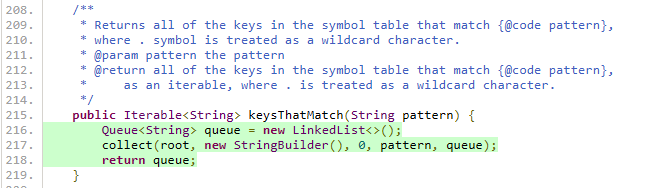
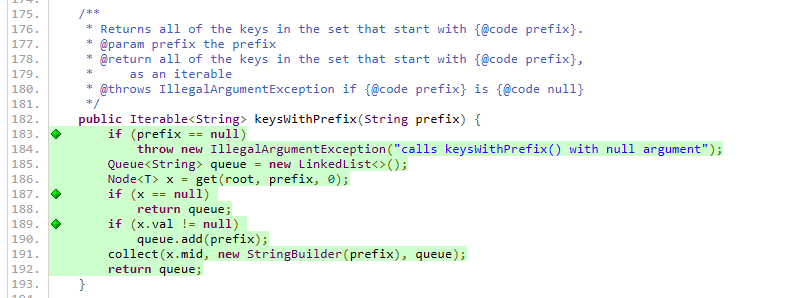
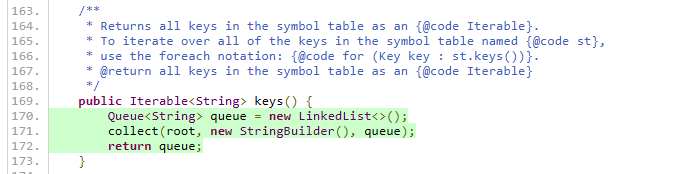
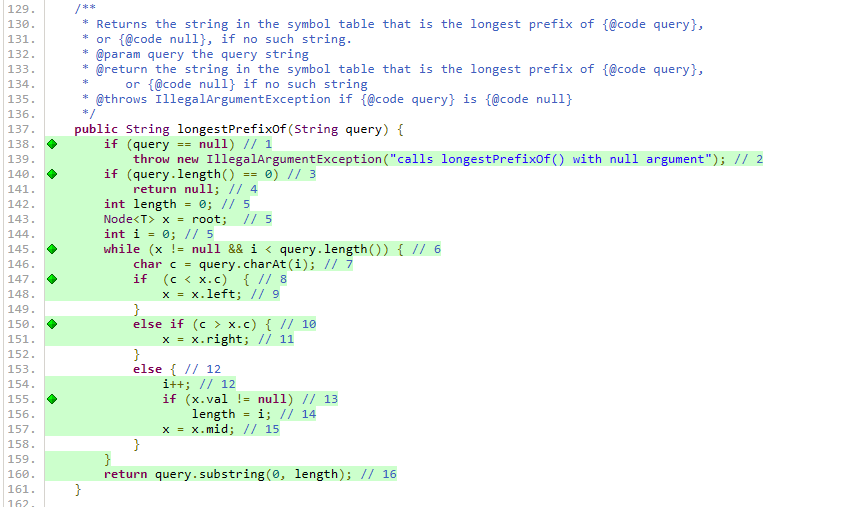
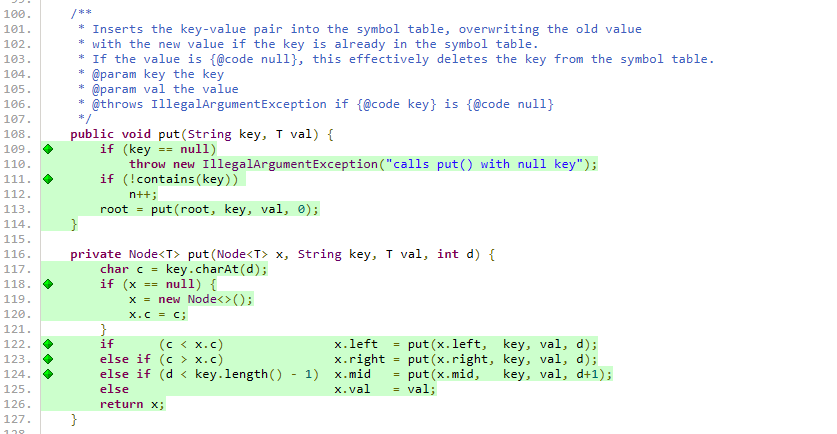
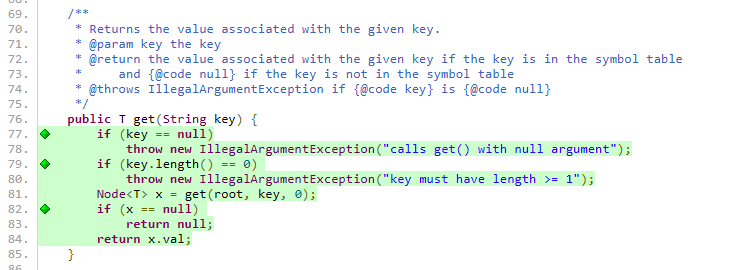
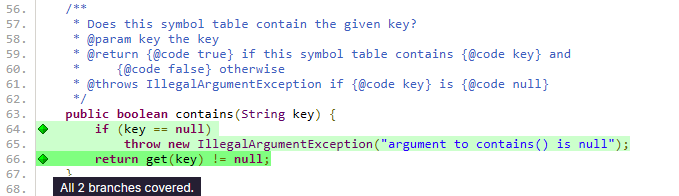
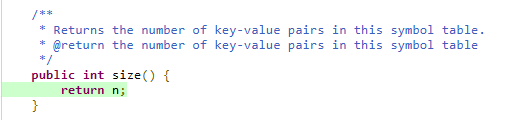
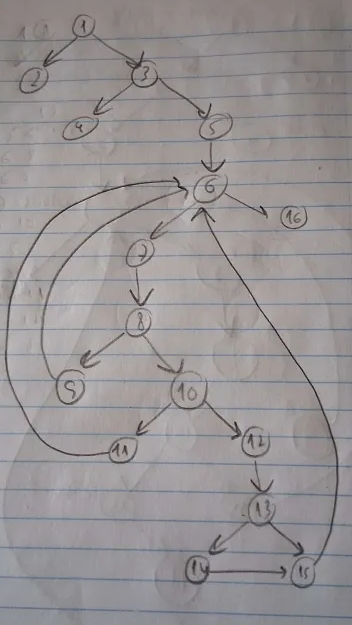
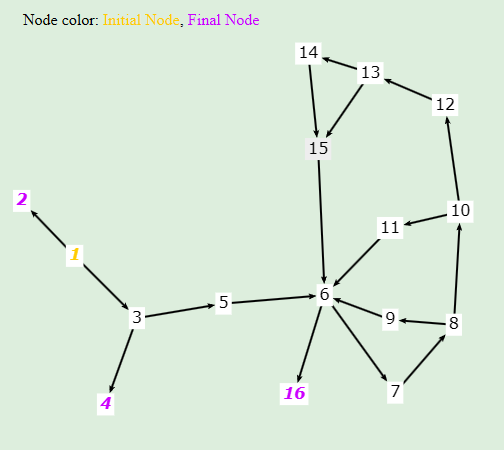
**Line and Branch Coverage**

