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.

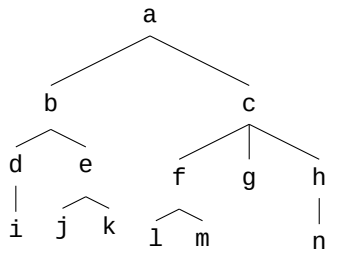
Άσκηση 2 Περιγράμματα και δέντρα Fibonacci

Προθεσμία παράδοσης: 29/11/2020

Έστω η δομή δεδομένων `Tree a` που αναπαριστά δέντρα, οι εσωτερικοί κόμβοι των οποίων φέρουν πληροφορία τύπου `a`.

```
data Tree a = Node a [Tree a]
  deriving (Eq, Show, Read)
```

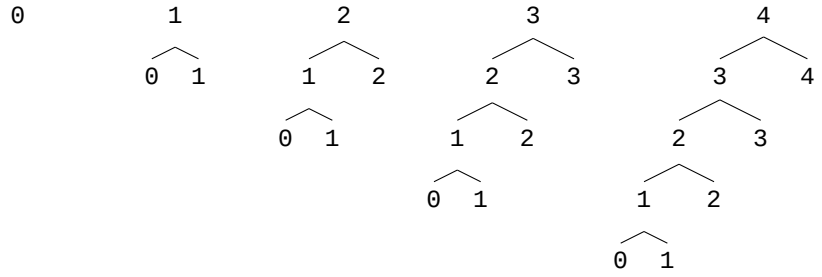
Για παράδειγμα, το παρακάτω δέντρο παριστάνεται ως εξής:




```
Node a [ Node b [ Node d [Node i []]
                , Node e [Node j [], Node k []]
                ]
       , Node c [ Node f [Node l [], Node m []]
                , Node g []
                , Node h [Node n []]
                ]
]
```

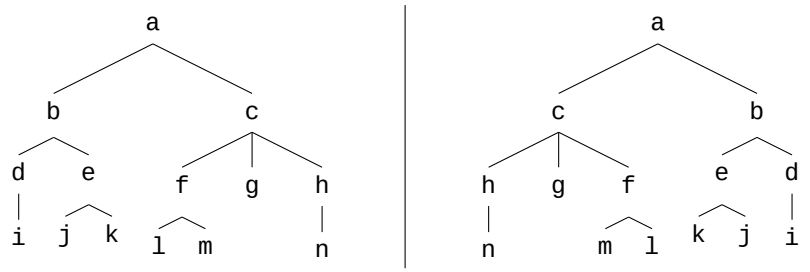
(a) Να γραφεί μία συνάρτηση `leftish :: Int → Tree a` τέτοια ώστε οι παρακάτω κλήσεις να κατασκευάζουν τα ακόλουθα δέντρα:

`leftish 0` `leftish 1` `leftish 2` `leftish 3` `leftish 4` *κ.ο.κ.*



(b) Να γραφεί μία συνάρτηση `mirror :: Tree a → Tree a` που κατασκευάζει το *κατοπτρικό* ενός δέντρου (δηλ. αυτό που προκύπτει αν κοιτάξουμε το δέντρο σε έναν καθρέφτη, στο οποίο σε κάθε κόμβο η σειρά των παιδιών του έχει αντιστραφεί).

Για το δέντρο του αρχικού παραδείγματος (αριστερά), το αποτέλεσμα της `mirror` θα πρέπει να είναι το εξής (δεξιά):



Ονομάζουμε *φύλλα* τους κόμβους της μορφής `Node x []`. Το *περίγραμμα* (fringe) ενός δέντρου ορίζεται ως η λίστα των φύλλων του δέντρου, όπως προκύπτουν από μία ενδοδιατεταγμένη (infix) διάσχιση. Για παράδειγμα, το περίγραμμα του δέντρου του αρχικού παραδείγματος (παραπάνω αριστερά) είναι η λίστα: `[i, j, k, l, m, g, n]`.

(c) Να γραφεί μία συνάρτηση `fringe_naive :: Tree a → [a]` που υλοποιεί την εύρεση του περιγράμματος ενός δέντρου με τον πιο προφανή αναδρομικό τρόπο.

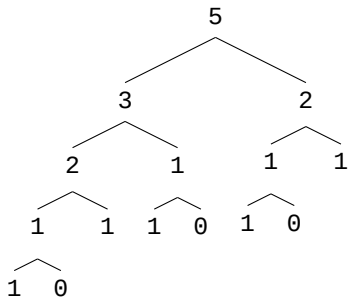
Ποια είναι η πολυπλοκότητά της (ως συνάρτηση του πλήθους n των φύλλων του δέντρου); Διαπιστώστε το εφαρμόζοντάς την σε δέντρα `leftish n` για αυξανόμενες τιμές του `n`. Ίσως σας φανεί χρήσιμη η βιβλιοθήκη System.TimeIt.

(d) Να γραφεί μία βελτιωμένη συνάρτηση `fringe :: Tree a → [a]` που υπολογίζει το περίγραμμα με πιο αποδοτικό τρόπο. Ποια είναι η πολυπλοκότητά της;

(e) Να γραφεί μία συνάρτηση `same_fringe :: Tree a → Tree a → Bool` που αποφασίζει αν τα περιγράμματα δύο δέντρων είναι ίσα. Ποια είναι η πολυπλοκότητά της; Πώς συμπεριφέρεται στην περίπτωση που τα περιγράμματα διαφέρουν;

(f) Αν η Haskell ήταν μία *πρόθυμη* (eager) γλώσσα, σε τι θα άλλαζε η απάντησή σας στο προηγούμενο ερώτημα; Πόσο εύκολο είναι να γράψετε μία εξίσου αποδοτική `same_fringe` π.χ. στην ML; Σκιαγραφήστε την υλοποίησή της.

Για κάθε φυσικό αριθμό n, ορίζουμε το δέντρο Fibonacci T_n ως εξής. Το T_n έχει στη ρίζα του τον n-οστό αριθμό Fibonacci F_n. Τα δέντρα T_0 και T_1 είναι φύλλα. Για κάθε n, το δέντρο T_{n+2} έχει ως αριστερό του παιδί το T_{n+1} και ως δεξί του παιδί το T_n. Για παράδειγμα, το δέντρο T_5 φαίνεται στο παρακάτω σχήμα:



(g) Να γραφεί μία συνάρτηση `fibtree_naive :: Int → Tree Int` που κατασκευάζει το T_n με τον πιο προφανή αναδρομικό τρόπο. Πόσο χώρο καταλαμβάνει στη μνήμη το δέντρο T_n;

(h) Να γραφεί μία βελτιωμένη συνάρτηση `fibtree :: Int → Tree Int` που κατασκευάζει το δέντρο T_n κατά τέτοιο τρόπο ώστε να καταλαμβάνει στη μνήμη χώρο $O(n)$.

(i) Πόσο χώρο καταλαμβάνει στη μνήμη το δέντρο που προκύπτει όταν αποτιμηθεί πλήρως η έκφραση `mirror (fibtree n)`;

Γράψτε ένα πρόγραμμα που να συνδυάζει όλα τα παραπάνω, συμπληρώνοντας αυτό το υπόδειγμα με τις απαντήσεις των παραπάνω ερωτημάτων, είτε σε μορφή κώδικα Haskell είτε σε σχόλια.