Facultad de Ciencias Exactas, Ingeniería y Agrime
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Estructura de Datos y Algoritmos II

Nombre y Apellido: Legajo:

# Examen Final

**1.** Un *trie* es una clase de árbol de búsqueda, donde las búsquedas se realizan por prefijos. Las claves son generalmente cadenas de caracteres y todos los nodos descendientes de un nodo comparten el mismo prefijo. La forma en la que se almacena la información permite hacer búsquedas eficientes de cadenas que comparten prefijos, por esta razón esta estructura es comúnmente usada para representar diccionarios en los teléfonos móviles, los cuales son usados por ejemplo para el autocompletado.

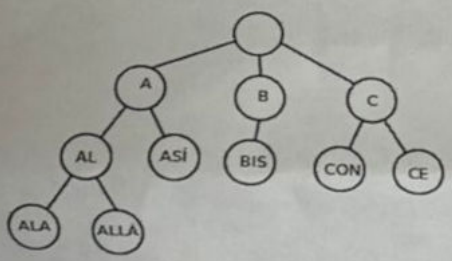
Utilizaremos la siguiente definición formal de *trie*:

Un árbol general $t$ es un *trie* si cumple los siguientes invariantes:

- La raíz de $t$ es la cadena vacía.
- Cada nodo de $t$ es prefijo de los nodos de sus descendientes.
- Nigún nodo de $t$ es prefijo de alguno de sus hermanos.

El siguiente es un ejemplo de un *trie* usado para representar las palabras: `""`, `"a"`,`"b"`,`"c"`,`"al"`,`"así"`,`"bis"`,`"con"`, `"ce"`, `"ala"` y `"allá"`.

$t =$




Usaremos el siguiente tipo de datos para representar esta estructura en Haskell:

```
data Trie = E | N String [Trie]
```

Definir las siguientes funciones en Haskell de manera eficiente:

a) isPrefix :: String → String → Bool, que dadas dos cadenas $s$ y $s'$ determina si $s$ es prefijo de $s'$.
Por ejemplo:

```
isPrefix "" "a" = True
isPrefix "abc" "abcbde" = True
isPrefix "abcde" "abc" = False
```

b) inv2 :: Trie → Bool, que determine si un árbol satisface el segundo invariante de *tries*.

c) inv3 :: Trie → Bool, que determine si un árbol satisface el tercer invariante de *tries*.

d) Usando las funciones definidas anteriormente definir una función isTrie :: Trie → Bool, que determine si un árbol es un *trie*.

e) prefixes :: String → Trie → [String], que dada una cadena y un *trie* $t$ devuelva todas palabras representadas en $t$ que tienen como prefijo la cadena dada.
Por ejemplo, prefixes `"al"` $t$ = `["al","ala","allá"]`

Dadas las siguientes funciones:

```
words :: Trie → [String]
words E = []
words (N s xs) = s : concat (map words xs)

lookUp :: String → Trie → Bool
lookUp s E = False
lookUp s (N s1 xs) | s ≡ s1 = True
                   | isPrefix s1 s = or (map (lookUp s) xs)
                   | True = False

elem :: a → [a] → Bool
elem _ [] = False
elem x (y : xs) = x ≡ y ∨ elem x xs
```

Probar por inducción estructural sobre $t$ que elem $s$ (words $t$) = lookUp $s$ $t$. **Ayuda:** Puede usar el siguiente lema elem $x$ ∘ concat = or ∘ map (elem $x$).

**2.** Un evento de ola de frío se define cuando tanto las temperaturas máximas como las mínimas son iguales o inferiores, por lo menos durante 3 días consecutivos, a cierto valor que depende de cada localidad (percentil 10 del semestre frío abril-agosto).
Dada una secuencia de valores de pares numéricos que representan las temperaturas máximas y mínimas diarias, y un valor $v$ que representa el percentil 10 de una localidad, se quiere calcular cuánto duró la ola de frío más larga.
Definir una función coldWaveDays : Seq (Float, Float) → Float → Nat que dada una secuencia de pares de temperaturas y un valor $v$, devuelva la cantidad de días de la ola de frío más larga.

coldWaveDays⟨(10,5),(15,4),(10,3)⟩6 = 0
coldWaveDays⟨(10,5),(5,4),(10,3)⟩6 = 0
coldWaveDays⟨(8,5),(7,5),(6,5),(5,4),(10,3),(2,5),(5,2)⟩7 = 3

Notar que en el primer ejemplo, no hubo ningún día con temperatura máxima y mínima menor o igual a 6, en el segundo hubo sólo un día con ambas temperautras iguales o menores a 6, pero se necesitan al menos 3 días consecutivos para que sea una ola de frío, mientras que en el tercer ejemplo, las temperaturas (7,5), (6,5) y (5,4) son todas inferiores o iguales a 7 y además corresponden a 3 días consecutivos, es decir, están dentro de una ola de frío.
Definir coldWaveDays con profundidad en $O(\lg n)$, donde $n$ es la longitud de la secuencia.