue(Object)		100%		n/a	0	1	0	1	0	1
checkKeyAndValue(Object, Object)		100%		n/a	0	1	0	3	0	1
remove(Object)		100%		n/a	0	1	0	1	0	1
removeValue(Object)		100%		n/a	0	1	0	1	0	1
hashCode()		100%		n/a	0	1	0	1	0	1
toString()		100%		n/a	0	1	0	1	0	1
compare(Comparable, Comparable)		100%		n/a	0	1	0	1	0	1
checkKey(Object)		100%		n/a	0	1	0	2	0	1
checkValue(Object)		100%		n/a	0	1	0	2	0	1
size()		100%		n/a	0	1	0	1	0	1
Total	299 of 2,003	85%	55 of 255	78%	52	193	63	465	2	65

Defects

We were able to track down a few sections of the code which are prone to cause bugs if left untreated given interactions that could possibly arise as the codebase of the Apache Commons Collection gets bigger. Additionally, we were able identify statements which could be classed as code smells or that did not follow the best practices / styling standards that are to be expected from such a widely-used tool as the Apache Commons Collection. These will be discussed below in the form of bullet-points in order of importance.

- **[Bugs]** On lines 326 (inside of method `firstKey()`), 340 (inside of the method `lastKey()`), 2127 (inside of the overridden `firstKey()` method) and 2135 (inside of the overridden `lastKey()` method) for the `TreeBidiMap` class, there is a very high risk that an unexpected `NullPointerException` might be thrown due to the fact that the nullity case is not handled and could potentially lead to harm when `VALUE.ordinal()` might be null. Since the logic behind `VALUE.ordinal()` could entail a returned null value, it is best to handle that case explicitly before trying to access a null index for array `rootNode`, like it is done in several other methods. For example, `VALUE.ordinal()` could've been called on a separate line (à la lines 2148 to 2148), so that the null return could directly be treated. Additionally, one could've surrounded the statements inside of a try/catch clause to take care of the particular exception. Accessing a reference to null is a rather critical mistake, because it could, in the best of cases, cause abrupt termination and, at worst, could provide attackers with useful debugging information that could lead to malicious attacks or could help hackers bypass security measures. Since the Apache Commons Collection isn't a safety critical system and is open source, the latter scenarios are unlikely to directly affect it. However, other safety critical systems might be relying on it, and therefore attackers could exploit these vulnerabilities through stack traces. It is always better to be precautionous, envision future changes that might happen and cover as many corner scenarios as possible inside of code so as to not introduce critical runtime bugs ahead in the development cycle. Anticipation is key.
- **[Code smell]** The string literal "Map is empty" is repeated 4 times across `TreeBidiMap`. Therefore, it should probably be extracted as a constant. Duplicate string literals (or duplicate code in general) make the process of refactoring more prone to errors due to the fact that we have to ensure that all occurrences of the duplicate strings have been updated, which is a very hard thing to keep track of within such an enormous codebase. However, constants can be referenced from many places and have to only be edited once if need be.
- **[Code smell]** A lot of explicit comparisons like "`== false`" are present across the code (for instance, on lines 1345, 1358, 1560, 1571, 1601, 1612) inside of the `TreeBidiMap` class. This, of course, doesn't have any impact on the functionality of the code and is a pretty minor mistake, but should definitely be pointed out within the context of such a reputable library as the Apache Commons Collection. Redundant boolean literals should ideally be removed because they affect code readability.
- **[Code smell / Bad practice]** In some parts of the code, the cognitive complexity measure is very high, making it rather hard to understand and maintain certain parts of the code that might need to be touched in the future. For instance, take a look at a method like `doRedBlackInsert` on line 898 of the `TreeBidiMap` class. Not only does it have several calls to other methods from the class (which, as mentioned, are very hard to trace), but it also doesn't provide any meaningful comments. The Javadoc comment of the method literally that "complicated red-black insert stuff" (line 892) is going on inside of this method, without further delving into how the method actually actually is supposed to work and what each method is supposed to do. There aren't any other comments explaining the nested conditions either, aside from two sections labelled as "dead code?". This, clearly, makes it even harder to browse and comprehend the intended functionality of the code, as well as tracing out dependencies in an already complex method. The same case could be made against other methods as well.

- **[Code smell]** Some parts of the class inside of the TreeBidiMap class are literally inaccessible and therefore unused (e.g. line 915). Not only does this reduce our coverage metrics, as discussed in the section above, but it also puts burden in several other ways. For instance, not only did we have to understand very complex code within the doRedBlackInsert method, but we also had to waste time on understanding the unreachable parts. Additionally, maintaining such code is also a pain, and the danger that someone might make a change that will inadvertently trigger the non-tested, previously unreachable code is always a possibility. Future merges will have to take into consideration these unused pieces of code to prevent any unintended behaviour, which also ends up in the spending of extra resources. Obviously, as far as the functionality of the code is concerned, this is not likely to be a major concern.
- **[Code smell]** On line 1469, the sub-class View does not override the equals method. We can see that View implements its own fields, yet does not provide a way of testing these for equality. If we are to use this class somewhere else, we run the risk of having two non-equivalent instances of the class being seen as equal because only the superclass fields will be considered for the equality check. However, upon our inspection of the code, nowhere did it seem to be an issue, since the equals method for the sub-class was never called anyway.

Appendix

Test Cases

ChainedClosure Tests

ID	Input	Test name	Lines covered (Path)	Conditions	Category
1.01	{switchClosure, chainedClosure, catchAndRethrowClosure}	<i>shouldReturnClosures()</i>	122-123	-	Statement coverage
1.02	{switchClosure, chainedClosure, catchAndRethrowClosure}	<i>shouldReturnNewChainedClosureWithMultipleClosuresFactoryOverload()</i>	46-47-48-50-51	48: closures.length > 0	Condition coverage
1.03	-	<i>shouldReturnEmptyClosureWithMultipleClosuresFactoryOverload()</i>	46-47-48-49	48: closures.length == 0	Condition coverage
1.04	Array.asList(switchClosure, chainedClosure, catchAndRethrowClosure)	<i>shouldReturnNewChainedClosureWithCollectionFactoryOverload()</i>	66-67-70-74-75-76-77-78-79-80	67: closures != null 70: closures.size() > 0	Condition coverage
1.05	null	<i>failWithNullPointerExceptionWithNullClosureWithCollectionFactoryOverload()</i>	66-67-68	68: closures == null	Condition coverage
1.06	Empty ArrayList	<i>shouldReturnEmptyClosureWithCollectionFactoryOverload()</i>	66-67-70-71-72	67: closures != null 70: closures.size() == 0	Condition / decision coverage
1.07	Array.asList(switchClosure, chainedClosure, catchAndRethrowClosure)	<i>shouldExecuteClosure()</i>	110-111-112-113-114	-	Statement coverage

TreeBidiMap Tests

hMap is a HashMap filled with {(1, "aaa"),(2, "bbb"),(3, "ccc"),(4, "ddd"),(5, "eee"),(6, "fff"),(7, "ggg")}

ID	Input	Test name	Lines/private methods covered (Path)	Conditions	Category
2.01	-	shouldCreateEmptyTreeBidiMap()	115-116-117-118	-	Statement
2.02	hMap	shouldCreateTreeBidiMapFromMapInput()	128-129-130-131	-	Statement
2.03	null	shouldFailTreeBidiMapWithNullPointerException()	128-129-130-131	-	Statement
2.04	TreeBidiMap(hMap)	shouldReturnSizeOfMap()	140-141-142	-	Statement
2.05	TreeBidiMap(hMap)	shouldClearMap()	270-271-modify-272-273-274-275-276	-	Statement
2.06	TreeBidiMap(hMap), and 1	shouldReturnKeyIsContainedFirst()	165-checkKey-checkNonNullComparable-lookupKey-lookup	o != null, o instanceof Comparable, node != null, cmp == 0	Branch
2.07	TreeBidiMap(hMap), and 7	shouldReturnKeyIsContainedLast()	165-checkKey-checkNonNullComparable-lookupKey-lookup	o != null, o instanceof Comparable, node != null, cmp != 0, cmp == 0	Branch
2.08	TreeBidiMap(hMap), and 50	shouldReturnKeyIsNotContained()	165-checkKey-checkNonNullComparable-lookupKey-lookup	o != null, o instanceof Comparable, node != null, cmp != 0, node == null	Branch
2.09	TreeBidiMap(hMap), and null	shouldFailContainsKeyWithNullPointerException()	165-checkKey-checkNonNullComparable	o == null	Branch
2.10	TreeBidiMap(hMap), and int[5]	shouldFailContainsKeyWithClassCastException()	165-checkKey-checkNonNullComparable	o != null, !(o instanceof Comparable)	Branch

2.11	TreeBidiMap(hMap), and “aaa”	<i>shouldReturnValueIsContainedFirst()</i>	181-checkValue-checkNonNullComparable-lookupValue-lookup	o != null, o instanceof Comparable, node != null, cmp == 0	Branch
2.12	TreeBidiMap(hMap), and “ggg”	<i>shouldReturnValueIsContainedLast()</i>	181-checkValue-checkNonNullComparable-lookupValue-lookup	o != null, o instanceof Comparable, node != null, cmp != 0, cmp == 0	Branch
2.13	TreeBidiMap(hMap), and “zzz”	<i>shouldReturnValueIsNotContained()</i>	181-checkValue-checkNonNullComparable-lookupValue-lookup	o != null, o instanceof Comparable, node != null, cmp != 0, node == null	Branch
2.14	TreeBidiMap(hMap), and null	<i>shouldFailContainsValueWithNullPointerException()</i>	181-checkValue-checkNonNullComparable	o == null	Branch
2.15	TreeBidiMap(hMap), and int[5]	<i>shouldFailContainsValueWithClassCastException()</i>	181-checkValue-checkNonNullComparable	o != null, !(o instanceof Comparable)	Branch
2.16	TreeBidiMap()	<i>shouldReturnEmptyBracesAsString()</i>	490-doToString	nodeCount == 0	Branch
2.17	TreeBidiMap(hMap)	<i>shouldReturnMapInBracesAsString()</i>	490-doToString	nodeCount != 0, hasNext, !hasNext	Branch
2.18	TreeBidiMap()	<i>shouldReturnMapIsEmpty()</i>	150-151	nodeCount == 0	Condition
2.19	TreeBidiMap(hMap)	<i>shouldReturnMapIsNotEmpty()</i>	150-151	nodeCount != 0	Condition
2.20	TreeBidiMap()	<i>shouldReturnNewEntrySet()</i>	432-433-436	entrySet != null	Statement
2.21	TreeBidiMap(hMap)	<i>shouldReturnEntryView</i>	432-433-436	entrySet != null	Statement
2.22	TreeBidiMap()	<i>shouldReturnNullKeyNotPresentTreeEmpty</i>	199-200-201-lookupKey(key)-202-node.getValue()	node.getValue() == null	Branch
2.23	TreeBidiMap(hMap)	<i>shouldReturnValueFromKey()</i>	199-200-201-lookupKey(key)-202-node	node.getValue() != null	Branch

