

Relatório de VVS01

Gabriel Henriques - 58182

Line Coverage

O *Line Coverage* foi atingido em todos os métodos públicos ao fazer o mínimo de testes necessários que passassem pelo menos uma vez por cada linha de código. Todos os métodos obtiveram coverage verde com exceção do *longestPrefixOf* e *put*, pois não foram feitos testes suficientes para que passassem em todas branches.

```
public String longestPrefixOf(String query) {
    if (query == null) //n1 P1:C1
        throw new IllegalArgumentException("calls longestPrefixOf() with null argument"); //2
    if (query.length() == 0) //n3 P2:C2
        return null; //n4
    int length = 0; //n5
    Node<T> x = root; //n5
    int i = 0; //n5
    while (x != null && i < query.length()) { //n6 P3:C3,C4
        char c = query.charAt(i); //n7
        if (c < x.c) /*n8 P4:C5 */ x = x.left; //n9
        else if (c > x.c) /*n10 P5:C6 */ x = x.right; //n11
        else {
            i++; //n12
            if (x.val != null) //n13 P6:C7
                length = i; //n14
            x = x.mid; //n15
        }
    }
    return query.substring(0, length); //16
}
```

```
public void put(String key, T val) {
    if (key == null)
        throw new IllegalArgumentException("calls put() with null key");
    if (!contains(key))
        n++;
    root = put(root, key, val, 0);
}
```

Branch Coverage

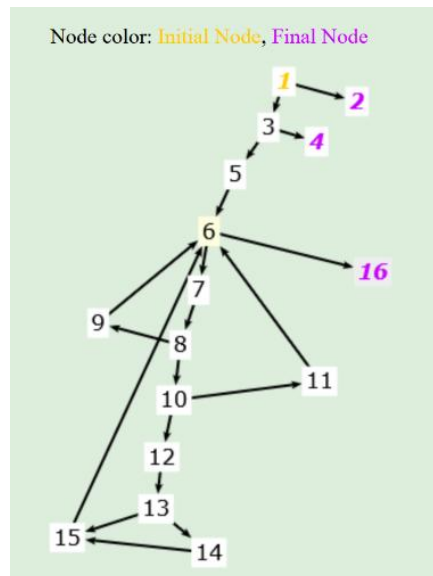
No *Branch Coverage*, foram feitos todos os testes já escritos no *Line Coverage*, com a adição de mais 3 testes no *longestPrefixOf* e mais 1 teste no *put*, de modo a satisfazer a coverage que antes aparecia em amarelo, por não serem testadas todas as branches dos métodos. Demonstrou-se então que o *Branch Coverage* testa mais exaustivamente o sistema, sem necessitar de uma maior análise em relação ao *Line Coverage*.

```
public String longestPrefixOf(String query) {
    if (query == null) //n1 P1:C1
        throw new IllegalArgumentException("calls longestPrefixOf() with null argument"); //2
    if (query.length() == 0) //n3 P2:C2
        return null; //n4
    int length = 0; //n5
    Node<T> x = root; //n5
    int i = 0; //n5
    while (x != null && i < query.length()) { //n6 P3:C3,C4
        char c = query.charAt(i); //n7
        if (c < x.c) /*n8 P4:C5 */ x = x.left; //n9
        else if (c > x.c) /*n10 P5:C6 */ x = x.right; //n11
        else {
            i++; //n12
            if (x.val != null) //n13 P6:C7
                length = i; //n14
            x = x.mid; //n15
        }
    }
    return query.substring(0, length); //16
}

public void put(String key, T val) {
    if (key == null)
        throw new IllegalArgumentException("calls put() with null key");
    if (!contains(key))
        n++;
    root = put(root, key, val, 0);
}
```

Edge-Pair Coverage

Para satisfazer o *Edge Pair Coverage*, foi necessário fazer o grafo do método *longestPrefixOf* tal como representado na seguinte imagem:



