```
module LKSA__2017_11_08 where
import Data.List

-------------------------------------
-- MURGILKETA
-------------------------------------
--1

lehenengoa_aurrekoetan_ez_lag :: [[Integer]] -> [Integer] -> [[Integer]]
lehenengoa_aurrekoetan_ez_lag s r
        | [] `elem` s              = error "Lehenengo zerrendako osagaien artean zerrenda hutsa agertzen da."
        | null s                   = s
        | (head (head s)) `elem` r = lehenengoa_aurrekoetan_ez_lag (tail s) (r ++ (head s))
        | otherwise                = (head s):lehenengoa_aurrekoetan_ez_lag (tail s) (r ++ (head s))

{-Konputazio-kostua: n = genericLength s, m = genericLength (concat s) eta p = genericLength r izanda,
   - [] zerrenda s zerrenda al dagoen begiratu behar da dei errekurtsibo bakoitzean:
     n + (n-1) + (n-2) + ... + 1 = (n*(n + 1))/2 urrats.
   - s hutsa al den aztertzea dei errekurtsibo bakoitzean egin beharko da: n urrats.
   - Kasu induktiboan r zeharkatu behar da. Baina r zerrendan s-ko zerrendak sartuz
     joango direnez, goiko muga edo kasu okerrena m + p elementu n aldiz aztertzea izango
     litzateke: (m + p)*n.
   - Guztira: ((n*(n + 1))/2) + n + ((m + p)*n)
   - q= handiena(n + 1,m + p) izanda, kostua O(q*q) izango litzatekeela esan dezakegu. Beraz, polinomiala da.
-}

-------------------------------------
lehenengoa_aurrekoetan_ez :: [[Integer]] -> [[Integer]]
lehenengoa_aurrekoetan_ez s
        | [] `elem` s          = error "Zerrenda horretako osagaien artean zerrenda hutsa agertzen da."
        | null s               = s
        | otherwise            = lehenengoa_aurrekoetan_ez_lag s []

{-Konputazio-kostua: n = genericLength s eta m = genericLength (concat s) izanda,
   - [] zerrenda s zerrenda al dagoen begiratu behar da behin: n urrats.
   - s hutsa al den aztertu behar da behin: urrats bat.
   - lehenengoa_aurrekoetan_ez_lag s [] deiaren kostua behin. Kasu okerrena m elementu n aldiz aztertzea izango
     litzateke: m * n.
   - Guztira: n + 1 + (m * n)
   - q= handiena(n,m) izanda, kostua O(q*q) izango litzatekeela esan dezakegu. Beraz, konputazio-kostua polinomiala da.
-}
-------------------------------------

-------------------------------------
```

```
--BUKAERAKO ERREKURTSIBITATEA
-------------------------------------
--2

posiziokoa_ezabatu :: Integer -> [Integer] -> [Integer]
posiziokoa_ezabatu n r
        | (n <= 0) || (n > (genericLength r))   = error "Posizioa ez da egokia."
        | otherwise                             = (genericTake (n - 1) r) ++ (genericDrop n r)

{-Konputazio-kostua: m = genericLength r izanda,
   - r behin zeharkatu behar da bere luzera kalkulatzeko: m urrats.
   - n urrats genericTake kalkulatzeko eta n urrats genericDrop kalkulatzeko: 2 * n.
   - (genericTake (n - 1) r) eta (genericDrop n r)  zerrendak elkartzeko m urrats.
   - Guztira: m + 2*n + m.
   - q = handiena(n,m) izanda, kostua O(q) izango da. Beraz, konputazio-kostua lineala da.
-}
-------------------------------------

posiziokoa_ezabatu_lag :: Integer -> [Integer] -> Integer -> [Integer] -> [Integer]
posiziokoa_ezabatu_lag n r z l
        | ((n - z) <= 0) || ((n - z) > (genericLength r))       = error "1. eta 3. parametroen arteko kendurak adierazten
duen posizioa ez da egokia."
        | (n - z) == 1                                          = l ++ (tail r)
        | otherwise                                             = posiziokoa_ezabatu_lag n (tail r) (z + 1) (l ++ [head
r])

{-Konputazio-kostua: m = genericLength r eta p = genericLength l izanda,
   - dei errekurtsibo bakoitzean r behin zeharkatu behar da bere luzera kalkulatzeko. Kasu okerrenean, hau da,
     ezabatu beharrekoa azkeneko posiziokoa denean: m + (m - 1) + (m - 2) + ... + 1 urrats.
     Beraz, (m*(m+1))/2 urrats kasu okerrenean.
   - (n - z) == 1 betetzen denean, l eta (tail r) zerrendak elkartzeko, kasu okerrenean m + p urrats.
   - Kasu induktiboan edo errekurtsiboan, l eta [head r] elkartzeko p + (p+1) + (p+2) + ... + (p + m) urrats: (m*p) +
(m*(m+1)/2) urrats.
