Unidad 3. Estructura de Datos

Introducción

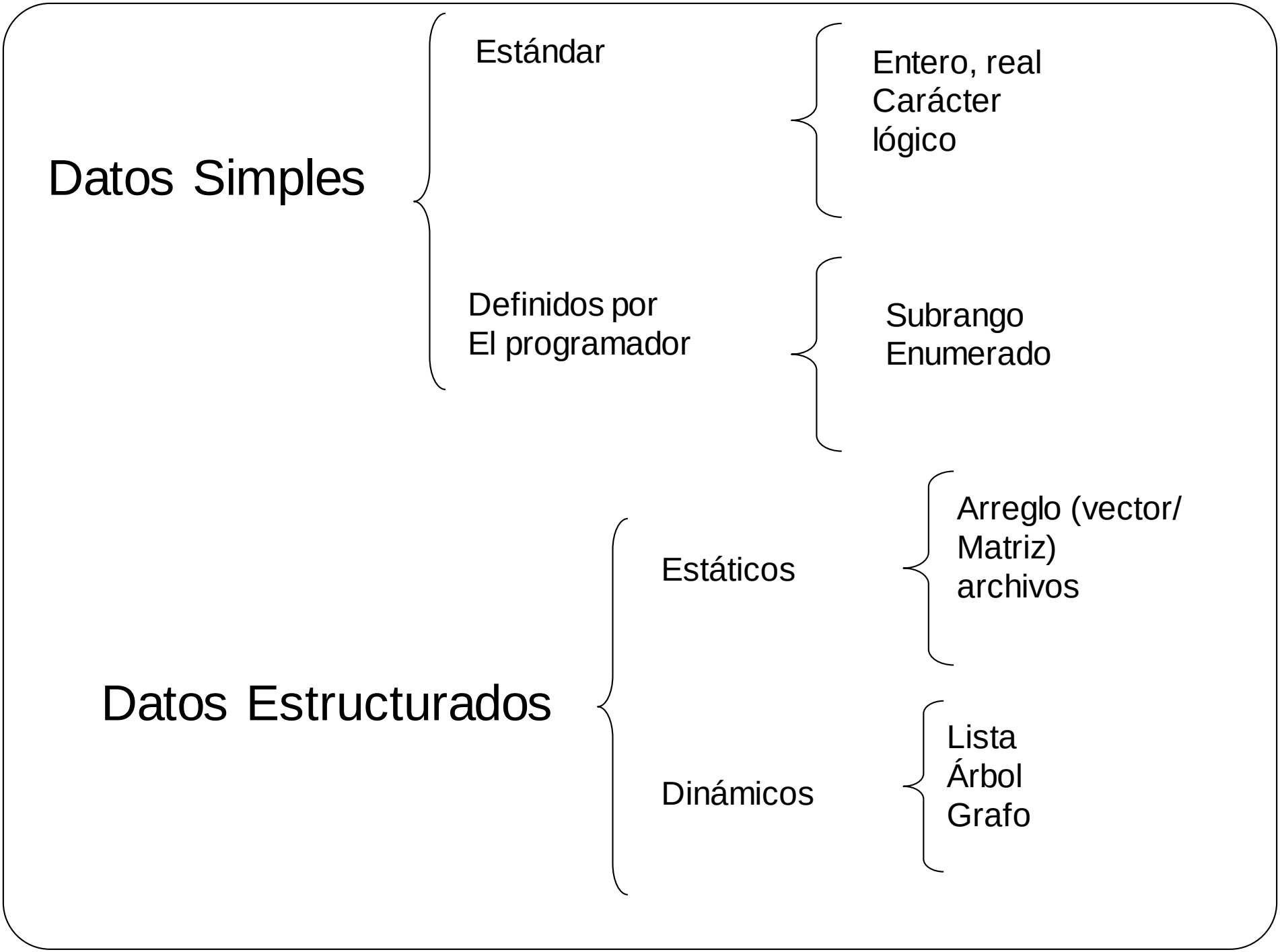
- Niklaus Wirth es autor del libro:
  - Programas = Algoritmos + Estructuras de datos
- Uno de los aspectos mas importantes de la programación es la creación de nuevos tipos de datos que sean apropiados para resolver problemas específicos.

Datos e Información

- Un dato es una representación simbólica de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades (Wikipedia)
- Los **datos** son cifras o elementos no significativos como tales sino hasta que son procesados y convertidos a una forma útil llamada **información**.
  Son la materia prima para obtener información.
- La **información** es el resultado del proceso de los datos.

Definición Estructura de Datos

Colección de datos que pueden ser caracterizados por su organización y las operaciones que se definen en ella.

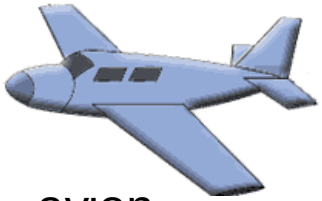

Datos Simples
Estándar
Entero, real
Carácter
lógico
Definidos por
El programador
Subrango
Enumerado
Datos Estructurados
Estáticos
Arreglo (vector/
Matriz)
archivos
Dinámicos
Lista
Árbol
Grafo

OBJETOS: COMPONENTES Y CARACTERÍSTICAS

computadora

empleado

factura



avion

reporte

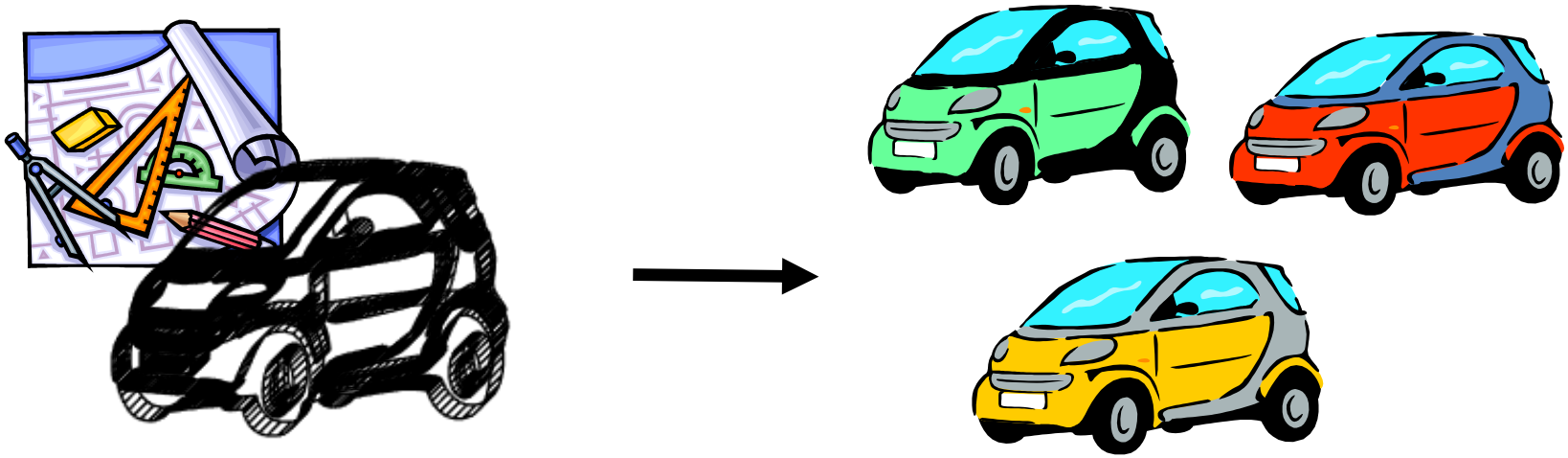
"Un objeto es un concepto, abstracción o cosa con un significado y límites claros en el problema en cuestión" Rumbaugth

posee

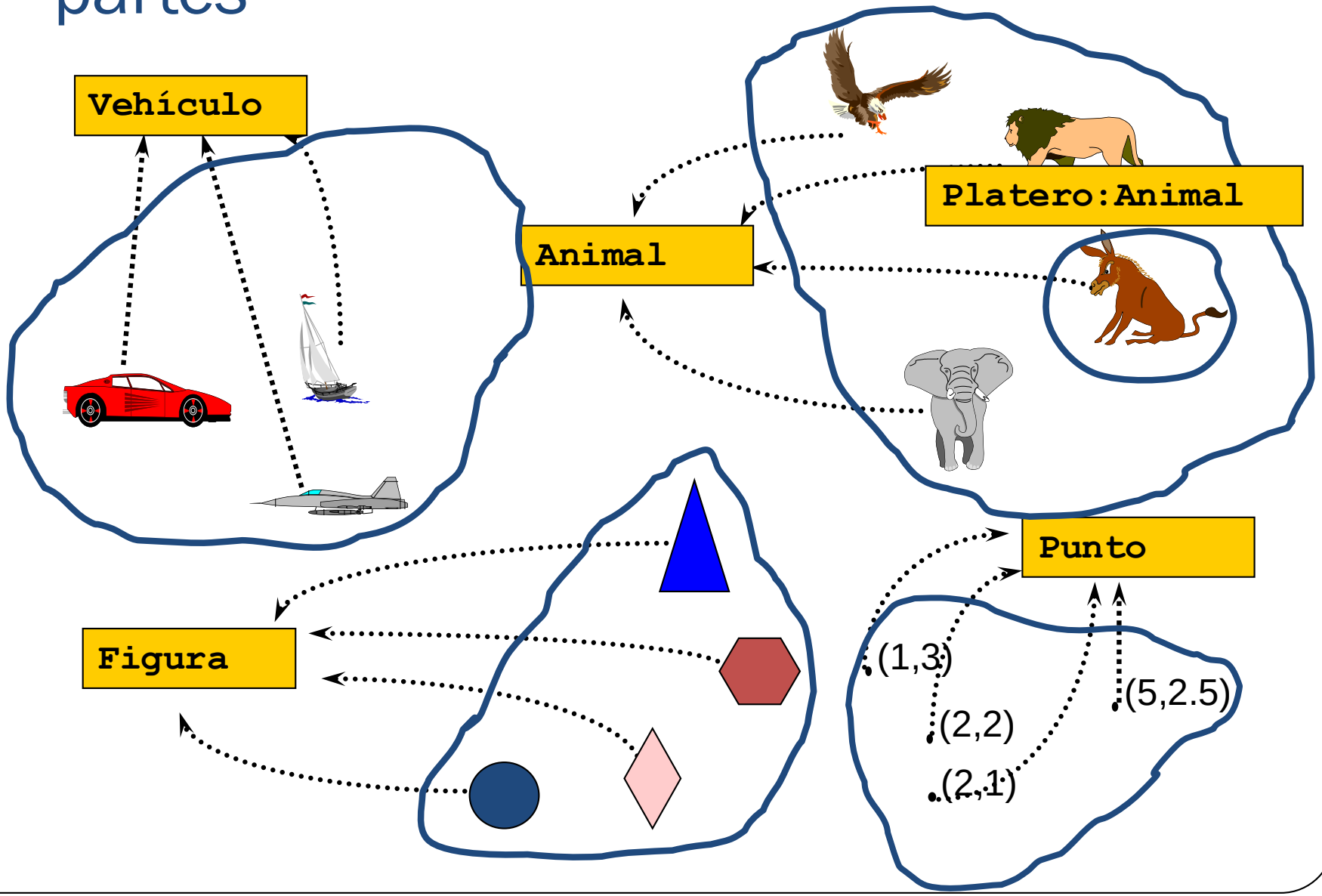
Estado
Comportamiento
Identidad (Boch)

Clases

- Una clase es una definición abstracta de un objeto
  - Define la estructura y el comportamiento compartidos por los objetos
  - Sirve como modelo para la creación de objetos
- Los objetos con similares caractreristicas pueden ser agrupados en clases

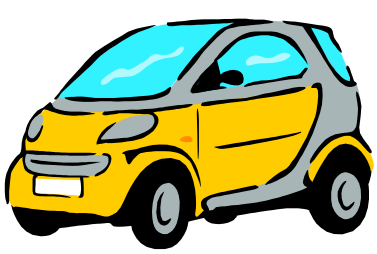
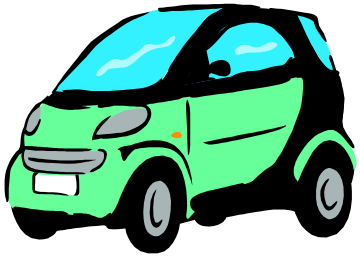
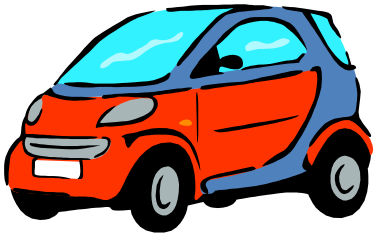
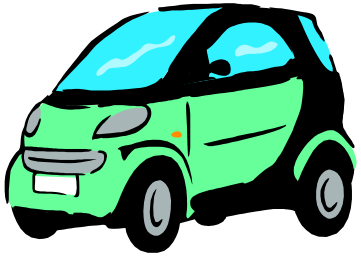


Las clases y los objetos están en todas partes



Identidad de objeto (OID)

- La identidad del objeto está representada por una ID de objeto (OID, por sus siglas en inglés), la cual es única de ese objeto.
- La OID es asignada por el sistema al momento de la creación del objeto y no puede ser cambiada en ninguna circunstancia.

Conceptos orientados a objetos

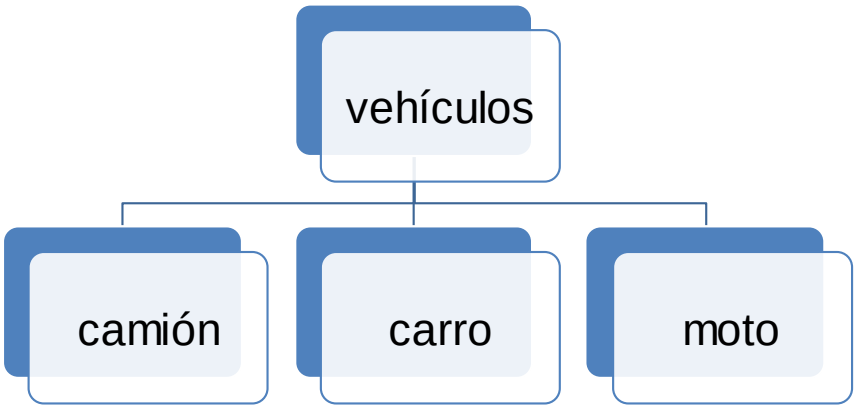
Atributos (Variables de instancia) → Describe a los objetos

Estado → Conjunto de valores que los atributos del objeto tienen en un momento dado

Mensajes y métodos → Código que realiza una operación específica con los datos del objeto.

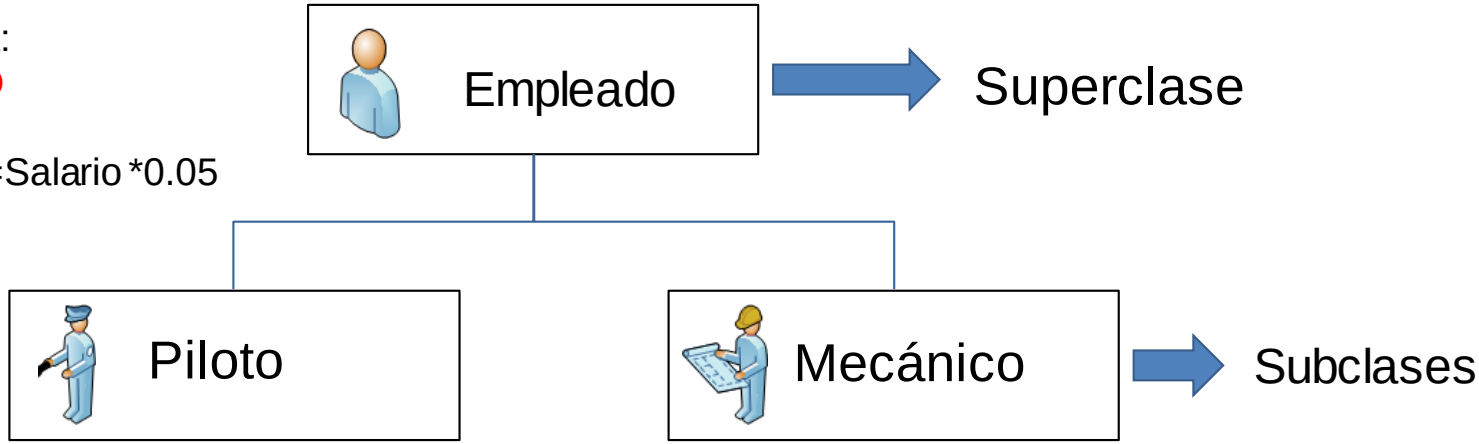
Protocolo → Conjunto de mensajes de clase, representa el aspecto público de un objeto

Herencia → Las clases se organizan en una jerarquía de clase



- Polimorfismo → Anularse una definición de método de superclase redefiniendo el método a nivel de subclase

Variable de instancia:
SALARIO
Método :
BONUS=Salario *0.05



Variable de instancia:
PAGOPORVUELO
Método :
BONUS=PagoPorVuelo *0.05

3.1. Técnicas de Enumeracion

¿Que es Enumeración?

- Una enumeración es un conjunto de constantes con nombre denominado lista de enumeradores
- Proporciona una manera eficaz de definir un conjunto de constantes enteras con nombre que se pueden asignar a una variable

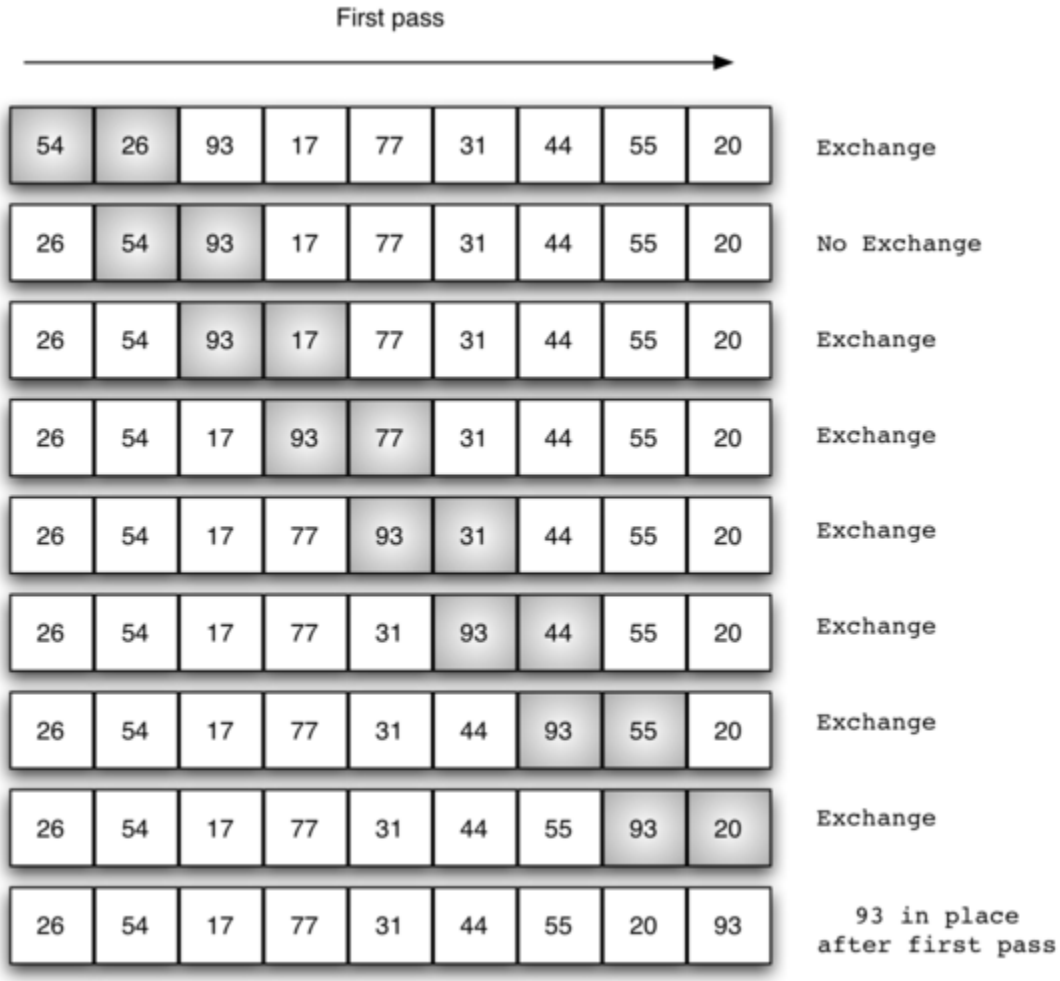
```
class Program
{
    enum DiasSemana
    {
        Lunes,
        Martes,
        Miercoles,
        Jueves,
        Viernes,
        Sabado,
        Domingo
    }

    static void Main(string[] args)
    {
    Console.WriteLine((int) DiasSemana.Lunes);
    Console.WriteLine((int) DiasSemana.Viernes);
    }
}
```

3.2. Búsquedas y Ordenamiento

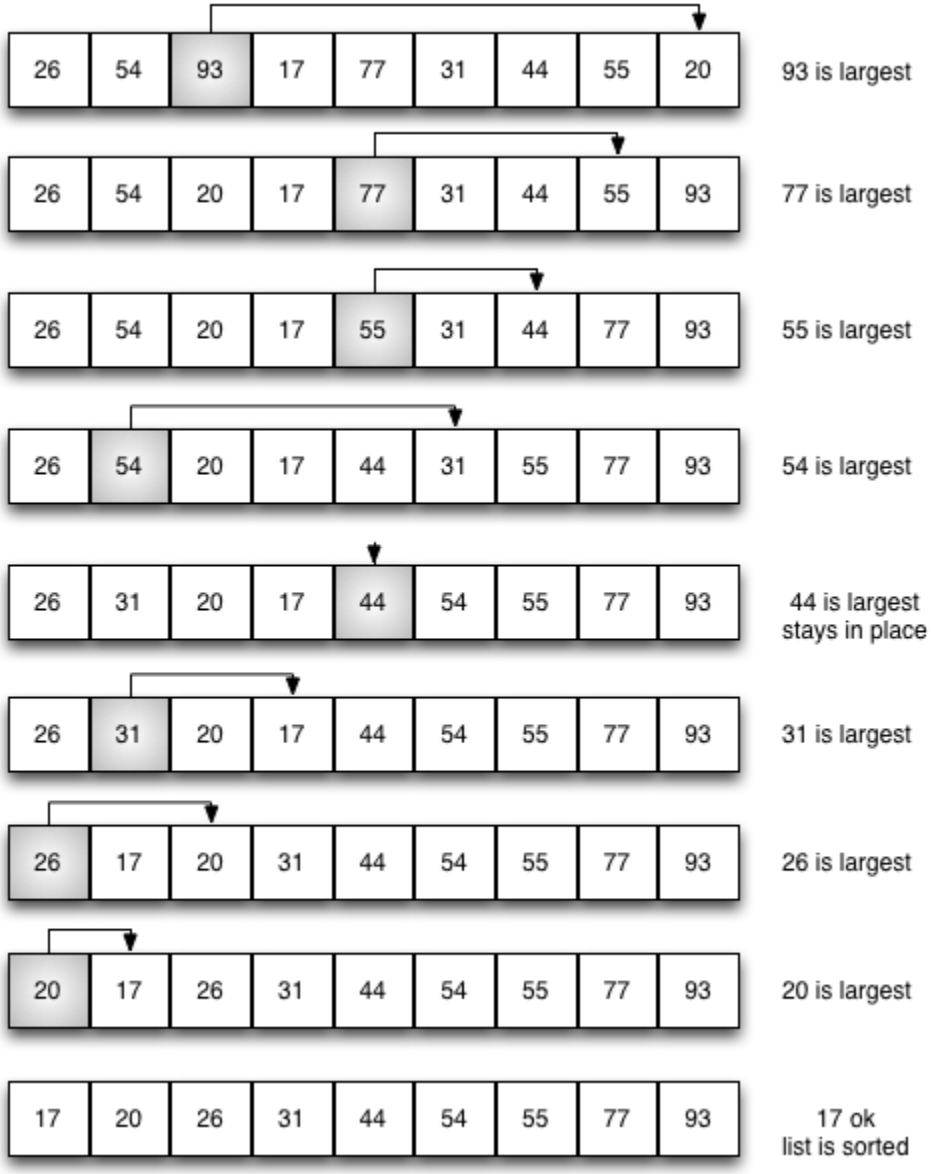
Ordenamiento

Es el proceso de ubicar elementos de una colección en algún orde

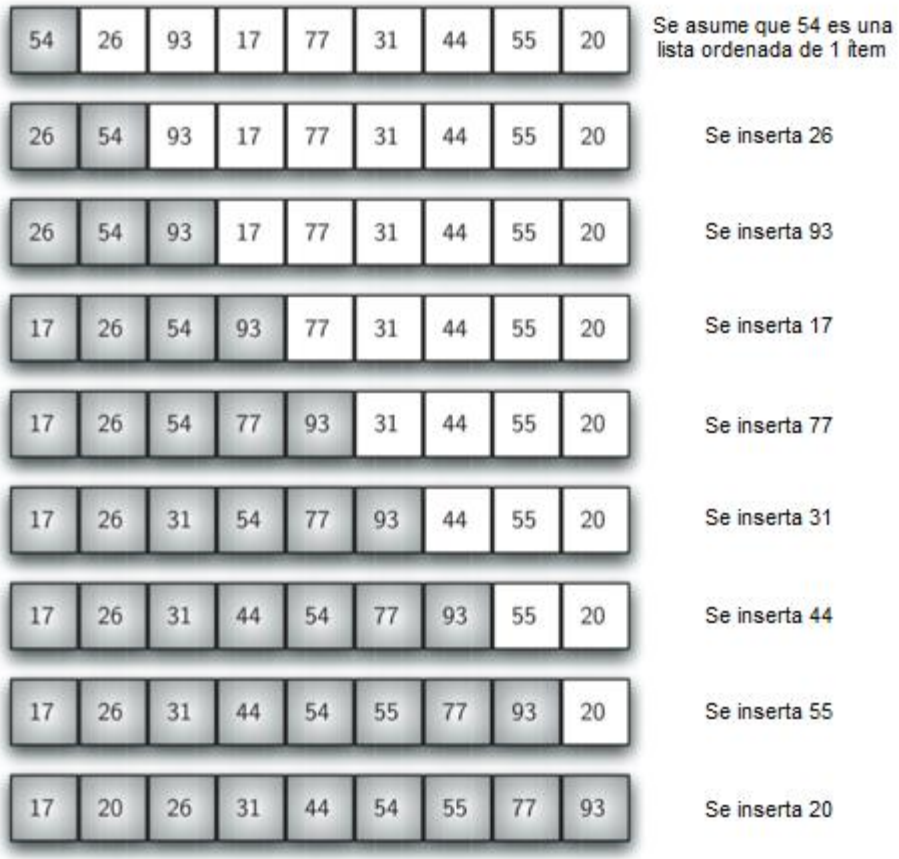
burbuja


First pass
54 26 93 17 77 31 44 55 20
Exchange
26 54 93 17 77 31 44 55 20
No Exchange
26 54 93 17 77 31 44 55 20
Exchange
26 54 17 93 77 31 44 55 20
Exchange
26 54 17 77 93 31 44 55 20
Exchange
26 54 17 77 31 93 44 55 20
Exchange
26 54 17 77 31 44 93 55 20
Exchange
26 54 17 77 31 44 55 93 20
Exchange
26 54 17 77 31 44 55 20 93
93 in place after first pass

