Teoría de los Lenguajes de Programación Práctica curso 2015-2016

Enunciado

Fernando López Ostenero y Ana García Serrano

Sumario

1. Introducción: Skyline de una ciudad....3
2. Enunciado de la práctica....3
2.1 Descripción detallada del algoritmo del Skyline....3
2.1.1 Combinación de dos líneas de horizonte....4
2.2 Tarea 1: Implementación en Haskell....6
2.2.1 Tipos de datos....6
2.2.2 Funciones a implementar....6
2.3 Tarea 2: Implementación en Prolog....7
2.3.1 Tipos de datos....7
2.3.2 Predicados a implementar....7
3. Cuestiones sobre la práctica....8
4. Documentación a entregar....9

1. Introducción: Skyline de una ciudad.

La línea de horizonte que dibujan los edificios de una ciudad se conoce como Skyline. En esta práctica vamos a implementar un algoritmo que partiendo de los datos de situación y altura de una serie de edificios, nos devuelva la línea de horizonte (consistente en una lista de coordenadas) o Skyline que forman dichos edificios.

Se trata de un algoritmo de tipo Divide y Vencerás, que divide el problema en dos (o más) subproblemas similares (pero de menor tamaño), los resuelve y combina las soluciones obtenidas en la solución final. Cuando el problema es de un tamaño cuya solución se puede realizar de forma trivial, ésta se devuelve directamente, sin necesidad de volver a dividirlo y combinar las soluciones.

En el caso que nos ocupa, el problema se dividirá en dos subproblemas similares: calcular el skyline de la mitad de los edificios por un lado y el skyline de la otra mitad por otro. La función de combinación recibirá, por tanto, dos líneas de horizonte que deberán ser combinadas en una única. El caso trivial consistirá en calcular la línea de horizonte de un único edificio.

2. Enunciado de la práctica

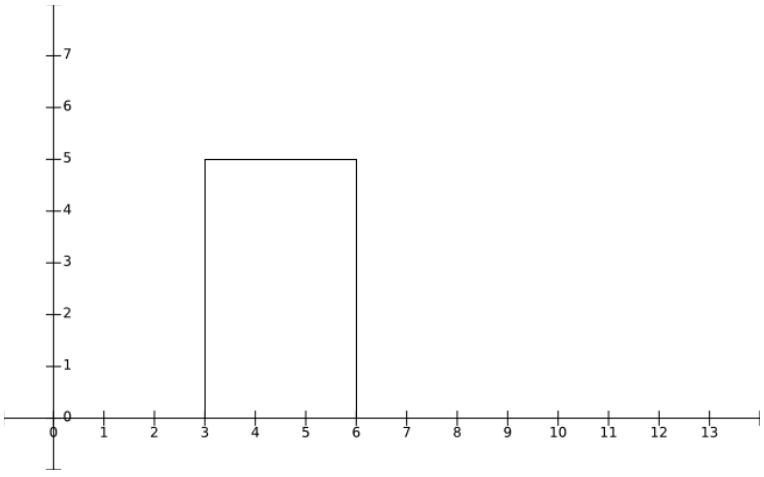
La práctica consiste en elaborar dos versiones de un programa que implemente el algoritmo del Skyline (tipo Divide y Vencerás): una en Haskell y otra en Prolog.

2.1 Descripción detallada del algoritmo del Skyline

Este algoritmo recibe como entrada un conjunto (o lista de edificios), cada uno de los cuales se supone rectangular. Dichos edificios se representan por una terna de números (x_1,x_2,h) que, sin pérdida de generalidad, supondremos enteros:

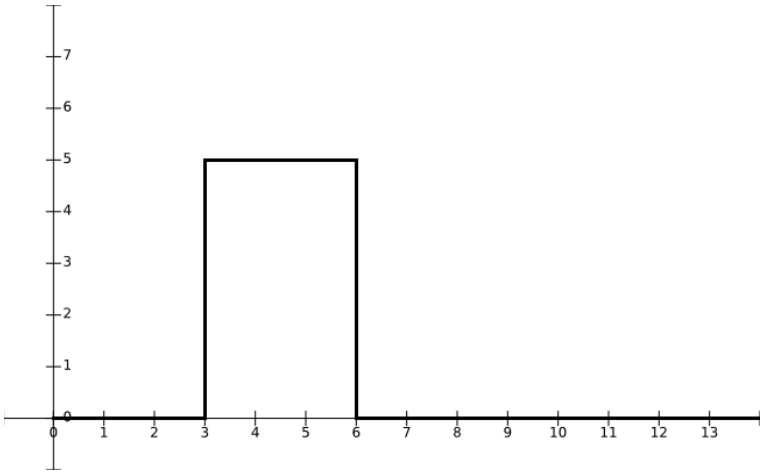
- x_1 primera coordenada x del edificio
- x_2 segunda coordenada x del edificio
- h altura del edificio

Así, si tenemos un edificio representado por la terna `(3,6,5)`, significará que dicho edificio se extiende desde la coordenada x 3 hasta la coordenada x 6 con una altura de 5:



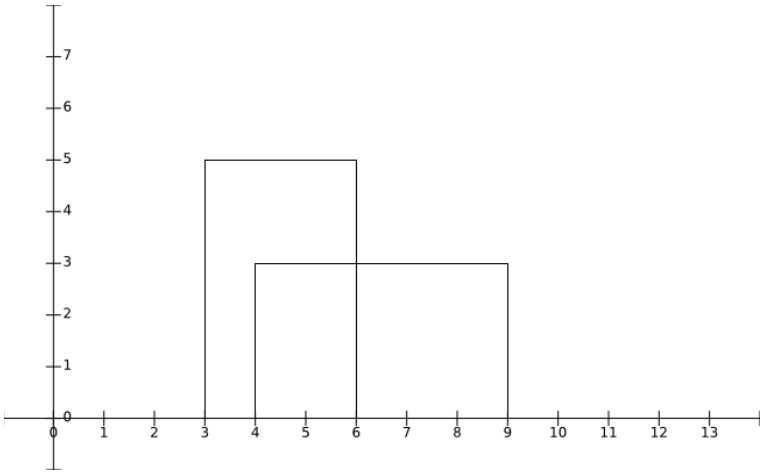
La salida devuelta por el algoritmo consiste en una línea de horizonte, que vendrá representada como una lista ordenada de puntos en el plano donde dicha línea cambia de altura. Así, pues, para el edificio anterior, la línea de horizonte sería `[(3,5),(6,0)]`, puesto que en la coordenada x 3, la

línea de horizonte sube desde la altura 0 a la altura 5 y en la coordenada x 6 la línea de horizonte baja nuevamente a altura 0.

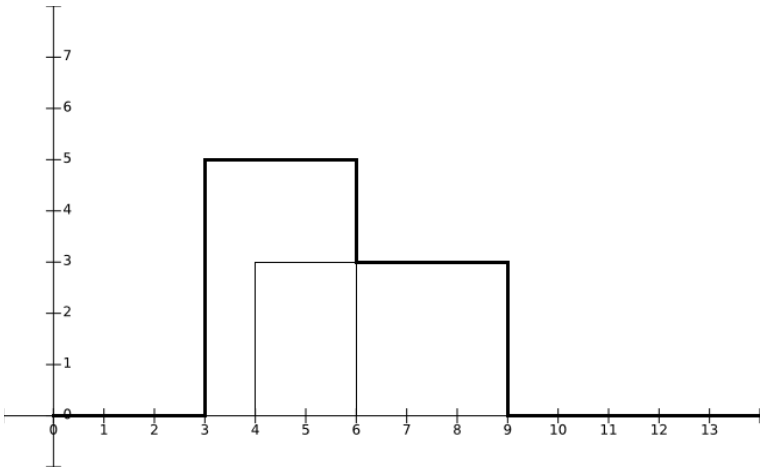


2.1.1 Combinación de dos líneas de horizonte

Cuando hay más de un edificio, la línea de horizonte debe representar todos los cambios de altura. Así, para dos edificios `(3,6,5)` y `(4,9,3)`:



la línea de horizonte resultante debería ser `[(3,5),(6,3),(9,0)]`:

