Paradygmaty programowania - ćwiczenia
Lista 4

Na wykładzie zostały zdefiniowane grafy. Poniżej przedstawiono funkcje, sprawdzające czy istnieje ścieżka pomiędzy zadanymi wierzchołkami grafu. Wykorzystano obejście grafu wszerz, co wymaga użycia kolejki (reprezentowanej tu przez listę) do przechowywania wierzchołków, oczekujących na zwizytowanie. Ze względu na możliwe cykle w grafie, konieczne jest pamiętanie już zwizytowanych wierzchołków.

OCaml

```
type 'a graph = Graph of ('a -> 'a list);;

let pathExists (Graph succ) (fromNode, toNode) =
  let rec search visited queue =
    match queue with
        []    -> false
      | h::t -> if List.mem h visited then search visited t
                else h = toNode || search (h::visited) (t @ succ h)
  in search [] [fromNode]
;;
```

Dla grafu g z wykładu:
pathExists g (4, 1) = true;;
pathExists g (0, 4) = false;;

Scala

```
sealed trait Graphs[A]
case class Graph[A](succ: A=>List[A]) extends Graphs[A]

def pathExists[A](g: Graph[A])(fromNode: A, toNode: A): Boolean =
  def search(visited: List[A])(queue: List[A]): Boolean =
    queue match
      case h::t =>
        if visited contains h then search(visited)(t)
        else h == toNode || search(h :: visited)(t ::: (g succ h))
      case Nil => false
  search(Nil) (List(fromNode))
```

Dla grafu g z wykładu:
pathExists(g)(4,1) == true
pathExists(g)(0,4) == false

Paradygmaty programowania - ćwiczenia
Lista 4

1. Podaj (i wyjaśnij!) typy poniższych funkcji (samodzielnie, bez pomocy kompilatora OCamla!) :
   a) **let** f1 x y z = x y z;; b) **let** f2 x y = **function** z -> x::y;;

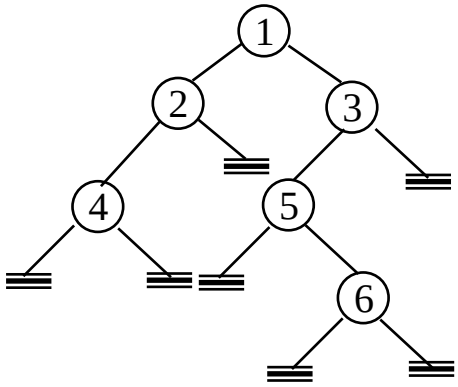
2. (OCaml) Napisz dowolną funkcję f: 'a -> 'b .

W poniższych zadaniach funkcje należy napisać w obu językach: OCaml i Scala (wykorzystując mechanizm dopasowania do wzorców!).
Na wykładzie zostały zdefiniowane drzewa binarne i grafy.
OCaml: type 'a bt = Empty | Node of 'a * 'a bt * 'a bt
type 'a graph = Graph of ('a -> 'a list)

3. Dla drzew binarnych, zdefiniowanych na wykładzie, napisz funkcję breadthBT : 'a bt -> 'a list obchodzącą drzewo wszerz i zwracającą zawartość wszystkich węzłów drzewa w postaci listy. Np. dla poniższego drzewa tt

breadthBT tt = [1; 2; 3; 4; 5; 6]

```
let tt = Node(1,
              Node(2,
                   Node(4,
                        Empty,
                        Empty
                       ),
                   Empty
                  ),
              Node(3,
                   Node(5,
                        Empty,
                        Node(6,
                             Empty,
                             Empty
                            )
                       ),
                   Empty
                  )
             );;
```



4. W *regularnym drzewie binarnym* każdy z węzłów jest bądź liściem, bądź ma stopień dwa (patrz np. Cormen i in.). Zauważ, że drzewa 'a bt są drzewami regularnymi – traktujemy konstruktor Empty jako liść.
   Długość ścieżki wewnętrznej *i* regularnego drzewa binarnego jest sumą, po wszystkich węzłach wewnętrznych drzewa, głębokości każdego węzła. Długość ścieżki zewnętrznej *e* jest sumą, po wszystkich liściach drzewa, głębokości każdego liścia. Głębokość węzła definiujemy jako liczbę krawędzi od korzenia do tego węzła.
   Napisz dwie funkcje, obliczające odpowiednio
   a) długość ścieżki wewnętrznej
   b) długość ścieżki zewnętrznej
   zadanego regularnego drzewa binarnego.
   Zauważ, że dla regularnych drzew binarnych o n węzłach wewnętrznych zachodzi $e = i + 2n$, np. dla powyższego drzewa tt $n = 6$, $i = 9$, $e = 21$. Ta interesująca własność nie jest potrzebna w tym zadaniu.

5. Dla grafów, zdefiniowanych na wykładzie, napisz funkcję depthSearch : 'a graph -> 'a -> 'a list, obchodzącą graf w głąb zaczynając od zadanego wierzchołka i zwracającą zawartość zwizytowanych węzłów w postaci listy. Np. dla grafu g z wykładu:
   depthSearch g 4 = [4; 0; 3; 2; 1]