Complejidad Temporal, Estructuras de Datos y Algoritmos

Objetivos de la materia

- Analizar Algoritmos y evaluar su eficiencia
- Estudiar estructuras de datos avanzadas: su implementación y aplicaciones

¿De qué se trata el curso?

Estudiar formas inteligentes de organizar la información, de forma tal de obtener Algoritmos eficientes.

Listas, Pilas, Colas
Árboles Binarios
Árboles AVL
Árboles Generales
Heaps
Tablas de Dispersion
Grafos

Estructuras de Datos

Insertar
Borrar
Buscar
Caminos mínimos
Ordenación

Algoritmos

Las estructuras de datos y sus Algoritmos son….

Computación Gráfica

Seguridad

Multimedia

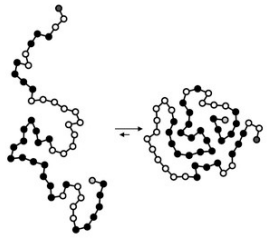
Biología

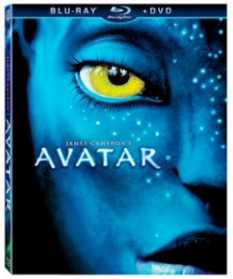
Inteligencia Artificial

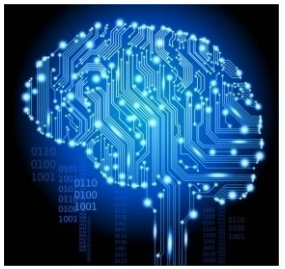
Internet

Física

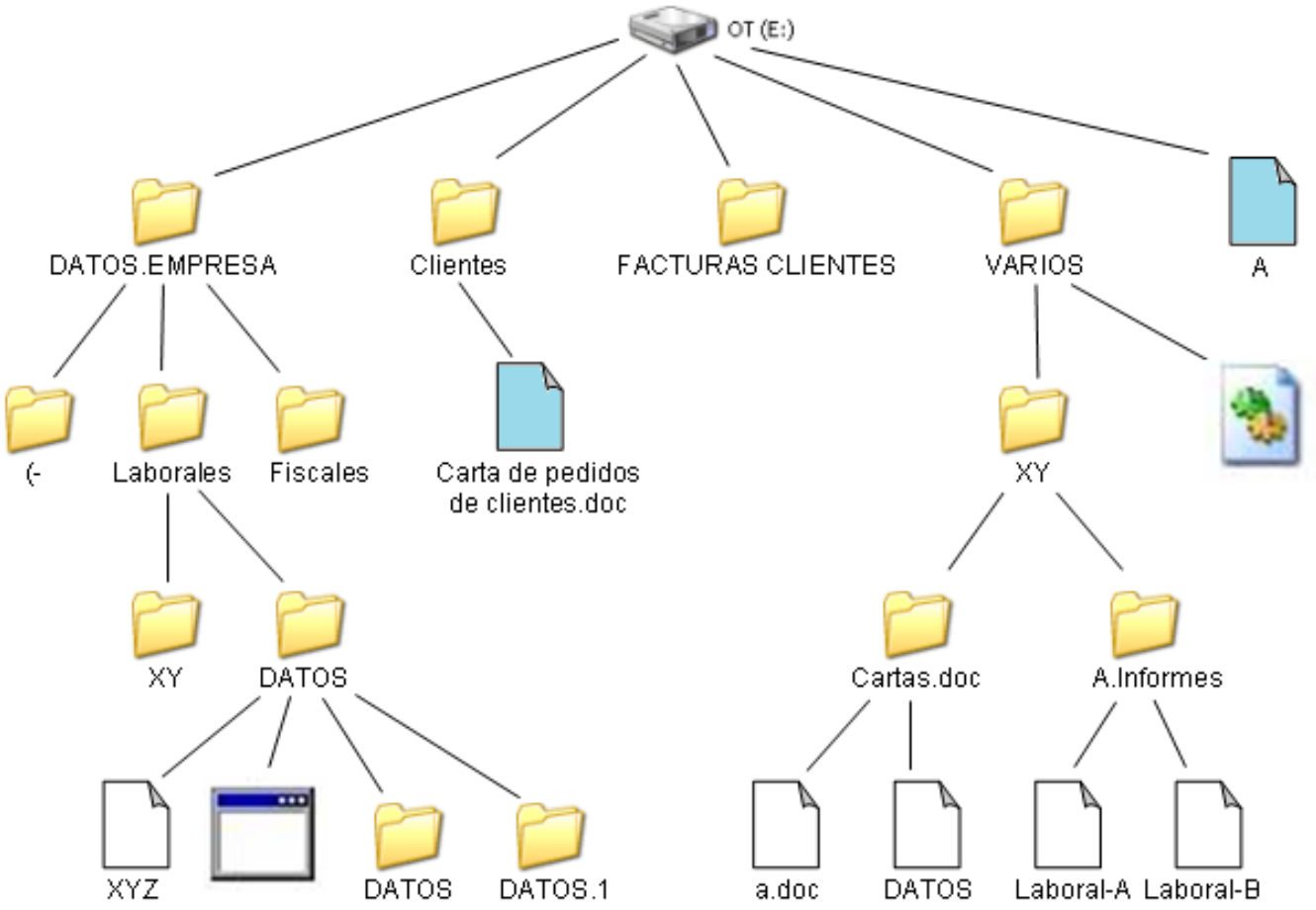
Utilizados en todos lados!







