Συναρτησιακός Προγραμματισμός 2008
Τελική Εξέταση

Η εξέταση διαρκεί 2 ώρες και αντιστοιχεί στο 55% του βαθμού του μαθήματος.
Συνολικό Άθροισμα Βαθμών: 550.
Επιτρέπονται: Σημειώσεις σχετικά με το μάθημα.
Όλα τα προγράμματα να γραφούν σε Haskell.
Καλή Επιτυχία!

1. Γράψτε μία συνάρτηση με όνομα `encode` που να παίρνει ως όρισμα μία λίστα `l`, και την κωδικοποιεί ως εξής:

   - κάθε μεγίστου-μήκους κομμάτι της `l` που περιέχει `n` εμφανίσεις του ιδίου αντικειμένου `x` κωδικοποιείται από ένα ζεύγος `(x,n)`.
   - ολόκληρη η `l` κωδικοποιείται ως λίστα τέτοιων ζευγών.

   Για παράδειγμα, η έκφραση
   `encode[2,2,3,3,3,3,2,2,2,6,3,3]`
   θα πρέπει να αποτιμηθεί σε
   `[(2,2),(3,4),(2,3),(6,1),(3,2)]`
   Ποιος είναι ο πιο γενικός πολυμορφικός τύπος που περιγράφει τη συνάρτησή σας;

   §

   ```
   encode [] = []
   encode (x:xs) = encode_streak x 1 xs

   encode_streak str n [] = [(str,n)]
   encode_streak str n (x:xs)
     = if str==x then encode_streak str (n+1) xs
       else (str,n):encode_streak x 1 xs
   ```

   Ο πιο γενικός τύπος είναι:

   ```
   (Eq a, Num b) => [a] -> [(a,b)]
   ```

   Αν και εμείς θέλουμε να το χρησιμοποιήσουμε μόνο για ακεραίους, οπότε θα μπορούσαμε να το περιορίσουμε και σε

```
(Eq a, Integral b) => [a] -> [(a,b)]
```

2. Το πρόβλημα των βασιλισσών περιγράφεται ως εξής: έστω μία σκακιέρα $n \times n$ όπου n ακέραιος με $n > 3$. Ζητείται να τοποθετηθούν n βασίλισσες σκακιού στη σκακιέρα, έτσι ώστε καμία να μην απειλεί την άλλη. Μία βασίλισσα απειλεί μία άλλη αν οι δύο βασίλισσες βρίσκονται στην ίδια σειρά, στήλη ή διαγώνιο. Χρησιμοποιώντας όποια αναπαράσταση δεδομένων θέλετε, γράψτε μία συνάρτηση με όνομα `queens` η οποία να παίρνει έναν ακέραιο `n` και να επιστρέφει μία λίστα με *όλες* τις λύσεις του προβλήματος σε σκακιέρα $n \times n$.

   Υπόδειξη: Χρησιμοποιήστε εκφράσεις διαχωρισμού.

   § Ένα τετράγωνο μιας σκακιέρας $n \times n$, αναπαρίσταται από ένα ζεύγος ακεραίων `(i,j)` από το 1 στο n. Μία λύση αναπαρίσταται από μία λίστα τετραγώνων.

   Η συνάρτηση `threat` αποφασίζει αν δύο βασίλισσες αλληλοαπειλούνται:

```
threat (x,y) (x',y') =
  x==x' || y==y' ||  x+y == x'+y'  ||  x-y == x'-y'
```

   Η συνάρτηση `accept` παίρνει ένα τετράγωνο `p` και μία λίστα τετραγώνων `l` στα οποία έχουν τοποθετηθεί βασίλισσες και αποφασίζει αν μια βασίλισσα επιτρέπεται να τοποθετηθεί στο `p`:

```
accept p l = not(or (map (threat p) l))
```

   Η συνάρτηση `queens` λύνει το πρόβλημα. Η συνάρτηση ξεκινάει να γεμίζει τη σκακιέρα από τα δεξιά προς τα αριστερά: η `queensaux n rest` καλείται όταν επιλύεται το πρόβλημα στην έκδοση $n \times n$ και μένουν να τοποθετηθούν `rest` βασίλισσες.

```
queens n = queensaux n n

queensaux n 0 = [[]]
queensaux n rest =
   [(rest,y):qs |  y<-[1..n], qs<-queensaux n (rest-1)
                , accept (rest,y) qs
   ]
```

3. Θέλουμε να αναπαραστήσουμε τους *απλούς μη κατευθυνόμενους γράφους χωρίς βάρη*. Δηλαδή, ένας γράφος αντιπροσωπεύεται από ένα σύνολο από *κόμβους* V μαζί με ένα σύνολο *ακμές* $E \subseteq V^2$. Για ένα γράφο με n κόμβους, όπου $n \in \mathbb{N}$, θα θεωρούμε ότι οι κόμβοι του είναι όλοι οι φυσικοί από το 0 έως το $n - 1$.