**Coverage Outcomes:**



**Edge-Pair and Prime Path Coverage**

**Graph outcomes:**



**Graph Nodes:**

* **N1:** if (query == null)
* **N2:** throw new IllegalArgumentException("calls longestPrefixOf() with null argument");
* **N3:** if (query.length() == 0)
* **N4:** return null;
* **N5:** int length = 0; Node<T> x = root; int i = 0;
* **N6:** while (x != null && i < query.length()) {
* **N7:** char c = query.charAt(i);
* **N8:** if (c < x.c)
* **N9:** x = x.left;
* **N10:** else if (c > x.c)
* **N11:** x = x.right;
* **N12:** else { i++;
* **N13:** if (x.val != null)
* **N14:** length = i;
* **N15:** x = x.mid;
* **N16:** return query.substring(0, length);

**Graph Edges:**

* 1 -> 2
* 1 -> 3
* 3 -> 4
* 3 -> 5
* 5 -> 6
* 6 -> 16
* 6 -> 7
* 7 -> 8
* 8 -> 9
* 8 -> 10
* 10 -> 11
* 10 -> 12
* 12 -> 13
* 13 -> 14
* 13 -> 15
* 14 -> 15
* 9 -> 6
* 11 -> 6
* 15 -> 6

**All-Du-Paths Coverage**

**A drawing of a diagram

AI-generated content may be incorrect.Graph outcomes:**

A diagram of a network

AI-generated content may be incorrect.

