Trabajo Práctico Integrador

Representación de Fixtures Deportivos como Árboles en Python Usando Listas

Alumnos:
• Luciano José Cartagena – lucianocartagena17@gmail.com
• Santiago Arroquigaray – arroqui192@gmail.com

Materia: Programación I
Profesor: Bruselario, Sebastián
Tutora: Carbonari, Verónica
Fecha de entrega: 09/06/2025

ÍNDICE

1. INTRODUCCIÓN

Las estructuras de datos son fundamentales en la programación, ya que permiten almacenar y manipular información de forma eficiente. Entre ellas, los árboles son estructuras jerárquicas ampliamente utilizadas en algoritmos de búsqueda, bases de datos, inteligencia artificial y más.

Los árboles binarios pueden ser utilizados para representar la estructura de un torneo deportivo de eliminación directa (tipo copa), donde cada partido lleva a otro en la ronda siguiente. Esta aplicación práctica demuestra cómo las estructuras de datos abstractas pueden modelar problemas del mundo real de manera elegante y eficiente.

En este trabajo, implementaremos un fixture deportivo usando listas anidadas en Python, sin clases ni objetos, aprovechando la capacidad de las listas de contener otras listas. Este enfoque permite centrarse en la lógica estructural del árbol sin distracciones de programación orientada a objetos.

2. OBJETIVOS

Objetivo General:

• Implementar y analizar un sistema de fixtures deportivos utilizando árboles binarios representados con listas anidadas en Python.

Objetivos Específicos:

- • Comprender cómo representar un fixture como un árbol binario utilizando únicamente listas.
- • Implementar operaciones básicas sobre árboles: creación, inserción, recorridos y búsqueda.
- • Simular partidos y avanzar ganadores en el árbol de manera dinámica.
- • Visualizar el árbol para mostrar el avance del torneo de forma clara.
- • Evaluar las ventajas y desventajas de este enfoque frente a implementaciones orientadas a objetos.
- • Analizar la complejidad algorítmica de las operaciones implementadas.
- • Promover el uso educativo de estructuras simples como herramienta didáctica.

3. MARCO TEÓRICO

3.1 Fundamentos de Estructuras de Datos

Las estructuras de datos son organizaciones específicas de información que permiten el almacenamiento y manipulación eficiente de datos (Cormen et al., 2009). Según Sedgewick & Wayne (2011), la elección de una estructura de datos apropiada es fundamental para el

diseño de algoritmos eficientes, ya que determina directamente la complejidad temporal y espacial de las operaciones.

3.2 Árboles como Estructuras Jerárquicas

Un árbol es una estructura de datos no lineal que consiste en un conjunto de nodos conectados por aristas, donde existe exactamente un camino entre cualquier par de nodos (Knuth, 1997). Formalmente, un árbol T se define como un conjunto finito de nodos tal que:

- • Existe un nodo especial llamado raíz
- • Los nodos restantes se particionan en subconjuntos disjuntos, cada uno formando un subárbol

Propiedades fundamentales:

- • Altura: Longitud del camino más largo desde la raíz hasta una hoja
- • Profundidad: Distancia desde la raíz hasta un nodo específico
- • Grado: Número máximo de hijos que puede tener un nodo

3.3 Árboles Binarios

Un árbol binario es un caso especial donde cada nodo tiene como máximo dos hijos, denominados hijo izquierdo e hijo derecho (Aho et al., 1983). Esta restricción permite implementaciones más eficientes y algoritmos específicos.

Tipos de árboles binarios:

- • Completo: Todos los niveles están llenos excepto posiblemente el último
- • Perfecto: Todos los nodos internos tienen exactamente dos hijos
- • Balanceado: La diferencia de altura entre subárboles no excede 1

3.4 Complejidad Algorítmica

Según Cormen et al. (2009), las operaciones básicas en árboles binarios tienen las siguientes complejidades:

Operación	Mejor Caso	Caso Promedio	Peor Caso
Búsqueda	O(log n)	O(log n)	O(n)
Inserción	O(log n)	O(log n)	O(n)
Eliminación	O(log n)	O(log n)	O(n)
Recorrido	O(n)	O(n)	O(n)

3.5 Recorridos de Árboles

Los algoritmos de recorrido permiten visitar todos los nodos de manera sistemática (Sedgewick & Wayne, 2011):

- • Preorden: Raíz → Izquierda → Derecha
- • Inorden: Izquierda → Raíz → Derecha
- • Postorden: Izquierda → Derecha → Raíz

3.6 Aplicaciones en Torneos Deportivos

Los árboles binarios son naturalmente adecuados para representar torneos de eliminación directa, donde cada nodo representa un encuentro y sus hijos los encuentros previos necesarios (Graham et al., 1994). Esta aplicación demuestra cómo las estructuras de datos abstractas modelan problemas del mundo real.

