2020

UNIVERSIDAD DE BURGOS

ESTRUCTURA DE DATOS

ALEJANDRO ORTEGA; JORGE RUIZ

Contenido

Introducción de la practica

La tercera práctica entregable consiste en la codificación de una nueva estructura de datos que extienda el Java Collection Framework.

Esta estructura se trata de un mapa bidimensional / multimapa /Tabla, en la que los contenidos estén indexados por dos parámetros genéricos, correspondientes a Fila y Columna.

El valor que guardará la tabla será un objeto Celda, que nos permitirá no sólo almacenar el valor, sino otros datos relacionados con la tabla. Además, como el valor se almacena dentro de este objeto, podemos manipular directamente la celda, sin necesidad de acceder continuamente a la tabla para realizar cambios sobre una celda concreta.

Complejidad Algorítmica

En este apartado veremos la complejidad algorítmica de cada uno de los métodos.

Métodos de la clase HashMapCell

HashMapCell ()

El constructor tiene una complejidad algorítmica de **O(1)**, ya que únicamente realiza operaciones básicas de acceso a memoria.

getRowKey ()

El método tiene una complejidad de **O(1)** ya que sólo realiza un acceso a memoria independientemente de la entrada.

getColumnKey()

El método tiene una complejidad de **O(1)** ya que únicamente realiza un acceso a memoria independientemente de la entrada.

getValue()

El método tiene una complejidad de **O(1)** ya que únicamente realiza un acceso a memoria independientemente de la entrada.

setValue()

El método tiene una complejidad de **O(1)** ya que únicamente realiza un acceso a memoria y operaciones algebraicas básicas.

equals()

Todos los métodos utilizados en este tienen una complejidad de O(1), por lo que su complejidad algorítmica es de **O(1)**.

hashCode()

Solo realiza operaciones aritméticas, por lo que su complejidad es de **O(1)**.

Métodos de la clase HashMapTable

HashMapTable()

El constructor tiene una complejidad de **O(1)**, ya que únicamente instancia un mapa vacío.

put (R, K, V)

El método utiliza .containsKey(), intenta acceder al valor que tiene un clave, y si lo encuentra devuelve true y si no false.

Dicho método tiene una complejidad de O(1).

Siguiendo la regla de la suma. O(1) de cotains() + O(1) de get de la fila + O(1) del get de la columna nos deja una complejidad de **O(1).**

remove (R,K)

El método remove solo realiza 1 acceso a memoria y 1 comparación por cada acceso a la tabla.

Puede realizar 3 .get() con complejidad O(1) seguidos, utilizando la regla de la suma sabemos que el máximo de esos 3 gets es **O(1)**.

get (Object, Object)

Al igual que el método anterior, únicamente se realizan operaciones simples, por lo que su complejidad es **O(1)**.

containsKeys (Object, Object)

Realiza 2 contains consecutives. Cada contains tiene una complejidad de O(1), y por la regla de la suma sabemos que el algoritmo tiene una complejidad de **O(1)**.

containsValue(Object)

Este método recorre por cada fila y por cada columna, todas las celdas de la tabla intentando encontrar el objeto pasado como parámetro.

Recorrer todas las filas tiene una complejidad de O(r), siendo r el número de filas, recorrer todas las columnas tiene una complejidad de O(c), siendo c el número de columnas.

Se recorren todas las filas por cada columna, por lo que aplicando la regla del producto (y suponiendo que la tabla tenga el mismo número de filas que columnas), obtenemos una complejidad de **O(r*c) = O(n^2).**

row(object)

En este método se recorren todas las filas, por no que siendo n el numero de filas, tenemos O(n). El resto de métodos empleados tienen una complejidad de O(1), por lo que este método tiene una complejidad de **O(n)**.

columna (Object)

Este método recorre todas las filas, y una vez en ellas, recorre todas las celdas para ver en que columna se encuentran.

Recorrer todas las filas tiene una complejidad de O(r), siendo r el número de filas.

Recorrer todas las celdas por fila tiene una complejidad de O(c), siendo c el numero de celdas.

El resto de métodos tienen complejidad de O(1), por lo que aplicando la regla del producto citada anteriormente0, y suponiendo que tenemos tantas celdas como columnas, obtenemos una complejidad de **O(r*c) = O(n^2)**.

cellSet()

Al igual que el método contains value, este método anida dos bucles, uno para recorrer filas y otro para recorrer columnas, por lo que tenemos O(n^2). A esto le añadimos el método add de la clase HashSet, con una complejidad de O(1), por lo que resulta en un método con **O(n^2)**.

size()

Recorre todas las filas, por lo que, siendo n el número de filas, tendríamos O(n).

Dentro de este bucle, llamamos al método size de HashMap, que tiene complejidad de O(1).

Así, obtenemos una complejidad de **O(n).**

isEmpty()

Solo realiza una operación booleana y un método con O(n), por lo que su complejidad es de **O(n).**

clear()

Llama a un método con complejidad O(1), por lo que su complejidad es O(1).