**Graph Edges:**

* 1 -> 2
* 1 -> 3
* 3 -> 4
* 3 -> 5
* 5 -> 6
* 6 -> 16
* 6 -> 7
* 7 -> 8
* 8 -> 9
* 8 -> 10
* 10 -> 11
* 10 -> 12
* 12 -> 13
* 13 -> 14
* 13 -> 15
* 14 -> 15
* 9 -> 6
* 11 -> 6
* 15 -> 6

**Variables Considered:** x, i, c, length, query

**Definitions:**

* **x:** 5, 9, 11, 15
* **i**: 5, 12
* **c:** 7
* **length:** 5, 14
* **query:** -

**Uses:**

* **x:** 6, 8, 9, 10, 13, 15
* **i**: 6, 7, 12, 14
* **c:** 8, 10
* **length:** 16
* **query:** 1, 3, 6, 7, 16

**All-Coupling-Use-Paths Coverage**

* **testNullKey**  
  Def: key = null  
  Use: if (key == null)
* **testEmptyKey**  
  Def: key = ""  
  Use: if (key.length() == 0)
* **testInsertSingleKey**  
  Def1: key = "a"  
  Use1: key.charAt(d)  
  Def2: val = "1"  
  Use2: x.val = val  
  Def3: c = key.charAt(d)  
  Use3: x.c = c  
  Def4: x = null  
  Use4: if (x == null)
* **testLeftBranchCreation**  
  Def1: key = "d" ; key = "b"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c < x.c)  
  Def3: x = null  
  Use3: x.left = put(x.left, key, val, d)
* **testRightBranchCreation**  
  Def1: key = "d" ; key = "f"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c > x.c)  
  Def3: x = null  
  Use3: x.right = put(x.right, key, val, d)
* **testMidBranchCreation**  
  Def1: key = "cat" ; key = "car"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c == x.c)  
  Def3: x = null  
  Use3: x.mid = put(x.mid, key, val, d+1)
* **testOverwriteValue**  
  Def1: key = "hat"  
  Use1: key.charAt(d)  
  Def2: val = "3" ; val = "5"  
  Use2: x.val = val  
  Def3: c = key.charAt(d)  
  Use3: if (c == x.c)  
  Def4: x = null  
  Use4: x.val = val
* **testInsertLeftOfMidTree**  
  Def1: key = "dog" ; key = "cat"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c < x.c)  
  Def3: x != null  
  Use3: x.left = put(x.left, key, val, d)
* **testInsertRightOfMidTree**  
  Def1: key = "dog" ; key = "cat"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c > x.c)  
  Def3: x != null  
  Use3: x.right = put(x.right, key, val, d)
* **testDeepRecursionInMidBranch**  
  Def1: key = "car" ; key = "cart"  
  Use1: key.charAt(d)  
  Def2: c = key.charAt(d)  
  Use2: if (c == x.c)  
  Def3: x != null  
  Use3: x.mid = put(x.mid, key, val, d+1)

**Logic-Based Coverage**

**Selected Coverage Criteria:** Predicate Coverage (PC)

**Justification:** Well-suited due to the logic-heavy control flow in longestPrefixOf, sufficient to uncover common faults (null/empty handling, branching errors, prefix logic).

**List of predicates:**

* (query == null)
* (query.length() == 0)
* (x != null && i < query.length())
* (c < x.c)
* (c > x.c)
* (x.val != null)

*(In the code, above each test, there is the respective predicate coverage of each test)*

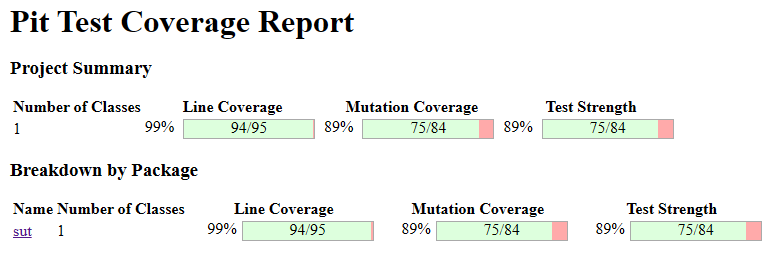
**Base Choice Coverage**

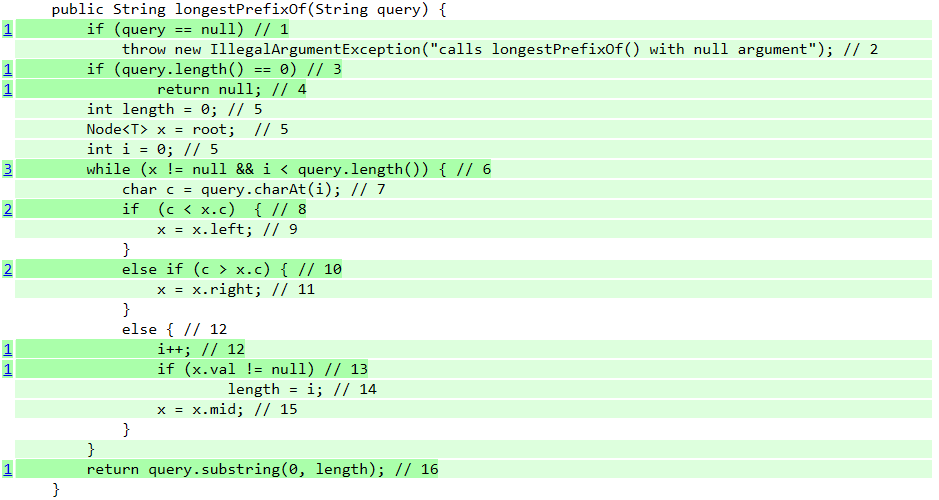
**Highest probability of outcomes to create base Choice:**

