**Report Software Testing**

**Progetto “1+”: FastJSON**

L’obiettivo del progetto *1+* è quello di sperimentare le tecniche di Software Testing e gli strumenti a supporto di esse, rendendo parametrici dei test cases di due classi di test preesistenti nel progetto *FastJSON*. Inoltre, si vuole valutare ed analizzare l’andamento della copertura strutturale al variare dei casi di test eseguiti e dei parametri passati in input agli stessi.

Le due classi di test in esame sono *JSONPatchTest\_0.java* e *MapTest2.java*.

Per quanto riguarda la configurazione del progetto, è stato inserito in una pipeline usando il framework di CI delle GitHub Actions. Per stimare la coverage ottenuta con l’esecuzione dei test e generarne dei report, viene utilizzato JaCoCo, che è stato agganciato al processo di build della pipeline. Tuttavia, non essendo possibile leggerne i risultati su SonarCloud, in quanto i sorgenti testati sono instrumentati on-the-fly a partire da un jar, i report sulla coverage vengono analizzati esclusivamente in seguito ad una build in locale.

Come possibile dedurre dal Codice originale della classe JSONPatchTest\_0.java, lo scopo di questa classe è quello di testare il metodo statico *apply* della classe *JSONPatch*. Tale metodo prende in input due rappresentazioni di JSONObject in formato Stringa, di cui la seconda descrive un pool di operazioni da effettuare sul primo JSONObject. Ognuna di queste operazioni può essere una add, remove, replace, move, copy o test.

La classe di test originale prevedeva un metodo di test separato per ogni passaggio di parametri differente al metodo da testare; inoltre, il risultato atteso è passato in maniera hardcoded. Dunque, in sostanza il test case risulta essere soltanto uno, ma invocato con parametri differenti. Quindi, la parametrizzazione dei test cases è avvenuta separando la logica di configurazione del test dall’effettiva esecuzione, operando nel seguente modo: il costruttore della classe di test chiama internamente un metodo privato *configureTestClass*, passandogli in input i parametri da utilizzare nel metodo di test; inoltre, viene passato anche il risultato atteso come output dell’invocazione del metodo sotto test.

Tale scelta è stata fatta in quanto FastJSON è basato su un concetto di oggetto JSON differente dallo standard ed implementare un oracolo per il calcolo dell’output atteso avrebbe richiesto una completa re-implementazione del metodo under test e della logica sottostante a livello di dominio. Tutto ciò è stato ritenuto andare oltre gli obiettivi del progetto *1+*.

Il metodo *configureTestClass* non fa altro che configurare l’ambiente di test, inizializzando gli attributi privati della classe di test con i due parametri di input da usare nell’invocazione del metodo sotto test e con il valore dell’output atteso. Nel costruttore è lasciata de-commentata solo una invocazione per volta al metodo di configure; in questo modo, si seleziona ad ogni build una chiamata al configure con parametri diversi, così da valutare il cambiamento ottenuto in termini di copertura.

In Codice 2 è riportata una visione complessiva del codice della classe *JSONPatchTest\_0* re-implementata per avere dei test parametrici.

Come già anticipato, la coverage viene stimata utilizzando il framework JaCoCo, il quale produce principalmente due metriche di copertura: una statement coverage, basata sul numero di istruzioni in bytecode coperte dall’esecuzione dei casi di test, ed una branch coverage, la quale prende in considerazione i costrutti di switch e gli if-statement.

La prima analisi viene effettuata invocando il metodo under test con una patch comprendente un pool di operazioni; in particolare, si tratta di una operazione di *replace*, una di *add* ed una di *remove*.

Com’è possibile vedere in Figura 1, si ottiene una statement coverage complessiva dei sorgenti del 14% e, in particolare, del package *com.alibaba.fastjson* (in cui è presente la classe *JSONPatch*) del 7%. La singola classe *JSONPatch* risulta avere una copertura strutturale del 51% ed una branch coverage del 38 % (Figura 2). La copertura dei singoli metodi della classe in questione è visibile in Figura 3.

Una seconda analisi è effettuata lanciando il test con una patch composta da una singola operazione di *move*.

