Universidad de Cundinamarca

Estructura de Datos

R3-A4-S14 Push and Pop

Javier Mateo Barrero Vanegas

Jim Alejandro Quiñones Martínez

Luis Ángel Martínez Cuenca

Harick Yesid Villarraga Rincon

Cristian Esteban Ruiz Parra

Docente: Dilia Ines Molina Cubillos

15 noviembre de 2025

Introducción

El presente informe tiene como propósito analizar, diseñar e implementar dos estructuras dinámicas fundamentales en el ámbito de la programación: las estructuras de tipo **LIFO o Pila** y **FIFO o Cola**. Estas estructuras permiten organizar y administrar datos según reglas específicas de entrada y salida, las cuales resultan esenciales para resolver problemas en los que el orden de procesamiento es determinante.

Para cumplir este objetivo, primero realzamos una investigación detallada de estas estructuras y su funcionamiento teórico, describiendo sus características, su forma de operación y los escenarios en los que son utilizadas. Posteriormente, se procederá a su implementación práctica en un lenguaje de programación, empleando nodos enlazados que permiten manejar la memoria de manera dinámica. Esta implementación incluirá operaciones básicas como inserción, eliminación y visualización de elementos, con el fin de demostrar el comportamiento real de cada estructura.

Objetivo General

Implementar y analizar el funcionamiento de las estructuras dinámicas de datos tipo LIFO (Pila) y FIFO (Cola), utilizando listas enlazadas para comprender su comportamiento, sus operaciones principales y su aplicación en la solución de problemas computacionales.

Objetivos Específicos

1. Describir las características, reglas de operación y diferencias fundamentales entre las estructuras LIFO y FIFO.
2. Diseñar e implementar una Pila y una Cola empleando nodos enlazados, incluyendo las operaciones de inserción, eliminación y visualización.
3. Evaluar el funcionamiento práctico de ambas estructuras mediante pruebas que permitan observar cómo gestionan los datos según su orden de entrada y salida.

PILA (STACK) – ESTRUCTURA LIFO

Naturaleza conceptual

Una pila es un ADT (Abstract Data Type) que modela el principio LIFO:

El último elemento insertado es el primero que se debe retirar.

Desde el punto de vista teórico, las pilas representan un tipo de memoria restringida donde solo se puede acceder al elemento más recientemente insertado, similar a una memoria temporal volátil.

Esto garantiza ordenamiento implícito y control secuencial inverso, útil para revertir procesos o deshacer acciones.

2. Modelo computacional y fundamentos teóricos

Las pilas tienen relación directa con:

- Máquinas de Pila (Stack Machines)

Lenguajes como Forth, PostScript y entornos como la JVM interpretan programas basándose en operaciones sobre pilas.

- Gramáticas libres de contexto (CFG)

Los autómatas de pila (PDA) utilizan una pila para reconocer lenguajes como los que requieren balanceo de paréntesis o estructuras anidadas.

Es decir, sin pilas no existiría el análisis sintáctico moderno.

- Recursividad y call stack

Cualquier función recursiva se apoya en una pila de activación para:

número de línea

variables locales

valores de retorno

dirección de retorno

Esto forma la base del funcionamiento de lenguajes como C, C++, Python, Java y más.

3. Implementación dinámica

Las pilas dinámicas se implementan típicamente con:

- Listas enlazadas

Cada nodo contiene:

*dato

*puntero al siguiente nodo

+El puntero top apunta al nodo superior.

Ventajas:

*Push y pop en tiempo constante O(1).

Limitaciones:

*Overhead por punteros.

*Fragmentación de memoria posible.

-Regiones de memoria heap

La pila puede crecer asignando nodos con malloc, new, etc.

4.análisis de complejidad

Operación Complejidad

push() O(1)

Operación	Complejidad
pop()	O(1)
top()	O(1)
search()	O(n)

La eficiencia O(1) en operaciones principales la convierte en una estructura extremadamente rápida.

5. Variantes de pilas

Pila con capacidad dinámica

Pila doble (two-way stack)

Pila compartida en un arreglo

Pila de mínima (min-stack)
Guarda siempre el mínimo de todos los elementos.

6. Aplicaciones avanzadas

Evaluación de expresiones postfix (polish inversa).

Backtracking en IA (laberintos, sudoku).

Deshacer/rehacer en editores de texto e IDEs.

Algoritmos DFS en grafos y árboles.

Transformación de expresiones infix → postfix.

Control de instrucciones en arquitecturas RISC.

COLA (QUEUE) – ESTRUCTURA FIFO

Naturaleza conceptual

Las colas implementan el principio FIFO:

El primer elemento que entra es el primero que debe salir.

Desde la teoría de sistemas, representan un mecanismo de flujo, ordenamiento temporal y procesamiento en cadena, útil cuando los eventos deben atenderse cronológicamente.

Modelo computacional y fundamentos matemáticos

Teoría de colas (Queueing Theory)

Las colas son modelo base en:

simulación de tráfico

telecomunicaciones

teoría de probabilidad (procesos Poisson)

optimización de rendimiento en sistemas operativos

balanceo de carga

Los modelos M/M/1, M/M/c, M/D/1 describen comportamiento de colas reales.

