

Trabajo Integrador Final

Materia: Programación I

Profesor/a: Julieta Trapé, Ariel Enferrel

Tutor/a: Sofia Raia, Ramiro Hualpa

Integrantes:

Joana Gamarra - Comisión 14

Virginia Pérez - Comisión 4

UTN FRSN - Primer cuatrimestre - 2025

Índice

Introducción	3
Desarrollo teórico	3
Desarrollo práctico	15
Conclusión	18
Bibliografía	19

Introducción

En las siguientes páginas abordaremos los conceptos teóricos de los procesos de estructura de datos avanzados: los árboles como jerarquía de organización de datos para una búsqueda eficiente de éstos. Hemos seleccionado el enfoque de Árbol Binario de búsqueda que consideramos se adecúa más a las necesidades requeridas del caso práctico propuesto y desarrollado.

Nos enfocamos en primeramente pensar un proyecto práctico con una funcionalidad real y concreta, para luego buscar información al respecto, casos similares en la web y finalmente ofreciendo las propuestas de diferentes tipos de árbol y el objetivo del proyecto para consulta con IA.

El proyecto elegido consta de la creación de listas de estudiante/nota siendo el usuario final un docente, quien teniendo una gran cantidad de alumnos necesita un programa que permita la incorporación del estudiante con su respectiva nota. Pudiendo luego buscar tanto a dicho estudiante u otro ingresado anteriormente en la lista. Se ofrecen 2 opciones, una para seleccionar que se muestren sólo a los aprobados y otra para mostrar sólo a los desaprobados. Se agrega además una opción de salida para finalizar el programa.

Desarrollo teórico

Importancia de los árboles

Los **árboles** son las estructuras de datos más utilizadas en el área de la informática, ya que es un método sumamente eficiente para distintos tipos de búsquedas grandes y complejas. Se caracterizan por **no ser lineales** y representar **relaciones jerárquicas**, donde la posición más alta la va a ocupar el nodo padre y se va ramificando obteniendo como resultado distintos niveles de jerarquía, en los que van a haber nodos hijos y nodos hermanos relacionados con su nodo padre.

“Se les llama estructuras dinámicas, porque las mismas pueden cambiar tanto de forma como de tamaño durante la ejecución del programa. Y estructuras no

lineales porque cada elemento del árbol puede tener más de un sucesor.” (Facultad de Estadística e Informática, 2021)

Es fundamental lograr entender cómo funcionan los árboles, ya que pueden ser muy útiles para mejorar, a través de su análisis, la eficiencia de algoritmos como el de búsqueda y ordenamiento, permitiendo así implementar algoritmos con mejor complejidad, además de optimizar recursos muy valiosos como lo son el tiempo de ejecución y la memoria.

Elementos de un árbol

Los árboles son representados a través de **grafos**, los cuales son estructuras de datos que se utilizan para plasmar las relaciones que pueden haber entre objetos. Estos grafos están compuestos por **nodos**, que son unidades básicas dentro de un árbol y se relacionan entre sí a través de arcos o ramas. La relación se da de forma jerárquica, es decir, es una relación de padre e hijos, donde cada nodo puede tener más de un hijo, pero un solo padre.

Se pueden clasificar según dónde estén ubicados en el árbol y según su relación con otros nodos.

Clasificación de nodos según la ubicación que tienen en el árbol:

- **Nodo raíz:** es el único nodo que no tiene padre, este se encuentra ubicado en la parte superior del árbol y es único, por lo que no pueden haber dos nodo raíz en un mismo árbol.
- **Nodo hoja:** son los nodos que no tienen hijos, siempre se van a encontrar en los extremos de la estructura del árbol.
- **Nodo rama o interno:** estos nodos se caracterizan por tener un padre y al menos un hijo, es decir, que tienen más ramificaciones debajo de ellos, propagando así la jerarquía del árbol.

Clasificación de nodos según su relación con otros nodos:

Nodo Padre: es aquel que tiene hijos, que se encuentran en conexión con él, pueden ser uno o más.

Nodo hijo: es aquel que se encuentra conectado con un nodo padre, por lo que su existencia depende de él y a su vez, puede ser padre de otros nodos.

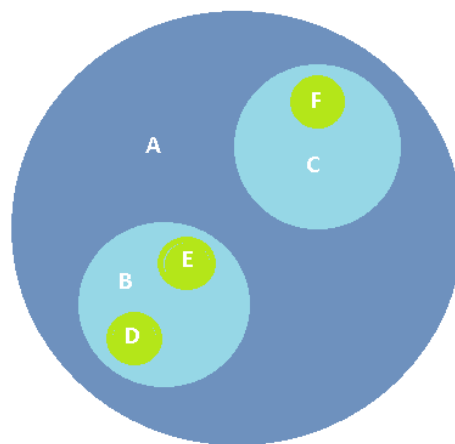
Nodo hermano: son aquellos que tienen el mismo padre dentro de la estructura, es por ello que se caracterizan por encontrarse a la misma distancia del nodo raíz y al mismo tiempo están en el mismo nivel jerárquicamente.