Após a criação do grafo, foi inserido no site <https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage> os seus edges, initial node e final nodes, e gerado pelo Algoritmo 2 os paths a percorrer pelos testes de modo a satisfazer o *Edge Pair Coverage*:

| Test Paths | Test Requirements that are toured by test paths directly |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| [1,3,5,6,7,8,10,12,13,15,6,16] | [1,3,5], [3,5,6], [5,6,7], [6,7,8], [7,8,10], [8,10,12], [10,12,13], [12,13,15], [13,15,6], [15,6,16] |
| [1,3,5,6,7,8,9,6,7,8,10,11,6,16] | [1,3,5], [3,5,6], [5,6,7], [6,7,8], [7,8,9], [7,8,10], [8,9,6], [8,10,11], [9,6,7], [10,11,6], [11,6,16] |
| [1,3,5,6,7,8,10,11,6,7,8,9,6,16] | [1,3,5], [3,5,6], [5,6,7], [6,7,8], [7,8,9], [7,8,10], [8,9,6], [8,10,11], [9,6,16], [10,11,6], [11,6,7] |
| [1,3,5,6,7,8,10,12,13,14,15,6,7,8,9,6,16] | [1,3,5], [3,5,6], [5,6,7], [6,7,8], [7,8,9], [7,8,10], [8,9,6], [8,10,12], [9,6,16], [10,12,13], [12,13,14], [13,14,15], [14,15,6], [15,6,7] |
| [1,3,5,6,16] | [1,3,5], [3,5,6], [5,6,16] |
| [1,2] | [1,2] |
| [1,3,4] | [1,3,4] |

Prime Path Coverage

Para o Prime Path Coverage do método *longestPrefixOf*, foi feito o mesmo procedimento, mas foi escolhido o Algoritmo 1 que gerou mais paths, porém mais curtos, reduzindo assim a complexidade na escrita dos testes:

