Γλώσσες Προγραμματισμού ΙΙ
Property Based Testing with QuickCheck

Μιλτιάδης Στούρας
ΑΜ: 03116022

Ακ. Έτος 2019-2020 - Ροή Λ

Producing Arbitrary Trees

Για τον τύπο Tree a:

```
data Tree a = Node a [Tree a]
```

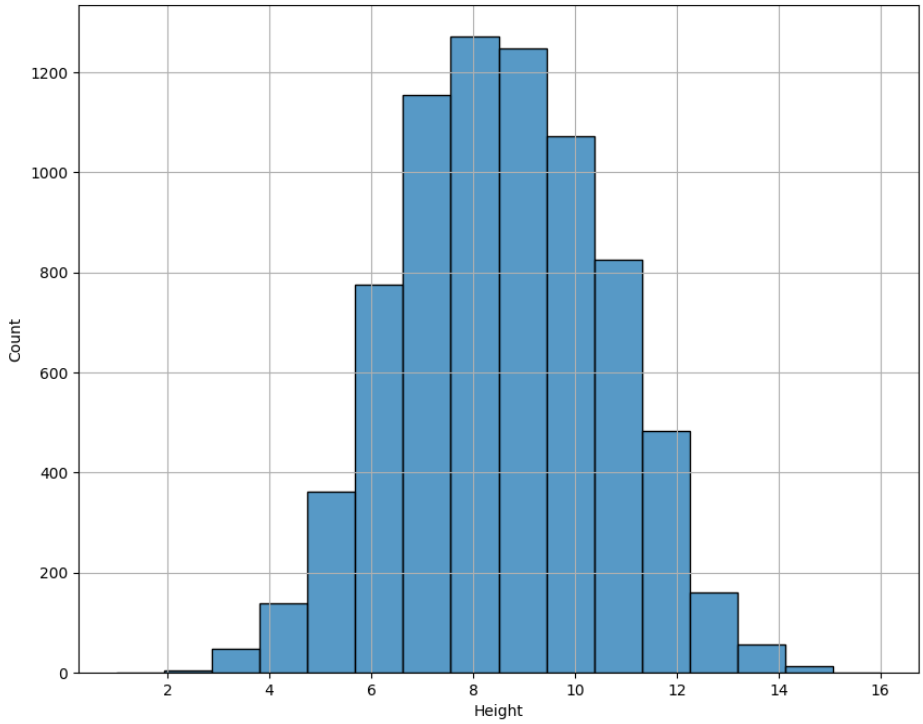
Υλοποιήσαμε την arbitrary ως εξής:

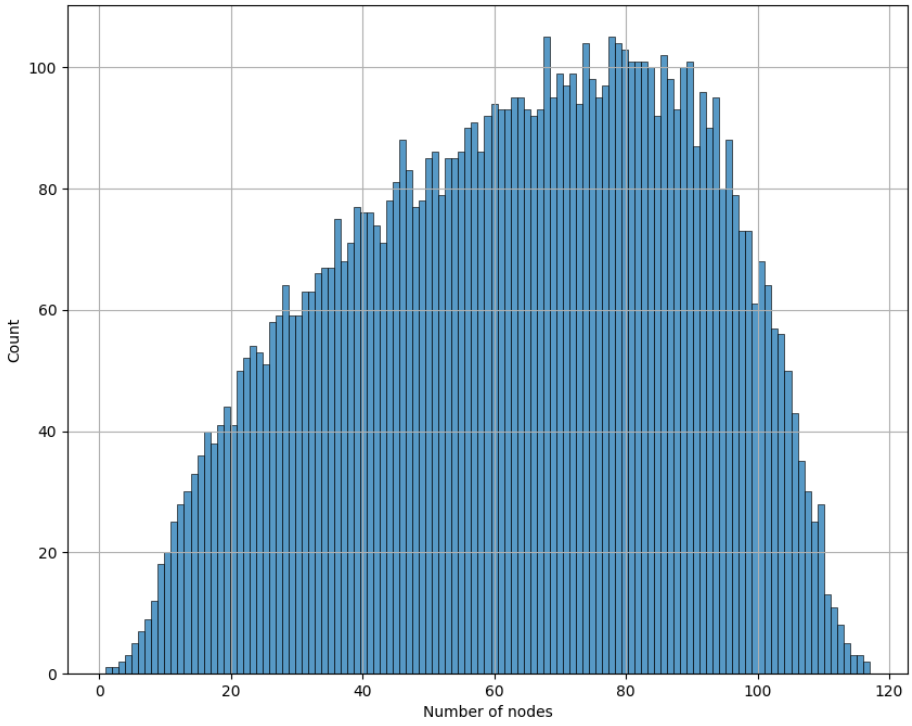
```
instance Arbitrary a => Arbitrary (Tree a) where

arbitrary = sized arbitraryTreeWithSize
  where arbitraryTreeWithSize 0 =
            do x <- arbitrary
               return (Node x [])

        arbitraryTreeWithSize n =
            do x <- arbitrary
               maxNumberOfChildren <- choose(1, n-1)
               childrenSizes <- chooseChildrenSizes maxNumberOfChildren (n-1)
               children <- sequence $ [arbitraryTreeWithSize] <*> childrenSizes
               return (Node x children)
```

