1. HS 2 - 235

```
:load Uloha235.hs
```

```
najmensi :: Int -> Integer
najmensi 0 = undefined
najmensi n =  sort (1 : [i ^ j | j <- [1..n], i <- [2, 3, 5]]) !! (n-1)
```

Snažil som sa nejako obmedziť pamäťovú zložitosť v tejto úlohe, ale neprišiel som sa vhodnejší algoritmus. Vždy sa takto vygeneruje $n \times 3 + 1$ čísel a potom v **usporiadanom** zozname vyberieme n-tý prvok.

Snažil som sa nejako osekať počet vygenerovaných mocnín $2, 3, 5$, no kvôli faktu, že pri vygenerovaní trojice s mocninou i môžu vzniknúť niektoré čísla menšie ako pri generovaní čísiel s mocninou $i-1$, mi nejaké stabilné riešenie unikalo. Nie som si úplne istý správnosťou, tak radšej spravím o úlohu viac ďalej.

```
*Uloha235> najmensi 1000
998959536101117514042111113533813 2...
(0.03 secs, 10,660,528 bytes)

*Uloha235> najmensi 10000
140714566777322864958486360136303 0...
(0.71 secs, 220,744,584 bytes)
```

2. HS2 - Binárny vyhľadávací strom - chýbajúci delete

```
:load BVS.hs
```

Keďže máme funkciu `insert` pripravenú, využil som to na prípravu nejakých testovacích stromov (a našiel krásne využitie `foldl` na vytváranie stromov). Pre vernú kópiu treba zadať prvky v poradí po úrovniach.

```
build :: (Ord t) => [t] -> BVS t
build [] = Nil
build xs = foldl (\accTree -> \x -> insert x accTree) Nil xs

-- https://static.javatpoint.com/ds/images/binary-search-tree.png
*BVS> build [30, 15, 60, 7, 22, 45, 75, 17, 27]
-Nod (Nod (Nod Nil 7 Nil) 15 (Nod (Nod Nil 17 Nil) 22 (Nod Nil 27 Nil))) -- left sub
30 -- root
-(Nod (Nod Nil 45 Nil) 60 (Nod Nil 75 Nil)) -- right sub

*BVS> build "python"
Nod (Nod Nil 'h' (Nod (Nod Nil 'n' Nil) 'o' Nil)) 'p' (Nod (Nod Nil 't' Nil) 'y' Nil)
```

Teraz k samotnému `delete`. Bez nejakej pokročilej správy vybalancovania, zvoľme jednoduchú taktiku. Po nájdení vrcholu s prvkom na vyhadzov:

- ak je listom, len ho zmažme (nahraďme `Nil`)
- ak nemá ľavého syna, nahraďme ho pravým synom
- ak nemá pravého syna, nahraďme ho ľavým synom
- inak ak ide o vnútorný vrchol, nahraďme tento vrchol hodnotou najmenšieho prvku v pravom podstrome a zároveň tento najmenší prvok zmažme.

Ako navrhuje zadanie, treba pripraviť funkciu `minBVS`, ktorá nájde minimálny prvok v strome rešpektujúc polymorfizmus. To nie je v prípade `BVS` vôbec zložité, je to totiž 'najľavejší' prvok stromu.

```
minBVS :: (Ord t) => BVS t -> t
minBVS Nil = error "prazdny nema minimum"
minBVS (Nod Nil val _) = val
minBVS (Nod left val right) = minBVS left
```

Môj kód na vymazanie prvku z `BVS` tak má nasledovnú podobu, pričom funkcia `decide` slúži ako nejaký switch toho, čo má `delete` urobiť v závislosti od vyššie popísaných scenárov.

```
delete  :: (Ord t) => t -> BVS t -> BVS t
delete  _ Nil = Nil
delete  x (Nod left val right)  | x < val = Nod (delete x left) val right
                                | x > val = Nod left val (delete x right)
                                | x == val = decide (Nod left val right)

decide :: (Ord t) => BVS t -> BVS t
decide (Nod Nil val Nil)   = Nil
decide (Nod left val Nil)  = left
decide (Nod Nil val right) = right
decide (Nod left val right) = Nod left (minBVS right) (delete (minBVS right) right)
```

Niekoľko ručných testov ukázalo, že by to malo fungovať korektne:

```
*BVS> delete 'p' $ build "python"
Nod (Nod Nil 'h' (Nod (Nod Nil 'n' Nil) 'o' Nil)) 't' (Nod Nil 'y' Nil)

*BVS> bvs = build [30, 15, 60, 7, 22, 45, 75, 17, 27]

*BVS> delete15 = delete 15 bvs

*BVS> delete15
Nod (Nod (Nod Nil 7 Nil) 17 (Nod Nil 22 (Nod Nil 27 Nil))) 30 (Nod (Nod Nil 45 Nil) 60 (

*BVS> delete30 = delete 30 delete15

*BVS> delete30
Nod (Nod (Nod Nil 7 Nil) 17 (Nod Nil 22 (Nod Nil 27 Nil))) 45 (Nod Nil 60 (Nod Nil 75 Ni
```

Nechce sa mi to teraz krajšie naformátovať, výpisy v tomto latexe utekajú do stratena, ale pointa je aj tak v ľavom podstrome.