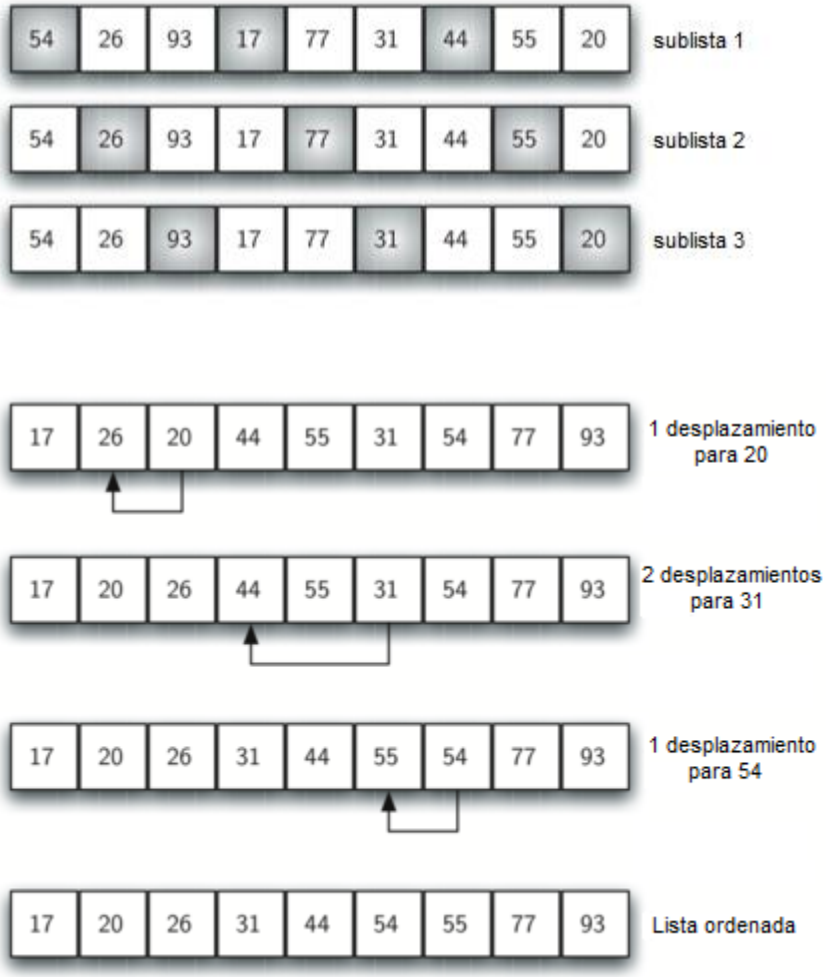
Selección


26 54 93 17 77 31 44 55 20
93 is largest
26 54 20 17 77 31 44 55 93
77 is largest
26 54 20 17 55 31 44 77 93
55 is largest
26 54 20 17 44 31 55 77 93
54 is largest
26 31 20 17 44 54 55 77 93
44 is largest
stays in place
26 31 20 17 44 54 55 77 93
31 is largest
26 17 20 31 44 54 55 77 93
26 is largest
20 17 26 31 44 54 55 77 93
20 is largest
17 20 26 31 44 54 55 77 93
17 ok
list is sorted

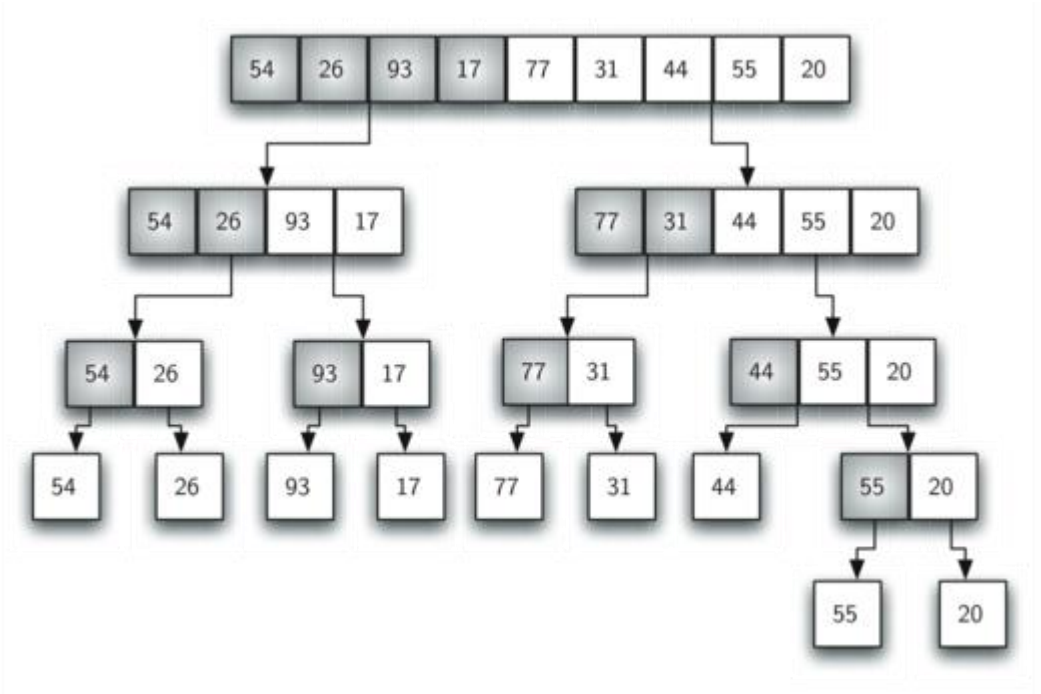
Inserción


54 26 93 17 77 31 44 55 20
Se asume que 54 es una lista ordenada de 1 ítem
26 54 93 17 77 31 44 55 20
Se inserta 26
26 54 93 17 77 31 44 55 20
Se inserta 93
17 26 54 93 77 31 44 55 20
Se inserta 17
17 26 54 77 93 31 44 55 20
Se inserta 77
17 26 31 54 77 93 44 55 20
Se inserta 31
17 26 31 44 54 77 93 55 20
Se inserta 44
17 26 31 44 54 55 77 93 20
Se inserta 55
17 20 26 31 44 54 55 77 93
Se inserta 20

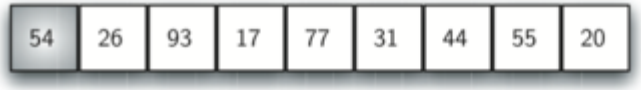
Shell


54 26 93 17 77 31 44 55 20
sublista 1
54 26 93 17 77 31 44 55 20
sublista 2
54 26 93 17 77 31 44 55 20
sublista 3
17 26 20 44 55 31 54 77 93
1 desplazamiento para 20
17 20 26 44 55 31 54 77 93
2 desplazamientos para 31
17 20 26 31 44 55 54 77 93
1 desplazamiento para 54
17 20 26 31 44 54 55 77 93
Lista ordenada

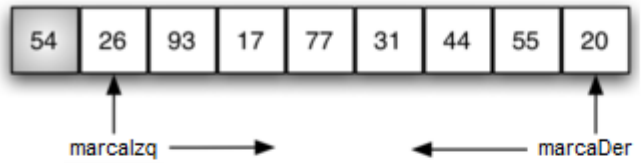
Mezcla


54 26 93 17 77 31 44 55 20
54 26 93 17
77 31 44 55 20
54 26
93 17
77 31
44 55 20
54
26
93
17
77
31
44
55 20
55
20

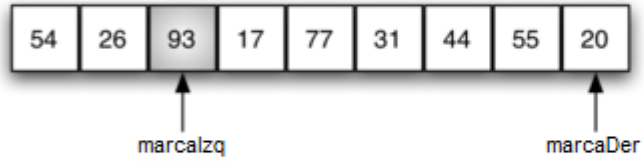
Rápida



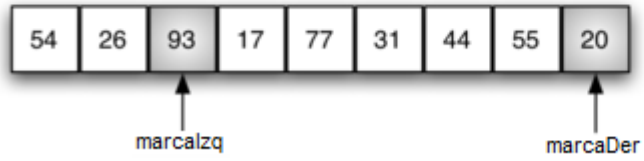
54 será el p
valor pivo



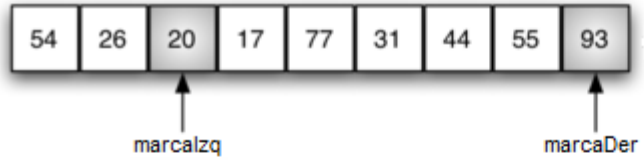
marcalzq y marcaDer convergerán en el punto de división



26<54 mover a la derecha
93>54 parar

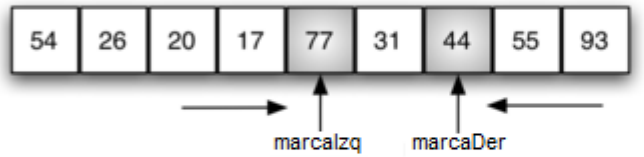


ahora marcaDer
20<54 parar

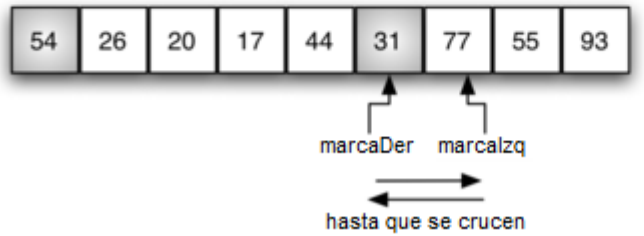


intercambiar 20 y 93

ahora continuar moviendo marcalzq y marcaDer



77>54 parar
44<54 parar
intercambiar 77 y 44



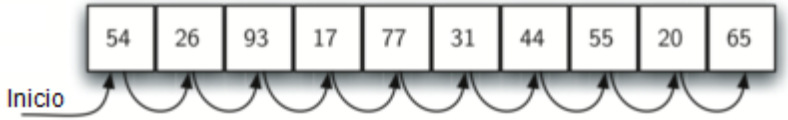
77>54 parar
31<54 parar
marcaDer<marcalzq
punto de división encontrado
intercambiar 54 y 31

Búsqueda

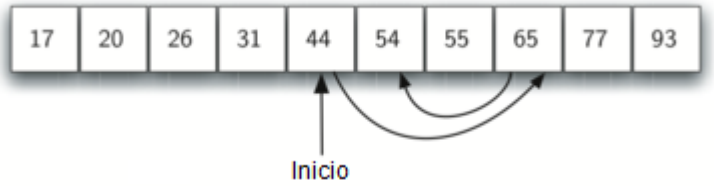
La búsqueda es el proceso algorítmico de encontrar un ítem particular en una colección de ítems.

Tipo de Búsqueda

- **Búsqueda Secuencial**



  - El arreglo debe estar ordenado.
  - Luego se va recorriendo el arreglo secuencialmente hasta encontrar el numero
- **Búsqueda Binaria**
  - Se comienza la búsqueda en el medio y se consulta si el numero
  - Se continua en función al resultado dela misma manera



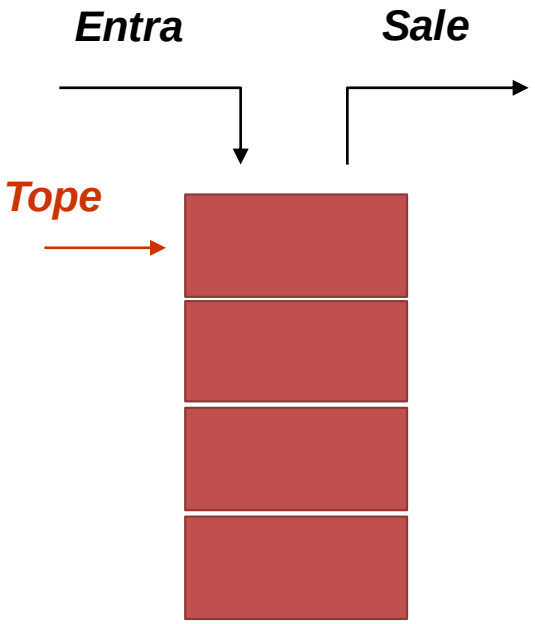
3.3 Pilas y Colas

PILAS

Definición:

Estructura de datos lineal donde los elementos pueden ser añadidos o removidos solo por un extremo.

Trabajan con filosofía LIFO (Last In-First Out).



Ejemplos:

- Pila de platos
- Pila de discos
- Pila de llamadas a funciones
- Pila de recursion
- Pila de resultados parciales de formulas aritméticas, etc.

Operaciones básicas con pilas

-**PUSH (insertar).-** Agrega un elementos a la pila en el extremo llamado **tope**.

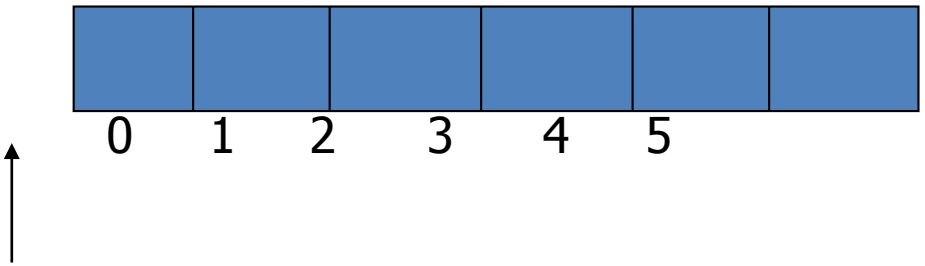
-**POP (remover).-** Remueve el elemento de la pila que se encuentra en el extremo llamado **tope**.

-**VACIA.-** Indica si la pila contiene o no contiene elementos.

-**LLENA.-** Indica si es posible o no agregar nuevos elementos a la pila.

Representación de pilas usando arreglos

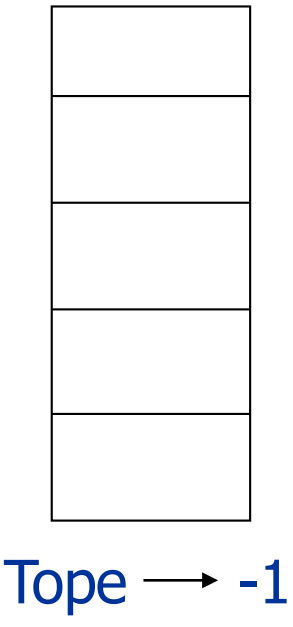
- Define un arreglo de una dimensión (vector) donde se almacenan los elementos.



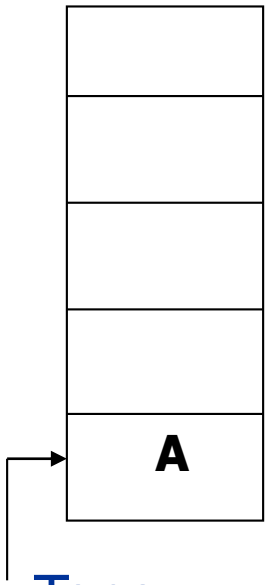
TOPE: Apunta hacia el elemento que se encuentra en el extremo de la pila. (inicialmente es -1).

Ejemplo

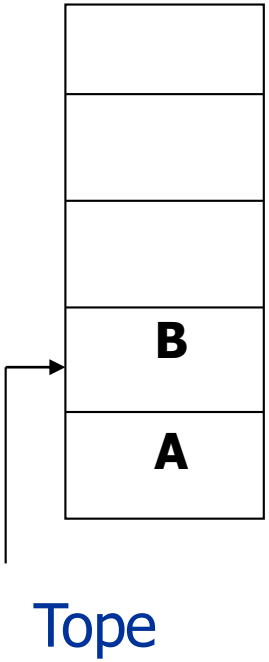
Inicio:



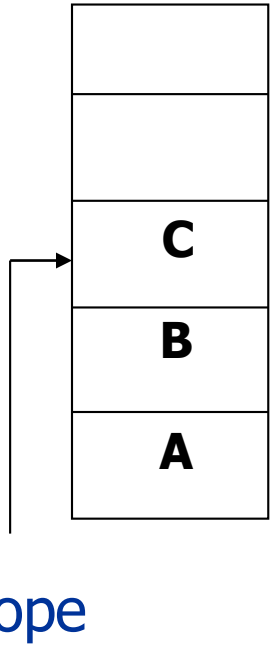
Insertar A:



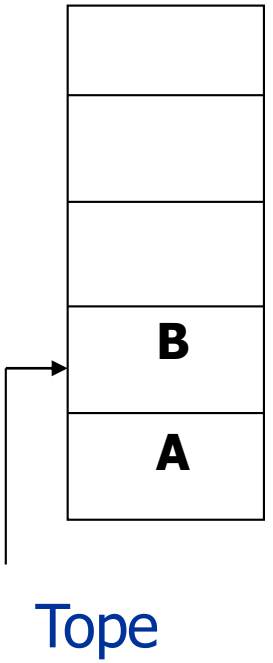
Insertar B:



Insertar C:



Eliminar elemento



Implementación con arreglos

- Una pila es una colección ordenada de objetos.
- En C# permiten almacenar colecciones ordenadas.
- La desventaja de implementar una pila mediante un arreglo es que esta última es de tamaño fijo, mientras que la pila es de tamaño dinámico.

Las operaciones básicas

```
{ class pila
    {
        public const int Maximo = 10;
        private int[] vector = new int[Maximo];
        private int tope = -1;
```

PUSH

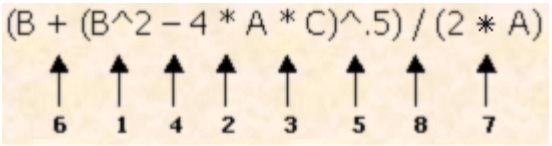
```
public void push(int valor)
{
    if (!LLena())
    {
        tope++;
        vector[tope] = valor;
    }
    else
        Console.WriteLine("La Pila esta llena");
}
```

POP

```
public int pop()
{
    if (!Vacia())
    {
        tope--;
        return vector[tope + 1];
            }
    else
        return -1;
}
```

Aplicaciones de las pilas

- Navegador Web
  - Se almacenan los sitios previamente visitados
  - Cuando el usuario quiere regresar (presiona el botón de retroceso), simplemente se extrae la última dirección (*pop*) de la pila de sitios visitados.
- Editores de texto
  - Los cambios efectuados se almacenan en una pila
  - Usualmente implementada como arreglo
  - Usuario puede deshacer los cambios mediante la operación "*undo*", la cual extraer el estado del texto antes del último cambio realizado
- Expresiones Aritméticas
  - Una expresión aritmética contiene constantes, variables y operaciones con distintos niveles de precedencia.



Aplicaciones de Pilas

- Funciones Recursivas
  - Las pilas pueden ser usadas para implementar la recursión en programas.
  - Una función o procedimiento recursivo es aquel que se llama a si mismo.
    - Factorial
    - Números de Fibonacci
    - Torres de Hanoi
    - Algoritmos de Ordenamiento de datos
  - Control de secuencia de programas.
    - Controla la Llamada a Subprogramas

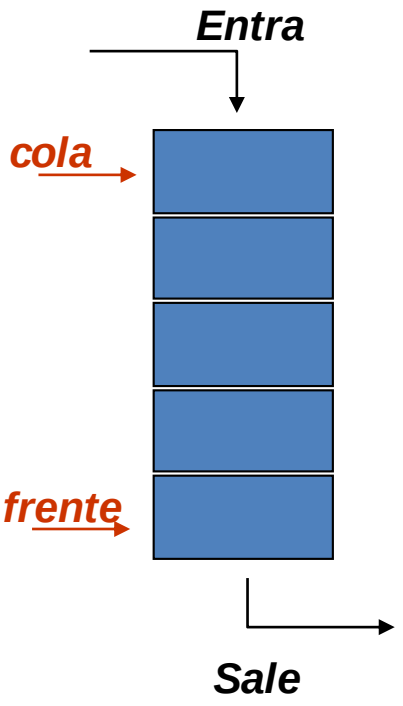
Ejemplos de expresiones

entrefijo	prefijo	posfijo
a+b	+ab	ab+
(a+b)*c	*+abc	ab+c*
(a-b)*(c-d)	*-ab-cd	ab-cd-*
((a+b)*c-(d-e))^(f+g)	^-*+abc-de+fg	ab+c*de--fg+^

Colas

Definicion. Es una lista lineal de elementos en la que las operaciones de insertar y eliminar se realizan en diferentes extremos de la cola.

Trabajan con filosofía FIFO (First In - First out), el primer elemento en entrar es el primer elemento en salir.



Ejemplos:

- Cola de automóviles esperando servicio en una gasolinera
- Cola de clientes en una ventanilla del banco para pagar un servicio
- Cola de programas en espera de ser ejecutados por una computadora.

Operaciones de una COLA

Insertar.- Almacena al final de la cola el elemento que se recibe como paramétro.

Eliminar.- Saca de la cola el elemento que se encuentra al frente.

Vacía.- Regresa un valor booleano indicando si la cola tiene o no elementos (true – si la cola esta vacia, false – si la cola tiene al menos un elemento).

Llena.- Regresa un valor booleano indicando si la cola tiene espacio disponible para insertar nuevos elementos (**true** – si esta llena y **false** si existen espacios disponibles).

Operaciones:

1.- Insertar A

2.- Insertar B

3.- Insertar C

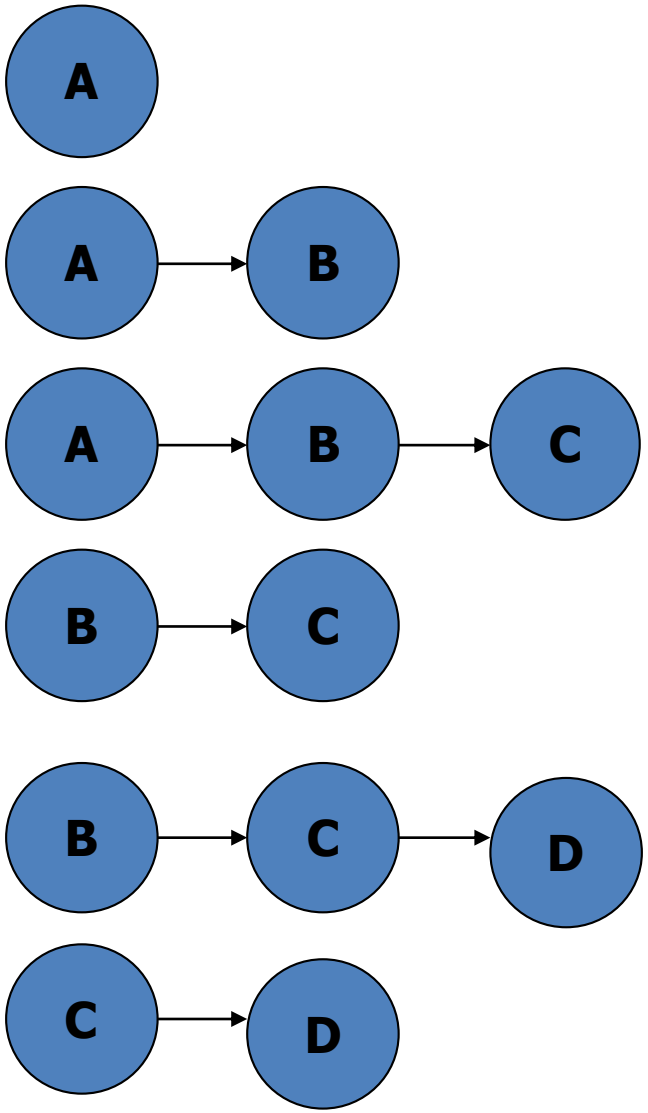
4.- Remover Elemento

5.- Insertar D

6.- Remover Elemento

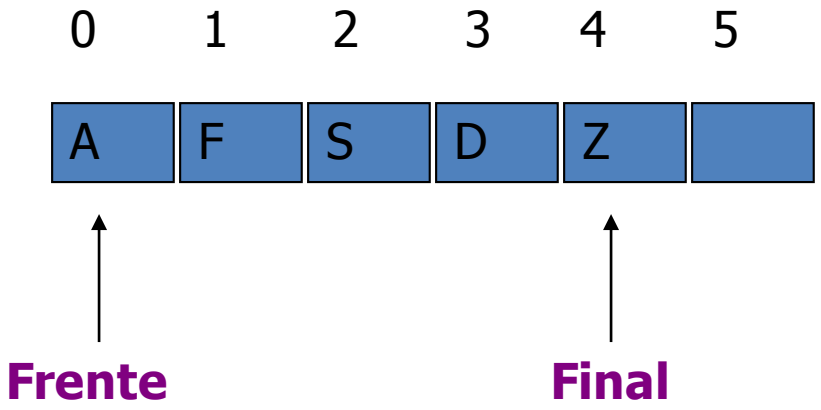
Estado de la cola:

Inicio: Cola Vacía



Representación usando arreglos

Las colas pueden ser representadas en arreglos de una dimensión (vector) manteniendo dos variables que indiquen el FRENTE y FINAL de los elementos de la cola.



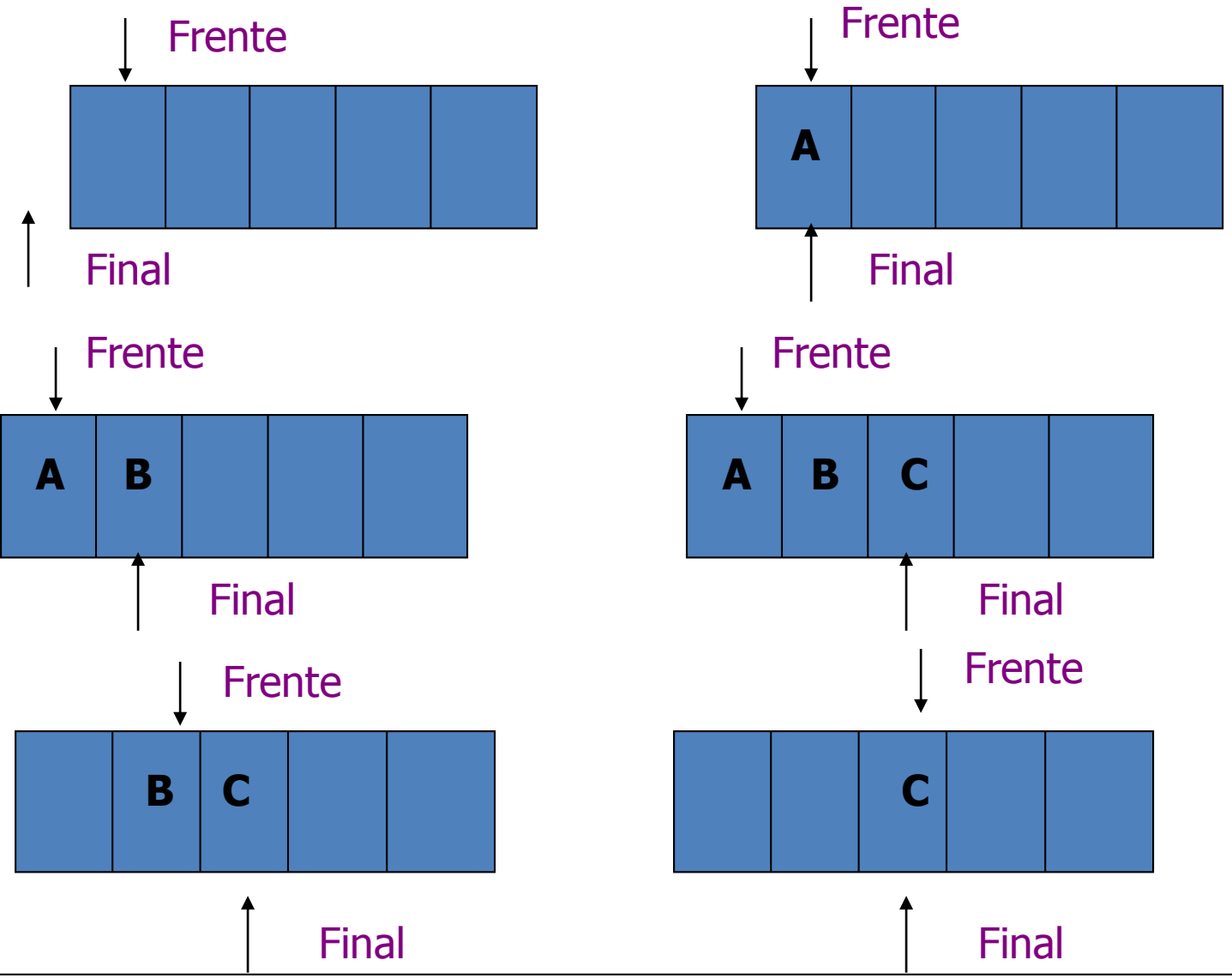
- Cuando la cola esta vacía el **frente > final** y no es posible remover elementos.

