Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática

Grado en Tecnologías de la Información

Práctica curso 2022-2023

Enunciado y orientaciones

1. Presentación del problema

Según el diccionario, se define stock como aquel conjunto de mercancías o productos que se tienen almacenados en espera de su venta o comercialización. Desde el punto de vista de los Tipos Abstractos de Datos, podríamos pensar en un stock como en una estructura indexada que almacena valores enteros no negativos (las unidades almacenadas de cada producto) a los cuales se accede mediante índices que se expresan con una cadena de caracteres, que representarían los productos.

Este tipo de estructuras indexadas suelen implementarse mediante tablas hash, las cuales se estudiarán en la asignatura de Programación y Estructuras de Datos Avanzadas, que sigue a la nuestra durante el primer cuatrimestre de segundo curso. Así pues, en nuestra práctica emplearemos otras estructuras para implementar un stock.

En esta práctica consideraremos dos implementaciones alternativas de un stock en la que almacenaremos los productos y el número de unidades que están almacenadas. La primera implementación utilizará una secuencia de pares `<índice,valor>`, mientras que la segunda empleará un árbol general en el cual los valores se almacenarán en los nodos hoja y el índice que lleva a cada valor se obtendrá mediante el recorrido desde la raíz del árbol al nodo hoja correspondiente.

2. Enunciado de la práctica

La práctica consiste en:

(i) Razonar e implementar dos soluciones alternativas para el tipo de datos abstracto stock, una basada en secuencias y otra basada en árboles generales. **La implementación utilizará los tipos de datos programados por el Equipo Docente**.
(ii) Implementar un programa que aplique ambas implementaciones.
(iii) Ejecutar el programa anterior sobre juegos de prueba empleando ambas implementaciones y realizar estudios comparativos del tiempo de ejecución de cada una de ellas.

2.1. Stock: Tipo de Datos Abstracto

El tipo de datos abstracto Stock que vamos a implementar funciona como una estructura indexada mediante índices formados por cadenas de caracteres. Así pues, este tipo abstracto de datos incluye las operaciones descritas en la siguiente interfaz:

```
/* Representa un Stock en el que se indexan valores enteros
 * bajo un indice formado por una cadena de caracteres
 */
public interface StockIF {

      /* Devuelve el valor indexado bajo el indice p.
       * @PRE: p != ""
       * Si no existe un valor indexado bajo el indice p,
       * devuelve el valor -1.
       */
       public int retrieveStock(String p);

      /* Indexa el valor u bajo el indice p.
       * @PRE: p >= 0
       * Si ya habia un valor bajo el mismo indice,
       * el nuevo valor substituye al anterior.
```

```
        */
        public void updateStock(String p, int u);

        /* Devuelve una secuencia de todos los pares <p,u>
         * presentes en el stock tales que:
         * -El valor indexado bajo el índice p es u
         * -El índice p comienza por la cadena prefix
         * Además, la secuencia deberá estar ordenada según
         * el orden alfabético de los productos
         */
        public SequenceIF<StockPair> listStock(String prefix);
}
```

Las implementaciones de `StockIF` utilizarán la siguiente clase:

```
/* Representa un par <producto,unidades> del Stock */
public class StockPair {
        private String producto;
        private int unidades;

        /* Constructor */
        public StockPair(String p, int u) {
                this.producto = p;
                setUnidades(u);
        }

        /* Modifica el número de unidades del par
         * @PRE: u >= 0
         */
        public void setUnidades(int u) {
                this.unidades = u;
        }

        /* Devuelve el nombre del producto */
        public String getProducto() {
                return this.producto;
        }

        /* Devuelve el número de unidades */
        public int getUnidades() {
                return this.unidades;
        }

        /* Construye una cadena para representar el par */
        public String toString() {
                return "("+this.producto+","+Integer.toString(this.unidades)+")";
        }
}
```

Preguntas teóricas de la sección 2.1

Antes de continuar, reflexiona y responde a la siguiente pregunta:

1. ¿Qué tipo de secuencia sería la más adecuada para devolver el resultado de la operación `listado(prefix)`? ¿Qué consecuencias tendría el uso de otro tipo de secuencias? Razona tu respuesta.

2.2. Implementación 1: *secuencia ordenada de pares indexados*

Esta implementación consiste en utilizar una secuencia ordenada de pares de la clase `StockPair` como estructura de datos de soporte para almacenar los elementos bajo sus correspondientes índices.

Preguntas teóricas de la sección 2.2

Antes de implementar esta solución, reflexiona y responde a las siguientes preguntas:

1. ¿Qué tipo de secuencia sería la adecuada para realizar esta implementación? ¿Por qué? ¿Qué consecuencias tendría el uso de otro tipo de secuencias?
2. ¿Cuál sería el orden adecuado para almacenar los pares en la secuencia? ¿Por qué? ¿Qué consecuencias tendría almacenarlos sin ningún tipo de orden?
3. ¿Afectaría el orden a la eficiencia de alguna operación prescrita por `StockIF`? Razona tu respuesta.

