Estructura de datos

Alejandro Ortega; Jorge Ruiz

2020

Contenido

[Introducción de la practica 3](#_Toc36315760)

[Complejidad Algorítmica 3](#_Toc36315761)

[Métodos de la clase NodoMixto 3](#_Toc36315762)

[NodoMixto() 3](#_Toc36315763)

[offer (E e) 3](#_Toc36315764)

[poll() 3](#_Toc36315765)

[peek() 3](#_Toc36315766)

[peek (int index) 3](#_Toc36315767)

[size() 3](#_Toc36315768)

[iterator() 3](#_Toc36315769)

[getSiguienteNodo() 3](#_Toc36315770)

[setSiguienteNodo(NodoMixto siguiente) 4](#_Toc36315771)

[setAnteriorNodo(NodoMixto anterior) 4](#_Toc36315772)

[getAnteriorNodo() 4](#_Toc36315773)

[Métodos de la clase IteradorNodo 4](#_Toc36315774)

[IteradorNodo() 4](#_Toc36315775)

[hasNext() 4](#_Toc36315776)

[next() 4](#_Toc36315777)

[Métodos de la clase ColaMixta 4](#_Toc36315778)

[ColaMixta(int tamañoNodo) 4](#_Toc36315779)

[offer(E e) 4](#_Toc36315780)

[poll() 4](#_Toc36315781)

[peek() 5](#_Toc36315782)

[peek(int index) 5](#_Toc36315783)

[Iterator() 5](#_Toc36315784)

[Size() 5](#_Toc36315785)

[isUltimoNodoLleno() 5](#_Toc36315786)

[isPrimerNodoVacío() 5](#_Toc36315787)

[añadirNodoVacioEnLaCola() 5](#_Toc36315788)

[eliminarNodo(NodoMixto nodo) 5](#_Toc36315789)

[Métodos de la clase IteradorMixto 5](#_Toc36315790)

[IteradorMixto() 5](#_Toc36315791)

[hasNext() 5](#_Toc36315792)

[next() 5](#_Toc36315793)

# 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()

### setValue()

### equals()

## 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)

### columna (object)

### cellSet()

### size()

### isEmpty()

### clear()

### size()

Su complejidad es de **O(1)**.

### iterator()

Lo único que hace es crear un nuevo objeto del tipo *IteradorNodo*, por lo que su complejidad es de **O(1)**.

### getSiguienteNodo()

Tiene una complejidad de **O(1)**

### setSiguienteNodo(NodoMixto siguiente)

Tiene una complejidad de **O(1)**

### setAnteriorNodo(NodoMixto anterior)

Tiene una complejidad de **O(1)**

### getAnteriorNodo()

Tiene una complejidad de **O(1)**

## Métodos de la clase IteradorNodo

### IteradorNodo()

El constructor del iterador tiene una complejidad algorítmica de **O(1)**.

### hasNext()

Este método, al solo hacer una comparación, tiene una complejidad de **O(1)**.

### next()

Este método solo usa el método get() de un ArrayList, que tiene complejidad O(1), por lo que este método también tiene complejidad de **O(1)**.

## Métodos de la clase ColaMixta

### ColaMixta(int tamañoNodo)

Lo único que hace este método es una llamada al constructor de la clase NodoMixto, que tiene complejidad de O(1), por lo que este constructor también tiene una complejidad de **O(1)**.

### offer(E e)

Los métodos que utiliza son el método offer(E e) de NodoMixto, con una complejidad de O(1), y los métodos añadirNodoVacioEnLaCola() y isUltimoNodoLleno(), con complejidad también de O(1), por lo que este método tiene una complejidad de **O(1)**.

### poll()

Los métodos que usa son isPrimerNodoVacío() y eliminarNodo(NodoMixto nodo), los dos solo con operaciones triviales, sin embargo, también usa el método poll() de NodoMixto, con complejidad de O(n), por lo que su complejidad algorítmica es **O(n)**.

### peek()

Utiliza el método peek() de NodoMixto, con complejidad de O(1), por lo que la complejidad de este método es también **O(1)**.

### peek(int index)

A parte de operaciones triviales con complejidad O(1), y del método getSiguienteNodo() de NodoMixto, también usa el método peek(int index) de dicha clase, que tiene una complejidad de O(n), por lo que la complejidad de este método es **O(n)**.

### Iterator()

Solo crea un nuevo objeto del tipo IteradorMixto, por lo que su complejidad es de **O(1)**.

### Size()

Tiene un bucle que se repite tantas veces como nodos tenga la cola, por lo que su complejidad es de **O(n)**, con n=nº de nodos.

### isUltimoNodoLleno()

Utiliza solo el método size de la clase NodoMixto, por lo que tiene una complejidad de **O(1)**.

### isPrimerNodoVacío()

Utiliza solo el método size de la clase NodoMixto, por lo que tiene una complejidad de **O(1)**.

### añadirNodoVacioEnLaCola()

utiliza los métodos getSiguienteNodo(), setSiguienteNodo y setAnteriorNodo, con complejidad de O(1), por lo que este método tiene también una complejidad de **O(1)**.

### eliminarNodo(NodoMixto nodo)

Utiliza los métodos getSiguienteNodo(), setSiguienteNodo(NodoMixto), setAnteriorNodo(NodoMixto) y getAnteriorNodo(), con complejidad de O(1), por lo que este método también tiene una complejidad de **O(1)**.

## Métodos de la clase IteradorMixto

### IteradorMixto()

Este constructor tiene una complejidad de **O(1)**.

### hasNext()

Utiliza los métodos getSiguienteNodo() y iterator() de la clase NodoMixto, así como el método hasNext() del IteradorNodo, con complejidad de O(1), por lo que tiene una complejidad de **O(1)**.

### next()

Solo utiliza el método hasNext(), y el método next() del IteradorNodo, por lo que su complejidad es de **O(1)**.