- Cuando la cola esta llena (**final** = maximo tamaño) no es posible insertar elementos nuevos a la cola.

- Cuando se remueven elementos el **frente** puede incrementarse para apuntar al siguiente elemento de la cola (*implementación con frente móvil*).

Ejemplo: Suponer que usamos un arreglo de 5 posiciones.
Usando la representación de frente fijo y frente movible.

Insertar:



Eliminar:

Implementación con arreglos

- Una cola es una colección ordenada de objetos.
- En c#, los arreglos permiten almacenar colecciones ordenadas.
- Misma desventaja: los arreglos tienen tamaño fijo.
- Uso eficiente mediante un arreglo circular.

Las operaciones básicas

```
class Cola<T>
{
        public const int Maximo = 10;
        private T[] arreglo = new T[Maximo];
        private int final = -1;
        private int frente = 0;
```

```
public void Insertar(T valor)
    {
        if (!LLena())
        {
            final++;
            arreglo[final] = valor;
        }
        else
            Console.WriteLine("La Cola esta llena");
    }
```

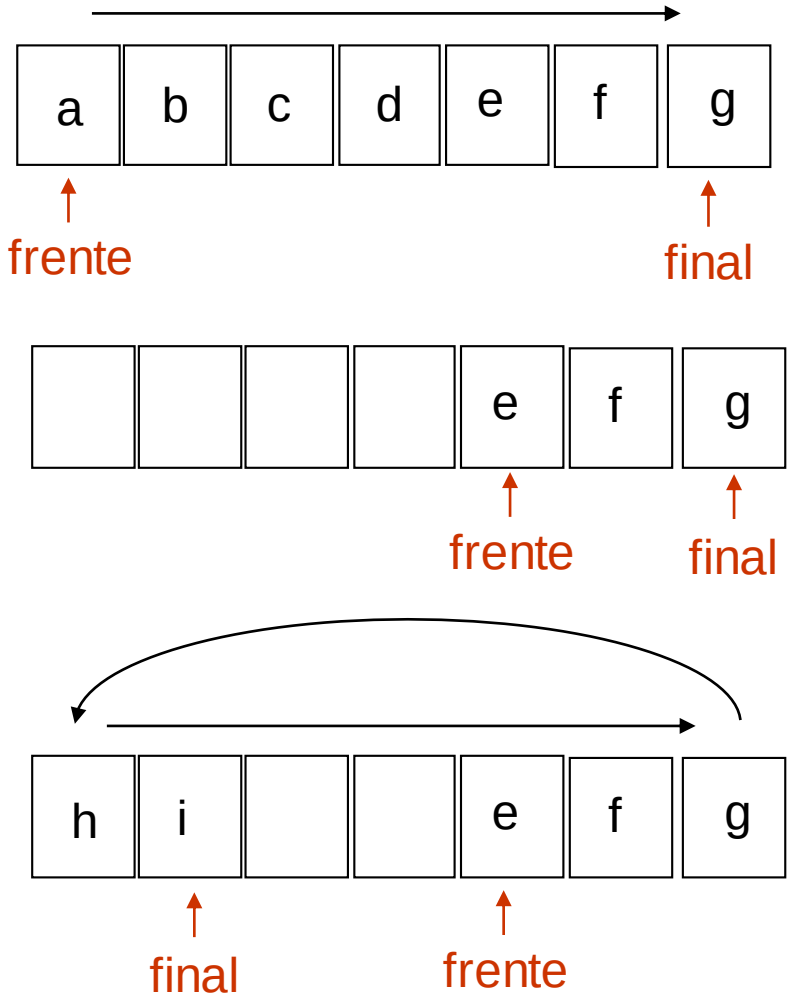
```
public T Eliminar()
{
    if (!Vacia())
    {
        frente++;
        return arreglo[frente-1];
    }
    else
    {
        Console.WriteLine("La Cola esta Vacia");
        return arreglo[frente];
    }
}
```

Tipos de colas

- **Cola simple:** Estructura lineal donde los elementos salen en el mismo orden en que llegan.
- **Cola circular:** Representación lógica de una cola simple en un arreglo.
- **Cola de Prioridades:** Estructura lineal en la cual los elementos se insertan en cualquier posición de la cola y se remueven solamente por el frente.
- **Cola Doble (Bicola):** Estructura lineal en la que los elementos se pueden añadir o quitar por cualquier extremo de la cola (cola bidireccional).

Colas circulares

- El objetivo de una cola circular es aprovechar al máximo el espacio del arreglo.
- La idea es insertar elementos en las localidades previamente desocupadas.
- La implementación tradicional considera dejar un espacio entre el frente y la cola.



Decolas o Bicolas

- **Las *Decolas* son casos particulares de listas y generalizaciones de colas en donde las eliminaciones e inserciones pueden realizarse en ambos extremos de la lista.**
  - **Dicola de entrada restringida**
  - **Dicola de salida restringida**

Colas de Prioridad

Una cola de prioridad es una cola a cuyos elementos se les ha asignado una prioridad, de forma que el orden en que los elementos son procesados sigue las siguientes reglas:

- **El elemento con mayor prioridad es procesado primero.**
- **Dos elementos con la misma prioridad son procesados según el orden en que fueron introducidos en la cola.**

Colas de Prioridad

Existen dos métodos básicos para la representación de colas de prioridad mediante estructuras lineales:

a) **Tener la cola siempre ordenada de acuerdo a las prioridades de sus elementos, y sacar cada vez el primer elemento de ésta, es decir, el de mayor prioridad. En este caso, cuando se introduce un elemento en la cola, debe insertarse en el lugar correspondiente de acuerdo a su prioridad.**

b) **Insertar los elementos siempre al final de la cola, y cuando se va a sacar un elemento, buscar el que tiene mayor prioridad.**

Aplicaciones de las colas

- En general, operaciones en redes de computadoras
  - Trabajos enviados a una impresora
  - Solicitudes a un servidor.
- Clientes solicitando ser atendidos por una telefonista
- Simulaciones de cualquier situación real en la que se presente una "organización" tipo cola

3.4. Recursividad

La recursividad consiste en resolver un problema a partir de casos más simples del mismo problema.

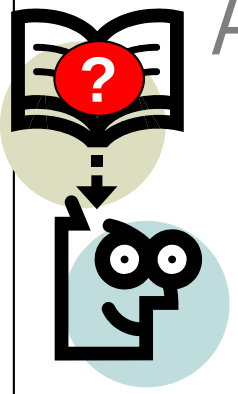
Una función recursiva es aquella que se "llama a ella misma"

Ejemplo: Factorial

- El factorial de 1 es 1:
  - $1! = 1$
- Y el factorial de un número arbitrario es el producto de ese número por los que le siguen, hasta llegar a uno:
  - $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$
- Podemos escribir el factorial de un número a partir del factorial del siguiente número:
  - $n! = n \cdot (n-1)!$

Listas Enlazadas

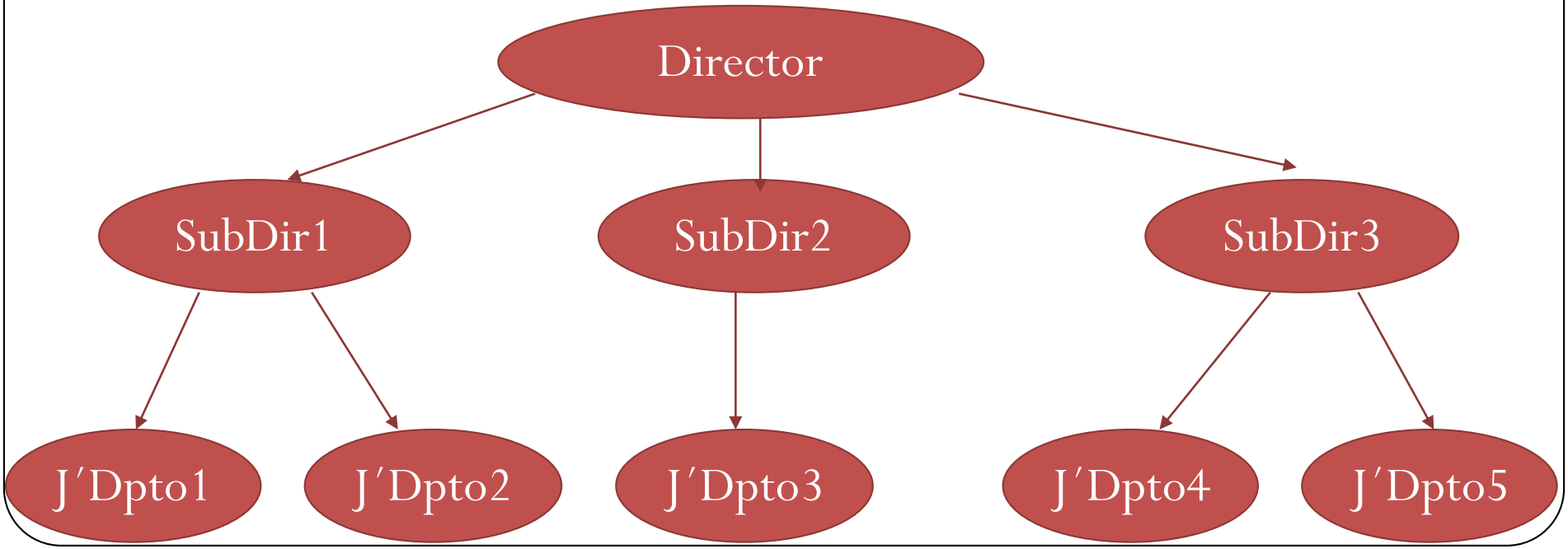
3.5 Arboles y Grafos



Arboles

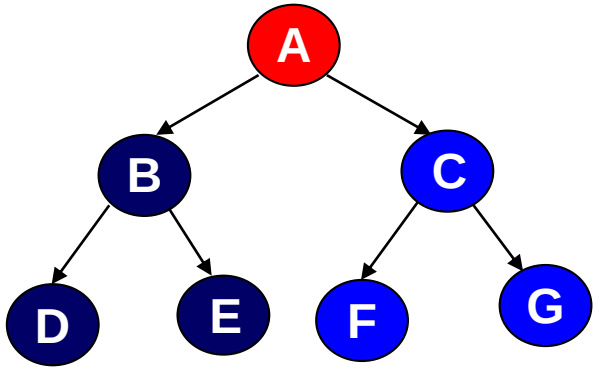
- ¿Qué estructura de datos se debe utilizar para representar estructuras jerárquicas o taxonómicas?

Ejemplo:

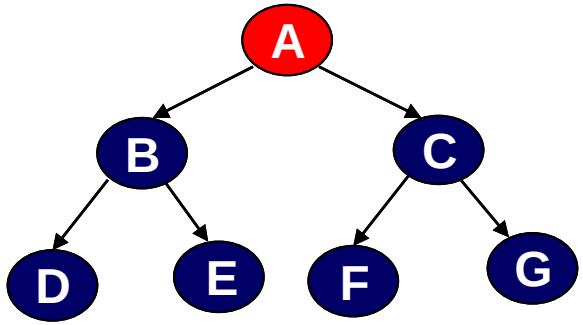


Definición de Árbol

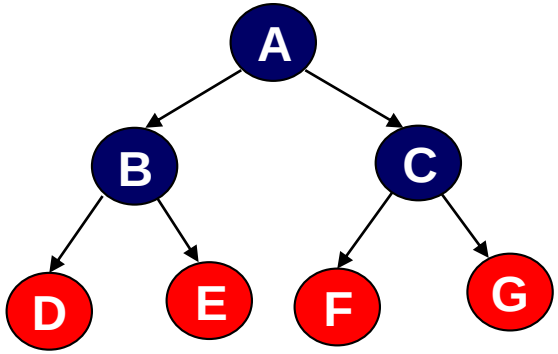
Un **árbol** (tree) es una estructura que consta de un conjunto finito *T* de nodos y una relación *R* (paternidad) entre los nodos tal que:



Hay un nodo, especialmente designado, llamado la ***raíz*** del árbol T.



Los nodos restantes, excluyendo la raíz, son particionados en m ($m \geq 0$) conjuntos disjuntos T1, T2, ..., Tm, cada uno de los cuales es, a su vez, un árbol, llamado ***subárbol*** de la raíz del árbol T.



A los nodos que no son raíces de otros subárboles se les denomina ***hojas*** del árbol T, o sea, no tienen **sucesores** o **hijos**.

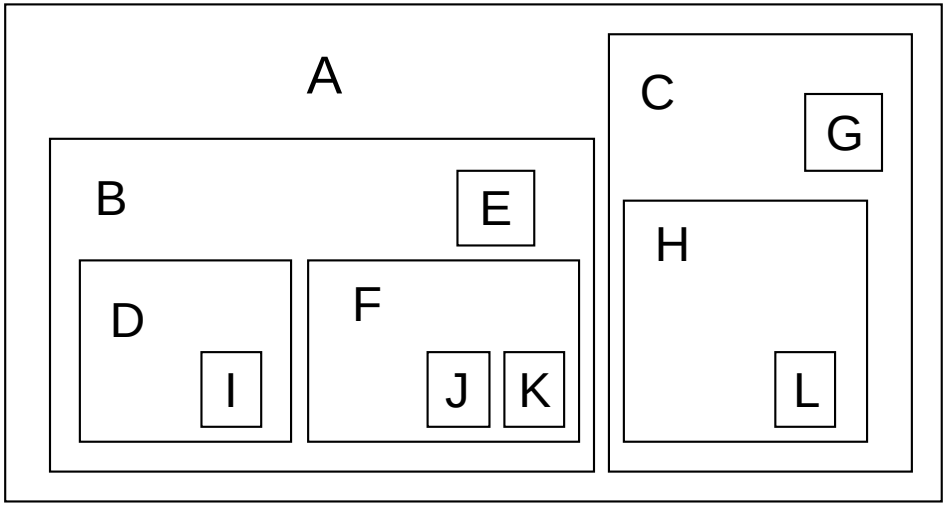
Definición de Árbol

- Si *n* es un nodo y A_1, A_2, A_3, A_4, A_5, …, A_k son árboles con raíces n_1, n_2, n_3, n_4,…, n_k . Se puede construir un nuevo árbol haciendo que *n* se constituya en ***padre*** de los nodos n_1, n_2, n_3, n_4,…, n_k.

- En dicho árbol, n es la raíz y A_1, A_2, A_3, A_4, A_5, …, A_k son los ***subárboles*** de la raíz.

- Los nodos n_1, n_2, n_3, n_4,…, n_k reciben el nombre de ***hijos*** del nodo n.

FORMAS DE REPRESENTACION DE UN ÁRBOL



Diagramas de Venn

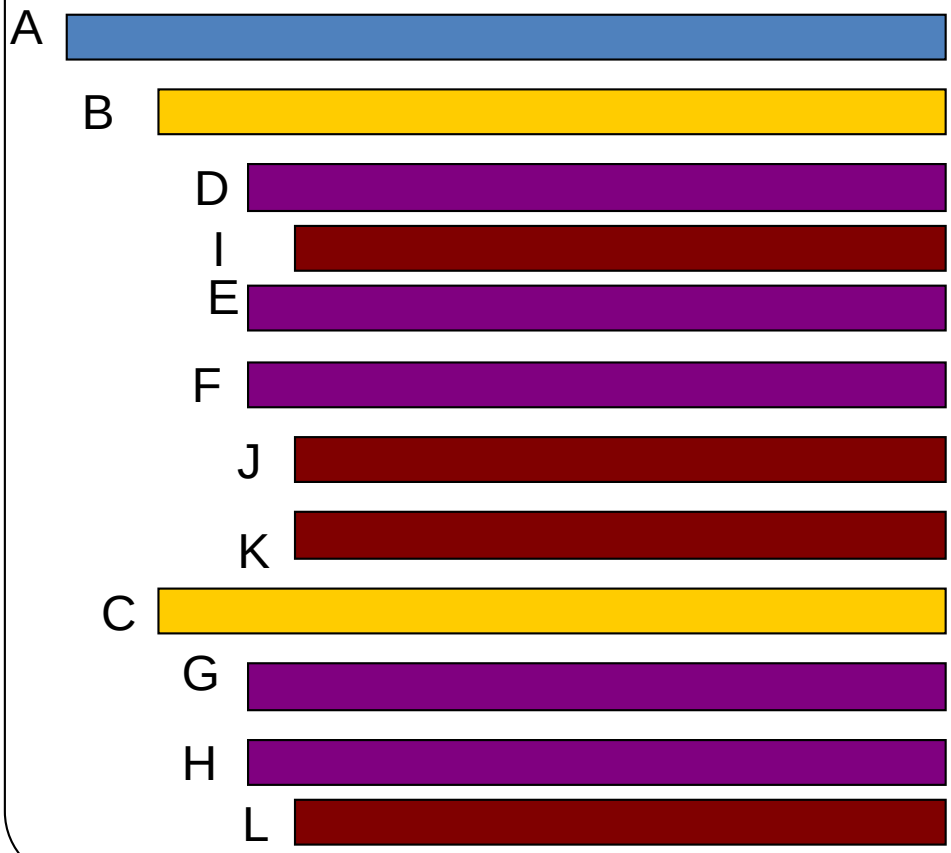
(A (B (D (I), E, F (J, K)), C (G, H (L))))

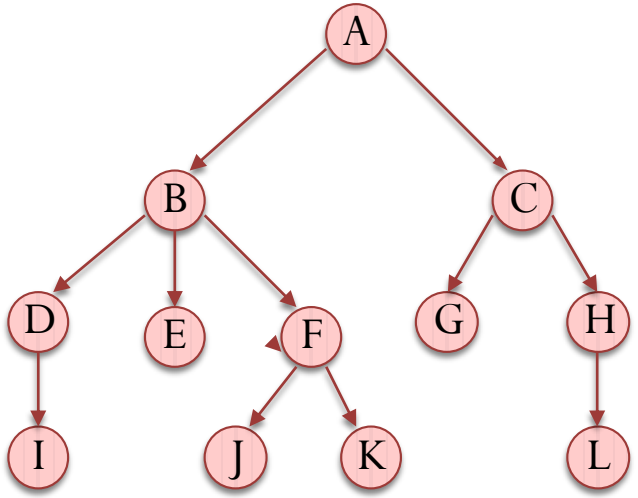
Anidación de paréntesis

NOTACION DECIMAL DEWEY

1.A, 1.1.B, 1.1.1.D, 1.1.1.1.I, 1.1.2.E, 1.1.3.F, 1.1.3.1.J, 1.1.3.2.K
1.2.C, 1.2.1.G, 1.2.2.H, 1.2.2.1.L

NOTACION IDENTADA



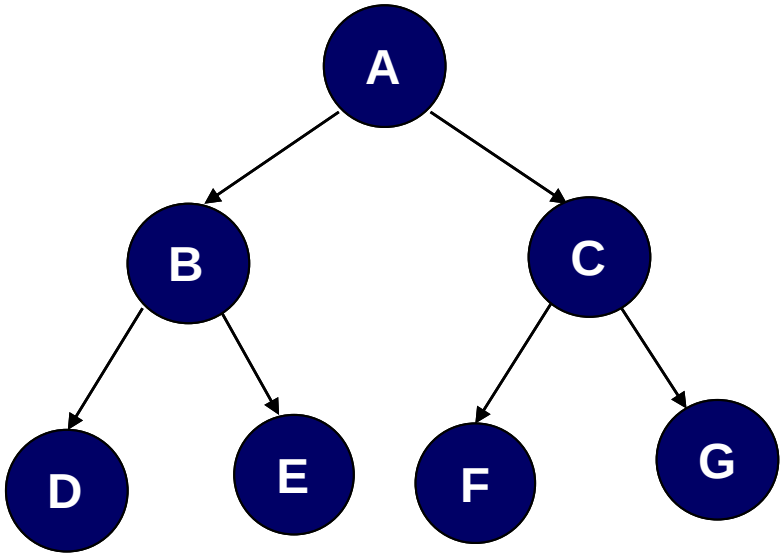


Grafo

Aclaraciones

- Si el conjunto finito T de nodos del árbol es vacío, entonces se trata de un ***árbol vacío***.
- En esta estructura existe **sólo un nodo** sin padre, que es la ***raíz*** del árbol.
- Todo nodo, a excepción del nodo raíz, tiene ***uno y sólo un padre***.
- Los subárboles de un nodo son llamados ***hijos***.

Ejemplos



Padre de C: A

Padre de E: B

Padre de G C

Padre de A:

Hijos de A: NO

Hijos de C: B C

Hijos de F: F G

NO

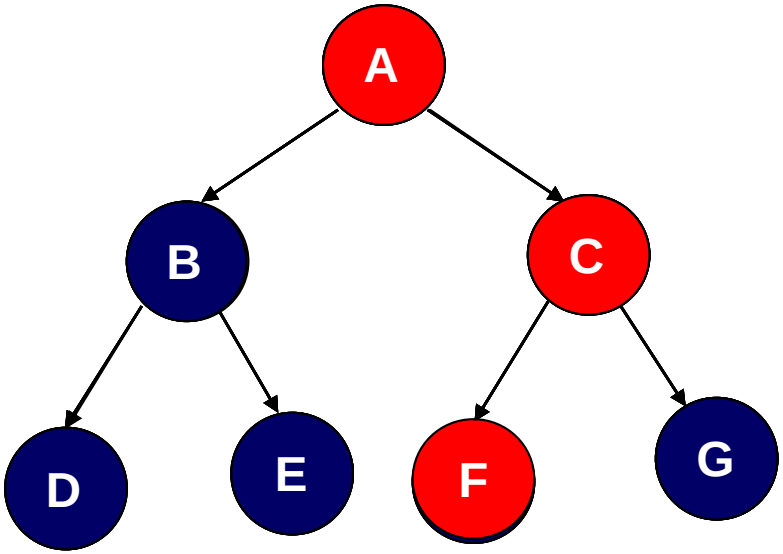
Aclaraciones

- Para todo nodo k, distinto de la raíz, existe una única secuencia de la forma:

  - $k_0, k_1, k_2, k_3, \ldots, k_n$, donde k_0=raíz y k_n=k

  - Con n >= 1, donde.
    - k_i es el sucesor de k_{i-1},
    - para 1 <= i <= n, o sea, cada nodo k_i de la
    - secuencia es la raíz de otro subárbol.

Ejemplos



Secuencias

de A a G

de A a E

de A a F

C es sucesor de A y

F es sucesor de C

Otras definiciones

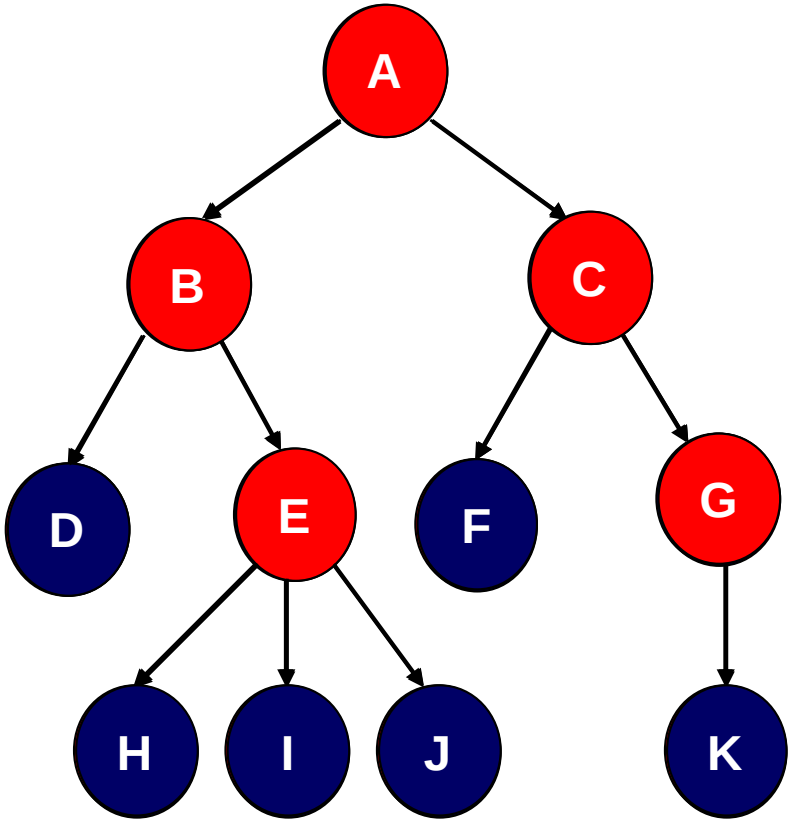
Grado de un nodo: cantidad de hijos de un nodo.

Grado de un árbol al mayor de los grados de todos sus nodos.

Nodo hoja a un nodo sin hijos o con grado = 0.

Nodo rama a un nodo que tiene hijos, o sea, a la raíz de un subárbol.

Ejemplos



Grado

de A: 2

de E: 2

de G: 3

de J: 3

Grado del árbol: 3

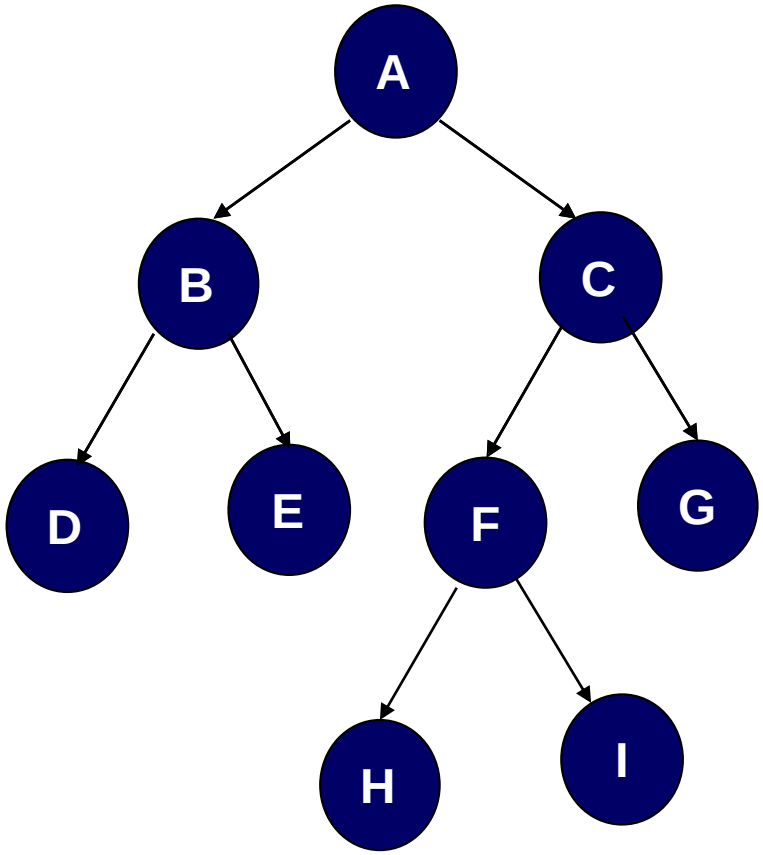
Nodos hojas: D, H, I, J, F, K

Nodos ramas: A, B, C, E, G

Otras definiciones

- ***Nivel de un nodo*** al nivel de su padre más uno. Por definición, la raíz del árbol tiene nivel 0. Esta definición es recursiva.

