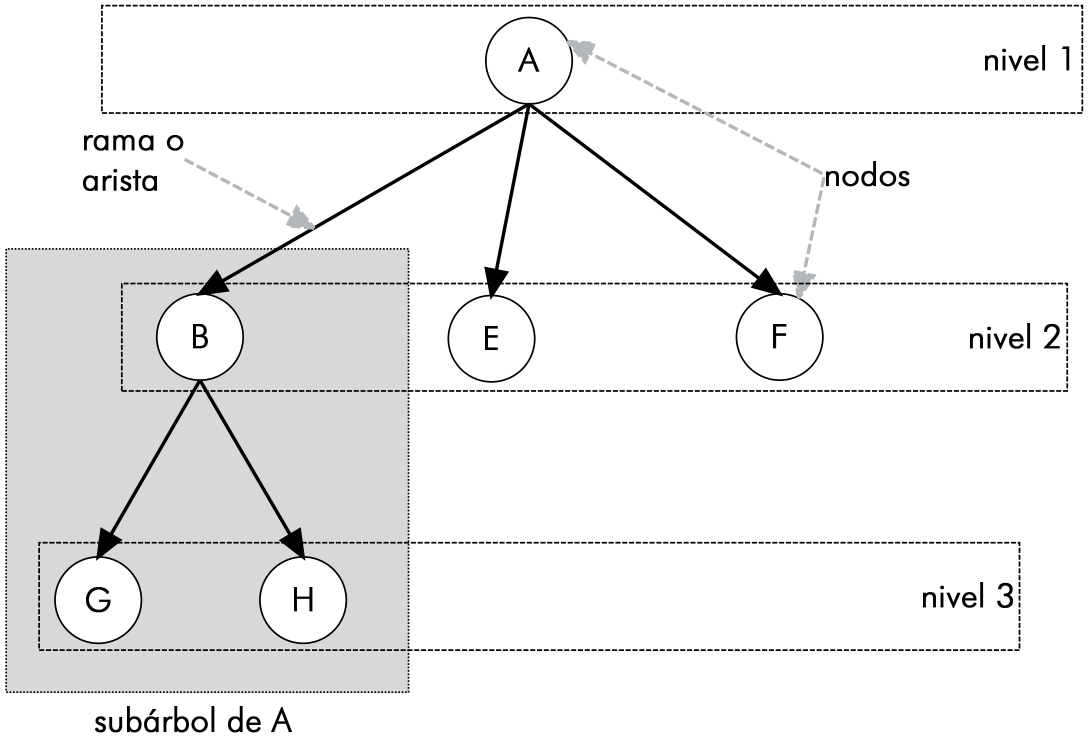
Programación II

TAD Árbol Binario

Terminología de Árboles



A raíz
A padre de B,E,F
A ascendiente de B, E, F, G, H
B, E, F hijos de A
G, H descendientes de A, B
B, E, F hermanos
E, F, G, H nodos terminales u hoja
B nodos interiores
A-B-G camino de longitud = 2 (nº de aristas)
Nivel de B = 2
Altura del árbol = 3 (nº de niveles del árbol)
Grado o aridad de B = 2
Grado o aridad del árbol = 3 (máxima en el árbol)

Figura 1: Representación gráfica de un árbol y la terminología empleada.

Árbol lleno.

Un árbol de altura h se dice que es lleno si todas sus hojas están al mismo nivel h y todos los nodos que están en niveles anteriores tienen el número máximo de hijos posibles[1].

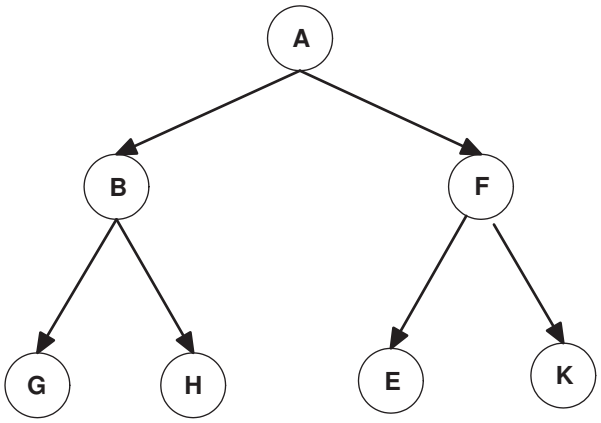


Figura 2: Ejemplo de árbol binario lleno

Árbol completo.

Un árbol de altura h se dice que es completo si está lleno hasta el nivel h-1 y si todos los nodos del nivel h están situados lo más a la izquierda posible.

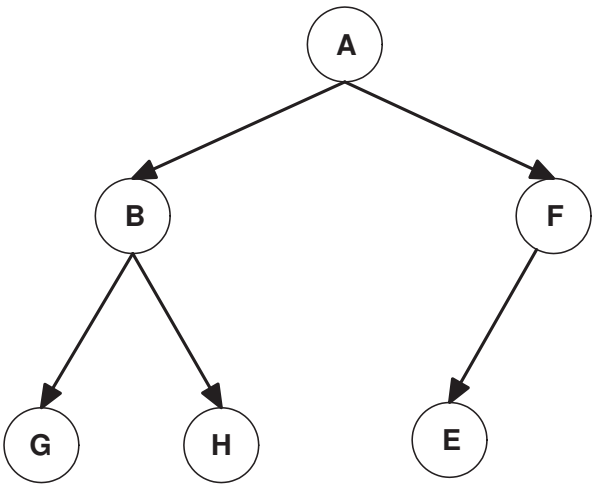
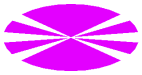


Figura 3: Ejemplo de árbol binario completo

[1] Nótese que ésta es una definición genérica. En caso de árboles binarios, lógicamente este número sería de 2.



Programación II

Especificación de Árboles Binarios

Especificación informal TAD Arbol Binario

TAD ArbolBinario
VALORES

- Un árbol binario es un conjunto de n elementos del mismo tipo llamados nodos donde n≥0:
  - O bien n=0, en cuyo caso se dice que el árbol es vacío
  - O bien existe un elemento distinguido llamado *raíz*, y el resto de los nodos se distribuyen en dos subconjuntos disjuntos A1 y A2 cada uno de los cuales es un *árbol binario*, llamados subárbol izquierdo y derecho, respectivamente.

OPERACIONES (SINTAXIS y SEMÁNTICA)

- Generadoras
  - `CreateEmptyTree` → `Tree`
    {*Objetivo*: Crea un árbol vacío
    Salida: Un árbol vacio
    Poscondición: El árbol sin datos}
  - `BuildTree (Tree, Item, Tree)` → `Tree, Boolean`
    {*Objetivo*: Crea un árbol con cierta información en la raíz y como hijos los árboles que se reciben en las entradas
    Entrada:

    Tree(1): Árbol que constituirá el hijo izquierdo
    Item: Contenido del elemento raíz
    Tree(2): Árbol que constituirá el hijo derecho

    Salida:

    Tree: Nuevo árbol construido y
    verdadero si se ha podido construir, falso en caso contrario}

- Observadoras
  - `LeftChild (Tree)` → `Tree`
    {*Objetivo*: Devuelve el árbol que constituye el hijo izquierdo del árbol
    Entrada:

    Tree: Árbol a manipular

    Salida:

    Tree: Árbol que constituye el hijo izquierdo o nulo si no existe

    Precondición:

    El árbol no está vacío}
  - `RightChild (Tree)` → `Tree`
    {*Objetivo*: Devuelve el árbol que constituye el hijo derecho del árbol
    Entrada:

    Tree: Árbol a manipular

    Salida:

    Tree: Árbol que constituye el hijo derecho o nulo si no existe

    Precondición:

    El árbol no está vacío}
  - `Root (Tree)` → `Item`
    {*Objetivo*: Devuelve el dato de la raíz del árbol
    Entrada:

    Tree: Árbol a manipular

    Salida:

    Item: Contenido del elemento de la raíz

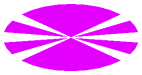
    Precondición:

    El árbol no está vacío}
  - `IsEmptyTree (Tree)` → `Boolean`
    {*Objetivo*: Determina si un árbol está vacío
    Entrada:

    Tree: Árbol a manipular

    Salida:

    Verdadero si el árbol está vacío, falso en caso contrario}



UNIVERSIDADE DA CORUÑA

Programación II

Recorridos de Árboles

Recorridos de Árboles

Objetivo: proporcionar un modo sistemático de visitar todos los nodos de un árbol

Recorridos en profundidad

Tipos de recorrido:

1. Preorden
   - Visitar la raíz
   - Visitar en preorden el subárbol A_1[2]
   - Visitar en preorden los subárboles $A_2, ..., A_n$[3]
2. Inorden
   - Visitar en inorden el subárbol A_1
   - Visitar la raíz
   - Visitar en inorden los subárboles $A_2, ..., A_n$
3. Posorden
   - Visitar en posorden el subárbol A_1
   - Visitar en posorden los subárboles $A_2, ..., A_n$
   - Visitar la raíz

[2]Si el árbol es binario, subárbol izquierdo.
[3]Si el árbol es binario, subárbol derecho.

Recorrido en anchura

- Se explora el árbol por niveles, comenzando en el nivel 1 de la raíz, luego el 2, etc.
- El recorrido no se realizará de forma recursiva sino iterativa, utilizando una cola como estructura de datos auxiliar.
- El procedimiento consiste en insertar en la cola (si no están vacíos) los subárboles izquierdo y derecho del nodo extraído de la cola, y seguir borrando e insertando hasta que la cola esté vacía.

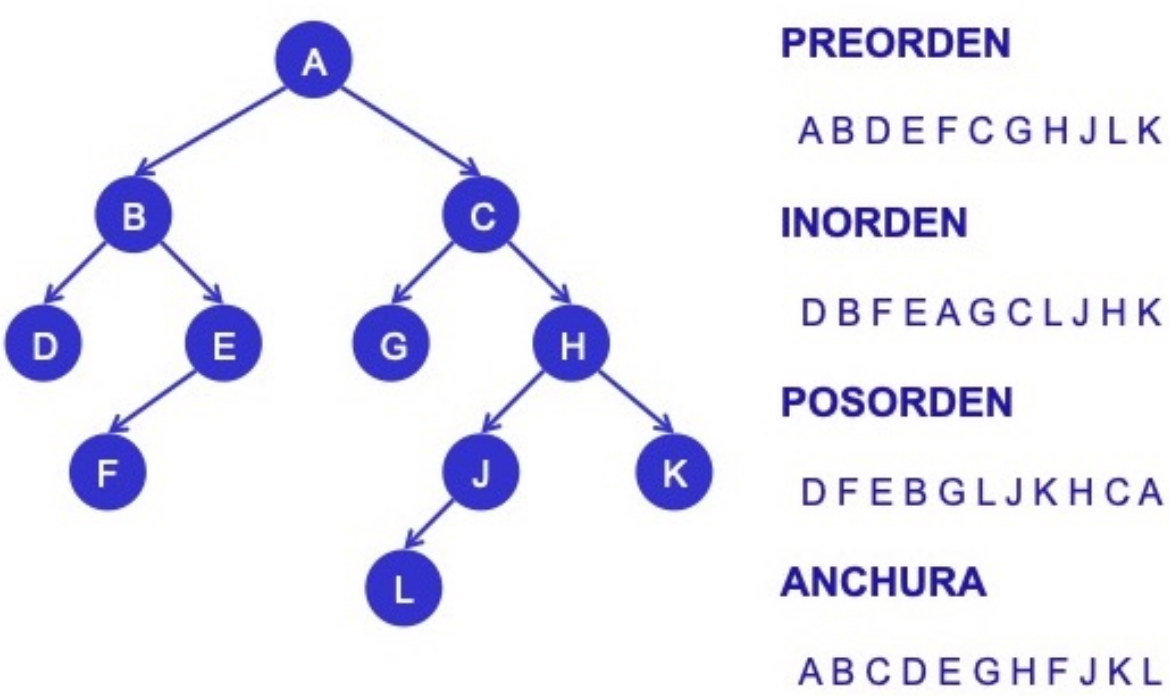


Figura 4: Ejemplos de recorridos sobre árboles binarios