

# Programowanie funkcyjne 2016

Grupa kzi  
Lista 4  
(8.11.2016)

UWAGA†: Jeśli zadanie jest niedospecyfikowane, to braki w specyfikacji można uzupełnić w dowolny sensowny(!) sposób.

## ZADANIA W HASKELLU

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 deklaracji `newtype`. (1 PTK) (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 v1 v2` zwraca najkrótszą ścieżkę z `v1` do `v2` w `g`, gdy taka istnieje, lub `Nothing` w.p.p. (5 PTK)
3. Zdefiniuj funkcję `ham_cycle :: Eq a => Graph a -> Maybe [a]`, która dla danego grafu zwraca cykl Hamiltona, jeśli taki istnieje, lub `Nothing` w.p.p. (5 PTK)  
Wskazówka: Problem znalezienia cyklu Hamiltona jest trudny, więc algorytm o optymalnej złożoności jest łatwy.
4. Zdefiniuj funkcję `rev :: Eq a => Graph a -> Graph a`, która odwraca wszystkie krawędzie w grafie. Poza pierwszym pobraniem złożoność pobrania wszystkich sąsiadów danego wierzchołka w grafie wynikowym powinna być  $\mathcal{O}(n)$ , 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  $\mathcal{O}(n + m)$ , gdzie  $m$  jest ilością krawędzi w grafie wejściowym, a  $n$  jest tym co wyżej. (4 PTK)

- a) +2 PTK 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 niepustym drzewem i jeśli tak zwraca wartość typu `Tree a` reprezentującą to drzewo, w.p.p. `Nothing`. (4 PTK)