Ejemplos



Nivel

de A: 0

de E: 0

de B: 2

de I: 3

de G: 1

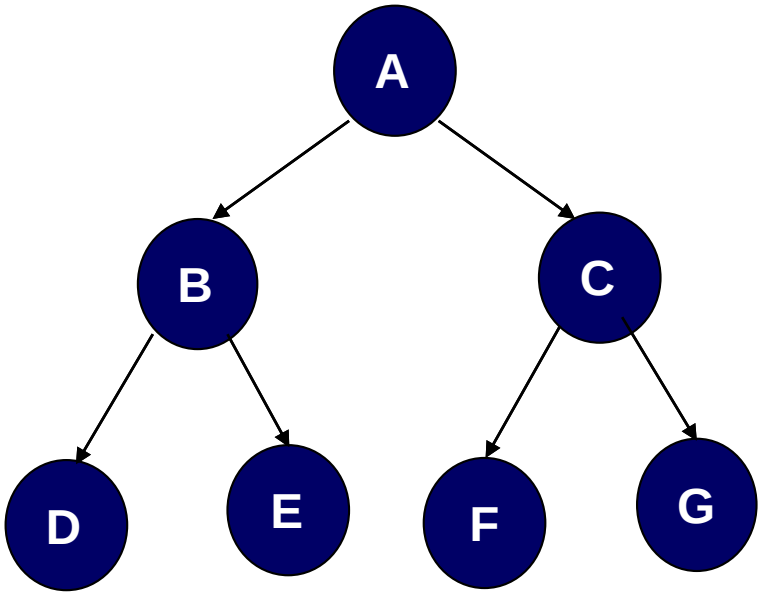
de H: 3

de I: 2

Otras definiciones

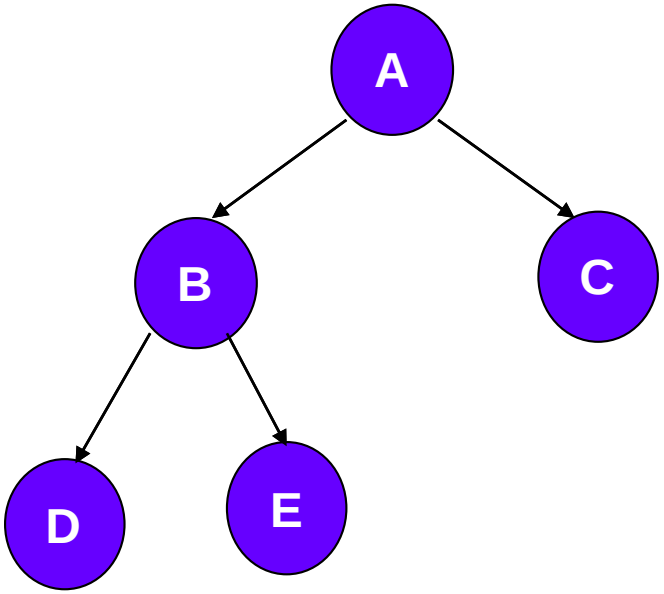
- ***Árbol completo de nivel n*** a un árbol en el que cada nodo de nivel n es una hoja y cada nodo de nivel menor que n tiene, al menos, un subárbol no vacío.

Ejemplos



Árbol completo de nivel 2

Cada nodo del nivel n es una hoja



Árbol *no completo* de nivel 2

Un nodo del nivel n-1 es una hoja

Otras definiciones

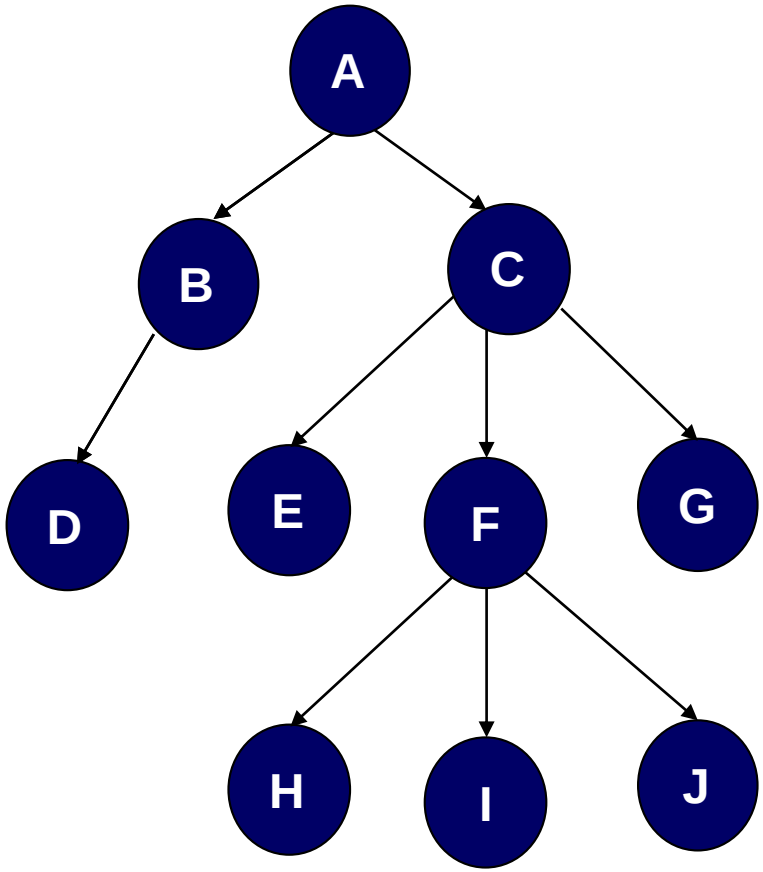
Padre de un nodo al nodo raíz del subárbol más pequeño que contiene a dicho nodo y en el cual él no es raíz.

Hijo de un nodo al (los) nodo(s) raíz(ces) de uno de sus subárboles.

Predecesor de un nodo al nodo que le antecede en un recorrido del árbol.

Hermano de un nodo a otro nodo hijo de su padre.

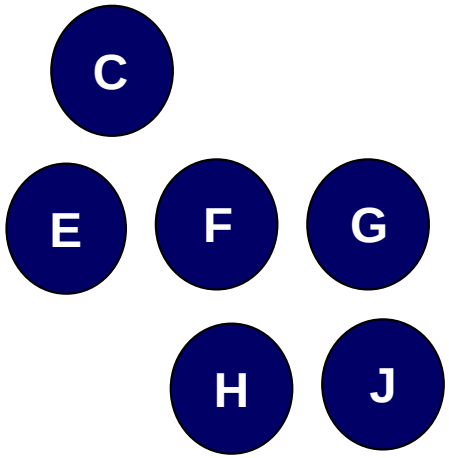
Ejemplos



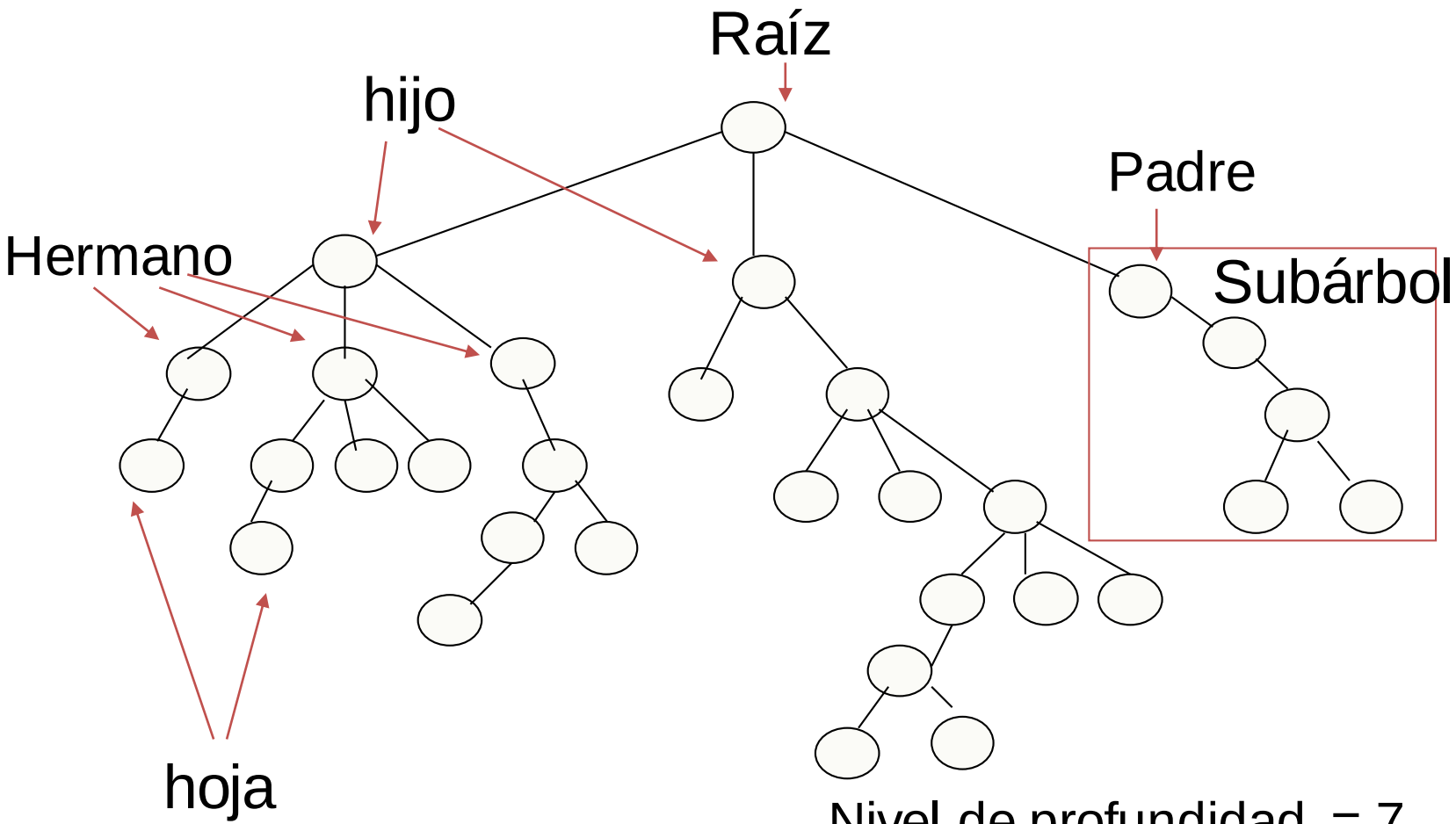
Padre de G:

Hijos de C:

Hermanos de I:



Conceptos Básicos (cont.)



Nivel de profundidad = 7

Grado de un nodo = 3

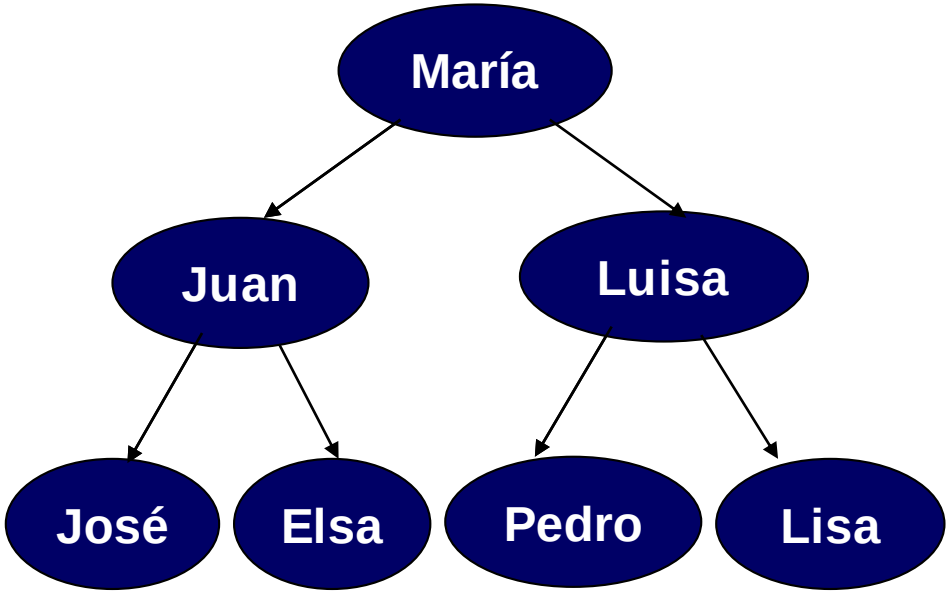
Grado del árbol = 3

Otras definiciones

Árbol ordenado a todo árbol para el que se considera el orden relativo de los sucesores o subárboles de cualquier nodo. Es decir, en un árbol ordenado se habla de primero, segundo o último hijo de un nodo en particular. El primer hijo de un nodo de un árbol ordenado es denominado el hijo mayor de ese nodo y el último hijo es denominado el menor.

*El **Árbol** es **ordenado*** si al intercambiar el orden relativo de los subárboles de un nodo, representa una situación semánticamente diferente.

Ejemplos: Árbol genealógico de María (sin los hermanos)



El árbol es ordenado

El primer subárbol corresponde al padre.

El segundo subárbol a la madre.

Otras definiciones

Árbol orientado a un árbol para el cual no interesa el orden relativo de los sucesores o subárboles de cualquier nodo, ya que sólo se tiene en cuenta la orientación de los nodos.

Ejemplo:

La estructura organizativa de una empresa, donde no es importante el orden de los subdirectores a la hora de representarlos en el árbol.

En la solución de problemas informáticos, los más utilizados son los árboles ordenados.

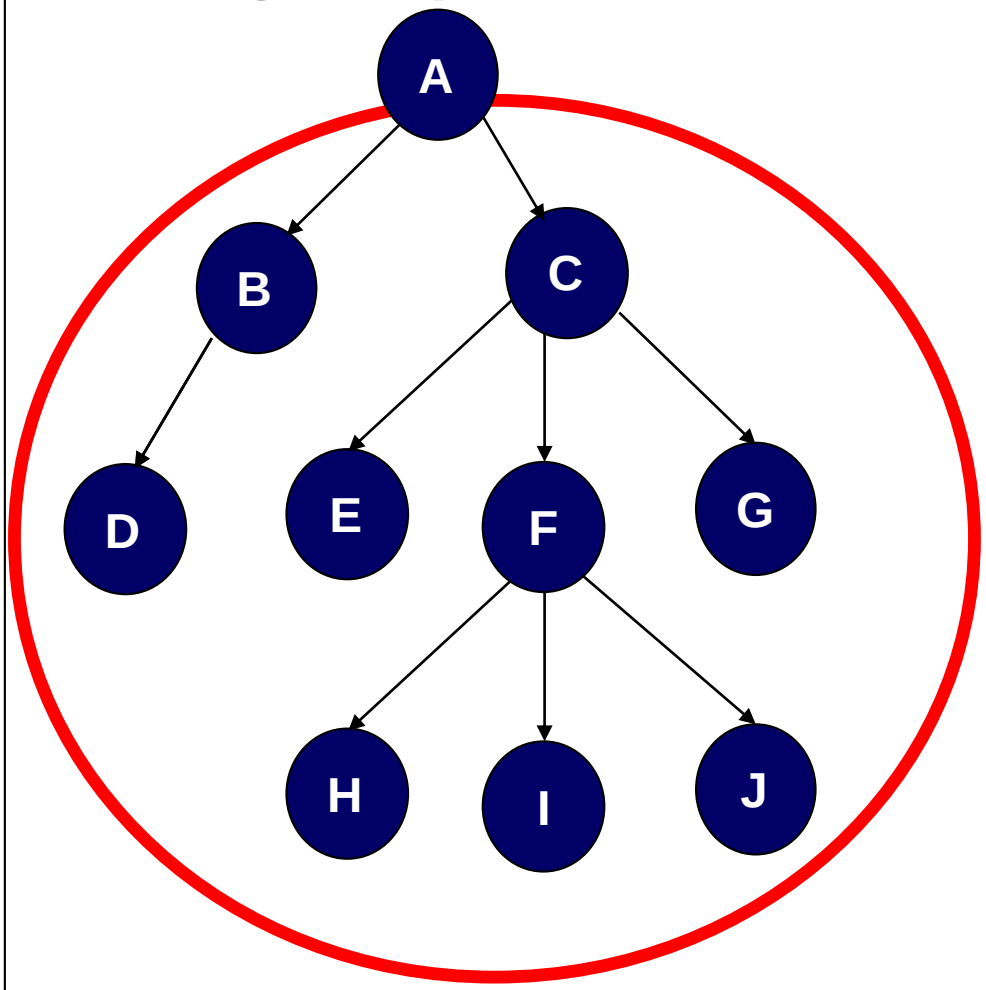
Otras definiciones

Una ***floresta*** es una colección de dos o más árboles disjuntos.

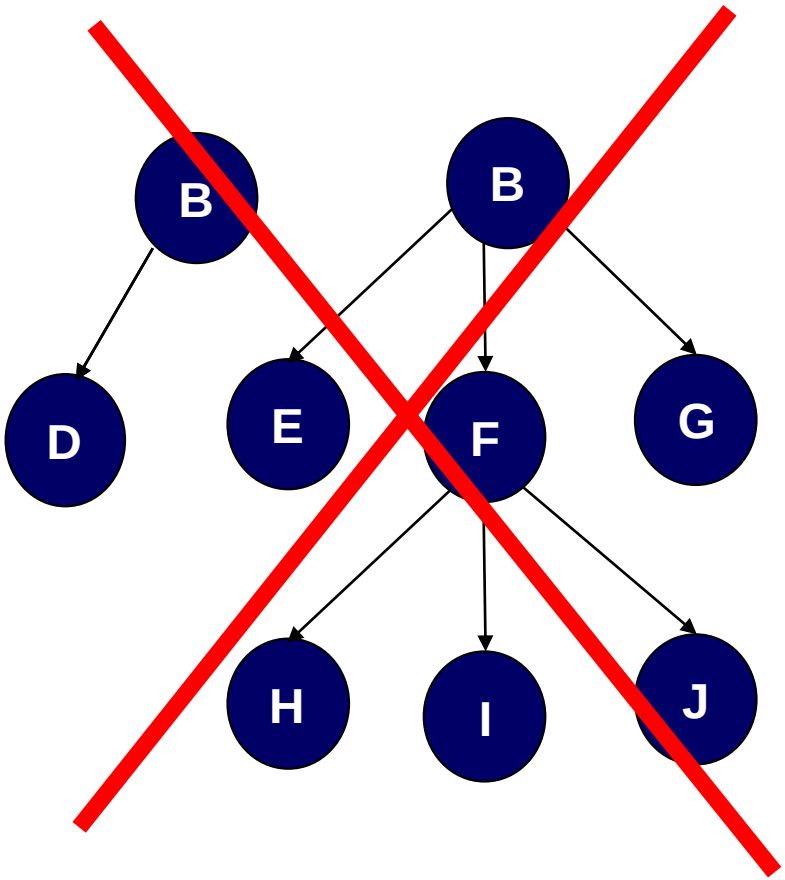
Aclaraciones:

- Disjuntos significa que no hay nodos en común entre dos árboles cualesquiera de la misma.
- De un árbol se obtiene una floresta al quitarle la raíz, si tiene dos hijos o más.
- De una floresta se obtiene un árbol al añadir un nodo que sea raíz de todos los árboles que la conforman.

Ejemplos



Es un árbol

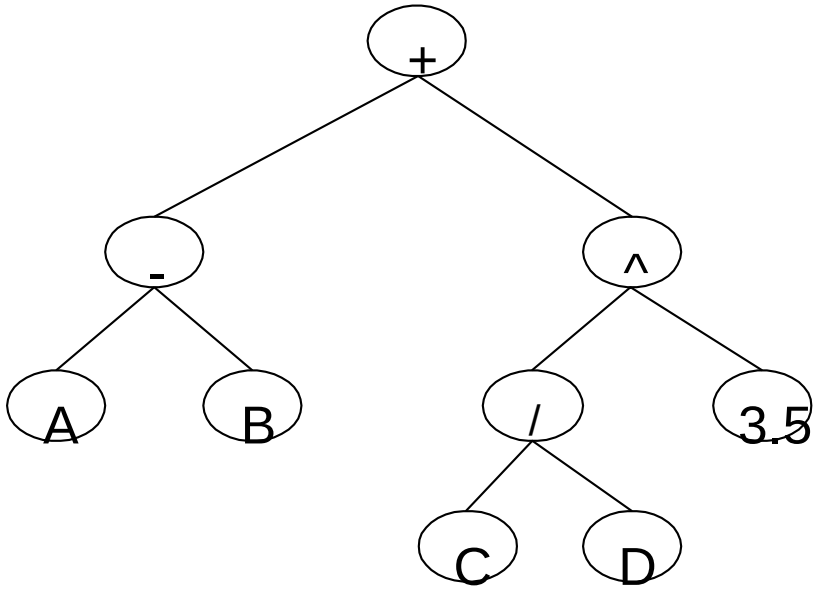


NO es una floresta

Tipos de árboles

Un árbol ordenado: Es aquel en el que las ramas de los nodos están ordenadas.

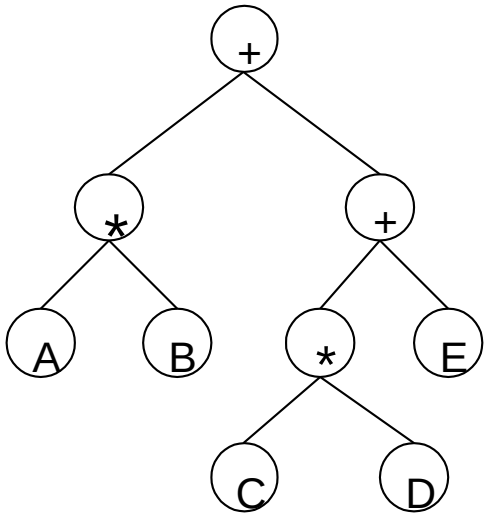
- Los de grado 2 se llaman árboles binarios.
- Cada árbol binario tiene un subárbol izquierda y subárbol derecha.



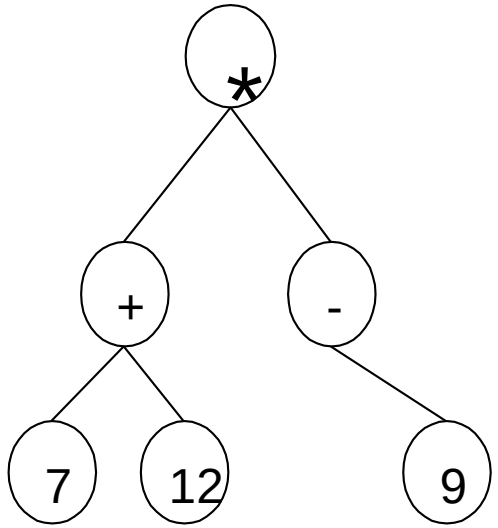
Tipos de árboles (cont.)

Árboles de expresión

- Representan un orden de ejecución



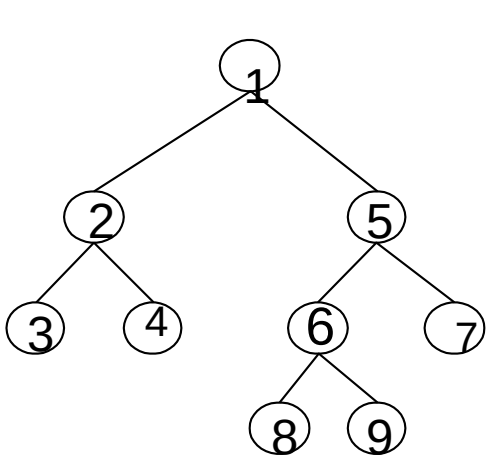
(A* B) + C * D + E

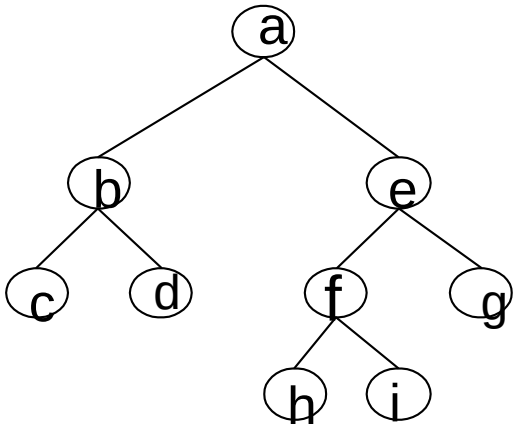


(7 + 12) * (-9) → -171

Tipos de árboles (cont.)

- ***Árboles similares***: Los que tienen la misma estructura (forma)



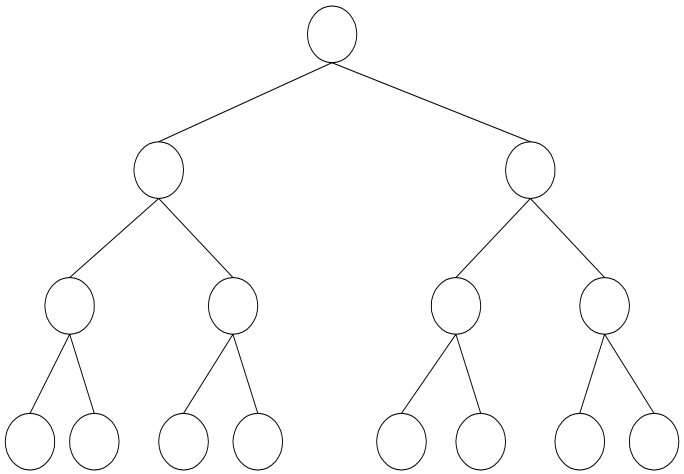


- ***Árboles Equivalentes:*** Son los árboles similares y sus nodos contienen la misma información.
- ***Árboles n-ario:*** Es un árbol ordenado cuyos nodos tiene N subárboles, y donde cualquier número de subárboles puede ser árboles vacíos

Tipos de árboles (cont.)

Árbol binario completo:

- Es un árbol en el que todos sus nodos, excepto los del ultimo nivel, tienen dos hijos.



- Número de nodos en un árbol binario completo = $2^h - 1$ (en el ejemplo h = 4, → 15) esto nos ayuda a calcular el nivel de árbol necesario para almacenar los datos de una aplicación.

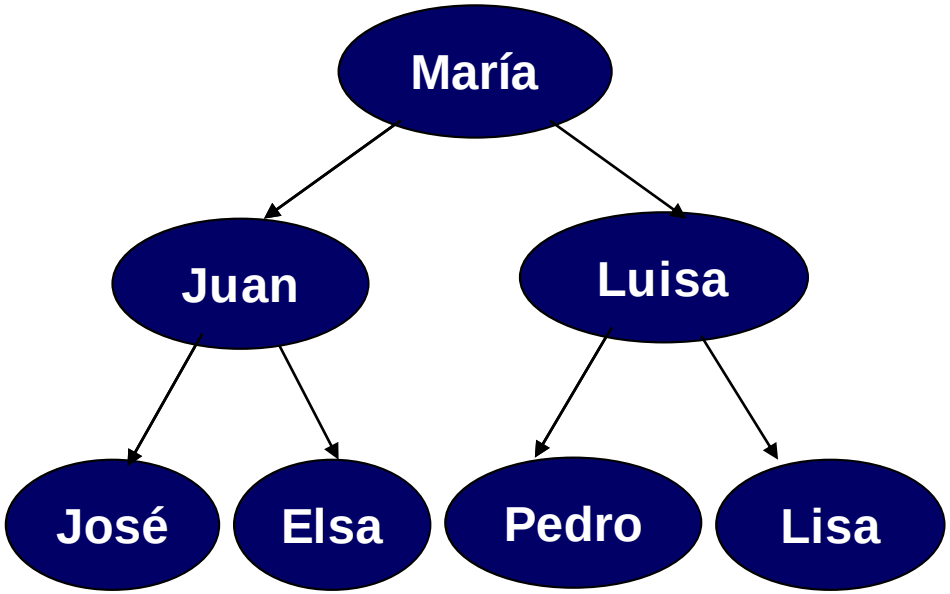
Definición de Árbol Binario

Un ***árbol binario*** (en inglés ***binary tree***) es un árbol ordenado de, a lo sumo, grado 2.

Aclaraciones:

- A lo sumo grado 2 significa que cada nodo tiene como máximo dos hijos, o sea, dos subárboles.
- Al ser ordenado el árbol, importa el orden de los subárboles, es decir, que será necesario especificar de cada nodo cuál es el hijo izquierdo y cuál el hijo derecho.

