ESTRUCTURA DE DATOS FUNDAMENTALES Y ALGORITMOS
GUÍA DE LABORATORIO 12-13
COLAS Y ARBOLES

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:

- Implementar colas utilizando diferentes estructuras de datos.
- Realizar operaciones básicas en colas (encolar, desencolar, peek).
- Implementar un árbol binario, un árbol de búsqueda binaria y un árbol AVL en Python.
- Aplicar árboles a diferentes problemas, como la clasificación de datos y la búsqueda de información

2. Equipos, herramientas o materiales

- Computador, Software: Python, Algoritmos, IDE.

3. Fundamento teórico

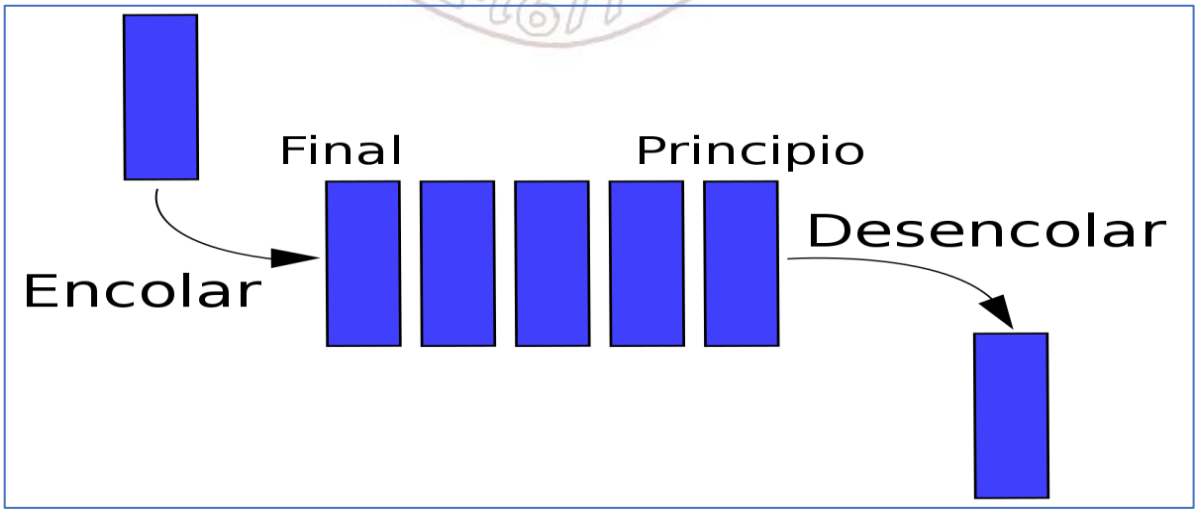
3.1. Conceptos Clave

3.2. COLAS

Una cola es una estructura de datos que sigue el principio **Primero en Entrar, Primero en Salir** (**FIFO**). Los elementos se añaden a la cola por un extremo (el final) y se eliminan por el otro extremo (el frente).

- **Tipos de colas:**
  - **Cola simple:** Implementa el comportamiento FIFO básico**.**
  - **Cola con prioridad:** Los elementos se ordenan según una prioridad y se eliminan en orden de mayor a menor prioridad

- **Representación Gráfica:**



- **Ventajas:**

  Las colas se pueden implementar en Python utilizando diferentes estructuras de datos:

  - **Listas:** La forma más simple de implementar una cola es usar una lista. Los elementos se añaden al final de la lista y se eliminan del principio**.**
  - **Módulo collections.deque:** Python proporciona el módulo **collections** que incluye la clase **deque**, la cual está diseñada específicamente para implementar colas**.**

- **Operaciones básicas en colas:**
  - **Encolar (enqueue):** Añadir un elemento al final de la cola.
  - **Desencolar (dequeue):** Eliminar un elemento del principio de la cola.
  - **Peek:** Consultar el elemento que está al frente de la cola sin eliminarlo.