Per quanto riguarda la copertura strutturale del codice sorgente, la Figura 4 ci dimostra che si hanno davvero pochi cambiamenti rispetto al caso precedente; in particolare, la percentuale di statement coverage del package *com.alibaba.fastjson* è esattamente uguale. Invece, la copertura della classe *JSONPatch* risulta essere aumentata in statement coverage (dal 51% al 58%), ma non in branch coverage (Figura 5). Infatti, guardando la copertura strutturale dei metodi della classe (Figura 6), essa risulta essere aumentata sia per il metodo *apply(Object, String)* che per *isObject(String)*. Per comprendere tali risultati, è stato analizzato il decompilato ottenuto a partire dal file *JSONPatch.class*. Il blocco di codice che si occupa di gestire l’operazione di *move* richiede molteplici istruzioni, più di quante ne richiedano un’operazione di *add*, di *replace* e di *remove* insieme.

L’altra classe di test è *MapTest2.java*. Essa include originariamente un solo metodo di test in cui viene testato il metodo statico *parseObject(String text, TypeReference<T> type)* della classe *JSON*. Tale metodo ha il compito di fare il parsing della stringa in input (rappresentante un JSON Object) in un oggetto il cui tipo è indicato dalla classe *TypeReference*. In particolare, per un JSON Object ci si aspetta che venga trasformato in una *Map<Object, Object>*. Il test passa se e solo se ad ogni chiave è associato il valore atteso.

Il codice originale della classe è riportato in Codice 3. Anche in questo caso, per la parametrizzazione si è fatto uso di un metodo *configureTestClass* invocato dal costruttore della classe di test. L’unico parametro passato al metodo di configure è la stringa rappresentante il *JSONObject*. La configurazione dell’ambiente di test consiste nel settare un attributo privato della classe di test con la stringa di cui sopra e nell’allocare un’istanza di *TypeReference<Map<Object,Object>>*, associando anch’essa ad un attributo privato.

Questa volta, è stato possibile fare un semplice parsing della stringa rappresentante il *JSONObject* per ottenerne un array di chiavi di tipo *Object* da utilizzare nel metodo di test. Inoltre, è stata utilizzata la libreria **JSON-JAVA (org.json)** per ottenere un oggetto JSON da cui estrarre gli output attesi per il test. Tali responsabilità sono state attribuite ad una classe *Oracle* interna alla classe di test, la quale agisce appunto da oracolo di test.

La nuova implementazione parametrica di *MapTest2* è consultabile in Codice 4.

Essendo il test originario eseguito con una sola rappresentazione in stringa di un *JSONObject*, non è possibile valutare la variazione della copertura al cambio dei valori dei parametri di test (a meno di progettare ed inserire nuovi parametri appropriati). La coverage che si ottiene a seguito dell’esecuzione del solo test in esame è consultabile in Figura 7 e Figura 8: la statement coverage su tutti i package ammonta al 5%, quella del package *org.alibaba.fastjson* (contenente la classe *JSON*) appena all’1%, mentre la statement coverage della classe *JSON* è del 9%. Sono valori molto bassi, ma giustificati dall’esecuzione di un solo test case su un metodo di una singola classe, la quale ha una size molto elevata.

Sono state in seguito valutate le metriche di copertura dell’intera test-suite, composta dai due metodi di test delle classi parametrizzate in precedenza. Ci si aspetta naturalmente un incremento dei valori di coverage rispetto a quelli analizzati precedentemente.

Relativamente ai risultati ottenuti dal primo run del test di *JSONPatchTest\_0.java*, la branch coverage è aumentata dal 6% al 7% su tutti i package (Figura 9), mentre, dal punto di vista percentuale, la statement coverage è rimasta assestata sul 14%. Tuttavia, è possibile notare che il numero di missed instructions è leggermente diminuito. Quindi, come ci si aspettava, si ha un miglioramento generale della coverage.

In particolare, è molto interessante notare come la statement coverage della classe *JSON* sia passata dal 9%, con l’esecuzione del test di *MapTest2,* al 14%, con l’esecuzione dell’intera test-suite (Figura 10). Questo perché ovviamente la classe viene coinvolta anche nel flusso d’esecuzione del test di *JSONPatchTest\_0*, molto probabilmente per eseguire il parsing delle stringhe in oggetti JSON.