Ejemplo



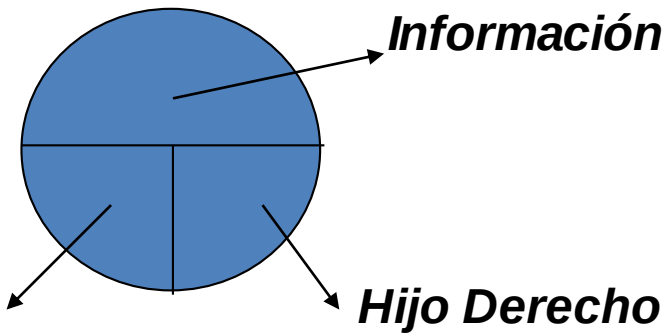
El árbol genealógico es un árbol binario.

Cada nodo tiene dos hijos

Es significativo el orden de los subárboles.

Árbol Binario: Características

- Cada nodo del árbol binario contiene:
- Una referencia a su información
- Un apuntador a su hijo izquierdo.
- Un apuntador a su hijo derecho.



Recorridos de un Árbol Binario

Los recorridos se clasifican de acuerdo al momento en que se visita la raíz del árbol y los subárboles izquierdo y derecho.

Existen tres recorridos:

- Recorrido en Preorden
- Recorrido en orden simétrico o inorden
- Recorrido en orden final o Postorden

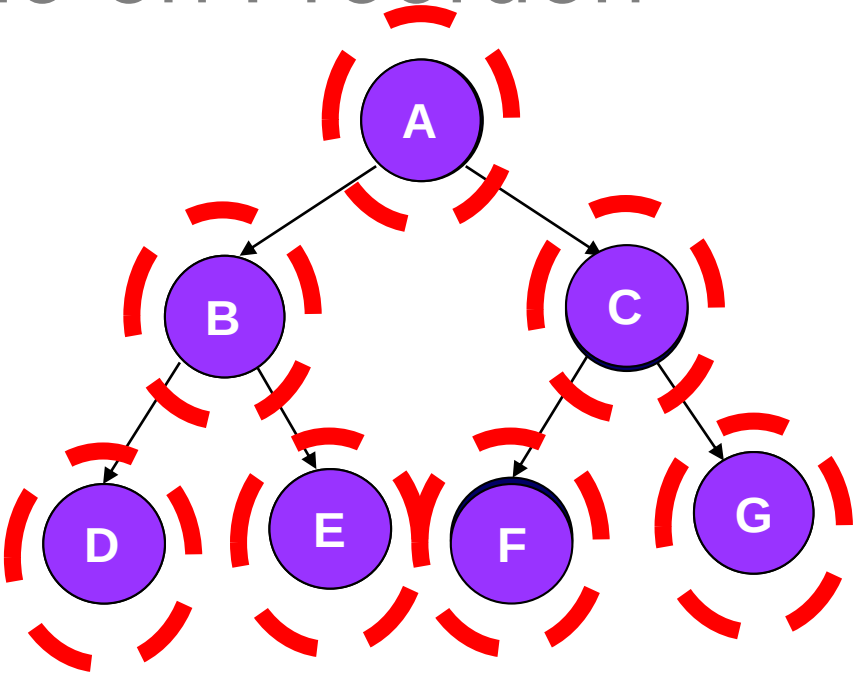
Recorrido en Preorden

1. Visitar la raíz.

2. Recorrer subárbol izquierdo en preorden.

3. Recorrer subárbol derecho en preorden.

Recorrido en Preorden



Recorrido:

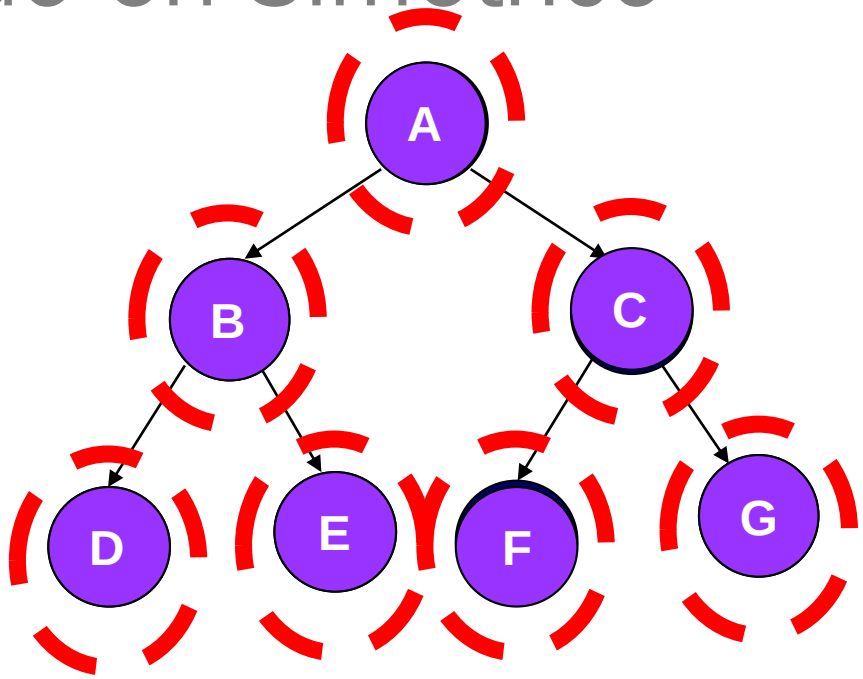
A	B	D	E	C	F	G

1. Raíz.
2. Subárbol izquierdo en preorden.
3. Subárbol derecho en preorden.

Recorrido en Simétrico

1. Recorrer subárbol izquierdo en simétrico.
2. Visitar la raíz.
3. Recorrer subárbol derecho en simétrico.

Recorrido en Simétrico



Recorrido

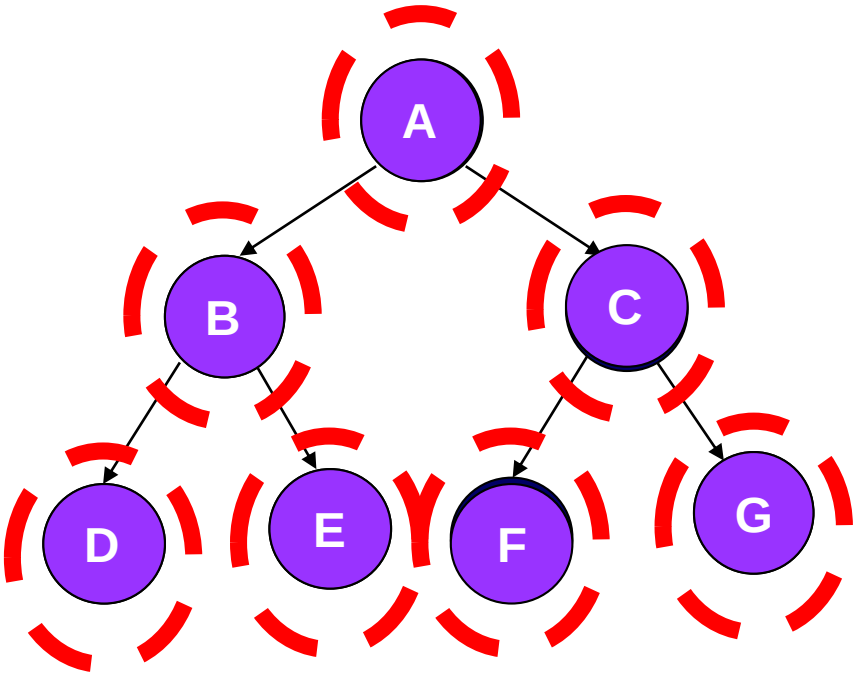
D B E A F C G

1. Subárbol izquierdo en simétrico.
2. Raíz.
3. Subárbol derecho en simétrico.

Recorrido en Postorden

1. Recorrer subárbol izquierdo en orden final.
2. Recorrer subárbol derecho en orden final.
3. Visitar la raíz.

Recorrido en Post Orden



Recorrido

1. Subárbol izquierdo en orden final.
2. Subárbol derecho en orden final.
3. Raíz.

D	E	B	F	G	C	A

Árbol Binario: Implementación en C#

```
class TBinTreeNode
{
protected:
  void*  aInfo;
  TBinTreeNode* aLeft;
  TBinTreeNode* aRight;
 TBinTreeNode* Left() {return aLeft;}
  void Left(TBinTreeNode* pNode) {aLeft = pNode;}
  TBinTreeNode* Right() {return aRight;}
  void Right(TBinTreeNode* pNode) {aRight = pNode;}
public:
  TBinTreeNode(void* pInfo) : aInfo(pInfo), aLeft(NULL), aRight(NULL) {}
  virtual int Degree();
  void* Info() {return aInfo;}
  virtual bool IsLeaf() {return (!aLeft && !aRight);}  //  Degree() == 0
};
```

```
class TBinTree {
protected:
   TBinTreeNode* aRoot;
public:
   TBinTree() {aRoot = NULL;}
   ~TBinTree();
   virtual void* DeleteNode(TBinTreeNode*);
   bool DivideTree(TBinTreeNode*, TBinTree* &, TBinTree* &);
   bool Empty(){return !aRoot;}
   TBinTreeNode* GetFather(TBinTreeNode*);
   virtual TGLinkedList* GetLeaves();
   virtual bool InsertNode(TBinTreeNode*, char, TBinTreeNode*);
   int NodeLevel(TBinTreeNode*);
   TBinTreeNode* Root() {return aRoot;}
   void Root(TBinTreeNode* pRoot) {aRoot = pRoot;}
   int TreeDegree();
   int TreeLevel();
};
```

Árboles de Búsqueda

Permiten realizar operaciones (recorridos, búsqueda de un elemento, etc) de forma más eficiente.

Hay dos momentos para la manipulación de un árbol:

- La construcción del árbol.
- El recorrido del árbol para realizar las operaciones requeridas según el problema a resolver.

Existen dos tipos especiales de árboles:

- Árboles lexicográficos.
- Árboles hilvanados.

Árboles Lexicográficos

Un ***árbol lexicográfico*** es un árbol binario que, recorrido en orden simétrico(inorder), permite obtener la información de los nodos en algún criterio de ordenamiento.

La técnica de construcción de un árbol lexicográfico consiste en un proceso recursivo que va colocando los nodos en el subárbol izquierdo o derecho del nodo raíz, según sea el criterio de ordenamiento deseado (ascendente o descendente).

Árboles Lexicográficos

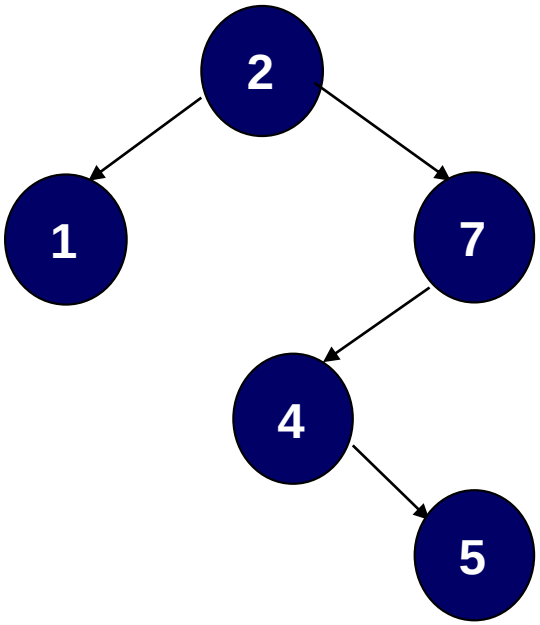
Siguiendo un ordenamiento ascendente:

1. Se compara el nodo que se quiere insertar con la raíz del árbol.
   - Si es menor, se coloca en el subárbol izquierdo siguiendo el mismo proceso.
   - Si es mayor, se coloca en el subárbol derecho siguiendo el mismo proceso.

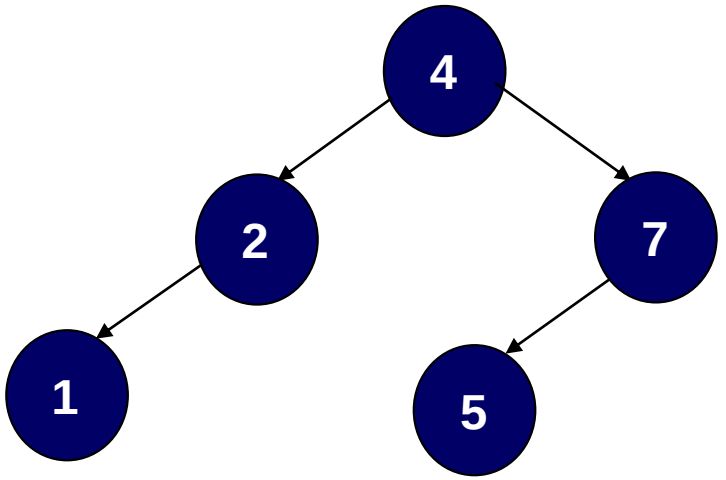
Árboles Lexicográficos: Ejemplo

Árbol lexicográfico con ordenamiento ascendente.

Lista: 2, 7, 1, 4, 5



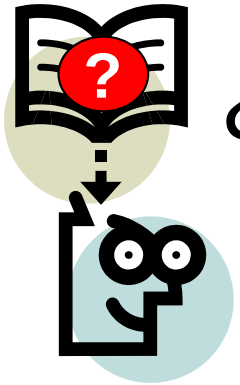
Lista: 4, 7, 2, 1, 5



Si se recorre en orden simétrico, se obtiene la información de sus nodos en orden ascendente: 1, 2, 4, 5, 7 con independencia del orden de la lista original.

Problemas

- El recorrido de árboles con programas recursivos resulta costoso ya que implica un gasto adicional de memoria y tiempo de ejecución. Para árboles muy grandes se puede desbordar el stack del sistema relativamente pronto.



¿Cuál es la solución?

Árboles hilvanados

Árboles Hilvanados

- Un ***árbol hilvanado*** (o ***árbol entrelazado***) es un árbol binario en el que cada hijo izquierdo de valor nulo es sustituido por un enlace o hilván al nodo que le antecede en orden simétrico (excepto el primer nodo en orden simétrico) y cada hijo derecho de valor nulo es sustituido por un enlace o hilván al nodo que le sigue en el recorrido en orden simétrico (excepto el último nodo en orden simétrico).

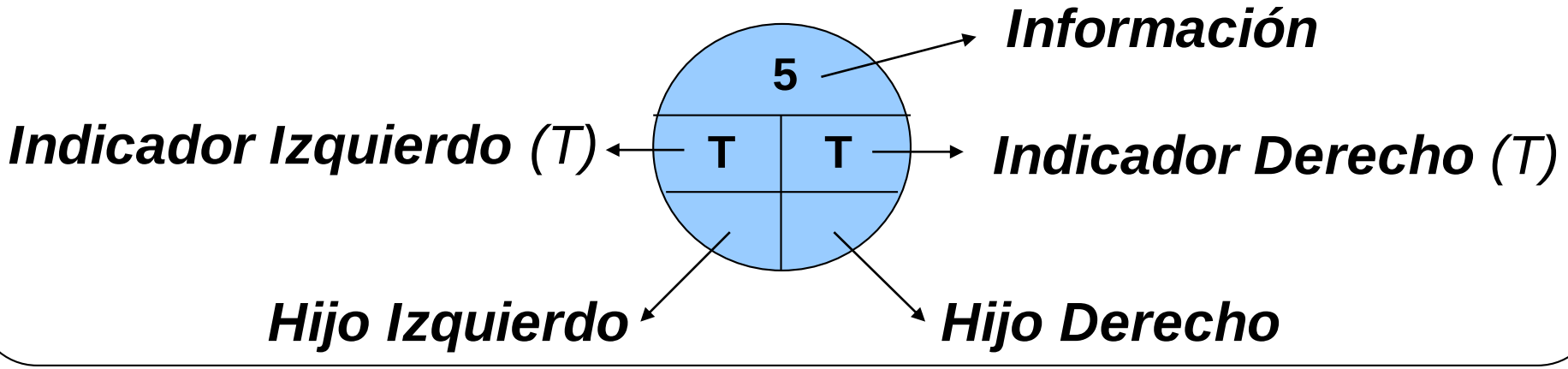
Árboles Hilvanados

Ahora, un recorrido en orden simétrico se puede implementar sin necesidad de recursión.

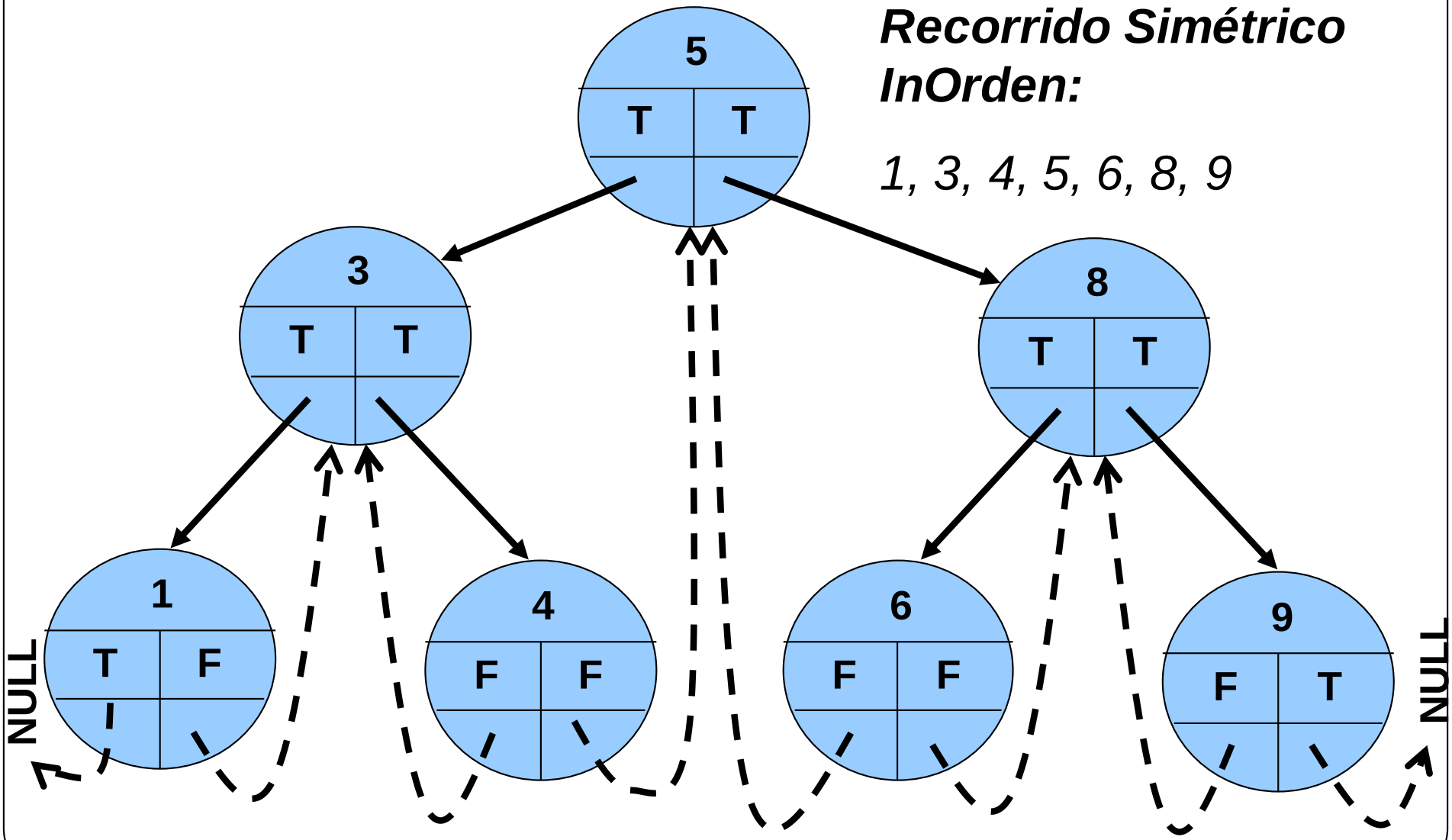
Sin embargo, se requiere que los nodos tengan en su estructura algún atributo que permita saber cuándo un enlace es real y cuándo se trata de un hilván. En este caso es necesario un atributo para cada hijo.

Árbol Hilvanado

- Cada nodo del árbol hilvanado contiene:
- Una referencia a su información
- Un apuntador a su hijo izquierdo.
- Indicador Izquierdo (Verdadero o Falso).
- Un apuntador a su hijo derecho.
- Indicador Derecho (Verdadero o Falso).



Árboles Hilvanados



Construyendo Árboles Hilvanados

1. Se coloca el nodo raíz del árbol

Si el nodo a insertar es el hijo izquierdo del nodo N:

- Se pone como hijo izquierdo del nodo a insertar a lo que era el hijo izquierdo del nodo N.

- Se pone como hijo derecho del nodo a insertar al nodo N.

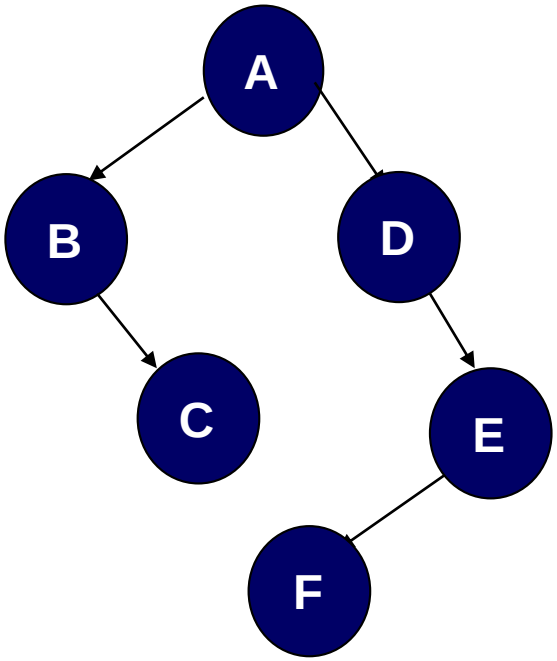
- Se pone como hijo izquierdo del nodo N al nodo a insertar.

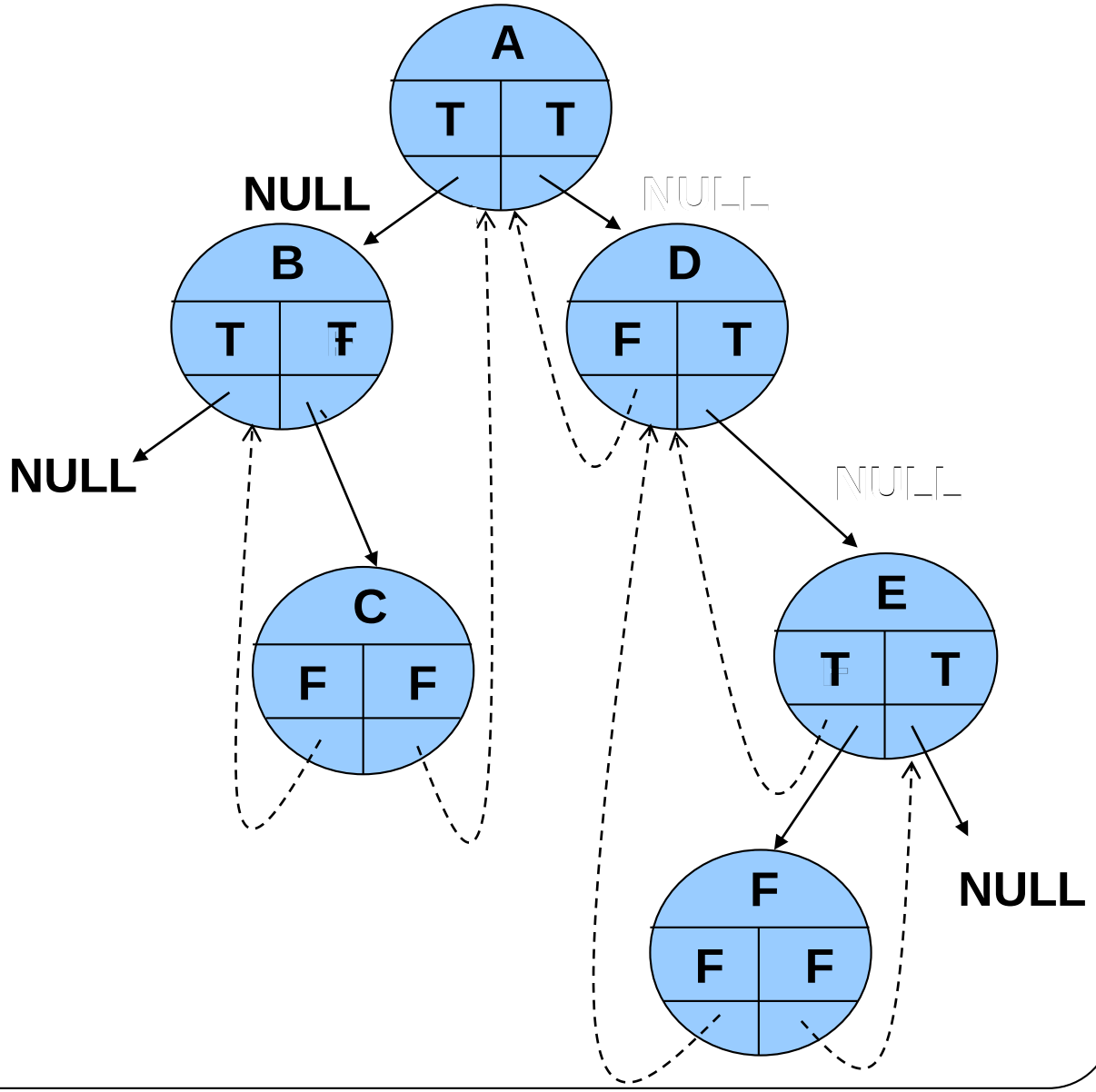
Construyendo Árboles Hilvanados

Si el nodo a insertar es el hijo derecho del nodo N:

- Se pone como hijo derecho del nodo a insertar a lo que era el hijo derecho del nodo N.
- Se pone como hijo izquierdo del nodo a insertar al nodo N.
- Se pone como hijo derecho del nodo N al nodo a insertar.

Construyendo Árboles Hilvanados



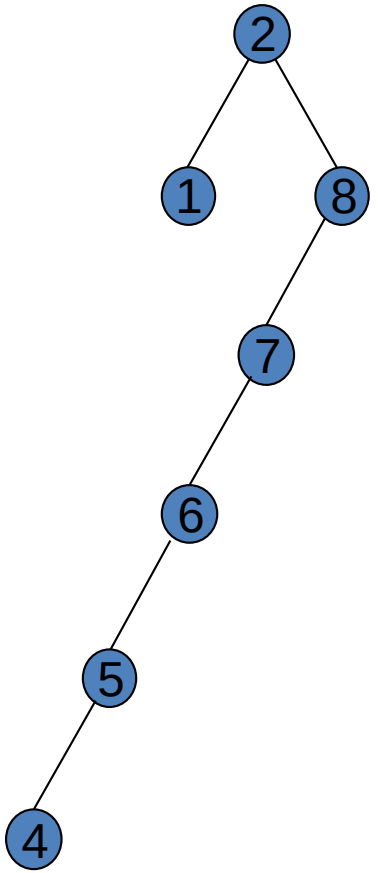


Árbol Hilvanado: Implementación en C++

```
class TThreadedTreeNode : public TBinTreeNode
{
private:
  bool aIsLeft;  // Indicador Izquierdo
  bool aIsRight; // Indicador Derecho
public:
   TThreadedTreeNode(void* pInfo): TBinTreeNode(pInfo) {
                                                  aIsLeft = false;
                                                  aIsRight = false;}
};
```

Árboles desbalanceados

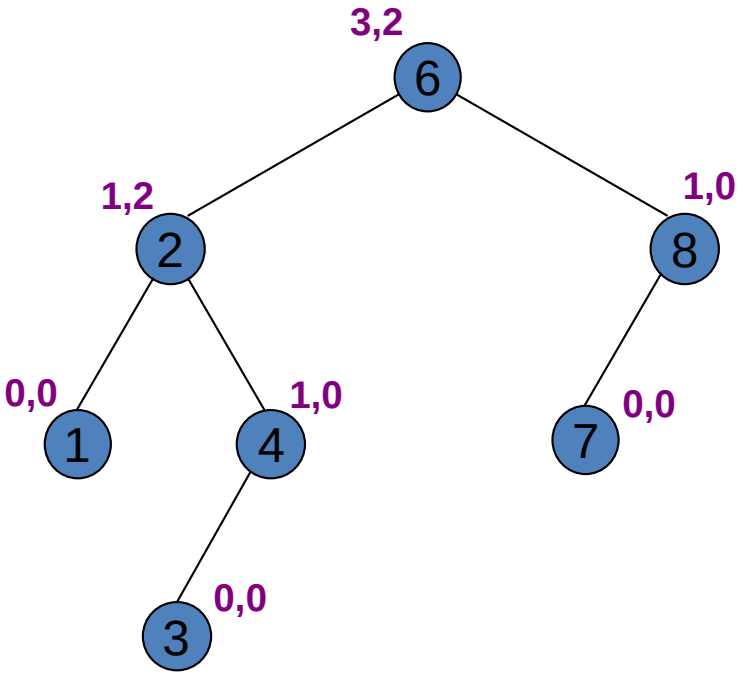
- La figura muestra un árbol binario de búsqueda, sin embargo éste NO facilita la búsqueda de elementos.
- El problema es que está muy desbalanceado.
- La solución es balancearlo, y con ello asegurar que asegurar que tiempo promedio de búsqueda sea de $O(\log_2 N)$.