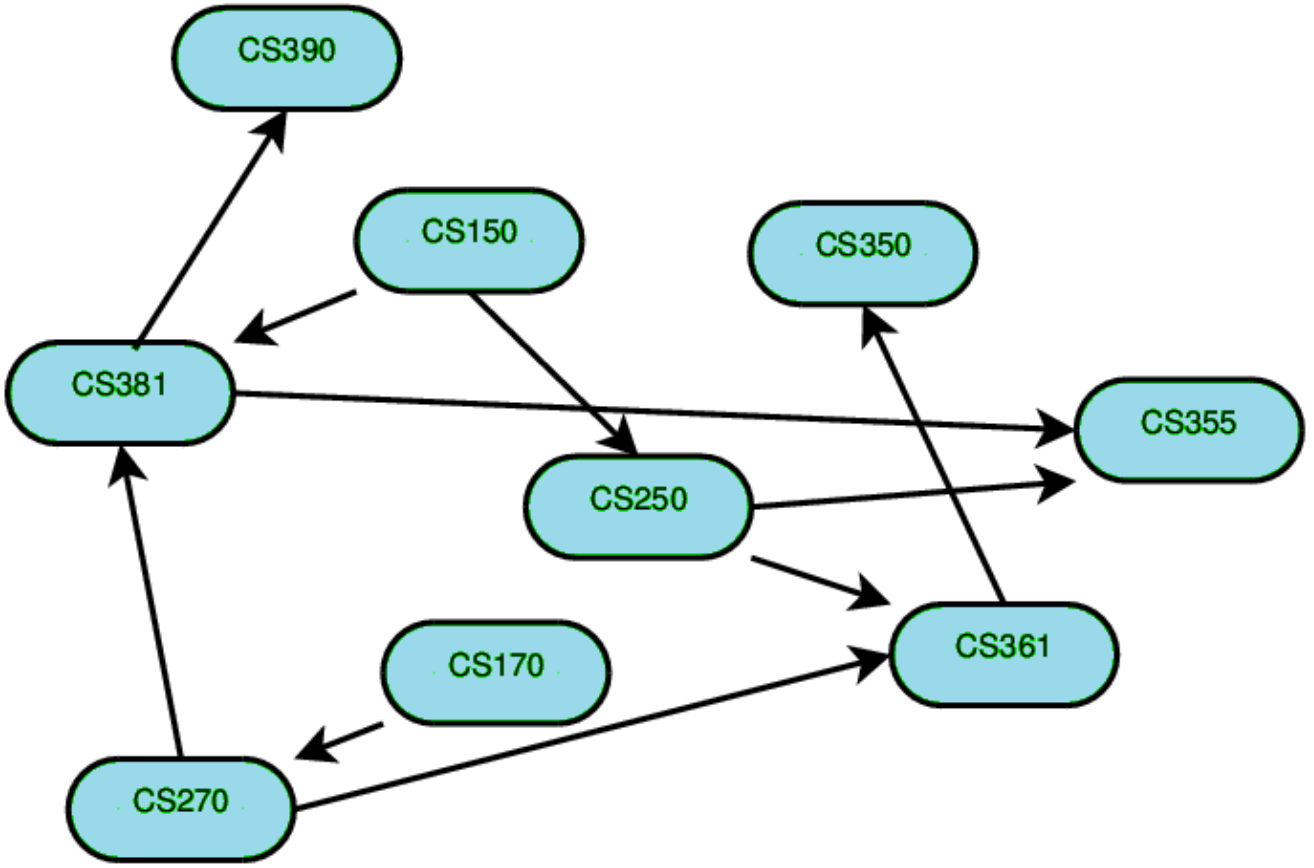
Ejemplo 1: Árbol de carpetas y archivos



Nodos: Carpetas/Archivos

Aristas: representan la relación “contiene”