De este modo se va formando una estructura jerárquica donde los nodos hijos se encuentran debajo del nodo padre y los nodos hermanos se encuentran en el mismo nivel jerárquico.

Representaciones gráficas de un árbol

Un árbol puede tener distintos tipos de representaciones gráficas, que sirven para visualizar y analizar cómo se relacionan los elementos entre sí.

- **Conjuntos anidados:** utiliza la teoría de conjuntos. El nodo raíz se muestra como el conjunto principal y sus nodos hijos aparecen como subconjuntos dentro de él. A su vez, cada subconjunto puede contener otros subconjuntos.

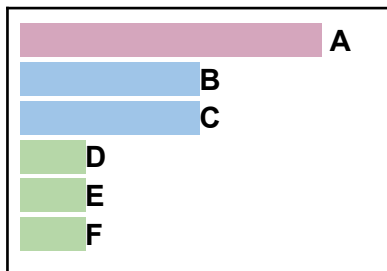


- **Paréntesis anidados:** esta representación es una forma textual de mostrar la estructura jerárquica de un árbol, donde cada nodo se escribe seguido de sus hijos entre paréntesis y el paréntesis más externo representa al nodo raíz. Si un nodo no tiene hijos entonces no se colocan paréntesis después de él, los hijos se separan por comas. Es muy útil para traducir el árbol a un lenguaje de programación.

```
( A ( B ( D, E ), C ( F ) ) )
```

- **Indentación:** esta representación usa indentaciones, es decir, espacios al comienzo de cada línea, para mostrar qué nodos están por encima o por debajo de otros.

Cada nivel del árbol es representado con una sangría más que el anterior. Cada nodo va en una línea distinta, y los hijos se colocan debajo de su padre, con menor indentación.



Propiedades de los árboles

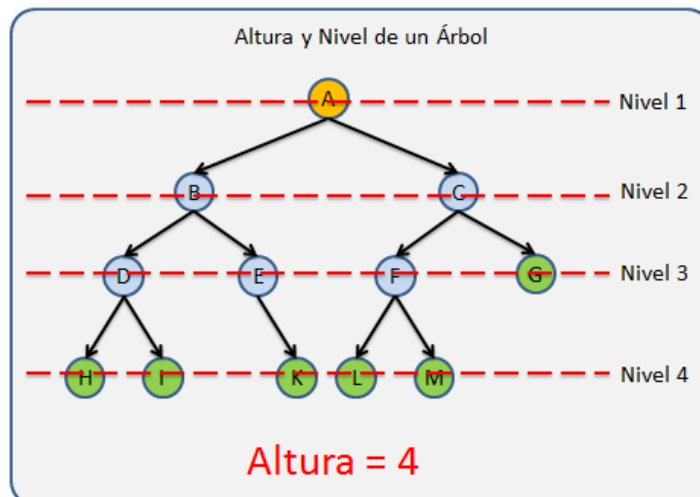
- Longitud de camino y profundidad

Podemos definir un **camino** como el recorrido o trayecto que hay que seguir para ir desde un nodo a otro ubicados ambos en el mismo árbol. O como una lista de ramas sucesivas necesarias para ir de un punto A a un punto B.

Por su parte la **longitud de camino** representa el número o cantidad de ramas por las cuales se debe pasar para llegar de un nodo a otro.

La **profundidad** está determinada por la distancia o longitud que hay entre un punto o nodo y el nodo raíz o principal (el cuál tiene una profundidad 0, ya que del nodo raíz al nodo raíz no hay longitud al pertenecer al mismo nivel).

- Nivel y altura



Los niveles se encuentran divididos por las líneas punteadas del gráfico de arriba y como podemos observar es posible encontrar más de un nodo por nivel pero es requisito indispensable que al menos haya uno.

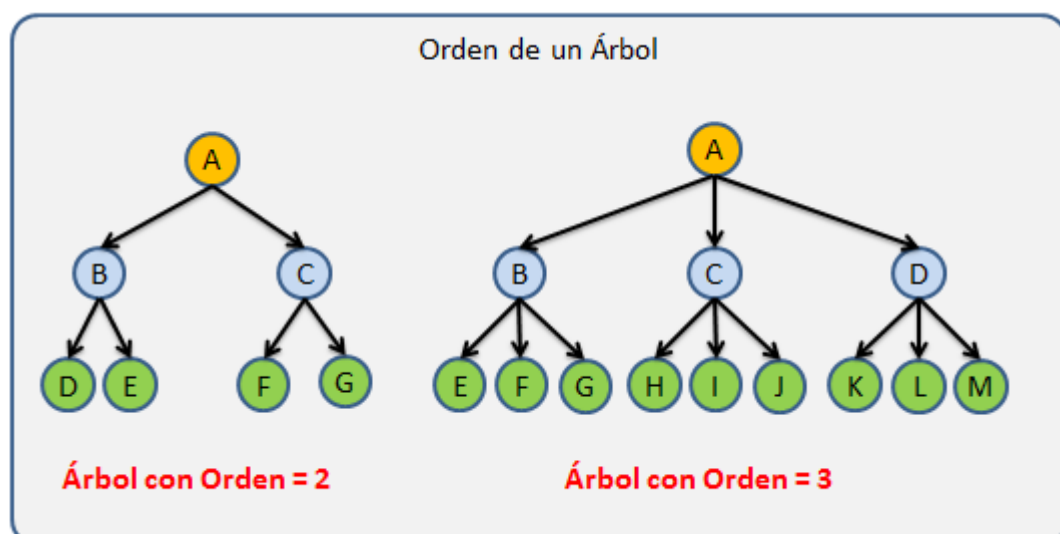
Dicho esto y habiendo representado de forma gráfica el concepto podemos definir al **nivel de un nodo** es la longitud o espacio “escalonado” que hay entre el nodo raíz (nivel 1) y cualquier otro nodo dentro del árbol que se calcula contando la cantidad de nodos que existen sobre el nodo raíz +1 y viceversa.

