ESTRUCTURA DE DATOS FUNDAMENTALES Y ALGORITMOS
GUÍA DE LABORATORIO 10-11
LISTAS ENLAZADAS DOBLES Y PILAS

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

1. Objetivos:

- Comprender la estructura de un nodo en una lista enlazada doble.
- Implementar operaciones básicas como insertar, eliminar y buscar elementos.
- Escribir algoritmos y codificarlos haciendo uso de listas enlazadas dobles.

2. Equipos, herramientas o materiales

- Computador, Software: Python, Algoritmos, IDE.

3. Fundamento teórico

3.1. Conceptos Clave

3.2. Listas Enlazadas Dobles

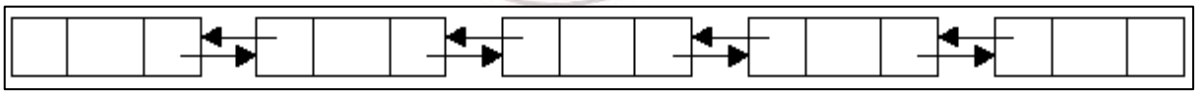
Una lista enlazada doble es una estructura de datos que almacena datos de forma secuencial, pero a diferencia de un array, los elementos no se encuentran contiguos en memoria. En cambio, cada elemento (nodo) se une al siguiente mediante dos punteros: uno que apunta al siguiente nodo (siguiente) y otro que apunta al nodo anterior (anterior).

- **Estructura de un Nodo:**

Un nodo en una lista enlazada doble tiene tres campos:

- **Dato:** Almacena el valor del elemento.
- **Siguiente:** Puntero que apunta al siguiente nodo en la lista.
- **Anterior:** Puntero que apunta al nodo anterior en la lista.

- **Representación Gráfica:**



- **Ventajas:**
  - **Inserción y eliminación eficientes**: Agregar o eliminar elementos en cualquier posición de la lista se realiza en tiempo constante O(1).
  - **Acceso flexible**: Se puede acceder a cualquier elemento de la lista en tiempo lineal O(n), recorriendo la lista desde el principio o desde el final.

- **Desventajas:**
  - **Mayor uso de memoria**: Cada nodo requiere espacio adicional para almacenar dos punteros.
  - **Acceso aleatorio lento**: Acceder a un elemento por su índice no es eficiente, ya que se requiere recorrer la lista desde el principio.

Implementación Nodo en Python – Lista Enlazada Doble:

```python
class Nodo:
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None
        self.anterior = None
```

Definir la clase ListaEnlazadaDoble:

```python
class ListaEnlazadaDoble:
    def __init__(self):
        self.inicio = None
        self.fin = None

```

Operaciones básicas en listas enlazadas dobles:

- **Insertar al principio de la lista:**

```python
class Nodo:
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None
        self.anterior = None

class ListaEnlazadaDoble:
    def __init__(self):
        self.inicio = None
        self.fin = None

    def esta_vacia(self):
        return self.inicio is None

    def agregar_al_principio(self, dato):
        nuevo_nodo = Nodo(dato)
        if self.esta_vacia():
            self.inicio = self.fin = nuevo_nodo
        else:
            nuevo_nodo.siguiente = self.inicio
            self.inicio.anterior = nuevo_nodo
            self.inicio = nuevo_nodo

# Ejemplo de uso:
lista_doble = ListaEnlazadaDoble()
lista_doble.agregar_al_principio(3)
lista_doble.agregar_al_principio(2)

actual = lista_doble.inicio
while actual is not None:
    print(actual.dato, end=" -> ")
    actual = actual.siguiente
print()
```

- **Insertar al final de la lista:**

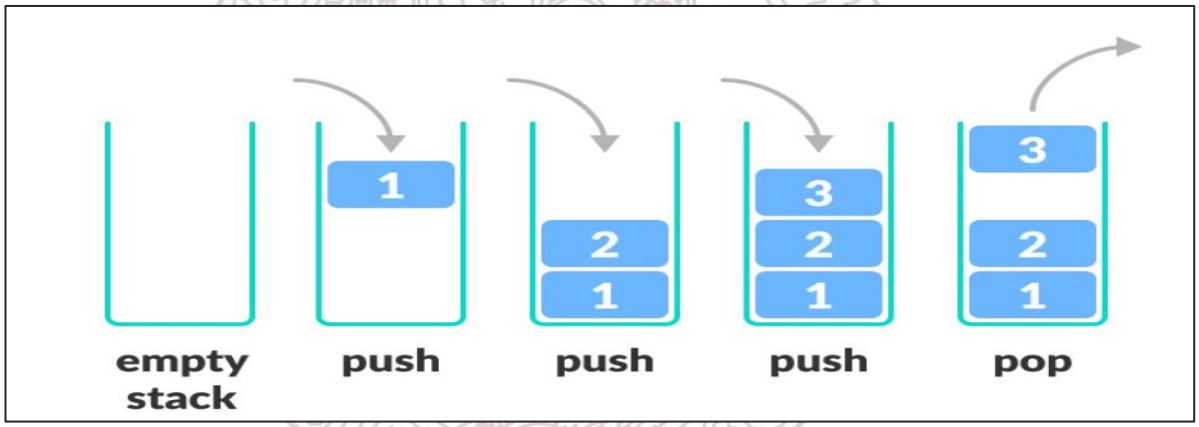
```python
def agregar_al_final(self, dato):
        nuevo_nodo = Nodo(dato)
        if self.esta_vacia():
            self.inicio = self.fin = nuevo_nodo
        else:
            nuevo_nodo.anterior = self.fin
            self.fin.siguiente = nuevo_nodo
            self.fin = nuevo_nodo
```

