AVL drevo

Avtor: JakMar17

Podatkovni tip "avltree"

AVL drevo je lahko:

- prazno
- ima podatke o višini, levem sinu in desnem sinu

```
type visina = int
type vrednost = int

type avltree =
| Empty
| Node of vrednost * visina * avltree * avltree
```

Primer avl drevesa

```
    5
   / \
  3   8
 / \
1   4
```

```
let t =
  Node (5, 3,
    Node (3, 2,
      Node (1, 1, Empty, Empty),
      Node (4, 1, Empty, Empty)
    ),
    Node(8, 2, Empty, Empty)
  )
```

Funkcija višina

```
let visina = function
  | Empty -> 0
  | Node (_, v, _, _) -> v
```

Ustvarjanje novega lista, vozlišča

Lista

Funkcija naj ustvari nov list

```
let list v = Node (v, 1, Empty, Empty)
```

Vozlišča

Funkcija ustvari novo vozlišče s sinovoma

```
let node (vrednost, levi, desni) =
    Node (vrednost, 1+ max(visina levi)(visina desni), levi, desni)
```

Drevo ⇒ seznam

Naloga: definirajte funkcijo toList: `avltree -> int list`, ki elemente drevesa vrne kot urejen seznam števil. Za združevanje seznamov ima OCaml operator `@`.

```
Rekurzivne funkcije je v OCaml potrebno definirati z besedo "let rec"
```

```
let rec toList = function
  | Empty -> []
  | (Node(x, _, l, d)) -> toList l @ [x] @ toList d
```

Iskanje

Algoritem, ki ugotovi, ali je dani x v drevesu t:

- če je t prazno drevo, se x ne pojavi;
- če je t vozlišče z vsebino y in poddrevesoma l ter r:
  - če je x = y, se x pojavi;
  - če je x < y, iščemo v poddrevesu l;
  - če je x > y, iščemo v poddrevesu r.

Najprej definiramo metodo za primerjavo dveh int znakov

```
type order = Manj | Enako | Vec

let primerjaj x y =
```

```
  match compare x y with
    | 0 -> Enako
    | r when r < 0 -> Manj
    | _ -> Vec
```

Rekurzivno iskanje - vrača true/false

```
let rec iskanje v = function
  | Empty -> false
  | (Node (vrednost, _, l, d)) ->
      begin match primerjaj v vrednost with
        | Enako -> true
        | Manj -> iskanje v l
        | Vec -> iskanje v d
      end
```

Vrtenje in uravnoteženje

Neuraunoteženost drevesa

Naloga: definirajte funkcijo `imbalance: avltree -> int`, ki vrne stopnjo neuravnoteženosti drevesa, tj. razliko med višinama levega in desnega poddrevesa.

```
(*od prej:*)
let visina = function
  | Empty -> 0
  | Node (_, v, _, _) -> v

(*na novo:*)
let imbalance = function
      | Empty -> 0
      | Node(_, _, l, d) -> visina l - visina d
```

Uravnoteženje drevesa

```
let rotacijaLevo = function
    | Node (x, _, a, Node(y, _, b, c)) -> node (y, node(x, a, b), c)
    | t -> t

let rotacijaDesno = function
    | Node (y, _, Node(x, _, a, b), c) -> node (x, a, node(y, b, c))
    | t -> t

let uravnotezi = function
    | Empty -> Empty
    | Node (x, _, l, d) as t ->
      begin match imbalance l with
```

```
      | 2 ->
        begin match imbalance l with
          | -1 -> rotacijaDesno (node (x, rotacijaLevo l, d))
          | _ -> rotacijaDesno t
        end
      | -2 ->
          begin match imbalance d with
            | 1 -> rotacijaLevo(node (x, l, rotacijaDesno d))
            | _ -> rotacijaLevo t
          end
      | _ -> t
      end
```

Dodajanje elementa v drevo

```
let rec dodaj x = function
    | Empty -> list x
    | Node (y, _, l, d) as t ->
        begin match primerjaj x y with
          | Enako -> t
          | Manj -> uravnotezi (node (y, dodaj x l, d))
          | Vec -> uravnotezi (node (y, l, dodaj x d))
    end
```

Odstranjevanje elementa iz drevesa

```
let rec odstrani x = function
    | Empty -> Empty
    | Node (y, _, l, d) ->
      let rec odstraniNaslednika = function
        | Empty -> failwith "Ne morem"
        | Node (x, _, Empty, d) -> (d, x)
        | Node (x, _, l, d) ->
          let (l', y) = odstraniNaslednika l in
            (uravnotezi (node (x, l', d)), y)
      in
      match primerjaj x y with
        | Manj -> uravnotezi (node (y, odstrani x l, d))
        | Vec -> uravnotezi (node (y, l, odstrani x d))
        | Enako ->
          begin match (l, d) with
            | (_, Empty) -> l
            | (Empty, _) -> d
            | _ -> let (d', y') = odstraniNaslednika d in
              uravnotezi (node (y', l, d'))
          end
```