Feladatok

Folyó mellett (1 pont)

Sok magyar település, amely a Duna és Tisza folyók partján fekszenek, neve kezdődik az őket átszelő folyó nevével. Készítsünk függvényt, ami megadja egy településnévről, hogy megmondható-e belőle, hogy a település az előbb említett folyók valamelyike mellett fekszik!

```
byDunaOrTisza :: String -> Bool

byDunaOrTisza "Dunaújváros"
byDunaOrTisza "Tiszaújváros"
byDunaOrTisza "Dunaföldvár"
byDunaOrTisza "Dunakeszi"
byDunaOrTisza "Dunaharaszti"
byDunaOrTisza "Tiszakécske"
not $ byDunaOrTisza "Budapest"
not $ byDunaOrTisza "Szeged"
not $ byDunaOrTisza ""
```

Duplák (1 pont)

Adott egy kételemű listákból álló sorozat. Adjuk meg azt a függvényt, ami megadja azon listák számát, amelyben mindkét elem megegyezik!

```
howManyDoubles :: Eq a => [[a]] -> Int

howManyDoubles [] == 0
howManyDoubles [[1,1],[2,2]] == 2
howManyDoubles [[11,11],[46,46],[100,100]] == 3
howManyDoubles [['a','a'],['b','b']] == 2
howManyDoubles [['a','a'],['c','d'],['b','b']] == 2
howManyDoubles [['a','b'],['c','d'],['e','f']] == 0
```

Black Jack pontok (1 pont)

Adjuk meg azt a függvényt, ami egy véges listának visszaadja az összegét egy `Just` konstruktorba csomagolva, ha az elemek összege nem nagyobb, mint 21 és nincs 11-esnél nagyobb érték a listában. Adjunk vissza `Nothing`-ot, ha az összeg nagyobb mint 21, vagy ha van olyan elem, ami nagyobb, mint 11.

```
blackJackPoints :: Integral a => [a] -> Maybe a

blackJackPoints [1,2,3,4,5] == Just 15
blackJackPoints [1,2,3,4,5,6] == Just 21
blackJackPoints [1,2,3,4,5,6,7] == Nothing
blackJackPoints [22] == Nothing
blackJackPoints [12] == Nothing
```

```
blackJackPoints [11,10] == Just 21
blackJackPoints [] == Just 0
blackJackPoints [3,1,12,2] == Nothing
blackJackPoints [9,1,2,10,12,4,5,6] == Nothing
```

Hárommal nem osztható elem (2 pont)

Adjuk meg azt a függvényt, ami megkeresi az első elemet egy tetszőleges egész értékeket tartalmazó listában, ami nem osztható 3-mal és megadja annak sorszámát egy `Just` konstruktorba csomagolva. Ha minden elem osztható 3-mal akkor adjunk vissza `Nothing`-ot. A sorszámozás 1-től indul.

```
notDivisibleByThree:: Integral a => [a] -> Maybe Int

notDivisibleByThree [3,3,3,3,3,3,3,3] == Nothing
notDivisibleByThree [] == Nothing
notDivisibleByThree [3] == Nothing
notDivisibleByThree [3,2] == Just 2
notDivisibleByThree [3,3,2,4] == Just 3
notDivisibleByThree [10..] == Just 1
notDivisibleByThree ([3,6..40] ++ [0..]) == Just 15
```

Csoport (2 pont)

Írjuk meg a `crowd` függvényt, amely visszaadja a paraméterül kapott számú tömeget. Egy tömegben lévő embert a `"(-_-)"` karaktersorozat írja le, majd a további emberek a központi figura két oldalára állnak egyenlő arányban, nekik csak a fél arcuk látszik. Amennyiben páros számú tömeget szeretnénk megidézni, a tömeg közepén két ember álljon. Negatív számú ember nehezen létezik, így feltehető, hogy ez a szám nem lesz negatív.

```
crowd :: Int -> String

crowd 1 == "(-_-)"
crowd 3 == "(-_(-_-)_-)"
crowd 4 == "(-_(-_-)(-_-)_-)"
crowd 11 == "(-_(-_(-_(-_(-_(-_-)_-)_-)_-)_-)_-)"
crowd 12 == "(-_(-_(-_(-_(-_(-_-)(-_-)_-)_-)_-)_-)_-)"
crowd 0 == ""
```

