Práctica 9

Estructura de datos

Grupo 201 GII

María Guzmán Valdezate

Guillermo López de Arechavaleta Zapatero

Contenido

[Introducción 3](#_Toc195955773)

[Descripción y Análisis de Métodos 4](#_Toc195955774)

[Consideraciones adicionales 5](#_Toc195955775)

[Conclusiones 5](#_Toc195955776)

## Introducción

En esta práctica se ha implementado una estructura de datos personalizada denominada ConjuntoLRU, que extiende AbstractSet e implementa la interfaz SortedSet. Su característica principal es aplicar una política de reemplazo LRU (Least Recently Used), eliminando el elemento menos recientemente accedido cuando se alcanza la capacidad máxima.

Para ello, se ha utilizado una estructura auxiliar (cacheLRUEnlazada) que almacena los elementos junto con un contador de accesos. El conjunto es capaz de actualizar internamente este contador cada vez que se accede a un elemento, bien sea por inserción o mediante iteración.

El objetivo del informe es describir el funcionamiento de esta estructura, analizar la eficiencia de sus principales métodos y valorar su comportamiento como conjunto con políticas de uso.

## Descripción y Análisis de Métodos

**add(E e)**

Descripción: Añade un elemento al conjunto. Si ya existe, actualiza su contador de acceso; si no existe y se ha alcanzado la capacidad, elimina el elemento menos recientemente accedido antes de insertarlo.

Complejidad algorítmica: O(n), ya que potencialmente hay que recorrer todos los elementos para identificar el menos usado.

**remove(Object o)**

Descripción: Elimina un elemento del conjunto si está presente.

Complejidad algorítmica: O(1), ya que se basa en una operación de eliminación sobre la estructura cacheLRUEnlazada.

**size()**

Descripción: Devuelve el número de elementos actualmente almacenados en el conjunto.

Complejidad algorítmica: O(1).

**first()**

Descripción: Devuelve el elemento menos recientemente accedido (menor contador).

Complejidad algorítmica: O(n), ya que requiere recorrer todos los elementos para encontrar el mínimo.

**last()**

Descripción: Devuelve el elemento más recientemente accedido (mayor contador).

Complejidad algorítmica: O(n), por el mismo motivo que first.

**headSet(E toElement)**

Descripción: Devuelve un subconjunto de elementos menos accedidos que el indicado.

Complejidad algorítmica: O(n), al requerir recorrer toda la estructura y comparar contadores.

**tailSet(E fromElement)**

Descripción: Devuelve un subconjunto con los elementos más accedidos que el indicado.

Complejidad algorítmica: O(n).

**subSet(E desde, E hasta)**

Descripción: Devuelve un subconjunto con los elementos cuyo número de acceso está entre los dos elementos especificados (inclusive el primero, exclusivo el segundo).

Complejidad algorítmica: O(n), por el recorrido completo y la doble condición de acceso.

**comparator()**

Descripción: Devuelve null ya que no se utiliza ningún comparador externo para el orden.

Complejidad algorítmica: O(1).

**iterator()**

Descripción: Devuelve un iterador que, al recorrer los elementos, actualiza el contador de acceso para reflejar su uso reciente.

Complejidad algorítmica: O(n), por la creación de la copia de las claves y la actualización durante la iteración.

## Conclusiones

La implementación de ConjuntoLRU demuestra cómo extender estructuras estándar en Java puede dotar de nuevas funcionalidades personalizadas a los conjuntos, como en este caso una política de acceso basada en LRU.

A pesar de que muchas operaciones presentan una complejidad lineal debido a la necesidad de buscar el elemento menos o más usado, se consigue un comportamiento eficiente en la mayoría de los casos prácticos gracias a la estructura interna.

El diseño ofrece una solución elegante y funcional para escenarios donde se requiere limitar la memoria o eliminar elementos antiguos, lo que puede ser útil, por ejemplo, en caches o gestores de recursos limitados.

Además, al implementar SortedSet, se permite trabajar con subconjuntos basados en el uso, lo que aporta un mayor nivel de control y versatilidad sobre los datos almacenados.