La **altura** se encuentra definida por la cantidad de “escalones” totales que abarcan el árbol. En el ejemplo de arriba podemos ver como la suma de todos los niveles integran el concepto trabajado en éste párrafo.

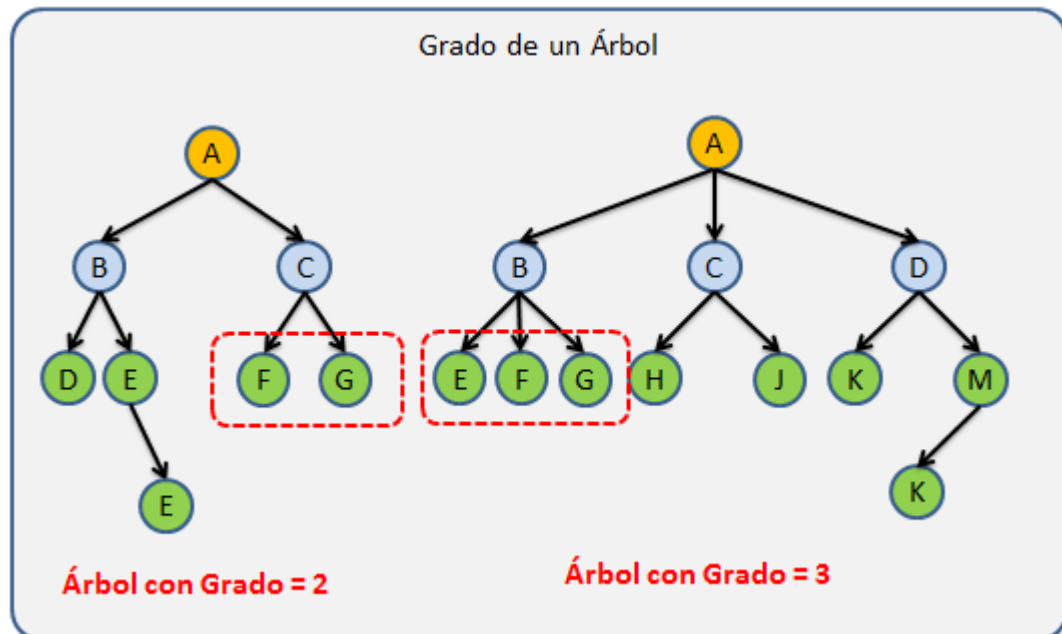
“Se calcula mediante recursividad tomando el nivel más grande de los dos sub-árboles de forma recursiva de la siguiente manera:” (1)

$$altura = \max(altura(hijo1), altura(hijo2), altura(hijoN)) + 1$$

- Grado y orden



El **orden** se encuentra representado por el número máximo de hijos que puede tener un nodo. Un árbol de orden 1 por ejemplo tendría una estructura lineal ya que sólo soportaría 1 hijo por nodo no pudiendo bifurcarse en ninguna de sus ramas

