0. Modul

Hozz létre egy modult `Hazi7` néven!

0+. Infó

Feladat leírások még változhatnak, de nem a használható eszközök leírásában. Esetleg segítségek kerülnek hozzájuk, hogy "Használjuk xy függvényt".

Ez a házi a teljes két heti csomag, kb. 5-5 lenne, ha a jövő hét lenne a határidő.
A házikhoz, meg igazából most már mindenhez is érdemes nézegetni az elérhető függvények listáját, legegyszerűbb a weboldalról, ahol a sok oldalon oda van írva a feladat, hogy `Prelude`-beli, `Data.Char`-beli, `Data.List`-beli függvény újradefiniálása, azok értelemszerűen a megfelelő modulok importálásával használhatók rendesen. A "Hajtogatásokat" még el fogom mondani az utolsó zh után, a "További függvények" részre nem marad feltétlen idő, de azokat is érdemes nézegtni, mert hasznos függvények vannak ott is.

1. Szétbontás adott tulajdonság mentén

Alakítsd át magasabb rendű függvénnyé a már meglévő `splitOn` függvényt úgy, hogy az elemeket a kapott predikátum (tulajdonság) mentén bontsa fel. Nem új megoldást kell készíteni, hanem a már beküldött kódot kell csak átalakítani.

2. Üres listák

Definiálj egy függvényt `countEmptyLists` néven, amely megszámolja, hogy egyenként hány darab üres lista található a listában található listák listájában. A listák listája értelemszerűen véges, hiszen meg kell számolni, hogy hány üres lista van benne. Bármely más szinten lehet végtelen lista is.

Ebben a feladatban tetszőleges eszközök használhatók! Add meg a függvény legáltalánosabb típusát is!

3. Térkép

Hozd létre azt a **konstans** listát `mapping :: [(Char,Char)]` névvel, amely a `0-9`, `A-Z`, `a-z` karakterekhez hozzárendeli a jelenlegihez `+3`-mal eltolt karaktert az angol ábécé és az ebben a sorrendben megadott intervallumok szerint (tehát 'a'-ból 'd' lesz, 'c'-ből 'f', '8'-ból 'B', 'X'-ből 'a'). A hozzárendelés ezeken az elemeken legyen ciklikus, tehát pl. 'z'-hez '2'-t rendelünk.

Ez konstans, tehát egy darab érték, épp ezért nincs teszt, mert nincs mit tesztelni.

De pl. `('0','3')`, `('9','C')`, `('y','1')` eleme lesz ennek a listának.

4. Titkosítás

Felhasználva a fent definiált hozzárendeléseket definiálj egy függvényt `encodeCaesar` néven, amely titkosít egy `String`-et lusta módon (tehát végtelenre is működjön) úgy, hogy minden egyes karaktert a `String`-ben eltol a megfelelő módon, magasabb rendű függvényt használva (vagyis **nem** lehet listagenerátort és rekurziót használni).

Ha a szövegben olyan karakter fordul elő, amely a térképünkben nem szerepel, akkor azt helyettesítsük `'?'` karakterrel.

5. Visszafejtés

Mivel a `mapping` egy bijektív hozzárendelés, vagyis minden egyes értékhez egy és csakis egy érték tartozik, visszafejteni is egyszerű az egész kódsort, a térképből a jobb oldali karakterhez a bal oldali karaktert kell rendelni. Ehhez készítsd el a `decodeCaesar` nevű függvényt. Ebben is szintén magasabb rendű függvényt kell alkalmazni, **nem** lehet rekurziót vagy listagenerátort.

Ha a szövegben olyan karakter fordul elő, amely a térképünkben nem szerepel, akkor azt helyettesítsük `'?'` karakterrel.

6. Tömörítés, visszaállítás

Volt anno annak idején a `compress` és a `decompress` függvény, amiket meg kellett írni. Most vannak szebb eszközeink ugyanezen függvények megírására. A feladat újraírni ezeket lehetőleg minél rövidebben! A megoldásban **ne** használj rekurziót és listagenerátort, használj magasabb rendű függvényt.

