Conjuntos y combinatoria

Taller de Álgebra I

Verano 2018

Conjuntos

type

Definamos un renombre de tipos para conjuntos:

```
type Set a = [a]
```

- Otra forma de escribir lo mismo pero más descriptivo.
- Podemos asumir que si nuestra función recibe un conjunto, no contendrá elementos repetidos (aunque Haskell no hace nada para verificarlo)
- Debemos asegurar que si devolvemos un conjunto, este no contenga elementos repetidos (Haskell tampoco hace nada automático).
- No hace falta preocuparse por el orden de los elementos (aunque en Haskell no lo sabe).

Conjuntos

type

Definamos un renombre de tipos para conjuntos:

```
type Set a = [a]
```

- Otra forma de escribir lo mismo pero más descriptivo.
- Podemos asumir que si nuestra función recibe un conjunto, no contendrá elementos repetidos (aunque Haskell no hace nada para verificarlo)
- Debemos asegurar que si devolvemos un conjunto, este no contenga elementos repetidos (Haskell tampoco hace nada automático).
- No hace falta preocuparse por el orden de los elementos (aunque en Haskell no lo sabe).

Ejercicios entre todos

- Definir `vacio :: Set Integer` que represente el conjunto vacío
- Implementar entre todos la función
  `agregar :: Integer -> Set Integer -> Set Integer`
  que dado un número n y un conjunto S nos devuelva el mismo conjunto agregándole el número n (ayuda: La función "pertenece" en Haskell existe y se llama "elem")

Conjuntos

Ejercicio

1. Implementar una función `incluido :: Set Integer -> Set Integer -> Bool` que determina si el primer conjunto está incluido en el segundo.
2. Implementar una función `iguales :: Set Integer -> Set Integer -> Bool` que determina si dos conjuntos son iguales.

```
Ejemplo> iguales [1,2,3,4,5] [2,3,1,4,5]
True
```

Más de listas

- Implementar entre todos la función `agregarATodas :: Integer -> [[Integer]] -> [[Integer]]` que dado un número *n* y una lista de listas *lls* agrega a *n* detrás de cada lista de *lls*

Más de listas

- Implementar entre todos la función `agregarATodas :: Integer -> [[Integer]] -> [[Integer]]` que dado un número *n* y una lista de listas *lls* agrega a *n* detrás de cada lista de *lls*

Ejercicios

1. Implementar una función `partes :: Integer -> Set (Set Integer)` que genere todos los subconjuntos del conjunto $\{1, 2, 3, \ldots, n\}$.

```
Ejemplo> partes 2
[[], [1], [2], [1, 2]]
```

Variaciones con repetición

Producto cartesiano

- Implementar una función
  `productoCartesiano :: Set Integer -> Set Integer -> Set (Integer, Integer)`
  que dados dos conjuntos genere todos los pares posibles (como pares de dos elementos) tomando el primer elemento del primer conjunto y el segundo elemento del segundo conjunto.

```
Ejemplo> productoCartesiano [1, 2, 3] [3, 4]
[(1, 3), (2, 3), (3, 3), (1, 4), (2, 4), (3, 4)]
```

Variaciones con repetición

Producto cartesiano

- Implementar una función
`productoCartesiano :: Set Integer -> Set Integer -> Set (Integer, Integer)`
que dados dos conjuntos genere todos los pares posibles (como pares de dos elementos) tomando el primer elemento del primer conjunto y el segundo elemento del segundo conjunto.

```
Ejemplo> productoCartesiano [1, 2, 3] [3, 4]
[(1, 3), (2, 3), (3, 3), (1, 4), (2, 4), (3, 4)]
```

Variaciones con repetición

- Implementar una función
`variaciones :: Set Integer -> Integer -> [[Integer]]` que dado un conjunto *c* y una longitud *l* genere todas las posibles listas de longitud *l* a partir de elementos de *c*.

```
Ejemplo> variaciones [4, 7] 3
[[4, 4, 4], [4, 4, 7], [4, 7, 4], [4, 7, 7], [7, 4, 4], [7, 4, 7], [7,
7, 4], [7, 7, 7]]
```

Permutaciones

Insertar un elemento en una lista

- Implementar una función `insertarEn :: [Integer] -> Integer -> Integer -> [Integer]` que dados una lista *l*, un número *n* y una posición *i* (contando desde 1) devuelva una lista en donde se insertó n en la posición i de l y los elementos siguientes corridos en una posición.

```
Ejemplo> insertarEn [1, 2, 3, 4, 5] 6 2
[1, 6, 2, 3, 4, 5]
```

Permutaciones

Insertar un elemento en una lista

- Implementar una función
  `insertarEn :: [Integer] -> Integer -> Integer -> [Integer]` que dados una lista *l*, un número *n* y una posición *i* (contando desde 1) devuelva una lista en donde se insertó n en la posición i de l y los elementos siguientes corridos en una posición.

```
Ejemplo> insertarEn [1, 2, 3, 4, 5] 6 2
[1, 6, 2, 3, 4, 5]
```

Permutaciones (DIFÍCIL!)

- Implementar una función

  `permutaciones :: Integer -> [[Integer]]`

  que genere todas las posibles permutaciones de los números del 1 al *n*.

```
Ejemplo> permutaciones 3
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

Más ejercicios

Implementar funciones que devuelvan

1. Todas las formas de ubicar n bolitas numeradas en k cajas.
2. Todas las listas ordenadas de k números distintos tomados del conjunto $\{1, \ldots, n\}$.
3. Todas las sucesiones de 0 y 1 de longitud 6 en las que hay tres 1's y tres 0's.
4. Todas las sucesiones de 0 y 1 de longitud 5 en las que hay mas 1's que 0's.
5. Implementar una función
   `subconjuntos :: Integer -> Integer -> Set (Set Integer)` que dados k y n enteros, genera todos los subconjuntos de k elementos del conjunto $\{1, 2, 3, \ldots, n\}$.

```
Ejemplo> subjconjuntos 2 3
[[1, 2], [2, 3], [1, 3]]
```

Recordar la demostración combinatoria de la igualdad

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$