# Programowanie funkcyjne 2014
# Lista 4

kzi

4 listopada 2014

**Uwaga:** Jeśli zadanie jest niedospecyfikowane, to braki w specyfikacji można uzupełnić w dowolny sensowny! sposób.

### Zadania w Haskell'u

Zadbaj o to, żeby twoje rozwiązania były w miarę wydajne.

1. Zdefiniuj typ algebraiczny `Graph a` dla grafów skierowanych w postaci list sąsiedztwa, o wierzchołkach typu `a`, który będzie się nadawał do użycia w kolejnych zadaniach. Użyj `newtype`. `(1ptk)`
   (Czy wiesz czemu `newtype` jest tutaj lepszy od `data`?)

2. Zdefiniuj funkcję `short_path :: Eq a => Graph a -> a -> a -> Maybe [a]`, która zaaplikowana do $g$ $v_1$ $v_2$ zwraca najkrótszą ścieżkę z $v_1$ do $v_2$ w $g$, gdy taka istnieje, lub `Nothing` wpp. `(5ptk)`

3. Zdefiniuj funkcję `ham_cycle :: Eq a => Graph a -> Maybe [a]`, która dla danego grafu zwraca cykl Hamiltona, jeśli taki istniej, lub `Nothing` wpp. `(5ptk)`

   *Wskazówka: Problem znalezienia cyklu Hamiltona jest trudny, więc algorytm jest łatwy.*

   *Wskazówka: Zdefiniuj pomocniczą funkcję, która dla grafu i wierzchołka sprawdza, czy istnieje ścieżka Hamiltona rozpoczynająca się w tym wierzchołku.*

4. Zdefiniuj funkcję `rev :: Eq a => Graph a -> Graph a`, która odwaraca wszystkie krawędzie w grafie. Poza pierwszym pobraniem złożoność pobrania wszystkich sąsiadów danego wierzchołka w grafie wynikowym powinna być conajwyżej $O(n)$, gdzie $n$ to liczba wierzchołków w grafie wejściowym.

   Uwaga: Jeśli zbiór krawędzi grafu jest reprezentowany przez funkcję (tak jak na wykładzie), to przy naiwnej implementacji pobranie wszystkich sąsiadów danego wierzchołka w grafie wynikowym będzie prawdopodobnie $O(n+m)$, gdzie $m$ jest ilością krawędzi w grafie wejściowym, a $n$ jest tym co wyżej. `(4ptk)`

   a) `+2pkt` jeśli krawędzie w grafach są reprezentowane za pomocą funkcji, a mimo tego dalej mamy dobrą złożoność.

5. Zdefiniuj typ rekurencyjny

   ```
   Tree a = Node { val :: a, chl :: [Tree a] }
   ```

   i funkcję `tree_of :: Graph a -> Maybe (Tree a)`, która sprawdza, czy graf jest drzewam i jeśli tak zwraca wartość typy `Tree a` reprezentującą to drzewo, wpp `Nothing`. `(4ptk)`