   - Guztira: (m*(m+1))/2 + (m + p) + (m*p) + (m*(m+1)/2).
   - q = handiena(m,p) izanda, kostua O(q^q) izango da. Beraz, konputazio-kostua polinomiala da.
-}
-------------------------------------
posiziokoa_ezabatu_be :: Integer -> [Integer] -> [Integer]
posiziokoa_ezabatu_be n r
        | (n <= 0) || (n > (genericLength r))   = error "Posizioa ez da egokia."
        | otherwise                             = posiziokoa_ezabatu_lag n r 0 []

{-Konputazio-kostua: m = genericLength r izanda,
   - r behin zeharkatu behar da bere luzera kalkulatzeko: m urrats.
```

```
   - posiziokoa_ezabatu_lag n r 0 [] deiaren kostua O(m*m)
   - Guztira: m + (m*m).
   - Beraz, kostua O(m*m) izango da. Ondorioz, konputazio-kostua polinomiala da.
-}

-------------------------------------
--ZERRENDA-ERAKETA
-------------------------------------
--3.1

luzerak :: [[Integer]] -> [Integer]
luzerak s = [genericLength y | y <- s]

{-Konputazio-kostua: m = genericLength s eta n = genericLength (concat s) izanda,
   - s zerrenda behin zeharkatu behar da: m urrats.
   - s zerrendako zerrenda bakoitza behin zeharkatu behar da: guztira n elementu, hau da, n urrats.
   - q = handiena(m,n) izanda, kostua O(q) izango litzatekeela esan dezakegu. Beraz, konputazio-kostua lineala da.
-}
-------------------------------------
--3.2

guztiak_berdinak :: [Integer] -> Bool
guztiak_berdinak s
        | (null s) = True
        | (genericLength s) == (genericLength [y | y <- s, y == (head s)])  = True
        | otherwise = False

{-Konputazio-kostua: m = genericLength s izanda,
   - s zerrenda hiru aldiz zeharkatuko da: 3*m urrats.
   - Beraz, konputazio-kostua O(m). Ondorioz, lineala da.
-}
-------------------------------------
--3.3

posiziokoak_hautatu :: Integer -> [[Integer]] -> [Integer]
posiziokoak_hautatu n s
        | n <= 0        = error "Posizioa ez da positiboa."
        | (genericLength s) > (genericLength [y | y <- (luzerak s), y >= n]) = error "Zerrenden luzerak ez dira egokiak."
        | otherwise  = [head (genericDrop (n-1) y) | y <- s]

{-Konputazio-kostua: m = genericLength s, p = genericLength (concat s) eta q = handiena(m,p) izanda,
   - Bigarren errore-kasua aztertzeko, genericLength bakoitza m urrats eta luzerak s kalkulatzeko q urrats: 2*m + q
urrats.
     Kasu okerrenean 3*q urrats.
```

```
   - otherwise kasuan: q urrats.
   - Beraz 4*q urrats guztira: O(q). Ondorioz, konputazio-kostua lineala da.
-}

posiziokoak_hautatu_2 :: Integer -> [[Integer]] -> [Integer]
posiziokoak_hautatu_2 n s
        | n <= 0        = error "Posizioa ez da positiboa."
        | not (null [y | y <- (luzerak s), y < n]) = error "Zerrenden luzerak ez dira egokiak."
        | otherwise  = [head (genericDrop (n-1) y) | y <- s]

{-Konputazio-kostua: m = genericLength s, p = genericLength (concat s) eta q = handiena(m,p) izanda,
   - Bigarren errore-kasua aztertzeko, zerrenda berria kalkulatzea m urrats eta luzerak s kalkulatzeko q urrats: m + q
urrats.
     Kasu okerrenean 2*q urrats.
   - otherwise kasuan: q urrats.
   - Beraz, 3*q urrats guztira: O(q). Ondorioz, konputazio-kostua lineala da.
-}
-------------------------------------
--3.4

zutabeak :: [[Integer]] -> [[Integer]]
zutabeak s
        | not (guztiak_berdinak (luzerak s)) = error "Zerrendak ez dira luzera berekoak."
        | null s = s
        | null (head s) = s
        | otherwise = [posiziokoak_hautatu y s | y <- [1..(genericLength (head s))]]

{-Konputazio-kostua: m = genericLength s, p = genericLength (concat s), q = handiena(m,p) eta
     r = genericLength (head s) izanda,
   - Errore-kasua aztertzeko, guztiak_berdinak kalkulatzea O(m) eta luzerak s kalkulatzea O(q). Beraz, O(q).
   - Bigarren eta hirugarren kasuak (null ...) urrats bakarrekoak dira.
   - otherwise kasuan: posiziokoak_hautatu 3*q eta genericLength (head s) kalkulatzea eta zeharkatzea 2*r.
     Gainera, r < q denez, kasu okerrena bezala 2*q har dezakegu. Beraz, 5*q.
