Контролна работа № 1 по Функционално програмиране
Специалност „Информационни системи“, I курс, 08.04.2023 г.

Задача 1

Дадени са `numBottles` бутилки за вода, които първоначално са пълни. Всеки `numExchange` празни бутилки могат да бъдат заменени с една пълна бутилка. Ако пълна бутилка бъде изпита, тя става празна.

Да се дефинира функция **`numDrink :: Int -> Int -> Int`**, която приема две естествени числа `numBottles` и `numExchange` и намира максималния брой бутилки вода, които могат да бъдат изпити. **Да се реализира линеен рекурсивен процес!**

Примери:

```
numDrink 9 3   → 13
numDrink 15 4  → 19
numDrink 761 3 → 1141
```

Задача 2

Разглеждаме числата от редицата на Фибоначи (индексирана от 0): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
Вашата задача е:

- да генерирате `n`-тото число от редицата на Фибоначи **посредством линеен итеративен процес**;
- да разбиете полученото число на подгрупи с дължина `k` (групата, образувана от най-младшите (най-десните) цифри, може да е с дължина, по-голяма от `k`). Възможно е група да започва с 0;
- за всяка от подгрупите да намерите и върнете *максималната двойка*. Максималната двойка за дадена подгрупа е двуелементен вектор с най-често срещаната цифра (представена като символ) и броя на срещанията ѝ в подгрупата. Ако няколко цифри се срещат най-често, да се върне най-малката.

Да се дефинира функция от по-висок ред **`aroundFib :: Integer -> (Int -> [(Char, Int)])`**, която приема неотрицателното цяло число `n` и връща унарна функция на естественото число `k`, която връща списък от максималните двойки за всяка подгрупа с дължина `k` на `n`-тото число на Фибоначи.

Примери:

```
(aroundFib 100) 25 → [('1',3)]
(aroundFib 180) 25 → [('1',5),('7',3)]
(aroundFib 1700) 25 →
[('1',4),('2',5),('0',6),('4',5),('5',7),('2',4),('6',7),('3',5),
 ('0',4),('8',5),('4',5),('4',4),('7',7),('7',6),('2',2)]
(aroundFib 500) 42 → [('0',6),('2',7),('2',6)]
(aroundFib 6000) 242 →
[('5',31),('8',33),('8',31),('7',35),('7',31),('4',7)]
```

Задача 3

Родител иска да даде по една бисквитка на всяко от децата си. Всяко дете `i` има изискване `g[i]`, което е минималният размер на бисквитката, с която то ще се задоволи. Всяка бисквитка `j` има размер `s[j]`. Ако `s[j] >= g[i]`, можем да дадем бисквитката `j` на детето `i` и то ще бъде доволно.

Да се дефинира функция **`numContentChildren :: [Int] -> [Int] -> Int`**, която приема списък `gs` с изискванията на децата и списък `ss` с размерите на бисквитките и намира максималния брой на децата, които ще бъдат доволни.

Примери:

```
numContentChildren [1, 2, 3] [1, 1] → 1
numContentChildren [1, 2] [1, 2, 3] → 2
```

Задача 4

Нека списък **B** от префиксни суми на списък **A** се дефинира като:

```
B[0] = A[0]
B[1] = A[0] + A[1]
B[2] = A[0] + A[1] + A[2]
...
B[n-1] = A[0] + A[1] + ... + A[n-1]
```

Нека списък **B** от суфиксни суми на списък **A** се дефинира като:

```
B[0] = A[0] + A[1] + A[2] + ... + A[n-1]
B[1] = A[1] + A[2] + ... + A[n-1]
...
B[n-2] = A[n-2] + A[n-1]
B[n-1] = A[n-1]
```

Да се дефинира функция **`prefixToSuffix :: (Num a) => [a] -> [a]`**, която приема списък от префиксни суми `xs` и връща списък от суфиксните суми на списъка, от който е образуван `xs`.

Примери:

```
prefixToSuffix [1, 3, 6, 10, 15]        → [15, 14, 12, 9, 5]
prefixToSuffix [1, 3, 6, 10, 15]        → [15, 14, 12, 9, 5]
prefixToSuffix [0]                      → [0]
prefixToSuffix [-1, -2, -3, -4, -5]     → [-5, -4, -3, -2, -1]
prefixToSuffix [1, -4, 2, 90, 100, -1]  → [-1, -2, 3, -3, -91, -101]
prefixToSuffix [1, 0, 1, 0, 1, 0, 1, 0] →
[0, -1, 0, -1, 0, -1, 0, -1]
prefixToSuffix [0, 0, 0, 0, 0, 0, 0, 0, 0, 1] → [1, 1, 1, 1, 1, 1, 1,
1, 1, 1]
prefixToSuffix [0, 0, 0, 0, 0, 0, 1, 1, 1, 1] → [1, 1, 1, 1, 1, 1, 1,
0, 0, 0]
```