TPFI 2023/24
Hw 2: Efficienza, Tipi & Prove

docente: I. Salvo, Sapienza Università di Roma
assegnato: 28 marzo 2024, consegna 16 aprile 2024

1. mergeSort "iterativo"

1. Definire una funzione Haskell che segue la seguente idea *bottom-up* per implementare l'algoritmo *mergeSort*:

> *Data una lista xs, creare una lista di liste lunghe 1, ciascuna contenente un elemento di xs, poi fondere a due a due le liste ordinate (eventualmente lasciando inalterata l'ultima lista quando il numero delle liste è dispari), finchè non rimane un'unica lista ordinata.*

Ad esempio, cominciando con `[5,3,4,2,1]` la funzione calcola le seguenti liste: `[[5],[3],[4],[2],[1]]`, poi `[[3,5],[2,4],[1]]`, `[[2,3,4,5],[1]]` e infine `[[1,2,3,4,5]]` da cui viene estratto il risultato finale `[1,2,3,4,5]`.

2. Accelerare la prima fase di questo algoritmo per trarre vantaggio da input "favorevoli". La migliorìa dovrebbe assicurare un comportamento lineare in casi particolarmente fortunati.

2. Alberi & funzionali sugli alberi

Considerare le seguenti definizione di alberi binari:

```
data BinTree a = Node a (Bintree a) (BinTree a) | Empty
data BinTree' a = Node' (BinTree' a) (BinTree' a) | Leaf a
```

1. Scrivere i funzionali *mapBT*, *mapBT'*, *foldrBT*, *foldrBT'*, *foldlBT*, e *foldlBT'* che generalizzano agli alberi `BinTree` e `BinTree'` gli analoghi funzionali *map*, *foldr* e *foldl* sulle liste. Riflettete accuratamente sui tipi che devono avere e su quali siano, di fatto, i principi di ricorsione sugli alberi binari.

2. Scrivere poi le seguenti funzioni usando *foldrBT* e *foldrBT'* (cercare di ottenere algoritmi lineari nel numero dei nodi):

(a) numero dei nodi di un albero binario;

(b) altezza dell'albero (= lunghezza in numero di archi del più lungo cammino radice-foglia)

(c) massimo indice di sbilanciamento (= massima differenza tra altezza sotto-albero destro/sinistro)

FACOLTATIVO: Gli alberi a branching illimitato si possono facilmente definire in Haskell come segue: `data Tree a = R a [Tree a]`. Come ai punti precedenti, scrivendo i funzionali *mapT*, *foldrT* e *foldlT*.

3. Nodi Equilibrati

Un nodo u di un albero (considerare a piacere i `BinTree` oppure i `Tree` dell'esercizio precedente) con valori numerici in tutti i nodi è detto *equilibrato* se la somma delle chiavi nel cammino dalla radice a u (esclusa la chiave in u) è esattamente uguale alla somma delle chiavi del sotto-albero radicato in u (compresa la chiave in u).

Scrivere una funzione *nodiEquilibrati :: Num a ⇒BinTree a → [a]* che preso in input un albero, restituisce la lista (eventualmente vuota) contenente tutti i valori nei nodi equilibrati.

Valutare la complessità della funzione.

4. Alberi Binari di Ricerca

Scrivere una funzione Haskell *listToABR :: Ord a ⇒ [a] → BinTree a* che sistema i valori di una lista in un albero binario di ricerca.

Determinare la complessità della funzione e chiedersi se si tratta di una complessità ottima rispetto al problema.

5. Derivazioni di programmi

La funzione *scanr :: (a → b) → b → [a] → [b]* può essere facilmente definita componendo *map*, *foldr* e *tails* (chiamata *suffissi* nell'Homework precedente):

```
scanr f e = map (foldr f e) .  tails
```

Usare la definizione sopra come specifica per derivare una definizione efficiente (cioè lineare nella lunghezza della lista) facendo manipolazioni algebriche, in analogia con quanto visto per *scanl* (slide lezione **9**).