Teoría de los lenguajes de programación.

Miroslav Vladimirov

email: miro.kv89@gmail.com

Teléfono: 676867565

1.Crear la función dibuja:

a) (*1 punto*). Implementar una función dibujaSkyline en **Haskell** que devuelva el dibujo del skyline según el algoritmo anteriormente descrito.

1.Transformamos la lista de coordenadas del skyline en una lista de alturas para cada coordenada x del horizonte.

```
--Funcion divideCoordenadas: auxiliar de dibujar. Dadas unas coordenadas
--(Skyline) devuelve una lista de alturas.
divideCoordenadas [] _ _= []
divideCoordenadas ((x1,x2):xs) n lastH
  | x1>n = lastH :divideCoordenadas ((x1,x2):xs) (n+1) lastH
  | otherwise = x2:divideCoordenadas xs (n+1) x2
```

2.Calculamos la altura máxima del skyline. (Utilizamos la función “maximum”)

```
--Funcion dibuja: dada una lista de coordenadas devuelve su dibujo en un
--String haciendo uso de las siguientes funciones auxiliares:
-- dibujar
-- dibujaLinea
-- divideCoordenadas

dibujaSkyline [] = []
dibujaSkyline (x:xs) = dibujar (divideCoordenadas (x:xs) 0 0) (maximum(divideCoordenadas (x:xs) 0
```

3.Para generar cada línea del dibujo del skyline, comenzamos creando la línea de caracteres para esa altura máxima y descendemos hasta llegar a 0. Para dibujar una línea de caracteres, en cada coordenada x ponemos un '*' si la altura del skyline en esa coordenada es mayor o igual que la altura de la línea que estamos dibujando y un ' ' en caso contrario.

4.Al llegar a la coordenada 0 ponemos una línea de guiones " para simular el suelo.

El punto 3 y 4 se presentan en los siguientes dos funciones:

```
--Funcion dibujar: auxiliar de dibuja. Dada una lista de alturas dibuja
--recursivamente linea a linea (mediante dibujaLinea) por pantalla su
--correspondiente dibujo.
dibujar [] n = []
dibujar (x:xs) n
  | n >=0 = dibujaLinea (x:xs) n ++ "\n" ++ dibujar (x:xs) (n-1)
  |otherwise = []

--Funcion dibujaLinea: auxiliar de dibujar. Dada una lista de alturas
--dibuja la linea de la altura N.
dibujaLinea [] _ = []
dibujaLinea (x:xs) n
  | n==0 = '-':dibujaLinea xs n
  | x>=n = '*':dibujaLinea xs n
  | otherwise = ' ':dibujaLinea xs n
```

A continuación se muestra el correcto funcionamiento de dibujaSkyline:

```
Prelude> :load "Skyline.hs"
[1 of 1] Compiling Skyline          ( Skyline.hs, interpreted )
Ok, modules loaded: Skyline.
*Skyline> putStr(dibujaSkyline([(1,11),(3,13),(9,0),(12,7),(13,8),(15,7),(16,3)
,(19,18),(22,6),(23,13),(29,0)]))
                    ***
                    ***
                    ***
                    ***
                    ***
   ******           *** ******
   ******           *** ******
 ********           *** ******
 ********           *** ******
 ********           *** ******
 ********     **    *** ******
 ********   ****    *** ******
 ********   ****    **********
 ********   ****    **********
 ********   ****    **********
 ********   ******************
 ********   ******************
 ********   ******************
------------------------------
```

b) (*1 punto*). Implementar un predicado dibujaSkyline en **Prolog** que devuelva el dibujo del skyline según el algoritmo anteriormente descrito.

Los pasos que se han seguido para la generación de dibujaSkyline en prolog han sido los mismos que en haskell, con la diferencia de que en haskell se uso la función “maximum”,mientras que aquí hubo que crear “maxLista” para conocer el elemento mayor de una lista:

1.Transformamos la lista de coordenadas del skyline en una lista de alturas para cada coordenada x del horizonte.

```
% Funcion divideCoordenadas: auxiliar de dibujar. Dadas unas coordenadas
% (Skyline) devuelve una lista de alturas.
divideCoordenadas([],_,_,[]).
divideCoordenadas([c(X1,X2)|Rx],N,LastH,[LastH|L]):- X1>N,
        divideCoordenadas([c(X1,X2)|Rx],(N+1),LastH,L),!.
divideCoordenadas([c(_,X2)|Rx],N,_,[X2|L]):-
        divideCoordenadas(Rx,(N+1),X2,L),!.
```

2.Calculamos la altura máxima del skyline.

```
%Funcion maxLista devuelve el elemento con mayor valor de una lista.
maxLista([],[]).
maxLista([X],X).
maxLista([X,Y|L],Z):-maxLista([Y|L],U),max(X,U,Z),!.
```

3.Para generar cada línea del dibujo del skyline, comenzamos creando la línea de caracteres para esa altura máxima y descendemos hasta llegar a 0. Para dibujar una línea de caracteres, en cada coordenada x ponemos un '*' si la altura del skyline en esa coordenada es mayor o igual que la altura de la línea que estamos dibujando y un ' ' en caso contrario.

4.Al llegar a la coordenada 0 ponemos una línea de guiones " para simular el suelo.