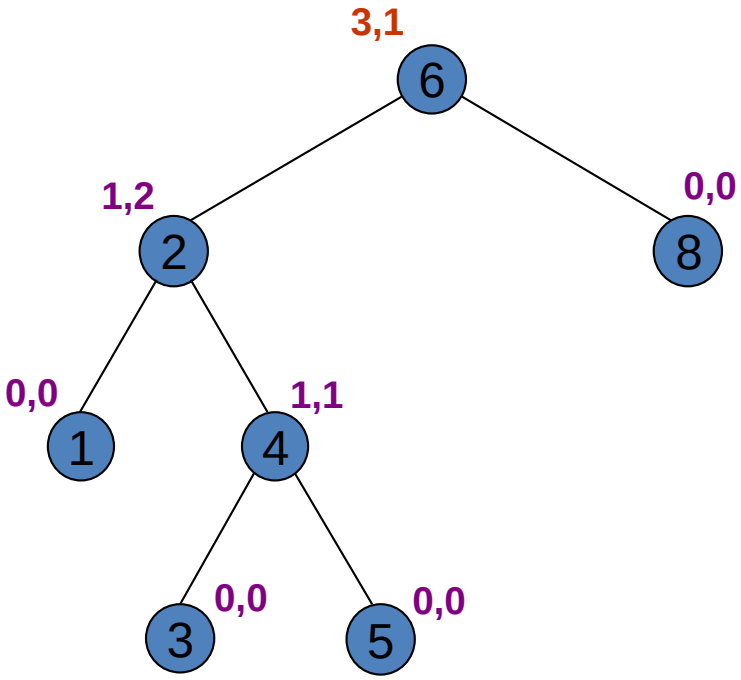


Árboles AVL

- Propuestos por **A**delson-**V**elskii and **L**andis.
- Árboles binarios con una condición de balance.
- Esta condición es usada para asegurar que en todo momento la altura del árbol es $O(\log_2 N)$.
- Condición de balance: para cada nodo del árbol, las alturas de sus subárboles izquierdo y derecho sólo pueden diferir como máximo en 1.
- La condición de balance debe mantenerse después de cada operación de inserción o eliminación.

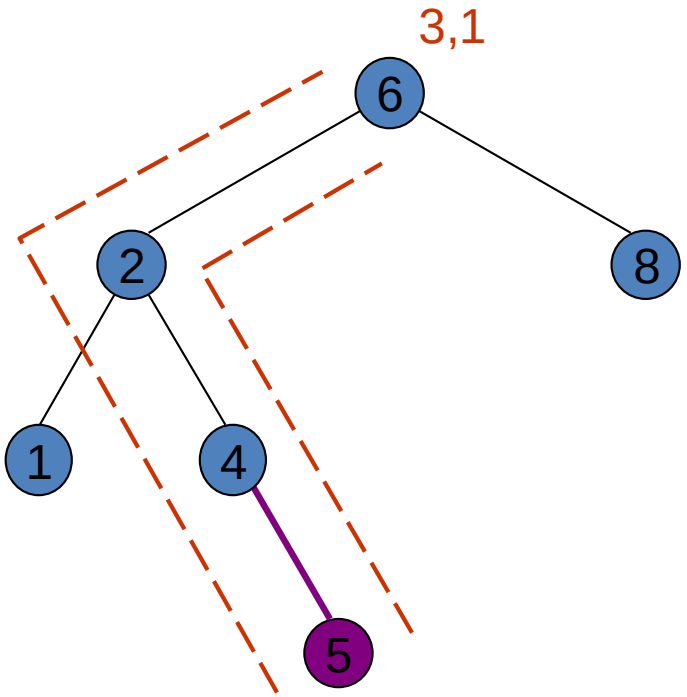
¿son árboles AVL?



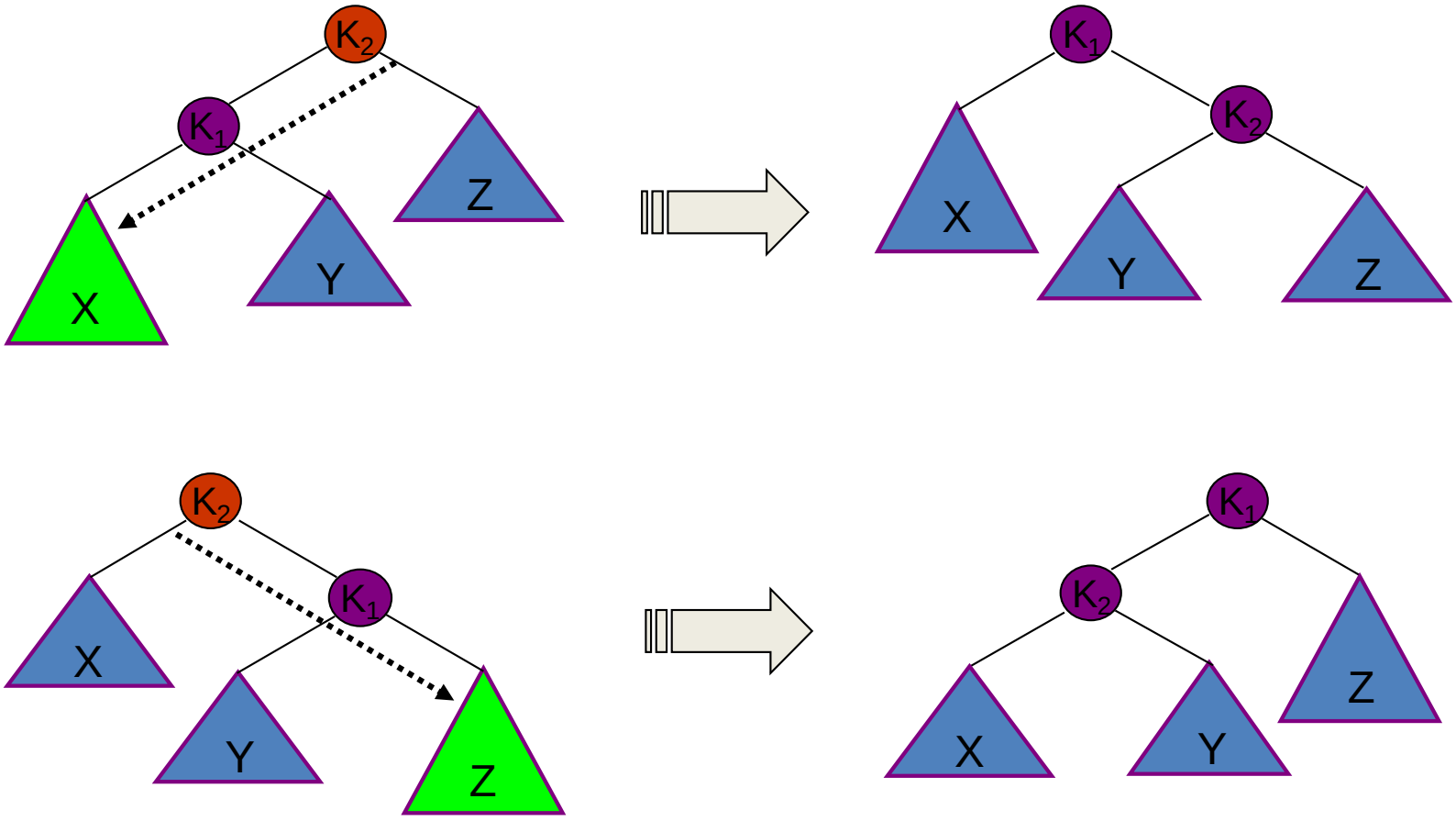


Construcción de un árbol AVL

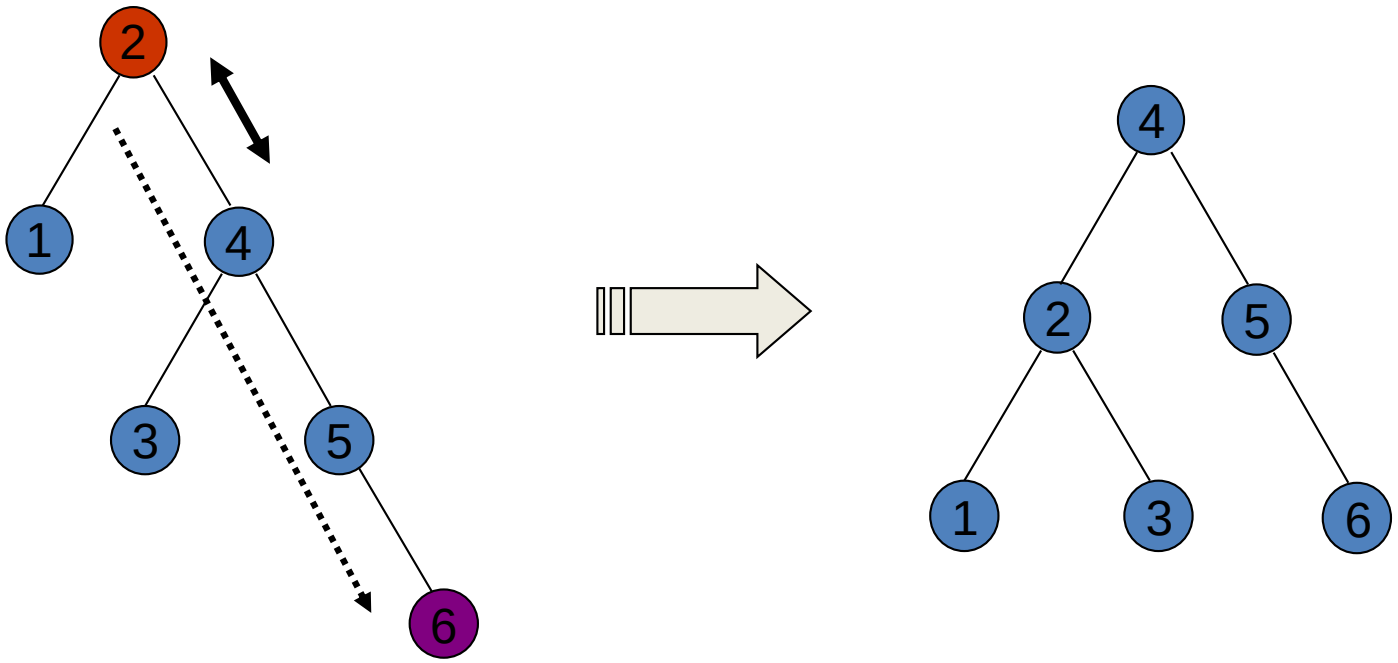
- Cuando se inserta un nodo se modifican las condiciones de balance en la trayectoria hacia la raíz.
- Si se presenta una desbalance, entonces es necesario hacer algunas modificaciones al árbol.
- Dos tipos de modificaciones:
  - Rotación simple
  - Rotación doble



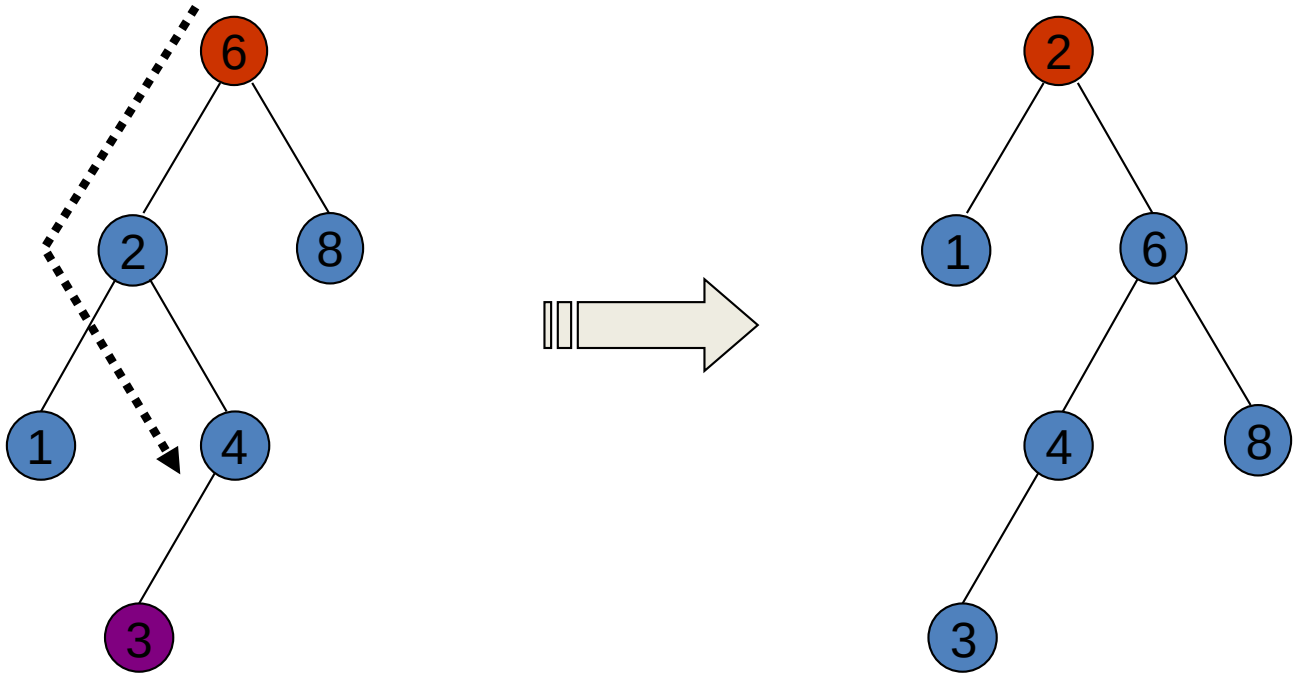
Rotación simple



Ejemplo rotación simple

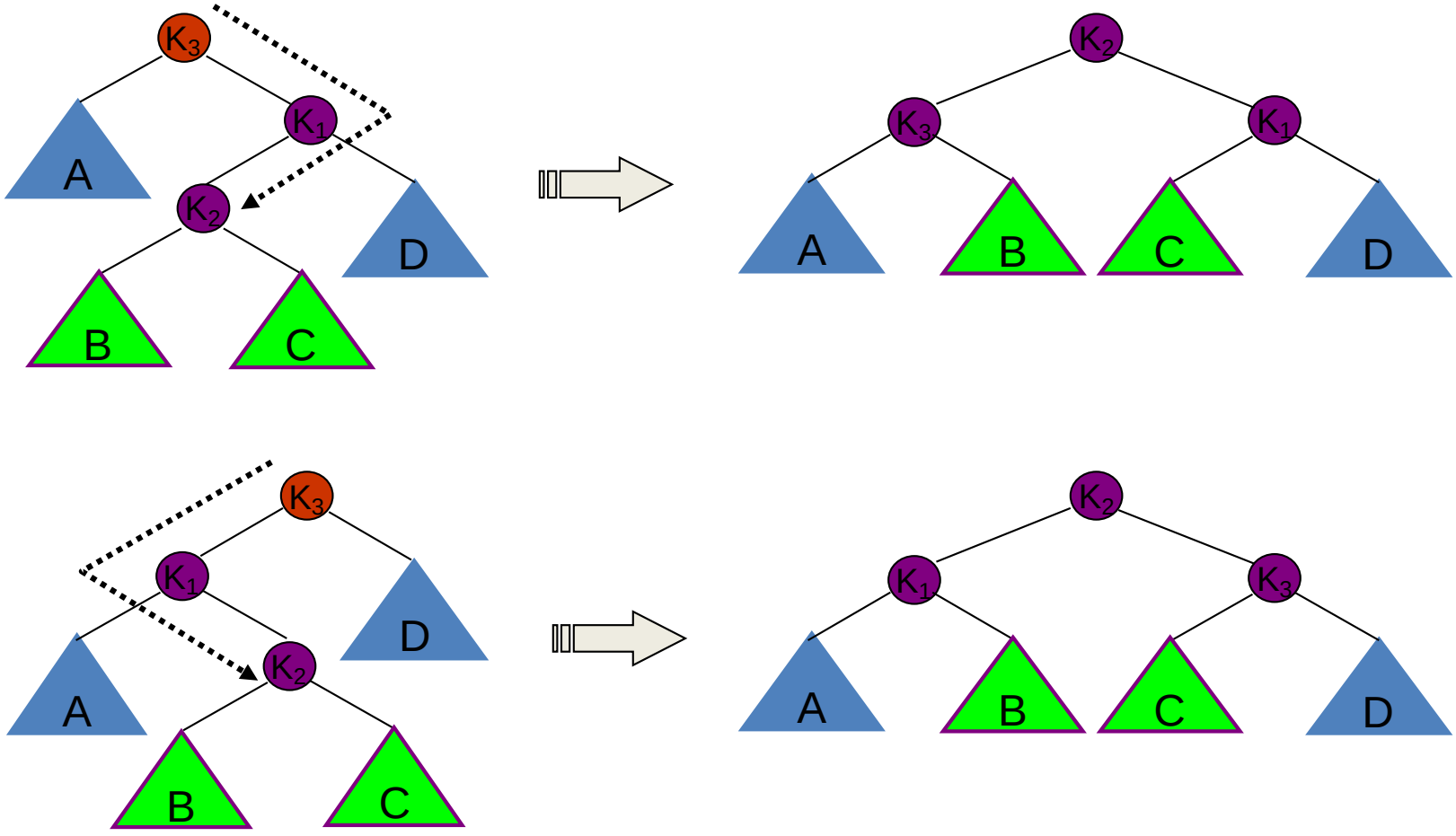


NO siempre es suficiente

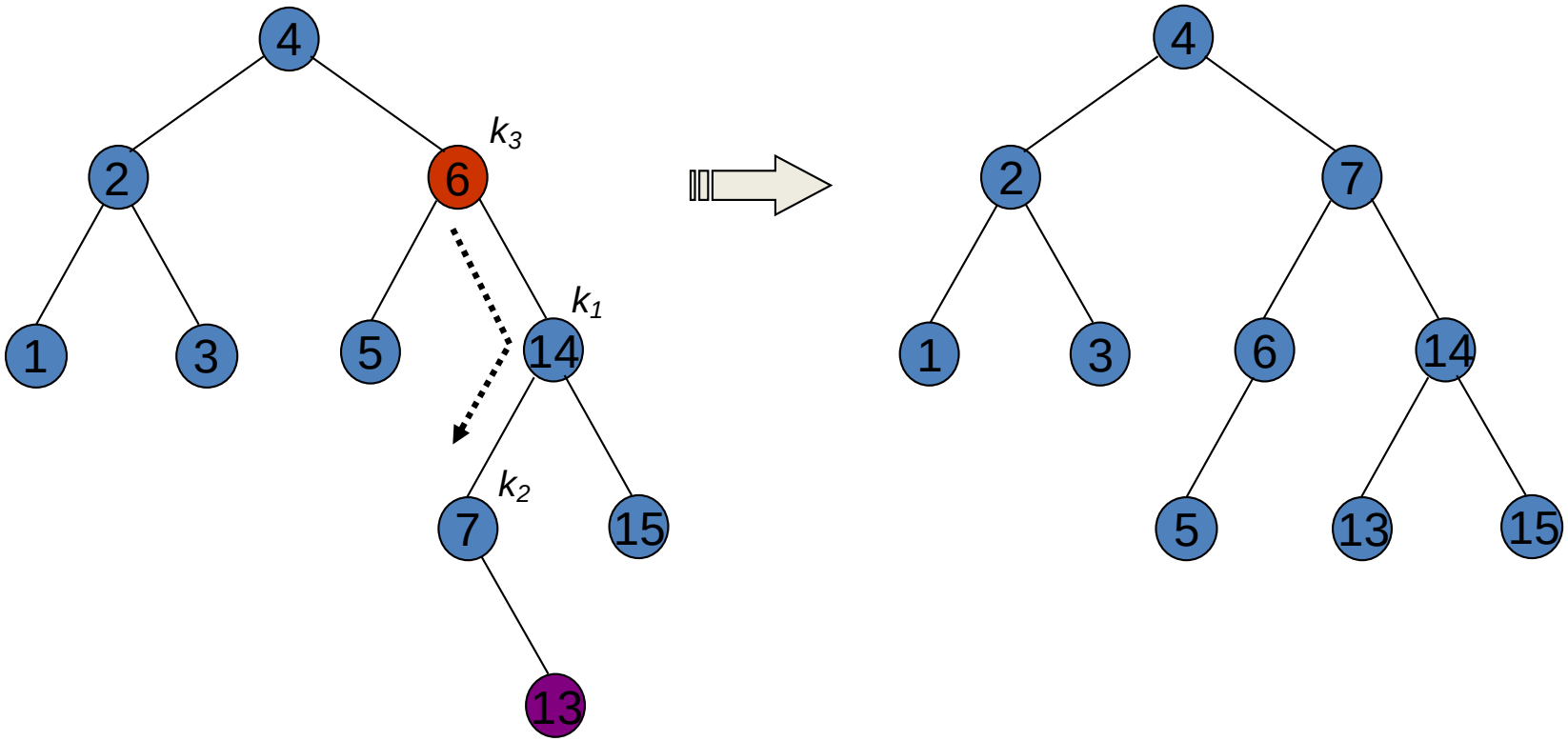


¡tenemos un caso de “pata de perro”!

Rotación doble

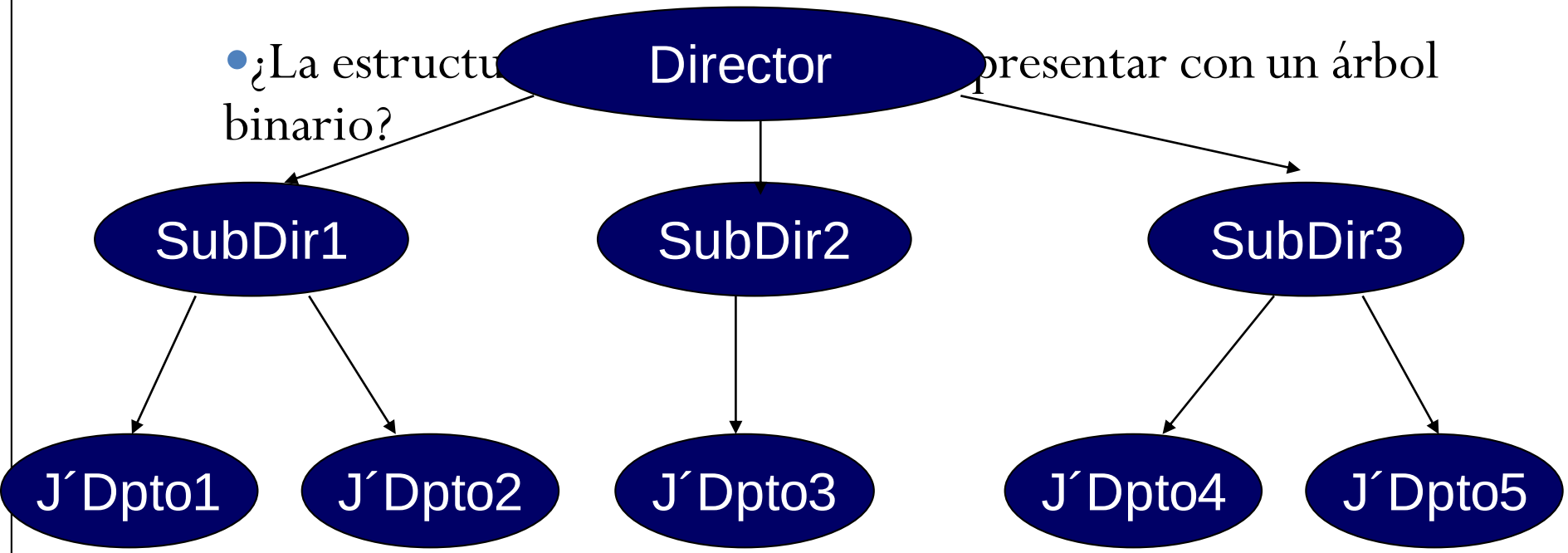


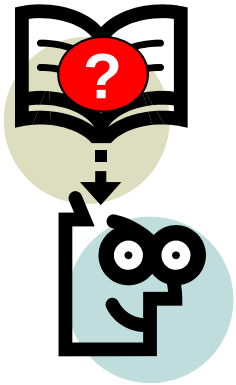
Ejemplo doble rotación



Árboles Generales

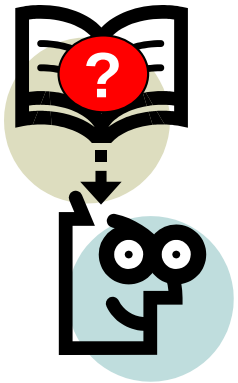
- ¿La estructu… presentar con un árbol binario?





Árboles Generales

- Son árboles cuyo grado es mayor que dos.



¿Cómo representarlos?

Árboles Generales

1

- Por cada nodo: la información y una lista de referencias a cada uno de sus hijos
- **Secuencial**: Se pierde espacio, cada nodo tiene un agrado diferente.

- **Enlazada**: la manipulación de la lista de hijos se hace difícil.