2.3. Implementación 2: *árbol general*

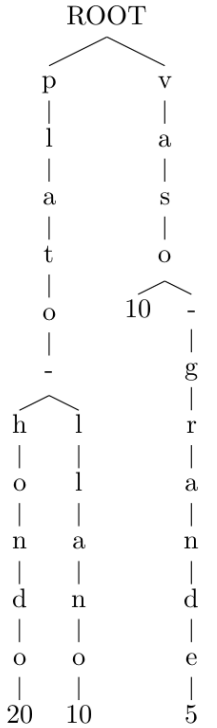
Para esta implementación, utilizaremos un árbol general que contiene tres tipos de nodos diferentes:

1. El nodo raíz, que sólo contendrá la lista de sus nodos hijo.
2. Los nodos internos, que almacenarán un carácter y la lista de sus nodos hijo.
3. Los nodos raíz, que almacenarán un valor entero.

De esta forma, el nodo raíz que almacena el número de unidades del producto p se localiza siguiendo desde la raíz los nodos que contienen los caracteres del producto p, que recordemos se representa mediante una cadena de caracteres.

Por ejemplo, a la derecha tenemos una representación de un stock en el que se almacenan cuatro productos con sus respectivas unidades:

- “plato-hondo”, con 20 unidades
- “plato-llano”, con 10 unidades
- “vaso”, con 10 unidades
- “vaso-grande”, con 5 unidades



En esta implementación, la operación `updateStock` será la encargada de añadir tantos nodos como sean necesarios para construir el camino desde la raíz hasta el nodo que deberá almacenar el número de unidades del producto.

Esta estructura de datos tiene un nombre propio: Trie, que fue acuñado en 1960 a partir de la sílaba central de la palabra *retrieval*.

Para representar esta estructura, utilizaremos una serie de clases que representarán el contenido de los diferentes tipos de nodos que podemos encontrarnos en un Trie:

1. Clase `Node`, que nos sirve para representar un nodo de la estructura. Se trata de una clase abstracta (es decir, no podremos instanciar ningún objeto de esta clase) de la que heredarán los tres tipos de nodos existentes. Prescribe una única operación `getNodeType()`, común a todos los nodos, que devuelve el tipo de nodo.
2. Clase `NodeRoot`, que nos sirve para representar el nodo raíz. No contiene ningún tipo de información y la operación `getNodeType()` devuelve el valor `ROOT`.
3. Clase `NodeInner`, que nos sirve para representar los nodos internos. Contiene un carácter que se usará para formar las cadenas que son los índices de la estructura y la operación `getNodeType()` devuelve el valor `INNER`.
4. Clase `NodeInfo`, que nos sirve para representar los nodos hoja. Contiene un entero que almacenará el número de unidades del producto que se forma por la concatenación de los caracteres contenidos en los nodos internos que se recorren en el camino desde la raíz hasta el nodo hoja. La operación `getNodeType()` devuelve el valor `INFO`.

Todas estas clases se dan ya programadas.

Preguntas teóricas de la sección 2.3

Nuevamente, antes de implementar esta solución, reflexiona y responde a las siguientes preguntas:

1. Utilizando papel y lápiz inserta más pares producto-unidades en el árbol del ejemplo. Compara al menos dos formas de colocar los hijos de un nodo y comenta qué ventajas tendría cada una de ellas. Razona tu respuesta.
2. La operación `listStock(String prefix)` nos indica que la secuencia de elementos de tipo `StockPair` que devuelve ha de estar ordenada de forma creciente según los productos. ¿Alguna de las opciones de colocar los hijos de un nodo de la pregunta anterior resulta más conveniente para mejorar la eficiencia de esta operación? Razona tu respuesta.

3. Diseño de la práctica

A continuación, veremos el diseño de clases de la práctica para su implementación. Para las clases ya programadas se explica su funcionamiento. Para las clases que deban ser completadas por los/las estudiantes se listan las operaciones que contienen y cuál será su comportamiento esperado.

3.1. StockIF.java

Se trata de la interfaz presentada en la sección anterior que especifica las operaciones de los stocks y **no debe modificarse**.

3.2. StockPair.java

Esta clase, también presentada en la sección anterior, representa los pares `<producto,unidades>`, siendo el índice una cadena de caracteres no vacía y las unidades un entero mayor o igual que cero. **No debe modificarse**.

3.3. StockSequence.java

Esta clase implementa la interfaz `StockIF.java` e implementará un stock tal y como se ha descrito en el apartado 2.2, empleando una secuencia de objetos de la clase `StockPair`. Junto a este enunciado se adjunta el esqueleto de esta clase.

