Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Algoritmos y Estructuras de Datos I
Segundo Cuatrimestre de 2024

Listas, funciones de alto orden y Listas por comprensión

NOTA Los ejercicios con * son para resolver en su casa.

Ejercicios

1. Definir las siguientes funciones:

   - `hd :: [a] -> a` retorna el primer elemento de una lista.
   - `tl :: [a] -> [a]` retorna toda la lista menos el primer elemento.
   - `last :: [a] -> a` retorna el último elemento de la lista.
   - `init:: [a] -> [a]` retorna toda la lista menos el último elemento.

2. Defina una función que dada una lista, retorne la reversa de la misma.

3. Defina una función que dadas dos listas, decida si las listas son iguales.

4. Defina una función que dada una lista decida si es un palíndromo o no.

5. Defina una función que dado un número natural, decida si el mismo es primo o no.

6. (*) Defina una función que dado un número natural `n`, retorne la lista de todos los números naturales primos menores que `n`.

7. (*) Defina una función que dada una lista de números, devuelva la lista sólo con los números primos.

8. (*) Investigue las definiciones de las funciones `take` y `drop`. Utilizando estas funciones implemente una función `cortar :: Int -> Int -> [Char] ->[Char]` que dados dos enteros `i` y `j` y un string `w`, devuelva el substring que se encuentra entre las posiciones `i` y `j`.

   ```
   > cortar 4 9 "hipopotamos"
   "opotam"

   > cortar 6 10 "hola␣mundo!"
   "mundo"
   ```

9. Escriba una función que dado un número retorne la lista de sus dígitos.

```
 > digitos 12645
[1, 2, 6, 4, 5]

> digitos (-1234)
[1, 2, 3, 4]

> digitos 0
[0]

 > digitos 004
[4]

 > digitos (-5)
[5]
```

10. (*) Utilizando ghc compile la función del ejercicio 2 para obtener código ejecutable. Puede consultar el siguiente foro para ayudarse: Haskell en 5 pasos
11. Generar una lista infinita de unos.
12. Generar una lista infinita de naturales comenzando desde un número dado.
13. Generar una lista con los primeros **n** naturales.
14. (*) Retornar los primeros 5 elementos de una lista infinita de enteros positivos.

Utilizando funciones de alto orden resolver:

15. Dada una lista de enteros, retornar sus cuadrados, es decir, dado $[x_0, x_1, \ldots, x_n]$ deberia retornar $[x_0^2, x_1^2, \ldots, x_n^2]$
16. Dado un entero positivo, retornar la lista de sus divisores.
17. Dada una lista de naturales, obtener la lista que contenga solo los números primos de la lista original.
18. (*) Dada una lista de naturales, retornar la suma de los cuadrados de la lista.

```
>sumacuad [2,3,1]
14
```

19. Dada una lista de naturales, retornar la lista con sus sucesores.

```
>listsuc [2,4,6,9]
[3,5,7,10]
```

20. Dada una lista de enteros, sumar todos sus elementos.

21. (*) Definir el factorial usando **fold**.

22. (*) Redefinir la función `and` tal que `and xs` se verifica si todos los elementos de xs son verdaderos. Por ejemplo:

```
>and [1<2, 2<3, 1 == 0]
False

>and [1<2, 2<3, 1/=0]
True
```

23. Usando `foldl` o `foldr` definir una función `tam::[a]->Int` que devuelve la cantidad de elementos de una lista dada. Dar un ejemplo en los cuales `foldr` y `foldl` evaluen diferente con los mismos parametros.

Utilizando listas por comprensión resolver:

24. Dada una lista de enteros, retornar sus sucesores.

25. (*) Dada una lista de naturales, retornar sus cuadrados.

26. Dada una lista de enteros, retornar los elementos pares que sean mayores a 10.

27. Dado un entero, retornar sus divisores.

28. (*) Definir la función `todosOcurrenEn :: Eq a => [a] -> [a] -> Bool` tal que `todosOcurrenEn xs ys` se verifica si todos los elementos de xs son elementos de ys. Por ejemplo: `todosOcurrenEn [1,5,2,5] [5,1,2,4] = True`, `todosOcurrenEn [1,5,2,5] [5,2,4] = False`

29. Dado un natural **n**, retornar los números primos comprendidos entre 2 y **n**.

30. Definir la lista infinita de los números pares.

31. Dadas dos listas de naturales, retornar su producto cartesiano.

```
> prodcart [2,3] [4,5,6]
[(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
```

32. Utilizando listas por comprensión determinar cual es el número natural entre 1 y 5000, que cumple con las siguientes restricciones:

    - Si lo divido por 1, da resto 0.

- Si lo divido por 2, da resto 1.
- Si lo divido por 3, da resto 2.
- Si lo divido por 4, da resto 3.
- Si lo divido por 5, da resto 4.
- Si lo divido por 6, da resto 5.
- Si lo divido por 7, da resto 6.
- Si lo divido por 8, da resto 7.
- Si lo divido por 9, da resto 8.
- Si lo divido por 10, da resto 9.

33. (*) Dadas una lista y un elemento retornar el número de ocurrencias del elemento x en la lista ys.

34. Escribir la función `split2 :: [a] -> [([a],[a])]`, que dada una lista `xs`, devuelve la lista con todas las formas de partir `xs` en dos. Por ejemplo:

```
> split2 [1,2,3]
[([],[1,2,3]), ([1],[2,3]), ([1,2],[3])
    ,([1,2,3],[])]
```

35. (*) Definir una función que, dada una lista de enteros, devuelva la suma de la suma de todos los segmentos iniciales.
Por ejemplo: `sumaSeg [1,2,3] = 0 + 1 + 3 + 6 = 10`.