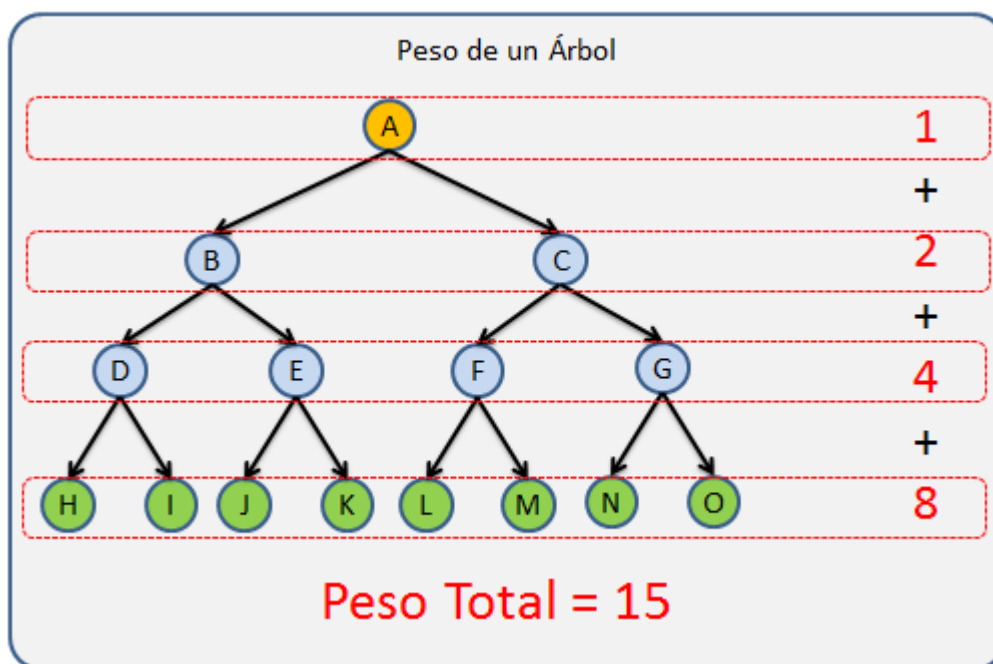


El **grado** es la mayor cantidad de hijos que tiene un nodo del árbol en cuestión y cuyo valor máximo está determinado por el número de orden.

“El grado se calcula contando de forma recursiva el número de hijos de cada sub-árbol hijo y el número de hijos del nodo actual para tomar el mayor, esta operación se hace de forma recursiva para recorrer todo el árbol.” (2)

$grado = \max(\text{contarHijos}(\text{hijo1}), \text{contarHijos}(\text{hijo2}), \dots, \text{contarHijos}(\text{hijoN}), \text{contarHijos}(\text{this}))$

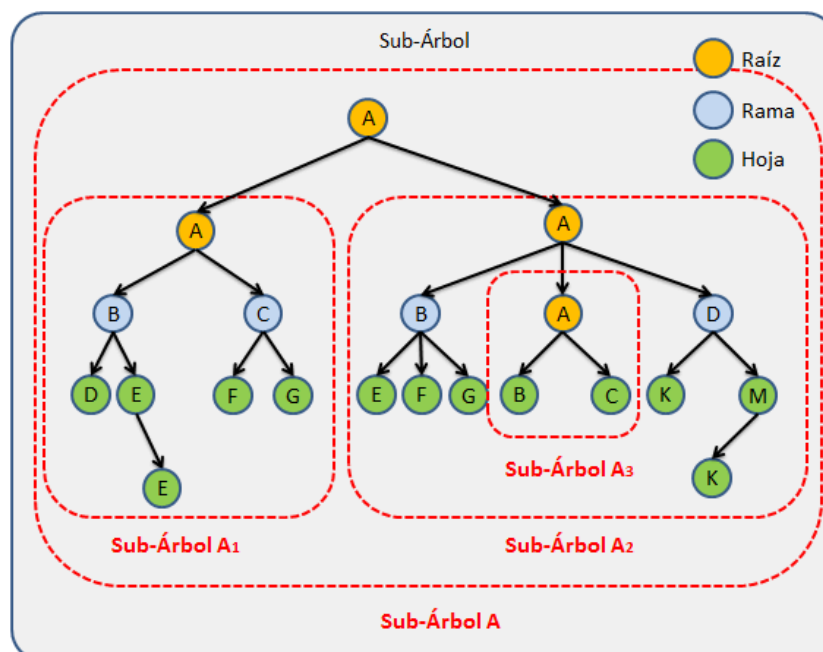
- Peso



El **peso** se ve reflejado por la suma total de los nodos que se encuentran incluidos en un mismo árbol. Una mayor cantidad de nodos totales representa un peso mayor y por lo tanto la cantidad de memoria en la ejecución de un código.

★ Sub-Árbol

A fin de comprender los conceptos relacionados al cálculo del grado, altura y peso de manera recursiva se decide agregar un pequeño apartado al tema.



En la imagen de arriba podemos observar como los sub-árboles no son más que subdivisiones realizadas dentro del árbol principal. “...podemos decir que un Árbol es un nodo Raíz con N Sub-Árboles.” (3) Este concepto sirve para utilizar el “Divide y

Vencerás” que nos permite extraer un sub-árbol y trabajarlo o analizarlo de manera independiente al resto.