Legalább n darab (2 pont)

Adjuk meg azt a függvényt, amely eldönti hogy egy elem megtalálható-e legalább n-szer egy listában!

```
atLeastNFrom :: Eq a => Int -> a -> [a] -> Bool

atLeastNFrom 0 3 [] == True
atLeastNFrom (-2) 'f' [] == True
```

```
atLeastNFrom (-10) 'f' "faablak" == True
atLeastNFrom 0 'a' "alma" == True
atLeastNFrom 2 'a' "alma" == True
atLeastNFrom 0 'd' "barack" == True
atLeastNFrom 1 'd' "barack" == False
atLeastNFrom 2 3 [3,3] == True
atLeastNFrom 4 3 [3,3] == False
atLeastNFrom 4 5 (repeat 5) == True
atLeastNFrom 31 'a' (cycle "abba") == True
```

Feltételes leképezés (2 pont)

Készítsünk függvényt, amely kap egy predikátumot, két leképezést/függvényt és egy listát, melynek elemeire alkalmazza az első vagy a második függvényt attól függően, hogy ha a feltétel teljesül az adott elemre, akkor az elsőt, ha nem teljesül, akkor a másodikat alkalmazza!

```
mapEither :: (a -> Bool) -> (a -> b) -> (a -> b) -> [a] -> [b]

mapEither even (+1) id [0..9] == [1,1,3,3,5,5,7,7,9,9]
mapEither (\x -> length x > 5) (\x -> "hosszu szo: " ++ x)
  id ["macska", "kutya", "lo", "etruszk cickany", "sas"] ==
  ["hosszu szo: macska","kutya","lo","hosszu szo: etruszk cickany","sas"]
mapEither (const False) (+2) (+1) [1..10] == [2..11]
mapEither (const True) (+1) (+2) [1..10] == [2..11]
mapEither odd (+2) (+2) [1..10] == [3..12]
take 10 (mapEither (\x -> x `mod` 5 == 0) (const 0) (*5) [1..]) ==
  [5,10,15,20,0,30,35,40,45,0]
```

Bukások száma (2 pont)

Egy tanár listák listájában tartja számon diákjai jegyeit (minden diákhoz egy lista tartozik, az osztályhoz pedig a listák listája). Azon tanulók, akiknek az átlaga nem éri el a 2-t, megbuknak. Lehetnek olyan diákok, akik nem feleltek még a félévben, ezeket hagyjuk figyelmen kívül. Definiáljuk azt a függvényt, amely megadja, hogy az osztály hány diákja bukott meg!

```
numberOfFails :: Integral a => [[a]] -> Int

numberOfFails [] == 0
numberOfFails [[],[]] == 0
numberOfFails [[5,5,4,5,3],[],[1,1,2,1,1],[2,2,2,2],[5,5]] == 1
numberOfFails [[2,2,3,2],[5,4,5],[2,2,2,2],[5,2,2,5]] == 0
numberOfFails [[1,1,5,5],[1,1,4,1,1],[2,2,3,2],[1,2,5,4,2],[1,2,3,1,2,1]] == 2
numberOfFails [[2,3,4,3,2,3],[1,2,1,1],[4,4,1,1,2,1,1],[1,3,5,4,4],[1,3,1,1]] == 2
```

Futamhossz kódolás (2 pont)

A futamhossz kódolás egy olyan szöveg tömörítési eljárás, ahol az adatban az ismétlődő karaktereket egyetlen értékként és számként tárolják az eredeti karaktersorozat helyett. Definiáld az `encode` függvényt, amely az előbb leírt módon tömörít egy paraméterként kapott szöveget!

