Изпит по Функционално програмиране

спец. Информатика, Компютърни науки, 2 поток и избираема
9.02.2022 г.

Задача 1. Даден е безкраен поток от естествени числа **s**. “Сегмент” на **s** наричаме максимално дълга строго намаляваща последователност от числа в **s**. Всеки поток може да се разбие еднозначно на безкрайна редица от непресичащи се крайни сегменти.
а) (6 т.) Да се реализира функция **segments**, която връща поток от сегментите на даден поток **s**.
б) (8 т.) Да се реализира функция **fillSegments**, която “попълва дупките” в сегментите в потока **s** така, че всеки сегмент, започващ с **a** да се замени със сегмент с последователните числа от **a** до 0.

Примери:
```
    segments [1,5,3,2,2,6,4,7,...] → [[1],[5,3,2],[2],[6,4],[7,...],...]
fillSegments [1,5,3,2,2,6,4,7,...] → [1,0,5,4,3,2,1,0,2,1,0,6,5,4,3,2,1,0,7,...]

    (segments '(1 5 3 2 2 6 4 7 ...)) → '((1) (5 3 2) (2) (6 4) (7 ...) ...)
(fillSegments '(1 5 3 2 2 6 4 7 ...)) → '(1 0 5 4 3 2 1 0 2 1 0 6 5 4 3 2 1 0 7...)
```

Задача 2. (18 т.) Котаракът Том е откраднал кокала на булдога Спайк и той е бесен. Спайк гони Том из голямата къща, в която живеят. Стаите в къщата са номерирани с цели числа, като във всяка има една или повече врати, водеща към съседна друга стая. За нещастие на Спайк, той и Том бягат с една и съща скорост, като на всеки “ход” от 5 секунди те едновременно преминават в съседна стая на тази, в която се намират. Спайк може да хване Том само ако се озоват в една и съща стая. За да не обикаля къщата до премаляване, кучето е принудено да прибегне до хитрост. Спайк познава много добре както плана на къщата, така и навиците на предвидимия Том, така че той има неориентиран и свързан граф **house**, представящ стаите в къщата, и едноместна числова функция **tom**, която по номер на стая **x** връща номер на съседна стая **y**, към която Том винаги преминава на следващия си ход. Ако функцията върне число, което не е валиден номер на съседна стая, счита се, че Том остава вцепенен от страх в същата стая. Помогнете на Спайк като реализирате функция от по-висок ред **spike**, която за подадени параметри **house**, **tom** и номера на стаи, в които първоначално се намират Том и Спайк, връща поредица от номера на стаи, през които минава Спайк на всеки ход, за да хване Том за възможно най-малко време. Спайк може да избере тактически да изчака Том в някоя стая в рамките на два или повече ходове, ако това ще му помогне да го хване по-бързо. Представянето на графа е по ваш избор.

Примери:
```
tom room = (room + 1) `mod` 3
house = [(0, [1, 2]), (1, [0, 2]), (2, [0, 1, 3]), (3, [2])]
spike house 3 tom 2 → [3, 2, 1] (пътят на Том е [2, 0, 1])
spike house 3 tom 0 → [3, 3, 2] (пътят на Том е [0, 1, 2])

(define (tom room) (modulo (+ room 1) 3))
(define house '((0 1 2) (1 0 2) (2 0 1 3) (3 2)))
(spike house 3 tom 2) → '(3 2 1) (пътят на Том е '(2 0 1))
(spike house 3 tom 0) → '(3 3 2) (пътят на Том е '(0 1 2))
```

Бонус: (8 т.) Том не винаги е напълно предвидим. Разглеждаме вариант на задачата, в който функцията **tom** връща не едно число, а списък от числа на възможни съседни стаи, в които може да премине Том. Да се разшири функцията **`spike`** така, че да връща такава поредица от ходовете на Спайк, че в най-лошия случай той ще хване Том възможно най-рано. Ако такава поредица не съществува, да се върне празния списък.