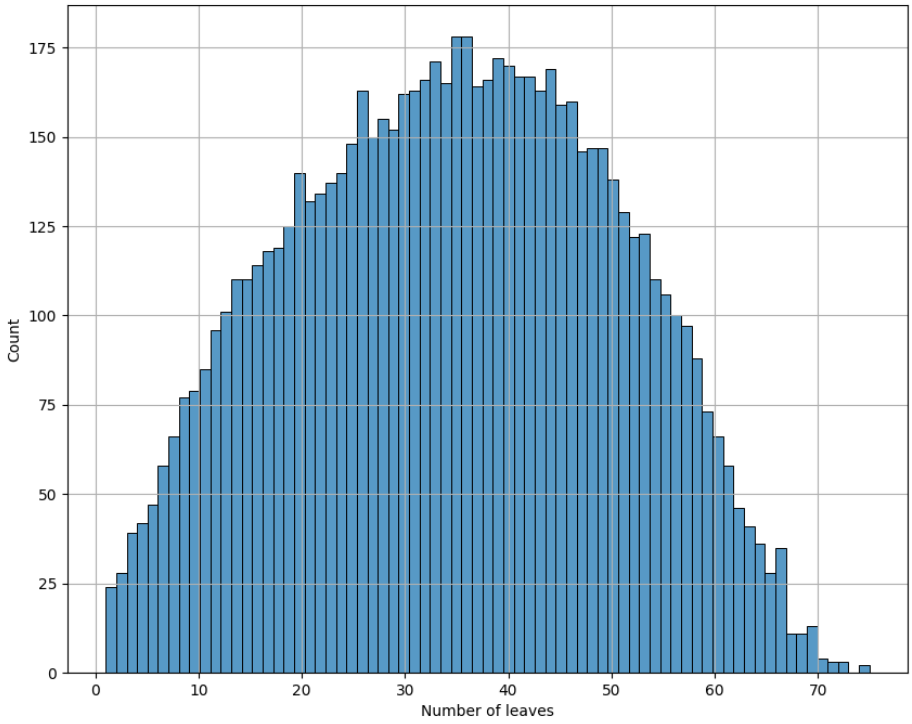
Όπου στην *arbitraryWithSize* n, για να κατασκευάσουμε ένα τυχαίο δέντρο μεγέθους n, διαλέγουμε έναν τυχαίο x για την ρίζα του δέντρου, έναν τυχαίο αριθμό, έστω k, από 1 έως $n-1$ ο οποίος είναι ο αριθμός των παιδιών της ρίζας. Στην συνέχεια, διαλέγουμε k τυχαίους αριθμούς οι οποίοι αθροίζουν στο n και κάθε ένας δηλώνει το μέγεθος του υποδέντρου κάθε παιδιού της ρίζας και κατασκευάζουμε αναδρομικά τα τυχαία υποδέντρα των παιδιών.
Για να επιβεβαιώσουμε ότι τα τυχαία δέντρα που φτιάχνουμε δεν έχουν κάποιο bias ως προς το ύψος και το φάρδος, φτιάξαμε ένα trivial property που ελέγχει αν ένα δέντρο είναι ίσο με τον εαυτό του ενώ ταυτόχρονα κάναμε collect τις παραμέτρους που θέλαμε για 10^4 testcases:

```
prop_tree_trivial t =
collect (height t, size t, leaves t) $ t == t
  where types = (t :: Tree Int)
```