			e.getValue()		
2.24	TreeBidiMap(hMap)	<i>shouldReturnNullKeyNotPresent</i>	199-200-201-lookupKey(key)-lookup-202-node.getValue()	node.getValue() == null	Branch
2.25	TreeBidiMap(hMap) Key = int[] array	<i>shouldThrowClassCastExceptionKeyWrongType_get()</i>	199-200-201-lookupKey(key)-node.getData()-202-node.getValue()	o != null, !(o instanceof Comparable)	Each Exception
2.26	TreeBidiMap(hMap) Key = null	<i>shouldThrowClassCastExceptionNull_get()</i>	199-200-201-lookupKey(key)-node.getData()-202-node.getValue()	o == null	Each Exception
2.27	TreeBidiMap()	<i>shouldReturnNullValueNotPresentTreeEmpty()</i>	292-293-checkValue(value)-294-lookupValue(value)-295-node.getKey()	node.getKey() == null	Branch
2.28	TreeBidiMap(hMap)	<i>shouldReturnKeyFromValue()</i>	292-293-checkValue(value)-294-lookupValue(value)-295-node.getKey()	lookupValue != null	Branch
2.29	TreeBidiMap(hMap)	<i>shouldReturnNullValueNotPresent()</i>	292-293-checkValue(value)-294-lookupValue(value)-295-node.getKey()	node.lookupValue == null	Branch
2.30	TreeBidiMap(hMap) Value = int[] array	<i>shouldThrowClassCastExceptionValueWrongType_getKey()</i>	292-293-checkValue(value)-294-lookupValue(value)-295-node.getKey()	o != null, !(o instanceof Comparable)	Each Exception
2.31	TreeBidiMap(hMap) value = null	<i>shouldThrowClassCastExceptionValueNull_getKey()</i>	292-293-checkValue(value)-294-lookupValue(value)-295-node.getKey()	o == null	Each Exception
2.32	TreeBidiMap() Key = 1 Value = "new"	<i>shouldIncreaseSizeToOne</i>	230-231-get(key)-232-doPut(key, value)-233	Key != null Value != null Both instanceof comparable)	Branch
2.33	TreeBidiMap(hMap) Key = 9 Value = "new"	<i>shouldIncreaseSizeByOne()</i>	230-231-get(key)-232-doPut(key, value)-233	Key != null Value != null Both instanceof	Branch

				comparable	
2.34	TreeBidiMap(hMap) Key = 1 Value = "new"	<i>shouldReplaceExistingValue()</i>	230-231-get(key)-232-doPut(key, value)-doRemove(key)-233	Key == existing key Value != null Both instanceof comparable	Branch
2.35	TreeBidiMap(hMap) Key = 9 Value = "aaa"	<i>shouldReplaceExistingKey()</i>	230-231-get(key)-232-doPut(key, value)-doRemove(key)-233	Key != null Value == existing value Both instanceof comparable	Branch
2.36	TreeBidiMap() Value = Int[] array Value = "new"	<i>shouldFailThrowClassCastException_put()</i>	230-231-get(key)-232-doPut(key, value)-233	value != null, !(o instanceof Comparable)	Each Exception
2.37	TreeBidiMap() Key = null Value = "new"	<i>shouldFailThrowNullPointerException_put()</i>	230-231-get(key)-232-doPut(key, value)-233	Value == null	Each Exception
2.38	TreeBidiMap() Key = 1	<i>shouldReturnNullNoMappingToRemove</i>	262-doRemoveKey(key)-560	Node == null	Branch
2.39	TreeBidiMap(hMap) Key = 1	<i>shouldReturnValueOfMappingRemoved</i>	262-doRemoveKey(key)-563	Node != null	Branch
2.40	TreeBidiMap(hMap) Key = 34	<i>shouldReturnNullValueToRemoveNotMapped()</i>	262-doRemoveKey(key)-560	Node == null	Branch
2.41	TreeBidiMap(hMap) Key = int[] array	<i>shouldThrowClassCastException_remove()</i>	262-doRemove-lookupKey-lookup-getData-1902	Key not instance of comparator	Each Exception
2.42	TreeBidiMap(hMap) Key = null	<i>shouldFailThrowNullPointerException_remove()</i>	262-doRemove-lookupKey	Key == null	Each Exception
2.43	TreeBidiMap() TreeBidiMap()	<i>shouldNotAlterNewMap()</i>	244-245	Maps empty	Statement
2.44	TreeBidiMap(hMap) TreeBidiMap()	<i>shouldChangeMapSizeToOtherMapSize()</i>	244-245-246-put(key, value)	Key != null Value != null Both instanceof comparable) && list empty	Statement

2.45	TreeBidiMap(hMap) TreeBidiMap()	<i>shouldNotAlterNewNonEmptyMap()</i>	244-245	Input map is empty	Statement
2.46	TreeBidiMap(hMap) TreeBidiMap({8, "zzz"}, {9, "yyy"}, {10, "xxx"}, {11, "www"}, {12, "vvv"}, {13, "uuu"}, {14, "ttt"})	<i>shouldChangeSizeToSumOfMaps()</i>	244-245-246-put(key, val)	Both maps non empty with no overlapping	Statement
2.47	TreeBidiMap(hMap) TreeBidiMap(hMap) {3, "hello"} {4, "wazaap"}	<i>shouldResultInValueMappingChange()</i>	244-245-246-put(key, val)	Overlapping key	Statement
2.48	TreeBidiMap(hMap) TreeBidiMap(hMap) {34, "bbb"} {23, "aaa"}	<i>shouldResultInKeyMappingChange()</i>	244-245-246-put(key, val)	Overlapping value	Statement
2.49	TreeBidiMap(hMap) TreeBidiMap(hMap)	<i>shouldNotAlterMap_putAll()</i>	244-245-246-put(key, val)	All overlapping entries	Statement
2.50	TreeBidiMap() Value = "aaa"	<i>shouldReturnNullNoValueMappingToRemove()</i>	310-doRemoveValue-lookupValue(value)-569	node = null	Branch
2.51	TreeBidiMap(hMap) Value = "aaa"	<i>shouldReturnKeyOfMappingRemoved()</i>	310-doRemove-doRedBlackDelete()-572	node != null	Branch
2.52	TreeBidiMap(hMap) Value = "wrong"	<i>shouldReturnNullKeyToRemoveNotMapped()</i>	310-doRemoveValue-lookupValue(value)-569- 594-598	node == null Map not empty	Branch
2.53	TreeBidiMap(hMap) Value = int[] array	<i>shouldFailThrowClassCastException_removeValue()</i>	310-doRemoveValue-lookupValue(value)	Value not instance of comparable	All Exceptions
2.54	TreeBidiMap(hMap) Value = null	<i>shouldFailThrowNullPointerException_removeValue()</i>	310-doRemoveValue-lookupValue(value)	Value == null	All Exceptions
2.55	TreeBidiMap()	<i>shouldFailEmptyFirst</i>	322-327	nodeCount ==	Branch

		<i>Key()</i>		0	
2.56	TreeBidiMap()	<i>shouldReturnOneElementFirstKey()</i>	322-325-leastNode(rootNode[KEY.ordinal()], KEY).getKey()	nodeCount != 0	Branch
2.57	TreeBidiMap(hMap)	<i>shouldReturnFirstKey()</i>	322-325-leastNode(rootNode[KEY.ordinal()], KEY).getKey()	nodeCount != 0	Branch
2.58	TreeBidiMap()	<i>shouldFailEmptyLastKey()</i>	336-341	nodeCount == 0	Branch
2.59	TreeBidiMap()	<i>shouldReturnOneElementLastKey()</i>	336-339-greatestNode(rootNode[KEY.ordinal()], KEY).getKey()	nodeCount != 0	Branch
2.60	TreeBidiMap(hMap)	<i>shouldReturnLastKey()</i>	336-339-greatestNode(rootNode[KEY.ordinal()], KEY).getKey()	nodeCount != 0	Branch
2.61	TreeBidiMap(), 1	<i>shouldReturnEmptyNextKey()</i>	352-checkKey(key)-nextGreater(lookupKey(key), KEY)-355-356	node == null, o == null	Branch
2.62	TreeBidiMap(), null	<i>shouldFailEmptyNextKey()</i>	352-checkKey(key)-nextGreater(lookupKey(key), KEY)-355-356	node == null, o = null	Branch
2.63	TreeBidiMap(), 1	<i>shouldReturnOneElementNextKey()</i>	352-checkKey(key)-nextGreater(lookupKey(key), KEY)-node.getKey(key)-356	node == null, o == null	Branch
2.64	TreeBidiMap(hMap), 1	<i>shouldReturnNextKey()</i>	352-checkKey(key)-nextGreater(lookupKey(key), KEY)-node.getKey()-356	node != null, !o instanceof Comparable	Branch
2.65	TreeBidiMap(), 1	<i>shouldReturnEmptyPreviousKey()</i>	367-checkKey(key)-nextSmaller(lookupKey(key), KEY)-370	node == null, o == null	Branch
2.66	TreeBidiMap(),	<i>shouldFailEmptyPreviousKey()</i>	367-checkKey(key)	node == null, o	Branch