| Test Paths | Test Requirements that are toured by test paths directly |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [1,3,5,6,7,8,10,12,13,14,15,6,7,8,9,6,16] | [1,3,5,6,7,8,10,12,13,14,15], [6,7,8,10,12,13,14,15,6], [7,8,10,12,13,14,15,6,7], [8,10,12,13,14,15,6,7,8], [10,12,13,14,15,6,7,8,9], [7,8,9,6,16], [6,7,8,9,6] |
| [1,3,5,6,7,8,9,6,7,8,10,12,13,14,15,6,16] | [7,8,10,12,13,14,15,6,16], [6,7,8,10,12,13,14,15,6], [9,6,7,8,10,12,13,14,15], [1,3,5,6,7,8,9], [7,8,9,6,7], [8,9,6,7,8], [6,7,8,9,6] |
| [1,3,5,6,7,8,10,12,13,14,15,6,7,8,10,12,13,15,6,16] | [1,3,5,6,7,8,10,12,13,14,15], [6,7,8,10,12,13,14,15,6], [7,8,10,12,13,14,15,6,7], [8,10,12,13,14,15,6,7,8], [13,14,15,6,7,8,10,12,13], [12,13,14,15,6,7,8,10,12], [10,12,13,14,15,6,7,8,10], [7,8,10,12,13,15,6,16], [6,7,8,10,12,13,15,6], [15,6,7,8,10,12,13,15] |
| [1,3,5,6,7,8,10,12,13,14,15,6,7,8,10,12,13,14,15,6,16] | [1,3,5,6,7,8,10,12,13,14,15], [7,8,10,12,13,14,15,6,16], [6,7,8,10,12,13,14,15,6], [7,8,10,12,13,14,15,6,7], [8,10,12,13,14,15,6,7,8], [13,14,15,6,7,8,10,12,13], [13,14,15,6,7,8,10,12,13], [12,13,14,15,6,7,8,10,12], [14,15,6,7,8,10,12,13,14], [15,6,7,8,10,12,13,14,15], [10,12,13,14,15,6,7,8,10] |
| [1,3,5,6,7,8,10,12,13,15,6,7,8,10,12,13,14,15,6,16] | [1,3,5,6,7,8,10,12,13,15], [7,8,10,12,13,14,15,6,16], [6,7,8,10,12,13,14,15,6], [15,6,7,8,10,12,13,14,15], [7,8,10,12,13,15,6,7], [8,10,12,13,15,6,7,8], [6,7,8,10,12,13,15,6], [12,13,15,6,7,8,10,12], [13,15,6,7,8,10,12,13], [10,12,13,15,6,7,8,10] |
| [1,3,5,6,7,8,10,12,13,14,15,6,7,8,10,11,6,16] | [1,3,5,6,7,8,10,12,13,14,15], [6,7,8,10,12,13,14,15,6], [7,8,10,12,13,14,15,6,7], [8,10,12,13,14,15,6,7,8], [12,13,14,15,6,7,8,10,11], [10,12,13,14,15,6,7,8,10], [7,8,10,11,6,16], [6,7,8,10,11,6] |
| [1,3,5,6,7,8,10,11,6,7,8,10,12,13,14,15,6,16] | [7,8,10,12,13,14,15,6,16], [6,7,8,10,12,13,14,15,6], [11,6,7,8,10,12,13,14,15], [1,3,5,6,7,8,10,11], [8,10,11,6,7,8], [7,8,10,11,6,7], [6,7,8,10,11,6], [10,11,6,7,8,10] |
| [1,3,5,6,7,8,10,12,13,15,6,7,8,9,6,16] | [1,3,5,6,7,8,10,12,13,15], [7,8,10,12,13,15,6,7], [8,10,12,13,15,6,7,8], [6,7,8,10,12,13,15,6], [10,12,13,15,6,7,8,9], [7,8,9,6,16], [6,7,8,9,6] |
| [1,3,5,6,7,8,9,6,7,8,10,12,13,15,6,16] | [7,8,10,12,13,15,6,16], [9,6,7,8,10,12,13,15], [6,7,8,10,12,13,15,6], [1,3,5,6,7,8,9], [7,8,9,6,7], [8,9,6,7,8], [6,7,8,9,6] |
| [1,3,5,6,7,8,10,12,13,15,6,7,8,10,12,13,15,6,16] | [1,3,5,6,7,8,10,12,13,15], [7,8,10,12,13,15,6,7], [7,8,10,12,13,15,6,16], [8,10,12,13,15,6,7,8], [6,7,8,10,12,13,15,6], [12,13,15,6,7,8,10,12], [13,15,6,7,8,10,12,13], [15,6,7,8,10,12,13,15], [10,12,13,15,6,7,8,10] |
| [1,3,5,6,7,8,10,12,13,15,6,7,8,10,11,6,16] | [1,3,5,6,7,8,10,12,13,15], [7,8,10,12,13,15,6,7], [8,10,12,13,15,6,7,8], [6,7,8,10,12,13,15,6], [12,13,15,6,7,8,10,11], [10,12,13,15,6,7,8,10], [7,8,10,11,6,16], [6,7,8,10,11,6] |
| [1,3,5,6,7,8,10,11,6,7,8,10,12,13,15,6,16] | [7,8,10,12,13,15,6,16], [1,3,5,6,7,8,10,11], [6,7,8,10,12,13,15,6], [11,6,7,8,10,12,13,15], [8,10,11,6,7,8], [7,8,10,11,6,7], [6,7,8,10,11,6], [10,11,6,7,8,10] |
| [1,3,5,6,7,8,10,11,6,7,8,9,6,16] | [1,3,5,6,7,8,10,11], [8,10,11,6,7,8], [7,8,10,11,6,7], [6,7,8,10,11,6], [10,11,6,7,8,9], [7,8,9,6,16], [6,7,8,9,6] |
| [1,3,5,6,7,8,9,6,7,8,10,11,6,16] | [1,3,5,6,7,8,9], [9,6,7,8,10,11], [7,8,10,11,6,16], [6,7,8,10,11,6], [7,8,9,6,7], [8,9,6,7,8], [6,7,8,9,6] |
| [1,3,5,6,7,8,10,11,6,7,8,10,11,6,16] | [1,3,5,6,7,8,10,11], [8,10,11,6,7,8], [7,8,10,11,6,7], [7,8,10,11,6,16], [6,7,8,10,11,6], [10,11,6,7,8,10], [11,6,7,8,10,11] |
| [1,3,5,6,7,8,9,6,7,8,9,6,16] | [1,3,5,6,7,8,9], [7,8,9,6,7], [7,8,9,6,16], [8,9,6,7,8], [6,7,8,9,6], [9,6,7,8,9] |
| [1,3,5,6,16] | [1,3,5,6,16] |
| [1,3,4] | [1,3,4] |
| [1,2] | [1,2] |