3.4. StockTree.java

Esta clase implementa la interfaz `StockIF.java` e implementará un stock tal y como se ha descrito en el apartado 2.3, empleando un árbol general con los nodos organizados según se ha explicado en dicha sección. Junto a este enunciado se adjunta el esqueleto de esta clase.

3.5 Clases Node.java, RootNode.java, InnerNode.java e InfoNode.java

Estas clases se utilizan para representar los diferentes nodos del árbol general que representará un stock según lo descrito en el apartado 2.3. Se utilizarán desde la clase `StockTree.java` y **no deben modificarse**.

3.5. Main.java

Esta clase contiene el programa principal y se da completamente terminada por parte del Equipo Docente y **no debe modificarse**. El funcionamiento es el siguiente:

1. El programa requiere tres argumentos:
    a. El primero es una cadena de caracteres que puede ser `SEQUENCE` o `TREE`, que indica cuál de las dos implementaciones propuestas de los stocks se va a emplear.

    b. El segundo es la ruta del fichero de entrada del programa. Un fichero de entrada será un fichero de texto en el que cada línea contiene una de las siguientes operaciones:

        - `compra producto n`: que representará una operación de compra de n unidades del producto indicado. Se asumirá que el producto es una cadena de caracteres no vacía que no contiene espacios y el número n es mayor o igual que cero.
        - `venta producto n`: que representará una operación de venta de n unidades del producto indicado. Se asumirá que el producto es una cadena de caracteres no vacía que no contiene espacios y el número n es mayor que cero.
        - `unidades producto`: que representará una operación de consulta del número de unidades del producto indicado almacenadas en el stock. Se asumirá que el producto es una cadena de caracteres no vacía.
        - `listado prefijo`: que representará una operación de consulta del listado de pares producto-unidades almacenados en el stock tales que producto comienza por el prefijo indicado, el cual podrá ser la cadena vacía (en cuyo caso sólo aparecería la palabra `listado` en la línea correspondiente).

    c. El tercero es la ruta del fichero de salida generado por el programa. La salida generada se explica en el punto 3.

2. Se comprueba que se han proporcionado adecuadamente los tres parámetros. En caso contrario, se lanza un mensaje de error y termina el programa. En el caso del primer argumento, se comprueba que la cadena de caracteres es `SEQUENCE` o `TREE` y se crea un stock vacío con la implementación correspondiente. Para el segundo argumento, se comprueba que existe el fichero de entrada y puede leerse. Por último, para el tercer argumento se comprueba que existe la carpeta donde se quiere escribir el fichero de salida y que dicho fichero puede ser escrito (o sobreescrito si ya existía).

3. El programa lee línea a línea el fichero de entrada, de manera que se aplican en orden las

operaciones correspondientes sobre el stock. Tras ejecutar cada operación, se escribe en el fichero de salida la línea leída, a continuación escribirá la cadena "`: `" y el resultado de la operación siguiendo este formato:

- `compra producto n`: se indicará que la operación de compra ha modificado el stock, ya que no hay errores asociados a estas operaciones.
- `venta producto n`: si la operación de venta puede realizarse (el producto está en stock y hay unidades suficientes), se indica que la operación de venta se ha podido realizar. En caso contrario la operación no se realizará (el stock no se modificará) y se indicará el motivo.
- `unidades producto`: si el producto está en el stock, se devolverá una cadena indicando el número de unidades almacenadas. Si no estuviera, se indicará que el producto nunca estuvo en el stock.
- `listado prefijo`: se devolverá una cadena conteniendo la secuencia, ordenada según el orden alfabético de los productos, de los pares producto-unidades almacenados en el stock tales que el producto comienza por la cadena prefijo especificada.

4. El programa termina mostrando el tiempo de ejecución en milisegundos del paso anterior.

4. Implementación.

Se deberá realizar un programa en Java llamado **`eped2023.jar`** que contenga todas las clases anteriormente descritas completamente programadas. Todas ellas se implementarán en un único paquete llamado:

`es.uned.lsi.eped.pract2022_2023`

Para la implementación de las estructuras de datos de soporte **se deberán utilizar las interfaces y las implementaciones proporcionadas por el Equipo Docente** de la asignatura. **No se permite utilizar otras estructuras propias de java**.

A través del Curso Virtual, el Equipo Docente proporcionará el código de todas las clases (total o parcialmente) implementado según la descripción que se ha hecho en este documento.

Esto significa que la lectura de los parámetros de entrada, lectura de los ficheros y salida del programa ya está programada por el Equipo Docente y no será necesario realizar ni modificar nada sobre estos temas.

