Второ контролно по Функционално програмиране
специалност ИС, 05.06.2021 г.

Задача 1. Да се дефинира функция `getIndices :: [Int] -> (Int -> (Int, Int))` , която получава един аргумент – списък от цели числа **xs** и връща анонимна функция на един аргумент – цяло число **n**. Върнатата функция трябва да намира двойка от индексите на числа от списъка, чиято сума е равна на **n**. Предполага се, че такива индекси винаги съществуват. Да се връща първата намерена двойка.

Примери:

```
(getIndices [2, 7, 11, 15]) 9 → (0,1) -- 2 + 7 = 9
(getIndices [3, 2, 4]) 6      → (1,2)
(getIndices [3, 3]) 6         → (0,1)
```

Задача 2. Стандартните списъци в Haskell са хомогенни, т.е. съдържат елементи от един и същ тип. Нека дефинираме наш полиморфен алгебричен тип „вложен списък“ **NestedList a**, който може да съдържа както „обикновени“ („атомарни“) елементи, така и други списъци от типа **NestedList a**.

```
data NestedList a = Elem a | List [NestedList a]
```

Да се дефинира функция `mapNested :: (a -> b) -> (NestedList a) -> (NestedList b)`, която получава вложен списък **list** и прилага подадена функция към всеки един от елементите му.

Примери:

```
mapNested (*3) (List [Elem 1, List [Elem 2, List [Elem 3, Elem 4], Elem 5]]) →
List [Elem 3,List [Elem 6,List [Elem 9,Elem 12],Elem 15]]
mapNested (take 2 . show) (List [Elem 15, List [Elem 200, List [Elem 351.52,
Elem 463.12], Elem 5]]) → List [Elem "15",List [Elem "20",List [Elem "35",Elem
"46"],Elem "5."]]
mapNested (*3) (Elem 1) → Elem 3
```

Задача 3. Едно дърво ще наричаме *специално*, ако оценката му спрямо подаден речник от думи е просто число.

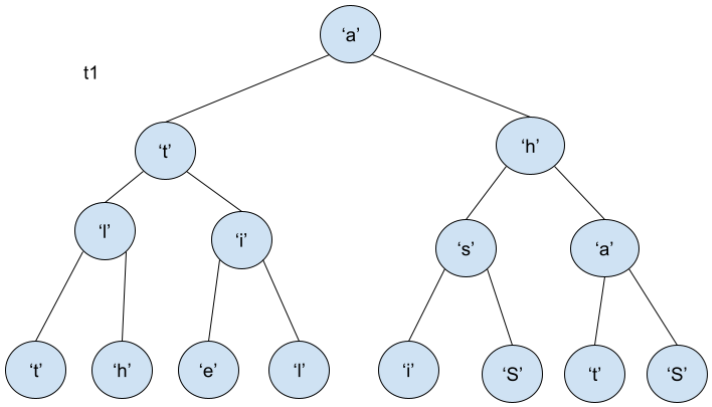
Правилата за оценяване на дърво са следните:

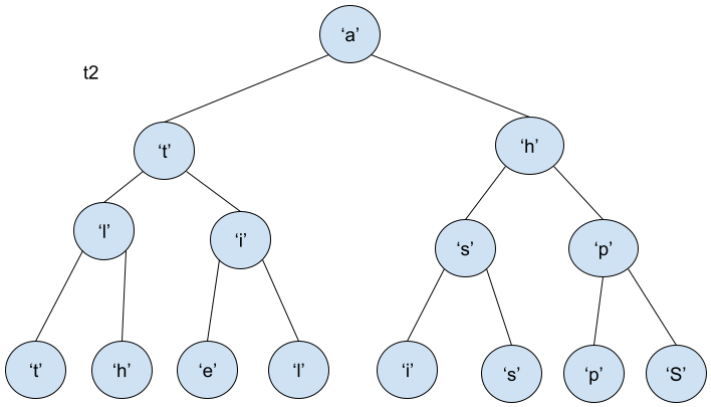
- оценката на дървото е равна на сумата от теглата на отделните думи от речника за всяко от нивата на това дърво;
- ако дума от речника е подниз на низа, образуван при обхождане на върховете (от ляво надясно) на дадено ниво на дървото, теглото на тази дума за това ниво е равно на сумата от дължината на думата и номера на нивото;
- в противен случай теглото на думата за разглежданото ниво е 0.

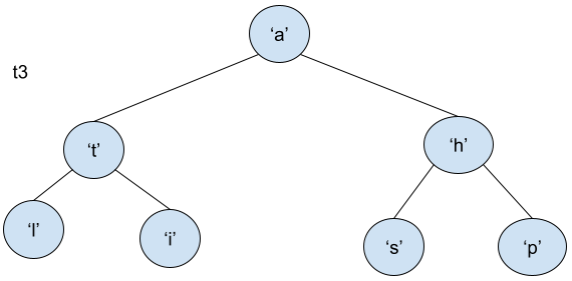
Да се дефинира функция `isPrimeDictionary` **`:: BTree -> Vocabulary -> Bool`**, която получава дърво от символи (знакове) и множество от думи и проверява дали дървото е *специално*.