Emlékeztetőül: Aki esetleg nem emlékszik, a compress függvény összetömörített egy listát úgy, hogy az egymás mellett lévő ugyanazon karaktereket megszámolta és visszaadta rendezett párként a számolt karaktert és a darabszámot. A decompress pedig visszaállította az eredeti szöveget. Most nem csak `String`-re kell működnie, hanem bármilyen listára, nyilván a megfelelő megszorításokkal.

Mindkét függvény esetében a listát csak egyszer járhatjuk végig! Nem jó, ha a gép túl sokat dolgozik, akkor több áramot is fogyaszt.

Segítég: Ne feledjük a `generic...` függvények létezését!

7. Sok függvényalkalmazás sok listán

Add meg az `apsOnLists` a függvényt, amely egy listányi függvényt alkalmaz listák listáján, azon belül a lista minden elemén. Elég a rövidebb lista hosszáig alkalmazni a függvényeket az elemeken. A függvénynek akkor is működnie kell, ha mindkét lista végtelen. Értelemszerűen konkrét eredményt nem fog adni sose, de lustaság miatt valamennyinek már elérhetőnek kell lennie. A megoldásban **ne** használj rekurziót és listagenerátort, használj magasabb rendű függvényt!

Segítség: Nem kell túlbonyolítani, a megoldás egyszerűbb, mint ahogy kinéz.

8. Lucas-sorozat jobban és gyorsabban

Definiáld a jól ismert Lucas-sorozatot `lucas :: [Integer]` néven. A sorozat továbbra is 2-vel kezdődik, 1-gyel folytatódik és minden n. elem az (n-1). és (n-2). elem összege, továbbá maga a sorozat végtelen. A függvénynek megközelítőleg a sorozat 100000. eleméig az 1 másodperces kiértékelési időkereten belül kell lennie! A megoldásban használj magasabb rendű függvényt, **ne** használj listagenerátort! Mivel a lista konstans és végtelen kell legyen, így rekurziót csak úgy lehet használni, ha top-level önmagára hivatkozik (tehát arra, hogy `lucas`, semmi másra).

9. Lucas-sorozat felhasználása

Definiáld az `isNotPrime :: Integral a => a -> Bool` függvényt, amely a Lucas-sorozat segítségével meghatározza egy számról, hogy biztosan **nem** prímszám-e.

A felhasználási módja a következő:

Legyen a vizsgált számunk `n`, erre vagyunk kíváncsiak, hogy prímszám-e. Vegyük a Lucas-sorozat `n.` elemét (a sorozat elemeit 0-tól számozva), az ott szereplő számból vonjunk ki `1`-et. Ha az úgy kapott szám `n`-nek nem a többszöröse, akkor `n` teljesen biztosan nem prímszám (ekkor adjon vissza a függvény `True` értéket). Ha az úgy kapott szám `n`-nek többszöröse, akkor nem tudunk a számról semmi biztosat állítani, ekkor a függvény adjon vissza `False` értéket.

10. Részlisták

Definiáld a `sublists :: [a] -> [[a]]` függvényt, amely egy listának az összes részlistáját megadja. Részlistának tekintendő a listából a közvetlenül egymás után következő elemekből képzett lista. A megoldásban használj magasabb rendű függvényeket, **ne** használj rekurziót, listagenerátort, lambdát, illetve ne írj ki explicit paramétereket se!

Segítség: Nézd meg az egyszerű rekurzió oldalát esetlegesen hasznos függvényekért.

