```
(* glej tudi https://www.cs.cornell.edu/courses/cs312/2007fa/lectures/lec14.html *)

type height = int
type value = int

datatype avltree = Empty | Node of value * height * avltree * avltree

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

fun height Empty = 0
  | height (Node (_, h, _, _)) = h

fun leaf v = Node (v, 1, Empty, Empty)

fun node (v, l, r) = Node (v, 1 + Int.max (height l, height r), l, r)

val t' =
    node (5,
        node (3,
            node (1, Empty, Empty),
            node (4, Empty, Empty)
        ),
        node (8, Empty, Empty)
    )

val t'' = node (5, node (3, leaf 1, leaf 4), leaf 8)

fun toList Empty = []
  | toList (Node (x, _, l, r)) = toList l @ [x] @ toList r

(* tip: int -> avltree -> bool *)
fun search x Empty = false
  | search x (Node (y, _, l, r)) =
    case Int.compare (x, y)
     of EQUAL => true
      | LESS => search x l
      | GREATER => search x r

val test1 = search 1 t
val test2 = search 42 t

fun rotateLeft (Node (x, _, a, Node (y, _, b, c))) = node (y, node (x, a, b), c)
  | rotateLeft t = t

(* alternativni zapis s case *)
fun rotateRight t =
  case t
   of Node (y, _, Node (x, _, a, b), c) => node (x, a, node (y, b, c))
    | _ => t

fun imbalance Empty = 0
  | imbalance (Node (_, _, l, r)) = height l - height r

fun balance Empty = Empty
  | balance (t as Node (x, _, l, r)) =
    case imbalance t
     of 2 =>
        (case imbalance l
          of ~1 => rotateRight (node (x, rotateLeft l, r))
           | _ => rotateRight t)
      | ~2 =>
        (case imbalance r
```

```
          of 1 => rotateLeft (node (x, l, rotateRight r))
          |  _ => rotateLeft t)
      | _ => t

fun add x Empty = leaf x
  | add x (t as (Node (y, _, l, r))) =
    case Int.compare (x, y)
     of EQUAL => t
      | LESS => balance (node (y, add x l, r))
      | GREATER => balance (node (y, l, add x r))

fun remove x Empty = Empty
  | remove x (Node (y, _, l, r)) =
    let fun removeSuccessor Empty = raise Fail "impossible"
          | removeSuccessor (Node (x, _, Empty, r)) = (r, x)
          | removeSuccessor (Node (x, _, l, r)) =
                    let val (l', y) = removeSuccessor l
                    in (balance (node (x, l', r)), y)
                    end
    in
        case Int.compare (x, y)
         of LESS => balance (node (y, remove x l, r))
          | GREATER => balance (node (y, l, remove x r))
          | EQUAL =>
            case (l, r)
             of (_, Empty) => l
              | (Empty, _) => r
              | _ =>
                let val (r', y') = removeSuccessor r
                in balance (node (y', l, r'))
                end
    end
```