   Χρησιμοποιώντας όποια αναπαράσταση δεδομένων θέλετε, υλοποιήστε ένα ΑΤΔ `Graph` που να αναπαριστά τέτοιους γράφους, μαζί με συναρτήσεις τέτοιες ώστε:

- Να είναι δυνατόν να κατασκευαστούν *όλοι* οι γράφοι (συμπεριλαμβανομένου και του μηδενικού γράφου) και *μόνον αυτοί*.
- Δωθέντος ενός γράφου, να μπορούμε να πάρουμε κάθε πληροφορία που θέλουμε από αυτόν, δηλ. ποιο είναι το μέγεθός του και αν μία ακμή είναι ή δεν είναι στο γράφο.

Υπόδειξη: Προσέξτε το γεγονός ότι ο γράφος είναι *μη κατευθυνόμενος*.

§ Αναπαριστούμε ένα γράφο με ένα ζεύγος `(n,l)`. Ο `n` είναι ένας φυσικός αριθμός που αναπαριστά τον αριθμό κόμβων του γράφου. Η `l` είναι μια λίστα ζευγών κόμβων του γράφου που αναπαριστούν τις ακμές που υπάρχουν στο γράφο.

```
module Graph
  (Graph, emptygraph, addedge, size, edge_exists)
where

data Graph = G (Int, [(Int,Int)])

emptygraph :: Int -> Maybe Graph
emptygraph n =
  if n>=0 then Just (G (n, [])) else Nothing

addedge :: Graph -> (Int,Int) -> Maybe Graph
addedge (G (n,es)) (v1,v2) =
  if 0<=v1 && v1<n && 0<=v2 && v2<n
  then Just (G (n, (v1,v2):es))
  else Nothing

size :: Graph -> Int
size (G (n,_)) = n

edge_exists :: Graph -> (Int,Int) -> Bool
edge_exists (G (n,es)) (v1,v2) =
  True `elem` (map (eqs (v1,v2)) es)

eqs :: (Int,Int)->(Int,Int)->Bool
eqs (x,y) (z,w) = (x==z && y==w) || (x==w && y==z)
```

Οι συναρτήσεις `emptygraph` και `addedge` δημιουργούν όλους τους γράφους. Οι συναρτήσεις επιστρέφουν `Maybe Graph` για να αποφευχθεί κατασκευή μη έγκυρων γράφων.

Οι συναρτήσεις `size` και `edge_exists` επιστρέφουν τον αριθμό κόμβων και αν μία ακμή βρίσκεται στον γράφο αντίστοιχα. Προσοχή στον έλεγχο `eqs` που λαμβάνει υπόψην του ότι ο γράφος είναι μη κατευθυνόμενος.

4. Υλοποιήστε μία μονάδα `AppCount` η οποία να μετράει τον αριθμό εφαρμογών συναρτήσης στην αποτίμηση μίας έκφρασης. Εφαρμογές της `return` θα πρέπει να συμπεριλαμβάνονται σε αυτόν τον αριθμό.

   Γράψτε μία συνάρτηση Haskell με όνομα `appcount` και τύπο
   `AppCount a -> Int`
   που να επιστρέφει αυτόν τον αριθμό.

   Για παράδειγμα, έστω οι παρακάτω ορισμοί

```
fact :: Int -> AppCount Int
fact 0 = do return 1
fact n = do x <- fact(n-1)
            return (x*n)
sc :: Int->AppCount Int
sc n = do return (n+1)
```

   Η αποτίμηση της έκφρασης
   `appcount (do x<-fact 3 ; sc x ; return x)`
   πρέπει να επιστρέφει 6.

   §

```
data AppCount a = AC (a,Int)
instance Monad AppCount where
  return x = AC (x,1)
  AC (x,c) >>= k = AC (x',c+c') where AC (x',c') = k x

appcount (AC (_,c)) = c
```