You are permitted to refer to the suggested references and lecture notes specific to this course. But you are not permitted to refer to any other resources or to communicate with anybody else during this exam.

Download the accompanying zip file from Blackboard. Solve each problem below using Haskell on the cs-parallel server. Do not modify the provided function signatures or file names. Test your functions using the examples in the provided zip file; you are encouraged to also create some additional examples to test more thoroughly. When you are ready to submit, compress your solution files into a zip file, and upload to Blackboard. Double-check that you have submitted all the files you intended.

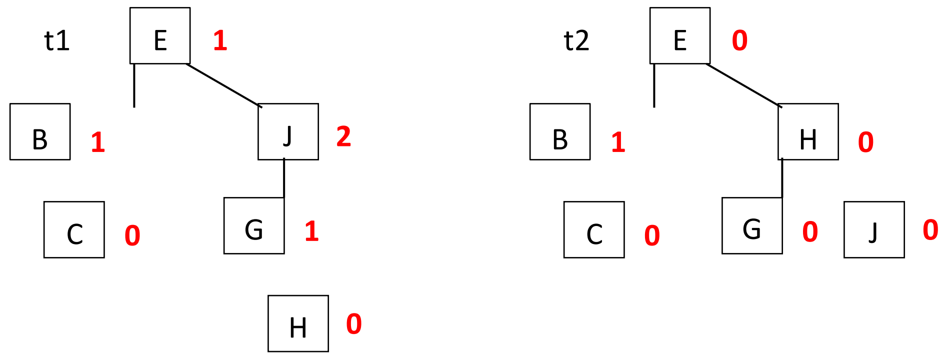
1. Numerical functions:
   a. (roundAwayFromZero x) takes a double x, and returns the nearest integer that is not closer to 0. Examples: roundAwayFromZero 3.2 returns 4, and roundAwayFromZero (–3.2) returns –4. Hint: Haskell provides several predefined functions that convert doubles to integers such as round, truncate, floor, ceiling.
   b. Given data type Number defined as follows:
      data Number = I Integer | D Double deriving (Eq, Ord, Read, Show)
      (plus x y) adds Numbers x and y, coercing integers to doubles only when necessary.
      Example: plus (I5) (D 3.14) returns (D 8.14).

2. (encode xs ys zs) replaces each character in xs by the corresponding character of ys within the string zs. Assume each character of zs will appear in xs. Examples: encode “abc” “123” “abacacb” returns “1213132”, and encode “abcd” “cdab” “ddccbbaa” returns “bbaaddcc”.

3. (subseq f xs) uses function f applied to the natural numbers to generate a sequence of indexes, and returns a subsequence of the elements of list xs at those indexes, stopping when an index is reached that is out of range. Examples:
   subseq (^2) [0..25] returns [0,1,4,9,16,25].
   subseq (^2) ['A'..'Z'] returns "ABEJQZ".
   subseq (2^) [0..25] returns [1,2,4,8,16].
   subseq (2^) ['A'..'Z'] returns "BCEIQ".

4. (binaryStrings k) generates a list of all binary strings of length k in ascending order.
Example: binaryStrings 3 returns ["000","001","010","011","100","101","110","111"].
Hint: There are 2^k strings. Consider the first 2^{k-1} strings and the last 2^{k-1} strings.

The next two problems refer to data type Tree and trees t1 and t2, defined as follows:
data Tree a = Nil | Node a (Tree a) (Tree a) deriving (Eq, Ord, Read, Show)
t1 = Node 'E' (Node 'B' Nil (Node 'C' Nil Nil)) (Node 'J' (Node 'G' Nil (Node 'H' Nil Nil)) Nil)
t2 = Node 'E' (Node 'B' Nil (Node 'C' Nil Nil)) (Node 'H' (Node 'G' Nil Nil) (Node 'J' Nil Nil))



5. Height-balance of a binary tree:
   a. (height t) returns the height of Tree t, where here we define height to be the number of levels in the tree. Examples: (height t1) returns 4 and (height t2) returns 3.
   b. (unbalanced t) returns the sum over all nodes of the Tree t of the height difference between the node's taller subtree and its shorter subtree. Examples: (unbalanced t1) returns 5 and (unbalanced t2) returns 1, due to summing the height differences of each node as shown in red in the above picture.

6. (createTree xs) takes a sorted list of keys, and returns a Tree which is as balanced as possible. Each node either has two subtrees with equal numbers of keys, or its right subtree has one more key than its left subtree. Example: createTree "BCEGHJ" returns the tree t2 given above.

7. (walk x ps) takes start vertex x and list of pairs ps which represents the edge list of a directed graph such that each vertex has at most one outward edge. The walk begins at x, proceeds along the only possible path, and stops when either it reaches a dead end or it reaches a cycle. If a dead end, return the vertex where the walk ends. If a cycle, return 0. Examples:
walk 1 [(1,5),(2,6),(3,1)] returns 5.
walk 2 [(1,5),(2,6),(3,1)] returns 6.
walk 3 [(1,5),(2,6),(3,1)] returns 5.
walk 4 [(1,5),(2,6),(3,1)] returns 4.
walk 3 [(1,2),(2,6),(3,1)] returns 6.
walk 3 [(1,2),(2,3),(3,1)] returns 0.
walk 3 [(1,2),(2,1),(3,1)] returns 0.
Hint: Write two recursive helper functions. One searches the edge list to find the successor of a given vertex. The other has a parameter which is a list of vertices along the path so far.