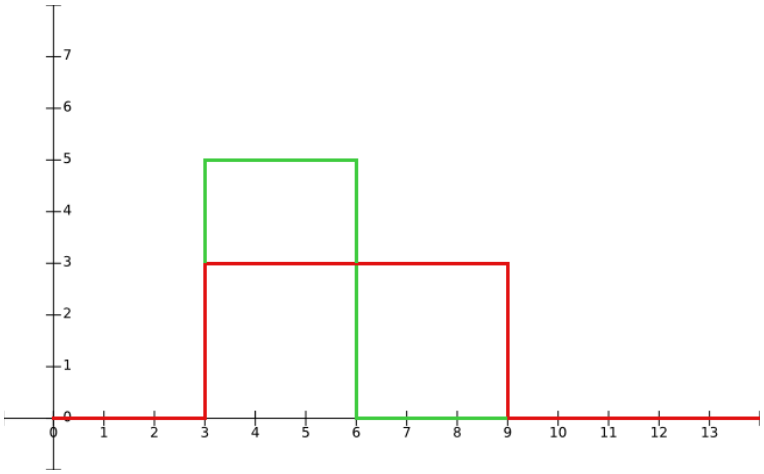


A continuación vamos a ver cómo se ha de realizar esta combinación de dos líneas de horizonte. La idea fundamental es comparar el primer punto de ambas líneas de horizonte y obtener la coordenada x y la altura del siguiente punto de la línea de horizonte resultado. El proceso continuará hasta que se hayan consumido todos los puntos de una de las líneas de horizonte.

Distinguiremos dos casos fundamentales según sea el siguiente punto de cada una de las dos líneas de horizonte a combinar:

Las dos líneas de horizonte comienzan en la misma coordenada x

Estaríamos ante un caso como el descrito en la siguiente figura:



Tenemos dos líneas de horizonte:

- La línea verde, representada por la lista `[(3,5),(6,0)]`
- La línea roja, representada por la lista `[(3,3),(9,0)]`

El siguiente punto de la línea de horizonte resultado tendrá, obviamente, coordenada x 3 y altura 5, por ser la mayor de las dos. Es decir: la coordenada x del siguiente punto será la coordenada x de cualquiera de los dos puntos de la línea de horizonte, mientras que la altura será la mayor de ambas.

Además, ninguno de estos puntos iniciales va a volver a influir en la elección de la coordenada x de los siguientes puntos, por lo que se pueden consumir ambos y continuar con el análisis.

Las dos líneas de horizonte comienzan en distinta coordenada x

Tras haber consumido los dos primeros puntos de ambas líneas de horizonte en el caso anterior, ahora tendríamos que:

- El resto de la línea verde estaría representado por la lista `[(6,0)]`
- El resto de la línea roja estaría representado por la lista `[(9,0)]`

Parece lógico pensar que el siguiente punto de la línea de horizonte resultado tendrá como coordenada x la menor de ambas (6 en este caso). Sin embargo, la altura debería ser 3 según se ve claramente en el dibujo, pero la altura de los primeros puntos del resto de las líneas es 0. ¿Cómo podemos obtener ese 3?

Si nos fijamos, la coordenada x del nuevo punto de la línea de horizonte resultado lo obtenemos de la línea verde, mientras que la altura (3) viene dada por la línea roja, ya que es la altura del último punto de dicha línea de horizonte.

Así pues, la función de combinación debe recordar cuál es la altura del último punto consumido de cada línea de horizonte, para poder así calcular correctamente la altura del siguiente punto de la línea de horizonte resultado. Originalmente todas las líneas de horizonte comienzan en altura 0, por lo que ese será el valor con el que habrá que inicializar esas variables.

