Programación 2

Tecnicatura Universitaria en Inteligencia Artificial

2022

Estructuras de datos avanzadas: Árboles

1. Árbol (Tree)

Los árboles son estructuras jerárquicas que se utilizan para modelar una amplia variedad de problemas y situaciones. Si bien son objeto de estudio de las ciencias de la computación, es fácil encontrar ejemplos cotidianos en los que los árboles se utilizan para estructurar cierta información.

En este apunte comenzaremos introduciendo una primera noción de árbol con su terminología relacionada, para luego pasar a las subclases de árboles y sus posibles implementaciones. Finalizaremos el tema analizando distintas formas de recorridos sobre estas estructuras.

1.1. Un primer ejemplo

Los sistemas operativos como Linux organizan las carpetas y los archivos en *sistemas de archivos* que utilizan una estructura de árbol. La jerarquía se da por el hecho de que una carpeta contiene archivos y, a su vez, otras carpetas.

La Figura 1 muestra el resultado de ejecutar el comando `tree addressable-2.8.5`, el cual lista los contenidos del directorio `addressable-2.8.5` en formato de árbol. Dicha carpeta contiene el código fuente de una librería de *Ruby*, por lo que dentro de ella habrá a su vez otras carpetas (en color azul) y archivos (en blanco). En este caso, decimos que `addressable-2.8.5` es el *directorio raíz*, por ser el más alto en los niveles de la jerarquía. En la figura se pueden ver archivos como `LICENSE.txt` o `native.rb`, los cuales en el contexto de un árbol llamaremos *hojas*, por no tener nada por debajo de ellos (los archivos no tienen archivos o carpetas dentro de ellos). A su vez, siguiendo el camino de las distintas carpetas, podemos determinar que la ruta para el archivo `native.rb` es `addressable-2.8.5/lib/addressable/idna/native.rb`. Esta ruta presenta el único *camino* posible desde la raíz (`addressable-2.8.5`) hasta la hoja `native.rb`.

1.2. Definición formal

La definición formal de árbol es la siguiente:

Un **árbol** T es un conjunto de puntos, a los que llamamos vértices, que pueden estar unidos con líneas, a las que llamamos aristas, que satisface lo siguiente: si v y w son vértices en T, existe un único camino desde v a w. Un **árbol con raíz** es un árbol en el que un vértice específico se designa como raíz.

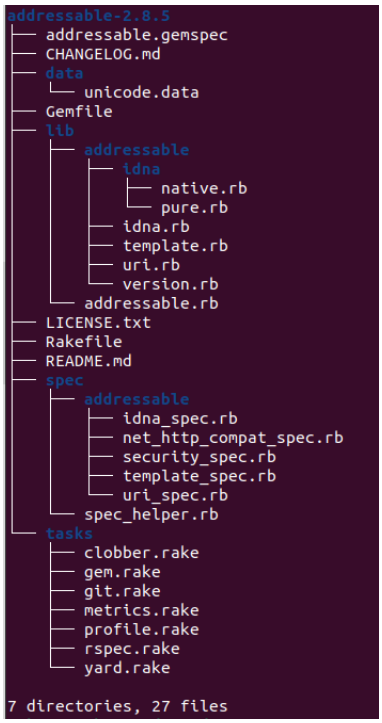

Figura 1: Despliegue de la carpeta addressable-2.8.5 en formato de árbol.

Volvamos a un ejemplo concreto para entender mejor esta definición. La Figura 2 muestra los resultados de las semifinales y finales de la competencia de tenis clásico en Wimbledon, que incluyó cuatro de los mejores jugadores en la historia del tenis. En Wimbledon, cuando un jugador pierde, sale del torneo. Los ganadores siguen jugando hasta que queda sólo una persona: el campeón.[1] La Figura 2 muestra que, en las semifinales, Mónica Seles venció a Martina Navratilova y Steffi Graf venció a Gabriela Sabatini. Las ganadoras, Seles y Graf, jugaron la final y Graf venció a Seles. Steffi Graf, al ser la única jugadora invicta se convirtió en la campeona de Wimbledon.

