Домашна работа №2 по Функционално програмиране
Специалност Компютърни науки, 2-ри курс, 1-ви поток
2017/2018 учебна година

Задача 1. Да се дефинира функция **`calcLuhnChecksum n`**, която приема целочислен аргумент **`n`** и пресмята неговата чексума на Лун.
Алгоритъмът на Лун се състои от следните стъпки:

1. Конструрира се списък **`lst`** от цифрите на даденото цяло число (списък от едноцифрените цели числа, записани чрез поредните цифри в десетичния запис на числото).
2. Обхождат се елементите на **`lst`** и тези на четна позиция се умножават по 2 (индексирането в случая започва от 1).
3. Събират се цифрите на числата, получени на предходната стъпка.
4. Получената сума се умножава по 9. Резултатът е последната цифра на полученото число.

Повече информация за алгоритъма на Лун, за неговите свойства и приложения може да се намери тук: https://en.wikipedia.org/wiki/Luhn_algorithm.
Пример
Нека разгледаме числото 7992739871.

Цифри	7	9	9	2	7	3	9	8	7	1
Удвояване на едноцифрените числа на четни позиции	7	18	9	4	7	6	9	16	7	2
Събиране на цифрите	7	9	9	4	7	6	9	7	7	2

Сумата на числата в клетките от третия ред е 67.
67 * 9 = 603, следователно търсената чексума е 3.

Задача 2. Да се дефинира функция **`gameOfLife board`**, която симулира една итерация от „*Играта на живота*“, наподобяваща зараждането, развитието и упадъка на съвкупност от живи организми.
Правила на играта са следните:

- Вселената е представена като безкрайна двумерна матрица. Елементите на матрицата се разглеждат като клетки, всяка от които може да се намира в едно от две възможни състояния: жива или мъртва (празна).

- Времето тече на дискретни стъпки (итерации). Всяка клетка взаимодейства със своите осем съседи (съседните клетки по ред, стълб и диагонал) и пресмята новото си състояние на базата на следните правила:
  - ✓ **живот:** всяка жива клетка с две или три живи съседни клетки остава жива на следващата итерация.
  - ✓ **умиране:** всяка жива клетка с по-малко от две живи съседни клетки умира от самота, а всяка жива клетка с повече от три живи съседни клетки умира от пренаселване.
  - ✓ **раждане:** всяка празна клетка с точно три живи съседни клетки се превръща в жива клетка.
- Всички раждания и умирания на дадена итерация се извършват едновременно.

Аргументът **`board`** представлява списък от двойки **(x,y),** съответни на координатите на живите клетки (това представяне позволява играта да се симулира върху безкрайно поле, стига броят на живите клетки да е краен).
Функцията трябва да върне списък с координатите на живите клетки на следващата итерация.
Примери

```
gameOfLife [(0,0),(0,1),(1,0),(1,1)] → [(0,0),(0,1),(1,0),(1,1)]
gameOfLife [(0,-1),(0,0),(0,1)] → [(-1,0),(0,0),(1,0)]
gameOfLife [(0,1),(1,2),(2,0),(2,1),(2,2)] →
           [(1,0),(1,2),(2,1),(2,2),(3,1)]
```

Задача 3. Ще наричаме едно двоично дърво *k*-балансирано, ако разликата между височините на лявото и дясното му поддървета е по-малка или равна на *k* и всяко от тези поддървета също е *k*-балансирано. Да се дефинира функция **`isBalanced tree k`**, която приема двоично дърво **`tree`** и целочислен аргумент **`k`** и връща резултата от проверката дали даденото дърво е *k*-балансирано.
За целта да се използва следната дефиниция на двоично дърво:

```
data Tree a = Empty | Node a (Tree a) (Tree a)
     deriving (Read, Show)
```