**Progetto 2: Apache BookKeeper**

TODO…

**Allegati**

|  |
| --- |
| public class JSONPatchTest\_0 **extends** TestCase **{**  public void test\_for\_multi\_0**()** **throws** Exception **{**  String original **=** "{\n" **+**  " \"baz\": \"qux\",\n" **+**  " \"foo\": \"bar\"\n" **+**  "}"**;**  String patch **=** "[\n" **+**  " { \"op\": \"replace\", \"path\": \"/baz\", \"value\": \"boo\" },\n" **+**  " { \"op\": \"add\", \"path\": \"/hello\", \"value\": [\"world\"] },\n" **+**  " { \"op\": \"remove\", \"path\": \"/foo\" }\n" **+**  "]"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"baz\":\"boo\",\"hello\":[\"world\"]}"**,** result**);**  **}**  public void test\_for\_add\_1**()** **throws** Exception **{**  String original **=** "{}"**;**  String patch **=** "{ \"op\": \"add\", \"path\": \"/a/b/c\", \"value\": [ \"foo\", \"bar\" ] }"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**,** result**);**  **}**  public void test\_for\_remove\_0**()** **throws** Exception **{**  String original **=** "{}"**;**  String patch **=** "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }\n"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{}"**,** result**);**  **}**  public void test\_for\_remove\_1**()** **throws** Exception **{**  String original **=** "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**;**  String patch **=** "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }\n"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"a\":{\"b\":{}}}"**,** result**);**  **}**  public void test\_for\_replace\_1**()** **throws** Exception **{**  String original **=** "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**;**  String patch **=** "{ \"op\": \"replace\", \"path\": \"/a/b/c\", \"value\": 42 }"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"a\":{\"b\":{\"c\":42}}}"**,** result**);**  **}**  public void test\_for\_move\_0**()** **throws** Exception **{**  String original **=** "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**;**  String patch **=** "{ \"op\": \"move\", \"from\": \"/a/b/c\", \"path\": \"/a/b/d\" }"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"a\":{\"b\":{\"d\":[\"foo\",\"bar\"]}}}"**,** result**);**  **}**  public void test\_for\_copy\_0**()** **throws** Exception **{**  String original **=** "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**;**  String patch **=** "{ \"op\": \"copy\", \"from\": \"/a/b/c\", \"path\": \"/a/b/e\" }"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"],\"e\":[\"foo\",\"bar\"]}}}"**,** result**);**  **}**  public void test\_for\_test\_0**()** **throws** Exception **{**  String original **=** "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}"**;**  String patch **=** "{ \"op\": \"test\", \"path\": \"/a/b/c\", \"value\": \"foo\" }"**;**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  assertEquals**(**"false"**,** result**);**  **}**  **}** |

Codice 1 Codice originale della classe JSONPatchTest\_0.java

|  |
| --- |
| public class JSONPatchTest\_0 **{**  /\* Instance under test is not needed, because the tested method is static\*/  private String original**;**  private String patch**;**  private String expected**;**  public JSONPatchTest\_0**()** **{**  configureTestClass**(**"{\n" **+** " \"baz\": \"qux\",\n" **+** " \"foo\": \"bar\"\n" **+** "}"**,**  "[\n" **+**  " { \"op\": \"replace\", \"path\": \"/baz\", \"value\": \"boo\" },\n" **+**  " { \"op\": \"add\", \"path\": \"/hello\", \"value\": [\"world\"] },\n" **+**  " { \"op\": \"remove\", \"path\": \"/foo\" }\n" **+** "]"**,**  "{\"baz\":\"boo\",\"hello\":[\"world\"]}"**);**  // configureTestClass("{}", "{ \"op\": \"add\", \"path\": \"/a/b/c\", \"value\": [ \"foo\", \"bar\" ] }",  // "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}");  //  // configureTestClass("{}", "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }\n",  // "{}");  // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }\n",  // "{\"a\":{\"b\":{}}}");  // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", "{ \"op\": \"replace\", \"path\": \"/a/b/c\", \"value\": 42 }",  // "{\"a\":{\"b\":{\"c\":42}}}");  // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", "{ \"op\": \"move\", \"from\": \"/a/b/c\", \"path\": \"/a/b/d\" }",  // "{\"a\":{\"b\":{\"d\":[\"foo\",\"bar\"]}}}");  // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", "{ \"op\": \"copy\", \"from\": \"/a/b/c\", \"path\": \"/a/b/e\" }",  // "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"],\"e\":[\"foo\",\"bar\"]}}}");  // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", "{ \"op\": \"test\", \"path\": \"/a/b/c\", \"value\": \"foo\" }",  // "false");  **}**  private void configureTestClass**(**String original**,** String patch**,** String expected**)** **{**  **this.**original **=** original**;**  **this.**patch **=** patch**;**  **this.**expected **=** expected**;**  **}**  @Test  public void testJSONPatch**()** **{**  String result **=** JSONPatch**.**apply**(**original**,** patch**);**  Assert**.**assertEquals**(**expected**,** result**);**  **}**  **}** |