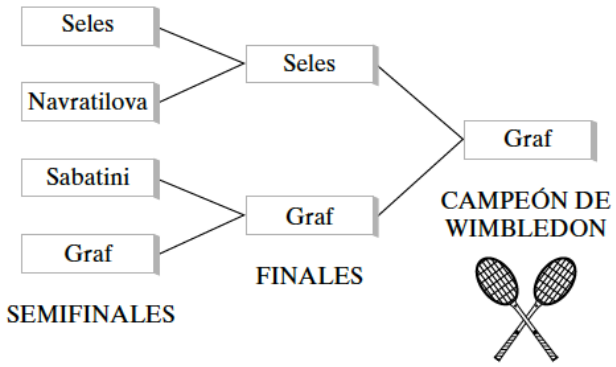

Figura 2: Semifinales y finales en Wimbledon.

Podemos modelar un torneo por eliminación sencilla como el de la figura 2 con un árbol. Es decir, con un conjunto de vértices que representarán a las jugadoras, y un conjunto de aristas que representarán cómo se configuran los distintos partidos en cada etapa del torneo. Esto puede verse en la Figura 3. A su vez, si se rota dicha figura, puede ser visto como un árbol natural (Figura 4).

[1] Esta competencia se conoce como torneo por eliminación sencilla.

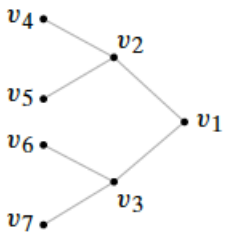

Figura 3: El torneo de la Figura 2 como un árbol.

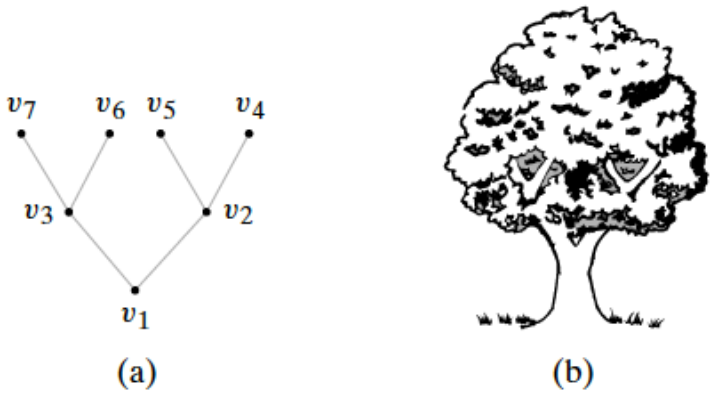

Figura 4: Árbol de la Figura 3 (a) girado, (b) comparado con un árbol natural.

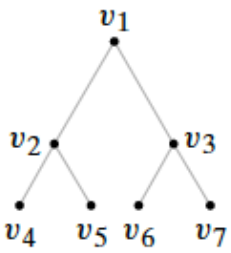

Figura 5: El árbol de la Figura 4 (a) con la raíz arriba.

Al contrario de los árboles naturales, cuyas raíces se localizan abajo, dibujaremos los árboles con raíces con la raíz hacia arriba. La Figura 5 presenta la forma en que se dibujaría el árbol de la Figura 3 (con v_1 como raíz). Primero, se coloca la raíz v_1 arriba. Abajo de la raíz y al mismo nivel, se colocan los vértices v_2 y v_3, a los que se puede llegar desde la raíz por una trayectoria simple de longitud 1. Abajo de estos vértices y al mismo nivel se colocan los vértices v_4, v_5, v_6 y v_7, a los que se llega desde la raíz por trayectorias simples de longitud 2. Se continúa así hasta dibujar el árbol completo.