All-Du-Paths Coverage

No *All-Du-Paths Coverage* para o *longestPrefixOf*, voltou-se a usar o mesmo grafo já criado anteriormente, mas desta vez foi também analisado em que *nodes* e *edges* cada variável é definida e usada, obtendo a seguinte tabela:

| Nodes & Edges: l | def(l) | use(l) |
|------------------|-------------|---------------|
| 1 | {query} | |
| (1,2), (1,3) | | {query} |
| 3 | | |
| (3,4), (3,5) | | {query} |
| 5 | {len, x, i} | |
| (5,6) | | |
| 6 | | |
| (6,7), (6,16) | | {x, i, query} |
| 7 | {c} | {query, i} |
| (7,8) | | |
| 8 | | |
| (8,9), (8,10) | | {c, x} |
| 9 | {x} | {x} |
| (9,6) | | |
| 10 | | |
| (10,11), (10,12) | | {c, x} |
| 11 | {x} | {x} |
| (11,6) | | |
| 12 | {i} | {i} |
| (12,13) | | |
| 13 | | |
| (13,14), (13,15) | | {x} |
| 14 | {len} | {i} |
| (14,15) | | |
| 15 | {x} | {x} |
| (15,6) | | |
| 16 | | {query, len} |

Posteriormente, foi inserido no site

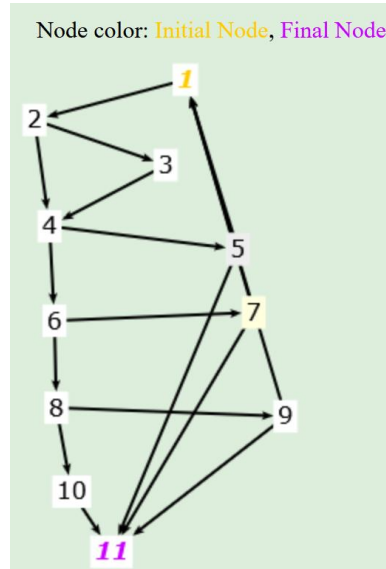
<https://cs.gmu.edu:8443/offutt/coverage/DFGraphCoverage> as edges, initial node, final nodes, defs e uses das variáveis e gerado os test paths para cada variável:

| Variable | All DU Path Coverage |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query | [1,3,4] [1,2] [1,3,5,6,16] [1,3,5,6,7,8,9,6,16] |
| length | [1,3,5,6,16] [1,3,5,6,7,8,10,12,13,14,15,6,16] |
| x | [1,3,5,6,7,8,9,6,16] [1,3,5,6,16] [1,3,5,6,7,8,10,11,6,16] [1,3,5,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,12,13,14,15,6,16] [1,3,5,6,7,8,9,6,7,8,9,6,16] [1,3,5,6,7,8,9,6,16] [1,3,5,6,7,8,9,6,7,8,10,11,6,16] [1,3,5,6,7,8,9,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,9,6,7,8,10,12,13,14,15,6,16] [1,3,5,6,7,8,10,11,6,7,8,9,6,16] [1,3,5,6,7,8,10,11,6,16] [1,3,5,6,7,8,10,11,6,7,8,10,11,6,16] [1,3,5,6,7,8,10,11,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,11,6,7,8,10,12,13,14,15,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,9,6,16] [1,3,5,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,10,11,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,10,12,13,14,15,6,16] |
| i | [1,3,5,6,7,8,9,6,16] [1,3,5,6,16] [1,3,5,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,12,13,14,15,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,9,6,16] [1,3,5,6,7,8,10,12,13,14,15,6,7,8,9,6,16] [1,3,5,6,7,8,10,12,13,15,6,7,8,10,12,13,15,6,16] [1,3,5,6,7,8,10,12,13,14,15,6,7,8,10,12,13,15,6,16] |
| c | [1,3,5,6,7,8,10,11,6,16] [1,3,5,6,7,8,9,6,16] [1,3,5,6,7,8,10,12,13,15,6,16] |

Algumas variáveis tinham test path iguais, fazendo com que um mesmo path conseguisse satisfazer diversos requisitos, então não foi necessário escrever tantos testes unitários quantos os gerados.

All-Coupling-Use-Paths Coverage

Para satisfazer o *All-Coupling-Use-Paths*, foi necessário analisar o método *put* e criar uma representação dele em grafo. As edges dos nodes 5,7 e 9 para o node 1 representam as chamadas recursivas.