	null	<i>ousKey()</i>	-nextSmaller(looku pKey(key), KEY)-370	== null	
2.67	TreeBidiMap(), 1	<i>shouldReturnOneElem entPreviousKey()</i>	367-checkKey(key) -nextSmaller(looku pKey(key), KEY)-node.getKey ()-371	node == null	Branch
2.68	TreeBidiMap(hMa p), 1	<i>shouldReturnPrevious Key</i>	367-checkKey(key) -nextSmaller(looku pKey(key), KEY)-node.getKey ()-371	node != null	Branch
2.69	TreeBidiMap(), index, result, expected	<i>shouldReturnEmptyKe ySet()</i>	387-388-KeyView(KEY)-390-391-392	keySet == null	Branch
2.70	TreeBidiMap(), index, result, expected	<i>shouldReturnOneElem entKeySet()</i>	387-392	keySet != null	Branch
2.71	TreeBidiMap(hMa p), index, result, expected	<i>shouldReturnKeySet()</i>	387-392	keySet != null	Branch
2.72	TreeBidiMap(), index, result, expected	<i>shouldReturnEmptyVa lues()</i>	409-410-ValueVie w(KEY)-412-413-4 14	valuesSet == null	Branch
2.73	TreeBidiMap(), index, result, expected	<i>shouldReturnOneElem entValues()</i>	409-414	valuesSet != null	Branch
2.74	TreeBidiMap(hMa p), index, result, expected	<i>shouldReturnValues()</i>	409-414	valuesSet != null	Branch
2.75	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnInverseBi diMap()</i>	455-456-Inverse()- 458-459-460	inverse == null	Branch
2.76	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnOneElem entInverseBidiMap()</i>	455-460	inverse != null	Branch
2.77	TreeBidiMap(hMa p), TreeBidiMap()	<i>shouldReturnInverseBi diMap()</i>	455-460	inverse != null	Branch
2.78	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnEmptyEq uals()</i>	470-doEquals(obj, KEY)-472	-	Statement

2.79	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnOneElementEquals()</i>	470-doEquals(obj, KEY)-472	-	Statement
2.80	TreeBidiMap(hMap), TreeBidiMap(hMap)	<i>shouldReturnEquals()</i>	470-doEquals(obj, KEY)-472	-	Statement
2.81	TreeBidiMap(), 0	<i>shouldReturnEmptyHashCode()</i>	480--doHashCode(KEY)-482	-	Statement
2.82	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnOneElementHashCode()</i>	480--doHashCode(KEY)-482	-	Statement
2.83	TreeBidiMap(hMap), TreeBidiMap(hMap)	<i>shouldReturnHashCode()</i>	480--doHashCode(KEY)-482	-	Statement
2.84	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnEmptyEntrySet()</i>	432-437	entrySet == null	Branch
2.85	TreeBidiMap(), TreeBidiMap()	<i>shouldReturnOneElementEntrySet()</i>	432-437	entrySet != null	Branch
2.86	TreeBidiMap(hMap), TreeBidiMap(hMap)	<i>shouldReturnEntrySet()</i>	432-437	entrySet != null	Branch
2.87	TreeBidiMap()	<i>shouldReturnFalseOnEmptySet()</i>	441-446	-	Statement
2.88	TreeBidiMap(), “{1=test}”	<i>shouldReturnTrueOneElementInSet()</i>	441-446	-	Branch
2.89	TreeBidiMap(hMap)	<i>shouldReturnTrueOnIteratorTypeFullSet()</i>	441-446	-	Branch
2.90	TreeBidiMap()	<i>shouldReturnTrueOnIteratorTypeEmptySet()</i>	441-446	-	Branch

2.91	TreeBidiMap(hMap)	<i>shouldReturnProperInverseSize()</i>	454-460	-	Branch
2.92	TreeBidiMap(), “{“A”=“B”}”	<i>shouldReturnProperInverseElementOrder()</i>	454-460	-	Branch
2.93	TreeBidiMap(),	<i>shouldReturnProperInverseEmptySize()</i>	454-460	-	Branch
2.94	TreeBidiMap(), “{“A”=“B”}”	<i>shouldReturnProperInverseOneElementSize()</i>	454-460	-	Branch
2.95	TreeBidiMap(hMap)	<i>shouldClearInverseBidiMap()</i>	454-460	-	Branch
2.96	TreeBidiMap(hMap)	<i>shouldReturnKeyIsContainedFirstInverse()</i>	454-460	-	Branch
2.97	TreeBidiMap(hMap)	<i>shouldReturnKeyIsNotContainedInverse()</i>	454-460	-	Branch
2.98	TreeBidiMap(hMap)	<i>shouldReturnValueIsContainedFirstInverse()</i>	454-460	-	Branch
2.99	TreeBidiMap(hMap)	<i>shouldReturnValueIsContainedLastInverse()</i>	454-460	-	Branch

2.100	TreeBidiMap(hMap)	<i>shouldReturnValuesNotContainedInverse()</i>	454-460	-	Branch
2.101	TreeBidiMap(hMap)	<i>shouldFailContainsValueWithNullPointerExceptionInverse()</i>	454-460	-	Branch
2.102	TreeBidiMap()	<i>shouldReturnMapIsEmptyInverse()</i>	454-460	-	Branch
2.103	TreeBidiMap(hMap)	<i>shouldReturnMapIsNotEmptyInverse()</i>	454-460	-	Branch
2.104	TreeBidiMap()	<i>shouldReturnNewEntrySetInverse()</i>	454-460	-	Branch
2.105	TreeBidiMap(hMap)	<i>shouldReturnEntryViewInverse()</i>	454-460	-	Branch
2.106	TreeBidiMap()	<i>shouldFailEmptyFirstKeyInverse()</i>	454-460	-	Branch
2.107	TreeBidiMap(), “{1=”aaa”}”	<i>shouldReturnOneElementFirstKeyInverse()</i>	454-460	-	Branch
2.108	TreeBidiMap(hMap)	<i>shouldReturnFirstKeyInverse()</i>	454-460	-	Branch

2.109	TreeBidiMap()	<i>shouldReturnNullKey NotPresentTreeEmptyI nverse()</i>	454-460	-	Branch
2.110	TreeBidiMap(hMa p)	<i>shouldReturnValueFro mKeyInverse()</i>	454-460	node.getValue() != null	Branch
2.111	TreeBidiMap(hMa p)	<i>shouldReturnNullKey NotPresentInverse()</i>	454-460	node.getValue == null	Branch
2.112	TreeBidiMap(hMa p), Int[] array	<i>shouldThrowClassCas tExceptionKeyWrongT ypeInverse_get()</i>	454-460	o != null, !(o instanceof Comparable)	Branch
2.113	TreeBidiMap(hMa p)	<i>shouldThrowClassCas tExceptionKeyNullInv erse_get()</i>	454-460	o == null	Branch
2.114	TreeBidiMap()	<i>shouldReturnEmptyHa shCodeInverse()</i>	454-460	-	Branch
2.115	TreeBidiMap(), “{1=”aaa”}”	<i>shouldReturnOneElem entHashCodeInverse()</i>	454-460	-	Branch
2.116	TreeBidiMap(hMa p)	<i>shouldReturnHashCod eInverse()</i>	454-460	-	Branch
2.117	TreeBidiMap()	<i>shouldFailEmptyLast KeyInverse()</i>	454-460	nodeCount == 0	Branch

2.118	TreeBidiMap(), “{1=”aaa”}”	<i>shouldReturnOneElementLastKeyInverse()</i>	454-460	nodeCount != 0	Branch
2.119	TreeBidiMap(hMap)	<i>shouldReturnLastKeyInverse()</i>	454-460	nodeCount != 0	Branch
2.120	TreeBidiMap()	<i>shouldReturnLastKeyInverse()</i>	454-460	node == null	Branch
2.121	TreeBidiMap()	<i>shouldFailReturnEmptyNextKeyInverse()</i>	454-460	node == null	Branch
2.122	TreeBidiMap(), “{1=”aaa”}”	<i>shouldReturnOneElementNextKeyInverse()</i>	454-460	node == null	Branch
2.123	TreeBidiMap(hMap)	<i>shouldReturnNextKeyInverse()</i>	454-460	node != null	Branch
2.124	TreeBidiMap()	<i>shouldReturnEmptyPreviousKeyInverse()</i>	454-460	node == null	Branch
2.125	TreeBidiMap()	<i>shouldFailEmptyPreviousKeyInverse()</i>	454-460	node == null	Branch
2.126	TreeBidiMap(), “{1=”aaa”}”	<i>shouldReturnOneElementPreviousKeyInverse()</i>	454-460	node == null	Branch

2.127	TreeBidiMap(hMap)	<i>shouldReturnPreviousKeyInverse()</i>	454-460	node != null	Branch
2.128	TreeBidiMap(), “{1=’new’}”	<i>shouldIncreaseSizeToOneInverse()</i>	454-460	Key != null Value != null Both instanceof comparable)	Branch
2.129	TreeBidiMap(), “{9=’new’}”	<i>shouldIncreaseSizeByOneInverse()</i>	454-460	Key != null Value != null Both instanceof comparable	Branch
2.130	TreeBidiMap(hMap), “{1=’new’}”	<i>shouldReplaceExistingValueInverse()</i>	454-460	Key == existing key Value != null Both instanceof comparable	Branch
2.131	TreeBidiMap(hMap), “{1=’new’}”	<i>shouldFailThrowClassCastExceptionInversePut()</i>	454-460	value != null, !(o instanceof Comparable)	Branch
2.132	TreeBidiMap(), “{null=’new’}”	<i>shouldFailThrowNullPointerExceptionInversePut()</i>	454-460	Value == null	Branch
2.133	TreeBidiMap()	<i>shouldReturnEmptyBracesAsStringInverse()</i>	454-460	nodeCount == 0	Branch
2.134	TreeBidiMap(hMap)	<i>shouldReturnMapInBracesAsStringInverse()</i>	454-460	nodeCount != 0, hasNext, !hasNext	Branch
2.135	TreeBidiMap()	<i>shouldReturnEmptyValuesInverse()</i>	454-460	inverse != null	Branch