- Sistemas operativos

Las colas son esenciales para:

planificación de procesos (FCFS, Round Robin)

buffers de entrada y salida

scheduling de CPU

colas de prioridad para hilos

- Computación distribuida

Las colas de mensajes (RabbitMQ, Kafka, SQS) se basan en el modelo FIFO.

Implementación dinámica

Las colas dinámicas se implementan con:

- Lista enlazada con nodos

Necesita dos punteros:

front

rear

Inserciones en el final, eliminaciones al frente: ambas O(1).

- Cola circular dinámica

Cuando se usa un arreglo circular, el rear y front se “envuelven” alrededor del arreglo.

Ventajas:

Eficiente en memoria.

Sin desplazamientos de elementos.

1. **Análisis de complejidad**

Operación	Complejidad
enqueue()	O(1)
dequeue()	O(1)
front()	O(1)
search()	O(n)

Al igual que la pila, las operaciones principales tienen costo constante.

2. **Variantes avanzadas de colas**

Cola doble (deque) - inserción y eliminación por ambos extremos.

Cola de prioridad -elementos con importancia.

Cola circular -reutiliza espacio en arreglos.

Cola bloqueante - usada en concurrencia para hilos.

Message Queues - comunicación entre procesos.

3. Aplicaciones avanzadas

Algoritmo de búsqueda BFS en grafos.

Planificación de CPU en sistemas operativos.

Simulaciones de servicios (bancos, aeropuertos).

Streaming de datos y buffers.

Control de paquetes en redes (routers/switches).

Sistemas de mensajería distribuida.

CODIGO

```
// funcion para pedir datos al usuario
import java.util.Scanner;

class Nodo {
    int dato;
    Nodo sig;
    Nodo(int d) { dato = d; }
}

class Bicola {
    private Nodo izq, der;

    // Insertar por la izquierda
    void insertarIzq(int x) {
        Nodo n = new Nodo(x);
```

```
    if (izq == null) izq = der = n;
    else { n.sig = izq; izq = n; }
}

// Insertar por la derecha
void insertarDer(int x) {
    Nodo n = new Nodo(x);
    if (der == null) izq = der = n;
    else { der.sig = n; der = n; }
}

// Atender por la izquierda
void atenderIzq() {
    if (izq == null) { System.out.println("Bicola vacía"); return; }
    System.out.println("Sale (izq): " + izq.dato);
    izq = izq.sig;
    if (izq == null) der = null;
}

// Atender por la derecha
void atenderDer() {
    if (der == null) { System.out.println("Bicola vacía"); return; }
    if (izq == der) { // Un solo nodo
        System.out.println("Sale (der): " + der.dato);
        izq = der = null;
        return;
```

```
        }
        Nodo aux = izq;
        while (aux.sig != der) aux = aux.sig;
        System.out.println("Sale (der): " + der.dato);
        der = aux;
        der.sig = null;
    }

    // Listar
    void listar() {
        if (izq == null) { System.out.println("Bicola vacía"); return; }
        Nodo a = izq;
        while (a != null) { System.out.print(a.dato + " "); a = a.sig; }
        System.out.println();
    }
}

public class practica {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Bicola b = new Bicola();
        int op;

        do {
            System.out.println("\n1. Insertar derecha");
            System.out.println("2. Insertar izquierda");
```

```
            System.out.println("3. Atender derecha");
            System.out.println("4. Atender izquierda");
            System.out.println("5. Listar");
            System.out.println("6. Salir");
            System.out.print("Opción: ");
            op = sc.nextInt();

            switch (op) {
                case 1: System.out.print("Valor: "); b.insertarDer(sc.nextInt()); break;
                case 2: System.out.print("Valor: "); b.insertarIzq(sc.nextInt()); break;
                case 3: b.atenderDer(); break;
                case 4: b.atenderIzq(); break;
                case 5: b.listar(); break;
                case 6: System.out.println("Adiós"); break;
                default: System.out.println("Opción inválida");
            }
        } while (op != 6);

        sc.close();
    }
}
```

Conclusión

El desarrollo de este trabajo permitió comprender de manera teórica y práctica el funcionamiento de las estructuras dinámicas de tipo LIFO (Pila) y FIFO (Cola), las cuales representan mecanismos fundamentales para la organización y administración de datos en diversos entornos de programación. Asimismo, la experimentación con estas estructuras permitió reconocer su importancia en la solución de problemas reales, como la gestión de procesos, el almacenamiento temporal de datos, la navegación entre estados y el procesamiento secuencial de tareas. El análisis realizado confirma que tanto las Pilas como las Colas son herramientas esenciales en el diseño de algoritmos eficientes y en la construcción de sistemas que requieren un manejo preciso del orden de ejecución. Por último, el estudio desarrollado no solo fortaleció el entendimiento conceptual de estas estructuras de datos, sino que también facilitó la adquisición de habilidades prácticas para su implementación, evidenciando su relevancia en la programación y su impacto en la optimización del rendimiento de las aplicaciones.