

TST JUnit Testing

Software Verification and Validation

2019–2020

João David
49448

Ye Yang
49521

09/05/2020

1 Instruction Coverage

1.1 size()

```
public int size() {  
    return n; //I1  
}
```

Test Case	Values	Expected / Actual	IC
sizeZeroTest	-	0	I1

1.2 contains(String key)

```
public boolean contains(String key) {  
    if (key == null) //I1  
        throw new IllegalArgumentException("argument to  
            contains() is null"); //I2  
    return get(key) != null; //I3  
}
```

Test Case	Values	Expected / Actual	IC
containsNullKey	null	IAE	I1, I2
containsNonNullKey	"someKey"	false	I1, I3

1.3 get(String key)

```

public T get(String key) {
    if (key == null) //I1
        throw new IllegalArgumentException("calls get() with
            null argument"); //I2
    if (key.length() == 0) //I3
        throw new IllegalArgumentException("key must have
            length >= 1"); //I4
    Node<T> x = get(root, key, 0); //I5
    if (x == null) //I6
        return null; //I7
    return x.val; //I8
}

```

Test Case	Values	Expected / Actual	IC
getNullKey	null	IAE	I1, I2
getEmptyStringKey	""	IAE	I1, I3, I4
getNonExistentKey	"someKey"	null	I1, I3, I5, I6, I7
getExistentKey	"key"	<value>	I1, I3, I5, I6, I8

1.4 put(String key, T val)

```

public void put(String key, T val) {
    if (key == null) //I1
        throw new IllegalArgumentException("calls put() with
            null key"); //I2
    if (!contains(key)) //I3
        n++; //I4
    root = put(root, key, val, 0); //I5
}

```

Test Case	Values	Expected / Actual	IC
putNullKey	null, 1	IAE	I1, I2
putValidNewKey	"someKey", 1	NoExep	I1, I3, I4, I5

1.5 longestPrefixOf(String query)

```

public String longestPrefixOf(String query) {
    if (query == null) //I1
        throw new IllegalArgumentException("calls
            longestPrefixOf() with null argument"); //I2
    if (query.length() == 0) //I3
        return null; //I4
    int length = 0; //I5
    Node<T> x = root; //I6
    int i = 0; //I7
    while (x != null /*I8*/ && i < query.length() /*I9*/) {
        char c = query.charAt(i); //I10
        if (c < x.c) /*I11*/ x = x.left; //I12
        else if (c > x.c) /*I13*/ x = x.right; //I14
        else {
            i++; //I15
            if (x.val != null) //I15
                length = i; //I17
            x = x.mid; //I18
        }
    }
    return query.substring(0, length); //I19
}

```

Test Case	Values	Expected / Actual	IC
longestPrefixOfNull	null	IAE	I1, I2
longestPrefixOf EmptyString	""	null	I1, I3, I4
longestPrefixOf AllInstructions	"c"	"c"	I1, I3, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, I15, I16, I17, I18, I19

1.6 keys()

```
public Iterable<String> keys() {
    Queue<String> queue = new LinkedList<>(); //I1
    collect(root, new StringBuilder(), queue); //I2
    return queue; //I3
}
```

Test Case	Values	Expected / Actual	IC
keysTest	-	Empty Iterator	I1, I2, I3

1.7 keysWithPrefix(String prefix)

```
public Iterable<String> keysWithPrefix(String prefix) {
    if (prefix == null) //I1
        throw new IllegalArgumentException("calls
            keysWithPrefix() with null argument"); //I2
    Queue<String> queue = new LinkedList<>(); //I3
    Node<T> x = get(root, prefix, 0); //I4
    if (x == null) //I5
        return queue; //I6
    if (x.val != null) //I7
        queue.add(prefix); //I8
        collect(x.mid, new StringBuilder(prefix), queue);
        //I9
    return queue; //I10
}
```

Test Case	Values	Expected / Actual	IC
keysWithPrefixNull	null	IAE	I1, I2
keysWithPrefix NonExistentPrefix	"prefix"	Iterator (size 0)	I1, I3, I4, I5, I6
keysWithPrefix ExistentPrefix	"c"	Iterator (size 1)	I1, I3, I4, I5, I6, I7, I8, I9, I10

1.8 keysThatMatch(String pattern)

```
public Iterable<String> keysThatMatch(String pattern) {
    Queue<String> queue = new LinkedList<>(); //I1
    collect(root, new StringBuilder(), 0, pattern, queue);
    //I2
    return queue; //I3
}
```

Test Case	Values	Expected / Actual	IC
keysThatMatchTest	"pattern"	Iterator (size 0)	I1, I2, I3