El cálculo de la altura se realiza, en general, de la siguiente forma:

1. Elegimos el punto que tenga la coordenada x más pequeña.
2. Comparamos la altura de ese punto con la altura del último punto consumido de la otra línea de horizonte y nos quedamos con el valor mayor.
3. Consumimos el punto elegido.

Aplicando este algoritmo al ejemplo:

1. Elegimos el punto `(6,0)`.
2. Comparamos su altura (0) con la altura del último punto consumido de la otra línea de horizonte (3) y nos quedamos con el valor mayor (3).
3. Consumimos el punto elegido, con lo que el resto de la línea de horizonte verde sería `[]` (la lista vacía).

Una vez que se ha consumido totalmente una de las dos líneas de horizonte, simplemente habrá que añadir el resto de la otra a la salida sin que haga falta realizar ningún tipo de cálculo adicional.

Es posible que en algunas ocasiones se produzcan dos puntos consecutivos con igual altura. Esto puede solucionarse sin más que comparar la altura del posible nuevo punto a generar con la del último generado y en caso de que sean la misma no generar el nuevo punto.

2.2 Tarea 1: Implementación en Haskell

En primer lugar vamos a ver una serie de indicaciones generales para orientar la primera tarea de la práctica, que consistirá en hacer una versión en Haskell (lenguaje funcional) de la práctica.

2.2.1 Tipos de datos

Los tipos de datos que vamos a utilizar en la práctica son los siguientes:

```
type Edificio = (Int,Int,Int)
type Coordenada = (Int,Int)
type Skyline = [Coordenada]
```

que representan, respectivamente, un edificio en forma de tripleta de enteros, una coordenada de una línea de horizonte (coordenada x y altura) y una línea de horizonte (o skyline).

2.2.2 Funciones a implementar

Habrá que implementar cuatro funciones, respetando los nombres que aquí se indican:

1. `resuelveSkyline`: se trata de la función principal que implementa el algoritmo del Skyline mediante un esquema Divide y Vencerás. Debe recibir un único parámetro consistente en una lista de edificios y deberá devolver un valor de tipo `Skyline` conteniendo la línea de horizonte que dibuja la lista de edificios de entrada
2. `edificioAskyline`: es la función que será llamada por `resuelveSkyline` cuando el esquema Divide y Vencerás no realiza nuevas subdivisiones del problema. Es decir, cuando se encuentra con el caso trivial. Recibe un único edificio y devuelve su línea de horizonte.

3. `divide`: es la función que será llamada por `resuelveSkyline` para realizar la división de la lista de edificios en dos mitades de igual tamaño (o, si la longitud es impar, una lista contendrá un edificio más que la otra). Recibe una lista de edificios y devuelve una tupla con las dos listas de edificios en las que se va a dividir el problema.
4. `combina`: es la función que será llamada por `resuelveSkyline` para combinar las soluciones parciales de los dos subproblemas. Recibirá dos líneas de horizonte y las combinará en una única línea utilizando el proceso descrito en la sección 2.1.1.

Si cualquiera de estas funciones requiriese de funciones auxiliares, éstas habrán de implementarse como subfunciones (utilizando para ello la cláusula `where`).

2.3 Tarea 2: Implementación en Prolog

A continuación presentamos las indicaciones para orientar la segunda tarea de la práctica, que consistirá en hacer una versión en Prolog (lenguaje lógico) de la práctica.

2.3.1 Tipos de datos

Se eligen como opciones de representación de términos (son términos, no son predicados, luego solo aparecen en argumentos de predicados):

- para edificio: `ed(`X_1`,`X_2`,H)`, siendo X_1 y X_2 las coordenadas inicial y final del edificio en el horizonte y `H` la altura
- para coordenadas: `c(X,Y)`, con `X` la coordenada del horizonte e `Y` la altura

Se puede comprobar que si se necesita tener una lista de edificios conocidos a priori, podría usarse (aunque no es el caso en este enunciado, ya que se pasan como parámetro en la pregunta) un conjunto de reglas (hechos) con la descripción de los edificios. Por ejemplo:

```
edificio(ed(3,6,5)).
edificio(ed(4,9,3)).
edificio(ed(8,11,2)).
```