5. Ejecución y juegos de prueba.

Para la ejecución del programa se deberá abrir una consola y ejecutar:

```
java –jar eped2023.jar <implementacion> <entrada> <salida>
```

siendo:

- **`<implementacion>`** SEQUENCE o TREE según el tipo de implementación a utilizar
- **`<entrada>`** nombre del fichero de entrada con las operaciones a realizar
- **`<salida>`** nombre del fichero de salida

El Equipo Docente proporcionará, a través del curso virtual, unos juegos de prueba para comprobar

el correcto funcionamiento del programa. Si se supera el juego de pruebas privado para tutores (que será diferente del juego de pruebas para estudiantes), la práctica tendrá una calificación mínima de 5 puntos, a falta de la evaluación del estudio empírico del coste y de las preguntas teóricas presentes en este enunciado.

6. Estudio del coste.

Queremos estudiar empíricamente el tiempo de ejecución de cada implementación dependiendo del tamaño del problema.

Para realizar esta tarea, el estudiante tiene libertad para generar su propio juego de pruebas, explicando el diseño de forma razonada en función de su uso para medir tiempos de ejecución y reportando sus resultados experimentales.

Preguntas teóricas de la sección 6

1. Defina el tamaño del problema y calcule el coste asintótico temporal en el caso peor de las operaciones `updateStock` y `retrieveStock` para ambas implementaciones.
2. Compare el coste asintótico temporal obtenido en la pregunta anterior con los costes empíricos obtenidos. ¿Coincide el coste calculado con el coste real?

7. Documentación y plazos de entrega.

La práctica supone un 20% de la calificación de la asignatura, y es necesario aprobarla para superar la asignatura. Además, será necesario obtener, al menos, un 4 sobre 10 en el examen presencial para que la calificación de la práctica sea tenida en cuenta de cara a la calificación final de la asignatura.

Los Centros Asociados organizarán sesiones de prácticas en las que tutores y tutoras monitorizarán y orientarán a los estudiantes en su realización de la práctica. La asistencia a estas sesiones no es obligatoria, pero se recomienda encarecidamente asistir a ellas. La interacción con el tutor o tutora en esas sesiones presenciales puede influir en la calificación de la práctica. Estas sesiones son organizadas por los Centros Asociados teniendo en cuenta sus recursos y el número de estudiantes matriculados, por lo que en cada Centro las fechas serán diferentes. Cada estudiante deberá, por tanto, dirigirse a su tutor/a para conocer las fechas de celebración de estas sesiones.

De igual modo, **el plazo y forma de entrega son establecidos por cada tutor/a de forma independiente en cada Centro Asociado, por lo que deberán ser consultados**.

La documentación que debe entregar cada estudiante consiste en:

- Memoria de práctica, en la que se deberán responder a las preguntas teóricas y se incluirá el diseño y resultados del estudio empírico de costes.
- Implementación en Java de la práctica, de la cual se deberá aportar **únicamente el directorio src del proyecto java**.
- Juego de pruebas diseñado para el estudio empírico de costes.

Adicionalmente, se deberá entregar **obligatoriamente** una copia de la práctica al Equipo Docente a través del enlace que se habilitará en el Curso Virtual en las siguientes fechas:

- Del 1 de mayo al 16 de junio de 2023 para la convocatoria de junio
- Del 15 de julio al 10 de septiembre de 2023 para la convocatoria de septiembre

¡¡¡ATENCIÓN!!! La entrega de la copia de la práctica se realiza al Equipo Docente, no al tutor o tutora que se encargará de su evaluación. **Si no se entrega la práctica al tutor o tutora, ésta no se evaluará y constará como suspensa**.

8. Calificación.

La calificación de la práctica será otorgada por el tutor o tutora, en función de las respuestas a las preguntas teóricas, la superación del juego de pruebas público y del juego de pruebas privado y el diseño y resultados del estudio empírico de costes. También se podrá tener en cuenta el grado de participación activa en las sesiones de tutorización relativas a la práctica. La puntuación se distribuye de la siguiente manera:

- 5 puntos en función del código de la práctica: corrección, presentación, documentación. **Para aprobar es imprescindible que el código supere el juego de pruebas privado que sólo tendrán los tutores y tutoras**.
- 2 puntos en función del estudio empírico del coste. Se valorará el diseño del juego de pruebas y la presentación y análisis de los resultados empíricos.
- 3 puntos en función de las respuestas a las preguntas teóricas. Se valorará la corrección de las respuestas, su justificación y el nivel de detalle.
- Hasta un punto adicional en función de la participación e interacción en las tutorías del centro orientadas a la práctica.

La práctica es un trabajo individual de cada estudiante. **Tanto el código como las respuestas a las preguntas teóricas y el diseño del juego de pruebas individual para el estudio empírico del coste deben ser originales de cada estudiante**. Si se detectan casos de plagio en cualquiera de estos aspectos se suspenderá a los estudiantes involucrados y podrán ser objeto de sanciones por parte de la universidad.