Universidad Autónoma del Estado de México

UAEM

CU UAEM ZUMPANGO

CENTRO UNIVERSITARIO
UAEM ZUMPANGO

INGENIERÍA EN COMPUTACIÓN

Unidad de Aprendizaje:

ESTRUCTURA DE DATOS

Unidad de Competencia III:

ESTRUCTURA NO LINEAL - ÁRBOLES

M. en C. Edith Cristina Herrera Luna

Marzo 2019

ESTRUCTURA DE DATOS

PROPOSITO DE LA UNIDAD DE APRENDIZAJE:

+ Conocer, analizar y aplicar estructuras de datos estáticas y dinámicas mediante programas para la solución de problemas informáticos.

OBJETIVO DE LA UNIDAD DE COMPETENCIA III:

+ Aplicar la estructura de datos árbol en el desarrollo de soluciones a problemas informáticos.

INTRODUCCIÓN

- El estudio de las estructuras de datos, sin duda es uno de los más importantes dentro de las carreras relacionadas con la computación, ya que el conocimiento eficiente de las estructuras de datos suele ser imprescindible en la formación de los alumnos debido a la trascendencia que un aprendizaje teórico-práctico de las mismas supondrá para su carrera.

- Una de las aplicaciones más interesantes y potentes de los apuntadores son las estructuras dinámicas de datos. Las estructuras básicas disponibles tienen una importante limitación: no pueden cambiar de tamaño durante la ejecución.

- En muchas ocasiones se necesitan estructuras que puedan cambiar de tamaño durante la ejecución del programa. Por supuesto, se pueden usar arreglos dinámicos, pero una vez creados, su tamaño también será fijo, y para hacer que crezcan o disminuyan de tamaño, deberán reconstruirse desde el principio.

INTRODUCCIÓN

- Las estructuras dinámicas permiten crear estructuras de datos que se adapten a las necesidades reales a las que suelen enfrentarse los programas. Pero no solo eso, también permitirá crear estructuras de datos muy flexibles, ya sea en cuanto al orden, la estructura interna o las relaciones entre los elementos que las componen.

- El estudio de las estructuras de datos se hará desde diversos puntos de vista: diseño de estructuras en respuesta a necesidades especificas, encapsulamiento de los tipos de datos usándolos en base a su especificación (propiedades funcionales) y no a su implementación y estudio de los principales tipos, tanto elementales como no elementales, dividiendo éstos últimos en estructuras lineales (listas, pilas, colas) y no lineales (árboles, grafos), analizándolos primero desde el punto de vista teórico pero sin perder de vista sus aplicaciones prácticas.

ÁRBOLES

CONTENIDO

1. Recursividad
   - + ¿Qué es recursión?
   - + Ejemplos
2. Árboles
   - + ¿Qué es un árbol?
   - + Grado
   - + Altura y profundidad
3. Árboles binarios
   - + Árbol lleno y perfecto
   - + Árboles de expresión
   - + Árboles de búsqueda
4. Recorridos en árboles binarios

RECURSIVIDAD

PROGRAMACIÓN 1

¿QUÉ ES RECURSIÓN?

- Es una técnica de programación que consiste en resolver un problema por medio de una o más divisiones simples de sí mismo.

- Es un método que en su definición tiene una o más invocaciones a sí mismo.

¿QUÉ ES RECURSIÓN

Un algoritmo recursivo debe incluir:

- Caso base:

  Es el caso básico, de parada o “fin” del algoritmo. Es la resolución de manera directa. Pueden existir uno o mas casos base.

- Case recursivo:

  Es un caso en el que el problema se divide en versiones mas pequeñas de si mismo. Pueden existir uno o varios casos recursivos.

¿CÓMO DISEÑAR UN ALGORITMO RECURSIVO?

Problema: Escribe un programa recursivo que pida al usuario un número entero N e imprima en pantalla los números enteros N, N-1, N-2,…,3, 2, 1

1. Reconoce el caso base y proporciona una solución para el.
   - Imprimir el número 1.
2. Identifica el caso que se “repite”: diseña una estrategia para dividir el problema en versiones más pequeñas del mismo considerando avanzar hacia el caso base.
   - Siempre se imprime un número (que cada vez va en decremento).
3. Combina las soluciones de los problemas más pequeños para obtener la solución del problema original.

¿CÓMO DISEÑAR UN ALGORITMO RECURSIVO?