Árboles Generales

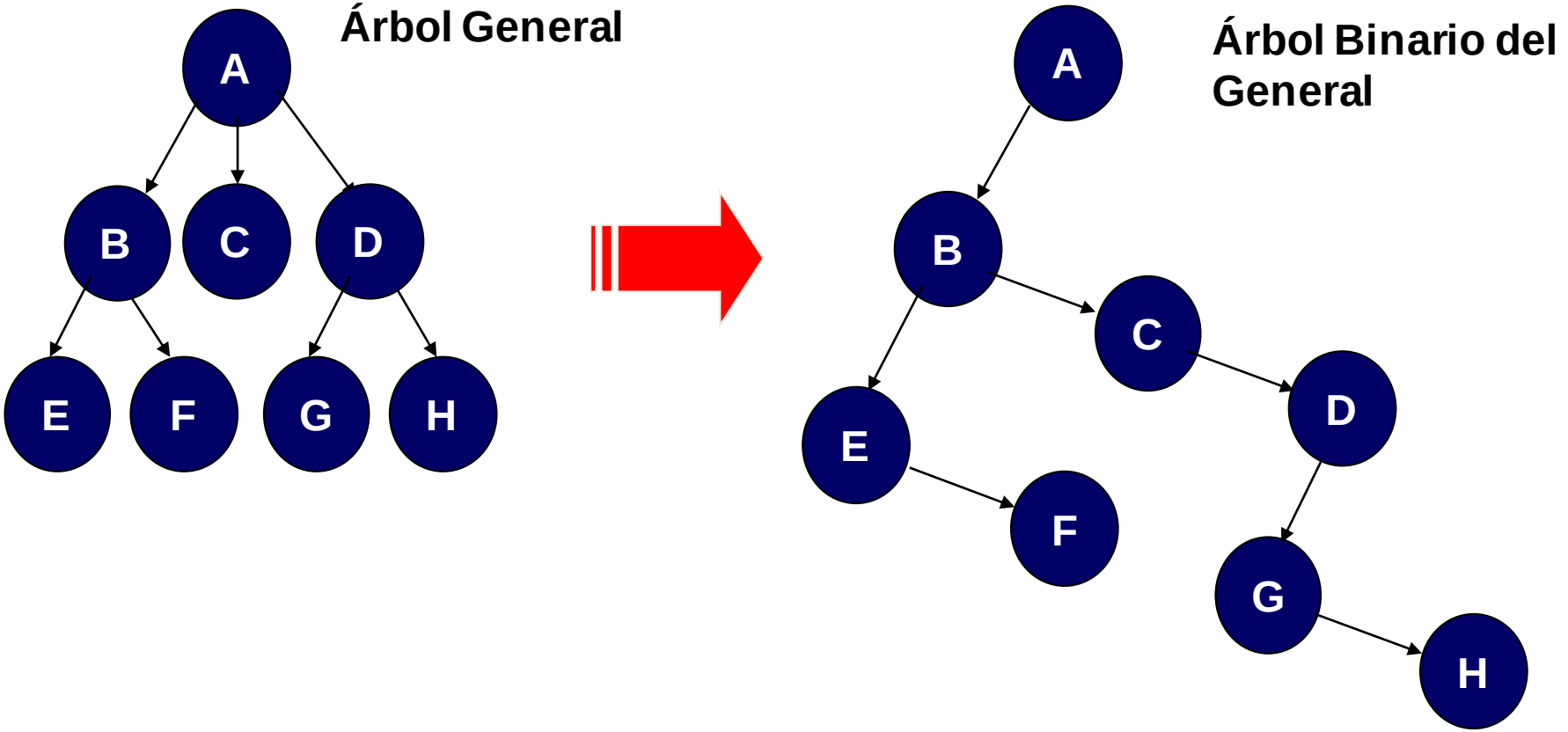
2 Transformar el árbol general en binario

Cada nodo tiene en su enlace izquierdo a su primer hijo en el general y a la derecha de un nodo van sus hermanos en el general.

Aclaraciones:

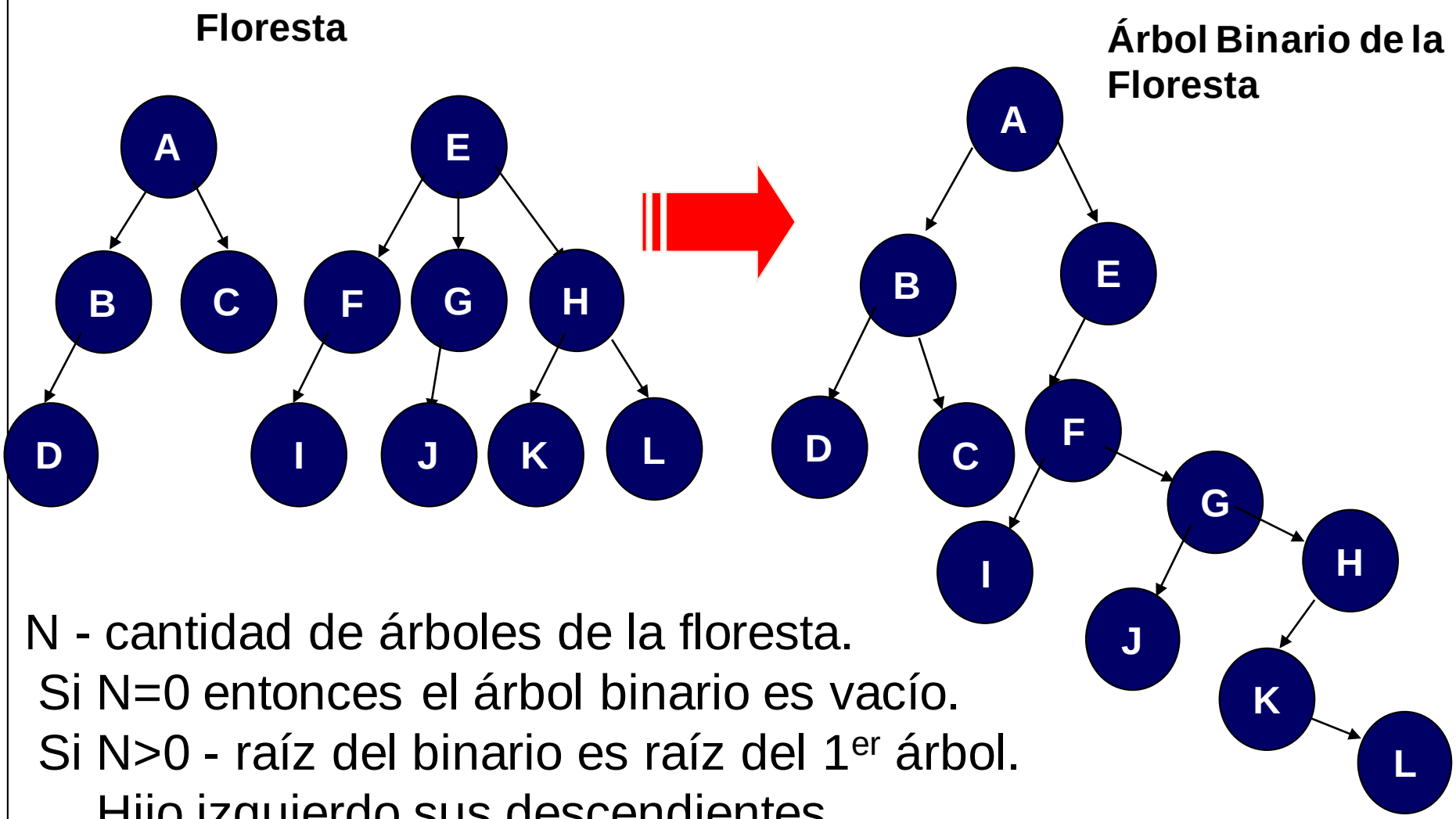
- El árbol se convierte en binario donde el **enlace izquierdo** representa al **primer hijo** (en el árbol general) y el **enlace derecho** al siguiente **hermano** (en el árbol general).
- El árbol es **ordenado** porque a la izquierda está su primer hijo (si lo tiene) y a la derecha estarán sus hermanos (si los tiene) con sus descendientes.

Transformación de General en Binario



El que no tiene hijo izquierdo es hoja en el general.
El que no tiene hijo derecho es el último hermano en el general.

Transformación de General en Binario



N - cantidad de árboles de la floresta.

Si N=0 entonces el árbol binario es vacío.

Si N>0 - raíz del binario es raíz del 1er árbol.

Hijo izquierdo sus descendientes.

Hijo derecho, la raíz del 2do árbol.

Árbol General: Implementación en C++

```
class TGBinTreeNode: public TBinTreeNode
{
public:
  TGBinTreeNode(void* pInfo): TBinTreeNode(pInfo) {}
  bool IsLeaf() {return !aLeft;}
  int Degree();
};
```

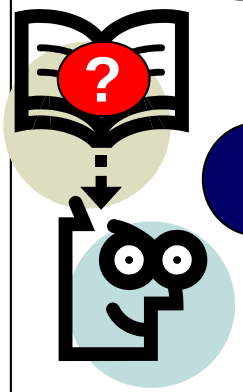
Árbol General: Implementación en C++

```
int TGBinTreeNode::Degree()
{
  int degree = 0;
  TBinTreeNode* cursor = Left();
  while (cursor)
  {
    degree++;
    cursor = cursor->Right();
  }
  return degree;
}
```

Árbol General: Implementación en C++

```
class TGBinTree: public TBinTree
{
public:
  void* DeleteNode(TGBinTreeNode*);
  TGBinTreeNode* GetFather(TGBinTreeNode*);
  TGLinkedList* GetLeaves();
  TGLinkedList* GetSons(TBinTreeNode*);
  bool InsertNode(TGBinTreeNode*, TGBinTreeNode*);
};
```

Colocación Secuencial de árboles



1. ¿Se puede colocar secuencialmente un árbol?

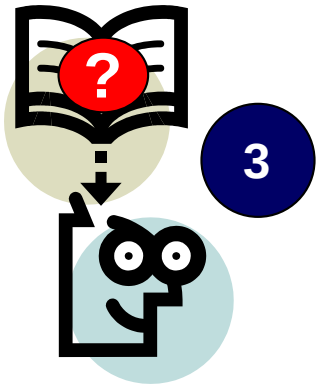
   Si

2. ¿Cuándo colocar secuencialmente un árbol?

Cuando debe recorrerse en múltiples ocasiones y no sufre frecuentes inserciones y/o eliminaciones.

Ejemplo: una fórmula que debe ser evaluada muchas veces.

Colocación Secuencial de Árboles



3 ¿Cómo colocar secuencialmente un árbol?

Los métodos más conocidos son:

Almacenamiento en ***Preorden Secuencial***.
Almacenamiento en ***Orden Familiar***.
Almacenamiento en ***Postorden Secuencial***.

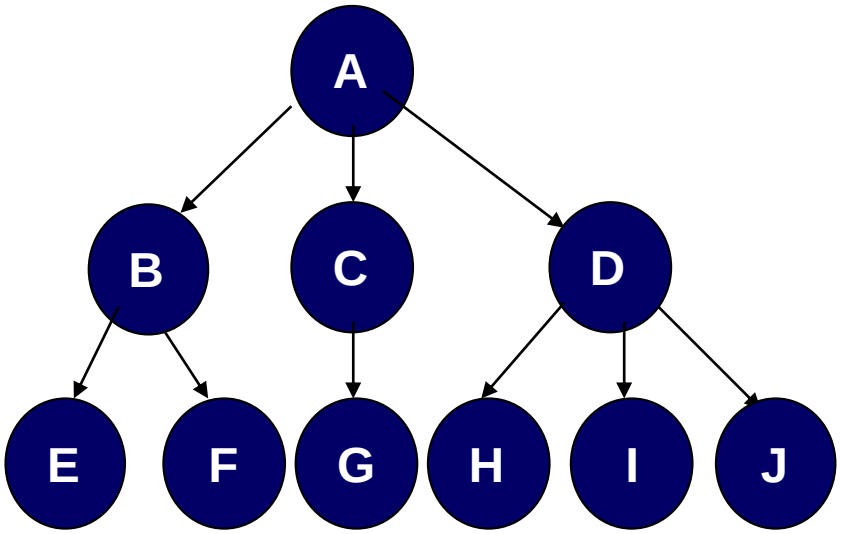
Colocación en Preorden Secuencial

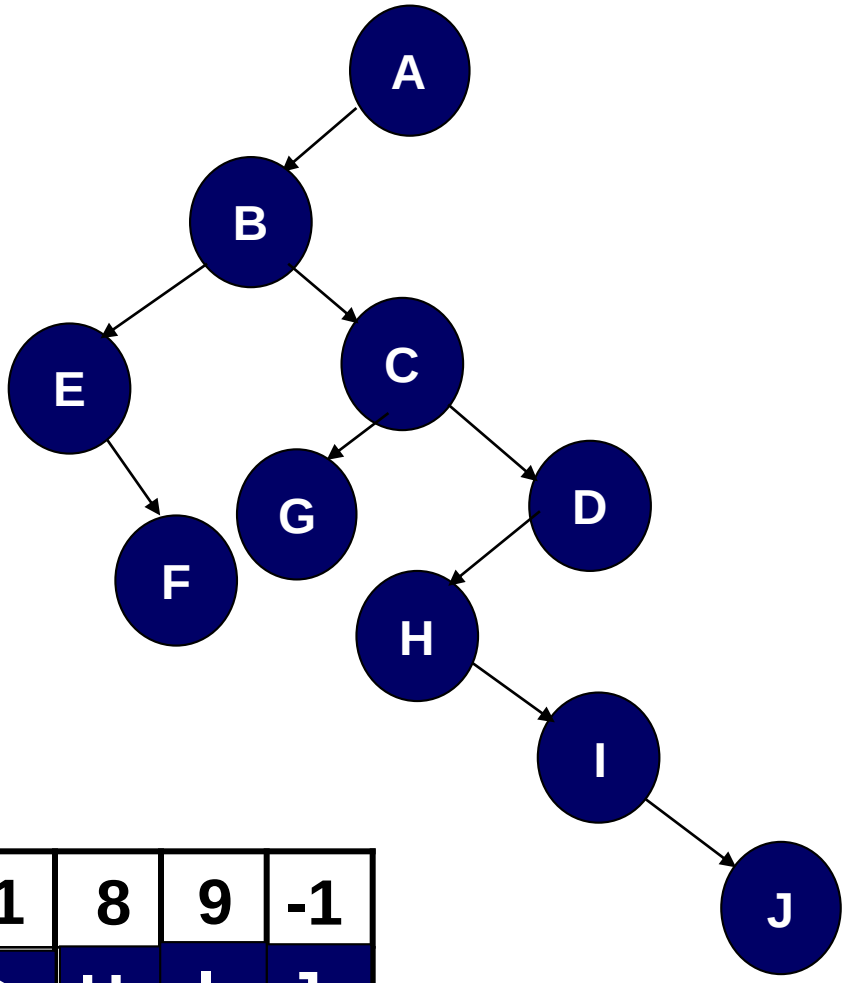
1. Se transforma a binario
2. Los nodos deben colocarse secuencialmente recorriendo al árbol en preorden.
3. Por cada nodo se registra tres campos:

INFO Árbol binario recorrido en Preorden **ENLDER** Siguiente hermano en el árbol general,
hijo derecho en el binario.
Convención: -1 si no existe

TERM Indica si el nodo es terminal (no tienen hijo en el general) y no tiene hijo izquierdo (en el binario).

Colocación en Preorden Secuencial





ENLDER	-1	4	3	-1	6	-1	-1	8	9	-1
INFO	A	B	E	F	C	G	D	H	I	J
TERM	F	F	T	T	F	T	F	T	T	T

Colocación en Preorden Secuencial

Aclaraciones:

- Los hermanos se obtienen a través del enlace derecho.

- Si un nodo no es terminal la siguiente posición de la lista secuencial está ocupada por su hijo. De lo contrario, es familia de otro nodo (hermano o hijo).

- Los subárboles están juntos, primero el padre y luego los hijos.

Implementación en C++

```
typedef int TIndex;
class TPreOrderNode
{
private:
  void* aInfo;
  TIndex aRightLink;
  bool aEnd;
public:
  TPreOrderNode(void* pInfo, bool pEnd) : aInfo(pInfo), aRightLink (-1),
                                          aEnd(pEnd){}
  void* Info() {return aInfo;}
  TIndex RightLink () {return aRightLink;}
  void RightLink(TIndex pRightLink) {aRightLink = pRightLink;}
  bool End() {return aEnd;}
};
```

Colocación en Orden Familiar

1. Se transforma a binario
2. Los nodos deben colocarse secuencialmente recorriendo al árbol en postorden invertido.
3. Por cada nodo se registra tres campos:

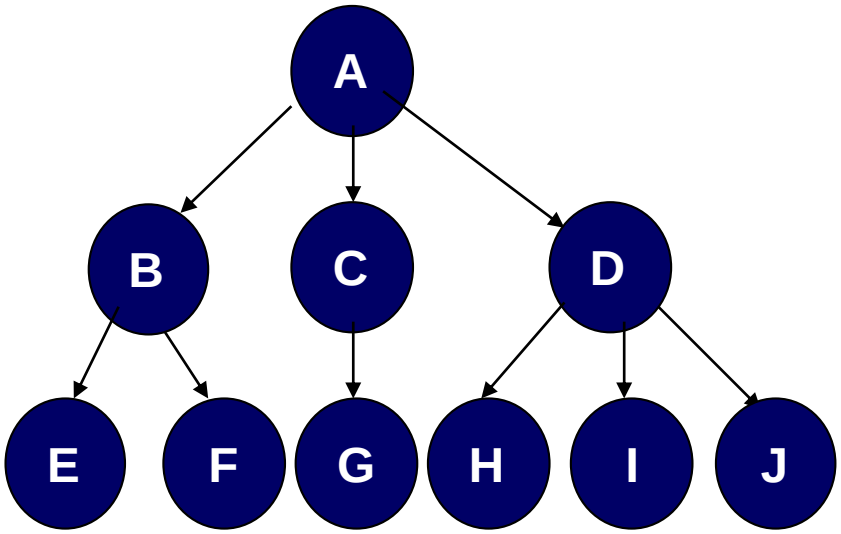
INFO Árbol binario recorrido en Postorden invertido

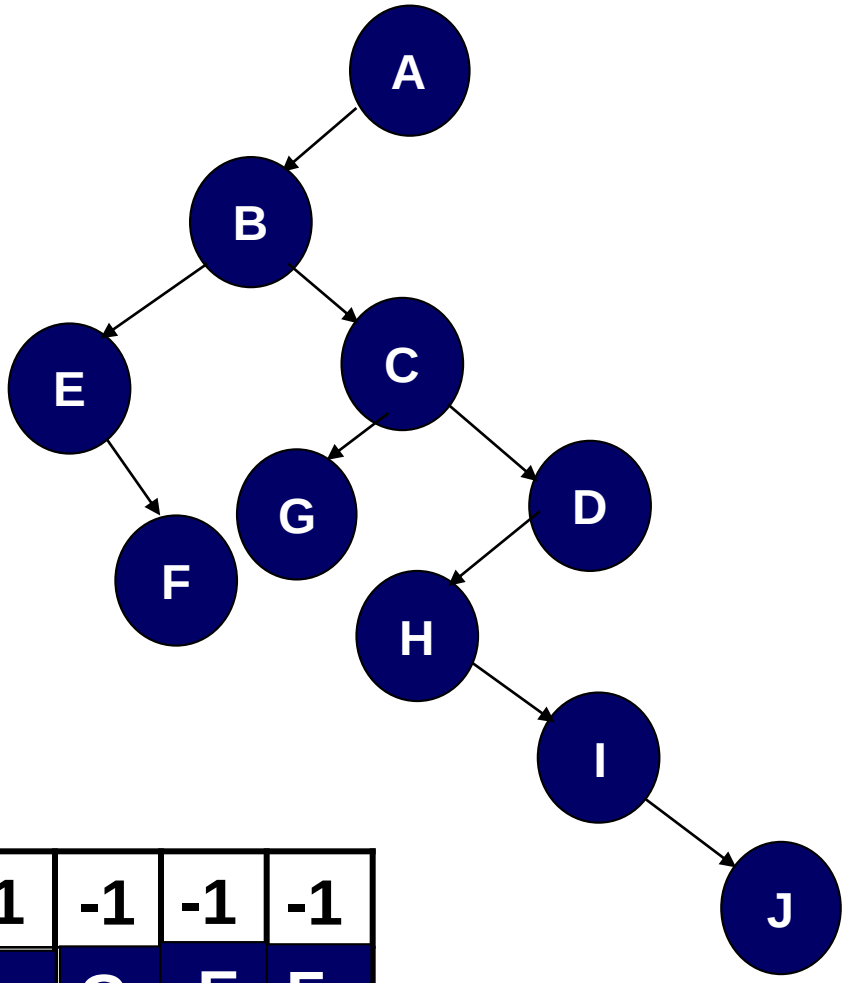
ENLIZQ primer hijo en el árbol general e hijo izquierdo en el binario.

Convención: -1 si no existe

TERM Indica último hermano en el árbol general y enlace derecho en NULL en el binario. Indica el nodo final de cada familia

Colocación en Orden Familiar





ENLIZQ	1	8	7	4	-1	-1	-1	-1	-1	-1
INFO	A	B	C	D	H	I	J	G	E	F
FAM	T	F	F	T	F	F	T	T	F	T

Colocación en Orden Familiar

Aclaraciones:

- El nodo raíz (si no es una floresta) y los nodos que no tienen un siguiente hermano se tienen FAM en T (True).
- El que sigue a un nodo es su hermano si FAM es F (False).
- Los hermanos están juntos secuencialmente.
- El enlace izquierdo indica el subíndice del primer hijo y los otros a continuación son los hermanos hasta que FAM tome valor True.

Implementación en C++

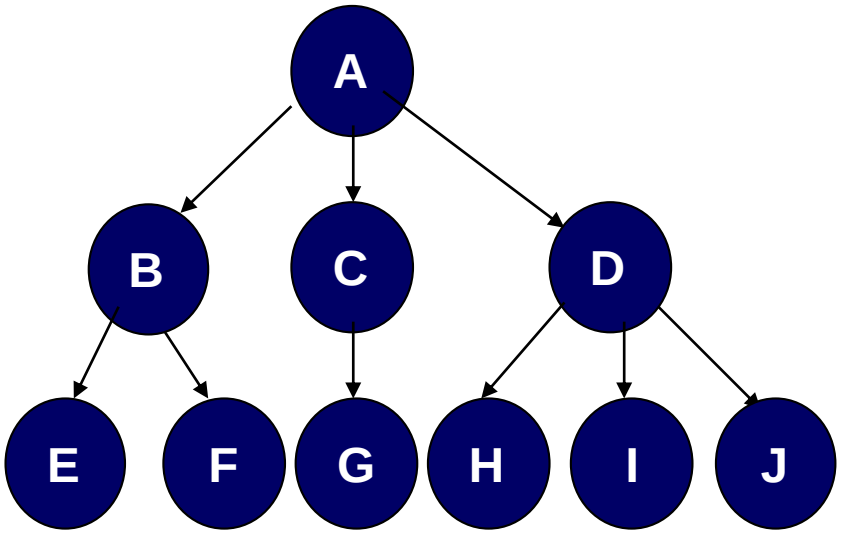
```
class TFamilyNode
{
private:
  void* aInfo;
  TIndex aLeftLink;
  bool aFamily;
public:
    TFamilyNode(void* pInfo, bool pFamily) : aInfo(pInfo), aLeftLink(-1),
                                             aFamily(pFamily){}
    void* Info() {return aInfo;}
    TIndex LeftLink () {return aLeftLink;}
    void LeftLink (TIndex pLeftLink) {aLeftLink = pLeftLink;}
    bool Family() {return aFamily;}
};
```

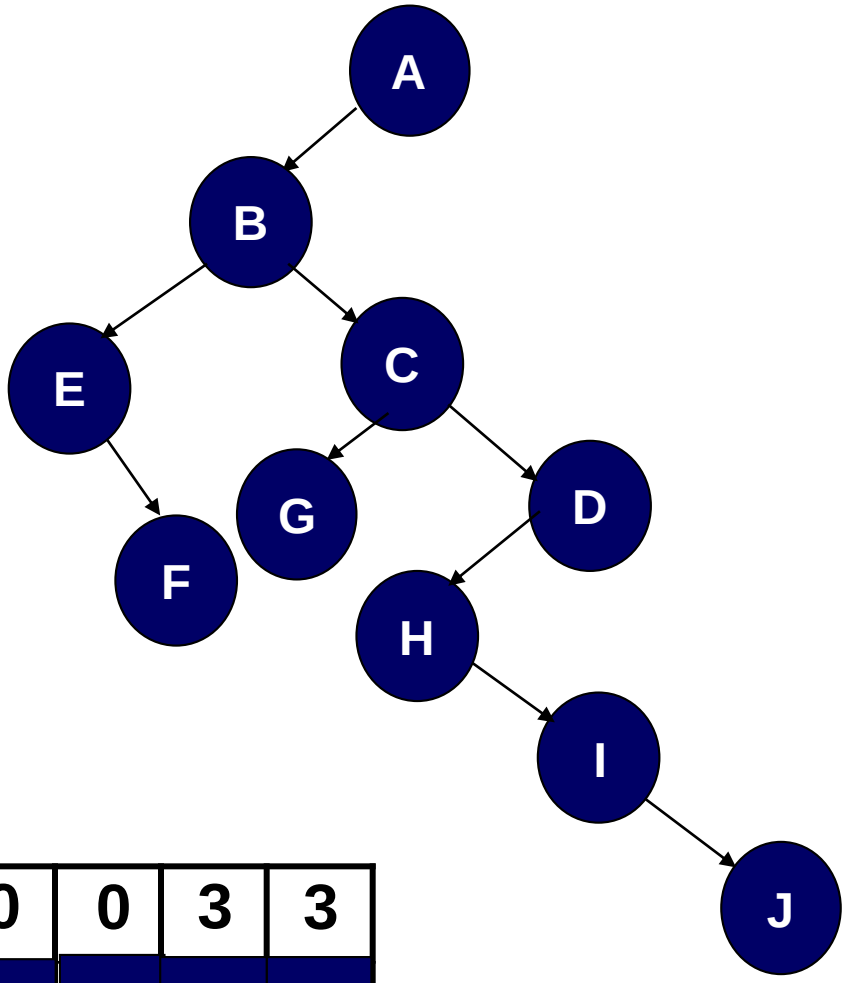
Colocación en Postorden Secuencial

1. Se transforma a binario.
2. Los nodos deben colocarse secuencialmente recorriendo al árbol en simétrico.
3. Por cada nodo se registra dos campos:

INFO	Árbol binario recorrido en Simétrico
GRADO	Grado del nodo

Colocación en Postorden Secuencial





GRADO	0	0	2	0	1	0	0	0	3	3
TERM	E	F	B	G	C	H	I	J	D	A

Colocación en Postorden Secuencial

Aclaraciones:

- Cada padre del árbol general está precedido de sus hijos, por tanto, es fácil encontrar el subárbol izquierdo de cada nodo del árbol binario. Se puede encontrar si recorremos la representación secuencial comenzando por el último elemento teniendo en cuenta el grado.
- Notar que si después de un padre aparece un nodo sin hijos el padre del primero se busca al final.

Ejemplo: el padre de C se busca al final.

Implementación en C++

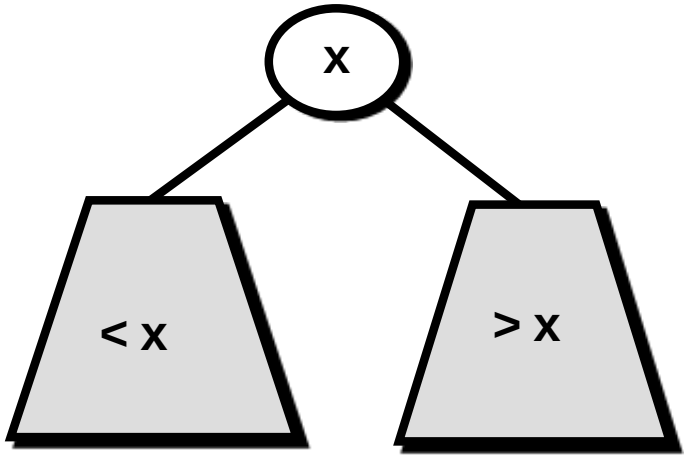
```
class TPostOrderNode
{
private:
  void* aInfo;
  int aDegree;
public:
  TPostOrderNode(void* pInfo, int pDegree) : aInfo(pInfo),
                                             aDegree(pDegree){}
  void* Info() {return aInfo;}
  int Degree() {return aDegree;}
};
```

Árboles B

- Los **árboles B** son muy usados en Bases de Datos.
- **Necesidades propias de las aplicaciones de BD:**
  - Muchos datos, básicamente conjuntos y diccionarios.
  - El acceso secuencial y directo debe ser rápido.
  - Datos almacenados en memoria secundaria (disco) en bloques.
- Existen muchas variantes: árboles B, B+ y B*.
- **Idea:** Generalizar el concepto de árbol binario de búsqueda a **árboles de búsqueda n-arios**.

