Estructuras de datos

Pilas y colas

Investigación de conceptos

Walter González Zúñiga

Aldo Barrera González

TID42M

Docente Milton Batres

UTCH | Universidad Tecnológica de Chihuahua

Contenido

REPORTE DE PRÁCTICA

Objetivo:

El objetivo de esta tarea es investigar y sintetizar los conceptos fundamentales de las pilas y colas.

Se abarcarán las definiciones y características de las pilas y colas además de las operaciones básicas que estas proveen.

Material:

- Formato de Reporte de Práctica
- Internet
- Computador

Desarrollo:

Introducción

Las estructuras de datos son componentes fundamentales en el desarrollo de software, proporcionando marcos organizados para el almacenamiento y manipulación eficiente de datos. Entre las diversas estructuras existentes, las pilas y colas destacan por su simplicidad conceptual y su amplia aplicación en la resolución de problemas computacionales.

Estas estructuras se distinguen por sus métodos específicos de acceso y manipulación de datos: las pilas siguiendo el principio **LIFO** (Last In, First Out) y las colas el principio **FIFO** (First In, First Out).

La combinación de teoría, ejemplos prácticos y visualizaciones permitirá desarrollar un entendimiento integral de estas estructuras de datos fundamentales.

Para todas las estructuras de este reporte, se estará usando la siguiente clase nodo:

```
public class Nodo<T> {
    T dato;
    Nodo<T> siguiente;
    public Nodo(T dato) {
        this.dato = dato;
        this.siguiente = null;  }

    public T getDato() {
        return dato;}

    public void setDato(T dato) {
        this.dato = dato;  }

    public Nodo<T> getSiguiente() {
        return siguiente;}

    public void setSiguiente(Nodo<T> siguiente) {
        this.siguiente = siguiente;}

    @Override
    public String toString() {
        return "Nodo{dato=" + dato + "}"; }}
```

Stacks (pilas)

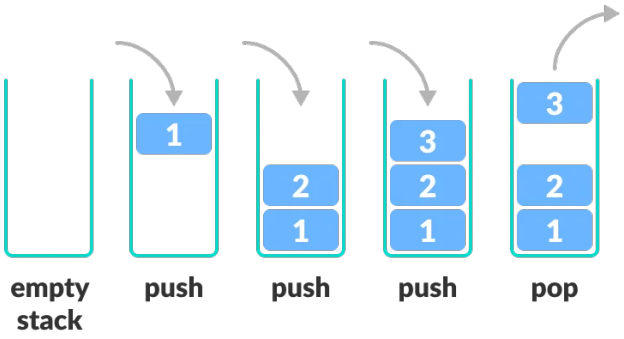
Una pila es una estructura de datos lineal que funciona bajo el principio "último en entrar, primero en salir". El elemento más reciente agregado será el primero en retirarse. Podemos imaginar una pila como elementos apilados uno sobre otro, donde solo tenemos acceso al elemento ubicado en la parte superior.

Características principales

Acceso restringido a través de su axioma para obtener solamente el ultimo elemento añadido.

Operaciones básicas:

- **Push**: Añade un elemento al tope de la pila.
- **Pop**: Elimina y devuelve el elemento del tope de la pila.
- **Peek (o Top)**: Devuelve el elemento en el tope sin eliminarlo.
- **isEmpty**: Verifica si la pila está vacía.



Creación de una pila

La pila se inicializa con un nodo base que podemos consultar directamente

```
public class Pilas<T> {
    private Nodo<T> cima;
    private int tamaño;

    public Pilas() {
        this.cima = null;
        this.tamaño = 0;
    }
```