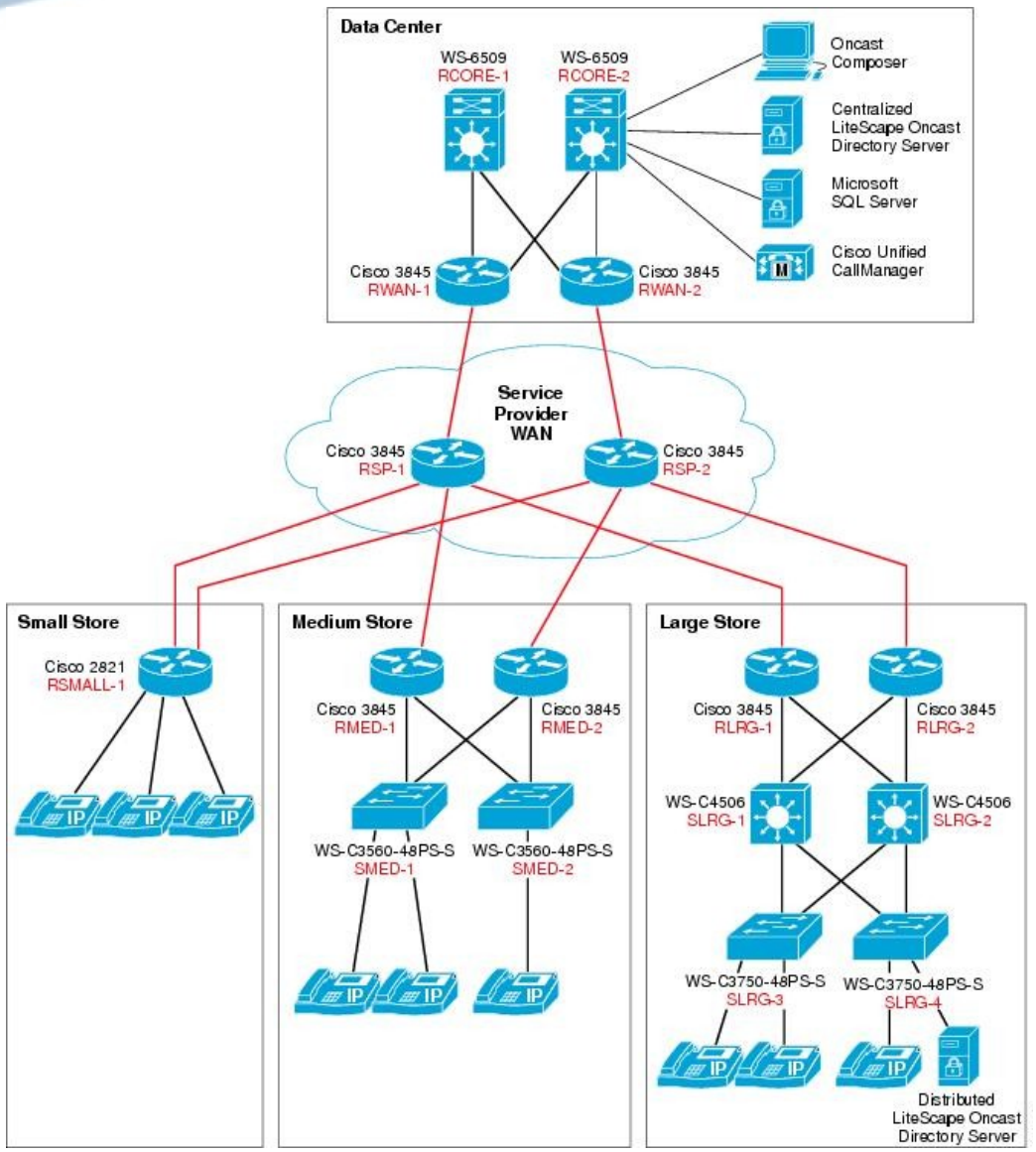
Ejemplo 2: Prerrequisitos de un curso



Nodos: Cursos

Aristas: relación de “prerrequisito”

Ejemplo 3: Esquema de una red informática

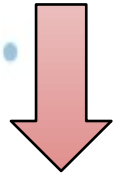


Nodos: Equipos

Aristas: representan las conexiones

Estructuras de Datos: Qué, Cómo y Por qué?

- Los programas reciben, procesan y devuelven datos
- Necesidad de organizar los datos de acuerdo al problema que vamos a resolver



Las estructuras de datos son formas de organización de los datos

Estructuras de Datos: Qué, Cómo y Por qué?