2.136	TreeBidiMap(), “{“bbb”=“aaa”}”	<i>shouldReturnOneElementValuesInverse()</i>	454-460	-	Branch
2.137	TreeBidiMap(), {5}	<i>shouldReturnElementWhoseRightChildIsAncestorOfNode()</i>	367-369-nextSmaller()-668-669-670-370-371	parent != null && child == parent.getLeft(dataElement)	Branch
2.138	TreeBidiMap(), HashMap({1,1}))	<i>shouldRemoveKeyWithNullReplacement()</i>	262-doRemoveKey -doRedBlackDelete	deletedNode.ge tParent(dataEle ment) == null	Branch
2.139	TreeBidiMap(), HashMap({1,3}, {2,3})) 2	<i>shouldRemoveKeyWithoutParentButWithOtherChild()</i>	262-doRemoveKey -doRedBlackDelete	981: deletedNode.ge tParent(dataEle ment) == null)	Branch
2.143	TreeBidiMap()	<i>shouldReturnEmptyInverseMapIterator()</i>	2206-2207-2208-2209	2207: isEmpty()	Condition
2.144	TreeBidiMap()	<i>shouldReturnInverseMapIterator()</i>	2206-2207-2210	2207: isEmpty()	Condition
2.145	TreeBidiMap()	<i>shouldThrowExceptionWithLastReturnGetKeyNodeInverseMapIterator()</i>	1777-1781	lastReturnedNo de == null	Condition
2.146	TreeBidiMap()	<i>shouldThrowExceptionWithLastReturnGetValueNodeInverseMapIterator()</i>	1786-1791	lastReturnedNo de == null	Condition
2.147	TreeBidiMap(hMap)	<i>shouldPutAllInsideOfInverseTreeBidiMap()</i>	2160-2161-put()-2163	-	Statement

2.148	TreeBidiMap(hMap)	<i>shouldReturnValueViewWithKeysetInverseTreeBidiMap()</i>	2198-2202	inverseEntrySet == null	Condition
-------	-------------------	--	-----------	-------------------------	-----------

BagUtils Tests

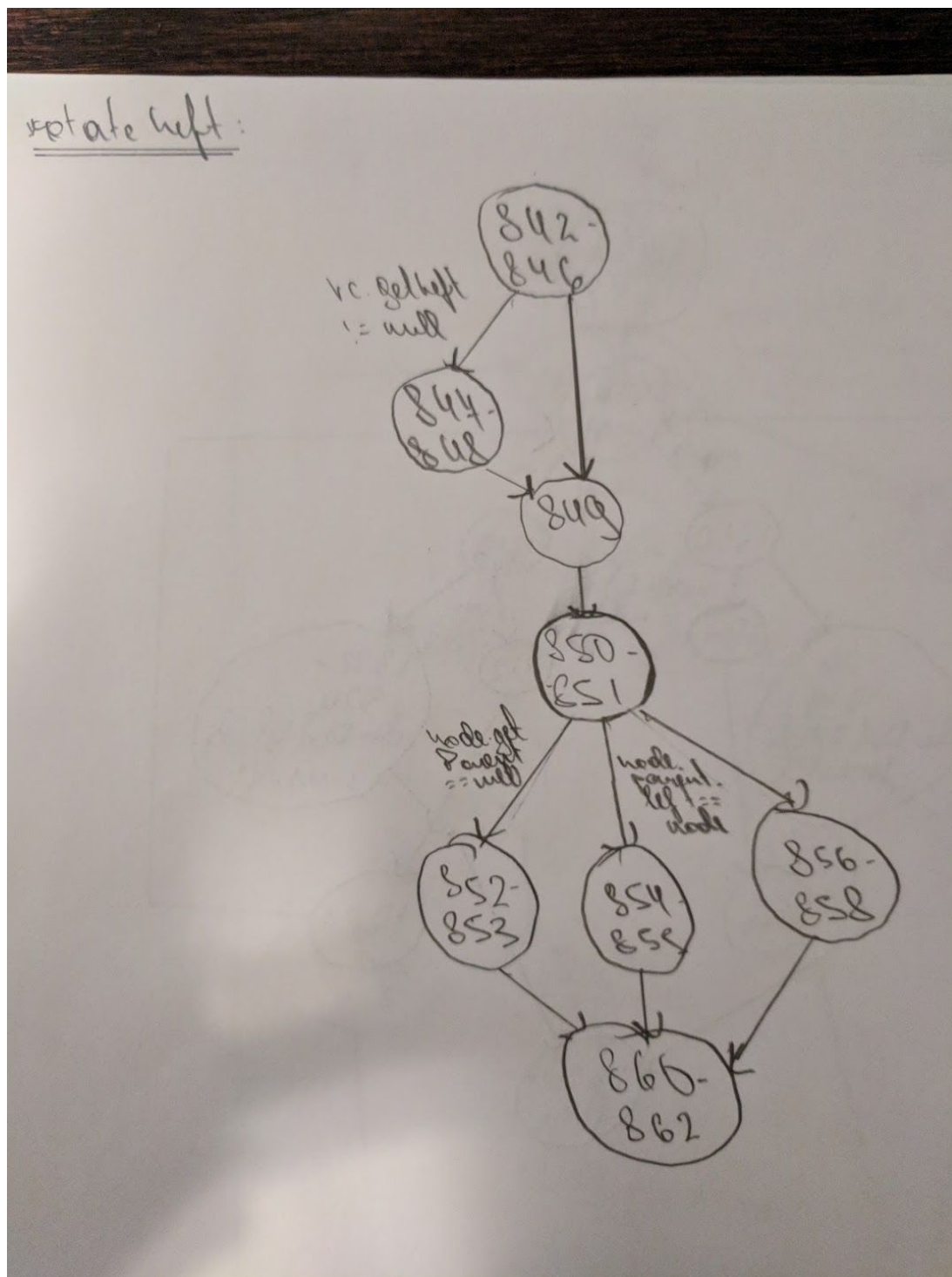
bagInstance is a TreeBag<Integer>, filled with {5, 3, 16, 1}

ID	Input	Test name	Lines covered (Path)	Conditions	Category
3.01	{bagInstance}	shouldReturnSynchronizedBag()	83-84-85	-	Statement coverage
3.02	{null}	<i>shouldFailSynchronizedBagWithNullPointerException()</i>	83-84-85	-	Statement coverage
3.03	{bagInstance}	<i>shouldReturnUnmodifiableBag()</i>	96-97-98	-	Statement coverage
3.04	{null}	<i>shouldFailUnmodifiableBagWithNullPointerException()</i>	96-97-98	-	Statement coverage
3.05	{bagInstance, NotNullPredicate.INSTANCE}	<i>shouldReturnPredicatedBag()</i>	115-116-117	-	Statement coverage
3.06	{null, FalsePredicate.INSTANCE} and {bagInstance, null}	<i>shouldFailPredicatedBagWithNullPointerException()</i>	115-116-117	-	Statement coverage
3.07	{bagInstance, ConstantTransformer(77)}	<i>shouldReturnTransformedBag()</i>	135-136-137	-	Statement coverage
3.08	{null, PredicateTransformer(NotNullPredicate.INSTANCE)} and {bagInstance, null}	<i>shouldFailTransformingBagWithNullPointerException()</i>	135-136-137	-	Statement coverage
3.09	{bagInstance}	<i>shouldReturnCollection</i>	148-149-150	-	Statement

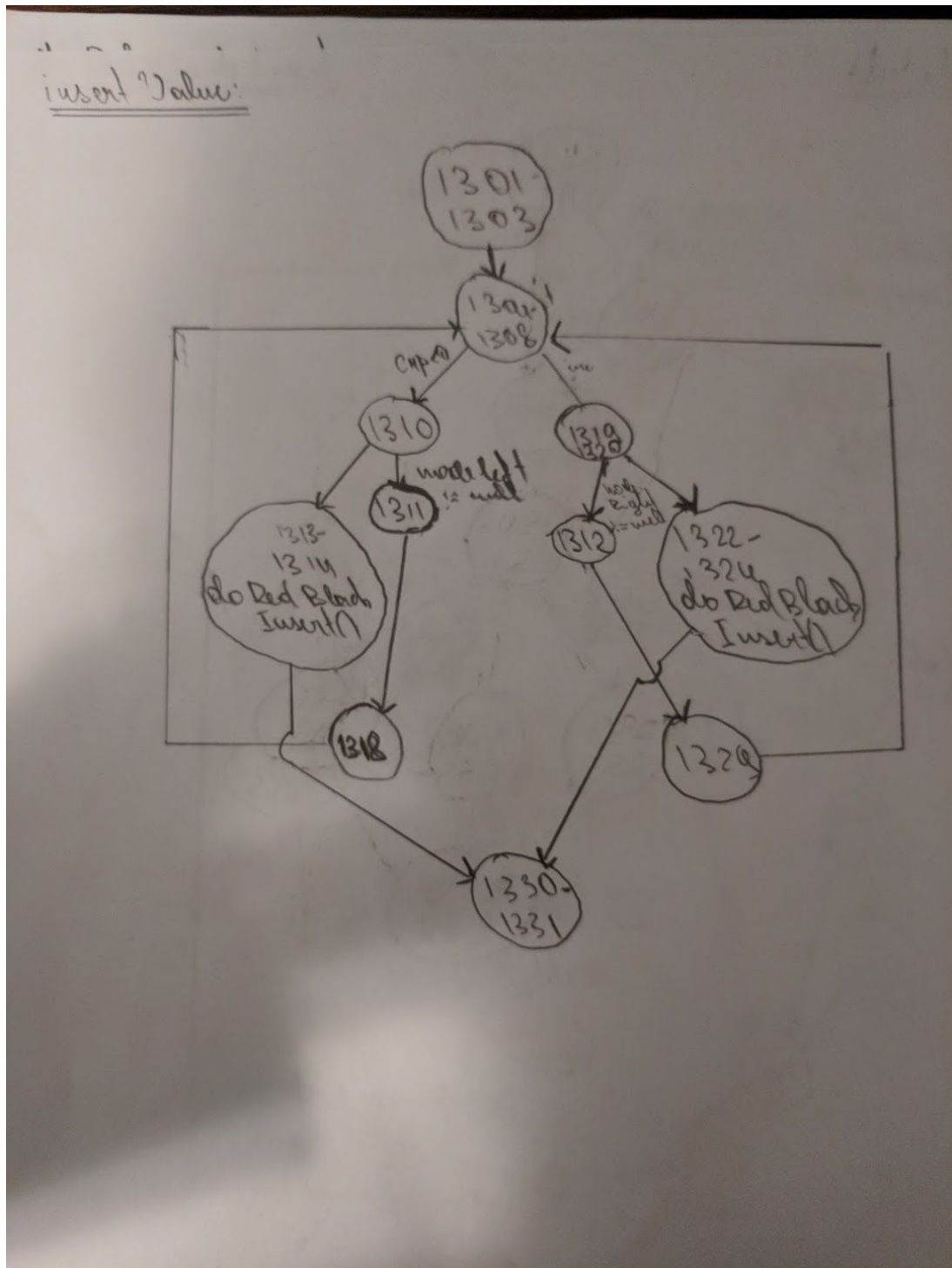
		<i>nBag()</i>			coverage
3.10	{null}	<i>shouldFailCollectionBagWithNullPointerException()</i>	148-149-150	-	Statement coverage
3.11	{bagInstance}	<i>shouldReturnSynchronizedSortedBag()</i>	179-180-181	-	Statement coverage
3.12	{null}	<i>shouldFailSynchronizeSortedBagWithNullPointerException()</i>	179-180-181	-	Statement coverage
3.13	{bagInstance}	<i>shouldReturnUnmodifiableSortedBag()</i>	193-194-195	-	Statement coverage
3.14	{null}	<i>shouldFailUnmodifiableSortedBagWithNullPointerException()</i>	193-194-195	-	Statement coverage
3.15	{bagInstance, NotNullPredicate. <i>INSTANCE</i> }	<i>shouldReturnPredicatedSortedBag()</i>	213-214-215-216	-	Statement coverage
3.16	{null, FalsePredicate. <i>INSTANCE</i> } and {bagInstance, null}	<i>shouldFailPredicatedSortedBagWithNullPointerException()</i>	213-214-215-216	-	Statement coverage
3.17	{bagInstance, ConstantTransformer(77)}	<i>shouldReturnTransformedSortedBag()</i>	235-236-237-238	-	Statement coverage
3.18	{null, PredicateTransformer(NotNullPredicate. <i>INSTANCE</i>)} and {bagInstance, null}	<i>shouldFailTransformingSortedBagWithNullPointerException()</i>	235-236-237-238	-	Statement coverage
3.19	-	<i>shouldReturnEmptyBag()</i>	247-248-249	-	Statement coverage
3.20	-	<i>shouldReturnEmptySortedBag()</i>	258-259-260	-	Statement coverage