Com o grafo feito, foi necessário analisar em que nodes são feitas as últimas definições das variáveis antes das chamadas recursivas, e também em que nodes da função chamada são primeiramente usadas as variáveis passadas, obtendo a seguinte tabela:

| Last-def | First-use |
|----------------------|----------------------|
| key: {1} | key: {i} |
| val: {1} | val: {v, vii, ix, x} |
| x: {1,3} | x: {ii} |
| d: {1,9} | d: {i} |
| x' : {v, vii, ix, x} | x' : {11} |

Com estas informações, foi então possível verificar que paths são necessários os testes percorrerem de modo a satisfazer a coverage *All-Coupling-Use-Paths*:

Paths:

| | | | | |
|-----|--------------------|----------------------|---------------------|--------------------|
| key | key {1} -> key {i} | | | |
| val | val {1} -> val {v} | val {1} -> val {vii} | val {1} -> val {ix} | val {1} -> val {x} |
| x | x {1} -> x {ii} | x {3} -> x {ii} | | |
| d | d {1} -> d {i} | d {9} -> d {i} | | |
| x' | x' {v} -> x' {11} | x' {vii} -> x' {11} | x' {ix} -> x' {11} | x' {x} -> x' {11} |

Logic-based Coverage

Como o método *longestPrefixOf* tem predicados simples, onde apenas um deles tem mais que uma cláusula, foi decidido não complicar demasiado e escolher um dos critérios básicos. Neste caso, foi escolhido o *Combinatorial Coverage* por ser o mais abrangente sem complicar demasiado a realização dos testes.

Base Choice Coverage

No Base Choice Coverage, cada uma das características foi subdividida binariamente, com exceção da última que foi ternariamente, e foram realizados testes que testam individualmente cada uma das partições das características.

PIT Mutation Coverage

Ao correr as mutações do programa através do PIT, foi possível em (quase) todas as *Coverage Criteria* do *longestPrefixOf* matar os mutantes gerados:

```
135 public String longestPrefixOf(String query) {
136 1 if (query == null) //n1 P1:C1
137     throw new IllegalArgumentException("calls longestPrefixOf() with null argument"); //2
138 1 if (query.length() == 0) //n3 P2:C2
139 1     return null; //n4
140     int length = 0; //n5
141     Node<T> x = root; //n5
142     int i = 0; //n5
143 3 while (x != null && i < query.length()) { //n6 P3:C3,C4
144     char c = query.charAt(i); //n7
145 2 if (c < x.c) /*n8 P4:C5 */ x = x.left; //n9
146 2 else if (c > x.c) /*n10 P5:C6 */ x = x.right; //n11
147     else {
148 1         i++; //n12
149 1         if (x.val != null) //n13 P6:C7
150             length = i; //n14
151         x = x.mid; //n15
152     }
153 }
154 1 return query.substring(0, length); //16
```

Com exceção da *Line Coverage* que não foi capaz de matar o seguinte mutante:

```
136 1 if (query == null) //n1 P1:C1
137     throw new IllegalArgumentException("calls longestPrefixOf() with null argument"); //2
138 1 if (query.length() == 0) //n3 P2:C2
139 1     return null; //n4
140     int length = 0; //n5
141     Node<T> x = root; //n5
142     int i = 0; //n5
143 3 while (x != null && i < query.length()) { //n6 P3:C3,C4
144     char c = query.charAt(i); //n7
145 2 if (c < x.c) /*n8 P4:C5 */ x = x.left; //n9
146 2 else if (c > x.c) /*n10 P5:C6 */ x = x.right; //n11
147     else {
148 1         i++; //n12
149 1         if (x.val != null) //n13 P6:C7
150             length = i; //n14
151         x = x.mid; //n15
152     }
153 }
154 1 return query.substring(0, length); //16
```

- | | |
|-----|--------------------------------------------|
| 143 | 1. changed conditional boundary → SURVIVED |
| | 2. negated conditional → KILLED |
| | 3. negated conditional → KILLED |

Este mutante poderia ter sido detetado e morto com a adição do seguinte teste:

```
@Test
public void testWithQueryPrefixOfPut() {
    // Test added after PIT mutation testing
    // lines 1,2,4,6-20
    TST<Integer> tst = new TST<>();
    tst.put("c", 1);
    tst.put("cab", 1);
    assertEquals("c", tst.longestPrefixOf("ca"));
}
```

JUnit QuickCheck

Para realizar os testes de QuickCheck, foi necessário acrescentar três métodos ao sistema: *equals()*, *delete()* e *clone()*. Foi também preciso criar três classes de geradores: *TrieGenerator*, *KeyGenerator* e *KeyListGenerator*.

