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

Γλώσσες Προγραμματισμού II

Οι ασκήσεις πρέπει να παραδοθούν 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 1 Περιγράμματα και δέντρα Fibonacci σε Haskell

Προθεσμία παράδοσης: 19/11/2008

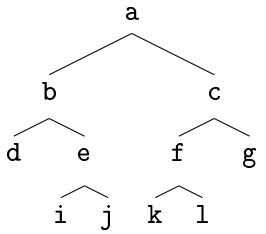
⇒ *Να παραδοθεί ένα αρχείο* `fringe.hs` *που να περιέχει τις απαντήσεις στα παρακάτω ερωτήματα* **(a)**–**(k)**, *είτε σε μορφή κώδικα Haskell είτε σε σχόλια. Το αρχείο πρέπει να μεταγλωττίζεται με* `ghc -o fringe fringe.hs` *και να εκτελείται σύμφωνα με τα ζητούμενα στο ερώτημα* **(j)**.

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

```
data Tree a = E | T a (Tree a) (Tree a)
  deriving (Show, Read)
```

Για παράδειγμα, το δέντρο του Σχήματος 1 παριστάνεται ως εξής:

```
T a (T b (T d E E) (T e (T i E E) (T j E E)))
    (T c (T f (T k E E) (T l E E)) (T g E E))
```



Σχήμα 1: Παράδειγμα.

(a) Να γραφεί μία συνάρτηση `mirror :: Tree a -> Tree a` που κατασκευάζει το *κατοπτρικό* ενός δέντρου (δηλ. αυτό στο οποίο όλα τα αριστερά παιδιά έχουν αντιμετατεθεί με τα δεξιά).

Ονομάζουμε *φύλλα* τους κόμβους της μορφής `T x E E`. 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 Σχήματος 1 είναι η λίστα `[d, i, j, k, l, g]`.

(b) 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);

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

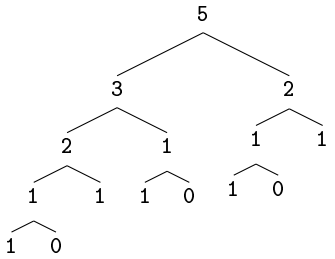
(d) Να γραφεί μία συνάρτηση `same_fringe :: Tree a -> Tree a -> Bool` που αποφασίζει αν τα περιγράμματα δύο δέντρων είναι ίσα.

(e) Ποια είναι η πολυπλοκότητα της `same_fringe`; 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, όπως αυτός ορίζεται στην ιστοσελίδα `http://en.wikipedia.org/wiki/Fibonacci_number`. 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 Σχήμα 2.



Σχήμα 2: Το δέντρο T_5.

(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)`;

(j) Να γραφεί ένα πρόγραμμα που να συνδυάζει όλα τα παραπάνω. Το πρόγραμμα πρέπει να ζητά από το χρήστη ένα φυσικό αριθμό n και να εκτυπώνει δύο γραμμές. Η πρώτη γραμμή πρέπει να περιέχει "`yes`" αν το περίγραμμα του T_n είναι ίσο με το περίγραμμα του κατοπτρικού του, διαφορετικά "`no`". Η δεύτερη γραμμή πρέπει να περιέχει "`yes`" αν το περίγραμμα του T_n είναι ίσο με το περίγραμμα του T_{n+1}, διαφορετικά "`no`".

(k) Δοκιμάστε το πρόγραμμά σας με σταδιακά αυξανόμενες τιμές του n. Γιατί όσο αυξάνει το n, η πρώτη γραμμή τυπώνεται αμέσως ενώ η δεύτερη καθυστερεί όλο και περισσότερο; Συμφωνεί αυτό με τις απαντήσεις σας στα προηγούμενα ερωτήματα;

Άσκηση 2 Αγνές συναρτησιακές δομές δεδομένων

Προθεσμία παράδοσης: 19/11/2008

Στο μάθημα είδαμε κάποιους τρόπους με τους οποίους δομές δεδομένων μπορούν να γραφούν σε μια γλώσσα αγνού συναρτησιακού προγραμματισμού. Ένα μεγάλο μέρος της διάλεξης βασίστηκε στο paper του Chris Okasaki για τα red-black trees σε συναρτησιακό περιβάλλον το οποίο είναι διαθέσιμο στο παρακάτω URL:

`http://www.eecs.usma.edu/webs/people/okasaki/jfp99.ps`

Όπως μπορείτε να παρατηρήσετε και μόνοι σας (Section 5), ο Okasaki χρησιμοποιεί κάπως διαφορετικούς μετασχηματισμούς από αυτούς που κάποιος συνήθως βρίσκει στα περισσότερα βιβλία αλγορίθμων και δομών δεδομένων για τα red-black trees.

Σχετικά πρόσφατα (Απρίλιος – Σεπτέμβριος 2008), ο Sedgewick πρότεινε μια παραλλαγή των red-black trees, τα λεγόμενα left-leaning red-black[1] trees των οποίων ο ορισμός, επιπλέον πληροφορίες, κώδικας σε Java, slides και video από ομιλίες είναι διαθέσιμα στα παρακάτω URL:

`http://www.cs.princeton.edu/~rs/talks/LLRB`
`http://www.cs.princeton.edu/~rs/talks/LLRB/LLRB.pdf`
`http://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf`

[1] Με κλίση προς τα αριστερά, κόκκινα, μαύρα, ... κάτι μου θυμίζει, κάτι μου θυμίζει :-)

Το ζητούμενο της άσκησης είναι να γράψετε σε Haskell τις συναρτήσεις που υλοποιούν τις λειτουργίες της εισαγωγής (`insert`) και της διαγραφής (`delete`) από ένα left-leaning red-black tree και να γράψετε μια σύντομη αναφορά με τα θέματα ή τυχόν προβλήματα που έπρεπε να επιλύσετε για να έχετε μια αγνά συναρτησιακή υλοποίηση της συγκεκριμένης δομής.

Άσκηση 3 Υλοποίηση γλωσσών συναρτησιακού προγραμματισμού

Προθεσμία παράδοσης: 24/11/2008

Στο παρακάτω URL υπάρχει μια πρόσφατη δημοσίευση που καταγράφει την ιστορία της ανάπτυξης της γλώσσας Haskell.

`http://research.microsoft.com/~simonpj/papers/history-of-haskell/history.pdf`

Διαβάστε το άρθρο, επικεντρώνοντας την προσοχή σας στα μέρη II και III, και αναφέρετε συνοπτικά (σε 2–4 σελίδες) τα σημεία που σας έκαναν μεγαλύτερη εντύπωση και γιατί.