4. METODOLOGÍA

4.1 Herramientas y Tecnologías

- • Lenguaje: Python 3.x
- • Paradigma: Programación funcional (sin clases ni objetos)
- • Estructura de datos: Listas anidadas
- • Metodología de desarrollo: Incremental con pruebas

4.2 Representación de Datos

Cada nodo del árbol se representa como una lista de tres elementos:

9. 1. Valor del nodo (nombre del partido)
10. 2. Subárbol izquierdo (partido previo 1)
11. 3. Subárbol derecho (partido previo 2)

Estructura: [nombre_partido, subárbol_izquierdo, subárbol_derecho]

5. IMPLEMENTACIÓN

5.1 Funciones Principales

El sistema se implementó con las siguientes funciones principales:

```
def crear_partido(nombre):
    """Crea un nuevo nodo que representa un partido."""
    return [nombre, [], []]

def asignar_subpartidos(nodo, izq, der):
    """Asigna los partidos previos (hijos) a un partido dado."""
    nodo[1] = izq  # Hijo izquierdo
    nodo[2] = der  # Hijo derecho
```

6. RESULTADOS Y ANÁLISIS

6.1 Funcionalidad Lograda

Se logró construir exitosamente un sistema que:

- • Representa torneos como árboles binarios usando listas
- • Visualiza la estructura jerárquica de forma clara
- • Simula el progreso de partidos dinámicamente
- • Implementa todos los recorridos estándar de árboles
- • Proporciona funciones de análisis (altura, conteo de nodos)

6.2 Ventajas del Enfoque

- • Simplicidad conceptual: Fácil de entender y implementar
- • Bajo nivel de complejidad sintáctica
- • Ideal para estudiantes principiantes
- • Visualización clara de la estructura jerárquica
- • No requiere conocimientos de POO

6.3 Desventajas del Enfoque

- • Menor flexibilidad y extensibilidad que implementaciones con clases
- • Ausencia de encapsulamiento de datos
- • Dificultad para implementar árboles balanceados automáticamente
- • Menos eficiente en memoria para árboles grandes

7. CONCLUSIONES

7.1 Conclusiones Generales

La implementación de árboles binarios mediante listas en Python ofrece una alternativa simple pero poderosa para comprender la estructura y operación de árboles binarios. Este enfoque permite centrarse en la lógica estructural del árbol sin distracciones de programación orientada a objetos.

7.2 Aprendizajes Obtenidos

- • Comprensión profunda de la estructura de árboles binarios
- • Aplicación práctica de algoritmos recursivos
- • Análisis de complejidad algorítmica
- • Modelado de problemas reales con estructuras de datos
- • Importancia de la elección apropiada de estructuras de datos

7.3 Recomendaciones

Se recomienda su uso en:

- • Cursos introductorios de estructuras de datos
- • Entornos donde se requiera una solución rápida y sencilla
- • Contextos educativos para enseñar recursión
- • Proyectos de demostración de conceptos

8. BIBLIOGRAFÍA

Fuentes Primarias:

- • Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). Data Structures and Algorithms. Addison-Wesley Publishing Company.
- • Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- • Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley Professional.
- • Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.

Fuentes Secundarias:

- • Graham, R. L., Knuth, D. E., & Patashnik, O. (1994). Concrete Mathematics: A Foundation for Computer Science (2nd ed.). Addison-Wesley Professional.
- • Miller, B. N., & Ranum, D. L. (2013). Problem Solving with Algorithms and Data Structures using Python (3rd ed.). Franklin, Beedle & Associates.
- • Weiss, M. A. (2011). Data Structures and Algorithm Analysis in C++ (4th ed.). Pearson.

Documentación Técnica:

- • Python Software Foundation. (2024). Python 3.12 Documentation. https://docs.python.org/3/
- • Python Software Foundation. (2024). Data Structures Tutorial. https://docs.python.org/3/tutorial/datastructures.html