Proyecto 2

Algoritmos y Estructuras de Datos I
Laboratorio

3 de septiembre de 2007

En este proyecto hay una serie de funciones para programar en Haskell. En todos los ejercicios escribir el tipo (y la clase) al que pertenece la función. Además poner todas las funciones en un módulo.

1. Escribir una función `contarLa :: String -> Integer` que cuenta la cantidad de apariciones de la subcadena `"la"` en un string.

2.

   a) Programar la función

   ```
   intercal :: a -> [a] -> [[a]]
   ```

   donde `intercal x ys` devuelve todas las posibles intercalaciones de `x` en la lista `ys`. Por ejemplo:

   ```
   intercal 100 [6,3] = [[100,6,3],[6,100,3],[6,3,100]]
   intercal 1 [3,4,5] = [[1,3,4,5],[3,1,4,5],[3,4,1,5],
                         [3,4,5,1]]
   intercal 'u' "clo" = ["uclo","culo","cluo","clou"]
   intercal 3 [] = [[3]]
   ```

   Ayuda. Pensar inducción sobre la lista. O sea, ver que devuelve la función para el caso en que la lista sea vacía y para el caso en que no lo sea. Para el caso no vacía es posible que necesite crear una función auxiliar. Para ella aplique también el razonamiento inductivo (se puede usar una función del proyecto anterior!).

   b) Programar la misma función `intercal` utilizando solo el funcional `map` y sin utilizar funciones auxiliares.

   Ayuda. Pensar inducción sobre la lista igual que en el ejercicio de item anterior. Para el caso base (lista vacía) es lo mismo. Para el caso inductivo ver si se puede usar `map`.

3. Programar la función `selectionSort` que dada una lista devuelve la misma ordenada pero implementando el siguiente algoritmo:

   Caso base: Si la lista es vacía devuelve la lista vacía.

   Caso inductivo: Si la lista es no vacía toma el menor elemento de la lista y lo mueve al principio de la lista ordenada por la llamada recursiva.

4. Programar la función `conjuntos` que dada una lista con elementos distintos entre sí devuelva todas las listas posibles con los subconjuntos de elementos de la lista original.

   Por ejemplo:

```
conjuntos [1,2,3] = [[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]
conjuntos [3] = [[],[3]]
conjuntos [] = [[]]
```

5. La rayuela es un juego en el cual hay una secuencia de n baldosas, y comenzando desde la primera hay que llegar a la última saltando de a una baldosa (salto corto) o de a dos (salto largo). Por ejemplo, si tengo 3 baldosas puedo saltar de la primera a la ultima (un salto largo) o del la primera a la segunda y despues a la última (dos saltos cortos). Programar una función `rayuelas n :: Integer -> [[Int]]` que devuelva todos los saltos posibles en una rayuela de n baldosas. Por ejemplo:

```
rayuela 3 = [[2],[1,1]]
rayuela 4 = [[1,2],[2,1],[1,1,1]]
rayuela 2 = [[1]]
rayuela 5 = [[2,2],[1,1,2],[1,2,1],[2,1,1],[1,1,1,1]]
```

6. El Baguenaudier es un juego (se mostró en clase) en el cual hay n anillos engarzados a una varilla los cuales tienen que ser destrabados de la misma. Los únicos movimientos que se pueden efectuar son sacar(poner) el primer aro o sacar(poner) el primer aro que le sigue al primero engarzado a la varilla (este último movimiento no se puede efectuar si todos los aros estas destrabados o si el único trabado es el último). Si representamos los n aros con una lista de tamaño n donde si el aro esta engarzado se reprenta con un 1 y en caso contrario con un 0, los movimientos posibles serán:

   $0, x, x, \cdots, x \quad \longleftrightarrow \quad 1, x, x, \cdots, x$

   $0, \cdots, 0, 1, 0, x, \cdots, x \quad \longleftrightarrow \quad 0, \cdots, 0, 1, 1, x, \cdots, x$

   Hacer una función `sacar` que devuelva los movimientos para sacar n anillos.

   Ayuda Los movimientos se pueden codificar con un `0` si se efectua el primer movimiento y con un `1` si se efectúa el segundo. Hay que hacer 2 funciones: `sacar n` y `poner n`. Por ejemplo:

```
sacar 2 = [1,0]
poner 2 = [0,1]
```

   Preguntas: ¿Encuentra algún patrón en el resultado? ¿A que se debe este comportamiento?

7. Don Cambiazo es el quiosquero del barrio. Es un hombre muy dedicado a la actividad comercial que realiza ya que la misma se ha convertido no solo en su medio de subsistencia si no en una obsesión. Todos los días (incluidos sábados, domingos y feriados) se levanta dos horas antes de abrir el quisco, se hace unos mates, prende LV3 y se dispone a meditar distintas maneras de mejorar la forma en que desarrolla su actividad laboral. La problemática que suele analizar

en estos momentos de autocontemplación es diversa y a menudo se centra en como hacer para tener siempre monedas para dar vuelto.

