Exam Functional Programming

Date: Thu 31 Mar 2022

Time: 9:00 – 10:30

Location: R10

Teacher: J. Geurts

Resources: laptop with EFIT

Grading: 100 points

To be submitted:

* the .elm file with implementations and test cases:
  + all exercises may be put in one .elm file
  + you don’t need to write “elm-test” test cases; function calls in the my\_results is sufficient (with explanations)
* screen shot of the output
* if you fear that some part of your intelligence remains unnoticed: a document with further explanations is always welcome

For **all** exercises:

* provide meaningful test cases (not only the given examples) with an explanation of the purpose of the test case
* do not use the list concatenator (++) when the list cons (::) is applicable
* ***only*** use the following standard functions: List.map, List.foldr, List.foldl, List.filter   
  (and the indicated functions in the exercises)

# diffOne (20p)

Write the function:

diffOne: List a -> List a -> Bool

It checks if the elements in two lists are pairwise identical except for the values on 1 index position.   
In other words: on one index position: the values must be different, on all other index positions: the values must be identical.

Examples:

-- differs only at the 2nd index position:  
diffOne [2,-3,**42**,9] [2,-3,**73**,9] == True

-- differs at 2nd and 4th index position:  
diffOne [2,-3,**42**,9,**333**] [2,-3,**73**,9,**444**] ==False

Tips:

* use list recursion and pattern matching

# runLengthEncoding (20p)

Write the function:

runLengthEncoding: Char -> List Char -> (Int, List Char)

For a given character, it counts how many times it occurs *at the beginning* of the list.   
It returns a new list where those characters are removed, together with the amount of occurences of this given character.

Examples:

runLengthEncoding 'a' ['a','a','a','b','x','a'] ==   
 (3, ['b','x','a'])  
 -- 3 a’s are detected; we stopped because of the b

runLengthEncoding 'B' ['A','A','A','B','X','A'] ==   
 (0, ['a','a','a','b','x','a'])   
 -- no B’s are found at the beginning

Tips:

* use list recursion and pattern matching

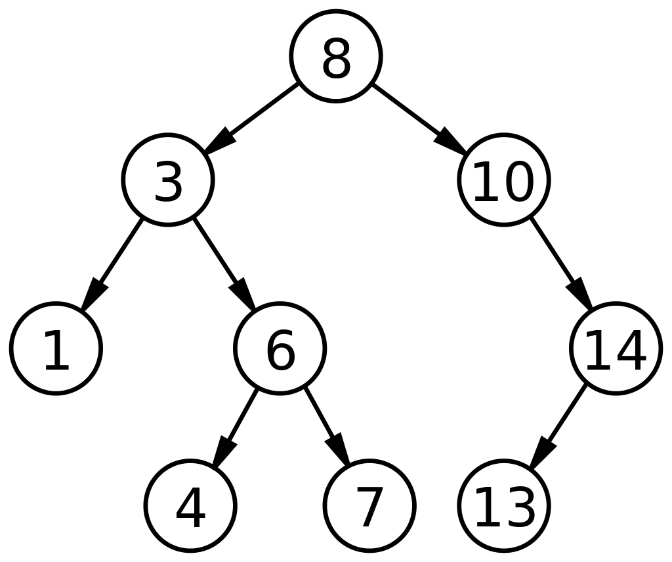
# BST (3x20p)

A Binary Search Tree (BST) is a tree with nodes. Each node contains an integer value, and a left subtree and a right subtree. (Sub)trees may be empty.

In a BST, all values in the left subtree are smaller than the value in the node, and all values in the right subtree are larger than the value in the node. There are no nodes with duplicate values.

For your information: a PDF of the wiki page is added for this exam. Be careful with your time management: a great deal of this website is not relevant for this exercise.

Example: (from Wikipedia):



Given is the following type definition:

type Tree  
 = Nil -- empty  
 | Node Int Tree Tree -- node with value + left + right

Write the functions:

toString: Tree -> String  
insert: Int -> Tree -> Tree  
search: Int -> Tree -> Bool

Function toString gives a human readable representation of the Tree. The exact layout (with the amount of spaces as seen below etc.) is not important, as long as the structure of the tree is recognizable.   
You may change the parameters of the function according to your wishes.   
Tip: add an Int parameter to indicate the depth of this (sub)tree.

Function insert creates a new tree where the integer value is added at the correct position.   
Note: it will always be added at a leaf of the tree. In the picture above: value 9 would be inserted as a new left node of 10, and value 5 would be inserted as a new right node of 4.

Function search checks if the requested integer value is in the tree.

Examples:

tr0 = Nil

tr1 = Node 4 Nil Nil

tr2 = Node 8 (Node 3 (Node 1 Nil Nil)

(Node 6 tr1 (Node 7 tr0 tr0))) Nil

tr3 = insert 10 tr2

tr4 = insert 14 tr3

tr5 = insert 13 tr4

-- tr2 is the left subtree of the wiki-example; tr5 is the full tree

my\_results: List String

my\_results =

[

"-- output --\n",

toString tr1,

toString tr3,

toString tr5,

pr <| search 6 tr5,

pr <| search 12 tr5,

"\n-- end --"

]

gives:

= 4

== 8

L- 3

L- 1

R- 6

L- 4

R- 7

R- 10

== 8

L- 3

L- 1

R- 6

L- 4

R- 7

R- 10

R- 14

L- 13

True

False

Tips:

* use list recursion and pattern matching
* to get a string of spaces of a given length (e.g. for indentation of a subtree at a certain depth), use String.padLeft, see:

