UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA

DIM0165 – PROGRAMAÇÃO FUNCIONAL

Martin A. Musicante

Lista 5

As questões desta lista são apenas para discussão dos temas das aulas. Não conta na nota, servindo apenas para motivar a reflexão sobre linguagens de programação e os seus paradigmas.

1. Visitar a descrição do módulo `Enum` em `https://hexdocs.pm/elixir/Enum.html` e se familiarizar com as suas funções e tipos de dados.

2. Escrever uma função em Elixir que receba como parâmetros um número n e uma lista contendo n valores reais e devolva um mapeamento representando o array de tamanho n, no qual cada elemento e_i seja representado por um par $i \to e_i$.

3. Escrever uma função em Elixir que receba como parâmetros dois números m e n e uma lista contendo $m \times n$ valores reais e devolva um mapeamento representando a matriz de tamanho $m \times n$, na qual cada elemento $e_{i,j}$ seja representado por um par $(i, j) \to e_{i,j}$.

4. Usar a representação da matriz dada acima para definir funções sobre matrizes (verificando se a operação é possível. As suas funções devem devolver o par `{:ok,` r`}`, caso seja possível fazer a operação e `:error`, caso contrário, sendo que r representa o resultado de cada operação):

   (a) Soma de matrizes;

   (b) Produto de duas matrizes;

   (c) Verificação de se uma matriz é triangular superior (isto é, a sua função devolve verdadeiro ou falso dependendo da matriz recebida como parametro ser ou nao uma matriz triangular superior).

   (d) Dada uma matriz, devolver uma submatriz quadrada principal (isto é, dada uma matriz de tamanho $m \times n$, devolver a matriz de tamanho $min(m, n)$, contendo o elemento que aparece na primeira linha e primeira coluna).

   (e) Dada uma matriz, devolver a submatriz diagonal principal da sua sub-matriz quadrada principal.

   (f) Dada uma matriz, devolver a submatriz diagonal secundária da sua sub-matriz quadrada principal.

   (g) Dada uma matriz, devolver a submatriz triangular inferior da sua sub-matriz quadrada principal.

   (h) Dada uma matriz, devolver a submatriz triangular superior da sua sub-matriz quadrada principal.

5. Escrever uma função que receba uma string e devolva um par de números representando, respectivamente, o número de letras maiúsculas e minúsculas que aparecem na string.

6. (adaptado de `https://www.cse.chalmers.se/edu/year/2019/course/TDA555-alex/exercises/3/`.

   Ocorrências em listas Defina as seguintes funções:

   `occursIn(xs, x):` que retorna True se x for um elemento de xs.

   `allOccurIn(xs, ys):` , que retorna True se todos os elementos de xs também forem elementos de ys.

   `sameElements(xs, ys):` , que retorna True se xs e ys tiverem exatamente os mesmos elementos.

   `numOccurrences(xs, x):` , que retorna o número de vezes que x ocorre em xs.

Nas implementações das funções acima, tente não usar recursão, mas use uma compreensão de listas! Além disso, veja se é possível usar funções já definidas nas bibliotecas do Elixir (por exemplo, a função `occursIn(xs, x)` poe ser favilmente umplementada usando `Enum.any?/2`.

De certa forma, as listas são como conjuntos: ambos são coleções de elementos. Mas a ordem dos elementos em uma lista é importante, enquanto que em um conjunto não importa, e o número de ocorrências em uma lista é importante, enquanto que em um conjunto não importa. *"bag"* é algo entre uma lista e um conjunto: o número de ocorrências é importante, mas a ordem dos elementos não é. Uma maneira de representar um *bag* é uma lista de pares de valores e o número de vezes que o valor ocorre *(keyword list*): por exemplo

```
iex(9)> ["a":  3, "b": 5]
[a: 3, b: 5]
```

Outra forma de representar um bag é mediante mapeamentos em Elixir:

```
iex(10)> %{"a" =>  3, "b" => 5}
%{"a" => 3, "b" => 5}
iex(11)> %{'a': 3, 'b': 5}
%{a: 3, b: 5}
iex(12)> %{a: 3, b: 5}
%{a: 3, b: 5}
```

Defina uma função `to_bag` para converter uma lista em um *bag*. Por exemplo, `bag ‘‘hello’’` deve ser [(‘h’,1),(‘e’,1),(‘l’,2),(‘o’,1)]

Crivo de Eratóstenes O crivo de Eratóstenes é um método antigo para encontrar números primos. Comece escrevendo todos os números de 2 a (digamos) 100. O primeiro número (2) é primo. Agora, risque todos os múltiplos de 2. O primeiro número restante (3) também é primo. Risque todos os múltiplos de 3. O primeiro número restante (5) também é primo... e assim por diante. Quando não restarem números, você terá encontrado todos os números primos no intervalo com o qual começou.

Defina uma função `crossOut` que receba uma lista de inteiros e um inteiro, de modo que `crossOut(ns, n)` remova todos os múltiplos de m de ns. Tente não implementar crossOut recursivamente, mas use uma compreensão de lista!

Agora defina uma função (recursiva!) `sieve` que receba uma lista de inteiros e devolva uma lista de inteiros. A função deve aplicar o crivo de Eratóstenes à lista de números fornecida e retorna uma lista de todos os números primos encontrados. Esta é uma função recursiva com uma lista como argumento, portanto, você deve garantir que a lista fique menor a cada chamada recursiva. Use uma lista de argumentos vazia como seu caso base. Use `sieve` para construir a lista de números primos de 2 a 100.

Jogos de números Investigaremos as propriedades dos números primos no intervalo de 2 a 100. Defina funções

- Para testar se n é um número primo (testar a sua função no intervalo de 2 a 100).
- Para testar se n é uma soma de dois números primos (testar a sua função no intervalo de 2 a 100).

A hipótese é que todo número par maior que dois pode ser expresso como a soma de dois números primos. Por exemplo, $4 = 2 + 2$, $6 = 3 + 3$, $8 = 3 + 5$. Isso é verdade para todos os números pares no intervalo de 4 a 100?

Compreensões de listas Experimente a função

```
def pairs(xs, ys) do
  for x <- xs, y <- ys do
    {x, y}
  end
end
```

e veja o que ela faz.

Uma **tríade pitagórica** é uma tripla de números inteiros (a, b, c) tal que $a^2 + b^2 = c^2$. Encontre todas as tríades pitagóricas com $a \leq b \leq c \leq 100$.

7. Escreva uma função Elixir para verificar se um número é "perfeito" ou não.

   De acordo com a Wikipedia: Na teoria dos números, um número perfeito é um número inteiro positivo que é igual à soma de seus divisores positivos próprios, ou seja, a soma de seus divisores positivos excluindo o próprio número (também conhecida como sua soma alíquota). Equivalentemente, um número perfeito é um número que é metade da soma de todos os seus divisores positivos (incluindo ele mesmo). Exemplo: O primeiro número perfeito é 6, porque 1, 2 e 3 são seus divisores positivos próprios, e $1 + 2 + 3 = 6$. Equivalentemente, o número 6 é igual à metade da soma de todos os seus divisores positivos: $(1 + 2 + 3 + 6)/2 = 6$. O próximo número perfeito é $28 = 1 + 2 + 4 + 7 + 14$. Este é seguido pelos números perfeitos 496 e 8128.