Codice 2 Classe JSONPatchTest\_0.java con test parametrico

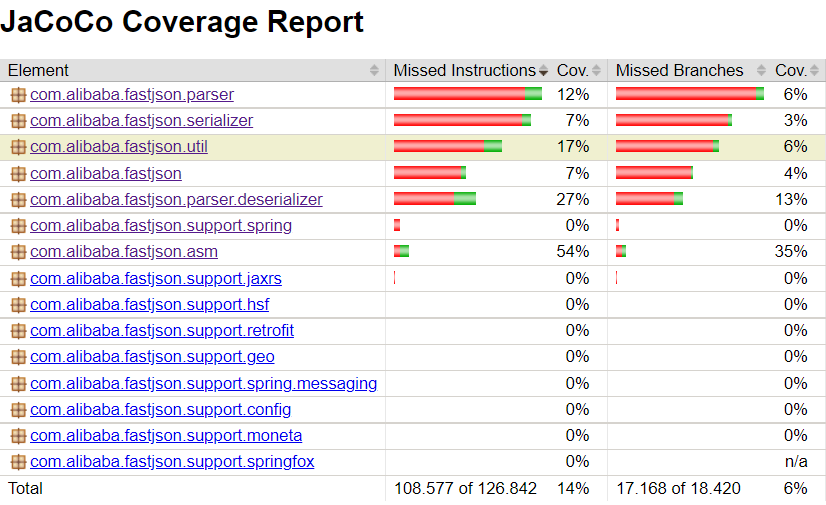


Figura 1 Coverage eseguendo solo JSONPatchTest\_0 con una patch da 3 operazioni



Figura 2 Coverage della classe JSONPatch con una patch da 3 operazioni

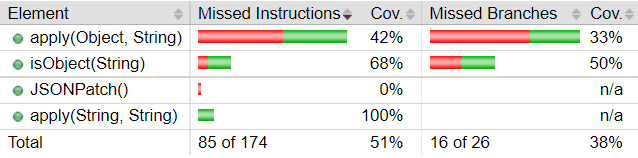


Figura 3 Coverage dei metodi della classe JSONPatch con una patch da 3 operazioni

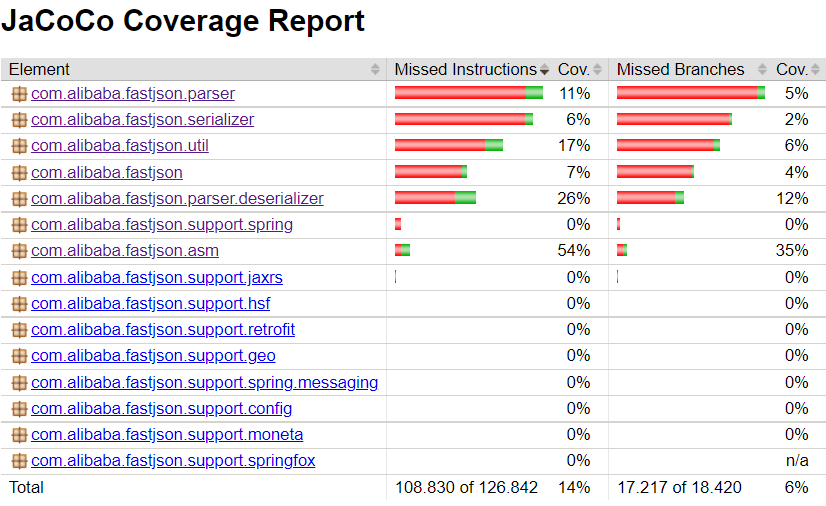


Figura 4 Coverage ottenuta eseguendo solo JSONPatchTest\_0 con una patch da una singola operazione



Figura 5 Copertura della classe JSONPatch a seguito dell'esecuzione di un test con una patch da una singola operazione