Megjegyzés: A számok szöveggé alakításához használhatjuk a `show` függvényt.

```
encode :: String -> String

encode "" == ""
encode "a" == "a"
encode "abc" == "abc"
encode "aa" == "a2"
encode "abbbbabccc" == "ab4abc3"
encode "aaabbbcccabc" == "a3b3c3abc"
encode "aaabbbcccabc" == "a3b3c3abc"
take 30 (encode (cycle "aabbbacaas")) == "a2b3aca2sa2b3aca2sa2b3aca2sa2b"
```

Két lista összefésülése (2 pont)

Adjuk meg azt a függvényt, amely eldönti, hogy két lista páronkénti összefésüléséből állt-e elő egy harmadik! Azaz, ha egy összefésülést (egy elem az elsőből, egy elem a másodikból) végzünk, akkor a harmadik lista áll elő. A listák nem feltétlenül azonos hosszúságúak, ebben az esetben az összefésülés a hosszabb lista maradék elemeivel folytatódik.

```
mergedOf :: Eq a => [a] -> [a] -> [a] -> Bool

mergedOf [] [] [] == True
mergedOf []      [1,2]   [1,2] == True
mergedOf [3,1]   []      [3,1] == True
mergedOf [1,3,5] [2,4,6] [1,2,3,4,5,6] == True
mergedOf [1,3..10] [2,4..10] [1..10] == True
mergedOf [1,3..10] [2,4..10] [1..10] == True
mergedOf ["Nem", "messze", "alma", "fajatol"] ["esik", "az", "a", "."]
  ["Nem", "esik", "messze", "az", "alma", "a", "fajatol", "."]
mergedOf [1] [] [] == False
mergedOf [] [3,4] [] == False
mergedOf [] [3,4] [4,3] == False
mergedOf [] [] [4,3] == False
mergedOf [1,2] [3,4] [1,2,3,4] == False
mergedOf [] [] [1..] == False
mergedOf [1..] [1..] [1..] == False
```

Időjárás

Adattípusok definiálása (1 pont)

Készíts egy `Weather` felsorolási adattípust, amely az időjárást reprezentálja! Rendelkezzen a következő három adatkonstruktorral: `Sunny`, `Cloudy`, `Rainy`. Kérjünk a fordítótól automatikus példányosítást a `Show` és `Eq` típusosztályokra!

Készíts egy `Forecast` algebrai adattípust, amely az időjárás előrejelzést reprezentálja! Legyen `Prediction` az egyetlen adatkonstruktora, amelynek az első paramétere egy `Weather` típusú érték, a második pedig egy `Int`, ami az előrejelzés valószínűségét adja meg százalékban. Kérjünk a fordítótól automatikus példányosítást a `Show` és `Eq` típusosztályokra!

Nyaralási terv (3 pont)

Készítsd el a `summerVacation` függvényt, amely egy több napra szóló előrejelzésből megadja, hogy melyik az a leghosszabb sorozat/időszak, amikor érdemes elmenni nyaralni. Egy sorozata akkor számít nyaraló időnek, ha nem tartalmaz egyetlen csapadékos napot sem. (Ha több ugyanolyan hosszú időszak van, add meg az elsőt.)

```
summerVacation :: [Forecast] -> [Weather]

summerVacation [Prediction Sunny 60, Prediction Cloudy 70] ==
  [Sunny,Cloudy]
summerVacation [Prediction Rainy 30, Prediction Sunny 50,
  Prediction Cloudy 65] == [Sunny,Cloudy]
summerVacation [Prediction Rainy 10, Prediction Sunny 49,
  Prediction Cloudy 57, Prediction Rainy 10, Prediction Sunny 49] ==
  [Sunny,Cloudy]
summerVacation [Prediction Rainy 10, Prediction Sunny 49,
  Prediction Cloudy 57, Prediction Rainy 10, Prediction Cloudy 57,
  Prediction Sunny 49] == [Sunny,Cloudy]
summerVacation [Prediction Rainy 10, Prediction Sunny 49,
  Prediction Cloudy 57, Prediction Rainy 10, Prediction Cloudy 57,
  Prediction Sunny 49, Prediction Sunny 49] == [Cloudy,Sunny,Sunny]
summerVacation [Prediction Sunny 90, Prediction Sunny 49,
  Prediction Cloudy 57, Prediction Rainy 10, Prediction Cloudy 57,
  Prediction Sunny 49, Prediction Sunny 49] == [Sunny,Sunny,Cloudy]
summerVacation [Prediction Rainy 10, Prediction Sunny 49,
  Prediction Cloudy 57, Prediction Rainy 10, Prediction Rainy 60,
  Prediction Rainy 40, Prediction Cloudy 57, Prediction Sunny 49,
  Prediction Sunny 49] == [Cloudy,Sunny,Sunny]
summerVacation [Prediction Rainy 10, Prediction Sunny 49,
  Prediction Rainy 60, Prediction Cloudy 57, Prediction Rainy 10,
  Prediction Rainy 60, Prediction Rainy 40, Prediction Cloudy 57,
  Prediction Sunny 49, Prediction Sunny 49, Prediction Rainy 60,
```

```
  Prediction Rainy 76, Prediction Cloudy 60] == [Cloudy,Sunny,Sunny]
```

n-edik legkisebb elem (3 pont)

