

Introducción a Trees

Hasta el momento, las estructuras de datos que hemos estudiado incluyen arraylists, linked lists, stacks, y queues. Estas estructuras se comportan de manera lineal. Son útiles para almacenar y modificar datos, pero presentan limitaciones cuando necesitamos representar relaciones o niveles jerárquicos. Hoy introduciremos una nueva estructura de datos. Los trees, o árboles, son estructuras jerárquicas.

Trees

Con los trees, cada elemento o nodo puede tener un único elemento anterior (predecessor) o múltiples elementos posteriores (children). Algunos ejemplos comunes de relaciones o datos que se pueden representar con árboles incluyen árboles genealógicos, el sistema de carpetas en los sistemas operativos y los decision trees, que se utilizan en el ámbito de Data Science para crear modelos de Machine Learning.

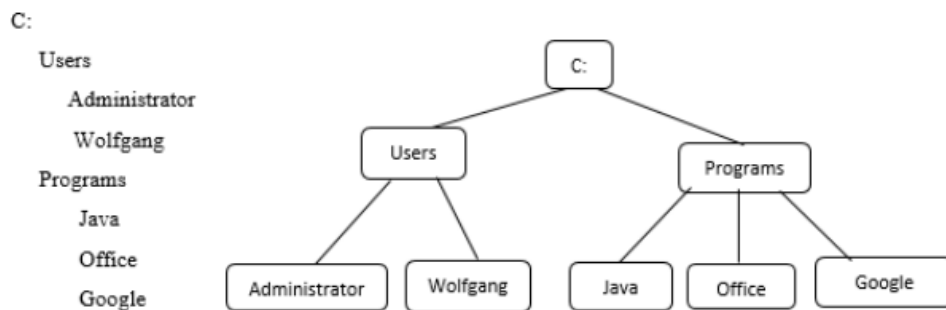


Ilustración 1: Representación de la Jerarquía de folders de un OS

Tree Terminología

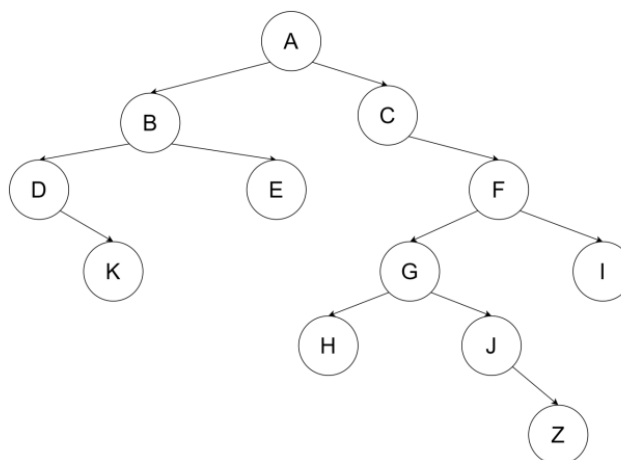


Ilustración 2: Ejemplo de un Binary Tree

Los árboles consisten en una colección de nodos. Los **nodos** son similares a los nodos que hemos visto en los linked lists, ya que contienen un valor y referencias a sus hijos. Por ejemplo en la ilustración 2, el nodo B tiene como valor "B" y las referencias a sus hijos, D y E. Al igual que en los árboles reales, los trees poseen una raíz (**root**), que es el nodo principal, siendo el nodo A.

Utilizando la ilustración 2, El nodo A es el root y tiene como hijos a B y C, mientras que D está referenciado por B. Otra forma de entender los nodos es compararlos con dicts o hashmaps, ya que tienen un key (el valor del nodo) y un value (sus hijos). Las conexiones entre nodos se llaman **branches o edges**. Cada nodo en un árbol debe tener un padre, excepto el root. Por ejemplo, el nodo A, siendo el root, no es referenciado por ningún otro nodo. Los nodos que comparten el mismo padre son llamados **siblings** (hermanos). Los nodos B y C son hermanos, al igual que G e I. Los nodos que no tienen hijos se llaman **leaf nodes (nodos externos)**, y los que tienen al menos un hijo son llamados **nodos internos**.

Los leaf nodes son K, E, H, I y Z, mientras que los nodos internos son A, B, C, D, F, G, J entre otros. Además, la relación padre-hijo puede generalizarse a la relación **ancestro-descendiente**. Los padres son un caso particular de ancestros, y los hijos son un caso particular de descendientes. En el árbol de la ilustración 2, el ancestro común de todos los nodos es el root, A. Un ancestro de E es B, pero no C. En cambio, Z es descendiente de G.