Tesztelő függvény

```
allPassed :: Bool
allPassed = null allTests

allTests :: [(String, Bool)]
allTests = [(str,test) |
   (str, test) <- [
        ("splitOn even [1,2,4,3,5,4] == [[1],[],[3,5],[]]", splitOn even [1,2,4,3,5,4] == [[1],[],[3,5],[
      , ("splitOn isUpper \"Rovid Szoveg NAGY betukkel.\" == [\"\", \"ovid \", \"zoveg \", \"\", \"\", \"
      , ("splitOn (==2) [2,3,4,5,6,2,3,2,2,3,2] == [[],[3,4,5,6],[3],[],[3],[]]", splitOn (==2) [2,3,4,5,
      , ("case splitOn (\\_ -> True) [(+1), (*2), (30 `div`)] of [[],[],[],[]] -> True; _ -> False", case
      , ("case splitOn (even . ($ 10)) [(+1), (*2), (30 `div`), (40 `div`)] of [[f],[g],[]] | f 10 == 11,
      , ("splitOn id [True, False, False, False, True, False, True, True, False] == [[],[False,False,Fals
      , ("null (countEmptyLists [])", null (countEmptyLists []))
      , ("countEmptyLists [[[],[3],[]],[],[[1,2,0],[4,3,2]],[[]]] == [2 :: Int,0,0,1]", countEmptyLists [
      , ("countEmptyLists [[\"\",\"alma\",repeat 'a'],[],[\"szilva\",\"körte\",\"barack\",\"szőlő\"],[\"\
      , ("countEmptyLists [[[],[3],[]],[],[[1,2,0],[1..],[],[]],[[]]] == [2 :: Float,0,2,1]", countEmptyL
      , ("countEmptyLists [[],[[(+1),(+2)]],[[]],[[(`div` 2), (div 2)],[(mod 2), (`mod` 2)], []], [[(3-)]
      , ("encodeCaesar \"alma\" == \"dopd\"", encodeCaesar "alma" == "dopd")
      , ("encodeCaesar \"Encoderz9\" == \"Hqfrghu2C\"", encodeCaesar "Encoderz9" == "Hqfrghu2C")
      , ("encodeCaesar \"körte\" == \"n?uwh\"", encodeCaesar "körte" == "n?uwh")
      , ("take 10 (encodeCaesar (repeat 'a')) == take 10 (repeat 'd')", take 10 (encodeCaesar (repeat 'a'
      , ("take 10 (encodeCaesar (repeat 'ű')) == take 10 (repeat '?')", take 10 (encodeCaesar (repeat 'ű'
      , ("decodeCaesar \"dopd\" == \"alma\"", decodeCaesar "dopd" == "alma")
      , ("decodeCaesar \"Hqfrghu2C\" == \"Encoderz9\"", decodeCaesar "Hqfrghu2C" == "Encoderz9")
      , ("decodeCaesar \"n?uwh\" == \"k?rte\"", decodeCaesar "n?uwh" == "k?rte" )
      , ("take 10 (decodeCaesar (repeat 'd')) == take 10 (repeat 'a')", take 10 (decodeCaesar (repeat 'd'
      , ("take 10 (decodeCaesar (repeat '?')) == take 10 (repeat '?')", take 10 (decodeCaesar (repeat '?'
      , ("null (compress [] :: [(Integer,Integer)]) == []", null (compress [] :: [(Integer,Integer)]))
      , ("compress \"almafa\" == [('a',1),('l',1),('m',1),('a',1),('f',1),('a',1)]", compress "almafa" ==
      , ("compress \"compress\" == [('c',1),('o',1),('m',1),('p',1),('r',1),('e',1),('s',2)]", compress "
      , ("compress \"sssssszzzzziiiillvvaaaaaaaa\" == [('s',5),('z',6),('i',4),('l',2),('v',2),('a',8)]",
      , ("compress [1,1,1,3,2,2,4,4,4,4] == [(1,3),(3,1),(2,2),(4,4)]", compress [1,1,1,3,2,2,4,4,4,4] ==
      , ("compress [1,1,1,3,2,2,4,4,3,4,4] == [(1,3),(3,1),(2,2),(4,2),(3,1),(4,2)]", compress [1,1,1,3,2
