Control de inventario Haskell

Sara Cubero García-Conde

Índice:

1. Análisis de la práctica.

La práctica se basa en la creación de un programa en Haskell para gestionar el stock de productos en una tienda utilizando un Trie como estructura de datos. El programa debe permitir al usuario almacenar, modificar y consultar el stock, además de listar productos en base a una cadena de caracteres. Se implementarán cuatro operaciones: createStock, updateStock, retrieveStock y listStock. La estructura de Trie se compone de un nodo raíz sin valor, nodos intermedios con caracteres y nodos hoja con elementos asociados a índices formados por cadenas de caracteres desde la raíz hasta el nodo hoja.

2. Descripción de solución.

Para resolver la práctica se ha llevado a cabo el desarrollo de las siguientes funciones de la StockControl.hs:

createStock: Esta función devuelve un stock vacío, simplemente crea un nodo raíz (ROOTNODE) con una lista vacía de hijos.

retrieveStock: Esta función devuelve el número de unidades de un producto en el stock. Si el producto no está en el stock, devuelve -1.

La función principal retrieveStock utiliza una función auxiliar **retrieveStock'** para buscar el producto en los nodos del árbol de stock. Si encuentra el nodo de información (INFONODE) correspondiente al producto, devuelve el número de unidades almacenadas en ese nodo.

updateStock: Esta función modifica el valor asociado a un producto en el stock, actualizando la cantidad de unidades almacenadas para ese producto. Si el producto no está en el stock, lo agrega. La función principal updateStock utiliza una función auxiliar **updateStock'** para actualizar el stock en función del producto y la cantidad de unidades proporcionadas.

La función auxiliar **updateStock'** realiza las modificaciones necesarias en el árbol de stock, agregando o actualizando nodos según corresponda.

listStock: Esta función devuelve una lista de pares (producto, existencias) del catálogo que comienzan con el prefijo proporcionado. La función principal listStock utiliza una función auxiliar listStock' para recorrer el árbol de stock y construir la lista de pares (producto, existencias) que cumplen con el criterio del prefijo. La función **isPrefixOf** verifica si un prefijo está contenido al inicio de una cadena dada.

bt (ya implementada por el equipo docente): Esta función es una implementación genérica de backtracking (búsqueda hacia atrás) que toma una función de éxito es, una función de continuación c y un nodo inicial n. La función de éxito ‘s’ verifica si un nodo dado es una solución válida, y la función de continuación ‘c’ genera los sucesores de un nodo dado. La función bt explora el espacio de búsqueda utilizando backtracking y devuelve una lista de soluciones válidas.

3. Cuestiones sobre la practica

1 (1’5 puntos). Supongamos una implementación de la práctica (usando la misma estructura aquí presentada) en un lenguaje no declarativo (como Java, Pascal, C…). Comente qué ventajas y qué desventajas tendría frente a la implementación en Haskell. Relacione estas ventajas desde el punto de vista de la eficiencia con respecto a la programación y a la ejecución. ¿Cuál sería el principal punto a favor de la implementación en los lenguajes no declarativos? ¿Y el de la implementación en Haskell? Justifique sus respuestas.

Algunas de las ventajas de la implementación de lenguajes no declarativos (imperativos) como Haskell son: La expresividad, ya que nos permiten un código más conciso y elegante para problemas complejos. La Inmutabilidad y ausencia de efectos secundarios, facilitando el razonamiento del código y la reduciendo probabilidad de errores. Algunas de las ventajas de los lenguajes no declarativos son: suelen ser más rápidos en tiempo de ejecución, permite un control más preciso de la estructura de datos y gestión de memoria. Son más familiares para la implementación y tienen una amplia gama de herramientas y bibliotecas disponibles.

2 (1’5 puntos). Indique, con sus palabras, cómo afecta el predicado predefinido no lógico corte (!) al modelo de computación de Prolog. ¿Cómo se realizaría este efecto en Haskell? ¿Considera que sería necesario el uso del corte en una implementación en Prolog de esta práctica? Justifique sus respuestas.

El corte en Prolog es como un interruptor que detiene la búsqueda de soluciones en una parte especifica del programa. Una vez que encuentras lo que buscabas es capaz de parar la búsqueda en ese punto y mejorar el rendimiento. En Haskell no existe el elemento de corte, pero probablemente puedas lograr algo parecido con otras funciones, por ejemplo, la función de backtraking tiene cierta relación ya que si encuentra la solución puede detenerse y devolver la solución. Si la implementación estuviera echa en Prolog, creo que sería de gran utilidad usar el corte para ahorrar tiempo en la búsqueda de soluciones principalmente en las consultas, pero no creo que fuera imprescindible.

3 (1 punto). Indique qué clases de constructores de tipos (ver capítulo 5 del libro de la asignatura) se han utilizado para definir el tipo Stock y para definir los nodos del backtracking utilizado en la función listStock. Justifique sus respuestas.

Tipo Stock: El tipo Stock se define utilizando un tipo de datos algebraico (ADT) con tres constructores: ROOTNODE, INNERNODE y INFONODE. Estos constructores representan diferentes tipos de nodos en la estructura Trie utilizada para almacenar el stock. data Stock = ROOTNODE [Stock] | INNERNODE Char [Stock] | INFONODE Int

El constructor ROOTNODE representa el nodo raíz y contiene una lista de nodos hijos. INNERNODE representa un nodo interno con un carácter y una lista de nodos hijos. INFONODE representa un nodo hoja que contiene la cantidad de unidades de un producto en stock.

Nodos del backtracking en la función listStock: La función listStock utiliza una función auxiliar llamada listStock', que explora la estructura Trie y encuentra todos los productos cuyos nombres comienzan con una cadena de caracteres específica. La función listStock' utiliza la misma estructura de datos Stock para representar y procesar los nodos en la estructura Trie durante la búsqueda.