Práctica 8

Estructura de datos

Grupo 201 GII

María Guzmán Valdezate

Guillermo López de Arechavaleta Zapatero

Contenido

[Introducción 3](#_Toc195513030)

[Descripción y Análisis de Métodos 4](#_Toc195513031)

[Conclusiones 5](#_Toc195513032)

## Introducción

a

## Descripción y Análisis de Métodos

**seleccionaPredicado (Collection<E> coleccion, Condicion condicion, E referencia)**

* Descripción: Este método filtra los elementos de la colección que cumplen una condición de comparación respecto a un valor de referencia, utilizando la interfaz Comparable<E>. Se itera sobre todos los elementos y se añade al conjunto resultado cada elemento que cumple la condición especificada.
* Complejidad algorítmica: Recorre todos los elementos de la colección y los compara, la operación de compareTo es de O (1), por lo que la complejidad total es de O (n).

**eliminaPredicado(Collection<E> coleccion, Condicion condicion, E referencia)**

* Descripción: Este método devuelve un conjunto con los elementos de la colección original que no cumplen la condición de comparación con respecto a un valor de referencia. Se usa igualmente la interfaz Comparable<E> para las comparaciones.
* Complejidad algorítmica: Recorre todos los elementos de la colección y los compara, la operación de compareTo es de O (1), por lo que la complejidad total es de O (n).

**seleccionaPredicado(Collection<E> coleccion, Condicion condicion, E referencia, Comparator<E> comparador)**

* Descripción:
* Complejidad algorítmica:

**eliminaPredicado(Collection<E> coleccion, Condicion condicion, E referencia, Comparator<E> comparador)**

* Descripción:
* Complejidad algorítmica:

**crearSetMismoTipo(Collection<E> coleccion)**

* Descripción:
* Complejidad algorítmica:

## Conclusiones

El avance en el desarrollo del MultiMapa ha facilitado un uso más profundo de mapas y listas en Java. Al permitir que haya varios valores por clave, esta estructura se ajusta a situaciones donde se necesita una relación uno-a-muchos, como en agrupaciones, vínculos o registros.

Se ha comprobado que, en líneas generales, los métodos presentan una eficiencia aceptable, siendo la mayoría O(1) o O(n) según el tamaño de las listas. Sin embargo, se ha encontrado un caso particular en la eliminación FIFO (remove(Object key)), que puede llegar a O(n) debido a la estructura de ArrayList. Este problema se podría solucionar utilizando una estructura diferente como LinkedList.

La implementación de un índice de acceso cíclico ofrece una utilidad interesante, ideal para sistemas que necesitan rotar tareas o acceder a datos de manera secuencial y repetida.

En resumen, esta experiencia ha mostrado cómo elegir bien las estructuras y hacer un análisis detallado de su comportamiento algorítmico puede llevar a soluciones sólidas, adaptativas y eficaces para una variedad de problemas.