Mintaillesztés, egyszerű rekurzió

0. Modul

Definiálj egy modult `Hazi4` néven!

1. Hegy

Hozzunk létre egy félhegyet `mountain` néven, amit jobbról lehet megmászni bal felfelé. A hegynek a szélessége 1-től n-ig 1-esével nő fentről lefelé. A hegynek egy `String`-nek kell lennie, új sorokkal elválasztva az egyes szélességeket. Új sort a `'\n'` karakterrel lehet írni. A hegy álljon `'#'` karakterekből. Az egyszerűség kedvéért lehet úgy is csinálni, hogy van `'\n'` a `String` végén, az ügyesebbeknek pedig lehet úgy is, hogy szép legyen és nincs `'\n'` a `String` végén. A megoldásban elágazás, `if-then-else` nem használható!

Segítség: Nem mondtam, hogy csak rekurzióval lehet megoldani ezt. Illetve van egy `(++) :: [a] -> [a] -> [a]` függvény is, amely két listát összefűz.

```
-- példa
mountain 5 == "#\n##\n###\n####\n#####\n" || mountain 5 == "#\n##\n###\n####\n#####"
-- konzolra kiírni a putStrLn függvénnyel lehet, ekkor a mountain 5 példára a következőt kell látni:
#
##
###
####
#####
```

2. 'a' karakterek egy szövegben

Definiáld a `countAChars` nevű függvényt, amely megszámolja egy szövegben található `'a'` karaktereket.

3. Lucas-sorozat

Számítsd ki a Lucas-sorozat n. elemét rekurzívan! A sorozat nulladik eleme a 2, első az 1, az n. eleme pedig az előző kettőnek az összege (mint a Fibonacci-sorozatban).
Függvény neve: `lucas`

Figyelem: Egy adott definíció esetén a 30. elemet már nagyon feltűnően nagyon lassan fogja kiszámolni, ezért azt javaslom, hogy arrafelé ne tesztelgessetek. Létezik annál jobb definíció is, de azt a mostani eszközökkel még nem lehet megcsinálni.

4. Hasznos függvény a jövőben

Definiáld a `longerThan` függvényt, amely megmondja, hogy listának több eleme van-e, mint egy megadott szám, és amely kikerüli a `length lista > n` defektjét végtelen listákon. Ugye a lista ha végtelen, akkor sosem kapjuk vissza azt, hogy `True` annak ellenére, hogy a végtelen nyilván nagyobb bármilyen egész számnál. Úgy szeretnénk megírni ezt a függvényt, hogy ilyen esetben is válaszoljon.
Használj rekurziót és mintaillesztést! Ne használj elágazást, illetve a `length` függvényt se!

Minden bemenetre tudnia kell válaszolni! (Tehát üres lista, végtelen lista, pozitív szám, negatív szám, 0 és ezek bármilyen típushelyes kombinációjára.)

5. Kiegészítés

Definiáld a `format` függvényt, amely kiegészít szóközökkel egy szöveget megadott szélességűre (ami egy tetszőleges egész szám)! Ha netán a megadott szélesség kisebb lenne, mint a kapott szöveg, akkor ne vágj le belőle! Negatív szélességekkel nem kell foglalkozni.
Használj mintaillesztést és rekurziót! Ne használj egyenlőségvizsgálatot és elágazást, továbbá ne használd a `replicate` és `genericReplicate` függvényeket sem!

Emlékeztető: Végtelen `String`-re is kell megoldást adnia.

Segítség: A megoldáshoz bőven elég csak a `(:)` függvény, mintaillesztés és rekurzió megfelelő használata.

6. Szöveg szavainak megszámozása

a.) Párhuzamos lépkedés a listán

Definiáld a `zip' :: [a] -> [b] -> [(a,b)]` függvényt, amely két lista elemeit összepárosítja azokon párhuzamosan haladva (tehát első elemet első elemmel, másodikat másodikkal, harmadikat harmadikkal, stb.). Ezt a függvény addig ismétli, amíg mindkét listában van elem; ha valamelyikből elfogynak az elemek, a művelet véget ér. (Természetesen a `zip` függvény létezik, amely ugyanezt csinálja, így az a függvény ebben a megoldásban **nem** használható.)

b.) Számozás

Definiáld a `numberWords` függvényt, amely megszámozza egy szöveg szavait 1-től kezdve. Az egyszerűség kedvéért a megszámozások legyenek csak egész számok.

Segítség: Használjuk a `words :: String -> [String]` függvényt, amely egy szöveget felbont annak szavaira whitespace-ek mentén (ebbe nem csak a szóköz tartozik, `'\n', '\r', '\f', '\t', '\v'` és még egy pár más).

7. Összefésülés

Definiáld a `merge` függvényt, amely két listát összefésül. Az első elem az első lista első eleme legyen, a második elem a második lista első eleme, majd így váltakozva következzenek az elemek. Ha az egyik listából elfogynak az elemek, akkor a másik lista maradék elemét fűzd az eredmény végéhez.