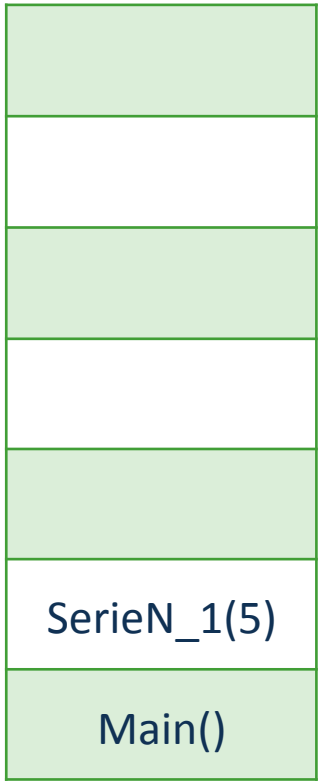
```
1   #include <iostream>
2
3   using namespace std;
4
5   void serieN_1( int n ){
6       if( n == 1 )                    Caso Base
7           cout<< n <<endl;
8       else{
9           cout<< n <<" ";
10          serieN_1(n - 1);        Caso Recursivo
11      }
12  }
13
14  int main(){
15
16      int num;
17      cout<<"\nIngresa un numero entero: "<<endl;
18      cin>>num;
19      serieN_1( num );
20
21      return 0;
22  }
```

¿CÓMO DISEÑAR UN ALGORITMO RECURSIVO?

Las pilas son útiles para realizar pruebas de escritorio y entender cómo funcionan los algoritmo recursivos.

Realiza la prueba de escritorio suponiendo que num = 5

SerieN_1(n)	n	n == 1	Impresión



¿Qué ocurre con el programa si primero se hace la llamada recursiva y posteriormente la impresión del número?

RECURSIÓN

- Escribe un programa recursivo que imprima cada termino de la siguiente serie y la suma hasta N, dado por el usuario.

Serie: $N^2 + (N-1)^2 + \ldots + 3^2 + 2^2 + 1^2 = S$

```
#include <iostream>
#include <cmath>
using namespace std;

int serie1( int n ){
    if( n <= 1 ){
        cout<< "1^2 ";
        return 1;
    }
    else{
        cout<< n << "^2 + ";
        return pow(n,2) + serie1( n-1 );
    }
}

int main(){

    int num;
    cout<<"\nIngresa un numero : "<<endl;
    cin>>num;
    long s = serie1( num );
    cout<< "= "<< s << endl;
    return 0;
}
```

RECURSIÓN

+ Escribe un programa recursivo que pida al usuario un número e imprima sus cifras en orden inverso, por ejemplo: 1234 muestra como resultado 4321.

Caso base: Imprimir un único digito (0-9).

Caso Recursivo:

- Imprimir el último dígito (resto de dividir entre 10) .
- Quitar el último dato para repetir el proceso (dividir entre 10).

```
1   #include <iostream>
2
3   using namespace std;
4
5   void invertir( int num ){
6       if( num <= 9 )
7           cout<< num << endl;
8       else{
9           cout<< num%10 << " ";
10          invertir( num/10 );
11      }
12  }
13
14  int main(){
15
16      int num;
17      cout<<"\nIngresa un numero : "<<endl;
18      cin>>num;
19      invertir( num );
20      return 0;
21  }
```