Примери:

```
type Vocabulary = [String]

data BTree = Nil | Node Char BTree BTree
deriving (Show)

vocabulary :: Vocabulary
vocabulary = ["the", "a", "Some", "swimming", "liStS", "lisp"]

t1 :: BTree
t1 = Node 'a' (Node 't' (Node 'l' (Node 't' Nil Nil) (Node 'h' Nil Nil)) (Node
'i' (Node 'e' Nil Nil) (Node 'l' Nil Nil))) (Node 'h' (Node 's' (Node 'i' Nil
Nil) (Node 'S' Nil Nil)) (Node 'a' (Node 't' Nil Nil) (Node 'S' Nil Nil)))

t2 :: BTree
t2 = Node 'a' (Node 't' (Node 'l' (Node 't' Nil Nil) (Node 'h' Nil Nil)) (Node
'i' (Node 'e' Nil Nil) (Node 'l' Nil Nil))) (Node 'h' (Node 's' (Node 'i' Nil
Nil) (Node 's' Nil Nil)) (Node 'p' (Node 'p' Nil Nil) (Node 'S' Nil Nil)))
```








```
t3 :: BTree
t3 = Node 'a' (Node 't' (Node 'l' Nil Nil) (Node 'i' Nil Nil)) (Node 'h' (Node
's' Nil Nil) (Node 'p' Nil Nil))

isPrimeDictionary t1 vocabulary → False
isPrimeDictionary t2 vocabulary → False
isPrimeDictionary t3 vocabulary → True
```

Обяснения:
За t1:

- За това дърво образуваните низове са: “a” (ниво 0), “th” (ниво 1), “lisa” (ниво 2) и “theliStS” (ниво 3).
- Думата "the" се среща един път на ниво 3 и има дължина 3. Текущата оценка е: 3 + 3 = 6.
- Думата "a" се среща два пъти на ниво 0 и ниво 2 и има дължина 1. Текущата оценка става: 6 + (0 + 1) + (2 + 1) = 10.
- Думите "Some" и "swimming" не се срещат. Текущата оценка не се променя.
- Думата "liStS" се среща един път на ниво 3 и има дължина 5. Текущата оценка става: 10 + (3 + 5) = 18.
- Думата "lisp" не се среща. Текущата оценка не се променя.
- 18 не е просто число и следователно функцията връща ***`False`***.

За t3:

- За това дърво образуваните низове са: “a” (ниво 0), “th” (ниво 1) и “lisp” (ниво 2).
- Думата "a" се среща един път на ниво 0 и има дължина 1. Текущата оценка става: 0 + 1 = 1.
- Думата "lisp" се среща един път на ниво 2 и има дължина 4. Текущата оценка става: 1 + (2 + 4) = 7.
- 7 е просто число и следователно функцията връща ***`True`***.

Задача 4. В град A живеят n души. Нека ги номерираме с етикети от 1 до n. От скоро се носи слух, че в града има измамник.

За да бъде този слух верен, трябва да са изпълнени следните две условия:

1. Измамникът няма доверие на никого.
2. Всички (с изключение на измамника) имат пряко доверие на измамника.

Да се дефинира функция `findJudge` **`:: Int -> [(Int, Int)] -> Int`** , която да намира етикета на измамника. Първият аргумент е броят на хората в града. Вторият аргумент е списък от двойки, които съдържат данни за това, кой на кого има пряко доверие.

Ако слухът се окаже неверен, то функцията да връща **-1**.

Забележки. Във всички тестове:

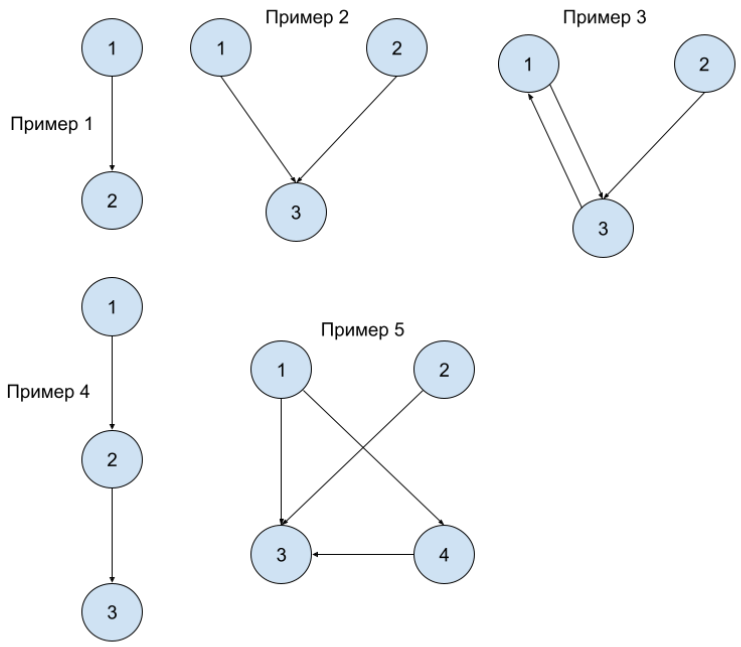
1. Точно един човек (ако съществува) ще отговаря и на двете условия.
2. Всеки човек ще познава поне един друг съгражданин.

Примери:

```
findJudge 2 [(1, 2)]                              → 2
findJudge 3 [(1, 3), (2, 3)]                      → 3
findJudge 3 [(1, 3), (2, 3), (3, 1)]              → -1
findJudge 3 [(1, 2), (2, 3)]                      → -1
findJudge 4 [(1, 3), (1, 4), (2, 3), (2, 4), (4, 3)] → 3
```

Подсказки:

1. Доверието не е транзитивно, т.е. ако номер 1 има доверие на номер 2 и номер 2 има доверие на номер 3, то от това **НЕ** следва, че номер 1 има доверие на номер 3.
2. Нека за горните примери видим какво се получава, ако на списъка съпоставим граф, в който ако има дъга от върха i до върха j, то i има пряко доверие на j:



От тук можем да направим два важни извода:

1. Понеже измамникът няма доверие на никого, то възелът с неговия номер трябва да е лист.
2. Трябва да има точно една дъга между измамника и всеки друг граждани.