Práctica 1
Números naturales y recursión

Estructuras discretas
Facultad de Ciencias, UNAM

Práctica: Naturales

La práctica está orientada a desarrollar métodos para el manejo de estructuras de números naturales según la axiomática de Peano; es decir, trabajaremos con la función sucesor S. Para esta práctica se considerará la siguiente estructura de dato:

```
data Natural = Cero | S Natural deriving Show
```

Y se realizarán las siguientes funciones sobre esta estructura de dato:

1. Definir una función `a_natural` que tome un entero (0,1,2,...) y lo lleve a la estructura de dato `Natural`. Por ejemplo `a_natura 0 = Cero` o bien `a_natural 2 = S (S Cero)`.

2. Definir una función `a_entero` que tome un dato `Natural` a su valor entero. Por ejemplo `a_entero 0 = Cero` o `a_entero (S (S Cero))`

3. Definir una función que realice una suma sobre la estructura de natural; es decir, que se ejecute como:

   ```
   suma_nat (S (S Cero)) (S Cero) = S (S (S Cero))
   ```

4. Definir una función que realice la multiplicación de la estructura de dato natural; por ejemplo que realice:

   ```
   mult_nat (S (S Cer)) (S (SCero)) = S (S (S (S Cero)))
   ```

Práctica: recursión

La recursión es esencial para definir funciones y estructuras. En los siguientes ejercicios se trabajará sobre números naturales (enteros o `Int` en Haskell):

1. Definir una función que tome como un argumento un entero y regrese el número de Fibonacci correspondiente a es entero. Por ejemplo: `fibo 4 = 3`.

2. Definir la función de multiplicaición entre dos enteros de forma recursiva:
   - $n \cdot 0 = 0$
   - $n \cdot (m+1) = n + n \cdot m$

3. Definir la función de potencia de forma recursiva:

- $n^0 = 1$
- $n^{m+1} = n \cdot n^m$

4. Definir la función de factorial de forma recursiva:

   - $0! = 1$
   - $(n+1)! = n! \cdot (n+1)$

5. Definir el algoritmo de la división en base a la función:

$$div(a,b) = \begin{cases} (0,a) & \text{si } a < b \\ f\big(div(a-b,b)\big) & \text{si } a \geq b \end{cases}$$

   donde $f(x,r) = (x+1,r)$.

Práctica: árboles

Para generar una estructura de dato de tipo árboles aprovecharemos la estructura recursiva y definiremos los árboles en Haskell de la siguiente forma:

```
data ArbolB a = Vacio | Nodo a (ArbolB a) (ArbolB a) deriving (Eq, Show)
```

Esta estructura nos dice que un árbol se puede definir como un elemento nulo, o bien agregando a un nodo otros árboles. Por ejemplo, podemos conformar un árbol a partir de dos árboles vacíos como:

```
t1 = Nodo "t1" Vacio Vacio
```

En este caso el elemento `"t1"` indica el nombre del nodo. De esta forma, podemos crear nuevos árboles como $Nodo”t2”t1t1$, etc. A partir de esta estructura se realizará lo siguiente:

1. Definir una función que calcule el número de nodos del árbol.

2. Definir una función que calcule la profundidad de un árbol.