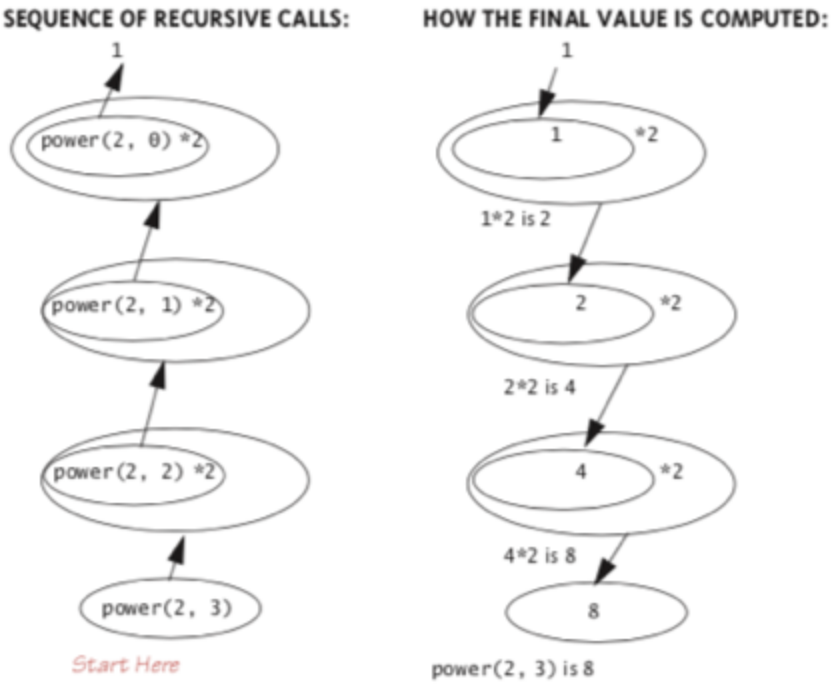
RECURSIÓN

- Cuando existen llamadas recursivas que retornan valores, se debe tener cuidado en realizar los procedimientos antes de regresar un valor, de acuerdo a dónde se encuentra la llamada recursiva. De igual forma, una pila es conveniente.

```
1  public class RecursionDemo2
2  {
3      public static void main(String[] args)
4      {
5          for (int n = 0; n < 4; n++)
6              System.out.println("3 to the power " + n
7                  + " is " + power(3, n));
8      }

9      public static int power(int x, int n)
10     {
11         if (n < 0)
12         {
13             System.out.println("Illegal argument to power.");
14             System.exit(0);
15         }

16         if (n > 0)
17             return ( power(x, n – 1)*x );
18         else // n == 0
19              return (1);
20     }
21 }
```



Fuente: Absolute Java. Walter Savitch.