- Un programa depende fundamentalmente de la organización de los datos
  - cómo se organizan los datos está relacionado con:
    - Implementación de algunas operaciones: pueden resultar más fácil o más difícil
    - La velocidad del programa: puede aumentar o disminuir
    - La memoria usada: puede aumentar o disminuir

Objetivos del curso respecto de las Estructuras de Datos

- Aprender a implementar las estructuras de datos.
- Estudiar diferentes representaciones e implementaciones para las estructuras de datos
- Aprender a elegir la “mejor” estructura de datos para cada problema

Algoritmos y su Análisis

- ¿Qué es un Algoritmo?
  - Es una secuencia de pasos que resuelven un problema
  - Es independiente del lenguaje de programación
- Existen varios Algoritmos que resuelven correctamente un problema
- La elección de un Algoritmo particular tiene un enorme impacto en el tiempo y la memoria que utiliza

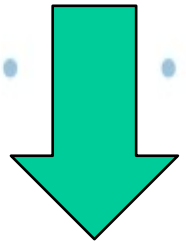
La elección de un Algoritmo y de la estructura de datos para resolver un problema son interdependientes

Objetivos del curso respecto del Análisis de los Algoritmos

- Entender los fundamentos matemáticos necesarios para analizar Algoritmos
- Aprender a comparar la eficiencia de diferentes Algoritmos en términos del tiempo de ejecución
- Estudiar algunos Algoritmos estándares para el manejo de las estructuras de datos y aprender a usarlos para resolver nuevos problemas

Problemas y Algoritmos

- Problemas:
  - Buscar un elemento en un arreglo
  - Ordenar una lista de elementos
  - Encontrar el camino mínimo entre dos puntos



Encontrar **el Algoritmo** que lo resuelve

Caso: Buscar un elemento en un arreglo

El arreglo puede estar:

- desordenado
- ordenado

Si el arreglo está **desordenado** → **Búsqueda secuencial**

64	13	93	97	33	6	43	14	51	84	25	53	95
0	1	2	3	4	5	6	7	8	9	10	11	12

Algoritmo: Búsqueda secuencial

```
public static int seqSearch(int[] a, int
   key)
{
   int index = -1;
   for (int i = 0; i < N; i++)
       if (key == a[i])
           index = i;
   return index;
}
```

¿Cuántas comparaciones hace?

Caso: Buscar un elemento en un arreglo

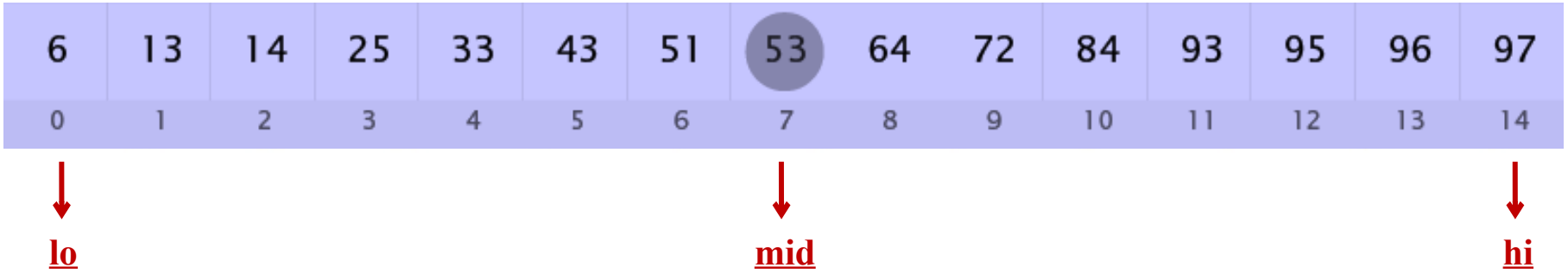
El arreglo puede estar:

- desordenado
- ordenado

Si el arreglo está **ordenado** →

Búsqueda binaria: Comparo la clave con la entrada del centro

- Si es menor, voy hacia la izquierda
- Si es mayor, voy hacia la derecha
- Si es igual, la encontré