Control Flow Graphs

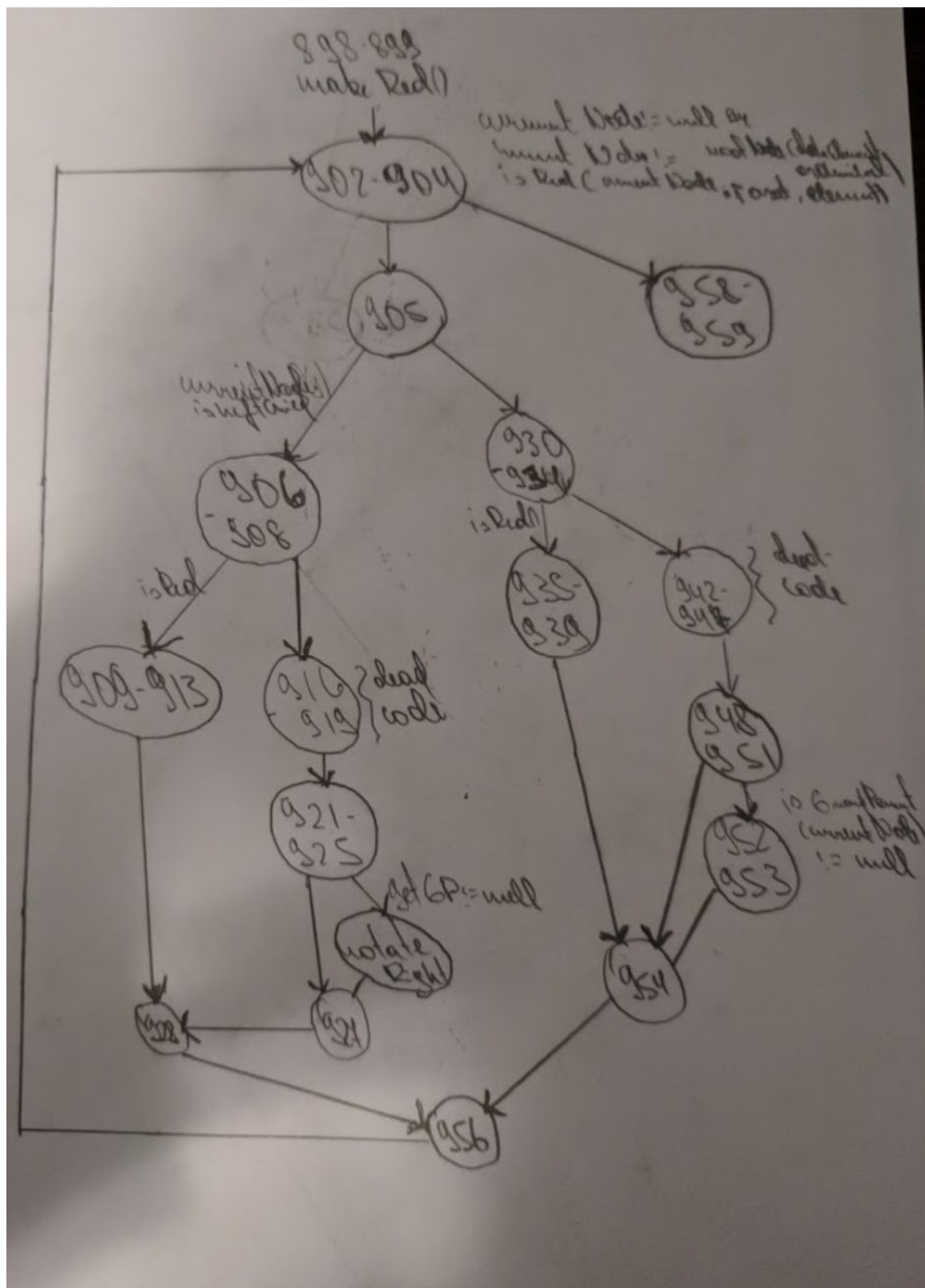
Graph 1 - rotateLeft



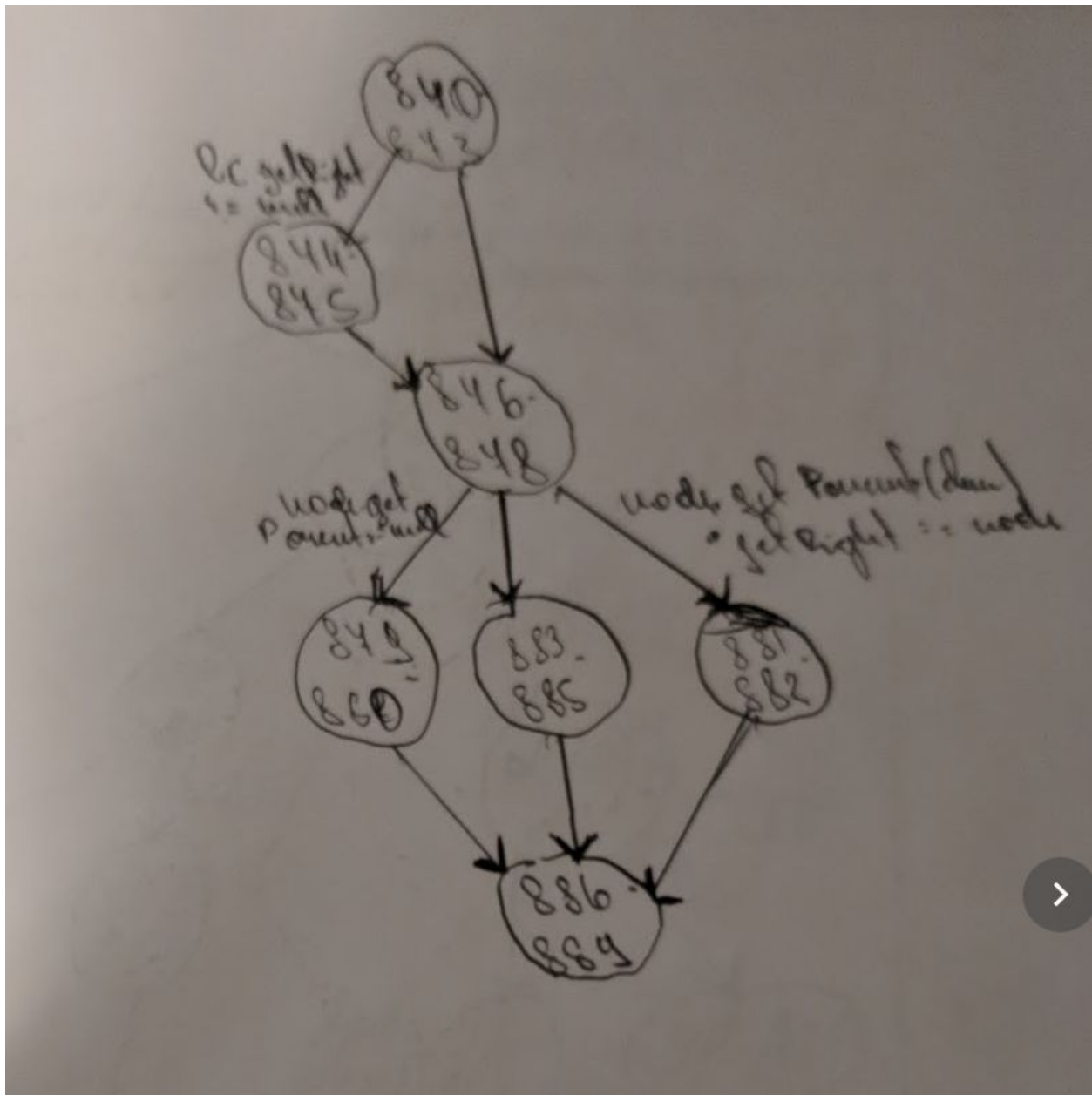
Graph 2 - insertValue



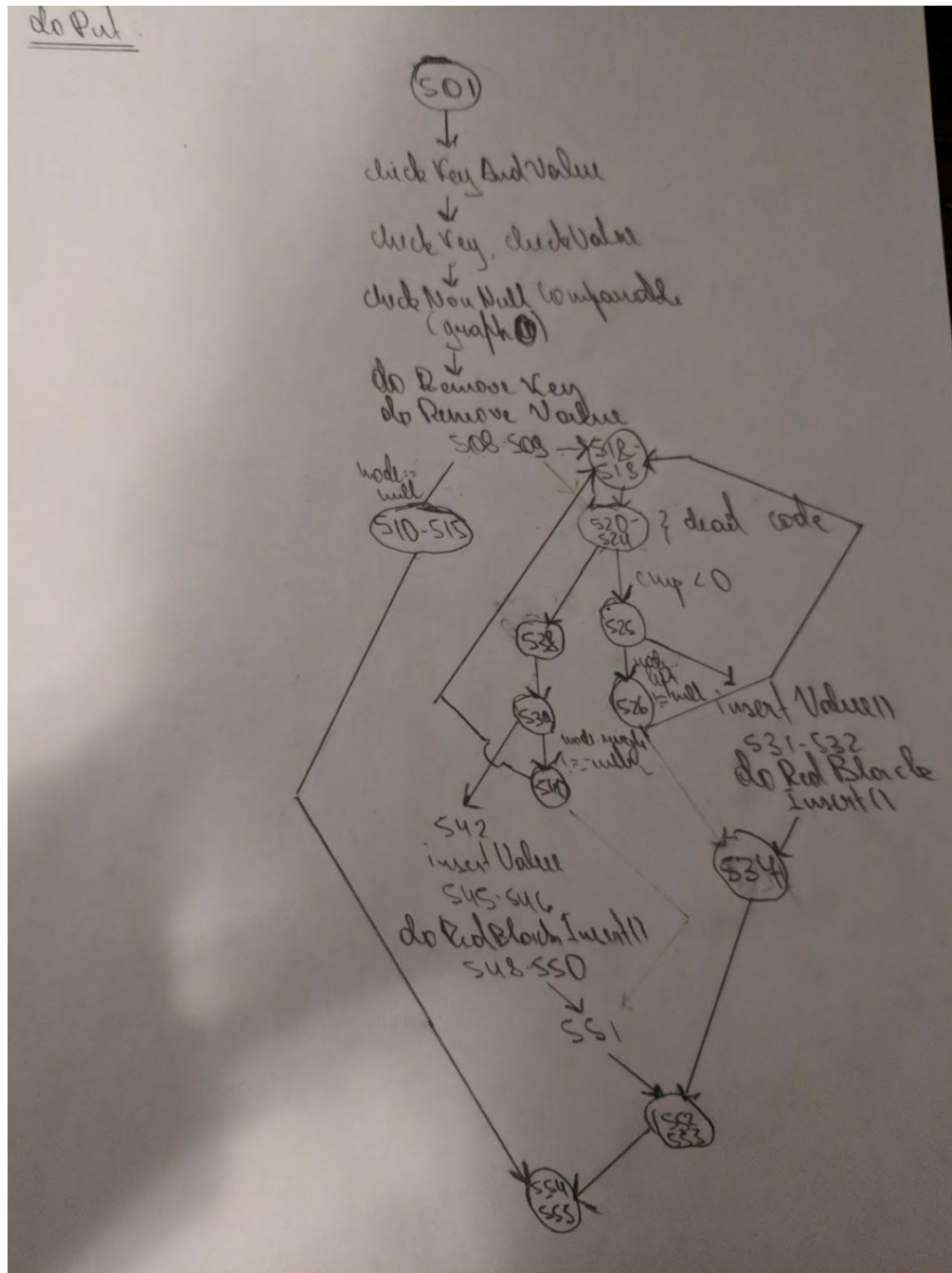