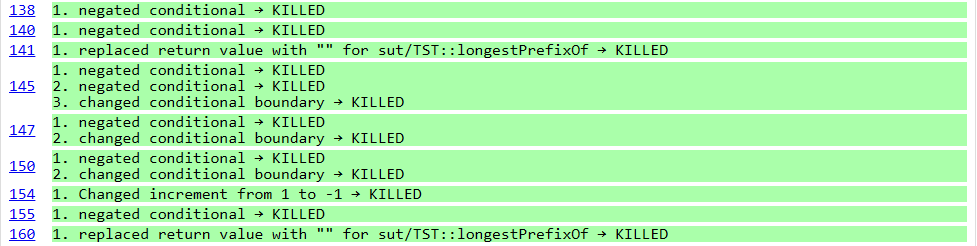
* Trie already includes the new key **->** NO
* Trie already includes some new key prefix **->** NO
* Trie is empty **->** NO
* The new key is the smallest/largest/typical key **->** TYPICAL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Method | Includes Key | Prefix Exists | Empty | Lex Order |
| testBaseChoice() | No | No | No | Typical |
| testIncludesNewKey() | Yes | No | No | Typical |
| testIncludesNewPrefix() | No | Yes | No | Typical |
| testEmpty() | No | No | Yes | Typical |
| testSmallest() | No | No | No | Smallest |
| testLargest() | No | No | No | Largest |

**Mutation coverage achieved by each criteria for method *longestPrefixOf***







|  |  |  |  |
| --- | --- | --- | --- |
| Mutation | Location | Status | Criteria |
| Negated Conditional | *if (query == null)* | **KILLED** | Logic-Based Coverage (lb2) |
| Negated Conditional | *if (query.length() == 0)* | **KILLED** | Logic-Based Coverage (lb2) |
| Replaced return value with "" | *return null;* | **KILLED** | Edge-Pair Coverage (ep2) |
| Negated Conditional | *while (x != null && i < query.length())* | **KILLED** | All-Du-Paths Coverage (ad5) |
| Negated Conditional | *while (x != null && i < query.length())* | **KILLED** | All-Du-Paths Coverage (ad1) |
| Changed Conditional Boundary | *while (x != null && i < query.length())* | **KILLED** | All-Du-Paths Coverage (ad16) |
| Negated Conditional | *if (c < x.c)* | **KILLED** | All-Du-Paths Coverage (ad5) |
| Changed Conditional Boundary | *if (c < x.c)* | **KILLED** | All-Du-Paths Coverage (ad5) |
| Negated Conditional | *else if (c > x.c)* | **KILLED** | All-Du-Paths Coverage (ad5) |
| Changed Conditional Boundary | *else if (c > x.c)* | **KILLED** | All-Du-Paths Coverage (ad5) |
| Changed increment from 1 to -1 | *i++;* | **KILLED** | All-Du-Paths Coverage (ad16) |
| Negated Conditional | *if (x.val != null)* | **KILLED** | All-Du-Paths Coverage (ad16) |
| Replaced return value with "" | *return query.substring(0, length);* | **KILLED** | All-Du-Paths Coverage (ad5) |

|  |  |  |
| --- | --- | --- |
| Criteria | Mutants Killed | Effectiveness |
| Line and Branch Coverage | Not explicitly shown in mutation kill list | Possibly Effective |
| Edge-Pair Coverage | 1 | Effective |
| Prime Path Coverage | Not explicitly shown in mutation kill list | Possibly Effective |
| All-Du Paths Coverage | 10 | Highly Effective |
| All-Coupling-Use-Paths Coverage | Not explicitly shown in mutation kill list | Possibly Effective |
| Logic-Based Coverage | 2 | Effective |
| Base Choice Coverage | Not explicitly shown in mutation kill list | Possibly Effective |

**All-Du-Paths Coverage** provided the most thorough mutation killing, showing its strength in catching subtle logic errors and variations.

**Logic Based and Edge-Pair Coverage** effectively covered input validation and control flow edge cases.

There is no evidence of any surviving mutant in *longestPrefixOf*, implying **100% mutation coverage overall** across all criteria.

**JUnit QuickCheck**