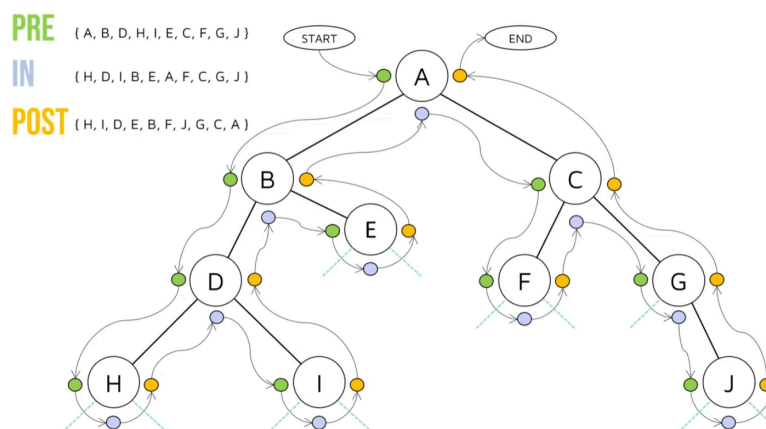
Tipos de árboles

❖ Árboles binarios

Es un árbol de estructura jerárquica de orden 2. Lo que significa, como desarrollamos arriba que 2 será la cantidad máxima de hijos permitidos por nodo. Al incluir dos como máximo es posible asignar un concepto genérico para referirnos a sus hijos. Así el nodo de la izquierda (que ocupa ese puesto) será definido en el código de una manera similar como *def nodo_izquierda* y lo mismo se hará con el derecho *def nodo_derecho*. La función principal es ser la base de otras estructuras más complejas que se detallarán más adelante.

- Formas de recorrer árboles binarios:

Comenzamos por definir el recorrido dentro de un árbol como el orden que se sigue al pasar entre nodo y nodo para completar una búsqueda. A continuación se desarrollarán los 3 tipos de búsqueda existentes:

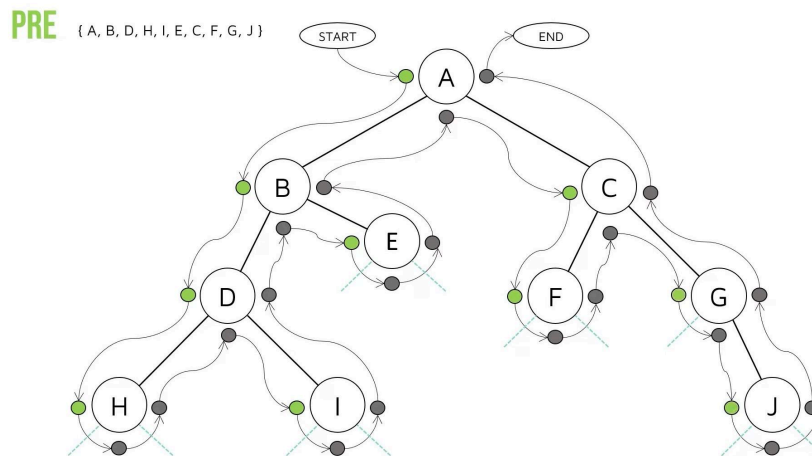


(4) Imagen ilustrativa del recorrido de los elementos de un árbol.

Tipos de recorridos:

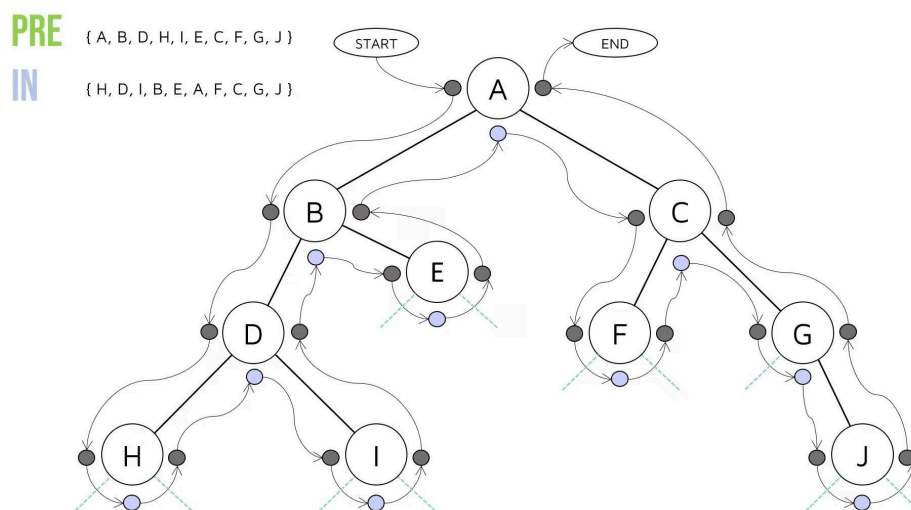
- Preorden:

En este formato de recorrido la búsqueda se inicia en el nodo raíz y comienza a desplazarse siempre por el lado izquierdo de cada uno de los nodos, como puede verse en el ejemplo gráfico de abajo.