Como la trayectoria simple de la raíz a cualquier vértice dado es única, cada vértice está en un nivel determinado de manera única. El nivel de la raíz es el nivel 0. Se dice que los vértices abajo de la raíz están en el nivel 1, y así sucesivamente. Entonces el nivel de un vértice v es la longitud de la trayectoria simple de la raíz a v. La altura de un árbol con raíz es el número máximo de nivel que ocurre.

Los vértices v_1, v_2, v_3, v_4, v_5, v_6, v_7 en el árbol con raíz de la Figura 5 están (respectivamente) en los niveles 0, 1, 1, 2, 2, 2, 2. La altura del árbol es 2.

1.3. Terminología asociada

Continuemos ahora introduciendo terminología relacionada a los árboles. Sea T un árbol con raíz v_0. Supongamos que x, y y z son vértices en T y que $(v_0, v_1, \cdots, v_n)$ es una trayectoria simple en T. Entonces:

- v_{n-1} es el **padre** de v_n.
- $v_0, \cdots, v_{n-1}$ son **ancestros** de v_n.
- v_n es un **hijo** de v_{n-1}.
- Si x es un ancestro de y, y es un **descendiente** de x.
- Si x e y son hijos de z, x e y son **hermanos**.
- Si x no tiene hijos, x es un vértice **terminal** (o una **hoja**).
- Si x no es un vértice terminal, x es un vértice **interno** (o una **rama**).
- El **subárbol** de T con raíz en x es el árbol con el conjunto de vértices V y el conjunto de aristas E, donde V es x junto con los descendientes de x y

  $E = \{e | e \text{ es una arista en una trayectoria simple de } x \text{ a algún vértice en } V\}$.

1.3.1. Revisitando los sistemas de archivos

Para ilustrar mejor los conceptos mencionados arriba, retomemos el ejemplo visto en la sección 1.1. Allí se presentó la estructura jerárquica de directorios y archivos como un árbol. En la Figura 1 se muestra el contenido del directorio `addressable-2.8.5`. Por una cuestión de legibilidad, en la Figura 6 se transcribe únicamente el contenido del subárbol con raíz en `lib` como un árbol con la raíz arriba.

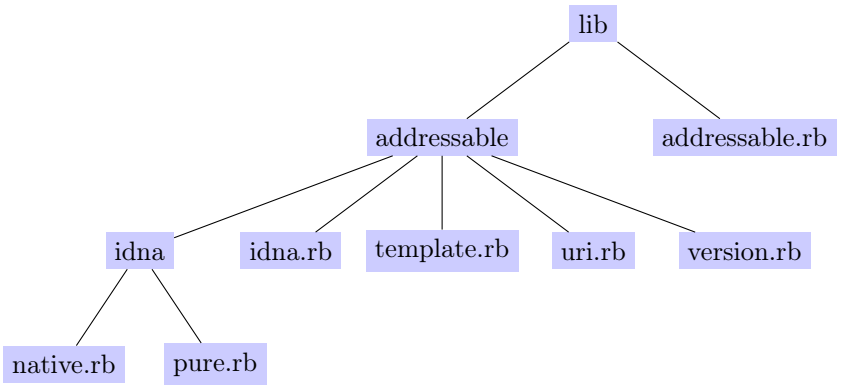

Figura 6: Despliegue del contenido del subdirectorio `lib` como un árbol con la raíz arriba.

Llamemos A al árbol con raíz que se muestra en dicha figura. Se pueden hacer las siguientes observaciones:

- La carpeta `lib` es el nodo padre de la carpeta `addressable`, así como también del archivo `addressable.rb`.
- Las carpetas `lib` y `addressable` son los ancestros de la carpeta `idna`. También son los ancestros del archivo *uri.rb* y de todos sus hermanos.
- Los archivos `native.rb` y `pure.rb` son los nodos hijos de la carpeta `idna`.
- El archivo `addressable.rb`, así como la carpeta `addressable` y todos sus hijos, son descendientes del

nodo `lib`.