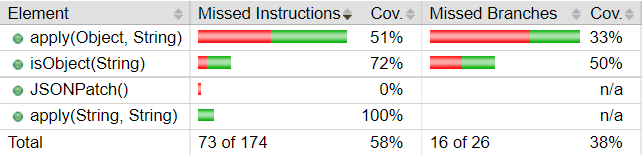


Figura 6 Coverage dei metodi della classe JSONPatch in seguito al test con patch con singola operazione

|  |
| --- |
| public class MapTest2 **extends** TestCase **{**  public void test\_map **()** **throws** Exception **{**  Map**<**Object**,** Object**>** map **=** JSON**.**parseObject**(**"{1:\"2\",\"3\":4,'5':6}"**,** **new** TypeReference**<**Map**<**Object**,** Object**>>()** **{});**  Assert**.**assertEquals**(**"2"**,** map**.**get**(**1**));**  Assert**.**assertEquals**(**4**,** map**.**get**(**"3"**));**  Assert**.**assertEquals**(**6**,** map**.**get**(**"5"**));**  **}**  **}** |

Codice 3 Codice originale della classe MapTest2

|  |
| --- |
| public class MapTest2 **{**  /\* The instance to be tested is not needed because the tested method is static\*/  private String jsonObject**;**  private TypeReference**<**Map**<**Object**,**Object**>>** typeReference**;**  public MapTest2 **(){**  configureTestClass**(**"{1:\"2\",\"3\":4,'5':6}"**);**  **}**  private void configureTestClass**(**String jsonObject**){**  **this.**jsonObject **=** jsonObject**;**  **this.**typeReference **=** **new** TypeReference**<>()** **{};**  **}**  /\*\*  \* This test method invokes the <i>parseObject</i> static method of the JSON class.  \* In particular, it checks that the conversion from the string representation  \* of the jsonObject to a Map between keys and values is well done.  \* The test is passed only if each value is associated to the correct key.  \*/  @Test  public void test\_map **()** **{**  Map**<**Object**,** Object**>** map **=** JSON**.**parseObject**(this.**jsonObject**,** **this.**typeReference**);**  // Use an oracle to compute the keys and the expected values bounded with those keys  Oracle oracle **=** **new** Oracle**();**  Object**[]** keys **=** oracle**.**getKeys**(**jsonObject**);**  Object**[]** expected **=** oracle**.**getExpectedResults**(**jsonObject**,** keys**);**  boolean passed **=** **true;**  **for** **(**int i **=** 0**;** i **<** keys**.**length**;** i**++){**  **if** **(**map**.**get**(**keys**[**i**])** **==** **null){**  **if** **(!** **(**map**.**get**(**keys**[**i**])** **==** expected**[**i**])){**  passed **=** **false;**  **break;**  **}**  **}**  **else** **if** **(!**map**.**get**(**keys**[**i**]).**equals**(**expected**[**i**])){**  passed **=** **false;**  **break;**  **}**  **}**  Assert**.**assertTrue**(**passed**);**  **}**  private static class Oracle**{**  /\*\*  \* This private method is used as a Software Testing oracle to know the  \* expected values of the JSON Object referred by the key in the input array of keys.  \* It's an oracle because it uses the library <b>org.json</b> as a trusted entity.  \*  \* **@param** jsonObject The string representing the JSONObject of the FastJSON project  \* **@param** keys The array of the objects representing the keys of the JSON Object  \* **@return** an array of Object, containing the values expected to be contained in the jsonObject,  \* in the same order of their keys  \*/  private Object**[]** getExpectedResults**(**String jsonObject**,** Object**[]** keys**){**  JSONObject obj **=** **new** JSONObject**(**jsonObject**);**  List**<**Object**>** expected **=** **new** LinkedList**<>();**  **for** **(**int i **=** 0**;** i **<** keys**.**length**;** i**++){**  expected**.**add**(**i**,** obj**.**get**(**String**.**valueOf**(**keys**[**i**])));**  **}**  **return** expected**.**toArray**();**  **}**  /\*\*  \* This private method is used to retrieve the array of keys  \* of the json object string representation in input.  \* **@param** jsonObject The string representing the JSONObject of the FastJSON project  \* **@return** an array of Objects, containing the keys of the JSONObject.  \*/  private Object**[]** getKeys**(**String jsonObject**){**  String noBrackets **=** jsonObject**.**replace**(**"{"**,** ""**);**  noBrackets **=** noBrackets**.**replace**(**"}"**,**""**);**  noBrackets **=** noBrackets**.**replace**(**" "**,** ""**);**  noBrackets **=** noBrackets**.**replace**(**"\n"**,** ""**);**  noBrackets **=** noBrackets**.**replace**(**"\r"**,** ""**);**  noBrackets **=** noBrackets**.**replace**(**"\t"**,** ""**);**  String**[]** pairs **=** noBrackets**.**split**(**","**);**  List**<**Object**>** ret **=** **new** LinkedList**<>();**  **for** **(**int i **=** 0**;** i **<** pairs**.**length**;** i**++){**  String key **=** pairs**[**i**].**split**(**":"**)[**0**];**  **if** **(**key**.**contains**(**"\""**)** **||** key**.**contains**(**"'"**)){**  ret**.**add**(**i**,** key**.**replace**(**"\""**,** ""**).**replace**(**"'"**,** ""**));**  **}**  **else** **if** **(**key**.**contains**(**"."**))**  ret**.**add**(**i**,** Float**.**valueOf**(**key**));**  **else**  ret**.**add**(**i**,** Integer**.**valueOf**(**key**));**  **}**  **return** ret**.**toArray**();**  **}**  **}**  **}** |