Adott egy monoton növekvő sorozat! Definiáljuk azt a függvényt, amely megadja egy ilyen lista `n`-edik legkisebb elemét! A függvény hagyja figyelmen kívül a multiplicitást, azaz, ha egy elem többször is szerepel, azt csak egyszer vegye figyelembe. Amennyiben az eredmény meghatározható, akkor azt `Just` konstruktorba csomagolva adja meg, ellenkező esetben az eredmény `Nothing` legyen. Az indexelés kezdődjön `1`-től.

```
nthSmallest :: (Integral a, Eq b) => a -> [b] -> Maybe b

nthSmallest 10 [] == Nothing
nthSmallest  0 [] == Nothing
nthSmallest 2 [1,2,4,5,5] == Just 2
nthSmallest 4 [0,1,1,2,2,3,4,5,5] == Just 3
nthSmallest 2 [1,1,1,1,4,5,5] == Just 4
nthSmallest (-2) [1,1,1,1,4,5,5] == Nothing
nthSmallest 6 [1,1,1,1,4,5,5] == Nothing
nthSmallest 2  (concat [ replicate i i | i <- [1..]]) == Just 2
nthSmallest 10 (concat [ replicate i i | i <- [1..]]) == Just 10
nthSmallest 10 (concat [ replicate i (2*i+1) | i <- [1..]]) == Just 21
```

Szöveg kiemelése (3 pont)

Definiáld a `highlight` függvényt, ami egy szövegben nagybetűssé alakítja egy másik szöveg minden előfordulását. Előfordulhat, hogy a keresett előfordulások átfednek egymással, pl.: az `"almalma"` szövegben kétszer is előfordul az `"alma"`, ekkor mindkét előfordulást ki kell emelni. Mindkét paraméter csak az angol ábécé kisbetűit tartalmazza és az első paraméterként kapott szövegről feltehető, hogy véges.

```
highlight :: String -> String -> String

highlight "alma" "almafa" == "ALMAfa"
highlight "alma" "almafalma" == "ALMAfALMA"
highlight [] "alma" == "alma"
highlight [] [] == []
highlight "alma" [] == []
highlight "alma" "almalma" == "ALMALMA"
highlight "hih" "hihihi" == "HIHIHi"
take 10 (highlight "alma" ('a' : (concat  (repeat "lma")))) == "ALMALMALMA"
take 15 (highlight "kell" (concat  (repeat "haskell"))) == "hasKELLhasKELLh"
```

k-durva számok (3 pont)

A számelmélet területén a *k-durva* számok olyan pozitív egész számok, melyek prímtényezői nem kisebbek `k`-nál.

Írjunk egy `roughNumber` függvényt, amely kap egy `k` értéket, és egy számot és eldönti, hogy a kapott szám *k-durva*-e.

Magyarázat: A `385` az 3-durva, mivel prímtényezői felírhatóak `[5,7,11]` alakban és mindegyikre igaz, hogy nem kisebbek, mint `3`. Ugyanakkor a `385` nem 7-durva, mert az `5` kisebb, mint a `7`.

```
roughNumber :: Int -> Int -> Bool

roughNumber 3 385
not $ roughNumber 7 385
not $ roughNumber 5 8
roughNumber 3 5
roughNumber 3 7
roughNumber 3 9
roughNumber 13 13
roughNumber 13 41
roughNumber 2 7
not $ roughNumber 7 25
and $ take 500 [roughNumber 3 n | n <-[3,5..]]
and $ take 500 [roughNumber 5 n | n <-[2..], (n `mod` 6) == 1 || (n `mod` 6) == 5]
and $ take 500 [roughNumber 2 n | n <-[2..]]
```