Graph 3 - doRedBlackInsert



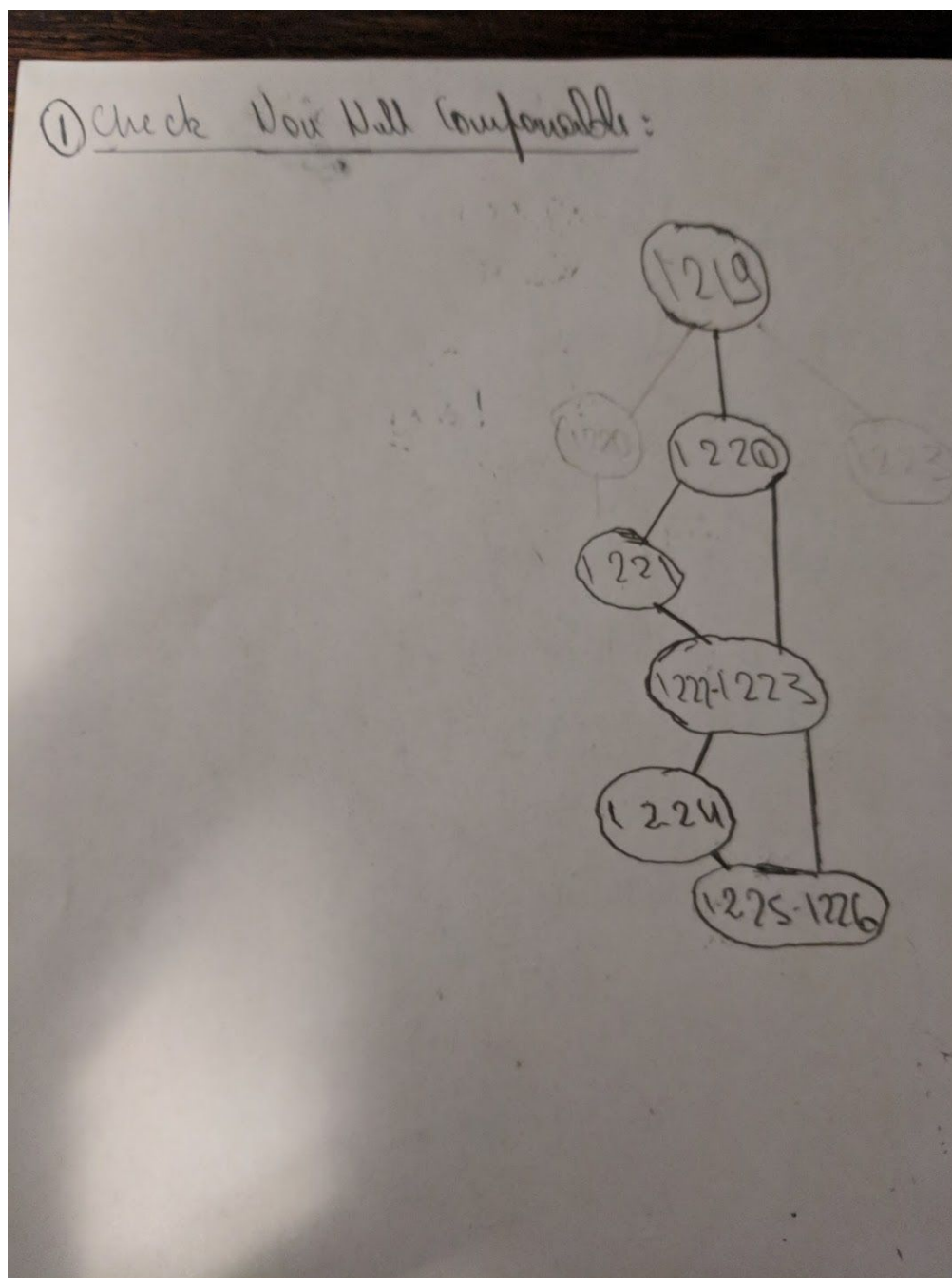
Graph 4 - rotateRight



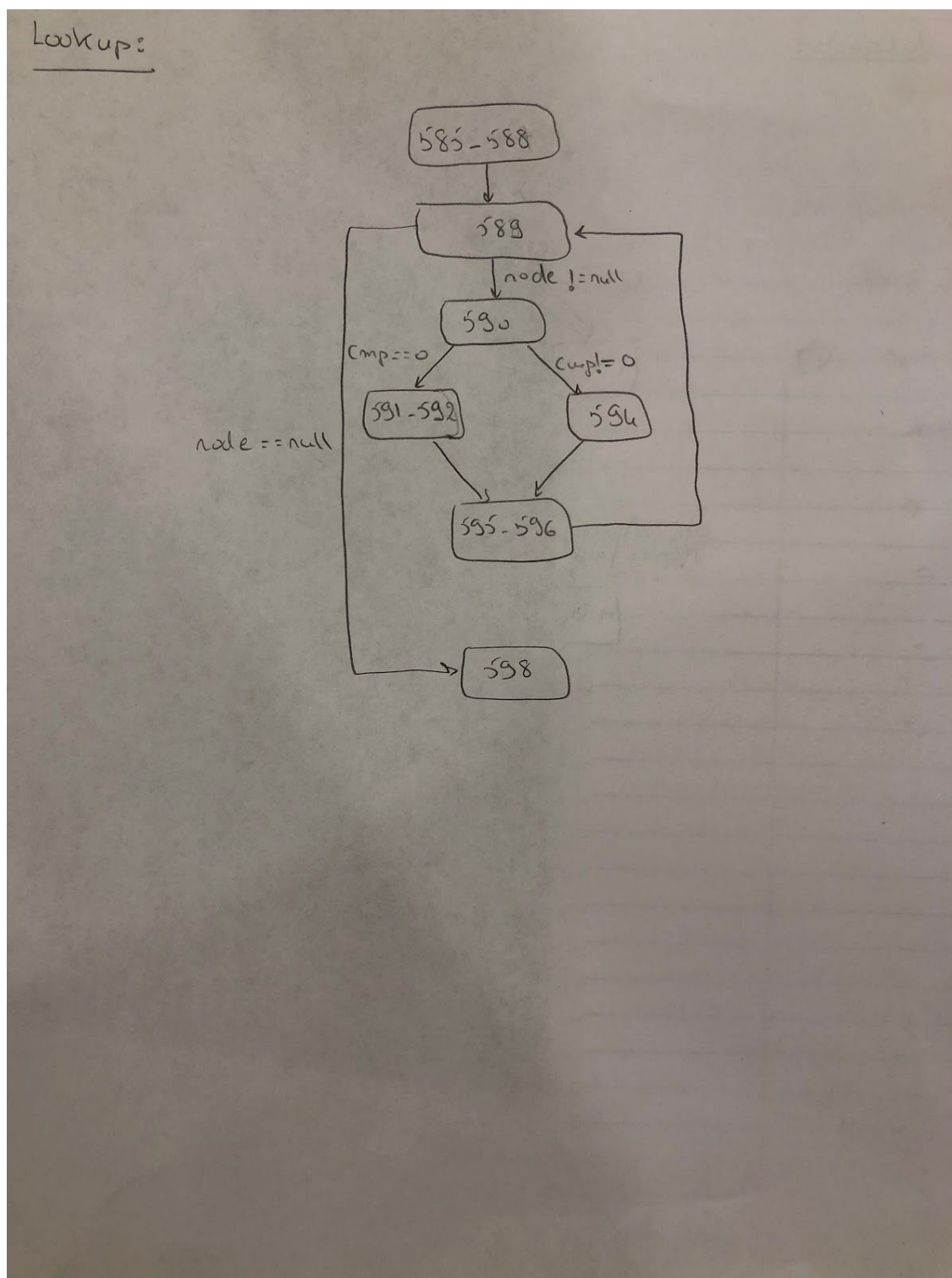
Graph 5 - doPut



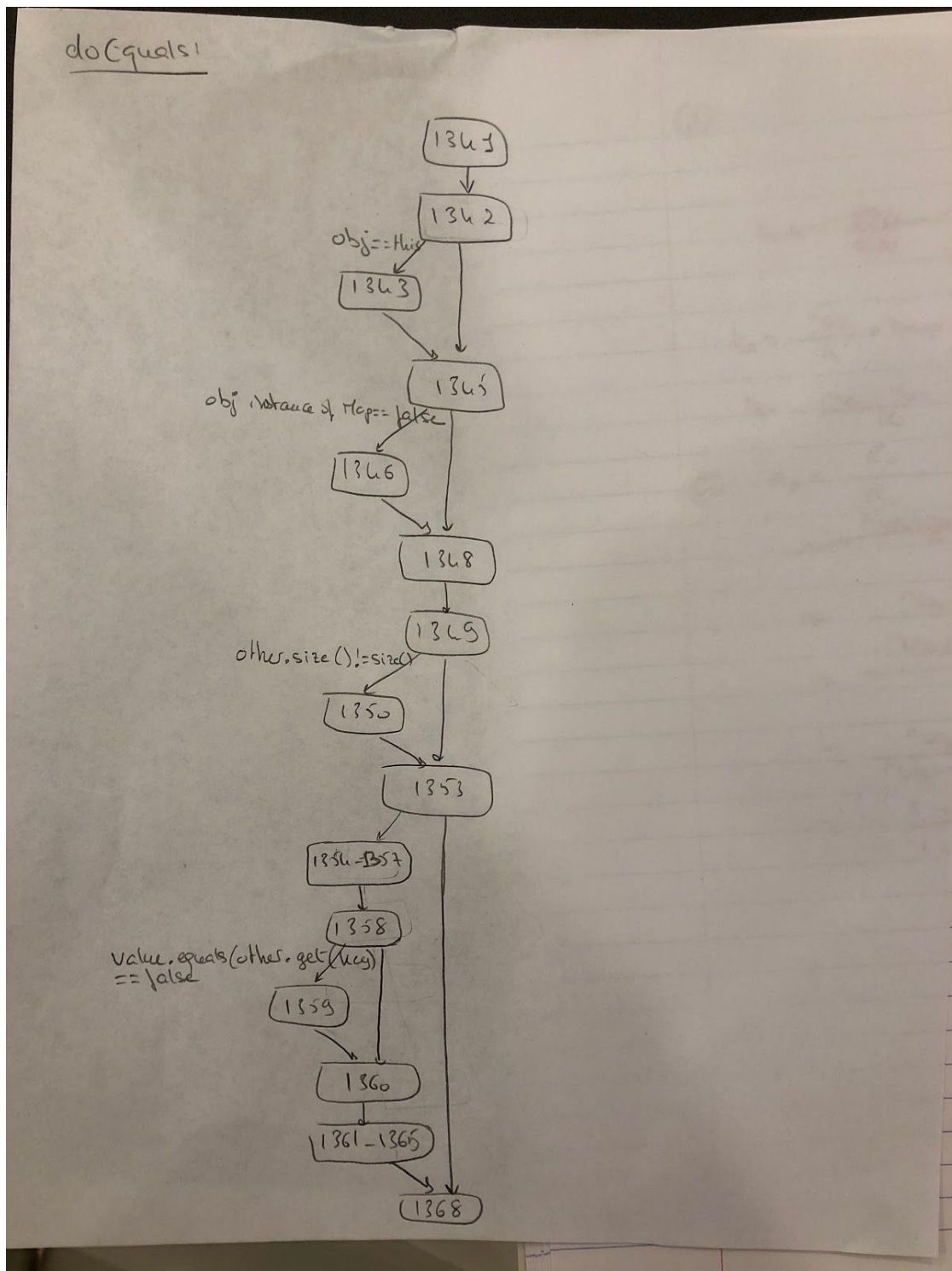