- **Eliminar:**

```python
def eliminar(self, dato):
        actual = self.inicio
        while actual is not None:
            if actual.dato == dato:
                if actual.anterior is not None:
                    actual.anterior.siguiente = actual.siguiente
                else:
                    self.inicio = actual.siguiente

                if actual.siguiente is not None:
                    actual.siguiente.anterior = actual.anterior
                else:
                    self.fin = actual.anterior

                return  # Se encontró y eliminó el nodo, salir del método

            actual = actual.siguiente
```

3.3. Pilas

Las pilas son un tipo de estructura de datos que se comportan como un "primero en entrar, primero en salir" o "LIFO". Esto significa que el último elemento agregado a la pila es el primer elemento que se puede eliminar. Es como una pila de platos, donde agregas un plato a la parte superior de la pila y el último plato que agregaste es el primer plato que puedes retirar:



Operaciones Básicas:

- **Apilar (push)**: Agregar un nuevo elemento a la pila.
- **Desapilar (pop):** Eliminar el último elemento de la pila y devolverlo.
- **Peek**: Ver el último elemento de la pila sin eliminarlo.
- **Tamaño**: Obtener la cantidad de elementos en la pila.

Aplicaciones:

Las pilas se utilizan en una gran variedad de aplicaciones, incluyendo:

- **Deshacer/Rehacer:** Permite deshacer y rehacer acciones en un programa.
- **Evaluación de expresiones:** Permite evaluar expresiones matemáticas en orden correcto.
- **Navegación web:** Permite mantener un historial de las páginas web visitadas.
- **Algoritmos**: Se utilizan en algoritmos como la búsqueda en profundidad y la búsqueda en anchura.

Implementación de pila en Python:

```python
class Pila:
    def __init__(self):
        self.items = []

    def esta_vacia(self):
        return len(self.items) == 0

    def apilar(self, elemento):
        self.items.append(elemento)

    def desapilar(self):
        if not self.esta_vacia():
            return self.items.pop()
        else:
            print("La pila está vacía. No se puede desapilar.")

    def cima(self):
        if not self.esta_vacia():
            return self.items[-1]
        else:
            print("La pila está vacía. No hay elementos en la cima.")


mi_pila = Pila()

print("¿La pila está vacía?:", mi_pila.esta_vacia())

mi_pila.apilar(5)
mi_pila.apilar(10)
mi_pila.apilar(15)

print("Elemento en la cima de la pila:", mi_pila.cima())

mi_pila.desapilar()
print("Pila después de desapilar un elemento:", mi_pila.items)

mi_pila.desapilar()
print("Pila después de desapilar otro elemento:", mi_pila.items)
```

4. Desarrollo y Actividades

Ejercicio parte 01 – Listas Enlazadas Dobles:

Duplicar Nodos:

1. Crea una lista con al menos 4 nodos, duplica cada nodo de la lista e imprime la lista original y la lista duplicada hacia adelante y hacia atrás.

Contar Nodos Pares e Impares:

2. Crea una lista con al menos 9 nodos, cuenta cuántos nodos tienen un dato par y cuántos tienen un dato impar e imprime la lista hacia adelante y hacia atrás.

Insertar Nodo en Posición Específica:

3. Crea una lista con al menos 5 nodos, inserta un nuevo nodo con el dato 15 en la posición 3 e imprime la lista hacia adelante y hacia atrás.

Eliminar Nodos Duplicados:

4. Crea una lista con nodos que contengan datos duplicados, elimina todos los nodos duplicados, dejando solo una instancia de cada dato e imprime la lista hacia adelante y hacia atrás.

Invertir la Lista:

5. Crea una lista con al menos 6 nodos, invierte el orden de la lista (el último elemento se convierte en el primero y viceversa) e imprime la lista hacia adelante y hacia atrás.

- **Ejercicios parte 02:**

Invertir una cadena:

6. Utilizar una pila para invertir el orden de los caracteres de una cadena.

Convertir número decimal a binario:

7. Implementar un programa que convierta un número decimal a su representación en sistema binario utilizando una pila.

Evaluar expresión posfija:

8. Crear un programa que evalúe una expresión matemática en notación posfija utilizando una pila.

Validar operadores anidados:

9. Verificar si los operadores en una expresión matemática están correctamente anidados utilizando una pila.

Ordenar una pila:

10. Ordenar los elementos de una pila de manera ascendente utilizando estructuras adicionales.

Eliminar duplicados:

11. Eliminar los elementos duplicados de una pila.

Implementar una calculadora sencilla:

12. Crear una calculadora que pueda realizar operaciones básicas (+, -, *, /) utilizando una pila para evaluar expresiones.

Comprobar palíndromos:

13. Utilizar una pila para comprobar si una palabra o frase es un palíndromo.

Simular el proceso de deshacer (undo):

14. Implementar un sistema simple de "deshacer" utilizando dos pilas, una para las acciones y otra para los deshaceres.