- Los archivos `native.rb` y `pure.rb` son nodos hermanos.
- `template.rb` es un vértice terminal (o una hoja).
- La carpeta `idna` es un vértice interno (o una rama).
- La carpeta `addressable` junto con todo su contenido es un subárbol de A.

1.4. Árboles Binarios (Binary trees)

Los **árboles binarios** están entre los tipos especiales más importantes de árboles con raíz. Todo vértice en un árbol binario tiene a lo sumo dos hijos. Más aún, cada hijo se designa como un hijo izquierdo o un hijo derecho. Cuando se dibuja un árbol binario, un hijo izquierdo se dibuja a la izquierda y un hijo derecho se dibuja a la derecha. La definición formal es la siguiente.

Un **árbol binario** es un árbol con raíz en el que cada vértice tiene ningún hijo, un hijo o dos hijos. Si el vértice tiene un hijo se designa como un hijo izquierdo o como un hijo derecho (pero no ambos). Si un vértice tiene dos hijos, un hijo se designa como hijo izquierdo y el otro como hijo derecho.

Un **árbol binario completo** es un árbol binario en el que cada vértice tiene dos o cero hijos.

1.4.1. Ejemplo de árbol binario

Como se mencionó anteriormente, los sistemas de archivos utilizan estructuras jerárquicas de tipo árbol para organizar los archivos. En particular, si cada carpeta contiene a lo sumo dos hijos (es decir, cada carpeta tiene dentro a lo sumo *a)* dos carpetas, *b)* un archivo y una carpeta, o *c)* dos archivos), entonces la jerarquía de un directorio se puede pensar como un árbol binario.

En la Figura 7 se muestra el despliegue del contenido del directorio `TrabajoFinal`. Dicho directorio se puede representar con un árbol binario, ya que todos los nodos tienen a lo sumo dos hijos. Para esto se debe definir un criterio para determinar cuál es el hijo izquierdo y cuál es el hijo derecho. En este caso, se considerará que el hijo que aparece *más arriba* en la figura, será el hijo de la izquierda. Por ejemplo, *Código* será el hijo izquierdo de `TrabajoFinal`, mientras que `Informe` será el hijo derecho. A su vez, cuando se tenga un único hijo, se considerará el hijo izquierdo. En la misma figura, se puede observar que si bien es un árbol binario, no es un árbol binario completo, ya que el nodo `main` tiene un único hijo. Es decir, dentro de la carpeta `main` hay un único archivo (`main.c`).

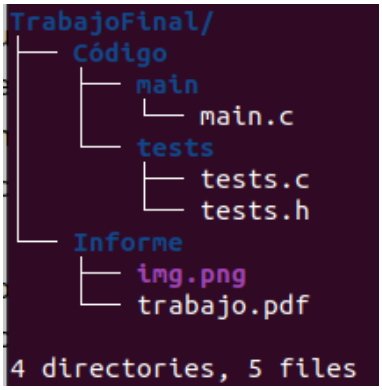

Figura 7: Despliegue del contenido del directorio TrabajoFinal.

En la Figura 8 se muestra el contenido del directorio `TrabajoFinal` como un árbol binario con la raíz arriba.