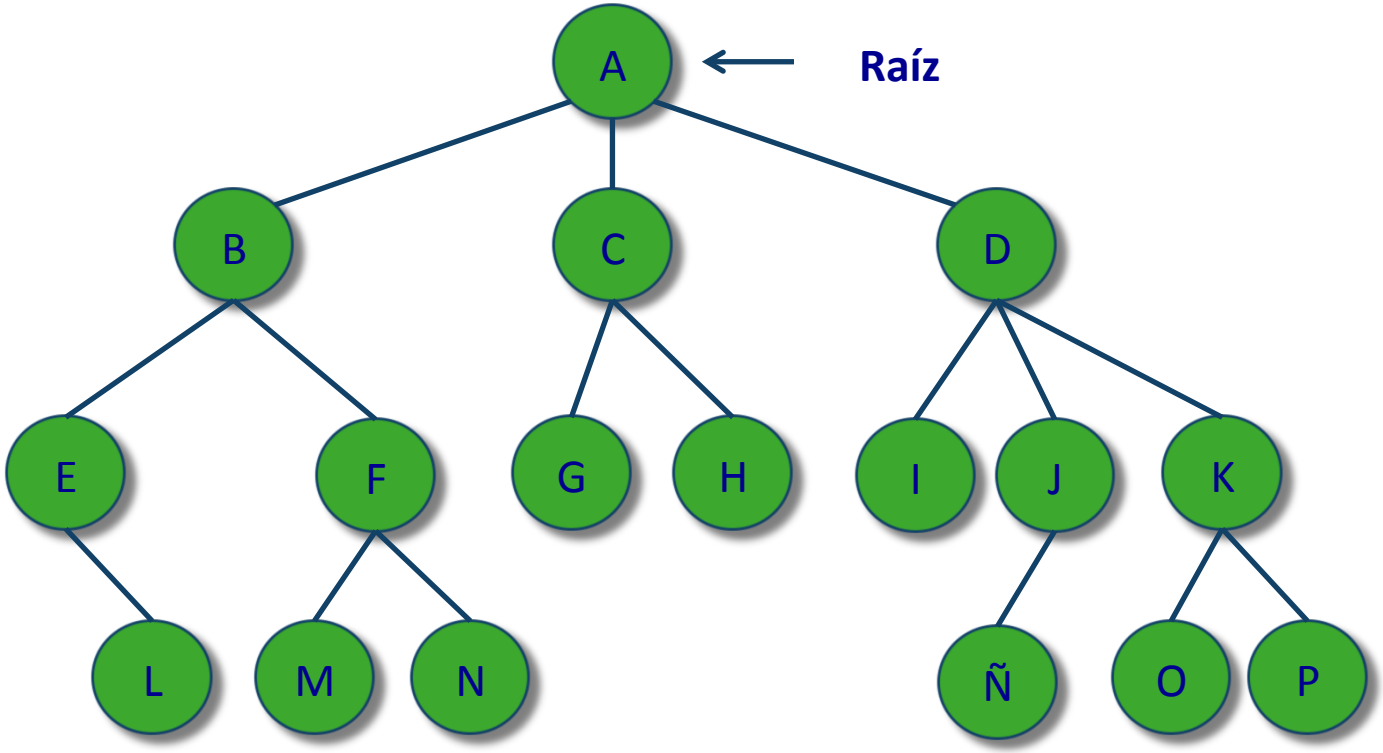
ÁRBOLES GENERALES

ESTRUCTURA DE DATOS

ED / ICO / ECHL

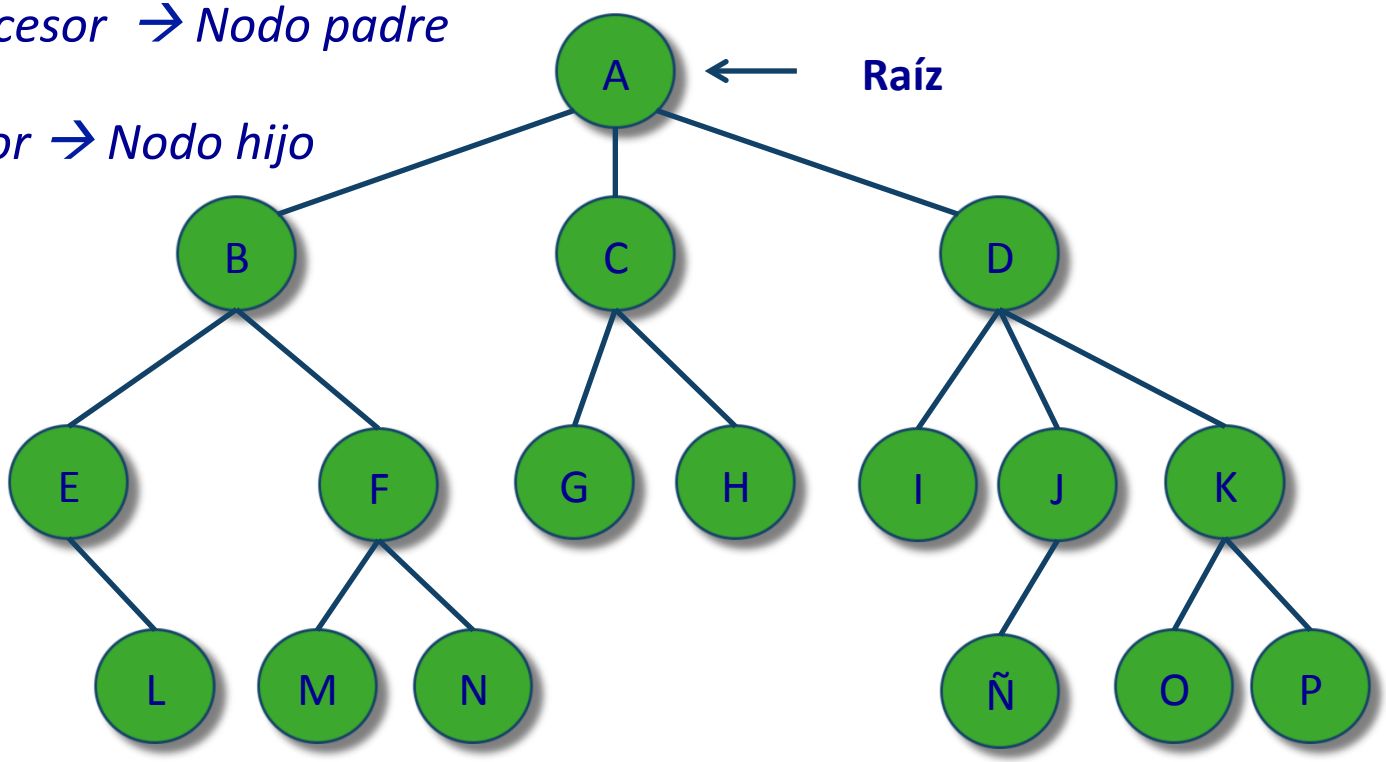
¿QUÉ ES UN ARBOL?

- Un **árbol** es un conjunto de nodos o elementos (que pueden ser nulos) organizados jerárquicamente, con cada nodo ligado a sus sucesores; consta de un nodo principal llamado raíz y cada nodo en el árbol -excepto la raíz- tiene un único padre.