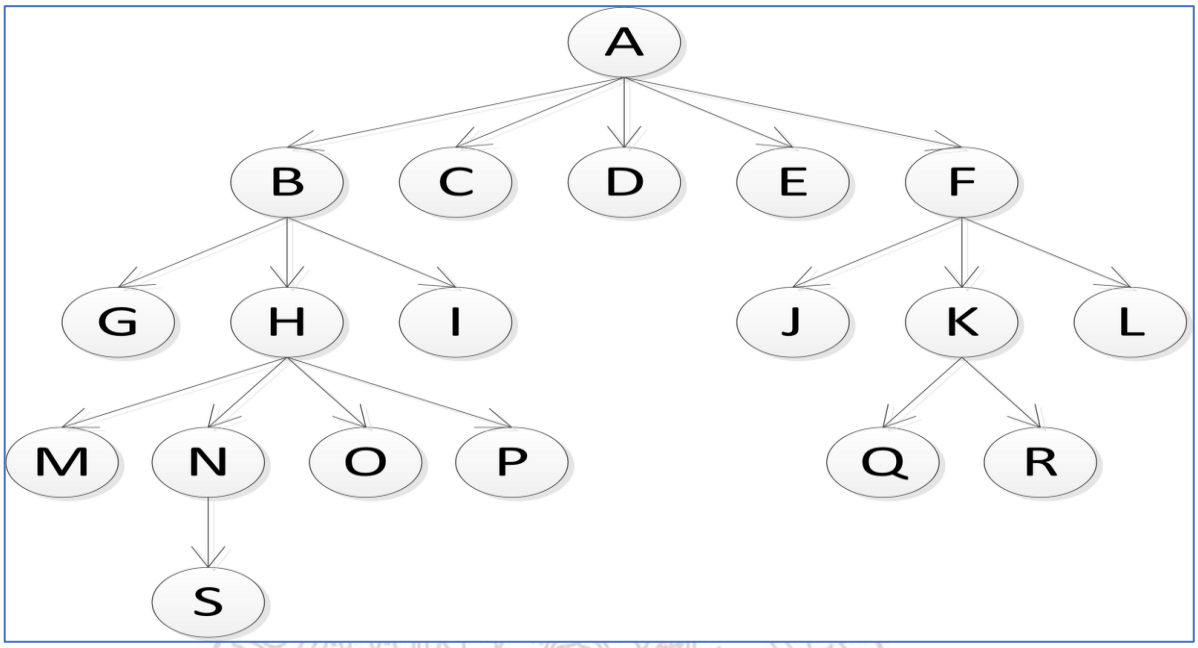
- **Aplicaciones de colas:**
  - Simulación de colas de espera.
  - Almacenamiento de datos para su procesamiento posterior.
  - Implementación de algoritmos de búsqueda y ordenación.

- **Operaciones básicas en colas:**
  - **Agregar y quitar elementos de la cola:**

```python
from collections import deque

class Cola:
    def __init__(self):
        self.items = deque()

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

    def encolar(self, item):
        self.items.append(item)

    def desencolar(self):
        if not self.esta_vacia():
            return self.items.popleft()
        else:
            return None

    def tamano(self):
        return len(self.items)

# Crear una instancia de la clase Cola llamada mi_cola
mi_cola = Cola()

print("¿La cola está vacía?", mi_cola.esta_vacia())

mi_cola.encolar(1)
mi_cola.encolar(2)
mi_cola.encolar(3)

print("Tamaño de la cola:", mi_cola.tamano())

elemento_desencolado = mi_cola.desencolar()
print("Elemento desencolado:", elemento_desencolado)
```

3.3. ARBOLES

Un árbol es una estructura de datos no lineal que consiste en nodos y aristas. Un nodo es un elemento de datos que puede tener hijos (otros nodos). Las aristas conectan a los nodos entre sí:



- **Tipos de Arboles:**
  - **Árboles binarios:** máximo de dos hijos por nodo
  - **Árboles N-arios:** número arbitrario de hijos por nodo
  - **Árboles AVL:** balanceados para optimizar operaciones

- **Aplicaciones:**
  - **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.