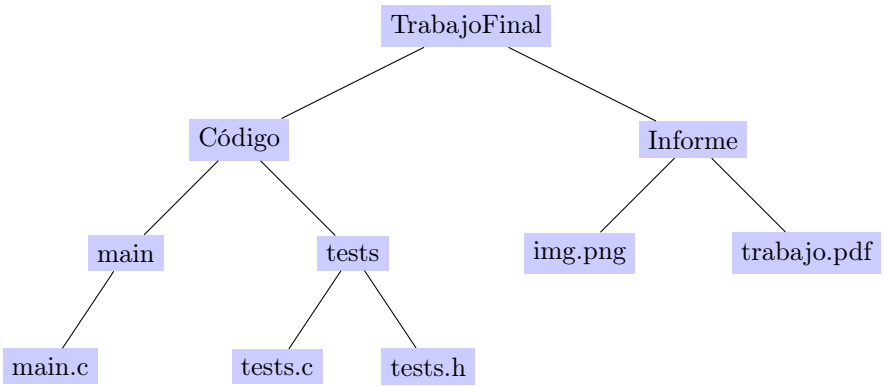

Figura 8: Despliegue del contenido del directorio TrabajoFinal como un árbol con la raíz arriba.

1.4.2. Implementación de árboles binarios

Representaremos a los árboles como estructuras enlazadas (i.e. compuestas por nodos). Un tipo común de árbol que ya presentamos es un árbol binario, en el que cada nodo contiene una referencia como máximo a otros dos nodos. Estas referencias se denominan subárboles izquierdo y derecho. Los nodos de los árboles contienen datos. Un diagrama de estado para un árbol se representa en la Figura 9.

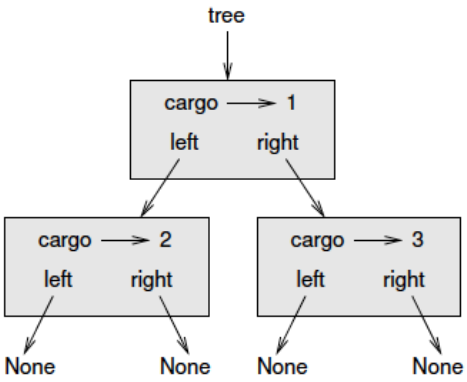

Figura 9: Representación de un árbol binario.

Los árboles son estructuras de datos recursivas porque se definen recursivamente.

Un árbol binario es:

- el árbol vacío, representado por `None`, o
- un nodo que contiene una referencia de objeto y dos referencias a árboles binarios.

Una implementación en `Python` de Árbol Binario (BinaryTree) se ve así:

```
class BinaryTree:
    def __init__(self, cargo, left=None, right=None):
        self.cargo = cargo
        self.left = left
        self.right = right

    def __str__(self):
        return str(self.cargo)
```

`cargo` puede ser de cualquier tipo, pero los argumentos para `left` y `right` deben ser de nodos de árboles. `left` y `right` son opcionales; el valor predeterminado es `None`. Para imprimir un nodo, simplemente imprimimos el dato en `cargo`.

Una forma de construir un árbol es de abajo hacia arriba. Asigne los nodos secundarios primero:

```
left = BinaryTree(2)
right = BinaryTree(3)
```

Luego cree el nodo padre y vincúlelo a los hijos:

```
tree = BinaryTree(1, left, right)
```

Podemos escribir este código de manera más concisa anidando invocaciones de constructores:

```
>>> tree = BinaryTree(1, BinaryTree(2), BinaryTree(3))
```

De cualquier manera, el resultado es el árbol de la Figura 9.

1.5. Árboles Binarios de Búsqueda

Un **árbol binario de búsqueda** (o en inglés Binary Search Tree) es un árbol binario T en el que se asocian datos a los vértices. Los datos están arreglados de manera que, para cada vértice v en T, cada dato en el subárbol de la izquierda de v es menor que el dato en v, y cada dato en el subárbol de la derecha de v es mayor que el dato en v.

Las palabras

OTRA PERSONA NO DIRÍA TODO JUNTO
LO TENDRÍA MEMORIZADO

se pueden colocar en un árbol de búsqueda binaria como se ve en la Figura 10. Observe que para cualquier vértice v, cada dato en el subárbol izquierdo de v es menor que (es decir, precede alfabéticamente) el dato en v y cada dato en el subárbol derecho de v es mayor que el dato en v.