Graph 6 - checkNonNullComparable



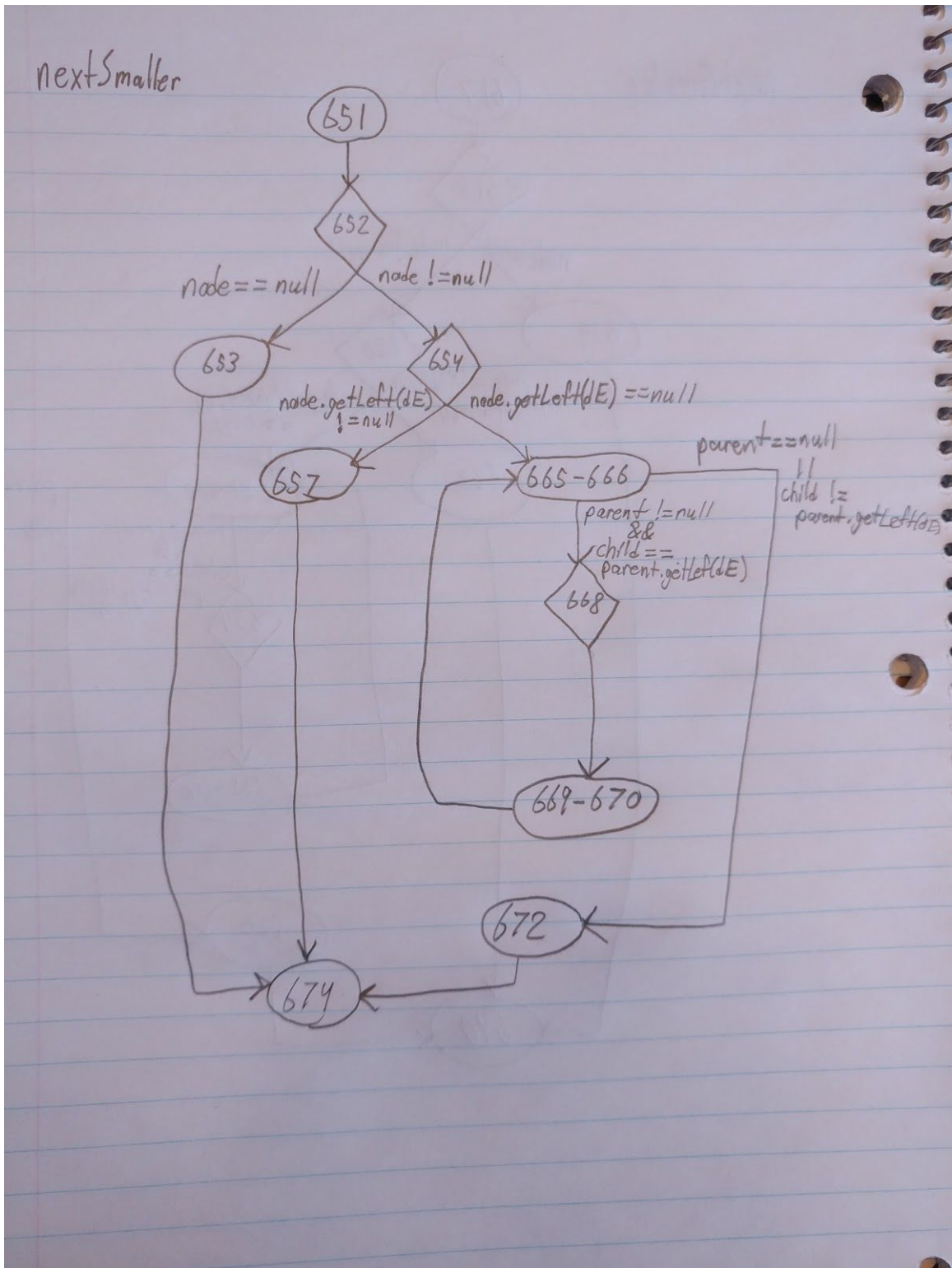
Graph 7 - lookup



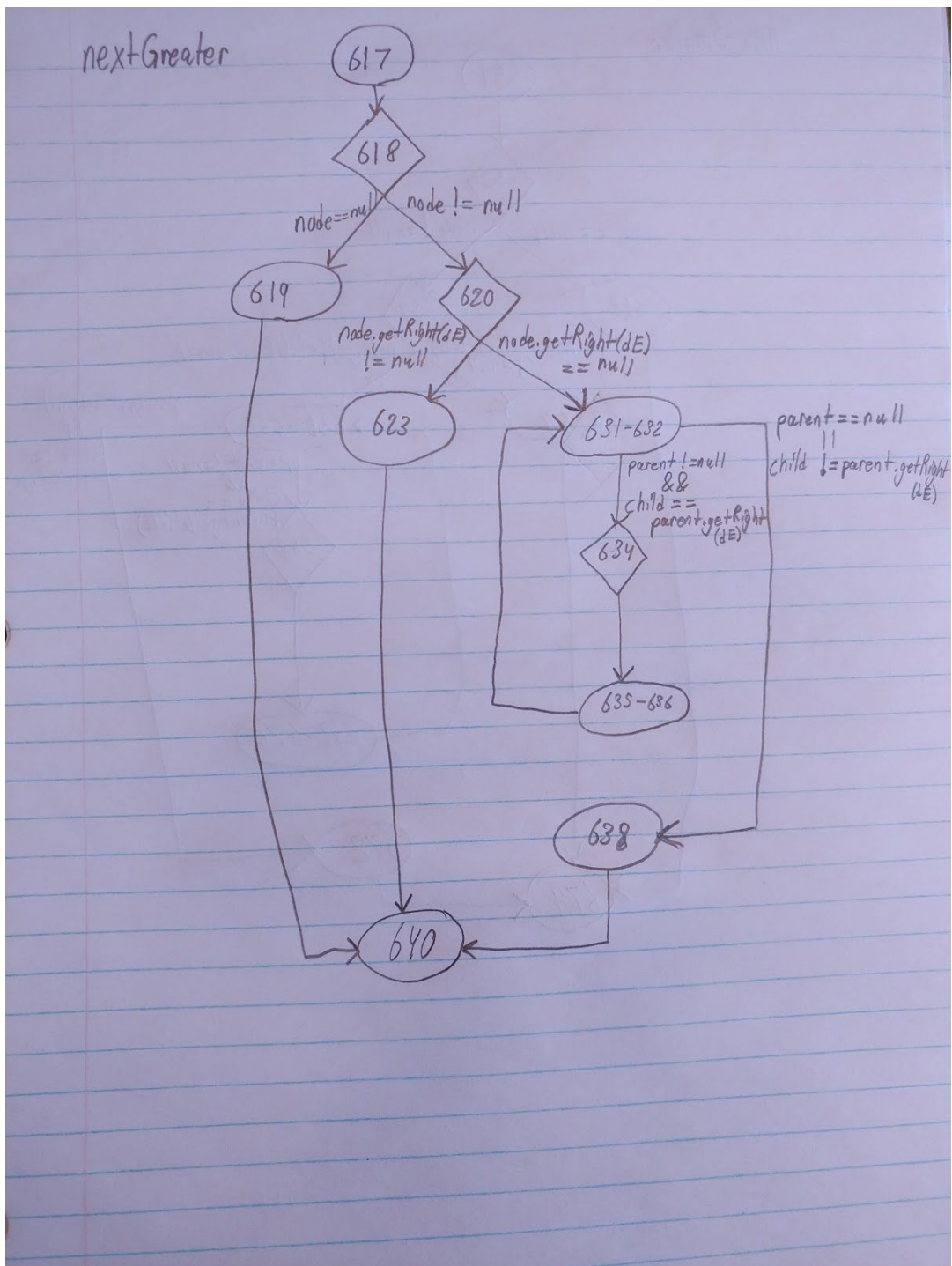
Graph 8 - doEquals



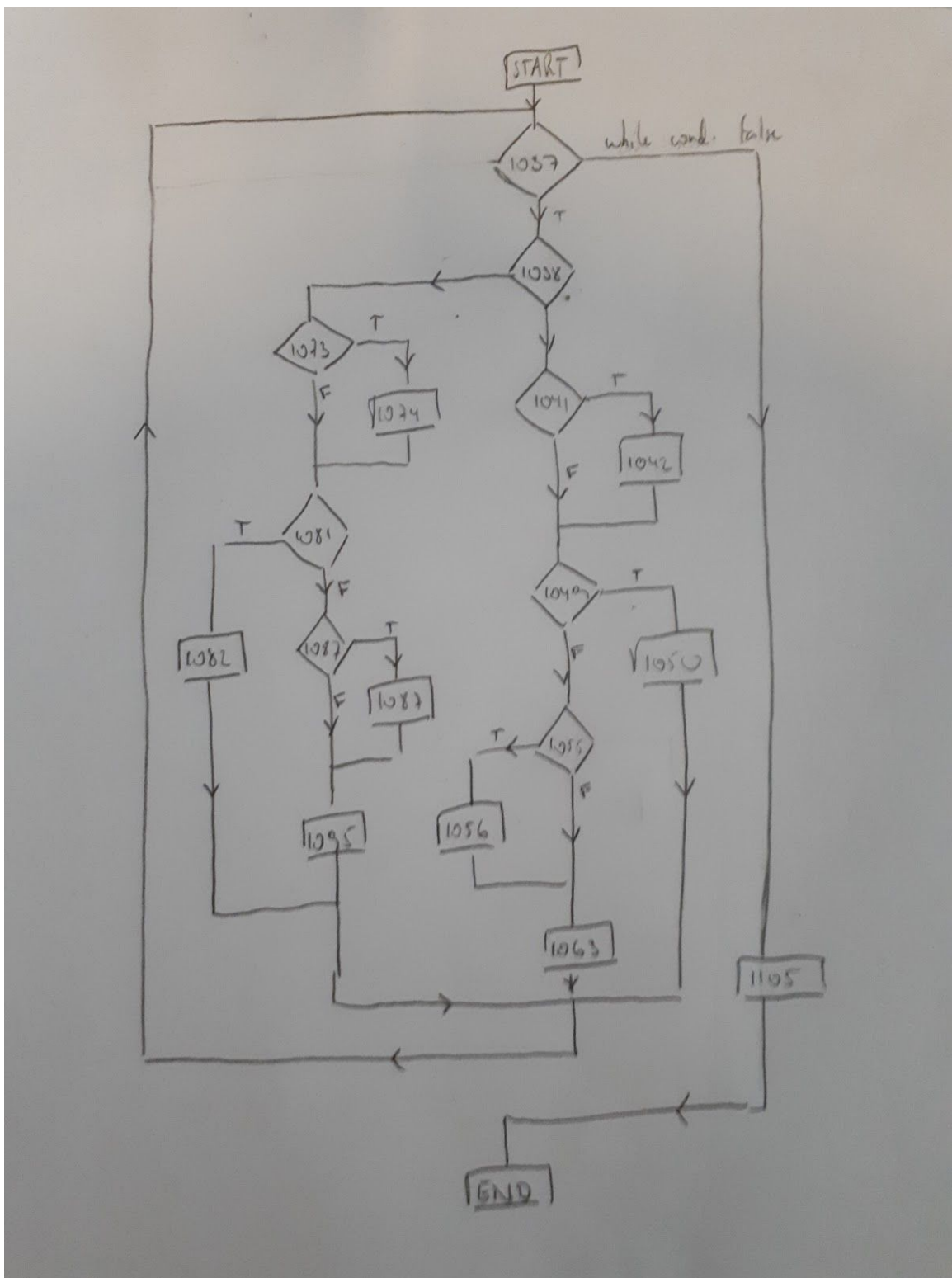