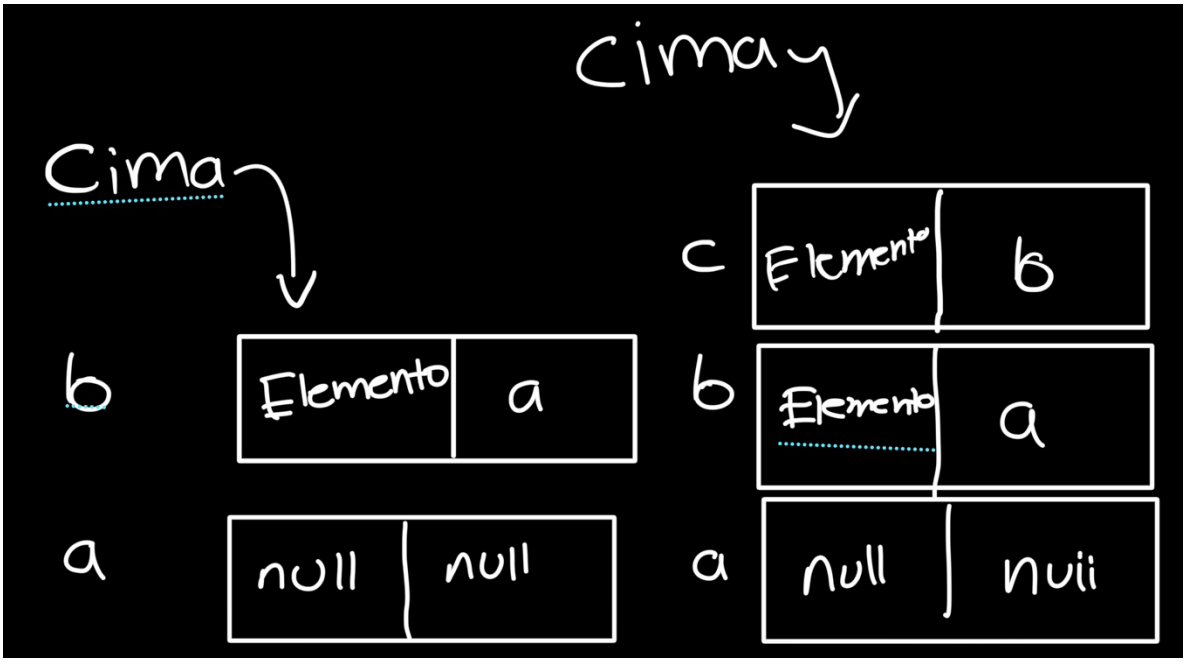
El metodo push se encarga de apilar un nuevo elemento a la pila. Crea un nuevo nodo con la información otorgada y setea a **cima** como su siguiente.

Despues, actualizamos la variable **cima** para que esta almacene al nuevoNodo que.

```
    public void push(T elemento) {
        Nodo<T> nuevoNodo = new Nodo<>(elemento);
        if (!estaVacia()) {
            nuevoNodo.setSiguiente(cima);
        }
        cima = nuevoNodo;
        tamaño++;
    }
```

El metodo pop se encarga de almacenar el dato que guarda el nodo que **cima** apunta, despues, actualizamos **cima** para que guardar el nodo al que apunta **cima**, decrementamos el tamaño de la pila y retornamos el valor almacenado.

```
    public T pop() {
        if (estaVacia()) {
            throw new IllegalStateException("La pila está vacía");
        }
        T elemento = cima.getDato();
        cima = cima.getSiguiente();
        tamaño--;
        return elemento;
    }
```



Para vaciar, hacemos que **cima** no guarde ningun nodo y seteamos el tamaño a 0.

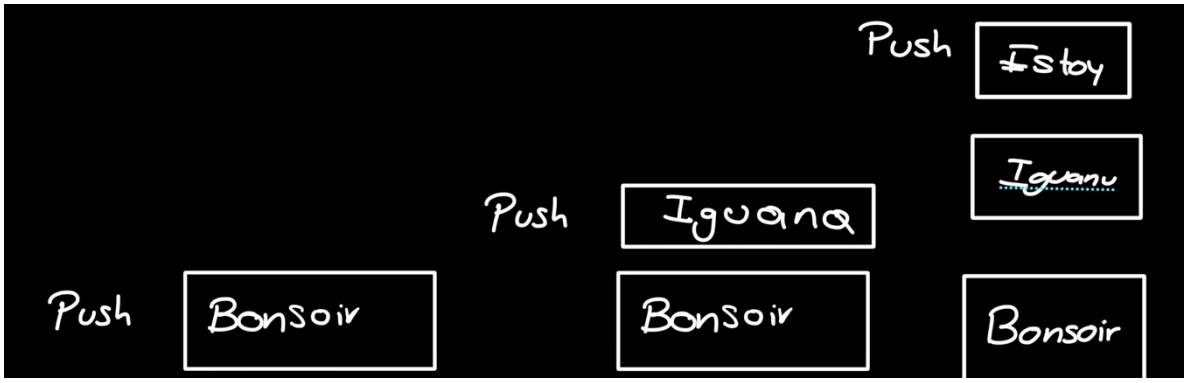
```
public void vaciar() {
    cima = null;
    tamaño = 0;
}
```