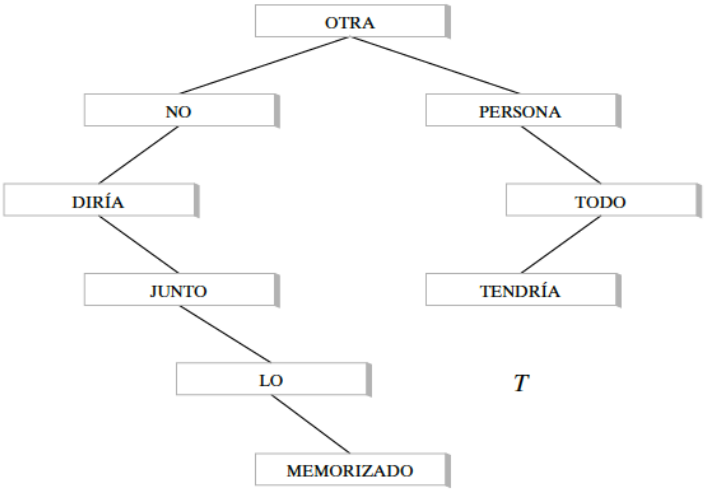

Figura 10: Un árbol de búsqueda binaria.

En general, habrá muchas maneras de colocar datos en un árbol de búsqueda binaria. La figura 11 muestra otro árbol de búsqueda binario que almacena dichas palabras.

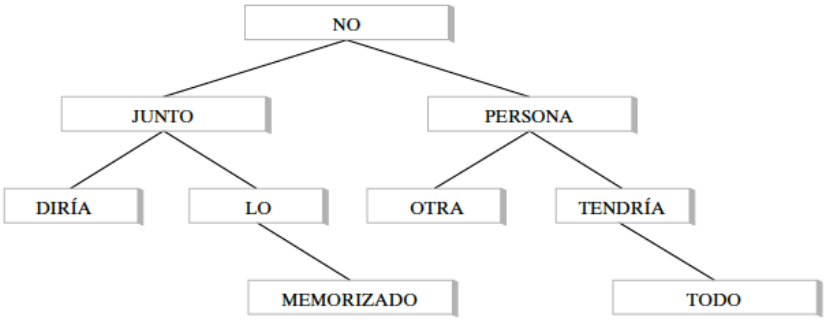

Figura 11: Otro árbol de búsqueda binaria que almacena las mismas palabras que el árbol en la Figura 10.

1.5.1. Construcción de un Árbol Binario de Búsqueda

La siguiente implementación en `Python` permite ingresar un nuevo dato `new_data` en un árbol binario de búsqueda `BST_tree`:

```
def insertBST(new_data, BST_tree):
  if BST_tree == None:
    return BinaryTree(new_data,None,None)
  else:
    if new_data < BST_tree.cargo:
      return BinaryTree(BST_tree.cargo, \
      insertBST(new_data,BST_tree.left), BST_tree.right)
    elif new_data > BST_tree.cargo:
      return BinaryTree(BST_tree.cargo, BST_tree.left, \
      insertBST(new_data,BST_tree.right))
    else:
      return BST_tree
```

Podemos crear un árbol binario de búsqueda a partir de datos en una lista `data_list`, usando la siguiente implementación en `Python`:

```
def createBST(data_list):
  BST_tree = None

  while(data_list!=[]):
    cargo = data_list.pop()
    BST_tree = insertBST(cargo,BST_tree)

  return BST_tree
```

1.5.2. Búsqueda de Datos en un Árbol Binario de Búsqueda

Los árboles binarios de búsqueda son útiles para localizar datos. Esto es, a partir de un dato D, es fácil determinar si D está presente en un árbol binario de búsqueda y dónde se localiza. Veamos la siguiente implementación en `Python`:

```
def searchBST(D, BST_tree):
  if BST_tree == None:
    return False
  else:
    if D < BST_tree.cargo:
      return searchBST(D, BST_tree.left)
```

```
    elif D > BST_tree.cargo:
      return searchBST(D,BST_tree.right)
    else:
      return True
```