Graph 9 - nextSmaller



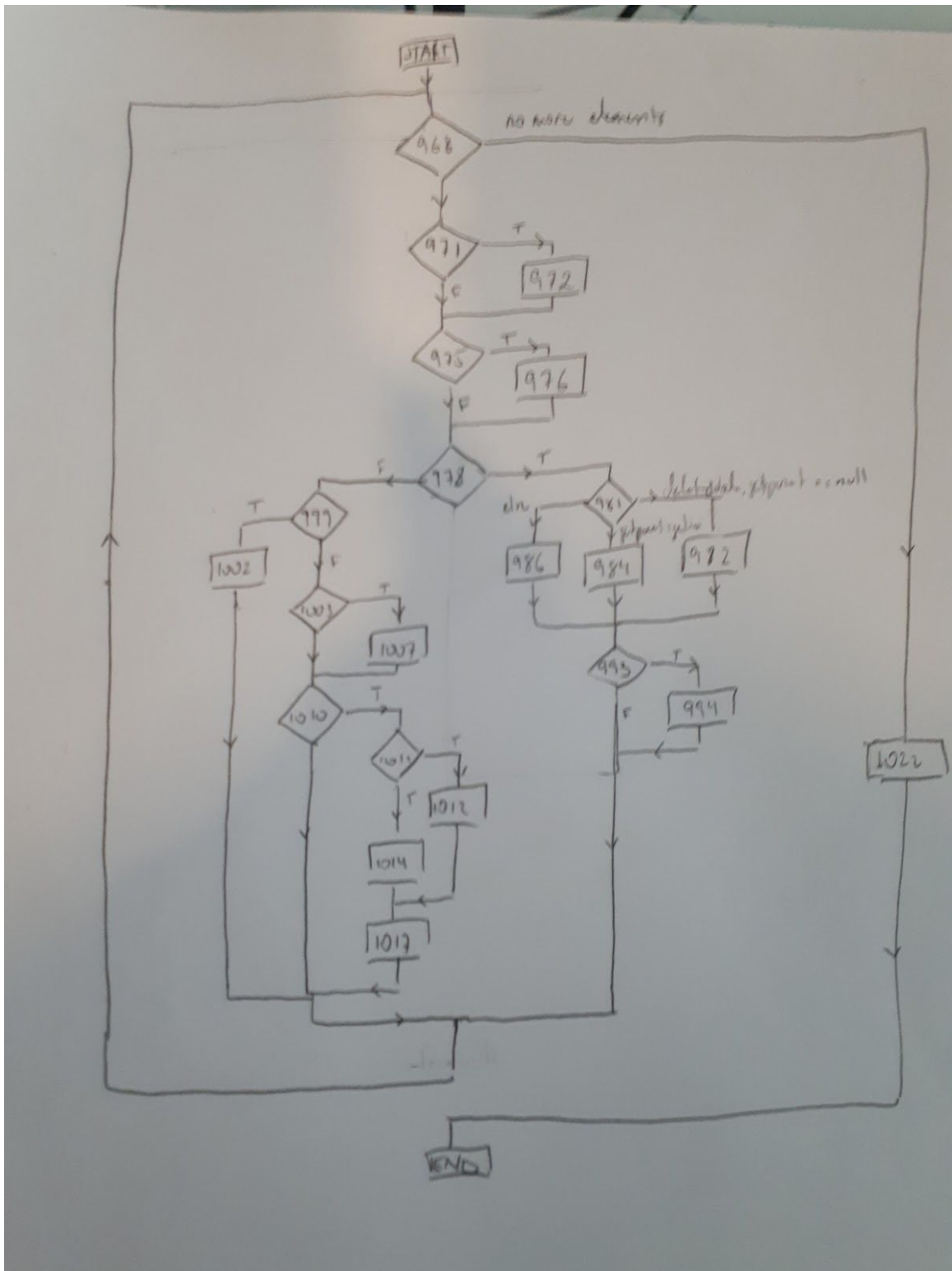
Graph 10 - nextGreater



Graph 11 - doRedBlackDeleteFixup



Graph 12 - doRedBlackDelete



Graph 13 - swapPosition

