Recorrido en árboles binarios

Braian Ramírez - 506222015 Brenda Vargas - 614222713 Viviana García - 506222713

I. Introducción

Los recorridos en árboles binarios son esenciales para entender la organización de sus nodos y son fundamentales en algoritmia y estructuras de datos. Presentaremos dos ejemplos de árboles binarios y analizaremos sus recorridos, proporcionando una comprensión clara de su importancia en programación y algoritmia.MEste informe se enfoca en analizar los recorridos en un árbol binario: preorden, inorden y postorden.

A continuación, se desarrolla lo mencionado anteriormente

II. Origen de los Ejercicios

Se establecen dos ejercicios de manera puntual, en ellos se tiene que encontrar los recorridos inorder, postorder y preorder. Se proporciona un seudocódigo el cual puede ser desarrollado en cualquier lenguaje de programación. En este caso se seleccionó el lenguaje Python para desarrollar dichos ejercicios.

El primer ejercicio propone los tres tipos de recorrido: inorder, postorder y preorder de un árbol. Donde se tiene que comprobar que el recorrido propuesto en el enunciado es correcto.

Figura 1. ejercicio 1

El segundo ejercicio es en un principio un árbol binario del cual se requiere hallar sus tres formas de recorrerlo, empleando el algoritmo anteriormente usado en el primer ejercico se obtuvieron las tres formas de recorrerlo (inorder, postorder y preorder).

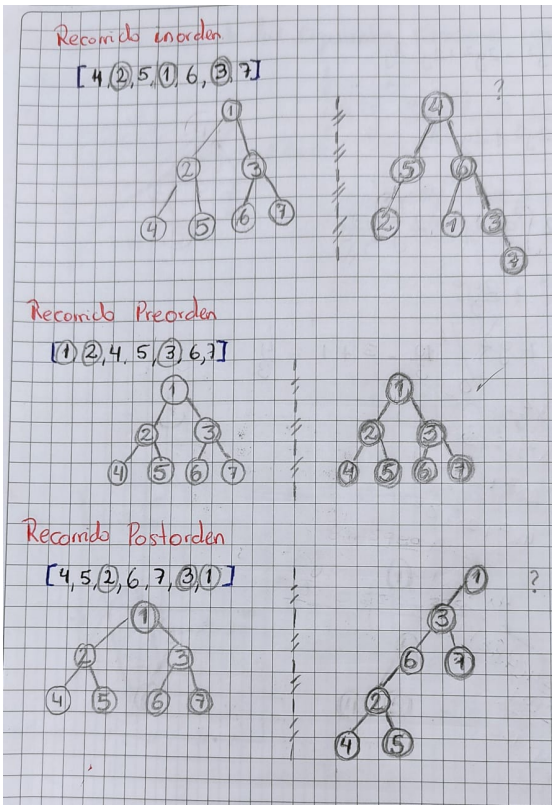

Figura 2. ejercicio 2

III. Análisis ejercicio 1

Se realizó una prueba a mano siguiendo las instrucciones de cada recorrido con respecto al planteado en el enunciado. Se encontró que tanto en el recorrido inorder como en el postorder, sus enunciados eran erróneos. Para corroborar que los recorridos se realizaron de manera correcta, empleamos el código elaborado en Python, obteniendo los mismos resultados que en la prueba desarrollada a mano.

```
El recorrido del segundo árbol en preorden es:
 [1, 2, 4, 5, 3, 6, 7]
El recorrido del segundo árbol en inorden es:
 [4, 2, 5, 1, 6, 3, 7]
El recorrido del segundo árbol en postorden es:
 [4, 5, 2, 6, 7, 3, 1]
```

Figura 3. ejercicio 2

IV. Análisis ejercicio 2

El análisis del árbol binario implicó la aplicación de los tres recorridos fundamentales. En primer lugar, se llevó a cabo el recorrido en preorden, donde la raíz se ubica al principio de la

secuencia de visitas. Luego, se ejecutó el recorrido en orden, que coloca la raíz en la posición intermedia del recorrido. Finalmente, se exploró el recorrido en posorden, en el cual la raíz queda al final de la secuencia.

