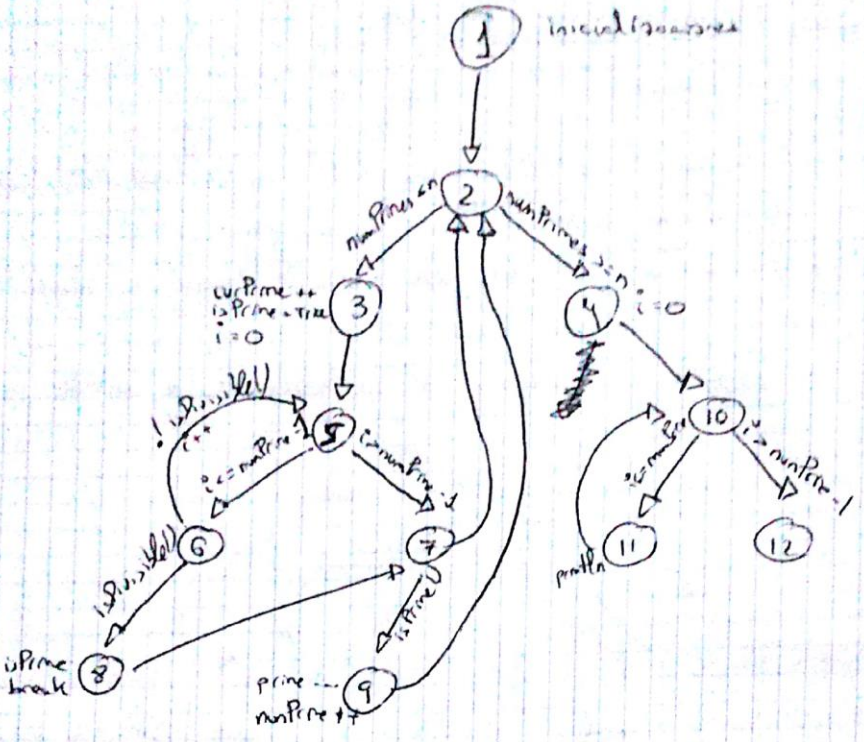
PrintPrimes ()



(4)

Node Coverage → { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }

Edge Coverage → { (1,2), (2,3), (2,4), (3,5), (5,6), (6,5), (5,7), (6,8), (8,7), (7,9), (7,2), (9,2), (4,10), (10,11), (11,10), (10,12) }

Prime Path → (1,2, 3,5, 6, 8, 7, 9), (1,2,4, 10,11), (1,2,4,10,12),
(1,2,3, 5, 7, 9), (2,3,5,6,8,7,9,2),
(2,3,5,6,8,7,2), (2,3,5,7,2), (2,3,5,7,9,2
(3, 5, 6, 8, 7, 9, 2, 4, 10, 11), (3,5,6,8,7,9,2,4,10,12),
(3,5,8,8,7,2,4,10,11), (3,5,6,8,7,2,4,10,12),
(5,6,5), (6,5,6), (3,5,7,2,3), (3,5,6,8,7,2,3),
(5,6,8,7,9,2,3,5), (5,6,8,7,2,3,5),
(6,8,7,9,2,3,5,6), (6,8,7,2,3,5,6), (10,11,10),
(11,10,11), (8,7,9,2,3,5,6,8) (8,7,2,3,5,6,8)
(7,9,2,3,5,6,8,7), (7,2,2,3,5,6,8,7)
(7,9,2,3,5,7), (7,2,3,5,7), (9,2,3,5,7,9)
(9,2,3,5,6,8,7,9), (11,10,12)

(5) Camino de prueba que satisfaga Nodo Coverage pero que no satisfaga Edge Coverage ¿Viable?

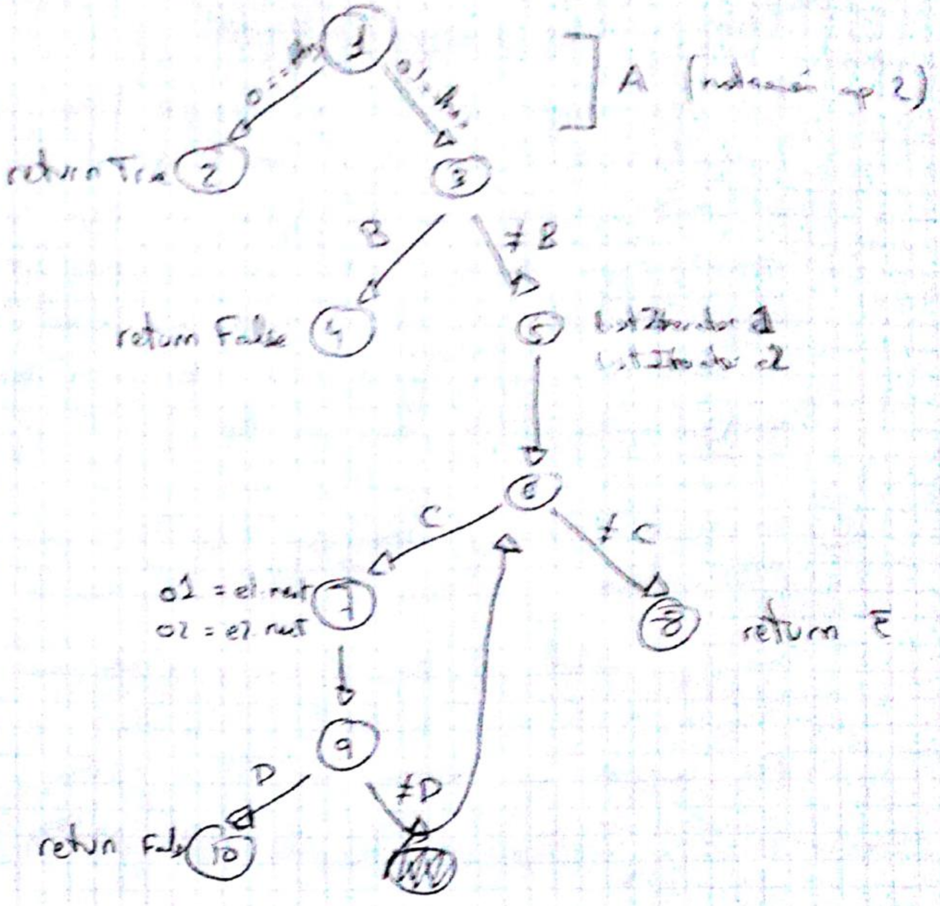
Para que cumpla el NC el camino tiene que ser el siguiente
{1, 2, 3, 5, 6, 8, 7, 9, 2, 4, 10, 11, 10, 12}
Con este camino no cumplimos el Edge Coverage, por tanto, tenemos que buscar un test que demuestre que es viable
Comprobando con el código, vemos que no es viable puesto que si entramos a (2) numPrime = 1, vamos a tener dos iteraciones en (5), por tanto, es inviable.