2.3.2 Predicados a implementar

Habrá que implementar cuatro predicados, al menos, respetando los nombres que aquí se indican:

1. `resuelveSkyline`: se trata del predicado principal que implementa el algoritmo del Skyline mediante un esquema Divide y Vencerás. Debe recibir un argumento consistente en una lista de edificios y devolverá una lista de coordenadas que describen la línea de horizonte que dibuja la lista de edificios de entrada.

```
resuelveSkyline([ed(3,6,5),ed(4,9,3),ed(8,11,2),ed(10,12,4)], D).

D = [c(3,5),c(6,3),c(9,2),c(10,4),c(12,0)].
```

2. `edificioAskyline`: es el predicado que será llamado o utilizado por `resuelveSkyline` cuando el esquema Divide y Vencerás no realiza nuevas subdivisiones del problema. Es decir, cuando se encuentra con el caso trivial. Recibe un único edificio y devuelve su línea de horizonte.
3. `divide`: es el predicado que será llamado o utilizado por `resuelveSkyline` para realizar la división de la lista de edificios en dos mitades de igual tamaño (o, si la longitud es impar, una lista contendrá un edificio más que la otra). Recibe una lista de edificios y devuelve las dos listas de edificios en las que se va a dividir el problema.

4. `combina`: es el predicado que será llamado o utilizado por `resuelveSkyline` para combinar las soluciones parciales de los dos subproblemas. Recibirá dos líneas de horizonte y las combinará en una única línea utilizando el proceso descrito en la sección 2.1.1.

Cuando cualquiera de estos predicados requiera predicados auxiliares, éstos habrán de implementarse.

3. Cuestiones sobre la práctica

Para responder a estas cuestiones deberá comparar la práctica programada en **Haskell** con la de **Prolog** y la que el Equipo Docente proporciona programada en **Java**, un lenguaje del paradigma de Programación Orientada a Objetos.

La respuesta a estas preguntas es optativa. Sin embargo, si el estudiante no responde a estas preguntas, la calificación de la práctica sólo podrá llegar a 5 puntos sobre 10.

1. El programa implementado puede transformar una lista de edificios en una línea de horizonte definida como una lista de coordenadas. Ahora proponemos una mejora al programa que permita visualizar el skyline mediante una serie de líneas en las que un asterisco (`'*'`) representará un edificio y un espacio (`' '`) la ausencia de edificio.

   Por ejemplo, para el skyline que forman los edificios

   ```
   [(3,6,5),(4,9,3),(8,11,2),(10,12,4)]                    (notación Haskell)
   ```

   cuya línea de horizonte (creada por `resuelveSkyline`) es:

   ```
   [(3,5),(6,3),(9,2),(10,4),(12,0)]                       (notación Haskell)
   ```

   el dibujo del skyline debería ser

   ```
      ***
      ***    **
      ****** **
      *********
      *********
   -------------
   ```

   Para ello vamos a realizar el siguiente proceso que transforma una línea de horizonte (en forma de lista de coordenadas) en una cadena de caracteres con el dibujo del skyline:

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

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

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

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.

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**?

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.

4. Documentación a entregar

Cada estudiante deberá entregar la siguiente documentación a su tutor de prácticas:

- Código fuente en **Haskell** que resuelva el problema planteado. Para ello se deberá entregar el fichero *Skyline.hs*, con las funciones descritas en este enunciado, así como todas las funciones auxiliares que sean necesarias.
- Código fuente en **Prolog** que resuelva el problema planteado. Para ello se deberá entregar el fichero *Skyline.pl*, con los predicados descritos en este enunciado, así como todos los predicados auxiliares que sean necesarios.
- Una memoria con:
    - Una pequeña descripción de las funciones/predicados programadas/os.
    - Las respuestas a las cuestiones sobre la práctica.