Una mañana, Don Cambiazo se despertó algo incomodo. El pobre hombre había dormido mal ya que tuvo una pesadilla recurrente toda la noche. En su pesadilla él estaba muy cómodo atendiendo el quiosco, cuando de pronto un nuevo cliente entró y le pidió un caramelo Media Hora. Con la confianza de saber que su quiosco tiene de todo, incluido esos horribles caramelos, saco sin mirar uno de ellos de una cajita del mostrador y se lo dio al cliente, al mismo tiempo que decía "son 34 centavos". El cliente le acercó una moneda de un peso, y él, de forma rutinaria, abrió la caja de las monedas para darle el vuelto. La caja tenía tres compartimentos con monedas de 5, 2 y 1 centavos cada uno. No solo eso, si no que cada compartimento contenía infinitas monedadas de cada valor. Esto le produjo mucho placer, ya que la disposición infinita de cambio era una de sus fantasías más codiciadas. Pero resultó en el sueño, que al momento de devolver el vuelto, él se quedaba inmóvil. Por más que intentaba calcular cuantas monedas de cada valor tenía que dar al cliente no lo podía hacer. Esta era una tarea rutinaria que Don Cambiazo la hacía casi sin pensar. Pero en su pesadilla no lo podía hacer. En ese momento el cliente se empezaba a reír, y el despertaba de forma violenta y exaltada.

Esa mañana, traumado por su pesadilla, Don Cambiazo no prendió la radio y se puso a pensar como solucionar el problema del cambio. El sabía que su quiosco siempre tenía un montón de monedas de cada valor (infinitas), pero le preocupaba profundamente no poder algún día calcular el cambio. A partir de esto, decidió entonces comprarse una computadora y hacer él mismo un programa que resuelva el problema. Mucha idea de programación Don Cambiazo no tenía así que, asesorado por un cliente alumno del FaMAF, se puso a leer el tutorial de Haskell y unos apuntes sobre inducción. Después de una semana de intenso estudio y algunas pruebas llegó a un programa que hacía algo parecido a lo que Don Cambiazo hacía manualmente. Al programa había que darle como parámetro una lista con los distintos valores disponibles de monedas en orden decreciente y la cantidad que se desea convertir en cambio. La función devuelve una lista de las monedas disponibles que sumadas dan el valor a cambiar. La primera versión del programa era la siguiente:

```
cambio :: Integral a => [a] -> a -> [a]
cambio [] 0 = []
cambio [] (v+1) = error "No hay cambio"
cambio (m:ms) v | m <= v = m: cambio  (m:ms) (v-m)
                | m > v = cambio ms v
```

Según Don Cambiazzo, el programa funciona así: hago inducción sobre la listas de valores de monedas para cambio. Si no tengo ninguna y el valor a convertir es 0, devuelvo la lista vacía (cambio de $0 es ninguna moneda). Si el valor a convertir es distinto de cero, devuelvo error (ya que no tengo cambio). Para el caso inductivo, si la moneda más grande es menor o igual que el valor, puedo dar esta moneda como cambio y volver a hacer todo el proceso con el valor a cambiar menos la moneda que dí. Si el valor de la moneda es mayor que el valor a cambiar, esa moneda no se puede usar para dar el cambio, entonces repito todo el proceso sin este valor de moneda (recordar que la lista de valores de monedas esta en orden decreciente).

Muy contento con su programa (no solo calcula el cambio si no que siempre lo devuelve con las monedas más grandes!!) Don Cambiazzo se fue a dormir, confiado que no iba a volver a tener esa pesadilla otra vez. Sin embargo no fue así. Esa noche tuvo el mismo sueño pero en la caja había monedas de 5 y 2 centavos únicamente. Al igual que en el otro sueño no pudo calcular el cambio. Pero esta vez el cliente en vez de reírse se fue enojado.

Inmediatamente después de despertarse prendió la computadora y probó reproducir su pesadilla en el Hugs:

```
Main> cambio [5,2] (100-34)
[5,5,5,5,5,5,5,5,5,5,5,5,5
  Program error: No hay cambio
```

Totalmente deprimido, Don Cambiazzo llegó tarde a atender el quiosco por primera vez en su vida. Al llegar estaba esperándole el cliente alumno del FaMAF para comprarle un caramelo Media Hora. Inmediatamente y antes de abrir la persiana del negocio le comentó su desventura con el Haskell. El cliente, después de ver el programa de Don Cambiazzo, le sugirió que pruebe hacer un programa que devuelva todas las soluciones posibles y que después tome la primera solución. Esto es:

```
cambio :: Integral a => [a] -> a -> [a]
cambio ms v = head  (posCambio ms v)

posCambio :: Integral a => [a] -> a -> [[a]]
posCambio [] 0     = [[]]
posCambio [] (v+1) = []
...
```

Ejercicio: Podría usted usted ayudar a Don Cambiazzo y completar el programa?

Don Cambiazzo, sorprendido por el consejo, atinó a preguntar porqué el programa solo toma la primer solución. Teniendo todas las soluciones en la pantalla podría elegir a mano la que más le guste y el costo del cálculo sería el mismo, ya que el programa aconsejado termina calculando todas las soluciones. Su cliente le respondió que si no lo hacía seguramente tendría que comprarse una computadora más grande.

Preguntas: ¿Es correcto el consejo del cliente? ¿Por qué?