El punto 3 y 4 se presentan en los siguientes dos funciones:

```
% Funcion dibujar: auxiliar de dibuja. Dada una lista de alturas dibuja
% recursivamente linea a linea (mediante dibujaLinea) por pantalla su
% correspondiente dibujo.
dibujar([],_,[]).
dibujar([X|Rx],N,R):-N>0,dibujaLinea([X|Rx],N,S),M is N-1,
        dibujar([X|Rx],M,T),concat(S,T,R),!.
dibujar([X|Rx],N,R):-N==0,dibujaLinea([X|Rx],N,S),concat("",S,R),!.

% Funcion dibujaLinea: auxiliar de dibujar. Dada una lista de alturas
% dibuja la linea de la altura N.
dibujaLinea([],_,"\n").
dibujaLinea([_|Rx],N,R):-N==0,dibujaLinea(Rx,N,S),concat("-",S,R),! .
dibujaLinea([X|Rx],N,R):-N>0,X>=N,dibujaLinea(Rx,N,S),concat("*",S,R),! .
dibujaLinea([_|Rx],N,R):-dibujaLinea(Rx,N,S),concat(" ",S,R),! .
```

El predicado dibujaSkyline funciona exactamente igual que en haskell:

```
% Funcion dibuja: dada una lista de coordenadas devuelve su dibujo en un
% String haciendo uso de las siguientes funciones auxiliares:
% -dibujar
% -dibujaLinea
% -divideCoordenadas
% -maxLista
dibujaSkyline([],[]).
dibujaSkyline(Lista,R):-divideCoordenadas(Lista,0,0,X),
        maxLista(X,Z),dibujar(X,Z,R),!.
```

Y por ultimo una muestra del funcionamiento del predicado, con los mismos datos de la muestra en haskell:


```
                    ***
                    ***
                    ***
                    ***
                    ***
   *******          *** *******
   *******          *** *******
  *********         *** *******
  *********         *** *******
  *********         *** *******
  *********     **  *** *******
  *********   ****  *** *******
  *********   ****  ***********
  *********   ****  ***********
  *********   ****  ***********
  *********   ******************
  *********   ******************
  *********   ******************
------------------------------
D = '                       ***         \n                    ***
\n                      ***         \n                    ***         \n
                    ***         \n    ******          *** ****** \n    *
*****           *** ****** \n *********          *** ****** \n *******
**            *** ****** \n *********          *** ****** \n *********
    **     *** ****** \n *********    ****    *** ****** \n *********    *
***    ********** \n *********    ****    ********** \n *********    ****
    ********** \n *********    ****************** \n *********    ********
********** \n *********    ****************** \n----------------------
--------\n'.
```

2.(*1 punto*). Compare la eficiencia, **referida a la programación**, del algoritmo del skyline en **Haskell**, **Prolog** y **Java**. ¿Qué lenguaje considera más adecuado escoger para la implementación de este algoritmo? Justifique su respuesta.

En cuanto a la eficiencia de la programación el lenguaje Java es el mas sencillo de escribir como de leer por el parecido que tiene al lenguaje natural. Por otro lado Haskell y Prolog necesitan de muchas menos lineas de código para hacer exactamente lo mismo que en Java. Según el libro base de la asignatura Prolog es un lenguaje ideal con una sintaxis concisa, ausencia de declaración de variables e independencia con el mecanismo de ejecución, aunque compromete otros principios como la legibilidad, la eficiencia en la ejecución y la fiabilidad. A falta de un conocimiento extenso de los lenguajes propuestos, he de reconocer que el tiempo empleado en Haskell ha sido mas del doble que el empleado en Prolog, por lo que a pesar de que Haskell me parezca mas eficiente, considero Prolog como el lenguaje adecuado para la implementación de este algoritmo.

3.(*1 punto*). Indique, con sus palabras, qué permite gestionar el predicado predefinido no lógico, corte (!), en **Prolog**. ¿Cómo se realiza este efecto en **Java**?

El predicado corte (!) permite detener la búsqueda en cuanto se encuentra el primer resultado. Prolog funciona como el algoritmo de vuelta atrás por lo que devuelve todas las respuestas posibles hasta llegar a "false" que indica que ya no hay mas. Con el corte conseguimos una sola respuesta y la detener la búsqueda tras ello con lo que el programa creado no hace una busqueda exhaustiva, sino que se detiene al encontrar la primera respuesta válida.

Para realizar lo mismo en Java deberíamos poner una condición "if" que en cuanto se cumpla, o dejé de cumplir, se detenga la búsqueda. Si lo que buscamos es la primera respuesta, es suficiente con utilizar un booleano en la condición y cambiar su valor al encontrar la respuesta::

```
boolean encontrado = false;
while(iterador.hasNext() && !encontrado){
     if(iterador.next().getCodigo() == miCodigo){
     encontrado = true;
     ...
     }
...
}
```

4.(*1 punto*). Relacione los tipos de datos del problema definidos en **Haskell**, **Prolog** y **Java** (algunos de ellos son subclases de la clase Skyline). Indique qué constructores de tipos se han utilizado en cada caso.

Los distintos tipos de datos utilizados mas allá de los predefinidos (Int, String), han sido :

- Edificio:

- Haskell: Edificio = (Int,Int,Int)
- Prolog: edificio(ed(X1,X2,H)).
- Java: Edificio (int x1, int x2, int h)

En los tres casos el tipo de dato Edificio consiste en un punto inicial, un punto final, y la altura del edificio.

- Coordenada:

- Haskell: Coordenada = (Int,Int)
- Prolog: coordenada(c(X1,Xh)).
- Java: Coordenada (int abcisa,int altura)

En los tres casos el tipo de dato Coordenada consiste en un punto inicial (abscisa) y una altura.

- Skyline:

- Haskell: Skyline = [Coordenada]
- Prolog: skyline([c(X1,Xh)|R]).
- Java: ArrayList<Coordenada> skyline;

En los tres casos el tipo de dato Skyline consiste en una lista de coordenadas.

Se entregarán tambien los ficheros Skyline.hs y Skyline.pl correspondientes al código generado en Haskell y Prolog respectivamente. En ambos archivos se presenta mediante comentarios una pequeña descripcion de las funciones/predicados programadas/os.