- **Implementación de Árbol Binario:**
  - **Clase Nodo:**

```python
class Nodo:
    def __init__(self, dato):
        # "dato" puede ser de cualquier tipo, incluso un objeto si se sobrescriben los operadores de comparación
        self.dato = dato
        self.izquierda = None
        self.derecha = None
```

  - **Insertar Dato:**

```python
    def __agregar_recursivo(self, nodo, dato):
        if dato < nodo.dato:
            if nodo.izquierda is None:
                nodo.izquierda = Nodo(dato)
            else:
                self.__agregar_recursivo(nodo.izquierda, dato)
        else:
            if nodo.derecha is None:
                nodo.derecha = Nodo(dato)
            else:
                self.__agregar_recursivo(nodo.derecha, dato)
```

- **Recorrido:**

```python
    def __inorden_recursivo(self, nodo):
        if nodo is not None:
            self.__inorden_recursivo(nodo.izquierda)
            print(nodo.dato, end=", ")
            self.__inorden_recursivo(nodo.derecha)

    def __preorden_recursivo(self, nodo):
        if nodo is not None:
            print(nodo.dato, end=", ")
            self.__preorden_recursivo(nodo.izquierda)
            self.__preorden_recursivo(nodo.derecha)

    def __postorden_recursivo(self, nodo):
        if nodo is not None:
            self.__postorden_recursivo(nodo.izquierda)
            self.__postorden_recursivo(nodo.derecha)
            print(nodo.dato, end=", ")
```

- **Búsqueda**

```python
    def __buscar(self, nodo, busqueda):
        if nodo is None:
            return None
        if nodo.dato == busqueda:
            return nodo
        if busqueda < nodo.dato:
            return self.__buscar(nodo.izquierda, busqueda)
        else:
            return self.__buscar(nodo.derecha, busqueda)
```

- **Modo de empleo de árbol**

```python
from arbol import Arbol

arbol = Arbol("Luis")
arbol.agregar("María José")
arbol.agregar("Maggie")
arbol.agregar("Leon")
arbol.agregar("Cuphead")
arbol.agregar("Aloy")
arbol.agregar("Jack")
nombre = input("Ingresa algo para agregar al árbol: ")
arbol.agregar(nombre)
arbol.preorden()
arbol.inorden()
arbol.postorden()
# Búsqueda
busqueda = input("Busca algo en el árbol: ")
nodo = arbol.buscar(busqueda)
if nodo is None:
    print(f"{busqueda} no existe")
else:
    print(f"{busqueda} sí existe")
```

4. Desarrollo y Actividades

Ejercicio parte 01 – Colas:

Verificar si una palabra es palíndroma

1. Implementa una función que determine si una palabra es palíndroma o no. Utiliza una cola para comparar los caracteres de la palabra en orden original y reverso.

Diseño de un sistema de gestión de pedidos

2. Crea un sistema de gestión de pedidos que utilice una cola para procesar los pedidos en el orden en que fueron recibidos. Implementa funciones para agregar pedidos, procesar pedidos y mostrar el estado actual de la cola.

Búsqueda de rutas en un laberinto

3. Desarrolla un programa que encuentre el camino más corto a través de un laberinto. Utiliza una cola para realizar un recorrido en anchura (BFS) desde el punto de inicio hasta el punto de destino.

Diseño de un sistema de gestión de tareas (Avanzado)

4. Implementa un sistema de gestión de tareas que permita agregar tareas, marcar tareas como completadas y mostrar la próxima tarea pendiente.

Ejercicios parte 02 - Arboles:

Contar nodos:

5. Implementar una función que cuente la cantidad de nodos en el árbol.
6. Implementar una función que cuente la cantidad de nodos hoja (que no tienen hijos).
7. Implementar una función que cuente la cantidad de nodos internos (que tienen al menos un hijo).

Calcular altura y profundidad:

8. Implementar una función que calcule la altura del árbol (la longitud del camino más largo desde la raíz hasta una hoja).
9. Implementar una función que calcule la profundidad de un nodo (la longitud del camino desde la raíz hasta el nodo).

Buscar el mínimo y el máximo:

10. Implementar una función que encuentre el nodo con el valor mínimo en el árbol.
11. Implementar una función que encuentre el nodo con el valor máximo en el árbol.