Codice 4 Classe MapTest2.java con test parametrico e inner class Oracle

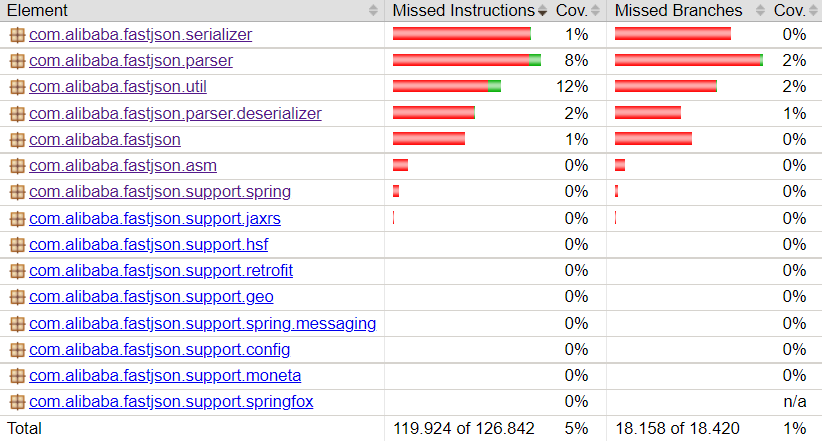


Figura 7 Coverage ottenuta eseguendo il solo test della classe MapTest2.java



Figura 8 Class coverage della classe JSON in seguito al test di MapTest2.java

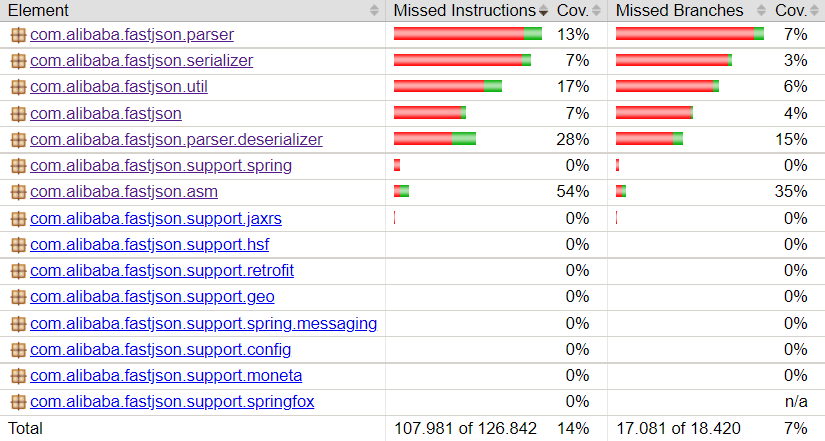


Figura 9 Coverage in seguito all'esecuzione dell'intera test-suite

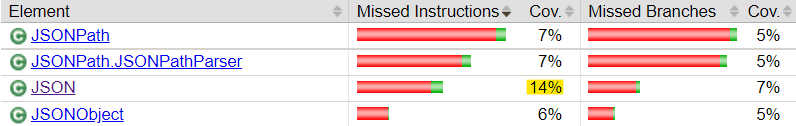


Figura 10 Coverage della classe JSON aumentata con l'esecuzione di tutti i test della test-suite