Algoritmo: Búsqueda binaria

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length-1;
   while (lo <= hi)
   {
       int mid = lo + (hi - lo) / 2;
       if (key < a[mid]) hi = mid - 1;
       else if (key > a[mid]) lo = mid + 1;
       else return mid;
   }
return -1;
}
```

¿Cuántas comparaciones hace?

¿Cuántas operaciones hace cada Algoritmo?

Búsqueda secuencial

N	Cantidad de operaciones
1000	1000
2000	2000
4000	4000
8000	8000
16000	16000

Hace N operaciones

Búsqueda binaria

N	Cantidad de operaciones
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14

Hace log(N) operaciones

¿Cómo medir el tiempo?

- ✓ **Manual**

  Tomando el tiempo que tarda

- ✓ **Automática**

  Usando alguna instrucción del lenguaje para medir tiempo

Análisis empírico

Correr el programa para varios tamaños de la entrada y medir el tiempo. Suponemos que cada comparación tarda 1 seg.

Búsqueda secuencial

N	Tiempo (seg)
1000	1000
2000	2000
4000	4000 ~ 1 hs.
8000	8000 ~ 2 hs
16000	16000 ~ 4 hs.

Búsqueda binaria

N	Tiempo (seg)
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14

Análisis de Algoritmos

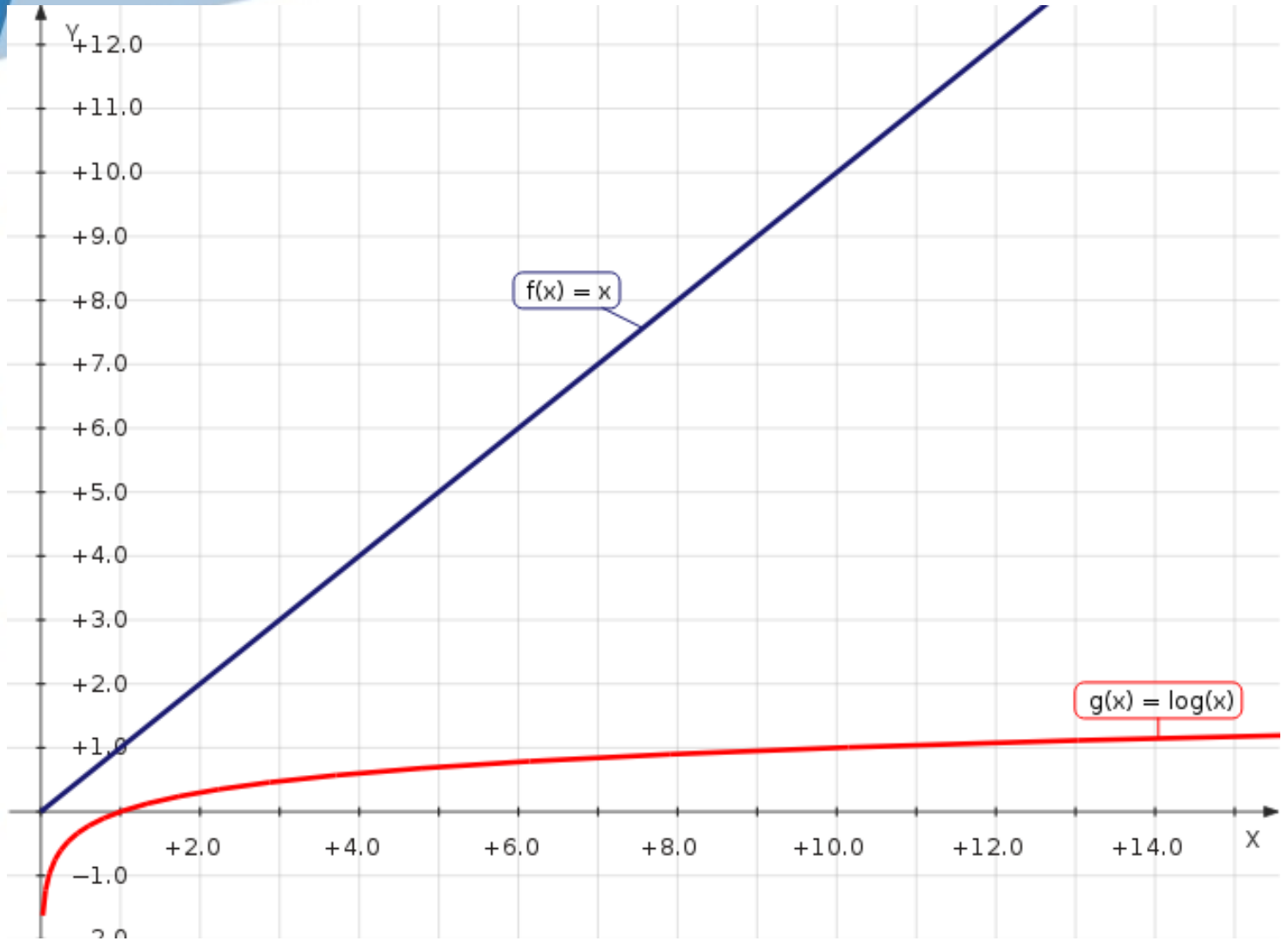
Existe un modelo matemático para medir el tiempo

Tiempo total:

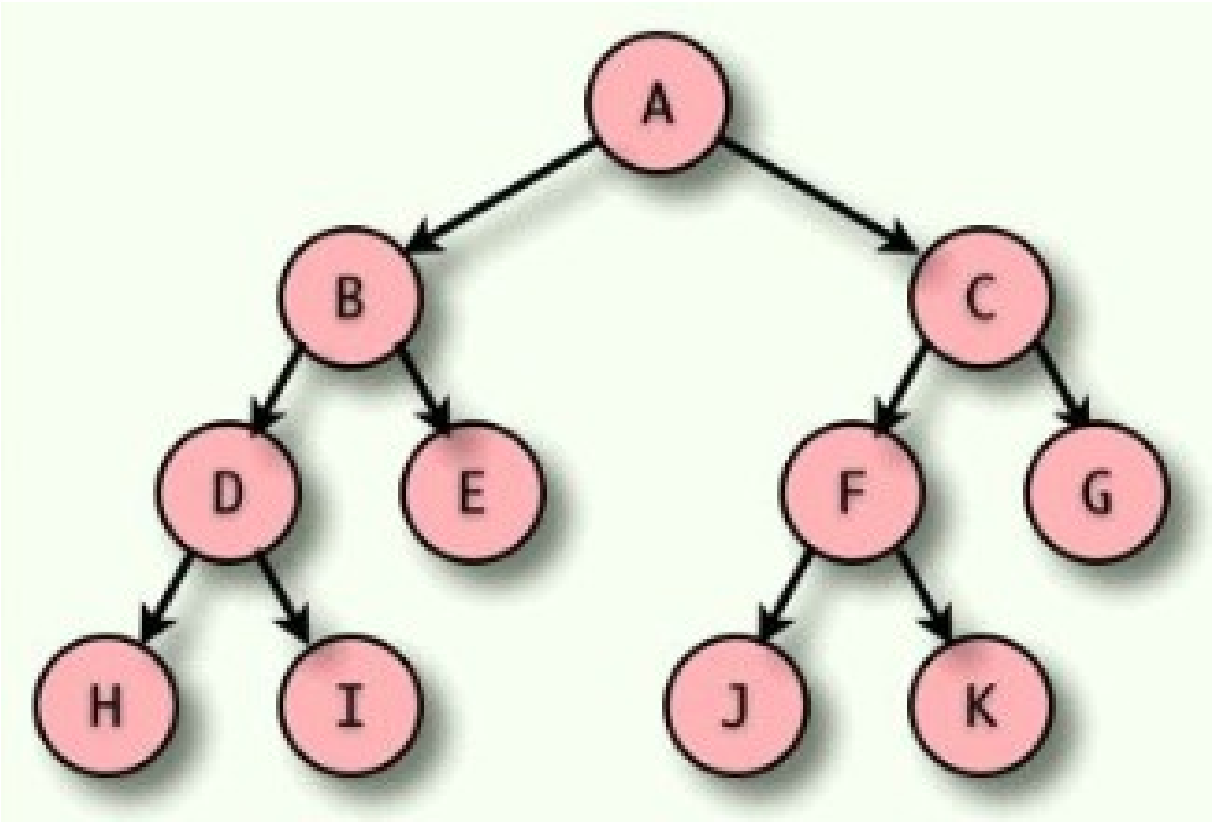
*Suma del **costo** x **frecuencia** de todas las operaciones*

- Es necesario analizar el Algoritmo para determinar el conjunto de operaciones
