Programowanie funkcyjne — kolokwium nr 1, 10.12.2020

Instrukcja: Każde zadanie należy przesłać na Pegaza w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie.

Zadanie 1. Niech `type Point = (Double, Double)`. Napisać funkcję

`minDist :: [Point] -> (Point, Point, Double)`,

która zwróci parę punktów najbliższych sobie na liście podanej jako argument, wraz z odległością między nimi. W przypadku większej liczby możliwych rozwiązań należy wybrać dowolne; także kolejność elementów w parze wynikowej nie ma znaczenia. W rozwiązaniu można użyć faktu, iż `(Double, Double)` jest w klasie `Eq`. Przykładowo,

`minDist [(1,5),(1,1),(10,1),(-4,1)] = ((1.0,1.0),(1.0,5.0),4.0)`.

Zadanie 2. Rozważmy następującą definicję drzewa:

`data Tree a = Empty | Node a (Tree a) (Tree a)`.

Napisać funkcję

`findPath :: Eq a => a -> Tree a -> [a]`,

która zwraca ścieżkę (w postaci listy) od korzenia drzewa do elementu podanego jako pierwszy argument. Proszę założyć, że w drzewie nie ma powtarzających się elementów. Na przykład dla

```
t=Node 10 (Node 5 (Node 4 Empty Empty) (Node 6 Empty Empty)) (Node 20
Empty Empty)
```

mamy `findPath 6 t = [10,5,6]` oraz `findPath 7 t = []`.

Zadanie 3. W permutacji liczb od 1 do n pozycję k nazywamy *zamykającą*, jeżeli wśród pozycji od 1 do k znajdują się wszystkie liczby od 1 do k. Napisać funkcję

`cp :: [Integer] -> [Integer]`,

która dla permutacji podanej jako lista wyliczy wszystkie jej pozycje zamykające. Przykładowo, `cp [1,2,3] = [1,2,3]`, zaś `cp [3,2,1] = [3]`.