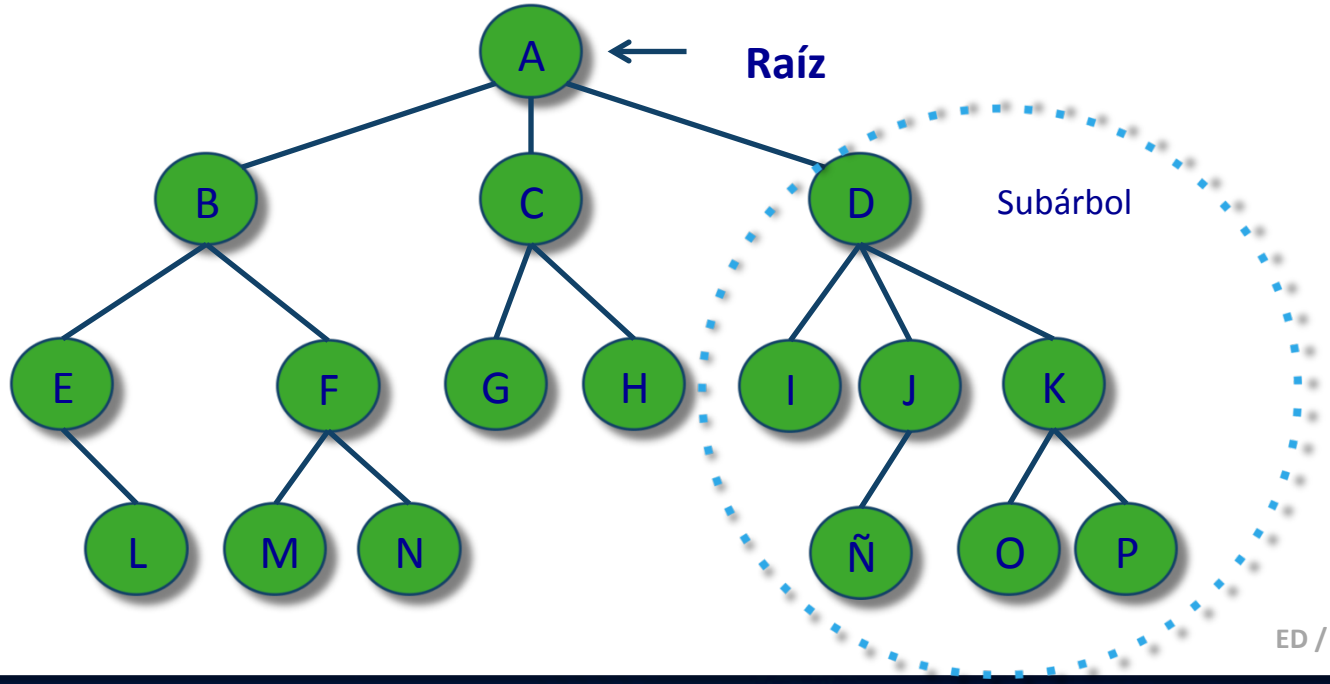
CARÁCTERÍSTICAS DE LOS ÁRBOLES

- Cada nodo en el árbol –a excepción de la raíz- tiene un único nodo padre.
- Los vínculos entre los nodos y sus nodos sucesores se denominan *ramas.*
- *Nodo predecesor → Nodo padre*
- *Nodo sucesor → Nodo hijo*



