Conjuntos y combinatoria II

Taller de Álgebra I

Segundo cuatrimestre 2019

Representando Conjuntos en Haskell

- Vamos a representar los conjuntos como listas. En Haskell, lo denotamos `type Set a = [a]`.
- Aunque para Haskell son sinónimos, para nostros tiene un significado específico:
  - Cuando ponemos que una función *tiene como parámetro* un `Set a`, asumimos que no tiene elementos repetidos.
  - Cuando ponemos que una función *reduce* a un `Set a`, nos comprometemos a que no tenga repetidos.
  - Debemos considerar dos listas con los mismos elementos como equivalentes, aún si para el operador == son distintas.

Del final de la última clase

Ejercicios

1. Implementar la función `agregarATodos :: Integer -> Set (Set Integer) -> Set (Set Integer)` que dado un número n y un conjunto de conjuntos cls agrega a n en cada conjunto de cls.
2. 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]]
```

Producto Cartesiano

Supongamos que tenemos dos conjuntos A, B (Con $|A| = n$,$|B| = m$). Queremos obtener el conjunto

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

En primer lugar, ¿Cuántos elementos tiene $A \times B$?
Por cada uno de los n elementos de A, hay un par con cada uno de los m elementos de B, por lo tanto,

$$|A \times B| = n \cdot m$$

¿Pero qué pasa si queremos *listar* todos estos pares? ¿Qué función deberíamos definir?

Producto Cartesiano

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)]
```

- ¿Cómo podemos encarar este ejercicio?
- Notar que tenemos dos parámetros sobre los que tenemos que hacer recursión para obtener todos los pares.
- Podría servir alguna idea como la de la suma doble...

Variaciones con repetición

Ahora consideremos el siguiente problema: ¿De cuántas maneras puedo tomar k elementos de A, considerando el orden y con reposición (es decir, pudiendo sacar varias veces el mismo elemento)?
Pues, para cada una de las k veces podría tomar cualquiera de los n elementos, por lo tanto tenemos

$$n^k$$

posibilidades.
Pero una vez más, nos gustaría poder listarlas.

Variaciones con repetición

- Implementar una función `variaciones :: Set Integer -> Integer -> Set [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]]
```

- ¿Cómo podemos pensar este ejercicio recursivamente?
- Notemos que en este caso, hay una relación entre `variaciones conj n` y `variaciones conj (n-1)`.
- Puede sernos útil pensar una función que dado un conjunto *C* y un conjunto de listas *L*, genere todas las listas producto de agregar *cada elemento de C* a *cada elemento de L*. (Que, de por sí, ¡es una recursión doble!)

Una más: Permutaciones

¿De cuántas maneras puedo ordenar los elementos de A? De $n!$ maneras.
Pensemos cómo construír estas maneras recursivamente:

- Un conjunto con 1 elemento claramente tiene un ordenamiento posible.
- Dados todos los ordenamientos para un conjunto con $(n-1)$ elementos. ¿Cómo obtengo los de el conjunto que tiene uno más?
- Para cada uno de esos ordenamientos tengo que insertar el nuevo elemento *en cada una de las posiciones posibles*.

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 -> Set [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 (también numeradas).
2. 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

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

3. Todas las listas ordenadas de k números distintos tomados del conjunto $\{1, \ldots, n\}$.
4. Todas las sucesiones de 0 y 1 de longitud 6 en las que hay tres 1's y tres 0's.
5. Todas las sucesiones de 0 y 1 de longitud 5 en las que hay mas 1's que 0's.