Para testar a propriedade “The order of insertion of different keys does not change the final tree value” foi necessário receber uma *Trie*, uma lista de *Keys*, e um valor gerado aleatoriamente. Foi inicialmente clonada a *Trie* recebida e de seguida foram adicionadas as *Keys* à *Trie* original. Depois a ordem das *Keys* foi baralhada e foram adicionadas à *Trie* clonada. Por fim, comparou-se a *Trie* original com a *Trie* clonada.

Para testar a propriedade “If you remove all keys from a tree, the tree must be empty” foi necessário receber uma *Trie* gerada aleatoriamente. Foram removidas todas as *Keys* presentes na *Trie*, e de seguida comparou-se a *Trie* com uma *Trie* iniciada vazia, e também se verificou que o seu *size* é 0.

Para testar a propriedade “Given a tree, inserting and then removing the same key value will not change its initial value” foi necessário receber uma *Trie*, uma *Key*, e um valor gerado aleatoriamente. Foi inicialmente clonada a *Trie* recebida, e de seguida foi adicionada a *Key* com o dado valor, e removida. Por fim comparou-se a *Trie* inicial (clonada) com a *Trie* final. Esta **propriedade não é verdadeira** para *Tries* que já contenham o valor a ser adicionado/removido, pois, após a adição/remoção, a *Trie* inicial seria diferente da final.

Para testar a propriedade “Selecting a stricter prefix keysWithPrefix returns a strict subset result” foi necessário receber uma *Trie*, uma lista de *Keys*, e um valor gerado aleatoriamente. Foram inicialmente buscadas todas as *Keys* da *Trie*, e uma delas foi selecionada aleatoriamente para ser a *Key* base. De seguida, foram adicionadas à *Trie* novas *Keys* aleatórias todas com o prefixo da *Key* base e guardou-se o *keysWithPrefix(base_key)*. Por fim, foi verificado que com todos os prefixos possíveis da *Key* base (removendo caracter a caracter) obtém-se sempre um subconjunto de *keysWithPrefix(base_key)*.

Conclusão:

Executando todos os testes, o sistema ficou com uma coverage total de 80,2%. Faltou testar o método privado *collect* e os métodos, posteriormente adicionados, *equals* e *delete*. O método privado *get* tem uma linha de código impossível de alcançar, pois é relacionada com um check já feito previamente no seu método público.

| Element | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
|------------------------------------------|----------|----------------------|---------------------|--------------------|
| ▼ VVS_assignment_1 | | 2 940 | 325 | 3 265 |
| ▼ src/main/java | | 461 | 237 | 698 |
| > startup | | 0 | 123 | 123 |
| ▼ sut | | 461 | 114 | 575 |
| ▼ TST.java | | 461 | 114 | 575 |
| ▼ TST<T> | | 461 | 114 | 575 |
| ■ collect(Node<T>, StringBuilder, int) | | 3 | 90 | 93 |
| ● delete(String) | | 20 | 11 | 31 |
| ● equals(Object) | | 44 | 8 | 52 |
| ■ get(Node<T>, String, int) | | 50 | 5 | 55 |
| ● clone() | | 24 | 0 | 24 |
| ■ collect(Node<T>, StringBuilder, Queue) | | 48 | 0 | 48 |
| ● contains(String) | | 15 | 0 | 15 |
| ● get(String) | | 29 | 0 | 29 |
| ● keys() | | 14 | 0 | 14 |
| ● keysThatMatch(String) | | 16 | 0 | 16 |
| ● keysWithPrefix(String) | | 40 | 0 | 40 |
| ● longestPrefixOf(String) | | 60 | 0 | 60 |
| ■ put(Node<T>, String, T, int) | | 65 | 0 | 65 |
| ● put(String, T) | | 27 | 0 | 27 |
| ● size() | | 3 | 0 | 3 |