CARÁCTERÍSTICAS DE LOS ÁRBOLES

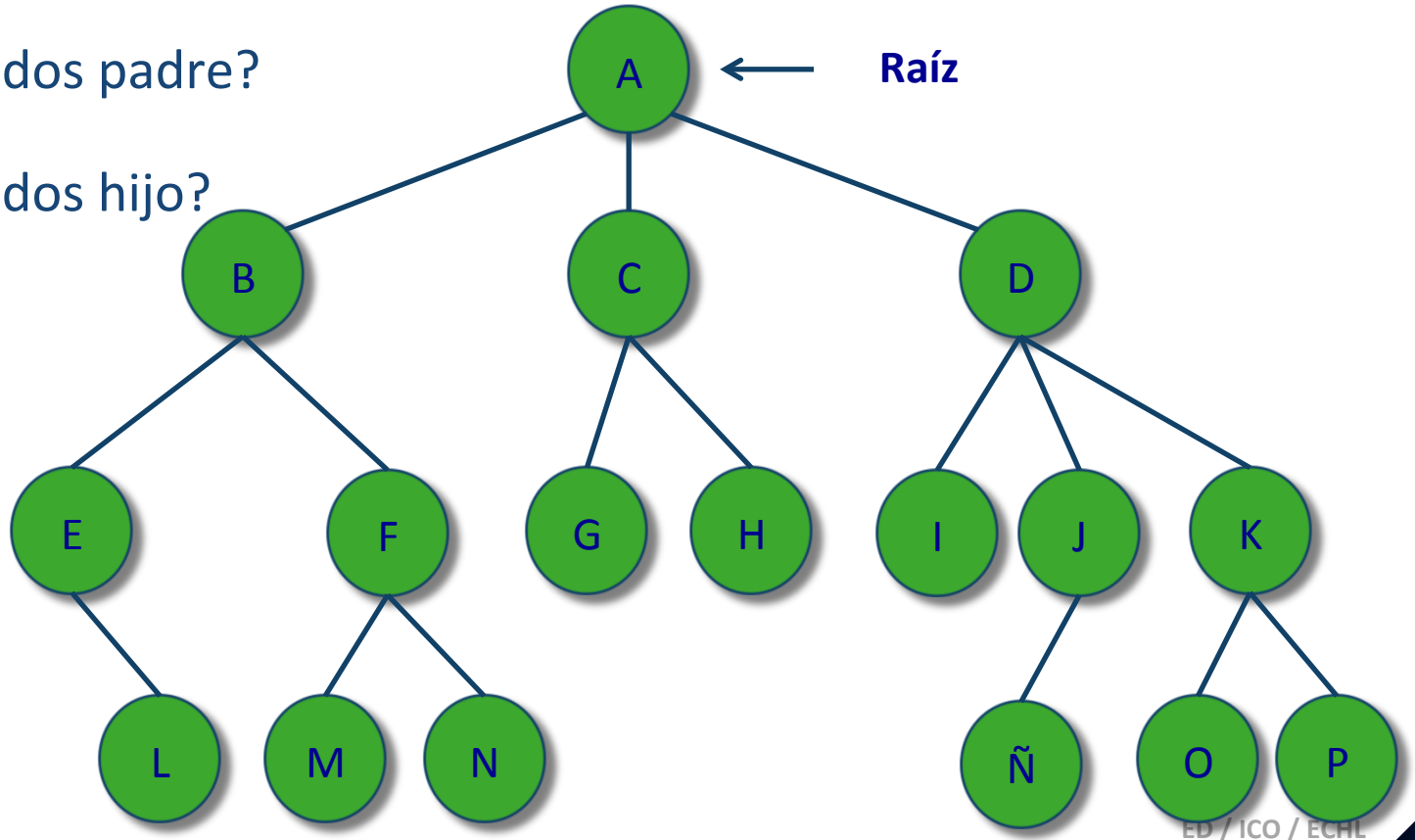
- Nodos hermanos → Nodos con el mismo nodo padre
- Nodo sin hijos → Nodo hoja, terminal, nodo externo
- Nodo que no es hoja ni raíz → Nodo interno
- Un árbol cuya raíz es un hijo de ese mismo nodo → Subárbol



CARÁCTERÍSTICAS DE LOS ÁRBOLES

En el árbol mostrado, indica:

+ ¿Cuántos nodo hoja existen?
+ ¿Cuántos nodos padre?
+ ¿Cuántos nodos hijo?