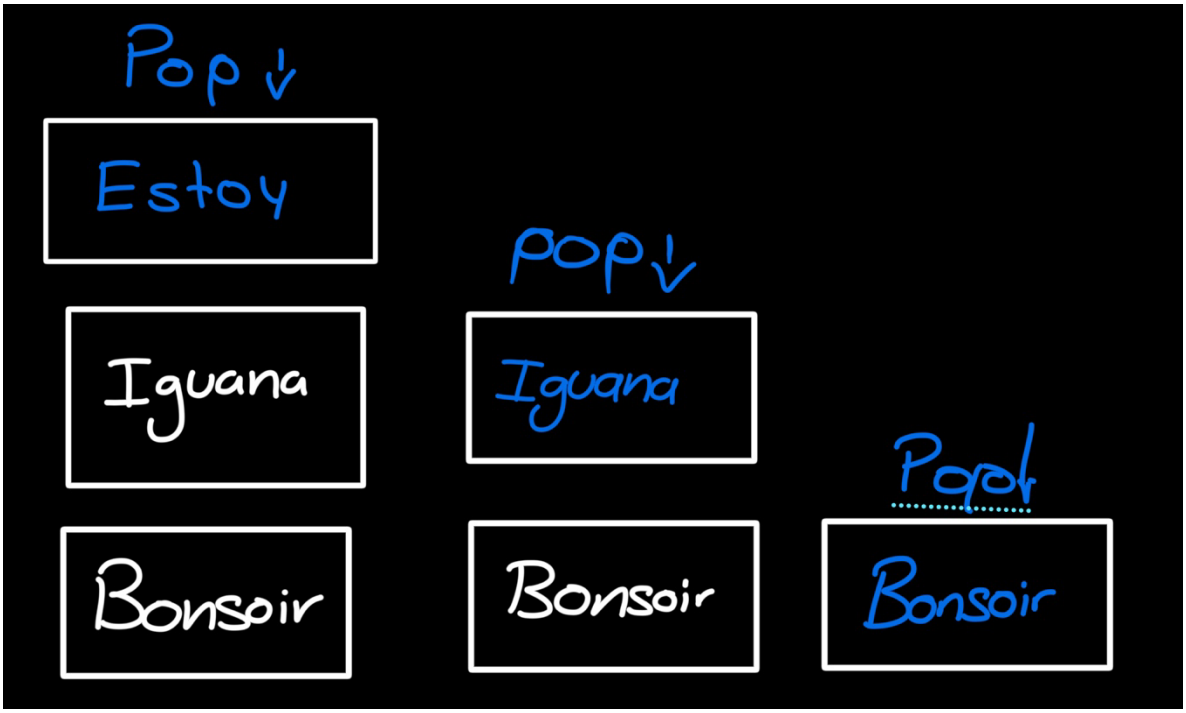
Ejemplo

Clipboard

```
Pilas<String> clipboard = new Pilas<>();
System.out.println("Simulación de Portapapeles");
clipboard.push("Bonsoir profe");
clipboard.push("iguana");
clipboard.push("estoy cansado");
System.out.println("\nPegando elementos (último al
primero):");
    System.out.println("Pegado: " + clipboard.pop());
    System.out.println("\nPegado: " + clipboard.pop());
    System.out.println("\nPegado: " + clipboard.pop());
```

```
Simulación de Portapapeles

Pegando elementos (último al primero):
Pegado: estoy cansado

Pegado: iguana

Pegado: Bonsoir profe
```





Queues

Una cola es una estructura de datos lineal que funciona bajo el principio "primero en entrar, primero en salir" (FIFO). El primer elemento agregado será el primero en retirarse. Podemos imaginar una cola como una fila de personas esperando, donde la primera persona que llegó será la primera en ser atendida.

Características principales

Acceso restringido a través de su axioma para obtener solamente el primer elemento añadido.

