Informe Segunda Practica Entregable

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

Esta segunda practica entregable consiste en la elaboración de una estructura de datos de tipo cola, con la particularidad de que esta cola está dividida en nodos de un tamaño fijo.

Cuando se añaden elementos nuevos a la cola, y no queda sitio en el ultimo nodo de esta, se crea uno nuevo. Asimismo, cuando se van eliminando elementos de la cola, y un nodo (el primero) queda vacío, se elimina este nodo de la cola.

Además, se implementa tanto un iterador, como un método peek (int index), que nos deja ver que elemento se encuentra en una posición de la cola.

# Complejidad Algorítmica

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

## Métodos de la clase NodoMixto

### NodoMixto()

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

### offer (E e)

La complejidad algorítmica de este método es **O(1)**, ya que solo se utiliza el método add(e) de un ArrayList con límite de elementos conocido.

### poll()

Su complejidad algorítmica es de **O(n)**, se usa el método remove de un ArrayList y tenemos que iterar por todo este array para encontrar el elemento que queremos buscar.

### peek()

Su complejidad algorítmica es de **O(1)**, ya que estamos usando el método get() de un ArrayList, que tiene esta complejidad.

### peek (int index)

Al igual que el anterior método, tiene una complejidad de **O(1)**.

### 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)**.