GRADO DE UN NODO

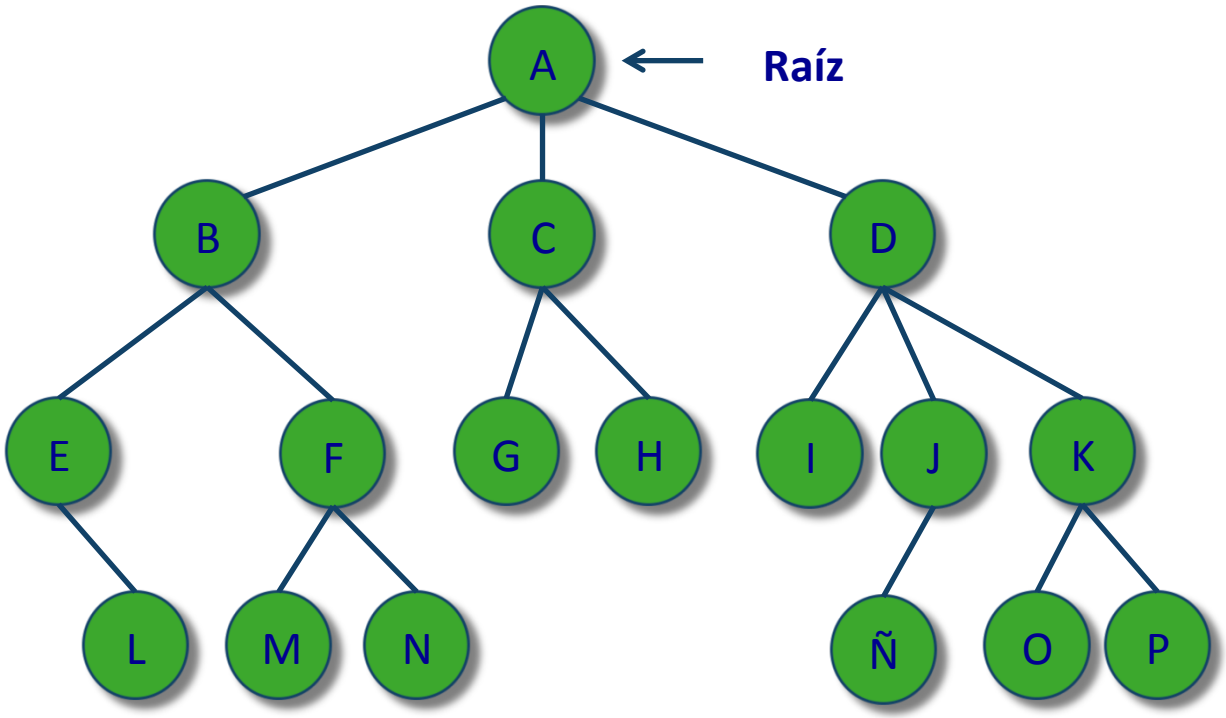
- Grado de un nodo → Es el número de nodos hijos que tiene.
- Grado de un árbol → Grado mayor de los nodos del árbol.