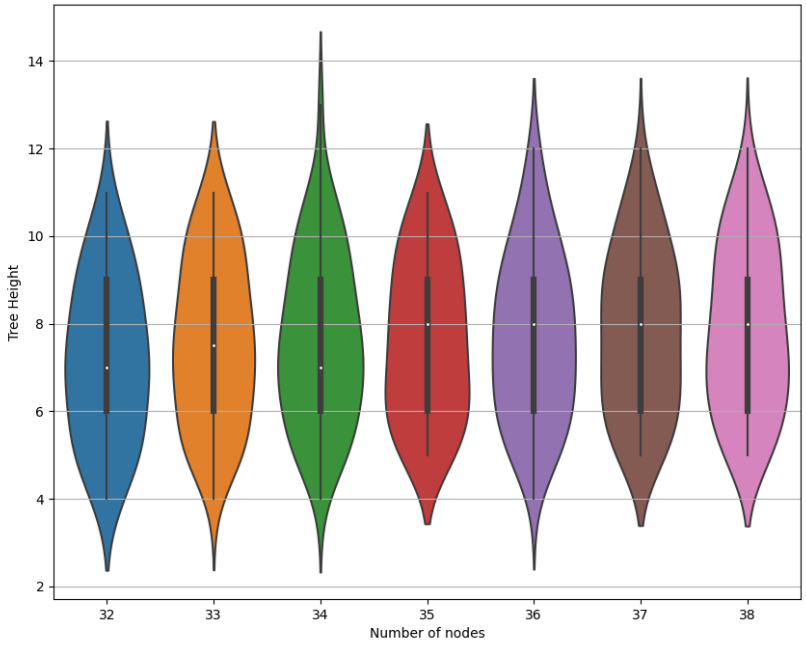
Μετρήσαμε τον αριθμό των κόμβων, το ύψος και τον αριθμό των φύλλων (ως μετρική του πόσο στενό ή πλατύ είναι ένα δέντρο). Τα ιστογράμματα των τιμών είναι τα παρακάτω:

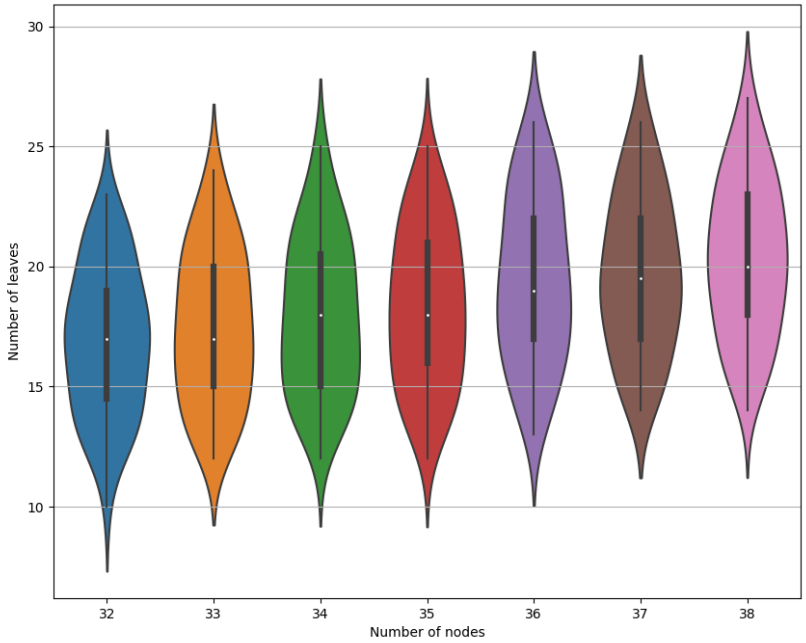

Count
Height
0
200
400
600
800
1000
1200
2
4
6
8
10
12
14
16


Count
Number of nodes
0
20
40
60
80
100
120



Για το πως συσχετίζεται το ύψος και το πλάτος με τον αριθμό των κόμβων, φτιάξαμε violin plots για κάποιες τιμές του αριθμού κόμβων και παρατηρήσαμε πως η κατανομή είναι αρκετά ομοιόμορφη και απλωμένη:





