Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
https://courses.softlab.ntua.gr/pl2/

Γλώσσες Προγραμματισμού II

Αν δεν αναφέρεται διαφορετικά, οι ασκήσεις πρέπει να παραδίδονται στους διδάσκοντες σε ηλεκτρονική μορφή μέσω του συνεργατικού συστήματος ηλεκτρονικής μάθησης `moodle.softlab.ntua.gr`. Η προθεσμία παράδοσης θα τηρείται αυστηρά. Έχετε δικαίωμα να καθυστερήσετε το πολύ μία άσκηση.

Άσκηση 4 Έλεγχος διάσχισης δέντρων

Προθεσμία παράδοσης: 2/2/2020

Έστω στη γλώσσα Haskell ο τύπος `Tree` του δέντρου με οσαδήποτε παιδιά ανά κόμβο, ο ίδιος που εξετάσαμε στα παραδείγματα με τη διάσχιση DFS/BFS, στην παράδοση της 30/10/2019.

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

Έστω επίσης οι παρακάτω συναρτήσεις από την ίδια παράδοση, οι οποίες διασχίζουν ένα δέντρο και αριθμούν τους κόμβους τους αντίστοιχα κατά σειρά DFS και BFS. Τους ορισμούς τους μπορείτε να τους βρείτε στην ιστοσελίδα του μαθήματος αλλά θα έχει μεγαλύτερη πλάκα αν προσπαθήσετε να τους ξαναγράψετε!

```
dfn :: Tree a → Tree (a, Int)
bfn :: Tree a → Tree (a, Int)
```

1. Υλοποιήστε μία γεννήτρια αυθαίρετων δέντρων που να επιτρέπει τον αυτόματο έλεγχο ιδιοτήτων για δέντρα με το εργαλείο `QuickCheck`. Συγκεκριμένα, υλοποιήστε ένα `instance` της μορφής:

   ```
   instance Arbitrary a ⇒ Arbitrary (Tree a)
   ```

   Επειδή ο τύπος `Tree` είναι αναδρομικός, φροντίστε να κάνετε χρήση του μέτρου στη γεννήτρια (`sized` και πιθανώς `resize`), έτσι ώστε η γεννήτρια να τερματίζει. Επίσης, φροντίστε ώστε τα δέντρα που γεννάτε να μην είναι πάντοτε είτε πολύ "πλατειά" είτε πολύ "στενά".

2. Υλοποιήσετε κατάλληλα τη συνάρτηση `shrink` στο `instance Arbitrary (Tree a)`, ώστε να βρίσκει "μικρά" δέντρα ως αντιπαραδείγματα ιδιοτήτων που δεν ισχύουν.

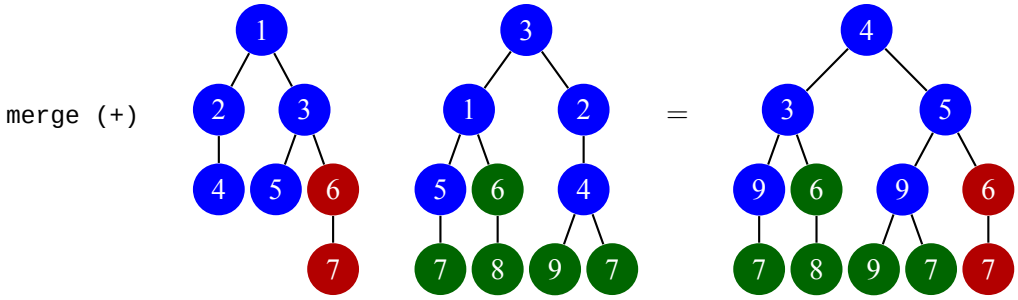
3. Ορίστε ένα σύνολο ιδιοτήτων που εγγυώνται τη σωστή λειτουργία των συναρτήσεων `dfn` και `bfn`. Εκφράστε αυτές ως συναρτήσεις σε Haskell, που να μπορούν να επαληθευθούν με το QuickCheck.

   Παραδείγματα τέτοιων ιδιοτήτων (αν και ίσως μπορείτε να βρείτε καλύτερες):

   - Και οι δύο συναρτήσεις θα πρέπει να διατηρούν το μέγεθος (δηλαδή το πλήθος των κόμβων) του δέντρου.
   - Και για τις δύο συναρτήσεις θα πρέπει η ρίζα του δέντρου να παίρνει τον αριθμό 1.

4. Υλοποιήστε ένα πρόγραμμα σε Haskell το οποίο, χρησιμοποιώντας το QuickCheck για αυθαίρετα δέντρα, να επαληθεύει τις παραπάνω ιδιότητες για τις συναρτήσεις `dfn` και `bfn`.

5. Ζητείται μία συνάρτηση `merge` η οποία να "συγχωνεύει" δύο δέντρα συνδυάζοντας τις τιμές των αντίστοιχων κόμβων με μία συνάρτηση `f`. Συγκεκριμένα, η συνάρτηση αυτή θα πρέπει να δέχεται ως παράμετρο τη συνάρτηση `f` και δύο δέντρα, `t1` και `t2`. Αν και τα δύο δέντρα έχουν έναν κόμβο *στην ίδια θέση*, με τιμές `x1` και `x2` αντίστοιχα, τότε το αποτέλεσμα θα πρέπει επίσης να έχει έναν κόμβο σε αυτή τη θέση με τιμή `f x1 x2`. Αντίθετα, αν μόνο ένα από τα δέντρα έχει κόμβο σε κάποια θέση, το αποτέλεσμα θα πρέπει να έχει κόμβο σε αυτή τη θέση με την ίδια τιμή. Για παράδειγμα:



όπου για διευκόλυνσή σας είναι σημειωμένοι με μπλε χρώμα οι κόμβοι που είναι στην ίδια θέση και στα δύο δέντρα, με κόκκινο οι κόμβοι που υπάρχουν μόνο στο `t1` και με πράσινο οι κόμβοι που υπάρχουν μόνο στο `t2`.

6. Κάποιος φίλος σάς δίνει την παρακάτω συνάρτηση `wrong`, η οποία κατά τον ισχυρισμό του επιτυγχάνει το ζητούμενο στο προηγούμενο ερώτημα και είναι καλύτερη από τη δική σας!

```
wrong :: (a → a → a) → Tree a → Tree a → Tree a
wrong f (Node x tsx) (Node y tsy) = Node (f x y) $ zipWith (wrong f) tsx tsy
```

Υλοποιήστε ένα πρόγραμμα σε Haskell το οποίο, χρησιμοποιώντας το QuickCheck για αυθαίρετα δέντρα, να επιδεικνύει ότι:

- η `wrong` είναι εσφαλμένη — για να το κάνετε αυτό, ορίστε μία κατάλληλη ιδιότητα που μία σωστή υλοποίηση της `merge` πρέπει να επαληθεύει,
- η υλοποίησή σας της συνάρτησης `shrink` παράγει όντως μικρά δέντρα ως αντιπαραδείγματα ιδιοτήτων, και
- η δική σας `merge`, που ορίσατε στο προηγούμενο υποερώτημα, επαληθεύει την ιδιότητα που χρησιμοποιήσατε για να βάλετε το φίλο σας στη θέση του.