Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática

Grado en Tecnologías de la Información

Práctica curso 2020-2021

Enunciado y orientaciones

1. Presentación del problema

Con la llegada de las vacunas del COVID, se organiza una estrategia de vacunación que tiene en cuenta no sólo el orden de llegada de los pacientes al centro de vacunación, sino también su pertenencia a grupos prioritarios. El criterio principal para ordenar la vacunación es considerar la prioridad de cada paciente en primer lugar (es decir, pacientes con una prioridad más alta serán atendidos primero) y ante pacientes con la misma prioridad atender primero a quienes llegaron antes.

La estructura de datos abstracta que se corresponde con esta situación es la *cola de prioridad*: una cola modificada en la que, en lugar de salir los elementos en el orden en el que entraron, primero saldrán los más prioritarios (ordenados en el orden de llegada).

En esta práctica consideraremos dos implementaciones alternativas para una cola de prioridad. La primera, llamada *bucket queue*, utiliza una estructura de datos secuencial para almacenar los diferentes niveles de prioridad. La segunda consiste en almacenar los niveles de prioridad utilizando un árbol binario de búsqueda. Una vez implementadas ambas opciones, las aplicaremos a dos casos diferentes, y haremos un estudio comparado de la eficiencia de ambas implementaciones en cada uno de los casos.

2. Enunciado de la práctica

La práctica consiste en

(i) Razonar e implementar dos soluciones alternativas para la estructura de datos abstracta cola de prioridad, una basada en secuencias y otra basada en árboles de búsqueda binaria. 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. Cola de prioridad: Estructura de Datos Abstracta

La estructura de datos abstracta *cola de prioridad* funciona como una cola modificada en la que los elementos no salen estrictamente en el mismo orden que entraron, sino que tienen asociada una prioridad y salen antes los de mayor prioridad. Si dos elementos tienen el mismo orden de prioridad, sale antes el que entró primero, como en una cola convencional. Esta estructura de datos incluye las siguientes operaciones:

`getFirst`: devuelve el elemento más prioritario (y que entró primero) de la cola con prioridad.
`enqueue`: inserta un elemento con una cierta prioridad en la cola con prioridad.
`dequeue`: elimina el elemento más prioritario (y que entró primero) de la cola con prioridad.

Utilizaremos la siguiente interfaz:

```
/* Representa una cola de prioridad, en la que los elementos    *
 * salen según su nivel de prioridad y su orden de entrada.     */
public interface PriorityQueueIF<E> extends SequenceIF<E> {
```

```
    /* Devuelve el elemento más prioritario de la cola con prioridad *
     * @Pre: !isEmpty()                                              *
     * @return: el elemento más prioritario de la cola con prioridad */
     public E getFirst();

    /* Inserta un elemento con una cierta prioridad en la cola con prioridad *
     * @param elem: el elemento a encolar (añadir).                          *
     * @param prior: prioridad                                                */
    public void enqueue(E elem, int prior);

    /* Elimina el elemento más prioritario de la cola con prioridad *
     * @Pre !isEmpty();                                             */
    public void dequeue();
}
```

Nótese que las colas con prioridad son un tipo de secuencias de elementos, de modo que esta interfaz hereda de la interfaz de las secuencias. Esto implica que cualquier implementación de `PriorityQueueIF` deberá incluir las operaciones de las secuencias además de las tres operaciones especificadas en la interfaz. Por otra parte, hemos definido el nivel de prioridad como un número entero, de manera que **un elemento será más prioritario cuanto mayor sea dicho número entero**. No obstante, en general las prioridades podrían ser de un tipo genérico que tenga definida una relación de orden que establezca el nivel de prioridad. Por ejemplo, un tipo enumerado con los valores “baja”, “media” y “alta”, de manera que alta>media>baja.

2.2. Implementación 1: *Bucket Queue*

Esta implementación consiste en utilizar una secuencia ordenada de colas, cada una de ellas con los elementos de un mismo nivel de prioridad. De esta forma, el primer elemento de la secuencia es la cola que contiene los elementos con mayor prioridad.

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. Describe el funcionamiento de las operaciones `getFirst`, `enqueue` y `dequeue` con esta estructura de datos.
3. Describe el funcionamiento del iterador de la cola con prioridad en esta implementación.

2.3. Implementación 2: Árbol Binario de Búsqueda

Esta implementación consiste en utilizar un árbol binario de búsqueda (ABB) ordenado según la prioridad de los elementos. Cada nodo representa un nivel de prioridad, de modo que mantiene una cola con los elementos de dicho nivel de prioridad en el orden en el que van llegando.

