TPFI 2022/23
Hw 1: Programmazione su Liste

docente: I. Salvo – Sapienza Università di Roma

assegnato: 13 marzo 2023, consegna 26 marzo 2023

Nota: Consegnare un unico file testo con estensione `.hs` (preferibilmente di nome `HW1-NomeCognome.hs`). Scrivere la soluzione di eventuali punti 'teorici' (esempio, esercizio **3.4**) semplicemente come commento nel codice Haskell.

Sono apprezzate (ove possibile) soluzioni "composizionali" che fanno uso di funzion(al)i predefiniti o visti a lezione.

Esercizio 1 Il numero di *inversioni* di una lista $xs = [x_0, x_1, \ldots, x_{n-1}]$ è il numero delle coppie di indici $i < j \in [0, n)$ tali che $x_i > x_j$.

1. scrivere una funzione `countInversions :: (Ord a) => [a] -> Int` che conta il numero di inversioni in una lista;

2. scrivere una funzione `inversions :: (Ord a) => [a] -> [(a, a)]` che restituisce le inversioni in una lista, cioè le coppie di elementi di una lista che non stanno nel giusto ordine;

3. progettare un algoritmo *divide-et-impera* che conta il numero delle inversioni in tempo pessimo $\Theta(n \log n)$, modificando opportunamente l'algoritmo *mergeSort*.

Esercizio 2

1. Scrivere una funzione `segmentiSommaS :: (Num a) => [a] -> a -> [[a]]` che data una lista numerica e un valore s restituisce tutti i segmenti (cioè sottoliste di elementi *consecutivi*) di somma s.

2. Scrivere una funzione `sottoListeSommaS :: (Num a) => [a] -> a -> [[a]]` che data una lista numerica e un valore s restituisce tutte le sottoliste (anche di *elementi non consecutivi*) di somma s.

Esercizio 3

1. Definire il funzionale `zipWith f xs ys` senza decomporre le liste `xs` e `ys`, ma usando un espressione che contenga `applyL` (aka `zapp`), `f` ed eventualmente `xs` e `ys`.

2. Definire il funzionale `map f xs` senza decomporre la lista `xs`, ma usando un espressione che contenga `foldr`, `f` ed eventualmente `xs`.

3. Definire il funzionale `map f xs` senza decomporre la lista `xs`, ma usando un espressione che contenga `foldl`, `f` ed eventualmente `xs`.

4. Argomentare brevemente sul perché non sia possibile definire `foldl` e `foldr` usando `map`.

Esercizio 4 (Facoltativo) Dato un numero intero positivo n, le *partizioni* di n sono tutti i modi in cui è possibile scrivere n come somma di altri numeri interi positivi.

Ad esempio, le partizioni di 4 sono le sequenze `[1,1,1,1]`, `[1,1,2]`, `[1,3]`, `[2,2]`, `[4]`. Sono considerate uguali partizioni che differiscono solo per l'ordine, quindi ad esempio, non vanno considerate nelle partizioni di 4 anche `[3,1]` oppure `[1,2,1]`.

1. Scrivere una funzione Haskell `part :: Int -> Integer` che calcola il numero di partizioni di un certo numero n. Ad esempio, `part 4` calcola 5.

2. Se invece considero diverse tra loro anche partizioni che differiscono solo per l'ordine, quante sono?

3. Scrivere poi una funzione Haskell `parts :: Int -> [[Int]]` che calcola la lista delle partizioni di n. Ad esempio, `parts 4` deve ritornare la lista `[[1,1,1,1], [1,1,2], [1,3], [2,2], [4]]` (potrebbe ovviamente essere diverso l'ordine in cui si scrivono, ma suggerisco di seguire un ordine tanto nella generazione che nel conteggio).

4. (Facile) Ma scrivere `part` usando `parts`? E la complessità è molto maggiore della `part` originaria?