3. HS 2 - Determinant

```
:load Determinant.hs

type Vector = [Int]
type Matrix = [[Int]]

val :: Matrix -> Int -> Int -> Int
val m row col = m !! row !! col

getRowWithoutColumn :: Matrix -> Int -> Int -> Vector
getRowWithoutColumn m row col = [val m row i | i <- [0..length m - 1], i /= col]

coMatrix :: Matrix -> Int -> Int -> Matrix
coMatrix m row col = [getRowWithoutColumn m i col | i <- [0..length m - 1], i /= row]

det :: Matrix -> Int
det m | n == 1 = val m 0 0
      | n == 2 =  val m 0 0  * val m 1 1 - val m 0 1 * val m 1 0
      | otherwise = sum [val m 0 col * (-1)^col * det (coMatrix m 0 col) | col <- c]
       where n = length m
             c = [0..n - 1]
```

S týmto algoritmom sme sa myslím, pohrali viac než dosť na predchádzajúcich úlohách, takže jedinou podstatnou úlohou bolo dostať tento algoritmus do `Haskell` syntaxe. To nebolo nakoniec nejako zložité a po pár chybových hláškach to myslím, že funguje. Jediný rozdiel medzi týmto a nejakým mojím `Go` riešením je, že miesto počítadla a cyklu, kde počítame determinanty podmatíc, si tieto determinanty ukladám do zoznamu a z neho vypočítame výslednu sumu.

```
*Determinant> det [[5]]
5
*Determinant> det [[3, 4], [2, -5]]
-23
*Determinant> det [[1, -6, 5], [2, 2, 5], [-1, -4, 1]]
34
*Determinant> det [[1, 2, 3, 4], [-2, 1, -4, 3], [3, -4, -1, 2], [4, 3, -2, -1]]
900
*Determinant> det [[0, 6, -2, -1, 5], [0, 0, 0, -9, -7],
                   [0, 15, 35, 0, 0], [0, -1, -11, -2, 1],
                   [-2, -2, 3, 0, -2]]
2480
```

4. HS 2 - Kruh

```
:load Kruh.hs
```

```haskell
kruh :: [[Int]]
kruh = backtrack [1..10]

backtrack :: [Int] -> [[Int]]
backtrack [] = [[]]
backtrack nums = [n:rem | n <- nums, rem <- backtrack (nums \\ [n]), check360 (n:rem)]

check :: [Int] -> Bool
check [] = True
check [_] = True
check (a:b:xs) = all (\i -> (a+b) `mod` i /= 0) [3, 5, 7] && check (b:xs)

check360 :: [Int] -> Bool
check360 xs | n == 10 = all (\i -> s `mod` i /= 0) [3, 5, 7] && check xs
            | otherwise = check xs
            where s = last xs + head xs
                  n = length xs
```

V tejto úlohe som išiel podľa prednášky, teda podľa vzoru magického backtracku, kde sme hľadali magické čísla z cifier `[1..10]`. Pomocou backtrackingu teda skúšame pridávať k jednotlivým cifrám zvyšné, podľa zadaných podmienok. Na kontrolu týchto podmienok nám treba nejakú, očividne kontrolnú funkciu, ktorú som nazval `check` a nerobí nič iné, ako rekurzívne skontroluje každú dvojicu zoznamu na fakt, že ich súčet nie je deliteľný ani jedným číslom z `[3,5,7]`.

Program mi začal pumpovať riešenia a po pocite víťazstva som sa pozrel na riešenie ako napr. `[1,3,5,8,9,2,6,10,7,4]`, ručne to prekontroloval a zistil, že som zabudol na pointu úlohy - kruh, t.j. nekontroloval som korektný súčet poslednej a prvej cifry. To vyriešilo pridanie ďalšej kontrolnej funkcie, ktorá pri zdanlivo úplnom riešení (`n == 10`) ešte prekontroluje túto dvojicu posledného a prvého prvku.

```
*Kruh> kruh
[[1,3,8,5,6,2,9,4,7,10],[1,7,4,9,2,6,5,8,3,10],[1,10,3,8,5,6,2,9,4,7], ...]
*Kruh> length kruh
40
```