Bónusz: Feltételes elem

Definiáld a `singletonIf :: (Eq a, Num a) => a -> [a]` függvényt, amely egy értéket belerak egy listába akkor, ha az pozitív, különben az eredmény legyen üres lista. A megoldásban ne használj elágazást!

```
singletonIf (1 :: Integer) == [1]
singletonIf 0 == []
singletonIf (-1) == []
singletonIf (10 :: Int) == [10]
```

```
singletonIf (-2.3) == []
singletonIf 12.25 == [12.25]
```

Tesztfüggvények

```
allPassed :: Bool
allPassed = null allTests

allTests :: [(String, Bool)]
allTests = [(str,test) |
   (str, test) <- [
        ("mountain 3 == \"#\n##\n###\n\" || mountain 3 == \"#\n##\n###\"", mountain 3 == "#\n##\n###\n" |
      , ("mountain 0 == \"\"", mountain 0 == "")
      , ("mountain (-10) == \"\"", mountain (-10) == "")
      , ("mountain 40 == \"#\n##\n###\n####\n#####\n######\n#######\n########\n#########\n##########\n###
      , ("countAChars \"alma\" == 2", countAChars "alma" == 2)
      , ("countAChars \"Én elmentem a vásárba a fél pénzemmel.\" == 3", countAChars "Én elmentem a vásárb
      , ("countAChars \"Ebben nincs.\" == 0", countAChars "Ebben nincs." == 0)
      , ("countAChars \"\" == 0", countAChars "" == 0)
      , ("countAChars \"A nagy A betű nem számít.\" == 1", countAChars "A nagy A betű nem számít." == 1)
      , ("lucas (0 :: Int) == (2 :: Double)", lucas (0 :: Int) == (2 :: Double))
      , ("lucas 1 == (1 :: Float)", lucas 1 == (1 :: Float))
      , ("lucas (5 :: Integer) == (11 :: Int)", lucas (5 :: Integer) == (11 :: Int))
      , ("lucas 7 == 29", lucas 7 == 29)
      , ("lucas 8 == 47", lucas 8 == 47)
      , ("lucas (20 :: Int) == (15127 :: Integer)", lucas (20 :: Int) == (15127 :: Integer))
      , ("[1..] `longerThan` 1000", [1..] `longerThan` 1000)
      , ("not ([] `longerThan` 0)", not ([] `longerThan` 0))
      , ("not (replicate 50 'a' `longerThan` 51)", not (replicate 50 'a' `longerThan` 51))
      , ("[1..50] `longerThan` 25", [1..50] `longerThan` 25)
      , ("[] `longerThan` (-10)", [] `longerThan` (-10))
      , ("[0,2..] `longerThan` (-1)", [0,2..] `longerThan` (-1))
      , ("[0,2..] `longerThan` (-100)", [0,2..] `longerThan` (-100))
      , ("[1] `longerThan` (-1)", [1] `longerThan` (-1))
      , ("[[1,2],[4,9,10],[1,0]] `longerThan` (-3)", [[1,2],[4,9,10],[1,0]] `longerThan` (-3))
      , ("format (10 :: Int) \"alma\" == \"alma      \"", format (10 :: Int) "alma" == "alma      ")
      , ("format (4 :: Integer) \"\" == \"    \"", format (4 :: Integer) "" == "    ")
      , ("format (5 :: Integer) \"szilva\" == \"szilva\"", format (5 :: Integer) "szilva" == "szilva")
      , ("format (0 :: Int) \"\" == \"\"", format (0 :: Int) "" == "")
      , ("format (0 :: Integer) \"barack\" == \"barack\"", format (0 :: Integer) "barack" == "barack")
      , ("take 61 (format (60 :: Integer) (repeat 'a')) == replicate 61 'a'", take 61 (format (60 :: Inte
      , ("null (numberWords \"\")", null (numberWords ""))
      , ("null (zip' [] [1,2,3])", null (zip' [] [1,2,3]))
      , ("null (zip' [1,2,3,4] [])", null (zip' [1,2,3,4] []))
      , ("null (zip' [] [1..])", null (zip' [] [1..]))
      , ("zip' [9,10,5,0,1,2] \"szilva\" == [(9,'s'),(10,'z'),(5,'i'),(0,'l'),(1,'v'),(2,'a')]", zip' [9,
      , ("zip' [9,10,5,0,1,2] \"alma\" == [(9,'a'),(10,'l'),(5,'m'),(0,'a')]", zip' [9,10,5,0,1,2] "alma"
      , ("zip' \"alma\" ['a'..] == [('a','a'),('l','b'),('m','c'),('a','d')]", zip' "alma" ['a'..] == [('
      , ("zip' [101..] \"meggy\" == [(101,'m'),(102,'e'),(103,'g'),(104,'g'),(105,'y')]", zip' [101..] "m
      , ("take 10 (zip' [101..] [1..]) == [(101,1),(102,2),(103,3),(104,4),(105,5),(106,6),(107,7),(108,8
      , ("numberWords \"En elmentem a vasarba fel penzzel.\" == [(1 :: Int,\"En\"),(2,\"elmentem\"),(3,\"
      , ("numberWords \"Az alma nem esett messze a fajatol!\" == [(1,\"Az\"),(2,\"alma\"),(3,\"nem\"),(4,
      , ("take 10 (numberWords (cycle \"alma \")) == [(1 :: Integer,\"alma\"),(2,\"alma\"),(3,\"alma\"),(
      , ("merge [1,3,5,7,9] [2,4,6,8] == [1..9]", merge [1,3,5,7,9] [2,4,6,8] == [1..9])
      , ("merge [] [1,10,12] == [1,10,12]", merge [] [1,10,12] == [1,10,12])
```

```
    , ("merge [1,10,12] [] == [1,10,12]", merge [1,10,12] [] == [1,10,12])
    , ("null (merge [] [])", null (merge [] []))
    , ("merge [1,2,3,4,5] [6,7] == [1,6,2,7,3,4,5]", merge [1,2,3,4,5] [6,7] == [1,6,2,7,3,4,5])
    , ("merge \"ab\" \"cdefgh\" == \"acbdefgh\"", merge "ab" "cdefgh" == "acbdefgh")
    , ("take 25 (merge \"szilvafa\" (repeat 'b')) == \"sbzbiblbvbabfbabbbbbbbbbb\"", take 25 (merge "sz
  ], not test]
```