Los resultados obtenidos una vez programado el código fueron:

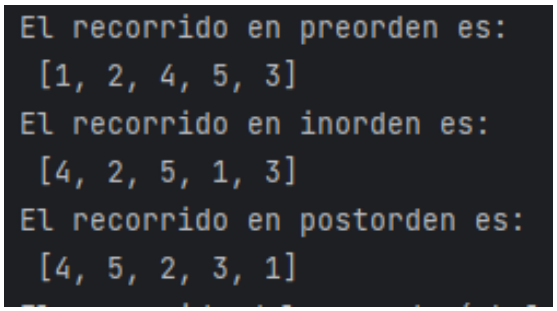

Figura 4. ejercicio 2

V. Código en Python

A continuación, se presenta el código Python utilizado para el análisis del recorrido con su respectivo análisis Big O por sesión de código..

```python
import json

class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.left = None
        self.right = None

    def __repr__(self) -> str:
        return f"<Nodo{self.valor}>"

    # Complejidad de tiempo para preorden: O(n)
    def preorden(self, result_list):
        result_list.append(self.valor)
        if self.left is not None:
            self.left.preorden(result_list)
        if self.right is not None:
            self.right.preorden(result_list)

    # Complejidad de tiempo para inorden: O(n)
    def inorden(self, result_list):
        if self.left is not None:
            self.left.inorden(result_list)
        result_list.append(self.valor)
        if self.right is not None:
            self.right.inorden(result_list)

    # Complejidad de tiempo para postorden: O(n)
    def postorden(self, result_list):
        if self.left is not None:
            self.left.postorden(result_list)
        if self.right is not None:
            self.right.postorden(result_list)
        result_list.append(self.valor)

# Crear el  rbol
arbol = Nodo(1)
arbol.left = Nodo(2)
arbol.right = Nodo(3)
arbol.left.left = Nodo(4)
arbol.left.right = Nodo(5)

#Crear segundo  rbol
arbol2 = Nodo(1)
arbol2.left = Nodo(2)
arbol2.right = Nodo(3)
arbol2.left.left = Nodo(4)
arbol2.left.right = Nodo(5)
arbol2.right.left = Nodo(6)
arbol2.right.right = Nodo(7)

resultado_preorden = []
resultado_inorden = []
resultado_postorden = []

# Realizar recorridos en el  rbol  1
arbol.preorden(resultado_preorden)  # Complejidad: O(n)
arbol.inorden(resultado_inorden)    # Complejidad: O(n)
arbol.postorden(resultado_postorden)  # Complejidad: O(n)

resultado2_preorden = []
resultado2_inorden = []
resultado2_postorden = []

# Realizar recorridos en el  rbol  2
arbol2.preorden(resultado2_preorden)  # Complejidad: O(n)
arbol2.inorden(resultado2_inorden)    # Complejidad: O(n)
arbol2.postorden(resultado2_postorden)  # Complejidad: O(n)

print("El recorrido en preorden es:\n", json.dumps(resultado_preorden))
print("El recorrido en inorden es:\n", json.dumps(resultado_inorden))
print("El recorrido en postorden es:\n", json.dumps(resultado_postorden))

print("El recorrido del segundo  rbol  en preorden es:\n", json.dumps(resultado2_preorden))
print("El recorrido del segundo  rbol  en inorden es:\n", json.dumps(resultado2_inorden))
print("El recorrido del segundo  rbol  en postorden es:\n", json.dumps(resultado2_postorden))
```

VI. Conclusión

Con base a los dos ejercicios presentados, hemos tenido la oportunidad de comprender en profundidad los tres métodos de recorrido en un árbol binario: preorden, inorden y postorden. Cada uno de estos métodos nos brinda una perspectiva única para explorar y analizar la estructura de un árbol, así como para manipular y procesar sus nodos de manera efectiva. Estos conocimientos resultan fundamentales para cualquier programador, ya que los recorridos en árboles binarios son herramientas esenciales en una variedad de aplicaciones, desde la gestión de datos hasta la resolución de problemas complejos en ciencias de la computación.