5. HS 2 - EU OS

```
:load EuOS.hs

children :: Int -> [[Int]] -> [Int]
children n xs = [x !! 1 | x <- xs, head x == n]

graph :: [[Int]] -> [[Int]]
graph xs = nub [n:children n xs | n <- [head rel | rel <- xs]]

processesToKill :: [[Int]] -> [Int]
processesToKill xs = nub [x | x <- b, x `notElem` a]
                where a = [head rel | rel <- xs]
                      b = concat [tail rel | rel <- xs]

kill :: [[Int]] -> [Int]
kill [] = []
kill xs = concat (filter (\e -> length e == 1) rem) ++ next
        where rem = [x \\ processesToKill xs | x <- xs]
              next = kill (filter (\e -> length e > 1) rem)

odblokuj :: [[Int]] -> [Int]
odblokuj xs = processesToKill (graph xs) ++ kill (graph xs)
```

Trošku som to asi skomplikoval, ale ako taktika to podľa mňa funguje a nič iné mi momentálne nenapadá. Pointa je taká, že sa najprv vytvorí zoznam zoznamov `graph`, kde sa zo vstupného zoznamu vytvorí akýsi graf závislostí, t.j. zoznamy tvaru `[a, x1, ..., xn]`, kde `a` je nejaký proces a `x1, ..., xn` sú procesy, od ktorých `a` závisí.

Metóda `processesToKill` nájde proces, ktorý nie je závislý na žiadnom inom (t.j. nenachádza sa v zozname prvých prvkov). Potom rekurzívna funkcia `kill` zabije tento proces, ktorý bol nezávislý od iných, čím uvoľnila ďalšie procesy (z ntíc v grafe sa odstráni tento proces). Tieto zabité procesy zreťazuje do výsledného zoznamu poradia zabitých procesov. Toto opakuje na ďalšie uvoľnené procesy až kým nezabije všetky.

```
*EuOS> odblokuj [[2,9],[6,8],[5,1],[1,6],[4,5],[3,7],[5,3],[7,8],[8,9]]
[9,2,8,6,7,1,3,5,4]

*EuOS> odblokuj [[2,9],[6,8],[5,1],[1,6],[4,5],[3,7],[5,3],[7,8],[8,9],[1,3]]
[9,2,8,6,7,3,1,5,4]
```

```
*EuOS> odblokuj [[10, 4], [8, 3], [3, 14], [2, 7], [4, 5],
                 [7, 12], [5, 12], [1, 10], [5, 3], [1, 2], [12, 8]]
[14,3,8,12,7,5,2,4,10,1]
```

Malo by to riešiť správne aj viacnásobné závislosti, čo vidno na druhom a treťom testovacom vstupe vyššie. Ako tak čítam po sebe, tak neviem či by som to sám pochopil, ale už nestíham písať zrozumitelnejšie, tak ak bude treba dovysvetliť, pokojne mi napíš.

6. HS 2 - Rodinky

- nerobil som

7. HS3 - Magický štvorec

```
:load Magicky.hs

-- https://github.com/paradigmy/Kod/blob/master/PRO7/haskell2.hs

transponuj :: [[Int]] -> [[Int]]
transponuj [] = []
transponuj ([]:xss) = transponuj xss
transponuj ((x:xs):xss) = (x:(map head xss)):(transponuj $ xs:(map tail xss))

magickyStvorec :: [[Int]] -> Bool
magickyStvorec xs = all (== head total) (tail total)
                where n = length xs
                      rows = [sum (xs !! row) | row <- [0..n-1]]
                      cols = [sum (transponuj xs !! row) | row <- [0..n-1]]
                      diag1 = [sum [xs !! row !! row | row <- [0..n-1]]]
                      diag2 = [sum [xs !! row !! (n-1-row) | row <- [0..n-1]]]
                      total = rows ++ cols ++ diag1 ++ diag2
```

Taktika v tejto úlohe je pomerne priamočiara, skontrolujem súčty prvkov v riadku, stĺpci, jednej aj druhej diagonále a tieto súčty na konci spojím do jedného zoznamu. Na tento zoznam aplikujem funkciu `all`, ktorá zistí, či sú všetky prvky v tomto výslednom zozname rovnaké, a tým pádom ide o magický štvorec. Pre kontrolu stĺpcov sa zišlo transponovať maticu, tak som si z prednášky vypožičal danú funkciu.

```
*Magicky> magickyStvorec [[16,3,2,13],[5,10,11,8],[9,6,7,12],[4,15,14,1]]
True

*Magicky> magickyStvorec [[16,3,2,13],[5,10,11,8],[9,6,7,12],[4,15,14,2]]
False -- zmeneny posledny prvok
```