Para determinar si un dato D está en el árbol de búsqueda binaria, comenzaríamos en la raíz. Después compararíamos D repetidas veces con el dato en el vértice actual. Si D es igual al dato en el vértice actual v, encontramos D y nos detenemos. Si D es menor que el dato en el vértice actual v, nos movemos al hijo izquierdo de v y repetimos el proceso. Si D es mayor que el dato en el vértice actual v, nos movemos al hijo derecho de v y repetimos el proceso. Se concluye que D no está en el árbol cuando el árbol en donde se debe buscar es vacío.

1.6. Recorridos de árboles

El recorrido de árboles se refiere al proceso de visitar de una manera sistemática todos los nodos del árbol, exactamente una vez. Tales recorridos están clasificados por el orden en el cual son visitados los nodos. Los siguientes algoritmos son descritos para un árbol binario, pero también pueden ser generalizados a otros árboles.

1.6.1. PreOrden (PreOrder)

Para recorrer un árbol binario no vacíio en *PreOrden*, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:

- Visite la raíz
- Recorra en PreOrden el sub-árbol izquierdo
- Recorra en PreOrden el sub-árbol derecho

En la Figura 12 se muestra el recorrido PreOrden para el árbol correspondiente al contenido del directorio `TrabajoFinal` (Figura 8).

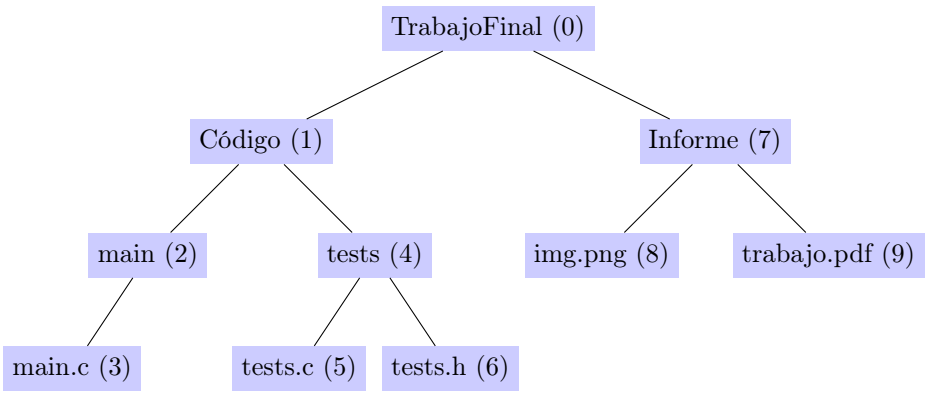

Figura 12: Recorrido PreOrden para un árbol binario. El número en cada nodo indica el orden de visita.

1.6.2. InOrden (InOrder)

Para recorrer un árbol binario no vacío en *inorden* (simétrico), hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Recorra en InOrden el sub-árbol izquierdo
- Visite la raíz

- Recorra en InOrden el sub-árbol derecho

En la Figura 13 se muestra el recorrido InOrden para el árbol correspondiente al contenido del directorio `TrabajoFinal` (Figura 8).