1.9 delete(String key)

```
public void delete(String key) {
    if (key == null) { //I1
        throw new IllegalArgumentException("calls put() with
            null key"); //I2
    }
    if (contains(key)) { //I3
        n--; //I4
        put(root, key, null, 0); //I5
    }
}
```

Test Case	Values	Expected / Actual	IC
deleteNull	null	IAE	I1, I2
deleteContains	"key"	Iterator (size 0)	I3, I4, I5

1.10 equals(Object obj)

```
public boolean equals(Object obj) {
    if (this == obj) //I1
        return true; //I2
    if (obj == null) //I3
        return false; //I4
}
```

```

    if (!(obj instanceof TST<?>)) //I5
        return false; //I6

    TST<T> other = (TST<T>) obj; //I7
    if (this.size() != other.size()) { //I8
        return false; //I9
    }

    Iterable<String> thisIterable = this.keys(); //I10
    for(String currKey : thisIterable){ //I11
        if(!this.get(currKey).equals(other.get(currKey))){
            //I12
            return false; //I13
        }
    }
    return true; //I14
}

```

Test Case	Values	Expected / Actual	IC
equalsSame	Same trie object	True	I1, I2
equalsNull	null	False	I1, I3, I4
equalsNotTrie	1	False	I1, I3, I5, I6
equalsDifSize	Trie with more keys	False	I1, I3, I5, I7, I8, I9
equalsDifContent	Trie with same keys	True	I1, I3, I5, I7, I8, I10, I11, I12, I14