Preguntas teóricas de la sección 2.3

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

1. Describe el funcionamiento de las operaciones `getFirst`, `enqueue` y `dequeue` con esta

estructura de datos.
2. Describe el funcionamiento del iterador de la cola con prioridad en esta implementación.

3. Diseño de la práctica

A continuación, vamos a ver el diseño de clases de la práctica necesario para su implementación. Para aquellas clases ya programadas, incluiremos una descripción de su funcionamiento y para aquellas que deban ser completadas por los estudiantes indicaremos cuál será su comportamiento esperado.

3.1. PriorityQueueIF.java

Se trata de la interfaz presentada en el apartado 2.1. que especifica las operaciones de las colas con prioridad y **no debe modificarse**.

3.2. SamePriorityQueue.java

Esta clase representa colas que tienen asociado un nivel de prioridad representado como un número entero. El funcionamiento de esta estructura es **exactamente el mismo que el de una cola normal**, de modo que implementa la interfaz `QueueIF`. Por otro lado, estas estructuras pueden compararse entre sí de acuerdo con su nivel de prioridad, de modo que implementa la interfaz `Comparable`. Esto último implica que esta clase debe implementar un método denominado `compareTo` que compara un objeto `SamePriorityQueue` con otro del mismo tipo. La descripción de las operaciones se encuentra en el esqueleto de esta clase, adjunto a este enunciado.

3.3. BucketQueue.java

Esta clase implementa las colas con prioridad de la manera descrita en el apartado 2.2 basada en una secuencia de elementos `SamePriorityQueue`. Además, incluye la clase privada `PriorityQueueIterator` que implementa un iterador para la cola con prioridad. La descripción de las operaciones se encuentra en el esqueleto de esta clase, adjunto a este enunciado.

3.4. BSTPriorityQueue.java

Esta clase implementa las colas con prioridad de la manera descrita en el apartado 2.3 basada en un árbol binario de búsqueda de elementos `SamePriorityQueue`. Además, incluye la clase privada `PriorityQueueIterator` que implementa un iterador para la cola con prioridad. La descripción de las operaciones se encuentra en el esqueleto de esta clase, adjunto a este enunciado.

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 `BQ` o `BST`, de manera que se indica cuál de las dos implementaciones propuestas de la cola con prioridad se va a emplear.
    b. El segundo es la ruta del fichero de entrada del programa. El formato de un fichero de entrada es el siguiente: cada línea del fichero indica una de las siguientes operaciones de la cola con prioridad: `enqueue`, `dequeue`, `size` e `iterator`. En caso de que la operación sea `enqueue`, a continuación, se indica el elemento y la prioridad, todo ello

separado por espacios en blanco. Por ejemplo “enqueue p54-9 9”.

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 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 `BQ` o `BST` y se crea una cola con prioridad vacía con la implementación correspondiente. En el caso del segundo y tercer argumento se comprueba que las rutas de los ficheros existen mediante la función auxiliar `existsFolder` de la clase `Main`.

3. El programa lee línea a línea el fichero de entrada, de manera que se aplican en orden las operaciones correspondientes sobre la cola con prioridad. En el caso de que la operación sea `size`, el programa escribe en el fichero de salida el número de elementos de la cola con prioridad en su estado actual. En el caso de que la operación sea `iterator`, el programa escribe la secuencia de elementos de la cola con prioridad en su estado actual, ayudándose de la función auxiliar `toString` de la clase `Main`.

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

4. Implementación.

Se deberá realizar un programa en Java llamado **`eped2021.jar`** que contenga todas las clases anteriormente descritas completamente programadas. Todas ellas se implementarán en un único paquete llamado:

`es.uned.lsi.eped.pract2020_2021`

Para la implementación de las estructuras de datos de soporte **se deberá utilizar las interfaces y las implementaciones proporcionadas por el Equipo Docente** de la asignatura.

A través del entorno 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 los estudiantes no tienen que 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 eped2021.jar <implementacion> <entrada> <salida>
```

siendo:

- **`<implementacion>`** BQ o BST 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 que los

estudiantes puedan comprobar el correcto funcionamiento del programa. Si se supera el juego de pruebas privado de los tutores (que será diferente del proporcionado a los estudiantes), la práctica tendrá una calificación mínima de 4 puntos, a falta de la evaluación del estudio empírico del coste y de las preguntas teóricas.

6. Estudio empírico del coste.

Queremos estudiar empíricamente el tiempo de ejecución de cada implementación dependiendo del tamaño del problema (número de operaciones y grado de acumulación de pacientes en espera) y de la cantidad de prioridades existentes.

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. Calcule el coste asintótico temporal en el caso peor de las operaciones `enqueue` y `dequeue` 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?