Los árboles son **estructuras recursivas**. Un árbol puede dividirse en varios subtrees, y cada conjunto de nodos puede formar un tree independiente que es un subtree del árbol original. Incluso, un solo nodo (nodos externos) también se considera un subtree. Esta propiedad de dividir el problema en versiones más simples nos recuerda al concepto de recursión, por lo que los árboles son considerados una estructura recursiva, ilustración 3.

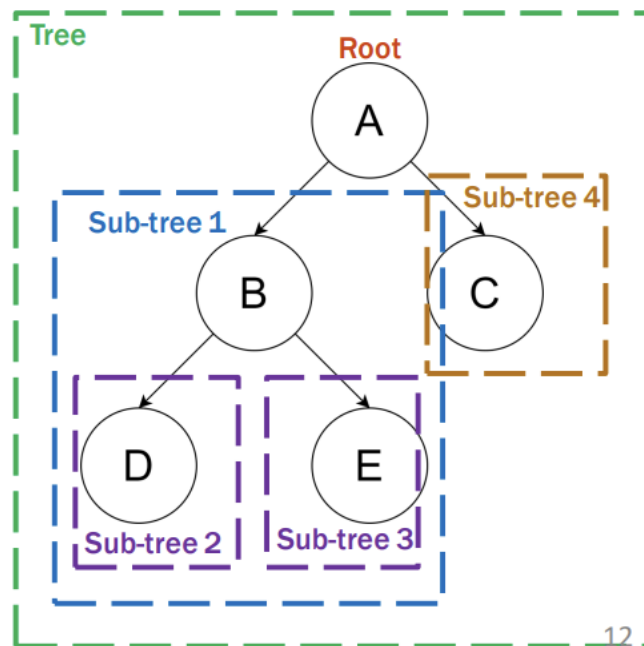


Ilustración 3: Sub-Trees

Una secuencia de nodos conectados por edges consecutivos se llama path. En el árbol en la ilustración 2, el path $A \rightarrow B \rightarrow E$ tiene una longitud de 2, mientras que el path $F \rightarrow G \rightarrow J \rightarrow Z$ tiene una longitud de 3.

Gracias a la naturaleza jerárquica de los árboles, podemos dividir los nodos en diferentes niveles relativos al root. Para calcular el nivel en un tree, contamos la distancia desde el root, que es el número de edges entre un nodo y el root. Algunos libros consideran el primer nivel como 0, pero en este curso lo consideramos a partir de 1.

La altura (height) de un árbol es la distancia más larga desde el root hasta un leaf node. En este árbol, podemos ver que la altura es 5. La altura es relativa a cada nodo. Por ejemplo, el subtree que incluye B, D, y E tiene una altura de 1, pero si incluimos A, la altura aumenta a 2. El depth (profundidad) se refiere al path entre un nodo y el root, indicando qué tan lejos estamos del root. Por ejemplo, el depth de A es 0, el de Z es 5, y el de D es 2.