Operaciones básicas:

- **Enqueue**: Añade un elemento al final de la cola.
- **Dequeue**: Elimina y devuelve el elemento del frente de la cola
- **Peek (o front)**: Devuelve el elemento de frente sin eliminar
- **isEmpty**: Verifica si la cola está vacía.

Creación de una cola

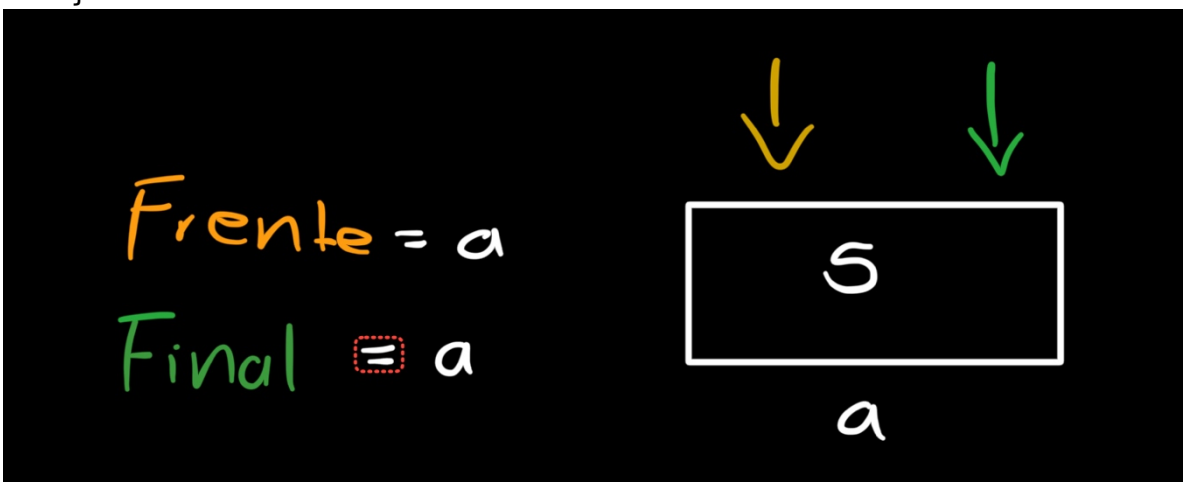
Se crean dos nodos para acceder a los datos: el frente y el final de la cola. El nodo del frente indica el siguiente dato que se extraerá, mientras que el nodo final señala después de qué elemento se realizará una nueva inserción.

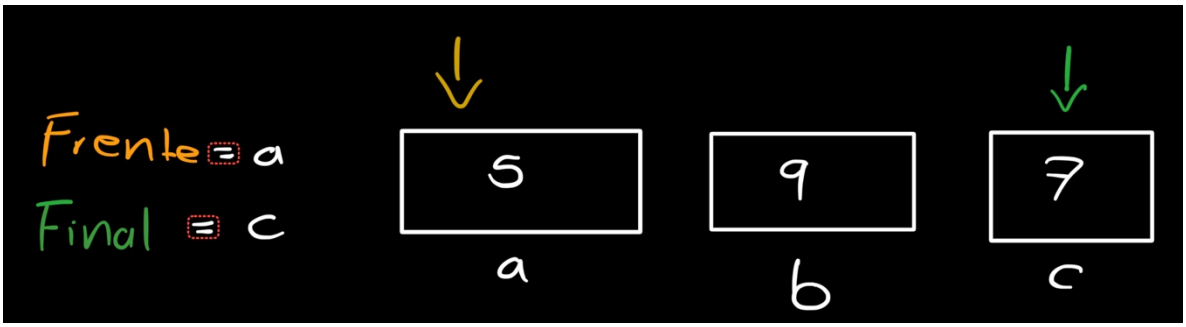
```
public class Cola<T> {
    private Nodo<T> frente;
    private Nodo<T> finall;
    private int tamaño;

    public Cola() {
        frente = null;
        finall = null;
        tamaño = 0;}
```

El metodo **encolar** o **enqueue,** se encarga de recibir el elemento que se insertará y lo almacena en un nuevo nodo.

Si el frente está vacío, guarda este nuevo nodo como frente y como final, si no, utiliza el nodo final para asignarlo despues de final y luego se almacena este nodo como nuevo final.