g(D) = 3

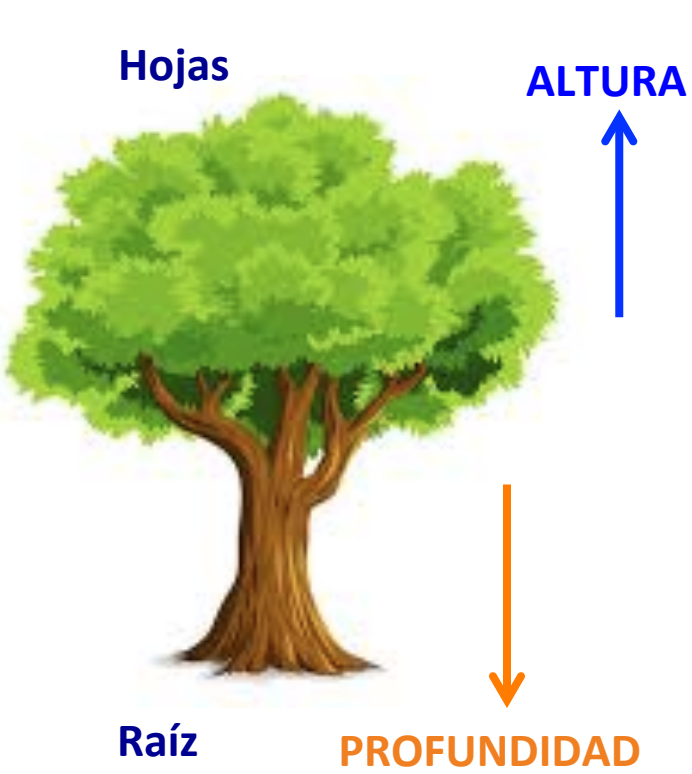
g(F) = 2

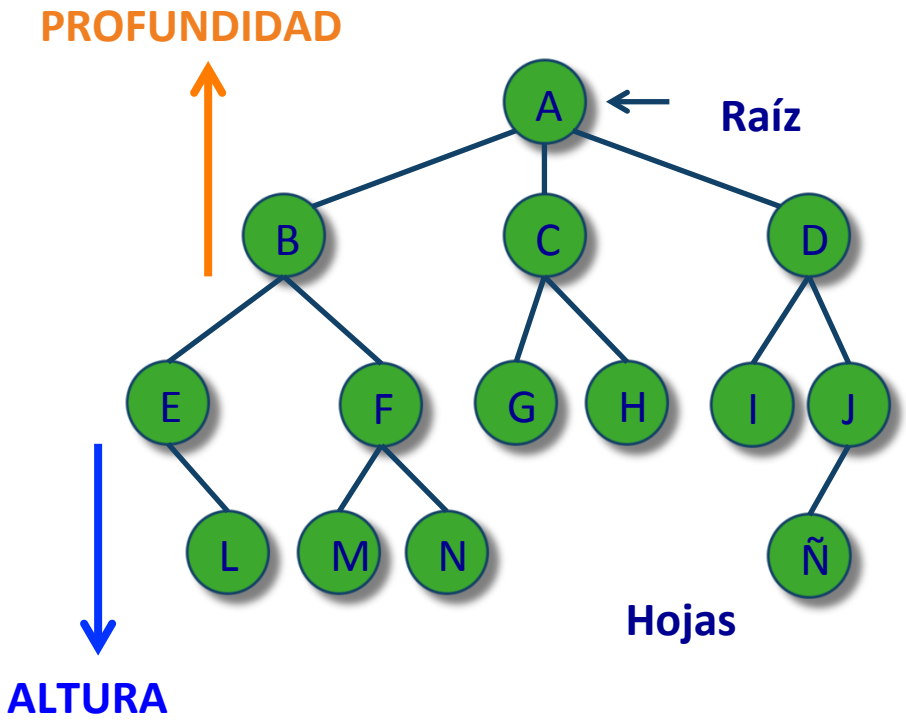
g(H) = 0

G = 3



ALTURA Y PROFUNDIDAD DE UN ARBOL





ALTURA DE UN NODO

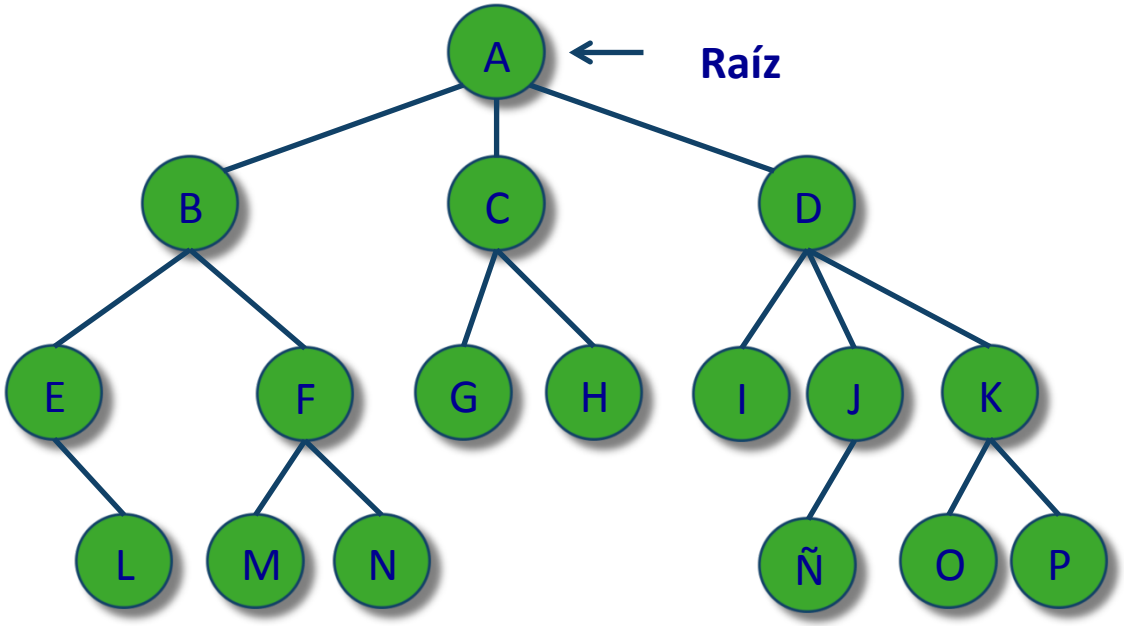
- ALTURA DE UN NODO → Número de nodos por el camino más largo desde ese mismo nodo hasta un nodo hoja. O el número de ramas por el camino más largo que va desde el nodo hasta el nodo hoja mas alejado, mas uno.

- ALTURA DEL ARBOL → Altura del nodo raíz.

h(f) = 2

h(d) = 3

H = 4



PROFUNDIDAD DE UN NODO

- PROFUNDIDAD → Es la medida de su distancia a la raíz. Se define recursivamente como:
  - Si el nodo *n* es la raíz del árbol *T*, su profundidad es cero.
  - Si el nodo *n* no es la raíz del árbol *T*, su profundidad es uno mas que la profundidad de su nodo padre

d(A) = 0

d(K) = d(D) + 1

=[d(A)+1]+1

=0 +1 +1 = 2