      , ("null (decompress [])", null (decompress []))
      , ("decompress [('k',1),('a',1),('t',2)] == \"katt\"", decompress [('k',1),('a',1),('t',2)] == "kat
      , ("decompress [('A',3),('b',2),('k',5),('D',3)] == \"AAAbbkkkkkDDD\"", decompress [('A',3),('b',2)
      , ("decompress [(1,1),(2,2),(3,3),(4,4)] == [1,2,2,3,3,3,4,4,4,4]", decompress [(1,1),(2,2),(3,3),(
      , ("apsOnLists [(+1),(*2),(10-)] [[4,5,9],[10,3,0,2,5], [20,10,6,1], [909,84,292,10]] == [[5 :: Int
```

```
    , ("null (apsOnLists [] [[1..]])", null (apsOnLists [] [[1..]]))
    , ("null (apsOnLists [sqrt,(^2)] [])", null (apsOnLists [sqrt,(^2)] []))
    , ("apsOnLists [(+i) | i <- [1,3..]] [[0,2,3,4],[2,5],[],[10,0,(-5)]] == [[1 :: Integer,3,4,5],[5,8
    , ("apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 0 == [100,13]", apsOnLists [(+2),(*3
    , ("apsOnLists [(\\a -> if a then (1 :: Integer) else 0)] [[True,False,True],[],repeat True] == [[1
    , ("take 20 (apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 1) == [3 :: Double, 6, 9, 1
    , ("apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 2 == [1,16,81,4096]", apsOnLists [(+
    , ("lucas !! 30 == 1860498", lucas !! 30 == 1860498)
    , ("lucas !! 50 == 28143753123", lucas !! 50 == 28143753123)
    , ("lucas !! 100 == 792070839848372253127", lucas !! 100 == 792070839848372253127)
    , ("lucas !! 1000 == 971941777359081752079819820793264737377978791553456850827280810847725188184448
    , ("lucas !! 50000 == 240997478643825988804980698063344994982641163681531996803773375195393376825 13
    , ("isNotPrime 4", isNotPrime 4)
    , ("isNotPrime 6", isNotPrime 6)
    , ("not (isNotPrime 7)", not (isNotPrime 7))
    , ("not (isNotPrime 11)", not (isNotPrime 11))
    , ("not (isNotPrime 17)", not (isNotPrime 17))
    , ("not (isNotPrime 23)", not (isNotPrime 23))
    , ("isNotPrime 230", isNotPrime 230)
    , ("not (isNotPrime 705) -- ez az első olyan szám, ami ezen a teszten átmegy, mint összetett szám,
    , ("isNotPrime 1000", isNotPrime 1000)
    , ("sublists \"\" == [\"\"]", sublists "" == [""])
    , ("sublists \"a\" == [\"\",\"a\"]", sublists "a" == ["","a"])
    , ("sublists \"ab\" == [\"\", \"a\", \"ab\", \"b\"]", sublists "ab" == ["", "a", "ab", "b"])
    , ("sublists \"abc\" == [\"\", \"a\", \"ab\", \"b\", \"abc\", \"bc\", \"c\"]", sublists "abc" == ["
    , ("sublists \"abcd\" == [\"\", \"a\", \"ab\", \"b\", \"abc\", \"bc\", \"c\", \"abcd\", \"bcd\", \"
    , ("sublists [1,2,1] == [[], [1], [1, 2], [2], [1, 2, 1], [2, 1], [1]]", sublists [1,2,1] == [[], [
    , ("sublists [True,False,False,True] == [[], [True], [True, False], [False], [True, False, False],
    , ("take 22 (sublists [1..]) == [[], [1], [1, 2], [2], [1, 2, 3], [2, 3], [3], [1, 2, 3, 4], [2, 3,
```

