6. Pilas (Stacks) y Colas (Queues)

Las **Pilas (Stacks)** y las **Colas (Queues)** son dos de las estructuras de datos lineales más fundamentales y utilizadas en ciencias de la computación. Aunque ambas almacenan colecciones de elementos, difieren en el orden en que los elementos se añaden y se eliminan.

—

0.1. Pilas (Stacks)

Una **Pila** es una estructura de datos abstracta que sigue el principio **LIFO (Last-In, First-Out)**, que significa "último en entrar, primero en salir". Imagina una pila de platos: solo puedes añadir o quitar platos de la parte superior.

Operaciones Básicas de una Pila:

Las operaciones principales de una pila son:

- **Push:** Añade un elemento a la cima de la pila. Complejidad $\mathcal{O}(1)$.
- **Pop:** Elimina y devuelve el elemento de la cima de la pila. Complejidad $\mathcal{O}(1)$.
- **Top (o Peek):** Devuelve el elemento de la cima de la pila sin eliminarlo. Complejidad $\mathcal{O}(1)$.
- **Empty:** Comprueba si la pila está vacía. Complejidad $\mathcal{O}(1)$.
- **Size:** Devuelve el número de elementos en la pila. Complejidad $\mathcal{O}(1)$.

Aplicaciones Comunes de las Pilas:

- Gestión de llamadas a funciones (pila de llamadas).
- Evaluación o comprobación de expresiones (Como con paréntesis).
- Algoritmos de backtracking.
- Reversión de una secuencia de elementos.
- **Recorrido en Profundidad (DFS - Depth-First Search):** Particularmente útil para árboles y grafos, donde la recursión implícitamente utiliza la pila de llamadas. Una implementación iterativa de DFS también hace uso explícito de una pila.

Implementación de Pilas en C++ (usando `std::stack`)

En C++, la Standard Template Library (STL) proporciona la clase adaptadora `std::stack`, que por defecto usa `std::deque` como contenedor subyacente, aunque también se puede usar `std::vector` o `std::list`.

Listing 1: Ejemplo de uso de `std::stack`

```cpp
#include <iostream>
#include <stack> // Necesario para std::stack
#include <string>

using namespace std;

int main() {
    stack<string> libros;

    // Push: A adir elementos a la pila
    libros.push("Cien a os de soledad");
    libros.push("Don Quijote de la Mancha");
    libros.push("1984");

    cout << "La pila tiene " << libros.size() << " libros."
        << endl;

    // Top: Ver el elemento superior
    cout << "Libro en la cima: " << libros.top() << endl;
         // Output: 1984

    // Pop: Eliminar elementos
    libros.pop();
    cout << "Despues de pop, la cima es: " << libros.top()
        << endl; // Output: Don Quijote de la Mancha

    // Recorrer y vaciar la pila (el pop elimina)
    cout << "Vaciando la pila:" << endl;
    while (!libros.empty()) {
        cout << libros.top() << endl;
        libros.pop();
    }

    cout << "La pila esta vacia? " << (libros.empty() ? "Si" : "No") << endl;

    return 0;
}
```

—

0.2. Colas (Queues)

Una **Cola** es una estructura de datos abstracta que sigue el principio **FIFO (First-In, First-Out)**, que significa "primero en entrar, primero en salir". Imagina una fila de personas esperando en una caja: la primera persona en la fila es la primera en ser atendida.

Operaciones Básicas de una Cola:

Las operaciones principales de una cola son:

- **Push (o Enqueue):** Añade un elemento al final de la cola. Complejidad $\mathcal{O}(1)$.
- **Pop (o Dequeue):** Elimina y devuelve el elemento del frente de la cola. Complejidad $\mathcal{O}(1)$.
- **Front (o Peek):** Devuelve el elemento del frente de la cola sin eliminarlo. Complejidad $\mathcal{O}(1)$.
- **Empty:** Comprueba si la cola está vacía. Complejidad $\mathcal{O}(1)$.
- **Size:** Devuelve el número de elementos en la cola. Complejidad $\mathcal{O}(1)$.

Aplicaciones Comunes de las Colas:

- Simulación de eventos (ej., cola de clientes en un supermercado).
- Gestión de tareas en sistemas operativos (cola de procesos).
- Colas de impresión o de descarga.
- Buffer de datos.
- **Recorrido en Anchura (BFS - Breadth-First Search):** Fundamental para explorar **árboles binarios** y grafos nivel por nivel. Las colas almacenan los nodos pendientes de visitar en el orden correcto para este tipo de recorrido.

Implementación de Colas en C++ (usando `std::queue`)

En C++, la STL proporciona la clase adaptadora `std::queue`, que al igual que `std::stack`, por defecto usa `std::deque` como contenedor subyacente.

Listing 2: Ejemplo de uso de `std::queue`

```cpp
#include <iostream>
#include <queue> // Necesario para std::queue
#include <string>

using namespace std;

```

```cpp
int main() {
    queue<string> clientes;

    // Push: Anadir elementos al final de la cola
    clientes.push("Ana");
    clientes.push("Juan");
    clientes.push("Maria");

    cout << "Hay " << clientes.size() << " clientes en la cola." << endl;

    // Front: Ver el elemento al frente
    cout << "Cliente al frente: " << clientes.front() << endl; // Output: Ana

    // Pop: Eliminar el elemento del frente
    clientes.pop();
    cout << "Despues de pop, el cliente al frente es: " << clientes.front() << endl; // Output: Juan

    // Recorrer y vaciar la cola
    cout << "Atendiendo clientes:" << endl;
    while (!clientes.empty()) {
        cout << clientes.front() << " atendido." << endl;
        clientes.pop();
    }

    cout << "La cola esta vacia? " << (clientes.empty() ? "Si" : "No") << endl;

    return 0;
}
```

—

0.3. Diferencias Clave y Cuándo Usarlas

Tanto las pilas como las colas son herramientas indispensables en el desarrollo de algoritmos, y entender su comportamiento y aplicaciones es crucial para resolver una amplia gama de problemas de manera eficiente. Su rol es especialmente prominente en los recorridos de árboles y grafos, donde la elección entre una pila o una cola define si la exploración se realiza en profundidad o en anchura.