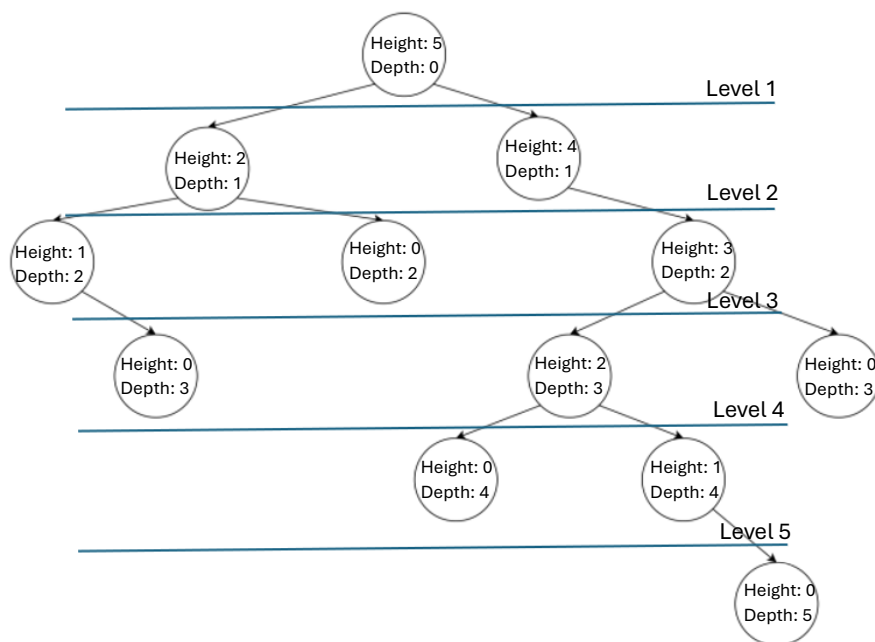


Ilustración 4: Height and Depth de un Tree

La diferencia entre height y depth es que height es la distancia mayor entre un nodo y su descendiente más lejano, mientras que depth es la distancia entre el root y un nodo. En el root, el depth es 0 y el height es el máximo path de sus descendientes. En los nodos externos o leaf nodes, el height es 0. El height se mide hacia abajo, mientras que el depth se mide hacia arriba. En este árbol, tanto el height como el depth son 5, ilustración 4.

Tipos de Trees

Los árboles se pueden clasificar en N-ary trees, donde cada nodo puede tener "n" hijos, ilustración 5. También existen los binary trees, donde cada nodo puede tener un máximo de 2 hijos. En este curso, nos enfocaremos en los binary trees. Un ejemplo común son los binary search trees, árboles ordenados. Un caso particular son los Huffman trees, que son binary trees usados para la compresión de texto.

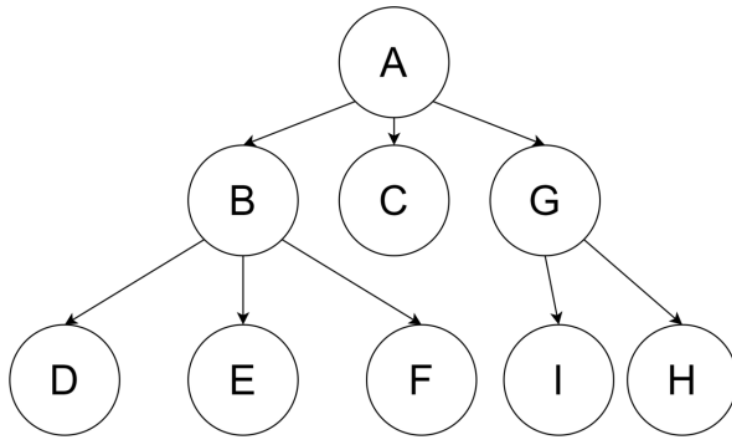


Ilustración 5: N-ary Tree Ejemplo

Clasificación de Binary Trees:

Un árbol cuyos paths tienen la misma longitud o difieren solo en uno se considera un árbol **Balanced**. Un árbol con todos sus niveles llenos, excepto el último, se considera **Complete**. Es importante mencionar que los niveles deben llenarse de izquierda a derecha. Un árbol **Full** es aquel donde cada nodo tiene 0 o 2 hijos.

Clasificación	Definición	Ejemplos	Ejemplos de Trees que no caen en la clasificación
Balanced Tree	Un árbol cuyos paths tienen la misma longitud o difieren solo en uno		
Complete Tree	Un árbol con todos sus niveles llenos, excepto el último. Los niveles deben llenarse de izquierda a derecha		
Full Tree	Cada nodo tiene 0 o 2 hijos.		

Tree Traversal

Existen tres maneras comunes de recorrer un tree. **In-order**, Se visita primero al hijo izquierdo, luego el nodo, y finalmente el hijo derecho (left, self, right). **Pre-order**, Primero se visita el nodo, luego el hijo izquierdo, y finalmente el derecho (self, left, right). Y **Post-order**, Se visita primero el hijo izquierdo, luego el derecho, y por último el nodo (left, right, self).

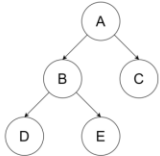
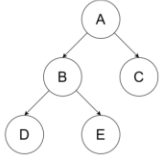
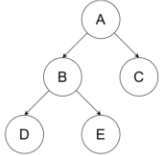
Tipo de Traversal	Código de Ejemplo	Tree de Ejemplo	Resultado del Traversal
In-order	<pre> If node is null return Else Go to left child Print data Go to right child </pre>	 <pre> graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) B --> E((E)) </pre>	{D, B, E, A, C}
Pre-order	<pre> If node is null return Else Print data Go to left child Go to right child </pre>	 <pre> graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) B --> E((E)) </pre>	{A, B, D, E, C}
Post-order	<pre> If node is null return Else Go to left child Go to right child Print data </pre>	 <pre> graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) B --> E((E)) </pre>	{D, E, B, C, A}

Ilustración 6: Ejemplos de Tree Traversals