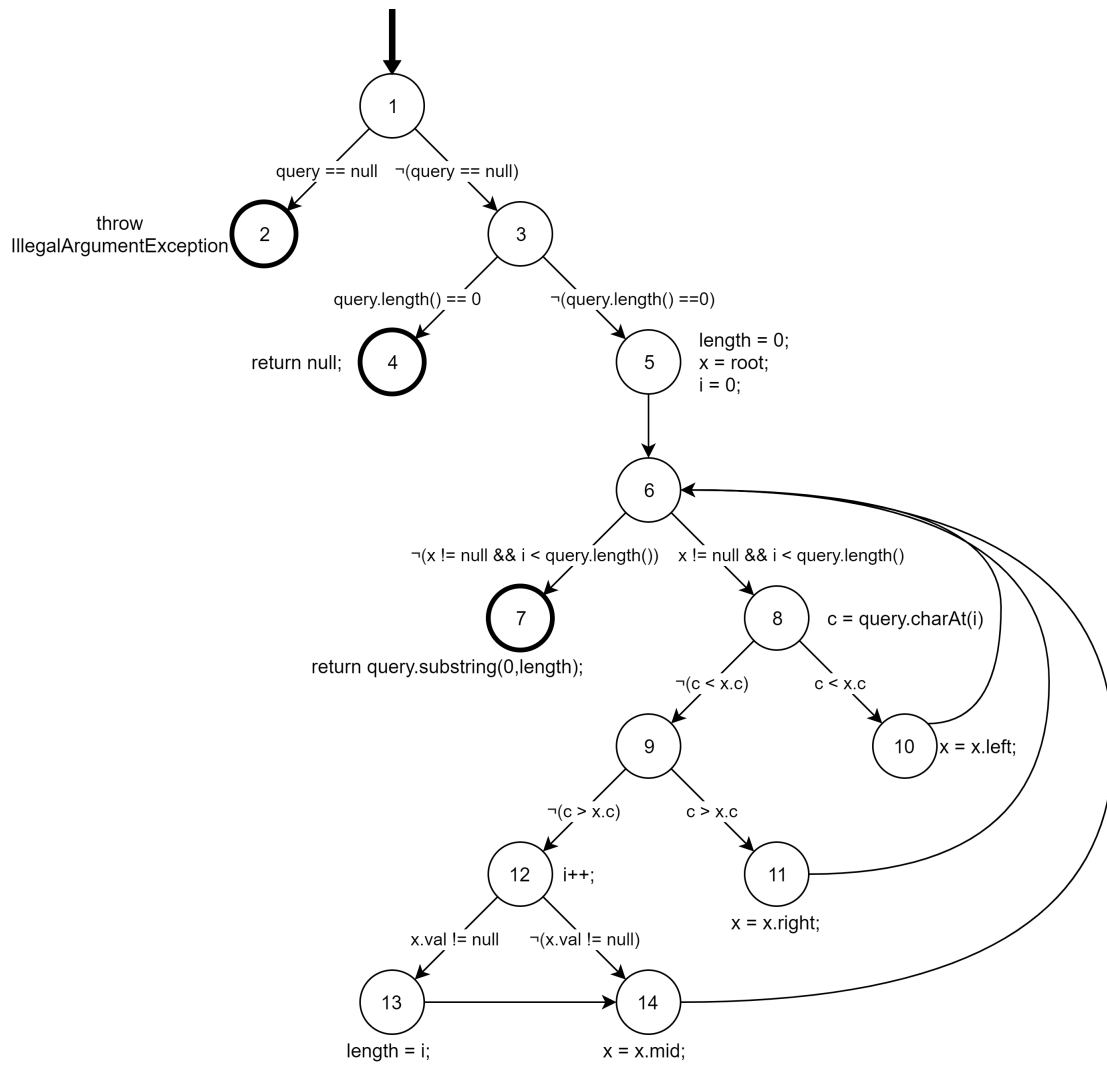


Figure 1: longestPrefixOf's Graph

2 Edge Coverage

3 Prime Path Coverage

4 All-Uses Coverage

nodes & edges : I	def(I)	use(I)
1	{root, query}	{}
(1,2), (1,3)	{}	{query}
3	{}	{}
(3,4), (3,5)	{}	{query}
4	{}	{}
5	{length, x, i}	{root}
(5,6)	{}	{}
6	{}	{}
(6,7), (6,8)	{}	{x, i, query}
7	{}	{query, length}
8	{c}	{i}
(8,9), (8,10)	{}	{c, x}
9	{}	{}
10	{x}	{x}
(10,6), (11,6), (14,6)	{}	{}
(9,11), (9,12)	{}	{c, x}
12	{x}	{x}
12	{i}	{i}
(12,13), (12,14)	{}	{x}
13	{length}	{i}
14	{x}	{x}

var	node	du(node,var)
query	1	[1,2], [1,3], [1,3,4], [1,3,5], [1,3,5,6,7], [1,3,5,6,8]
root	1	[1,3,5]
length	5	[5,6,7]
	13	[13,14,6,7]
x	5	[5,6,7], [5,6,8], [5,6,8,10], [5,6,8,9], [5,6,8,9,11] [5,6,8,9,12], [5,6,8,9,12,13], [5,6,8,9,12,13,14], [5,6,8,9,12,14]
	10	[10,6,7], [10,6,8], [10,6,8,10], [10,6,8,9], [10,6,8,9,11], [10,6,8,9,12], [10,6,8,9,12,13] [10,6,8,9,12,13,14], [10,6,8,9,12,14]
	11	[11,6,7], [11,6,8], [11,6,8,10], [11,6,8,9], [11,6,8,9,11], [11,6,8,9,12], [11,6,8,9,12,13] [11,6,8,9,12,13,14], [11,6,8,9,12,14]
	14	[14,6,7], [14,6,8], [14,6,8,10], [14,6,8,9] [14,6,8,9,11], [14,6,8,9,12], [14,6,8,9,12,13] [14,6,8,9,12,13,14], [14,6,8,9,12,14]
i	5	[5,6,7], [5,6,8], [5,6,8,9,12], [5,6,8,9,12,13]
	12	[12,13], [12,13,14,6,7], [12,13,14,6,8] [12,13,14,6,8,9,12], [12,14,6,7], [12,14,6,8], [12,14,6,8,9,12]
c	8	[8,9], [8,10], [8,9,11], [8,9,12]

Test	put ops	Values	Expected	Test Path
1	{ }	null	IAE	[1,2]
2	{ }	""	null	[1,3,4]
3	{ }	"query"	""	[1,3,5,6,7]
4	{ sea }	"a"	""	[1,3,5,6,8,10,6,7]
5	{ sea }	"t"	""	[1,3,5,6,8,9,11,6,7]
6	{ sea, s, e, a }	"sea"	"sea"	[1,3,5,6,8,9,12,13,14,6,8,9,12,14,6,8,9,12,13,14,6,7]
7	{ sea, t, a }	"set"	""	[1,3,5,6,8,9,12,14,6,8,9,12,14,6,8,9,11,6,7]
8	{ sea, ball, c }	"c"	"c"	[1,3,5,6,8,10,6,8,9,11,6,8,9,12,13,14,6,7]
9	{ sea, cat, b }	"b"	"b"	[1,3,5,6,8,10,6,8,10,6,8,9,12,13,14,6,7]
10	{ sea, up, w }	"w"	"w"	[1,3,5,6,8,9,11,6,8,9,11,6,8,9,12,13,14,6,7]
11	{ sea }	"sd"	""	[1,3,5,6,8,9,12,14,6,8,10,6,7]
12	{ sea }	"su"	""	[1,3,5,6,8,9,12,14,6,8,9,11,6,7]
13	{ sea, w }	"t"	""	[1,3,5,6,8,9,11,6,8,10,6,7]

5 Logic-based Coverage

References

- [1] SiLK - CERT NetSA <https://tools.netsa.cert.org/silk/docs.html>
- [2] List of TCP and UDP port numbers https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers