```
1   -- UNIT TEST -------------------------S
2
3   {-
4
5   -- 1
6   -- finds triangle
7
8   > o = Node (Node (Node (Leaf 0) 0 (Leaf 0)) 0 (Node (Leaf 0) 0 (Leaf 0))) 0
    (Node (Node (Node (Leaf 0) 0 (Leaf 0)) 0 (Node (Leaf 0) 0 (Leaf 0))) 0 (Node
    (Node (Node (Leaf 0) 0 (Leaf 0)) 1 (Node (Leaf 0) 0 (Leaf 0))) 1 (Node (Node
    (Leaf 0) 0 (Leaf 0)) 1 (Node (Leaf 0) 0 (Leaf 0)))))
9
10  > 3 == maxPath o
11
12  True
13
14
15  -- 2
16  -- finds zero in a sea of negatives
17
18  > z = Node (Node (Node (Leaf (-1)) (-1) (Leaf (-1))) (-1) (Node (Leaf (-1))
    (-1) (Leaf (-1)))) (-1) (Node (Node (Node (Leaf (-1)) (-1) (Leaf (-1))) (-1)
    (Node (Leaf (-1)) (-1) (Leaf (-1)))) (-1) (Node (Node (Node (Leaf (-1)) (-1)
    (Leaf (-1))) (-1) (Node (Leaf (-1)) (-1) (Leaf (-1)))) 0 (Node (Node (Leaf
    (-1)) (-1) (Leaf (-1))) (-1) (Node (Leaf (-1)) (-1) (Leaf (-1))))))
19
20  > 0 == maxPath z
21
22  True
23
24  -}
25
26  -- UNIT TEST -------------------------E
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
```

```haskell
49   -- PROPERTY TEST ---------------------------------------------S
50
51   -- 1
52   treeGenRight :: Int -> Int -> Tree Int
53   treeGenRight n m
54     | n <= 1    = Leaf m
55     | otherwise = Node (Leaf 0) m (treeGenRight (n - 1) m)
56
57
58
59   treeGenLeft :: Int -> Int -> Tree Int
60   treeGenLeft n m
61     | n <= 1    = Leaf m
62     | otherwise = Node (treeGenLeft (n - 1) m) m (Leaf 0)
63
64
65
66   treeGenLR :: Int -> Int -> Tree Int
67   treeGenLR n m
68     | n <= 1    = Leaf m
69     | otherwise = Node t m t
70                   where
71                     t = treeGenLR (n - 1) m
72
73
74
75   qcBoundary :: Int -> Int -> Property
76   qcBoundary n m =
77     m >= 0 && n >= 0 && n < 20 ==>
78       (maxPath (treeGenLeft n m) + maxPath (treeGenRight n m) - m)
79       ==
80       maxPath (treeGenLR n m)
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
```

```haskell
104   --2
105   treeGen :: [Int] -> Tree Int
106   treeGen [m] = Leaf m
107   treeGen (m:ms) = Node t m t
108                   where
109                       t = treeGen ms
110
111
112
113   mirrorTree :: Tree Int -> Tree Int
114   mirrorTree (Leaf a) = Leaf a
115   mirrorTree (Node l a r) = Node (mirrorTree r) a (mirrorTree l)
116
117
118
119   qcMirror :: [Int] -> Property
120   qcMirror xs =
121     not (null xs) && length xs <= 15 ==>
122     ml == ml'
123     where
124       l = treeGen xs
125       ml = maxPath l
126       l' = mirrorTree l
127       ml' = maxPath l'
128
129   -- PROPERTY TEST ------------------------------------------E
```