Задача 3. (18 т.) В играта Wordle играчът се опитва да познае тайна n-буквена дума, като за всяко предположение получава като отговор n-буквен низ от символи **`+`**, **`-`** и **`?`**. Всеки символ означава дали съответната му буква е правилна, както следва:

- ➕ буквата присъства в тайната дума точно на тази позиция
- ➖ буквата не присъства в тайната дума
- ❓ буквата присъства в тайната дума, но на друга позиция

Един ход на играта Wordle се представя наредена двойка от предположение и отговор.
Да се реализира функция **`wordle`**, която по зададен списък от ходове, връща:

- n-буквената тайна дума, ако има единствена дума, която отговаря на дадените ходове
- `"many solutions"`, ако има повече от една тайна дума, която отговаря на дадените ходове
- `"no solution"`, има противоречие в дадените отговори, т.е. няма нито една n-буквена дума, за която всички отговори да са коректни.

Тук "дума" наричаме всяка поредица от (евентуално повтарящи се) малки латински букви, т.е. не е задължително поредицата да е коректна английска дума. При решение на езика Scheme, низовете могат да се представят като списък от символи (например `'(c a t)` вместо `"cat"`).
Упътване: могат да се генерират всички възможни n-буквени думи и да се елиминират тези, за които дадените указания не пасват.

Примери:

```
wordle [("cat", "+-+"), ("use", "?--")] → "cut"
wordle [("cat", "???"), ("dog", "--+")] → "no solution"
wordle [("dog", "?--"), ("dad", "?++")] → "many solutions"

(wordle '(((c a t) . (+ - +)) ((u s e) . (? - -)))) → '(c u t)
(wordle '(((c a t) . (? ? ?)) ((d o g) . (- - +)))) → "no solution"
(wordle '(((d o g) . (? - -)) ((d a d) . (? + +)))) → "many solutions"
```

Бонус: (8 т.) Да се разшири функцията **`wordle`** така, че вместо `"many solutions"` да връща ново предположение такова, че ако то се окаже грешно, да елиминира максимален брой възможности. С други думи, за всички възможни отговори на това предположение, минималният брой на елиминираните думи да е възможно най-голям.

Забележка: използването на всички стандартни функции в R^5RS и Prelude, както и на функциите accumulate, accumulate-i, filter, foldr, foldl, foldr1, foldl1, head, tail, map-stream, filter-stream и специалната форма cons-stream е позволено, но не е задължително.

```
(define (accumulate op nv a b term next)
  (if (> a b) nv
          (op (term a) (accumulate op nv (next a) b term next))))

(define (accumulate-i op nv a b term next)
  (if (> a b) nv
          (accumulate-i op (op nv (term a)) (next a) b term next)))

(define (filter p l)
```

```
  (cond ((null? l) l)
              ((p (car l)) (cons (car l) (filter p (cdr l))))
              (else (filter p (cdr l)))))

(define (foldr op nv l)
  (if (null? l) nv
         (op (car l) (foldr op nv (cdr l)))))

(define (foldl op nv l)
  (if (null? l) nv
         (foldl op (op nv (car l)) (cdr l))))

(define (foldr1 op l)
  (if (null? (cdr l)) (car l)
         (op (car l) (foldr1 op (cdr l)))))

(define (foldl1 op l)
  (foldl op (car l) (cdr l)))

(define-syntax cons-stream
  (syntax-rules ()
        ((cons-stream h t) (cons h (delay t)))))

(define head car)

(define (tail s) (force (cdr s)))

(define (map-stream f . streams)
  (cons-stream (apply            f (map head streams))
               (apply map-stream f (map tail streams))))

(define (filter-stream p? s)
  (if (p? (head s))
     (cons-stream (head s) (filter-stream p? (tail s)))
                           (filter-stream p? (tail s))))
```