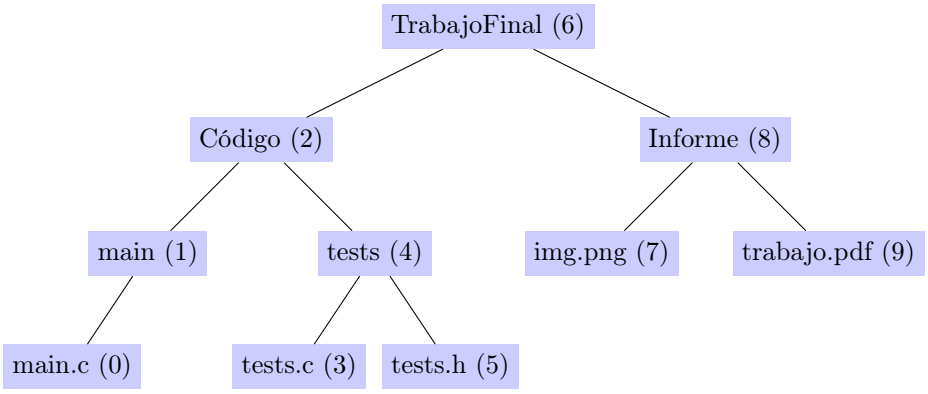

Figura 13: Recorrido InOrden para un árbol binario. El número en cada nodo indica el orden de visita.

1.6.3. PostOrden (PostOrder)

Para recorrer un árbol binario no vacío en *postorden*, hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Recorra en PostOrden el sub-árbol izquierdo
- Recorra en PostOrden el sub-árbol derecho
- Visite la raíz

En la Figura 14 se muestra el recorrido PostOrden para el árbol correspondiente al contenido del directorio `TrabajoFinal` (Figura 8).

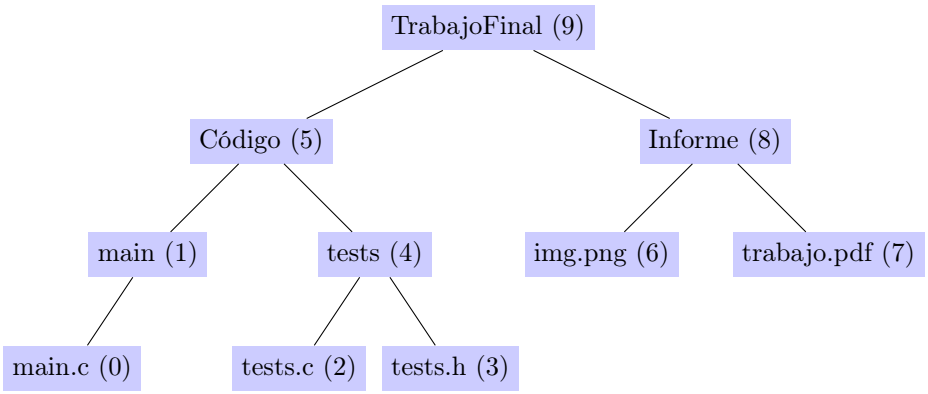

Figura 14: Recorrido PostOrden para un árbol binario. El número en cada nodo indica el orden de visita.

En general, la diferencia entre *PreOrden*, *InOrden* y *PostOrden* es cuándo se recorre la raíz. En los tres, se recorre primero el sub-árbol izquierdo y luego el derecho.

1.6.4. Implementando Recorridos de Árboles

Imprimimos los nodos usando la siguiente implementación en `Python` del recorrido PreOrden

```
def printTreePreOrder(tree):
    if tree == None:
        return
    print(tree.cargo)
    printTreePreOrder(tree.left)
    printTreePreOrder(tree.right)
```

Imprimimos los nodos usando la siguiente implementación en `Python` del recorrido InOrden

```
def printTreeInOrder(tree):
    if tree == None:
        return
    printTreeInOrder(tree.left)
    print(tree.cargo)
    printTreeInOrder(tree.right)
```

Imprimimos los nodos usando la siguiente implementación en `Python` del recorrido PostOrden

```
def printTreePostorder(tree):
    if tree == None:
        return
    printTreePostorder(tree.left)
    printTreePostorder(tree.right)
    print(tree.cargo)
```

2. Bibliografia

Referencias

[1] A. Downey, J. Elkner y C. Meyers, *How to Think Like a Computer Scientist: Learning with Python*, Green Tea Press, Wellesley, Massachusetts, 2008.

[2] R. Johnsonbaugh, *Matemáticas Discretas*, 6ta Edición, Pearson Educación, México, 2005.