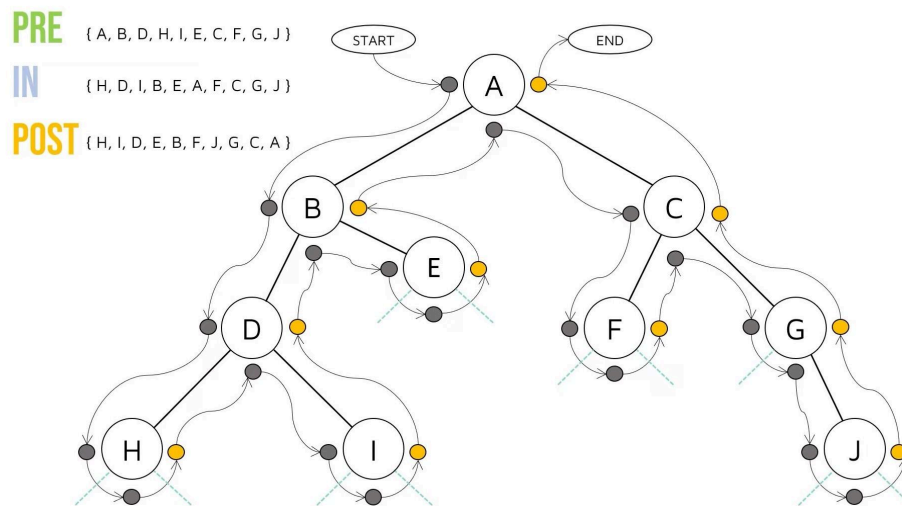
- Inorden:

Implica el inicio por la última hoja izquierda del nivel más bajo (en éste caso H de nivel 4). Como se puede observar en el gráfico se pasa por el resto de los nodos más próximos, siempre priorizando el lado izquierdo para luego terminar con ese costado, pasar por el nodo raíz y luego finalizar la búsqueda por los componentes más a la izquierda del ramal derecho.



- Postorden:

Podemos observar que en éste modelo también el recorrido comienza por el lado izquierdo con la salvedad de que en vez de continuar su recorrido con el nodo padre continúa con el hermano. Una vez completado el nodo de su nivel sube arriba para volver a hacer los mismo. Además de esa diferencia se ve que en vez de circular por el nodo raíz, el recorrido se desplaza por el lado derecho y finalizado éste último termina en el nodo principal.



Nota: Estas formas de recorridos son aplicables a todo tipo de árbol que tenga de base el modelo binario. Pueden incluirse en este caso: árbol n-ario, binario de búsqueda, etc.

❖ Binario de Búsqueda

Es, por definición, un árbol binario organizado por valores. Acomoda y organiza los valores en relación a su peso o número. Posicionando a la izquierda los de menor valor y a la derecha los de mayor valor. Este es el árbol elegido para el desarrollo práctico del programa seleccionado.

Nos detuvimos en considerar las variables de los árboles de búsqueda binaria para nuestro caso que son:

Ventajas:

- La posibilidad de recorrer los nodos en forma preorden, inorden, postorden.
- Inserción y eliminación de forma ordenada, ya que se mantiene el orden de manera automática.
- Permite encontrar elementos de forma eficiente.

Desventajas:

- La eliminación de un nodo puede ser complicada, ya que puede generar errores si no se implementa bien.

- Si se requiere mayor eficiencia y rendimiento estable, es preferible el uso de otros árboles binarios.
-

❖ AVL/Rojo-Negro

Son Árboles binarios por búsqueda que aplican reglas para mantener el equilibrio automáticamente entre todas sus ramas y nodos.

❖ Heap

También es un árbol binario, pero su prioridad se basa en la relación entre padre e hijos (Min-Heap / Max-Heap).

❖ Trie

No es estrictamente binario, ya que cada nodo puede tener múltiples hijos basados en caracteres.

❖ B/B+

Se diseñan para bases de datos y pueden tener múltiples hijos por nodo (no son estrictamente binarios).

❖ N-Ario

Generaliza el concepto de árboles, permitiendo n hijos por nodo.

❖ Árbol de decisión

En aprendizaje automático, pueden tener más de dos ramas dependiendo de las condiciones.

Los primero 3 de estos árboles **derivan directamente** de estructuras binarias, mientras que otros **expanden el concepto** para manejar datos de manera más eficiente según la aplicación.



Tabla resumen:

Tipo de Árbol	Hijos por nodo	Usos principales
Binario	2	Base de otras estructuras
Binario de búsqueda	2	Ordenamiento y búsqueda
AVL / Rojo-Negro	2	Balanceo automático
Trie	variable	Búsqueda de palabras / autocompletado
B/B+	muchos	Sistemas de archivos, bases de datos
N-ario	muchos	Jerarquías como menús, carpetas, genealogía
Árbol de decisión	variable	Clasificación, IA, toma de decisiones
Heap	2	Heapsort, colas de prioridad

(5) Imagen ilustrativa del resumen de los tipos de árbol.