Tesztek

```
splitOn even [1,2,4,3,5,4] == [[1],[],[3,5],[]]
splitOn isUpper "Rovid Szoveg NAGY betukkel." == ["", "ovid ", "zoveg ", "", "", "", " betukkel."]
splitOn (==2) [2,3,4,5,6,2,3,2,2,3,2] == [[],[3,4,5,6],[3],[],[3],[]]
case splitOn (\_ -> True) [(+1), (*2), (30 `div`)] of [[],[],[],[]] -> True; _ -> False
splitOn id [True, False, False, False, True, False, True, True, False] == [[],[False,False,False],[False]
null (countEmptyLists [])
countEmptyLists [[[],[3],[]],[],[[1,2,0],[4,3,2]],[[]]] == [2 :: Int,0,0,1]
countEmptyLists [["","alma",repeat 'a'],[],["szilva","körte","barack","szőlő"],[""]] == [1 :: Int,0,0,1]
countEmptyLists [[[],[3],[]],[],[[1,2,0],[4,3,2],[],[]],[[]]] == [2 :: Float,0,2,1]
countEmptyLists [[],[[(+1),(+2)]],[[]],[[(`div` 2), (div 2)],[(mod 2), (`mod` 2)], []], [[(3-)],[],[]]] =
encodeCaesar "alma" == "dopd"
encodeCaesar "Encoderz9" == "Hqfrghu2C"
encodeCaesar "körte" == "n?uwh"
take 10 (encodeCaesar (repeat 'a')) == take 10 (repeat 'd')
take 10 (encodeCaesar (repeat 'ű')) == take 10 (repeat '?')
decodeCaesar "dopd" == "alma"
decodeCaesar "Hqfrghu2C" == "Encoderz9"
decodeCaesar "n?uwh" == "k?rte"
take 10 (decodeCaesar (repeat 'd')) == take 10 (repeat 'a')
take 10 (decodeCaesar (repeat '?')) == take 10 (repeat '?')
compress [] == []
compress "almafa" == [('a',1),('l',1),('m',1),('a',1),('f',1),('a',1)]
compress "compress" == [('c',1),('o',1),('m',1),('p',1),('r',1),('e',1),('s',2)]
compress "sssssszzzzziiiiillvvaaaaaaaa" == [('s',5),('z',6),('i',4),('l',2),('v',2),('a',8)]
```

```
compress [1,1,1,3,2,2,4,4,4,4] == [(1,3),(3,1),(2,2),(4,4)]
compress [1,1,1,3,2,2,4,4,3,4,4] == [(1,3),(3,1),(2,2),(4,2),(3,1),(4,2)]
decompress [] == []
decompress [('k',1),('a',1),('t',2)] == "katt"
decompress [('A',3),('b',2),('k',5),('D',3)] == "AAAbbkkkkkDDD"
decompress [(1,1),(2,2),(3,3),(4,4)] == [1,2,2,3,3,3,4,4,4,4]
apsOnLists [(+1),(*2),(10-)] [[4,5,9],[10,3,0,2,5], [20,10,6,1], [909,84,292,10]] == [[5 :: Int,6,10],[20
null (apsOnLists [] [[1..]])
null (apsOnLists [sqrt,(^2)] [])
apsOnLists [(+i) | i <- [1,3..]] [[0,2,3,4],[2,5],[],[10,0,(-5)]] == [[1 :: Integer,3,4,5],[5,8],[],[17,7
apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 0 == [100,13]
take 20 (apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 1) == [3 :: Double, 6, 9, 12, 15, 18,
apsOnLists [(+2),(*3),(^4)] [[98,11], [1..], [1,2,3,8]] !! 2 == [1,16,81,4096]
lucas !! 30 == 1860498
lucas !! 50 == 28143753123
lucas !! 100 == 792070839848372253127
lucas !! 1000 == 971941777359081752079819820793264737377978791553456850827280810847725188184448152690806 1
lucas !! 50000 == 240997478643825988804980698063344994982641163681531996803773375195393376825136487164660
isNotPrime 4
isNotPrime 6
not (isNotPrime 7)
not (isNotPrime 11)
not (isNotPrime 17)
not (isNotPrime 23)
isNotPrime 230
not (isNotPrime 705) -- ez az első olyan szám, ami ezen a teszten átmegy, mint összetett szám, eddig a po
isNotPrime 1000
sublists "" == [""]
sublists "a" == ["","a"]
sublists "ab" == ["", "a", "ab", "b"]
sublists "abc" == ["", "a", "ab", "b", "abc", "bc", "c"]
sublists "abcd" == ["", "a", "ab", "b", "abc", "bc", "c", "abcd", "bcd", "cd", "d"]
sublists [1,2,1] == [[], [1], [1, 2], [2], [1, 2, 1], [2, 1], [1]]
sublists [True,False,False,True] == [[], [True], [True, False], [False], [True, False, False], [False, Fa
take 22 (sublists [1..]) == [[], [1], [1, 2], [2], [1, 2, 3], [2, 3], [3], [1, 2, 3, 4], [2, 3, 4], [3, 4
```