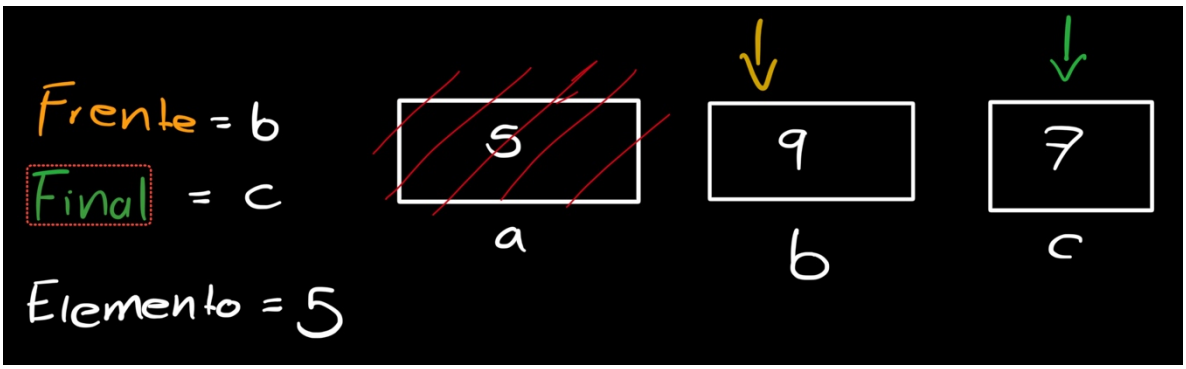
```
public void encolar(T elemento) {
    Nodo<T> nuevoNodo = new Nodo<>(elemento);
    if (estaVacia()) {
        frente = nuevoNodo;
        final = nuevoNodo;

    } else {
        finall.setSiguiente(nuevoNodo);
    }
    finall = nuevoNodo;
    tamaño++;
}
```





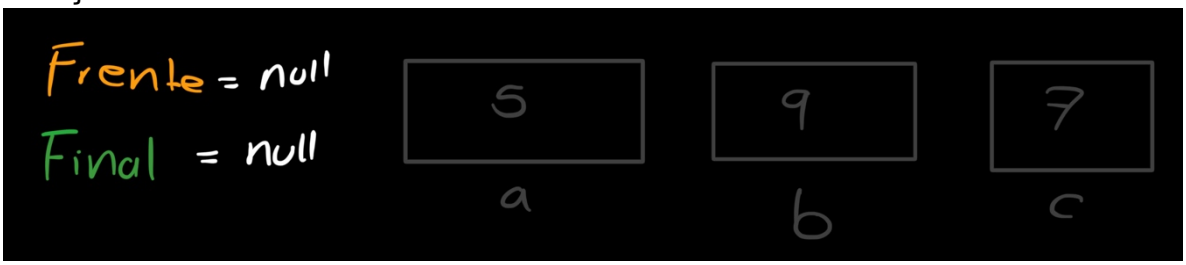
El método **dequeue** o **desencolar** se encarga de eliminar y devolver el elemento del frente de la cola. Almacena el dato que guarda el nodo del frente, actualiza el frente al siguiente nodo, decrementa el tamaño y retorna el valor almacenado.

```
public T desencolar() {
    if (estaVacia()) {
        throw new IllegalStateException("La cola está vacía");
    }
    T elemento = frente.getDato();
    frente = frente.getSiguiente();
    tamaño--;

    if (frente == null) {
        finall = null;
    }
    return elemento;
}
```



Para vaciar la cola, solo actualizamos los punteros de frente y finall con null.

```
public void vaciar() {
    frente = null;
    finall = null;
    tamaño = 0;
}
```



Ejemplo

Tomemos por ejemplo un sistema de ordenes para restaurante, cada orden tiene una mesa asignada, un platillo y un ID, por lo que podemos crear una cola de objetos Orden y así preparar los platillos en el orden que fueron pedidos.