   - Guztira: O(q) + 5*q, hau da, O(q). Ondorioz, konputazio-kostua lineala da.
-}

-------------------------------------
--3.5

posi_bider_batu :: [Integer] -> [Integer] -> Integer
posi_bider_batu r s
        | (genericLength r) /= (genericLength s) = error "Luzera desberdinak."
        | null r = error "Zerrenda hutsak dira."
        | otherwise = sum [x*y | (x,y) <- (zip r s)]
```

```
{-Konputazio-kostua: m = genericLength s, p = genericLength r, q = handiena(m,p) eta
     r = genericLength (head s) izanda,
   - Errore-kasua aztertzeko, m + p urrats. Kasu okerrenean 2*q. Beraz, O(q).
   - Bigarren kasua (null ...) urrats bakarrekoa da.
   - otherwise kasuan: zip r s zerrenda kalkulatzea q urrats, zerrenda hori zeharkatzea
     q urrats eta sum kalkulatzea beste q urrats. Beraz, 3*q urrats. Ondorioz, O(q).posiziokoak_hautatu 3*q eta
genericLength (head s) kalkulatzea eta zeharkatzea 2*r.
     Gainera, r < q denez, kasu okerrena bezala 2*q har dezakegu. Beraz, 5*q.
   - Guztira: O(q) + O(q) = O(q). Ondorioz, konputazio-kostua lineala da.
-}

-------------------------------------
--3.6

errenkada_kalkulatu :: [Integer] -> [[Integer]] -> [Integer]
errenkada_kalkulatu r s
        | not(guztiak_berdinak (luzerak (r:s))) = error "Luzera desberdineko zerrendak."
        | (null r) || (null s) = error "Bietako bat hutsa da."
        | otherwise = [posi_bider_batu r y | y <- s]

{-Konputazio-kostua: m1 = genericLength s, m2 = genericLength (concat s), q = handiena(m1,m2),
     p = genericLength r eta w = handiena(q, p) izanda,
   - Lehenengo errore-kasua aztertzeko, guztiak_berdinak kalkulatzea O(m1 + 1) eta
     luzerak (r:s) kalkulatzea O(m2 + p).
     Beraz, O(w).
   - Bigarren errore-kasua (null ...) bi urratsekoa da, konstantea beraz.
   - otherwise kasuan: s zeharkatzea O(m1); posi_bider_batu O(handiena(m2, p)).
     Beraz, O(2*w) = O(w).
   - Guztira: O(w). Ondorioz, konputazio-kostua lineala da.
-}


-------------------------------------

--3.7

matrize_biderketa :: [[Integer]] -> [[Integer]] -> [[Integer]]
matrize_biderketa r s
        | not (guztiak_berdinak (luzerak r))  = error "1. zerrendan luzera desberdineko zerrendak daude."
        | not (guztiak_berdinak (luzerak s))  = error "2. zerrendan luzera desberdineko zerrendak daude."
        | (null r) || (null s) = error "Zerrendetakoren bat hutsa da."
        | (null (head r)) || (null (head s)) = error "Zerrenda hutsez osatutako zerrenda."
        | (genericLength (head r)) /= (genericLength s) = error "1. zerrendako zerrenden luzera eta 2. zerrendako
```

```
zerrenda-kopurua desberdinak dira."
        | otherwise = [errenkada_kalkulatu y (zutabeak s) | y <- r]

{-Konputazio-kostua: m1 = genericLength s, m2 = genericLength (concat s), q1 = handiena(m1,m2),
   p1 = genericLength r, p2 = genericLength (concat r) eta q2 = handiena(p1,p2) izanda,
   - Lehenengo errore-kasua aztertzeko, guztiak_berdinak kalkulatzea O(p2) eta luzerak r kalkulatzea O(q2).
     Beraz, O(q2).
   - Bigarren errore-kasua aztertzeko, guztiak_berdinak kalkulatzea O(m2) eta luzerak s kalkulatzea O(q1).
     Beraz, O(q1).
   - Hirugarren eta laugarren kasuak (null ...) urrats bakarrekoak dira.
   - Bosgarren errore-kasua aztertzeko, genericLength (head r) kalkulatzeko urrats-kopurua p2
     baino txikiagoa da, hau da, q2 baino txikiagoa. Bestalde, genericLength s kalkulatzeko urrats-kopurua
     m1 da. Beraz, q1 baino txikiagoa. Guztira O(q1+q2).
   - otherwise kasuan: r zeharkatzea O(p1); zutabeak s kalkulatzea O(q1) da; p1 aldiz egin behar da: O(q1*q2);
     errenkada_kalkulatu O(handiena(q1,q2)); errenkada_kalkulatu kasu txarrenean p2 aldiz egin begar da;
     Beraz, O(handiena(q1,q2)*handiena(q1,q2)).
   - Ondorioz, konputazio-kostua polinomiala da.
-}

---
--------------------------------------
```