Desarrollo práctico

Caso práctico desarrollado en Python

```

1
2 # Definimos una clase ArbolBusqueda que contendrá la raíz del árbol y
3 # métodos para insertar estudiantes.
4 class ArbolBusqueda:
5     # Inicializamos el árbol de búsqueda con una raíz vacía.
6     def __init__(self):
7         self.raiz = None
8
9     # Método para insertar un estudiante en el árbol.
10    def insertar(self, estudiante):
11        self.raiz = self._insertar_recursivo(self.raiz, estudiante)
12
13    # Método recursivo para insertar un estudiante en el árbol de búsqueda.
14    def _insertar_recursivo(self, nodo, estudiante):
15        if nodo is None:
16            return Nodo(estudiante)
17        if estudiante.calificacion < nodo.valor.calificacion:
18            nodo.izquierda = self._insertar_recursivo(nodo.izquierda, estudiante)
19        else:
20            nodo.derecha = self._insertar_recursivo(nodo.derecha, estudiante)
21        return nodo
22
23    # Método para buscar un estudiante por nombre en el árbol.
24    def buscar(self, nombre):
25        return self._buscar_recursivo(self.raiz, nombre)
26
27    # Función recursiva para buscar un estudiante por nombre en el árbol.
28    def _buscar_recursivo(self, nodo, nombre):
29        if nodo is None:
30            return None
31        if nodo.valor.nombre.lower() == nombre.lower():
32            return nodo.valor
33        encontrado = self._buscar_recursivo(nodo.izquierda, nombre)
34        if encontrado is None:
35            encontrado = self._buscar_recursivo(nodo.derecha, nombre)
36        return encontrado
37
38    # Método para mostrar los estudiantes en orden (preorden, inorden, postorden).
39    def mostrar_inorden(self, nodo):
40        if nodo:
41            self.mostrar_inorden(nodo.izquierda) #hijo izquierdo
42            print(nodo.valor)
43            self.mostrar_inorden(nodo.derecha) #hijo derecho
44
45    def mostrar_preorden(self, nodo):
46        if nodo:
47            print(nodo.valor)
48            self.mostrar_preorden(nodo.izquierda) #hijo izquierdo
49            self.mostrar_preorden(nodo.derecha) #hijo derecho
50
51    def mostrar_postorden(self, nodo):
52        if nodo:
53            self.mostrar_postorden(nodo.izquierda) #hijo izquierdo
54            self.mostrar_postorden(nodo.derecha) #hijo derecho
55            print(nodo.valor)
56
57    # Métodos para mostrar los estudiantes aprobados y desaprobados.
58    def mostrar_aprobados(self, nodo):
59        if nodo:
60            self.mostrar_aprobados(nodo.izquierda)
61            if nodo.valor.calificacion >= 60:
62                print(nodo.valor)
63            self.mostrar_aprobados(nodo.derecha)
64
65    def mostrar_desaprobados(self, nodo):
66        if nodo:
67            self.mostrar_desaprobados(nodo.izquierda)
68            if nodo.valor.calificacion < 60:
69                print(nodo.valor)
70            self.mostrar_desaprobados(nodo.derecha)
71
72    # Definimos una clase Estudiante que contendrá el nombre y la calificación del estudiante.
73    class Estudiante:
74        def __init__(self, nombre, calificacion):
75            self.nombre = nombre
76            self.calificacion = calificacion
77
78        def __repr__(self):
79            return f"Estudiante: {self.nombre} - Calificación: {self.calificacion}"
80
81    # Para la creación de un árbol debemos definir una clase Nodo que contenga un valor y referencias a sus hijos
82    class Nodo:
83        def __init__(self, estudiante):
84            self.valor = estudiante
85            self.izquierda = None
86            self.derecha = None
87

```



```

91
92 # ----- Menú -----
93
94 def menu():
95     print("--- Menú Gestor de Calificaciones ---")
96     print("1. Insertar estudiante")
97     print("2. Buscar estudiante")
98     print("3. Mostrar todos (preorden, inorden, postorden)")
99     print("4. Mostrar aprobados (>= 60)")
100    print("5. Mostrar desaprobados (< 60)")
101    print("6. Salir")
102    return input("Seleccione una opción: ")
103
104
105 # ----- Programa Principal -----
106
107 arbol = ArbolBusqueda()
108
109 # Insertamos algunos estudiantes iniciales
110 for nombre, nota in [("Alicia", 85), ("Bastian", 75), ("Carlos", 90), ("David", 55), ("Emily", 95), ("Federico", 30)]:
111     arbol.insertar(Estudiante(nombre, nota))
112
113 seguir = True # Variable para controlar el bucle del menú, se corta si se selecciona la opción de salir.
114 # Bucle principal del programa. Muestra el menú y permite interactuar con el árbol.
115 while seguir:
116     opcion = menu()
117     if opcion == "1":
118         nombre = input("Nombre del estudiante: ")
119
120         # Validamos que el nombre contenga solo letras o espacios
121         if not nombre.replace(" ", "").isalpha():
122             print("Nombre inválido. Debe contener solo letras y espacios.")
123             continue
124
125         nombre = nombre.lower().strip() # Normalizamos el nombre
126
127         # Verificamos si el estudiante ya existe en el árbol
128         if arbol.buscar(nombre):
129             print("El estudiante ingresado ya existe.")
130             continue
131
132         try:
133             nota = float(input("Calificación (1 a 100): "))
134             if 1 <= nota <= 100:
135                 arbol.insertar(Estudiante(nombre, nota))
136                 print("Estudiante insertado.")
137             else:
138                 print("Calificación inválida.")
139         except ValueError:
140             print("Entrada no válida.")
141
142     elif opcion == "2":
143         nombre = input("Nombre a buscar: ")
144         resultado = arbol.buscar(nombre)
145         if resultado:
146             print(f"Encontrado: {resultado}")
147         else:
148             print("Estudiante no encontrado.")
149
150     elif opcion == "3":
151         print("--- Estudiantes (preorden) ---")
152         arbol.mostrar_preorden(arbol.raiz)
153         print("--- Estudiantes (inorden) ---")
154         arbol.mostrar_inorden(arbol.raiz)
155         print("--- Estudiantes (postorden) ---")
156         arbol.mostrar_postorden(arbol.raiz)
157
158     elif opcion == "4":
159         print("--- Aprobados (>= 60) ---")
160         arbol.mostrar_aprobados(arbol.raiz)
161
162     elif opcion == "5":
163         print("--- Desaprobados (< 60) ---")
164         arbol.mostrar_desaprobados(arbol.raiz)
165
166     elif opcion == "6":
167         print("Saliendo...")
168         seguir = False # Termina el bucle del menú.
169     else:
170         print("Opción inválida.")