Se crea la cola restaurante

```
Cola<Orden> restaurante = new Cola<>();
```

El metodo agregar orden toma los datos de la orden, crea un objeto Orden y lo agrega a la cola restaurante.

```
      agregarOrden(restaurante, "Mesa Wayne", "Batburger Especial", 1);
        agregarOrden(restaurante, "Mesa Arkham", "Ensalada Venenosa
de Ivy", 2);
        agregarOrden(restaurante, "Mesa Gordon", "Café de la GCPD",
3);
        agregarOrden(restaurante, "Mesa Cobblepot", "Pescado del
Iceberg Lounge", 4);
        agregarOrden(restaurante, "Mesa Nigma", "Acertijo Verde", 5);
        agregarOrden(restaurante, "Mesa Alfred", "Té Inglés de la
Mansión", 6);
```

Comenzamos un tryCatch, ya que si intentamos desencolar cuando la lista está vacía, se lanza una excepción.

Podemos ver la proxima orden al hacer peek para conocer el siguiente valor en la cola.

```
        try {
            System.out.println("Próxima orden a preparar: " +
restaurante.primero());
```

Mientras la cola no esté vacía, vamos a almacenar la orden de enfrente de la cola en ordenActual y por ende, la sacamos de la cola.

```
            while (!restaurante.estaVacia()) {
                Orden ordenActual = restaurante.desencolar();
                System.out.println("\nPreparando orden: " +
ordenActual); }
```

```
        } catch (IllegalStateException e) {
            System.out.println("Error: " + e.getMessage());    }
```

Podemos seguir agregando ordenes al restaurante

```
        agregarOrden(restaurante, "Mesa 5", "Sushi", 5);}
```

```
Recibiendo órdenes:
Nueva orden recibida: Orden #1 - Batburger Especial para Mesa Wayne
Estado actual de la cola:
Cola: Orden #1 - Batburger Especial para Mesa Wayne
Nueva orden recibida: Orden #2 - Ensalada Venenosa de Ivy para Mesa Arkham
Estado actual de la cola:
Cola: Orden #1 - Batburger Especial para Mesa Wayne <- Orden #2 - Ensalada Venenosa de Ivy para Mesa Arkham
Nueva orden recibida: Orden #3 - Té Inglés de la Mansión para Mesa Alfred
Estado actual de la cola:
Cola: Orden #1 - Batburger Especial para Mesa Wayne <- Orden #2 - Ensalada Venenosa de Ivy para Mesa Arkham <- Orden #3
és de la Mansión para Mesa Alfred

Procesando órdenes en la Bat-cocina:
Próxima orden a preparar: Orden #1 - Batburger Especial para Mesa Wayne

Preparando orden: Orden #1 - Batburger Especial para Mesa Wayne
Órdenes pendientes:
Cola: Orden #2 - Ensalada Venenosa de Ivy para Mesa Arkham <- Orden #3 - Té Inglés de la Mansión para Mesa Alfred

Preparando orden: Orden #2 - Ensalada Venenosa de Ivy para Mesa Arkham
Órdenes pendientes:
Cola: Orden #3 - Té Inglés de la Mansión para Mesa Alfred

Preparando orden: Orden #3 - Té Inglés de la Mansión para Mesa Alfred
Órdenes pendientes:
Cola vacía
```

Conclusiones:

Esta investigación me permitió explorar más bibliografía sobre temas previamente tratados en clase. El libro "Grokking Algorithms" sirvió nuevamente como excelente referencia para comprender los temas. Además, realizar dibujos de las estructuras facilitó significativamente su entendimiento.

Despues de haber hecho esta estructura con listas dobles, la implementación de los ejemplos fue sumamente sencilla y cada vez se entiende más por que no se usan arrays.

Los stacks es el tema que nos parece más interesante de los dos y siempre es fascinante el pensar como se le ocurrió esto a alguien.

Arboles va a estar más bueno.

Referencias

Bhargava, A. (2016). Grokking algorithms: An illustrated guide for programmers and other curious people. Manning Publications.

GeeksforGeeks. (2024, November 3). Stack Data Structure. GeeksforGeeks. https://www.geeksforgeeks.org/stack-data-structure/

Getting started with data structures: Nodes cheatsheet | Codecademy. (n.d.). Codecademy. https://www.codecademy.com/learn/getting-started-with-data-structures-java/modules/nodes-java/cheatsheet

GeeksforGeeks. (2024, November 3). Queue data structure. GeeksforGeeks. https://www.geeksforgeeks.org/queue-data-structure/

Queue Data Structure with examples (2024) by LogicMojo. (n.d.). https://logicmojo.com/data-structures-queue