Tesztesetek

```
mountain 3 == "#\n##\n###\n" || mountain 3 == "#\n##\n###"
mountain 0 == ""
mountain (-10) == ""
mountain 40 == "#\n##\n###\n####\n#####\n######\n#######\n########\n#########\n##########\n###########\n#
countAChars "alma" == 2
countAChars "Én elmentem a vásárba a fél pénzemmel." == 3
countAChars "Ebben nincs." == 0
countAChars "" == 0
countAChars "A nagy A betű nem számít." == 1
lucas (0 :: Int) == (2 :: Double)
lucas 1 == (1 :: Float)
lucas (5 :: Integer) == (11 :: Int)
lucas 7 == 29
lucas 8 == 47
lucas (20 :: Int) == (15127 :: Integer)
[1..] `longerThan` 1000
not ([] `longerThan` 0)
not (replicate 50 'a' `longerThan` 51)
[1..50] `longerThan` 25
[] `longerThan` (-10)
[0,2..] `longerThan` (-1)
[0,2..] `longerThan` (-100)
[1] `longerThan` (-1)
[[1,2],[4,9,10],[1,0]] `longerThan` (-3)
format (10 :: Int) "alma" == "alma      "
format (4 :: Integer) "" == "    "
format (5 :: Integer) "szilva" == "szilva"
format (0 :: Int) "" == ""
format (0 :: Integer) "barack" == "barack"
take 61 (format (60 :: Integer) (repeat 'a')) == replicate 61 'a'
null (numberWords "")
null (zip' [] [1,2,3])
null (zip' [1,2,3,4] [])
null (zip' [] [1..])
zip' [9,10,5,0,1,2] "szilva" == [(9,'s'),(10,'z'),(5,'i'),(0,'l'),(1,'v'),(2,'a')]
zip' [9,10,5,0,1,2] "alma" == [(9,'a'),(10,'l'),(5,'m'),(0,'a')]
zip' "alma" ['a'..] == [('a','a'),('l','b'),('m','c'),('a','d')]
zip' [101..] "meggy" == [(101,'m'),(102,'e'),(103,'g'),(104,'g'),(105,'y')]
take 10 (zip' [101..] [1..]) == [(101,1),(102,2),(103,3),(104,4),(105,5),(106,6),(107,7),(108,8),(109,9),
numberWords "En elmentem a vasarba fel penzzel." == [(1 :: Int,"En"),(2,"elmentem"),(3,"a"),(4,"vasarba")
numberWords "Az alma nem esett messze a fajatol!" == [(1,"Az"),(2,"alma"),(3,"nem"),(4,"esett"),(5,"messz
take 10 (numberWords (cycle "alma ")) == [(1 :: Integer,"alma"),(2,"alma"),(3,"alma"),(4,"alma"),(5,"alma
merge [1,3,5,7,9] [2,4,6,8] == [1..9]
merge [] [1,10,12] == [1,10,12]
merge [1,10,12] [] == [1,10,12]
null (merge [] [])
merge [1,2,3,4,5] [6,7] == [1,6,2,7,3,4,5]
```

```
merge "ab" "cdefgh" == "acbdefgh"
take 25 (merge "szilvafa" (repeat 'b')) == "sbzbiblbvbabfbabbbbbbbbbb"
```