  - **Costo** depende de la máquina, del compilador, del lenguaje
  - **Frecuencia** depende del Algoritmo y de la entrada

Análisis de Algoritmos



Árboles Binarios



Árbol Binario: Definición

- *Un árbol binario es una colección de nodos, tal que:*
  - *puede estar vacía*
  - *puede estar formada por un nodo distinguido R, llamado **raíz** y dos sub-árboles T_1 y T_2, donde la raíz de cada subárbol T_i está conectado a R por medio de una arista*

Descripción y terminología

- Cada nodo puede tener a lo sumo dos nodos hijos.
- Cuando un nodo no tiene ningún hijo se denomina ***hoja***.
- Los nodos que tienen el mismo nodo padre se denominan ***hermanos***.

Descripción y terminología

- Conceptos a usar:
  - ***Camino***: desde n_1 hasta n_k, es una secuencia de nodos n_1, n_2, ….,n_k tal que n_i es el padre de n_{i+1}, para $1 \leq i < k$.
    - La longitud del camino es el número de aristas, es decir k-1.
    - Existe un camino de longitud cero desde cada nodo a sí mismo.
    - Existe un único camino desde la raíz a cada nodo.
  - ***Profundidad***: de n_i es la longitud del único camino desde la raíz hasta n_i.
    - La raíz tiene profundidad cero.

Descripción y terminología

- ***Grado*** de n_i es el número de hijos del nodo n_i .
- ***Altura*** de n_i es la longitud del camino más largo desde n_i hasta una hoja.