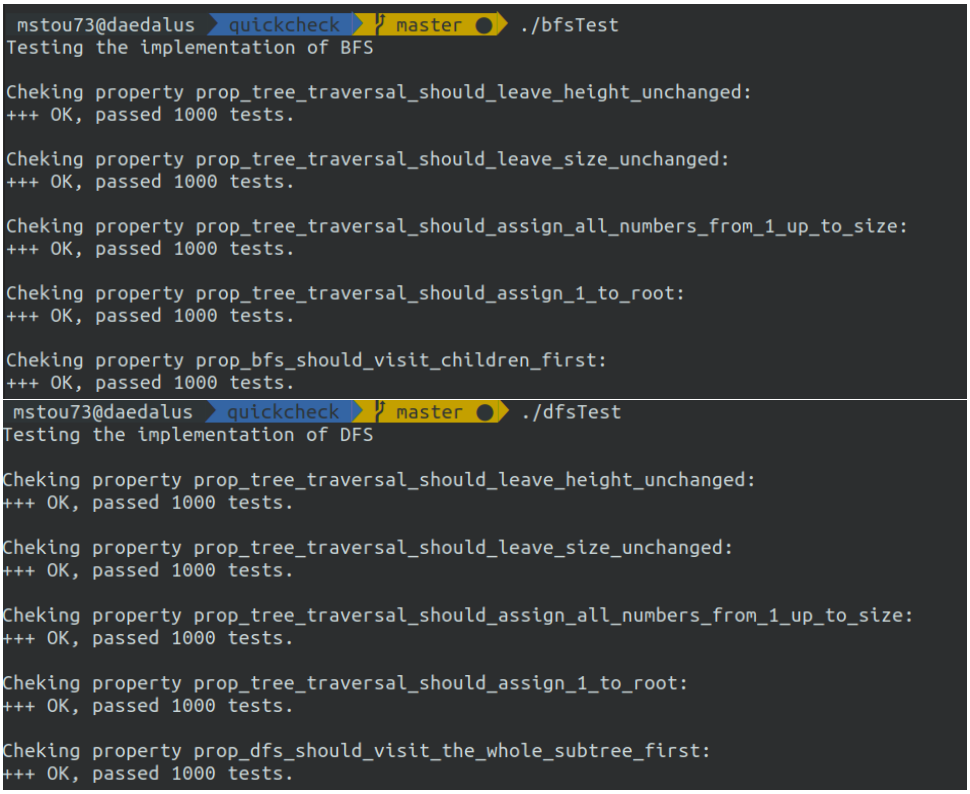
Τέλος, υλοποιήσαμε την συνάρτηση *shrink* ως εξής:

```
shrink (Node _ []) = []
shrink (Node x children) = keepTheRoot ++ keepAChild ++ keepAllChilds
  where
     keepTheRoot = [(Node x [])]
     keepAChild  = children -- quickcheck will recursively call shrink on them
     keepAllChilds = [(Node x shrinkedChildren) | shrinkedChildren <- mapM shrink children]
```

Defining Properties of Tree Traversals

Υλοποιήσαμε τις παρακάτω ιδιότητες για τον έλεγχο των tree traversal συναρτήσεων:

- *prop_tree_traversal_should_leave_height_unchanged*
- *prop_tree_traversal_should_leave_size_unchanged*
- *prop_tree_traversal_should_assign_all_numbers_from_1_up_to_size*
- *prop_tree_traversal_should_assign_1_to_root*
- Για την BFS την ιδιότητα *prop_bfs_should_visit_children_first*, που ελέγχει αν στο πρώτο επίπεδο του δέντρου οι χρόνοι άφιξης των παιδιών είναι διαδοχική και ξεκινούν από το 2.
- Για την DFS, την ιδιότητα *prop_dfs_should_visit_the_whole_subtree_first*, που ελέγχει πως ο χρόνος άφιξης του $i+1$ παιδιού ισούται με τον χρόνο άφιξης του i-οστού παιδού συν το πλήθος κόμβων του υποδέντρου του i-οστού παιδιού.



Testing BFS and DFS Implementations

Χρησιμοποιώντας την QuickCheck, ελέγξαμε ότι οι υλοποιήσεις των BFS και DFS ικανοποιούν τις παραπάνω ιδιότητες. (Προφανώς οι παραπάνω ιδιότητες δεν αρκούν για να αποδείξουν την ορθότητα των bfs, dfs)

Testing the Implementation of *merge*

Υλοποιήσαμε την σωστή *merge* διορθώνοντας το λάθος της εκφώνησης. Παράλληλα, υλοποιήσαμε μια απλή ιδιότητα για να βρούμε ένα λάθος testcase για την υλοποίηση της εκφώνησης. Μια απλή ιδιότητα της merge (+), είναι ότι αν Sum(T) είναι το άθροισμα των κόμβων του δέντρου T, τότε θα πρέπει να ισχύει ότι $\text{Sum}(T_1) + \text{Sum}(T_2) = \text{Sum}(\text{merge } (+)\ T_1\ T_2)$. Αντίστοιχη ιδιότητα ισχύει για κάθε συνάρτηση f που είναι προσεταιριστική και αντιμεταθετική. Ελέγχοντας την παραπάνω ιδιότητα για την πρόσθεση, η quickCheck χρησιμοποιώντας την συνάρτηση shrink που υλοποιήσαμε καταφέρνει να βρει το minimal αντιπαράδειγμα για την υλοποίηση της εκφώνησης: