Instituto Politécnico Nacional

Escuela Superior de Cómputo

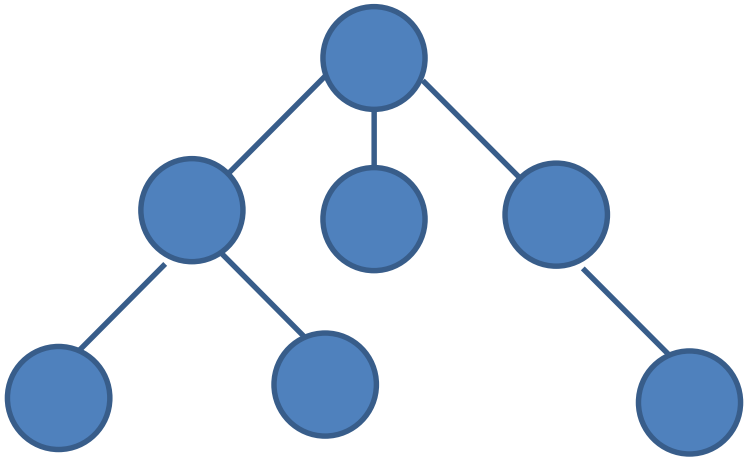
Estructuras de Datos

Tema 08: TAD Árbol

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- Descripción del TAD Árbol
- Especificación del TAD Árbol
- Implementación del TAD Árbol
- Aplicaciones del TAD Árbol



Descripción del TAD Árbol

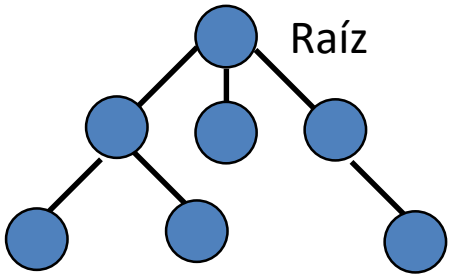
- Los árboles representan las **estructuras de datos no lineales y dinámicas más importantes** en computación.

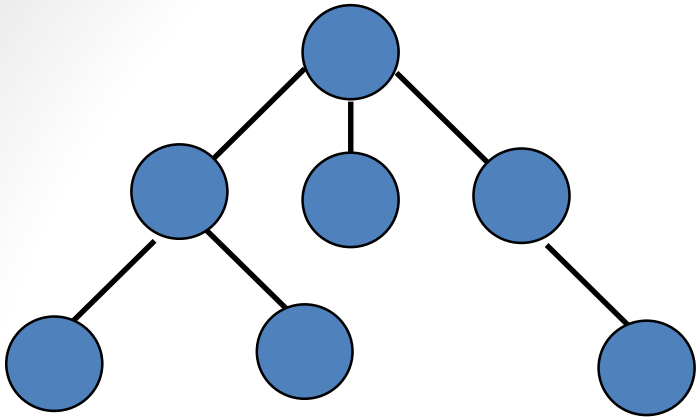
- Un árbol es una **estructura jerárquica** aplicada sobre una colección de elementos u objetos llamados **nodos**; uno de los cuales es conocido como **raíz**. Además se crea una relación o **parentesco entre los nodos** dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etc.

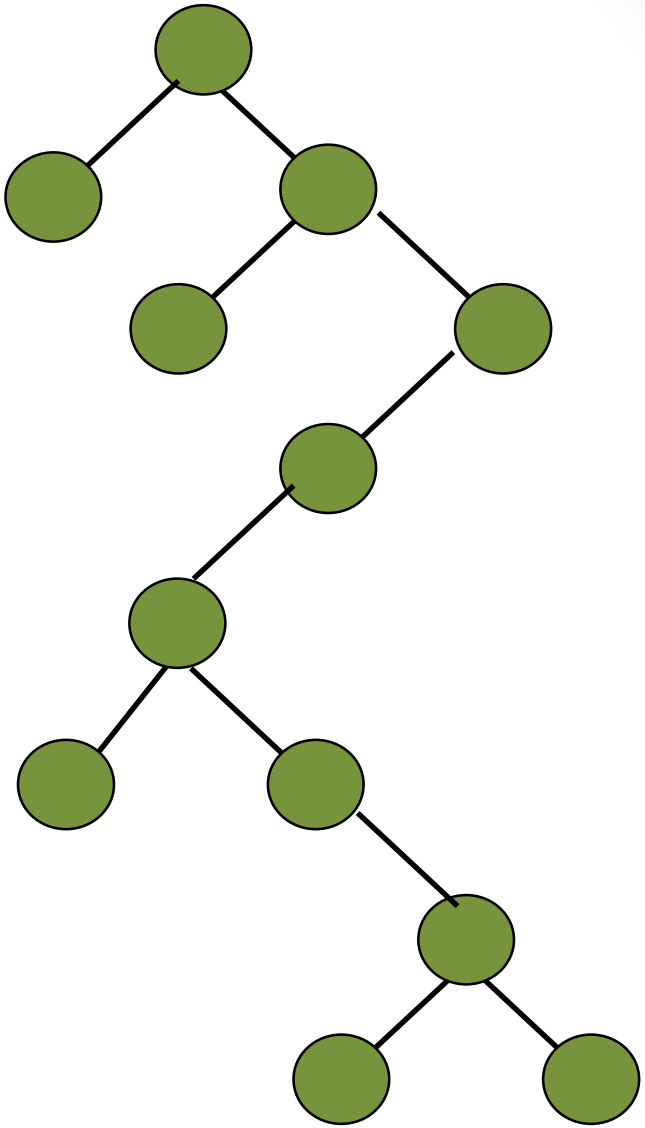
- Un árbol es una colección de elementos entre los cuales existe una **estructura jerárquica** definida mediante una **relación de paternidad** entre los elementos.

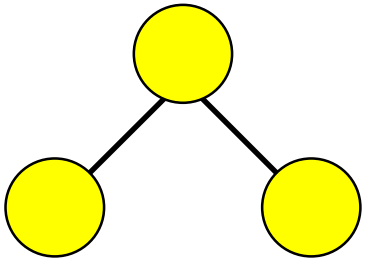
- Entre los elementos, que llamaremos nodos, se distingue uno que es el **nodo raíz**.

- Matemáticamente, un árbol es un **grafo no orientado, conexo y acíclico** en el que existe un vértice destacado llamado raíz.







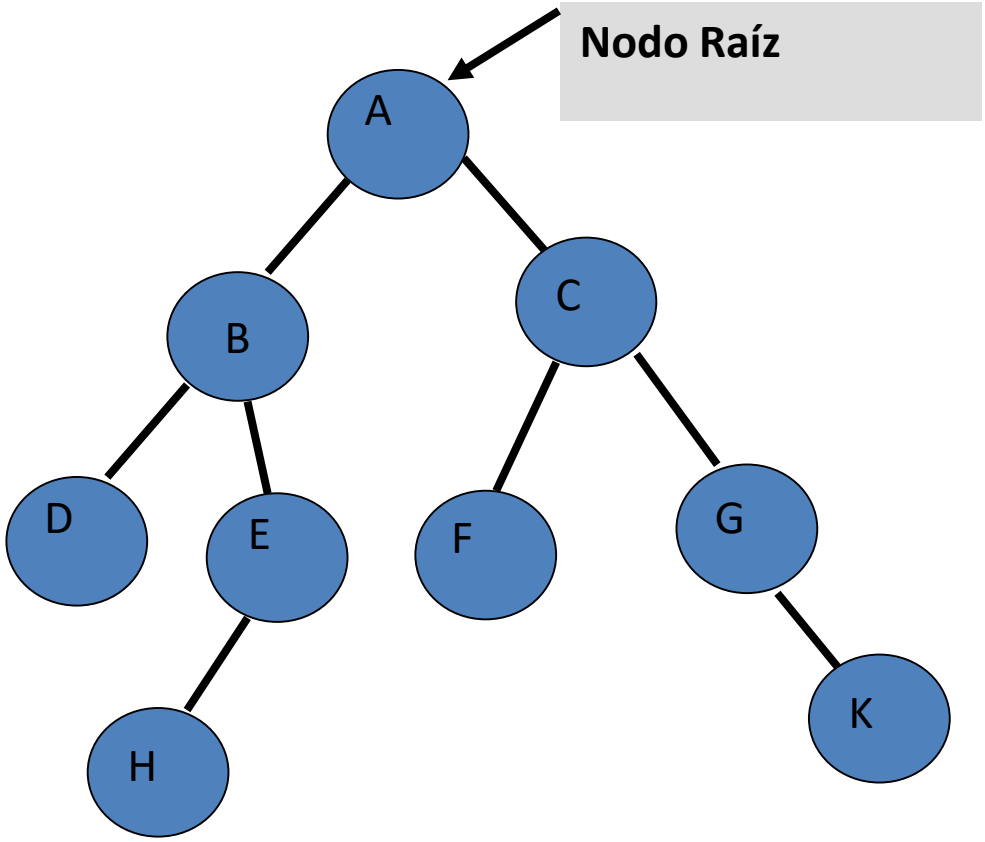


Árbol *A n*-ario

- Un **árbol A n-ario** (en adelante sólo árbol), siendo **n≥1**, es un conjunto de elementos del mismo tipo tal que:
  - O bien es el **conjunto vacío**, en cuyo caso se le denomina **árbol vacío**.
  - O bien no es vacío, en cuyo caso **existe un elemento distinguido llamado raíz**, y el resto de los elementos se distribuyen en **m subconjuntos disjuntos A1 ,..., Am**, **con 0≤m≤n**, cada uno de los cuales es un **árbol n-ario**, llamados **subárboles** del árbol original.

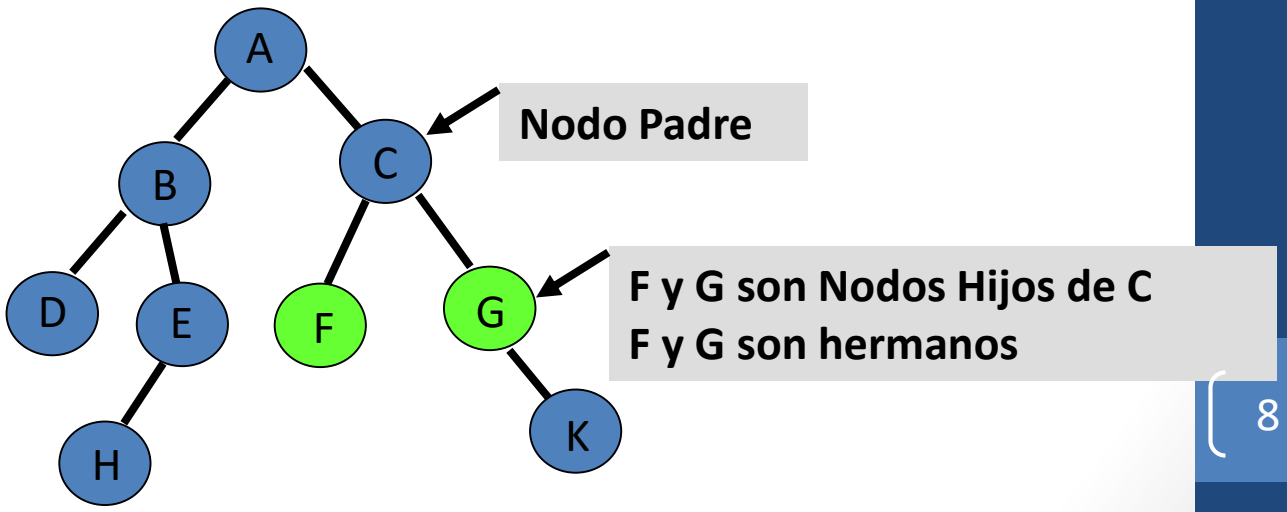
Conceptos y definiciones para los árboles

- **Nodo:** Cada elemento en un árbol.
- **Nodo Raíz:** Primer elemento agregado al árbol.

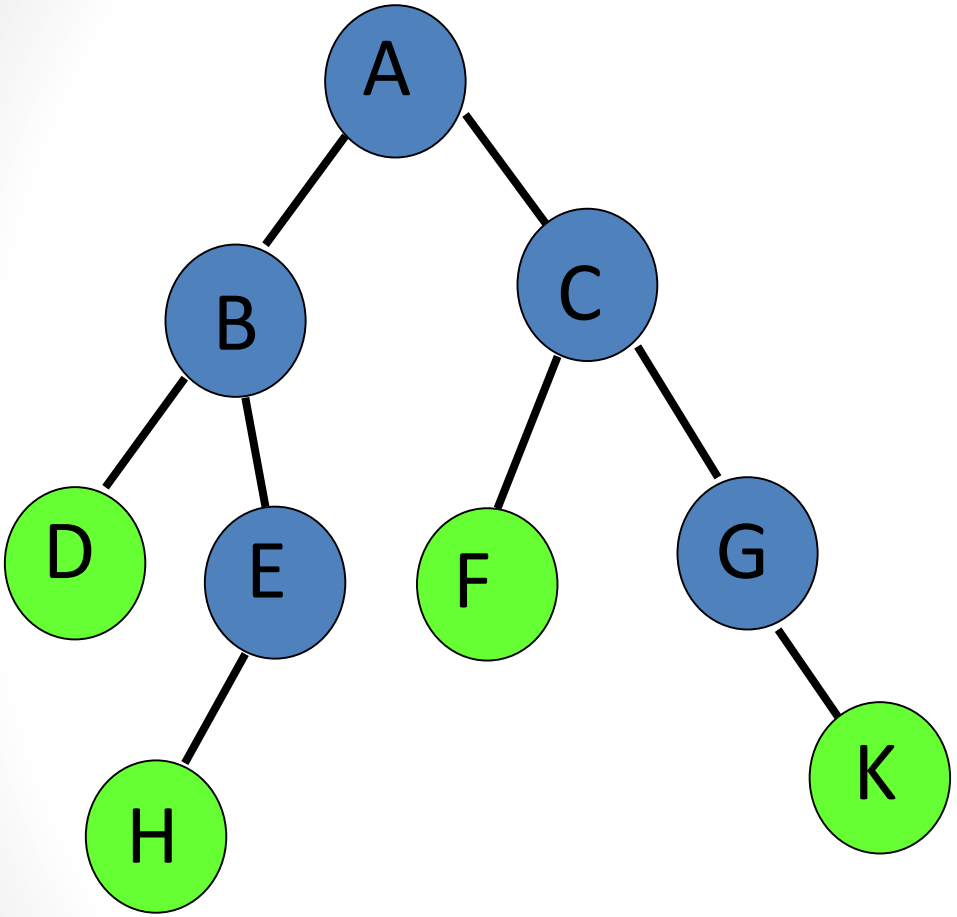


Conceptos y definiciones para los árboles

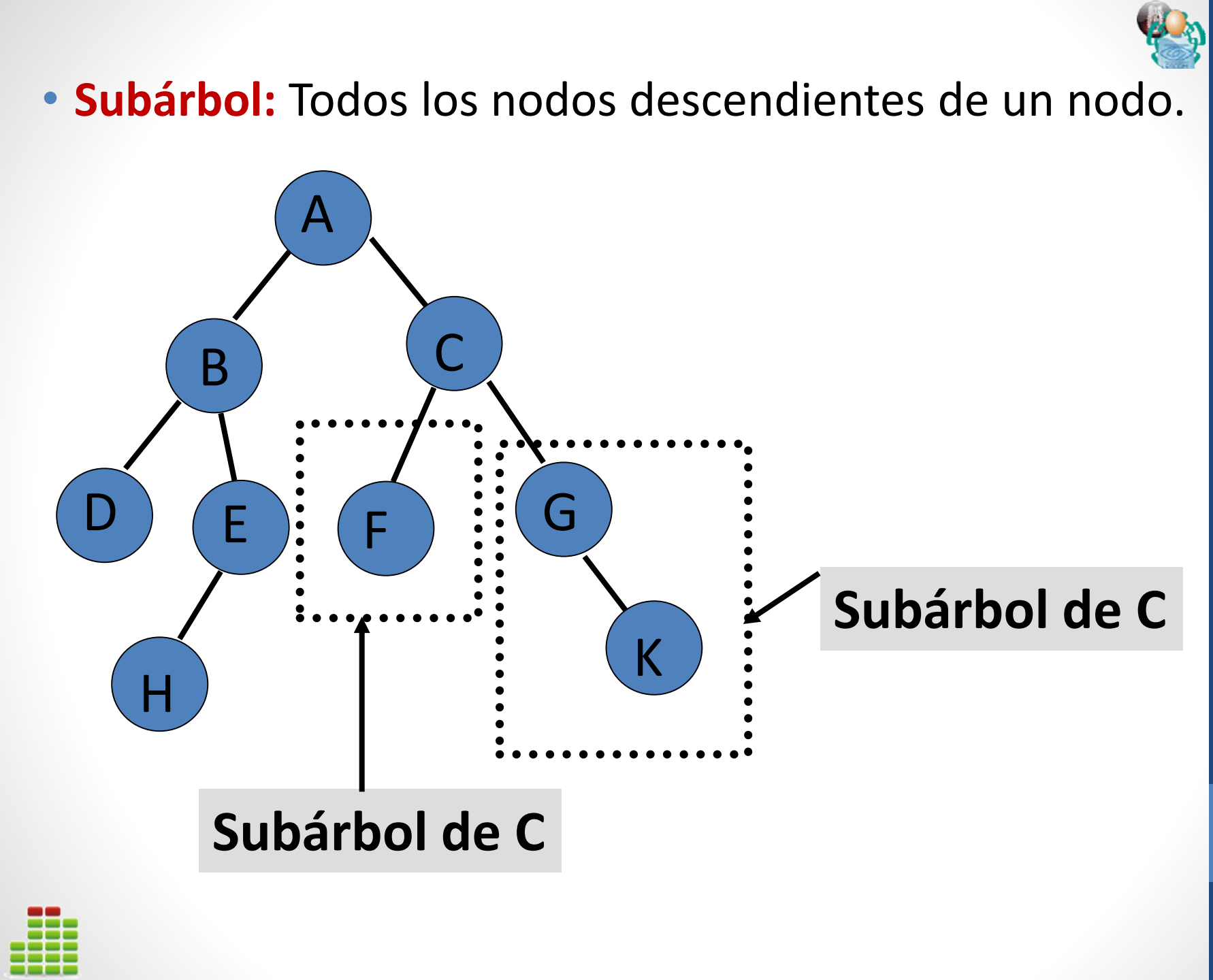
- **Nodo Padre:** Se le llama así al nodo predecesor de un elemento.
- **Nodo Hijo:** Es el nodo sucesor de un elemento.
- **Hermanos:** Nodos que tienen el mismo nodo padre.



- **Nodo Hoja:** Aquel nodo que no tiene hijos.



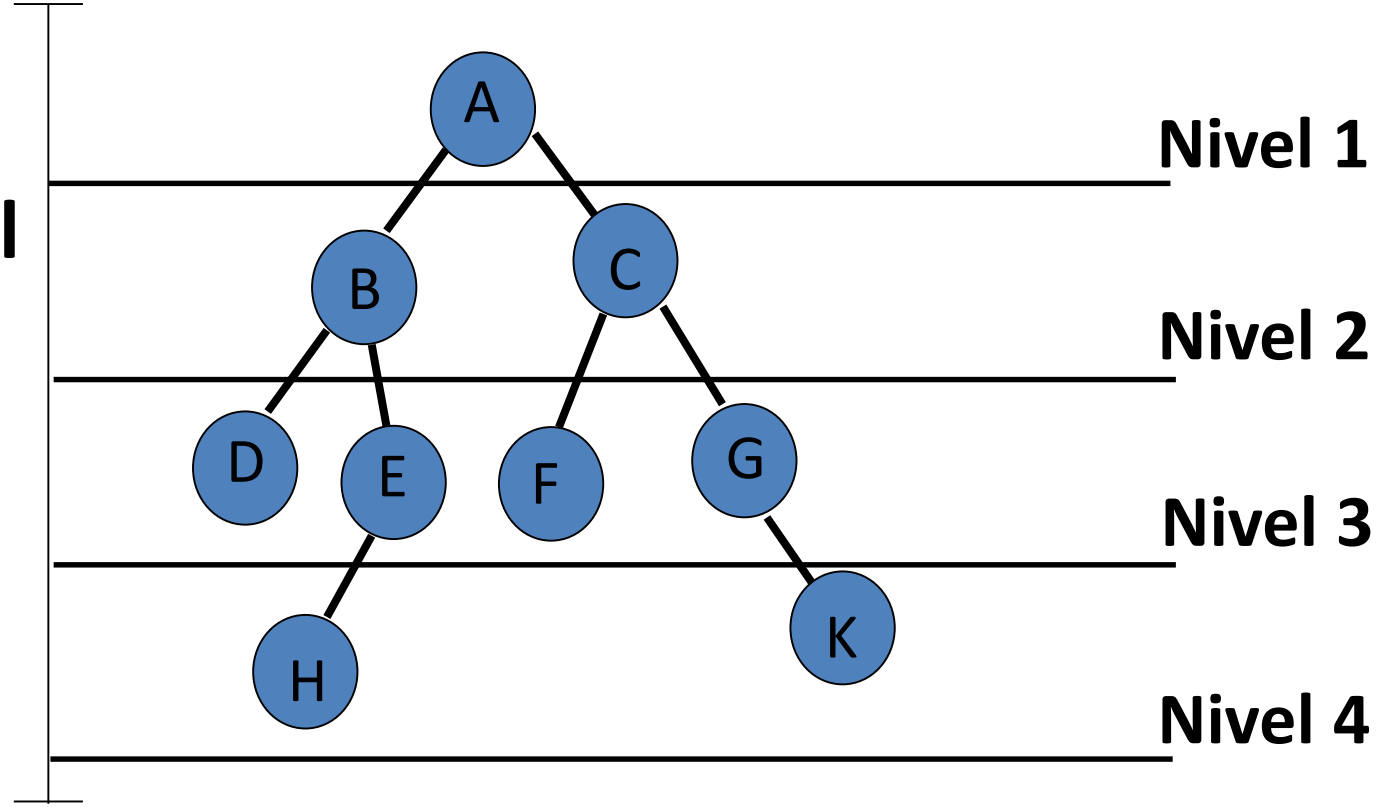
D, H, F y K son Nodos Hojas



- **Subárbol:** Todos los nodos descendientes de un nodo.

Altura y Niveles

Altura del árbol = 4



La Altura es la cantidad de niveles.

Especificación del TAD Árbol

Cabecera

- ***Nombre:*** Árbol (Tree)
- ***Lista de operaciones:***
  - ***Operaciones de construcción***
    - ***Inicializar (Initialize):*** *Recibe un árbol A y lo inicializa para su trabajo normal.*
    - ***Eliminar (Destroy):*** *Recibe una un árbol A y lo libera completamente.*
  - ***Operaciones de posicionamiento y búsqueda***
    - ***Raíz (Root):*** *Recibe un árbol A y devuelve la posición de la raíz.*
    - ***Padre (Parent):*** *Recibe un árbol A y una posición p, devuelve la posición de padre de p.*
    - ***Hermano derecho (Right Brother):*** *Recibe un árbol A y una posición p, devuelve la posición del hermano derecho de p.*

- ***Hijo izquierdo (Son Left):*** *Recibe un árbol A y una posición p, devuelve la posición del hijo más a la izquierda de p.*
- ***Buscar (Search):*** *Recibe un árbol A y un elemento e, devuelve la posición del elemento en el árbol A.*

- ## *Operaciones de consulta*
  - ***Vacía (Empty):*** *Recibe un árbol A y devuelve verdadero en caso de que el árbol A este vacío.*
  - ***Nodo Nulo (Null Node):*** *Recibe un árbol A y una posición p, devuelve verdadero si la posición p del árbol A es nula o incorrecta.*
  - ***Leer nodo (Read Node):*** *Recibe un árbol A y una posición p, devuelve el elemento contenido en el nodo con posición p del árbol A.*

- ## *Operaciones de modificación*
  - ***Nuevo Hijo(New Son):*** *Recibe un árbol A, una posición p y un elemento e, se añade a e como hijo más a la izquierda del nodo con posición p.*

- ***Nuevo Hermano(New Brother):*** *Recibe un árbol A, una posición p y un elemento e, se añade a e como hermano derecho del nodo con posición p*

- ***Eliminar Hijo(Delete Son):*** *Recibe un árbol A y una posición p, se elimina al hijo más a la izquierda y todos sus descendientes del nodo con posición p.*

- ***Eliminar Hermano(Delete Brother):*** *Recibe un árbol A y una posición p, se elimina al hermano derecho y todos sus descendientes del nodo con posición p.*

- ***Eliminar Nodo(Delete Node):*** *Recibe un árbol A y una posición p, se elimina al nodo con posición p y todos sus descendientes.*

- ***Remplazar Nodo (Replace Node):*** *Recibe un árbol A, una posición p y un elemento e, se remplaza a e del nodo con posición p en A*

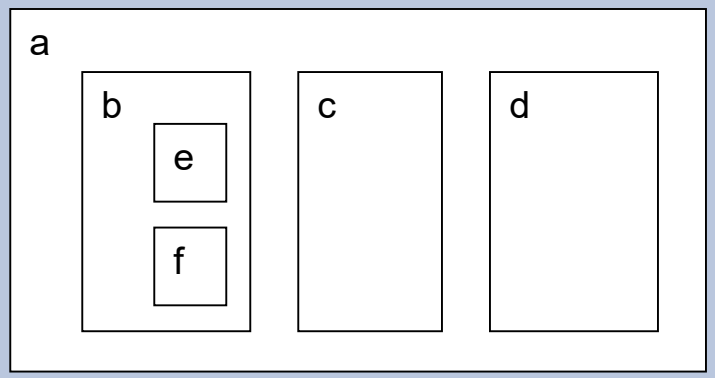
Descripción

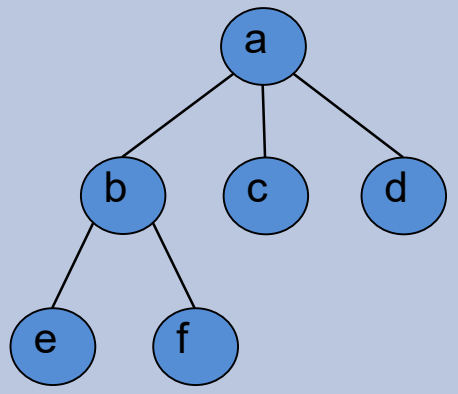
- Un árbol es una estructura:
  - ***Jerárquica*** porque los componentes están a distinto nivel.
  - ***Organizada*** porque importa la forma en que esté dispuesto el contenido.
  - ***Dinámica*** porque su forma, tamaño y contenido pueden variar durante la ejecución.

- Un árbol **puede ser**:
  - *Vacío,*
  - *Una raíz + subárboles.*

Descripción

- Representación de los arboles
  - Mediante diagramas de Venn
  - Mediante grafos
  - Mediante paréntesis anidados





(a (b (e,f), c, d))

Operaciones

- ***Inicializar (Initialize):*** ***recibe<-árbol(A);***
  - ***Initialize (A)***
  - Efecto: Recibe un árbol **A** y lo inicializa para su trabajo normal.

- ***Eliminar (Destroy):*** ***recibe<-árbol(A);***
  - ***Destroy (A)***
  - Efecto: Recibe un árbol **A** y lo libera completamente.

- ***Raíz (Root):*** ***recibe<-árbol(A); retorna -> posición***
  - ***Root (A)***
  - Efecto: Recibe un árbol **A** y retorna la posición de la raíz de **A**, si el árbol es vacío devuelve una posición nula.

- ***Padre (Parent):*** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***Parent(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P**, devuelve la posición de padre de **p**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si **P** es la raíz se devuelve una posición nula.

- ***Hermano derecho (Right Brother):*** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***RightBrother(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P**, devuelve la posición del hermano derecho de **p**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si **P** no tiene hermano derecho devuelve una posición nula.

- ***Hijo izquierdo (Son Left):*** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***SonLeft(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P**, devuelve la posición del hijo más a la izquierda de **p**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si **P** no tiene hijos devuelve una posición nula.

- ***Buscar (Search):*** ***recibe<-árbol(A), elemento (E); retorna -> posición***
  - ***Search(A,E)***
  - Efecto: Recibe un árbol **A** y un elemento **E**, devuelve la posición del elemento **E** en el árbol **A**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si **E** no es encontrado devuelve una posición nula.

- ***Vacia (Empty):*** ***recibe<-árbol(A); retorna -> booleano***
  - ***Empty(A)***
  - Efecto: Recibe un árbol **A** y devuelve **verdadero** en caso de que el árbol A este vacío, devuelve **falso** en caso contrario.

- ***Nodo Nulo (Null Node):*** ***recibe<-árbol(A), posición (P); retorna -> booleano***
  - ***NullNode(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P**, devuelve verdadero si la posición **P** del árbol **A** es nula o incorrecta y devuelve falso en caso contrario.

- ***Leer Nodo(Read Node):*** ***recibe<-árbol(A), posición (P); retorna -> elemento***
  - ***ReadNode(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P,** devuelve **el elemento** en la posición **P** del árbol **A**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

- ***Nuevo Hijo (New Son):*** ***recibe<-árbol(A), posición (P), elemento E;***
  - ***NewSon(A,P,E)***
  - Efecto: Recibe un árbol **A**, una posición **P** y un elemento **E**, se añade un nodo que contenga **E** como hijo más a la izquierda del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si el árbol **A** es vacío se agrega a **un nodo raíz con E.**

- ***Nuevo Hermano(New Brother)*** ***recibe<-árbol(A), posición (P), elemento E;***
  - ***NewBrother(A,P,E)***
  - Efecto: Recibe un árbol **A**, una posición **P** y un elemento **E**, se añade a un nodo que contenga a **E** como hijo más a la izquierda del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida y **P** no puede ser la posición de la raíz del árbol **A**.

- ***Eliminar Hijo (Delete Son):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteSon(A,P)***
  - Efecto: Recibe un árbol **A** y una posición se elimina al hijo más a la izquierda y todos sus descendientes del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

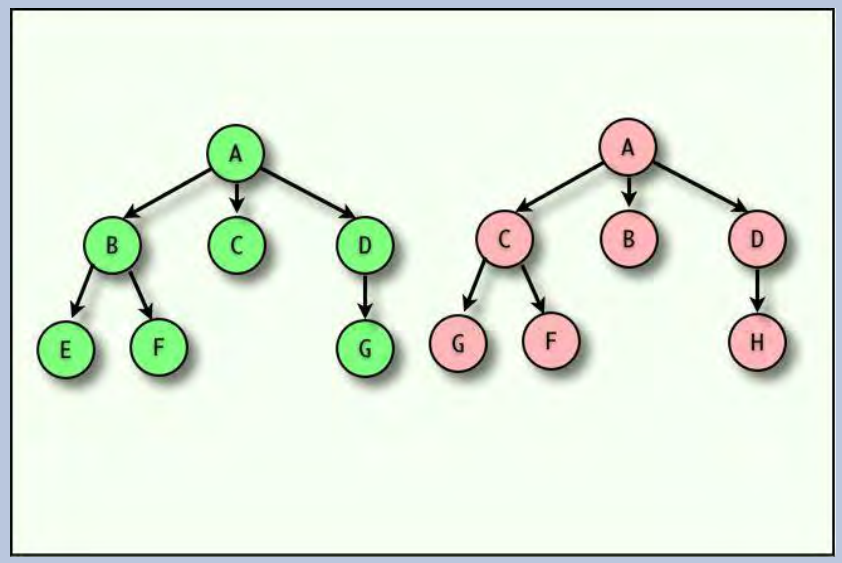
- ***Eliminar Hermano(Delete Brother):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteBrother(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P,** se elimina al hermano derecho y todos sus descendientes del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

- ***Eliminar Nodo(Delete Node):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteNode(A,P)***
  - Efecto: Recibe un árbol **A** y una posición **P,** se elimina al nodo con posición **P** y todos sus descendientes.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

- ***Remplazar Nodo(Replace Node):*** ***recibe<-árbol(A), posición (P), elemento (E);***
  - ***ReplaceNode(A,P)***
    - Efecto: Recibe un árbol **A,** una posición **P** y un elemento **E,** se remplaza a **E** del nodo con posición **P** en **A**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

Observaciones

- Cuando el orden de los subárboles importa, se dice que es un **árbol *ordenado***.



El árbol de la izquierda es ordenado y el árbol de la derecha es un árbol no ordenado.

- La raíz del árbol *A* es *padre* de la raíz de los subárboles A_1 ,..., A_m y las raíces de los subárboles A_1 ,..., A_m son *hijos* de la raíz de *A*.

- Las listas enlazadas son estructuras lineales
  - Son flexibles pero son secuenciales, un elemento detrás de otro.

- Los árboles
  - Junto con los grafos son estructuras de datos no lineales
  - Superan las desventajas de las listas
  - Sus elementos se pueden recorrer de distintas formas, no necesariamente uno detrás de otro

- Son **muy útiles** para la búsqueda y recuperación de información.

Conceptos

- ***Camino***: Si existe una secuencia de nodos $n_1,\ldots,n_k$ tal que n_i es padre de n_{i+1}, para $1 \leq i \leq k$, entonces la secuencia se denomina camino del nodo n_1 al nodo n_k. La *longitud* de un camino es el número de nodos del camino menos 1. Existe un camino de longitud 0 de todo nodo a sí mismo.

- ***Ascendiente/Descendiente***: Un nodo *a* es ascendiente de un nodo *b* (y *b* descendiente de *a*), si existe un camino del nodo *a* al nodo *b*. Por tanto todo nodo es ascendiente (y descendiente) de sí mismo. Los ascendientes (y descendientes) de un nodo, excluido el propio nodo, se denominan ascendientes (y descendientes) *propios*.

- ***Hoja***: Nodo sin descendientes propios.

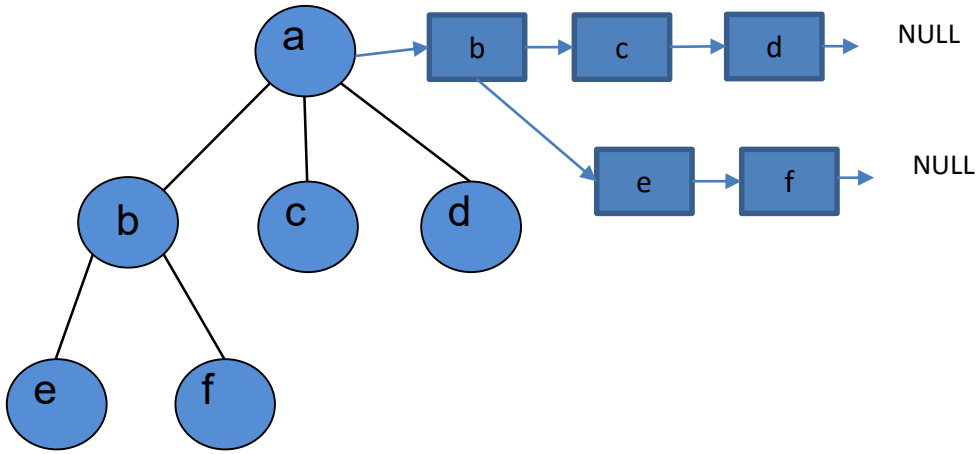
- ***Altura***: La ***altura de un nodo*** en un árbol es la longitud del camino más largo de ese nodo a una hoja. La **altura de un árbol** es la altura de la raíz.

- ***Profundidad***: La profundidad de un nodo es la longitud del camino único desde ese nodo a la raíz.

Implementación del TAD Árbol

- La implementación adecuada de esta estructura de datos deberá ser **inherentemente dinámica**.

- La implementación de un nodo es básicamente un objeto capaz de almacenar un elemento y una lista de los hijos que tiene (para ello puede utilizarse el **TAD Lista**).



Aplicaciones del TAD Árbol

- Los árboles se emplean en la mayoría de los casos donde es necesario **analizar** **un conjunto de posibles soluciones**, **conjuntos de solución y caminos en teoría de grafos o técnicas de análisis de algoritmos**.

- Los árboles son eficientes para realizar **búsquedas en conjuntos de información muy grandes**.

- También se emplean en la representación de **árboles sintácticos**, es decir, árboles que contienen las derivaciones de una **gramática** necesarias para obtener una determinada frase de un lenguaje.