  - Las hojas tienen altura cero.
  - La altura de un árbol es la altura del nodo raíz.
- ***Ancestro/Descendiente***: si existe un camino desde n_1 a n_2, se dice que n_1 es ancestro de n_2 y n_2 es descendiente de n_1.

Descripción y terminología

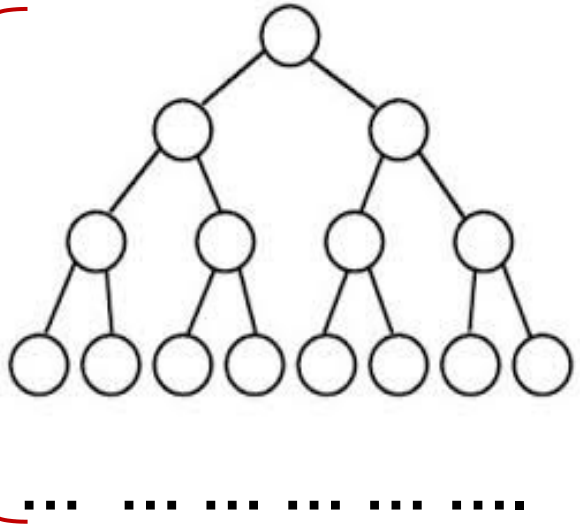
- ***Árbol binario lleno***: Dado un árbol binario T de altura ***h***, diremos que T es *lleno* si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h).

  Es decir, recursivamente, T es *lleno* si :
  1.- T es un nodo simple (árbol binario lleno de altura 0), o
  2.- T es de altura h y sus sub-árboles son llenos de altura h-1.

Descripción y terminología

- ***Cantidad de nodos en un árbol binario lleno***:

  Sea T un árbol binario lleno de altura ***h***, la cantidad de nodos N es $(2^{h+1} - 1)$



Nivel 0 → 2^0 nodos

Nivel 1 → 2^1 nodos

Nivel 2 → 2^2 nodos

Nivel 3 → 2^3 nodos

……

Nivel h → 2^h nodos

$N = 2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^h$

La suma de los términos de una serie geométrica de razón 2 es:

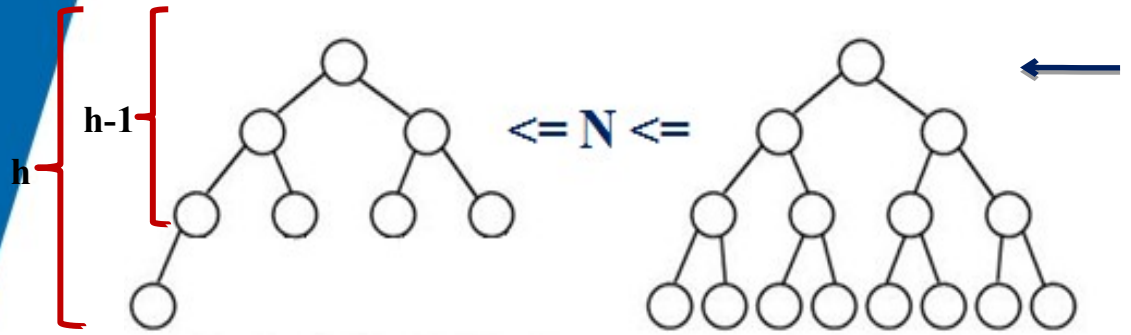
$(2^{h+1} - 1)$

Descripción y terminología

- ***Árbol binario completo***: Dado un árbol binario T de altura h, diremos que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.

- ***Cantidad de nodos en un árbol binario completo***:

  Sea T un árbol binario completo de altura ***h***, la cantidad de nodos N varía entre (2^{h}) y $(2^{h+1} - 1)$



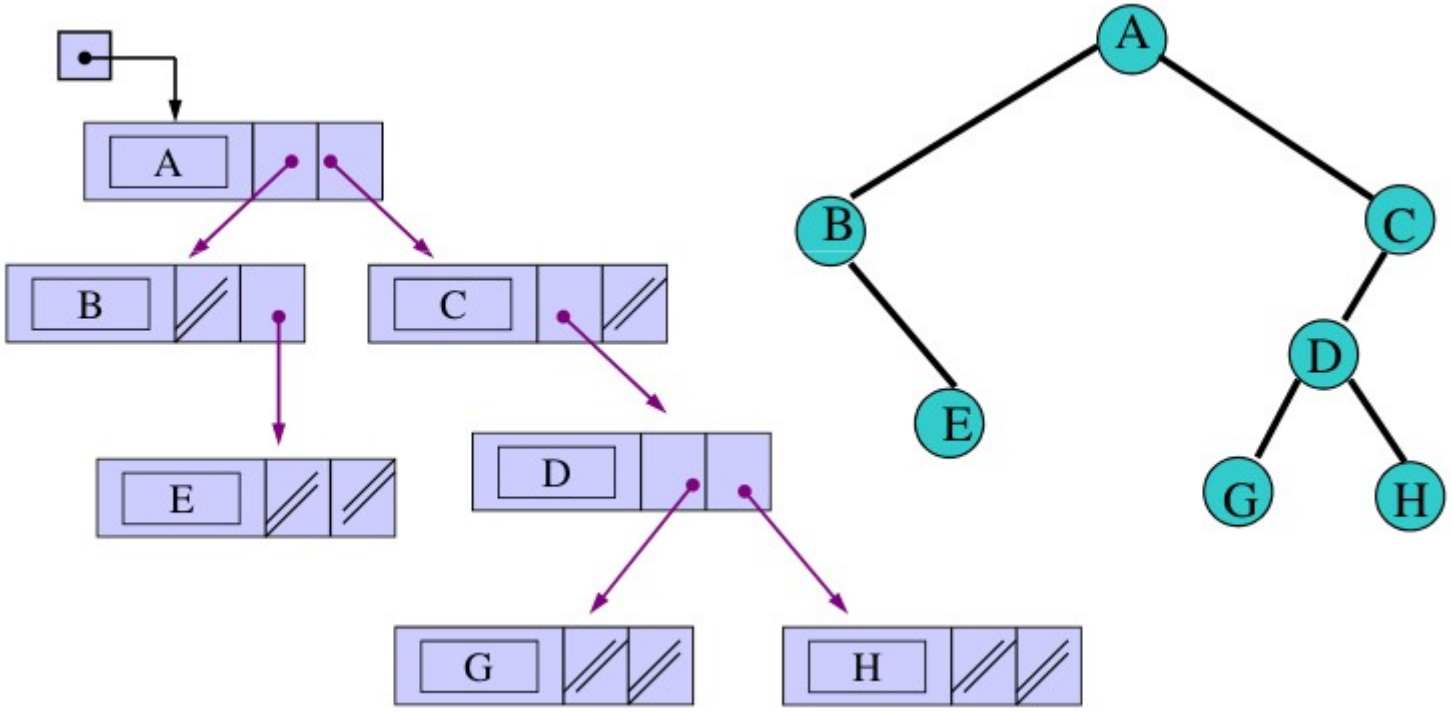
- Si el árbol es lleno
  $\mathbf{N} = (2^{h+1}-1)$

- Si no, el árbol es lleno en la altura *h-1* y tiene por lo menos un nodo en el nivel *h:*
$\mathbf{N} = (2^{h-1+1}-1)+1=(2^{h}-1+1)$

Representación

- ArbolBinario:
  En un árbol binario vacío su referencia es nula.
  En caso de no estar vacío, tiene:
  Información propia del nodo
  - Un ArbolBinario como hijo izquierdo
  - Un ArbolBinario como hijo derecho

Representación



Recorridos

- **Preorden**

  Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.

- **Inorden**

  Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho

- **Postorden**

  Se procesan primero los hijos, izquierdo y derecho, y luego la raíz

- **Por niveles**

  Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

Recorrido: Preorden

```
public void preorden() {
   imprimir (dato);
   si (tiene hijo_izquierdo)
     hijoIzquierdo.preorden();
   si (tiene hijo_derecho)
     hijoDerecho.preorden();
}
```

Recorrido: Por niveles

```
public void  porNiveles() {
   encolar(raíz);
      mientras (cola no se vacíe) {
         desencolar(v);
         imprimir (dato de v);
         si (tiene hijo_izquierdo)
            encolar(hijo_izquierdo);
      si (tiene hijo_derecho)
            encolar(hijo_derecho);
     }
}
```