```

División de tareas entre los integrantes

A lo largo del desarrollo del Trabajo Integrador, la división de tareas fue clave para optimizar el tiempo y mejorar la eficiencia del equipo. Esta organización nos permitió enfocarnos en subtarefas específicas que se podían ir desarrollando en poco tiempo. Se llevaron a cabo reuniones periódicas para evaluar el avance de cada integrante, debatir sobre decisiones importantes y proponer nuevas ideas para implementar en el proyecto. Asimismo, se establecieron plazos para la finalización de cada tarea, lo que facilitó cumplir con los objetivos en tiempo y forma. La comunicación entre los integrantes fue constante y clara, hubo intercambios de ideas, lo cual favoreció la elaboración de un trabajo bien estructurado, con conceptos definidos y coherentes.

Durante el desarrollo del Trabajo Práctico, los integrantes del grupo realizaron una investigación sobre las *funciones* necesarias para implementar una estructura de Árbol binario, así como sobre el uso de las *class* (clases). También se repasaron contenidos de unidades previas, una vez concluida la investigación, se trabajó de forma colaborativa en la programación del código hasta finalizarlo con éxito.

Conclusión

El desarrollo de este Trabajo Integrador nos permitió aplicar e integrar diversos conocimientos adquiridos a lo largo del curso. A pesar de que la consigna sugería implementar el proyecto utilizando listas, decidimos orientarlo a objetos, ya que nos despertó curiosidad aprender sobre esta metodología y consideramos que ofrecía una mayor flexibilidad, claridad y escalabilidad.

La implementación no representó grandes dificultades; contamos con el apoyo de herramientas de inteligencia artificial que nos ayudaron a comprender mejor el enfoque orientado a objetos y su funcionalidad. Haber tomado esta decisión fue una experiencia muy enriquecedora, ya que nos motivó a investigar más allá de lo propuesto y nos permitió adquirir nuevos conocimientos y habilidades. Sin lugar a dudas, nos llevamos un aprendizaje muy valioso que refuerza nuestra actitud proactiva frente a los desafíos.

Elegir un enfoque más complejo nos exigió un compromiso real con la investigación, no se trató solo de leer y comprender, sino de aplicar lo aprendido en la práctica, donde realmente se ponía a prueba la calidad del trabajo previo y se identificaban posibles temas a reforzar. El proceso nos llevó a enfrentarnos a conceptos nuevos como la programación orientada a objetos y a profundizar en el funcionamiento de los árboles binarios. Gracias a esto, no solo


logramos desarrollar un programa funcional, sino que también adquirimos herramientas que nos serán útiles en futuros proyectos académicos y profesionales.

Desde lo personal y grupal, aprendimos a organizarnos mejor, a dividir tareas de forma eficiente y a mantener una comunicación clara. A nivel técnico, incorporamos nuevas formas de pensar la lógica de un programa, de estructurar datos y de resolver problemas de forma más eficiente. Sentimos que este trabajo no solo aportó a nuestro aprendizaje individual, sino también a nuestra capacidad de trabajo colaborativo.

En definitiva, este proyecto fue una oportunidad para demostrar que con motivación, esfuerzo y trabajo en equipo es posible ir más allá de lo esperado, asumiendo desafíos que enriquecen nuestro recorrido formativo.

Referencias bibliográficas

Facultad de Estadística e Informática, (2021). *Estructura de datos: Apuntes de clase* [Archivo PDF]. [Clase8-Arboles](#)

Universidad Tecnológica Nacional, (2025). Árboles: *Apuntes de cátedra* [Archivo PDF].  [Árboles.pdf](#)

Imágenes obtenidas de propiedades de los árboles de:

<https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/>

(1)(2)(3)[Estructura de datos - Árboles - Oscar Blancarte - Software Architecture](#)

(4)https://www.youtube.com/watch?v=Jo2euX89Oz8&ab_channel=AprendeSinEspinass

(5) imagen obtenida de IA:

<https://chatgpt.com/>

 Herramienta de edición de video: Microsoft Clipchamp.