TPFI 2021/22
Hw 1: Programmazione su Liste

docente: I. Salvo – Sapienza Università di Roma

assegnato: 8 marzo 2021, consegna 14 marzo 2021

Nota: Consegnare un unico file testo con estensione `.hs` (preferibilmente di nome `HW1-NomeCognome.hs`). Scrivere la soluzione di eventuali punti 'teorici' (esempio, esercizio **3.3**) semplicemente come commento nel codice Haskell.

Esercizio 1 Date due liste xs e ys, diciamo che la lista zs è uno *shuffle* di xs e ys se zs contiene tutti i valori di xs e ys nello stesso ordine in cui appaiono in xs e ys

Esempio Date le liste `[1,2,3]` e `[4,5]`, alcuni possibili shuffle sono `[1,2,3, 4,5]`, `[1,2,4,3,5]`, `[4,5,1,2,3]` etc. (quanti sono?)

Non sono shuffle invece (perché l'ordine non è corretto): `[1,2,3,5,4]`, `[1,3,4,2,5]`, `[5,1,4,2,3]`

1. Definire una funzione Haskell: `shuffle :: Eq a => [a] -> [a] -> [a] -> Bool` tale che `shuffle xs ys zs` è `True` se e solo se `zs` è uno shuffle di `xs` e `ys`. Assumente che `ys` e `xs` non abbiano elementi in comune.

2. Immaginare di avere una funzione `genShuffle :: [a] -> [a] -> [[a]]` che genera tutti i possibili shuffles tra `xs` e `ys`. Dare una definizione di `shuffle` (decisamente inefficiente) che non decompone liste, ma usa `genShuffle`.

3. Facoltativo: scrivere una funzione `shufflePlus` corretta anche nel caso in cui ci siano elementi duplicati o in comune tra `xs` e `ys`.

4. Facoltativo: scrivere la funzione `genShuffle :: [a] -> [a] -> [[a]]`.

Esercizio 2 Un *segmento* di una lista xs è una qualsiasi sottosequenza consecutiva di xs.

Definire una funzione Haskell: `segments :: [a] -> [[a]]` che presa una lista xs in input ritorna la lista di tutti i segmenti di xs. Ad esempio:

```
> segments [1,2,3]
[[], [1], [1,2],[1,2,3], [2], [2,3], [3]]
```

Esercizio 3

1. Definire il funzionale `zipWith f xs ys` senza decomporre le liste `xs` e `ys`, ma usando un espressione che contenga `applyL`, `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?