(6) Camino de prueba que satisfaga Edge Coverage pero no Prime Paths ¿Es viable?

Para que cumpla el EC el camino puede ser el siguiente:
{1, 2, 3, 5, 6, 5, 6, 8, 7, 9, 2, 3, 5, 7, 2, 4, 10, 11, 10, 12}
No cumple con todos los Prime Paths, y como vemos comprobando con el código se puede realizar un test, por tanto, es viable

(7) La mayoría de los Prime Paths no son viables por la forma en que está escrita el método PrintPrimes

Equals()



③ Test para cubrir Nodo Coverage

```
//Test para llegar al 3er return
List <String> list1 = new..
List <String> list 2 = new..

list1.add ("foo")
list2.add ("bar")

assumeFalse (list1. equals (list2))
```

Nodos visitados:
(1, 3, 5, 6, 7, 9, 10)

```
//Test
List <String> list1 = new...
Integer Int2 = new Integer()
list1. add ("foo")
assumeFalse (list1 . equals (Int2))
```

Nodos visitados:
(1, 3, 4)

```
//Test
  List<String> list1 = new...
  list1.add ("foo")
  assumeTrue (list1.equals(list1))
```

Nodos visitados:
(1, 2)

```
//Test
  List<String> list1 = new...
  List<String> list2 = new...
  list1.add ("foo")
  list2.add ("foo")
  assumeTrue (list1.equals(list2))
```

Nodos visitados:
(1, 3, 5, 6, 8)

(4) Edge-pair Coverage

Totales -> [1,3,4], [1,3,5], [3,5,6], [5,6,7], [5,6,8], [6,7,9], [7,9,6], [9,6,7], [9,6,8], [7,9,10]

```
//Test [1, 3, 4]
 List _ list1 = new--
 List _ list2 = new--
 assumeFalse (list1.equals(null))
```

```
//Test
  List<String> list1 = new..
  list<String> list2 = new...
  list1.add("cat")
  list1.add ("dog")
  list2.add ("cat")
  list2.add ("dog")
  assumeTrue ( list1.equals(list2))
```

Probados: (nuevos)
[1,3,5], [3,5,6], [5,6,7], [6,7,9], [9,6,7], [9,6,8], [7,9,6]

```
//Test 1 de Nodo Coverage (anterior)
   ⋮
  list1.add ("foo")
  list2.add ("bar")
  assumeFalse (list1.equals(list2))
```

Probados (nuevos):
[7,9,10]

```
//Test 4 de Nodo Coverage (anterior)
 ⋮
 list1.add ("foo")
 list2.add ("foo")
 assumeTrue (list1.equals(list2))
```

Probados (nuevos):
[5,6,8]

(5) Prime Paths

Caminos totales :

[1, 3, 5, 6, 7, 9, 10], [1, 2],
[1, 3, 4], [illegible]
[1, 3, 5, 6, 8], [6, 7, 9, 6],
[7, 9, 6, 7], [9, 6, 7, 9]

```
//Test [1, 3, 5, 6, 7, 9] 10]
   ⋮
 list2.add ("bat")
 assumeFalse (list1.equals (list2))
```

```
//Test 2 de Node Coverage (anterior)        [1, 3, 4]
list list1 = new...
Integer Int1 = new...
list1.add ("foo")
assumeFalse (list1.equals (Int1))
```

```
//Test 3 de Node Coverage (anterior)        [1, 2]
  ⋮
 list1.add ("foo")
 assumeTrue (list1.equals (list1))
```

```
//Test 2 de Edge-Pair Coverage (anterior)   [6, 7, 9, 6], [7, 9, 6, 7]
  ⋮                                          [9, 6, 7, 9]
 lis1.add ("dog")
 list1.add ("cat")
 list2.add ("dog")
 list2.add ("cat")
 assumeTrue (list1.equals(list2))
```

```
//Test 4 de Node Coverage (anterior)        [1, 3, 5, 6, 8]
 list1.add ("foo")
 list2.add ("foo")
 assumeTrue (list1.equals (list2))
```