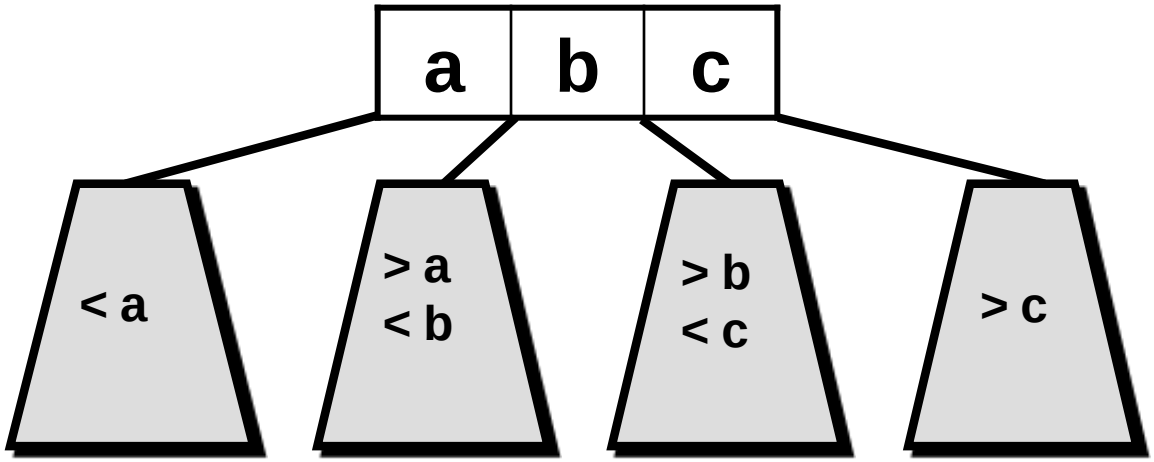
Árboles B

Árbol Binario de Búsqueda



Árbol de Búsqueda N-ario

- En cada nodo hay **n** claves y **n+1** punteros a nodos hijos.



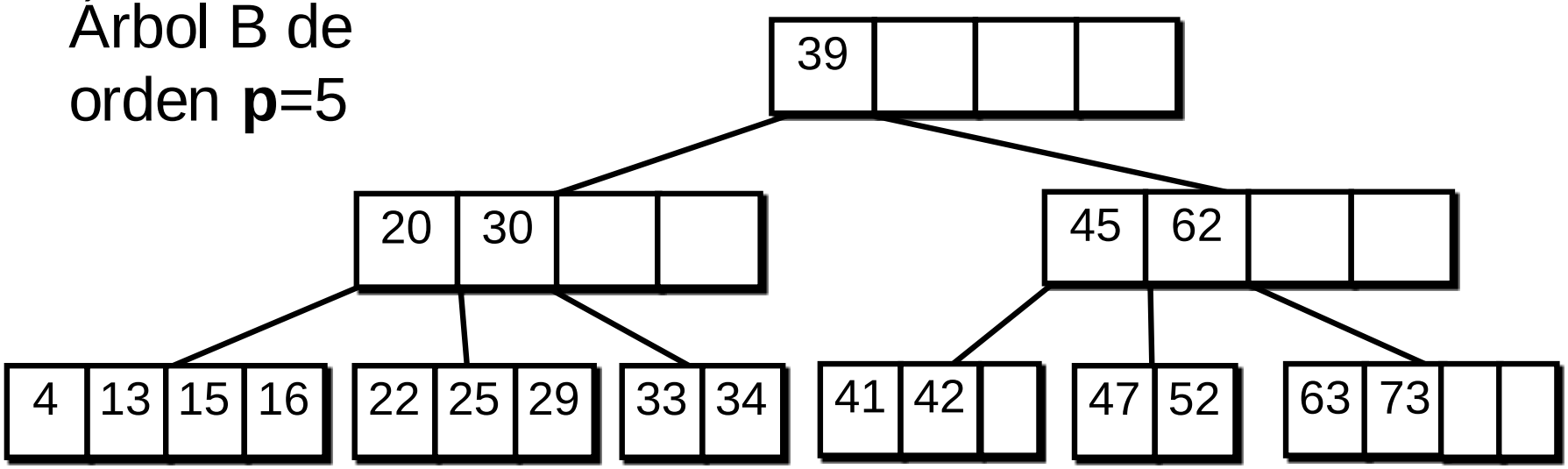
Árboles B

- **Definición: Un árbol B de orden p** es un árbol n-ario de búsqueda, que cumple las siguientes propiedades:
  - **Raíz del árbol:** o bien no tiene hijos o tiene como mínimo tiene **2** y como máximo **p**.
  - **Nodos internos:** tienen entre $\lceil p/2 \rceil$ y **p** hijos.
  - **Nodos hoja:** todas los nodos hojas deben aparecer al mismo nivel en el árbol (condición de balanceo).

- **Idea intuitiva:** Cada nodo tiene **p** posiciones (**p** punteros y **p-1** claves) que deben "llenarse" como mínimo hasta la mitad de su capacidad.

Árboles B

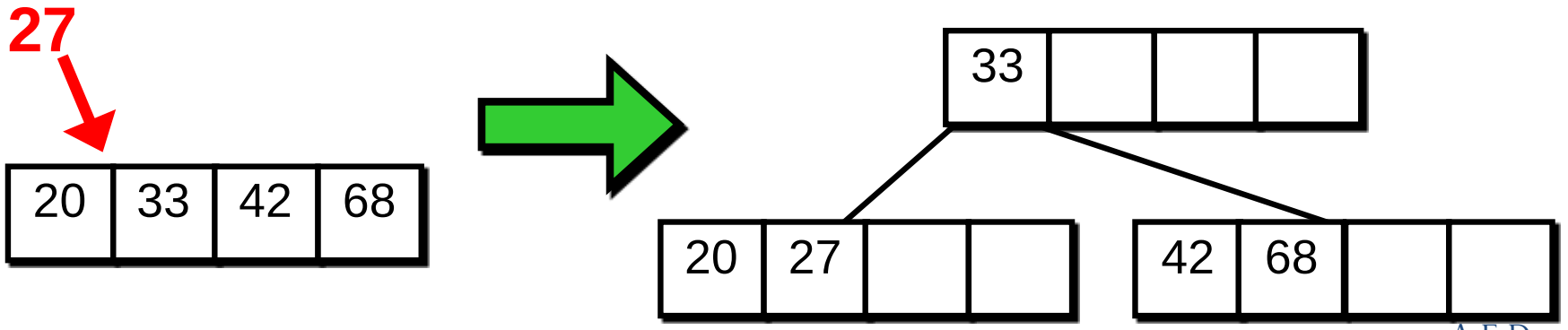
Árbol B de orden **p**=5



- **Búsqueda**: igual que en los árboles binarios, eligiendo la rama por la que seguir.
- La altura del árbol es ~ $\mathbf{log_{p/2}\ n}$, en el peor caso.

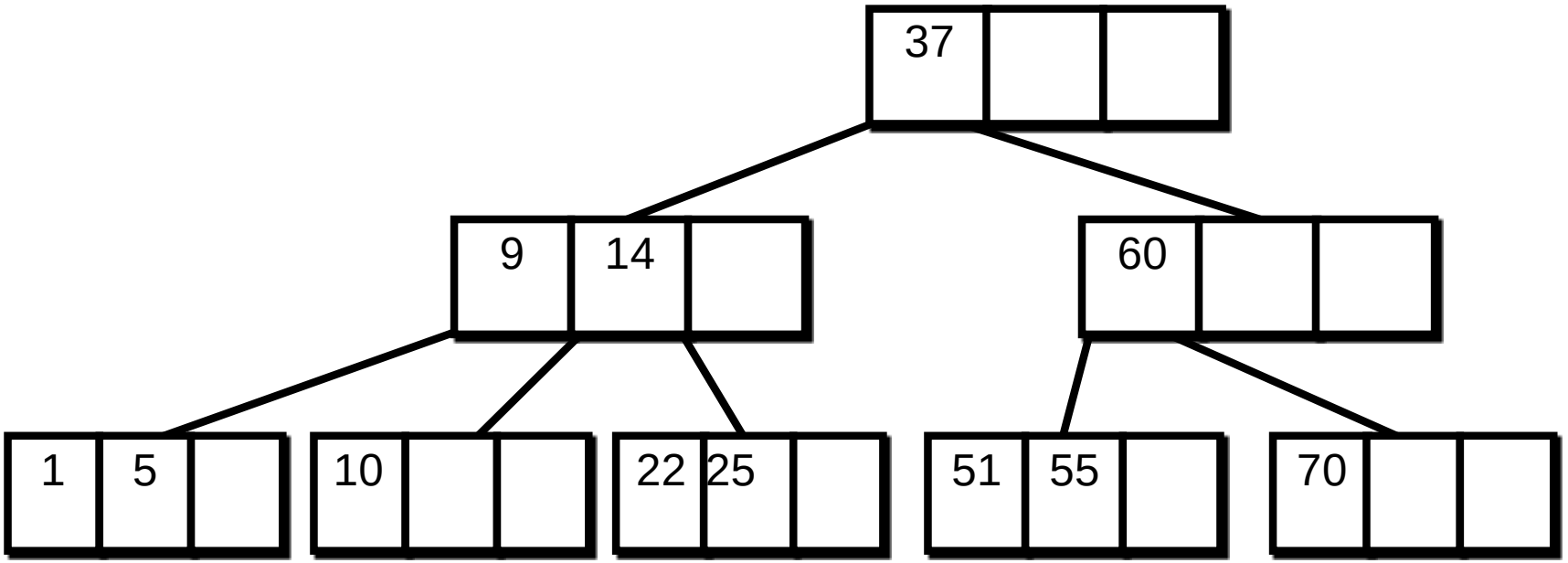
Arboles B

- **Inserción de entradas en un árbol B:** Buscar el nodo hoja donde se debería colocar la entrada.
  - **Si quedan sitios libres** en esa hoja, insertarlo (en el orden adecuado).
  - **Si no quedan sitios** (la hoja tiene **p-1** valores) partir la hoja en **2** hojas (con $\lceil (p-1)/2 \rceil$ y $\lfloor (p-1)/2 \rfloor$ nodos cada una) y añadir la mediana al nodo padre.
  - Si en el padre no caben más elementos, repetir recursivamente la partición de las hojas.



Árboles B

- **Ejemplo:** En un árbol B de orden **p**=4, insertar las claves: 37, 14, 60, 9, 22, 51, 10, 5, 55, 70, 1, 25.



- ¿Cuál es el resultado en un árbol B de orden **p**=5?

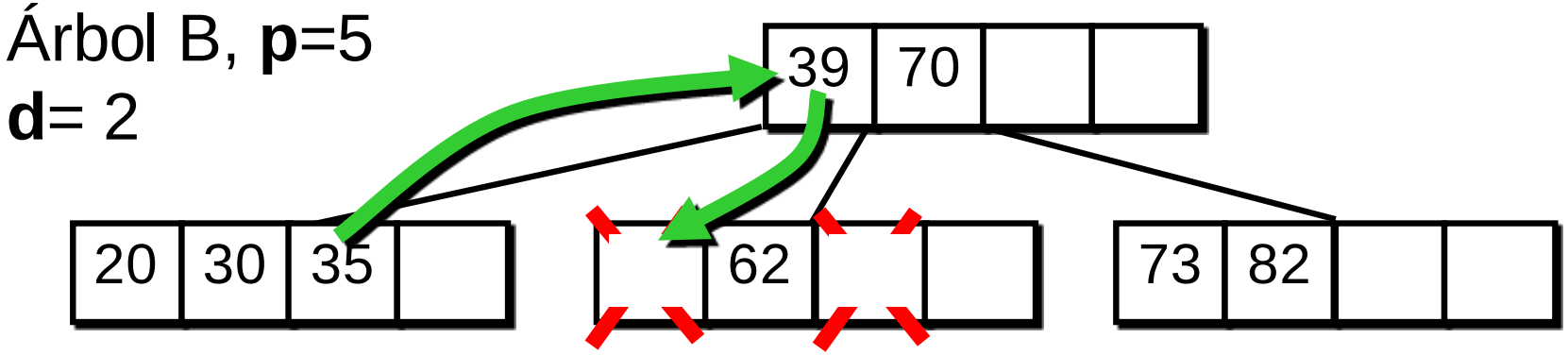
Árboles B

- **Eliminación de entradas en un árbol B:** Buscar la clave en el árbol.
  - **Nodo interno (no hoja):** Sustituirla por la siguiente (o la anterior) en el orden. Es decir, por la mayor de la rama izquierda, o la menor de la rama derecha.
  - **Nodo hoja:** Eliminar la entrada de la hoja.

- **Casos de eliminación en nodo hoja. $d = \lfloor (p-1)/2 \rfloor$**
  - Nodo con más de **d** entradas: suprimir la entrada.
  - Nodo con **d** entradas (el mínimo posible): reequilibrar el árbol.

Árboles B

- **Eliminación en nodo con d entradas**:
  - **Nodo hermano con más de d entradas:** Se produce un proceso de **préstamo** de entradas:
    Se suprime la entrada, la entrada del padre pasa a la hoja de supresión y la vecina cede una entrada al nodo padre.

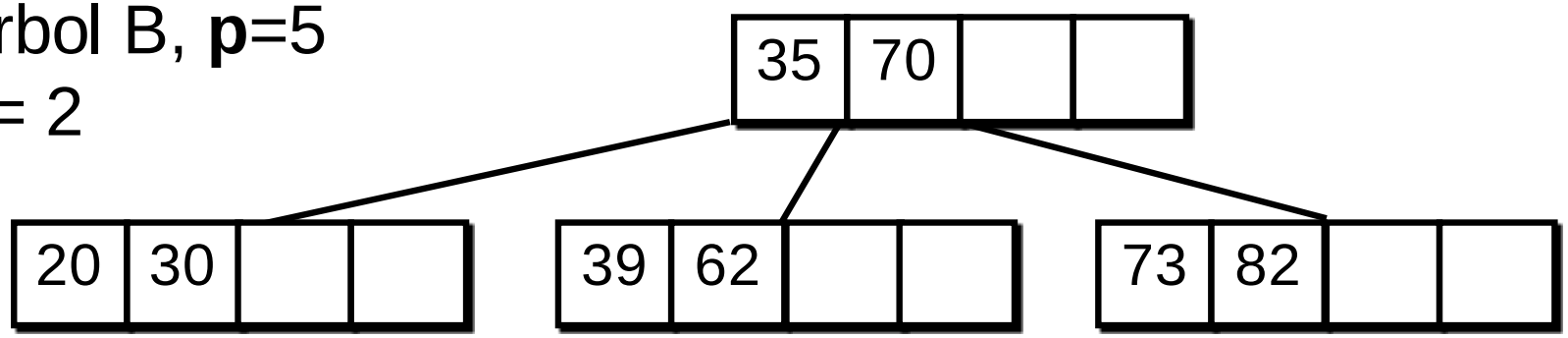


- **Ejemplo.** Eliminar 67, 45.

Árboles B

- **Eliminación en nodo con d entradas**:
  - **Nodo hermano con más de d entradas:** Se produce un proceso de **préstamo** de entradas:
    Se suprime la entrada, la entrada del padre pasa a la hoja de supresión y la vecina cede una entrada al nodo padre.

Árbol B, **p**=5
d= 2

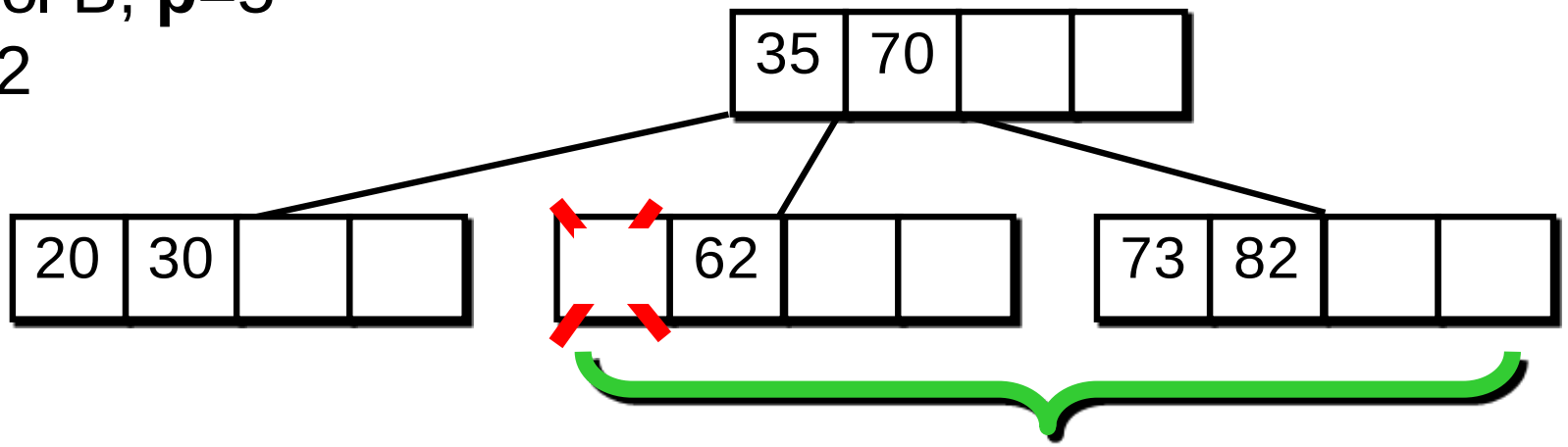


- **Ejemplo.** Eliminar 67, 45.

Árboles B

- **Ningún hermano con más de d entradas:** Con la hoja donde se hace la supresión (**d-1** entradas) más una hoja hermana (**d** entradas) más la entrada del padre, se hace una nueva hoja con **2d** entradas.

Árbol B, **p**=5
d= 2

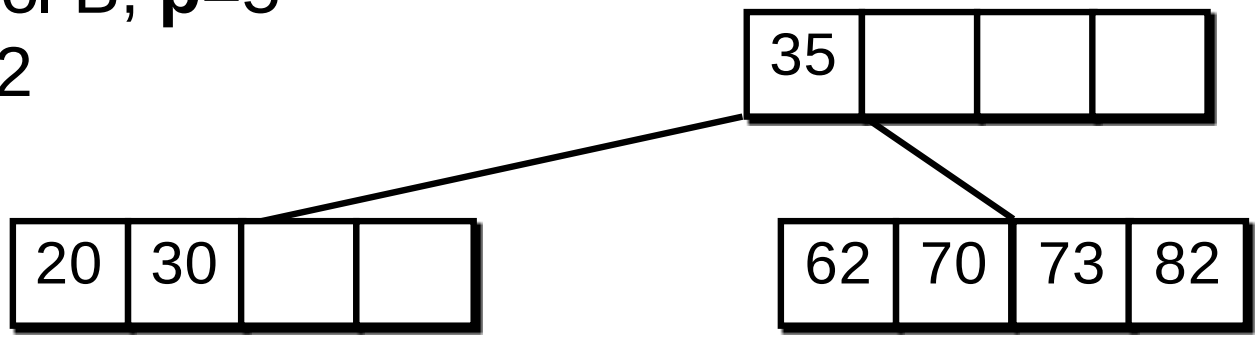


- **Ejemplo.** Eliminar 39.

Árboles B

- **Ningún hermano con más de d entradas:** Con la hoja donde se hace la supresión (**d-1** entradas) más una hoja hermana (**d** entradas) más la entrada del padre, se hace una nueva hoja con **2d** entradas.

Árbol B, **p**=5
d= 2



- **Ejemplo.** Eliminar 39.
- **Ojo:** se suprime una entrada en el padre. Se repite el proceso de eliminación en el nivel superior.

Árboles B

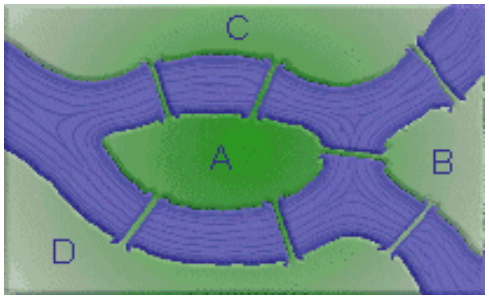
Conclusiones

- El orden de complejidad es proporcional a la altura del árbol, ~ $\log_{p/2} n$ en el peor caso.
- Normalmente, el orden **p** del árbol se ajusta para hacer que cada nodo esté en un bloque de disco, minimizando el número de operaciones de E/S.
- **Representación en memoria:** mejor usar AVL.
- **Representación en disco:** mejor usar árboles B.

Grafos

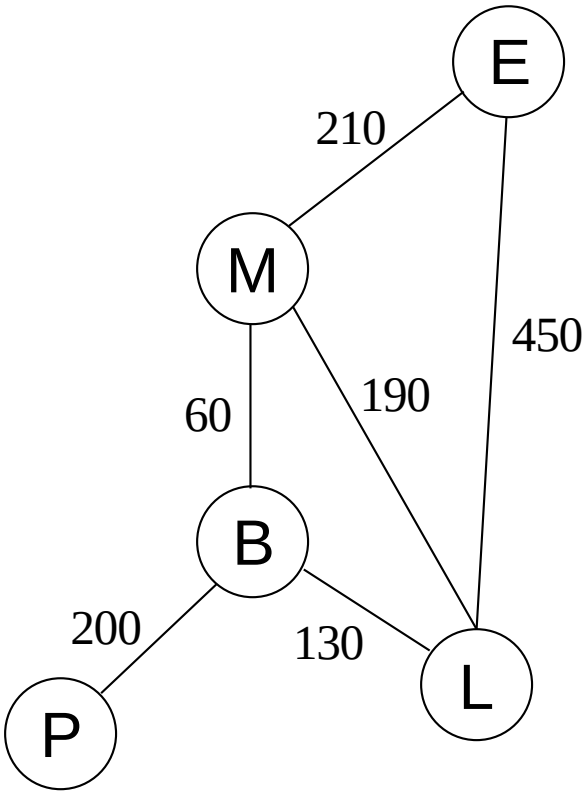
Introducción

La Teoria de Grafos nace del análisis sobre una inquietud presentada en la isla Kueiphof en Koenigsberg (Pomerania) ya que el río que la rodea se divide en dos brazos.



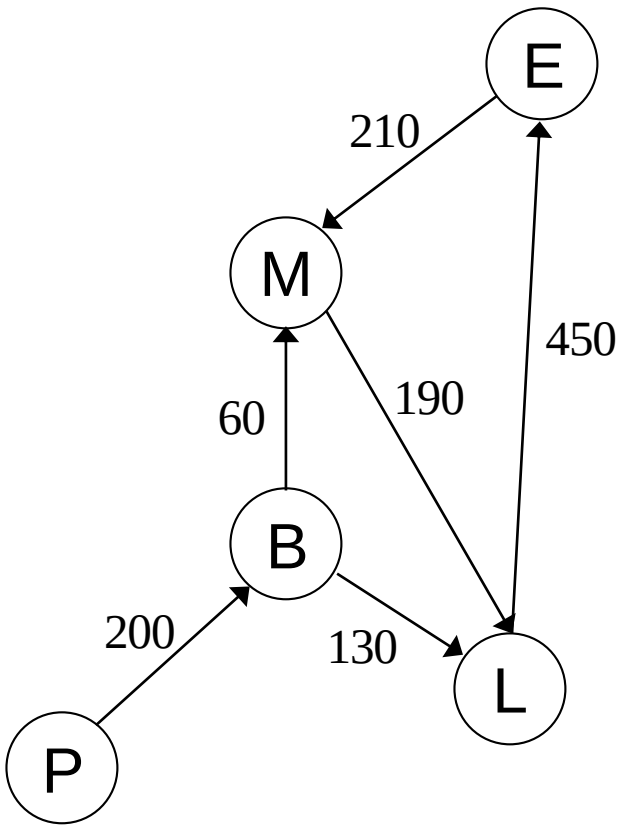
Sobre los brazos estaban construidos *siete puentes* y para los habitantes era motivo de distracción descubrir un itinerario de manera que pudieran regresar al punto de partida, después de haber cruzado por los siete puentes pero pasando sólo una vez por cada uno de ellos.

Definición



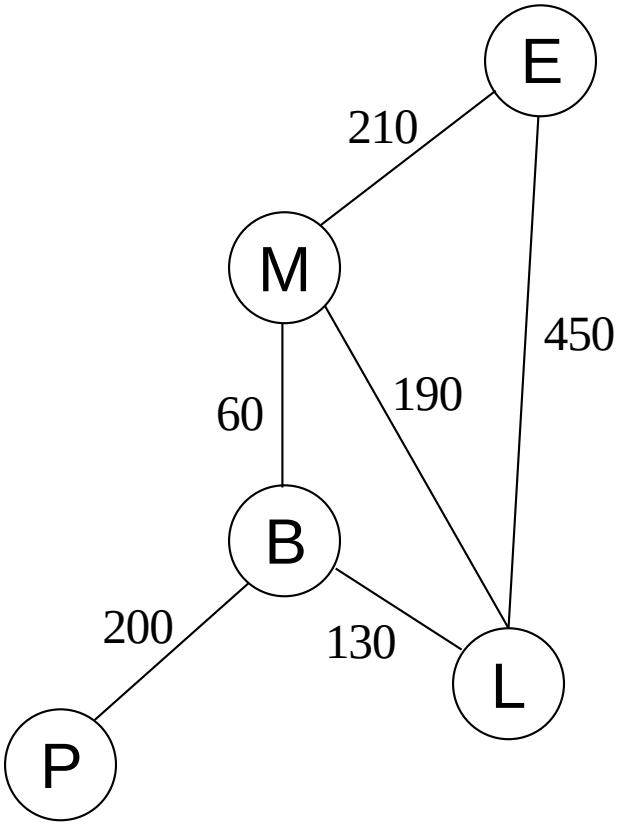
- Un grafo $G=(V,E)$ consiste de un conjunto de nodos V y un conjunto de ramas E.
- Cada nodo es definido por una etiqueta o dato.
- Cada rama es un par (v,w) donde $v,w \in V$. Las ramas pueden tener un peso y una dirección.
- El grado de un nodo indica el número de ramas conectadas a él (número de nodos adyacentes)

Grafos dirigidos



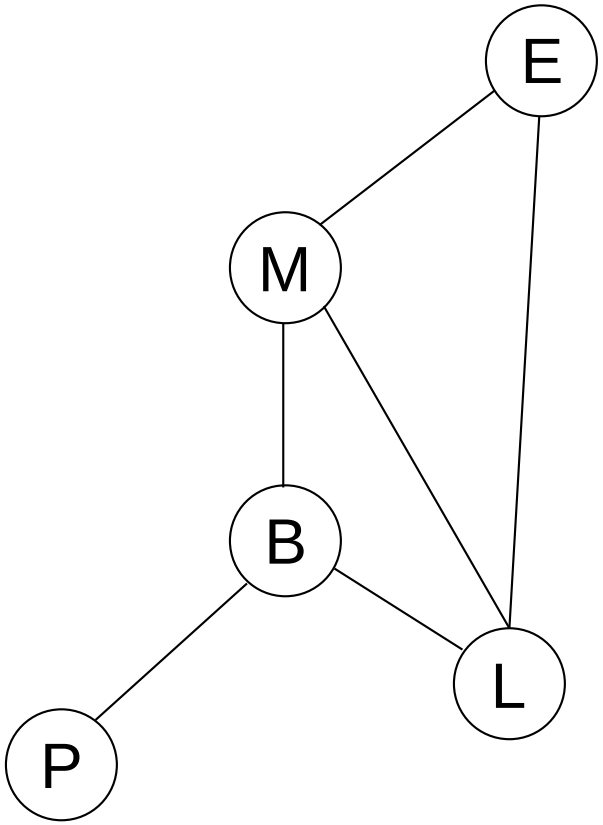
- En un grafo dirigido las ramas son pares ordenados.
  - Implica que $(u,v) \neq (v,u)$
  - "Las líneas se convierten en flechas"
- El "*indegree*" de un nodo es el número de ramas entrantes
- El "*outdegree*" de un nodo es el número de ramas salientes.

Más conceptos (1)



- Un trayectoria o recorrido es una secuencia de nodos $w_1, w_2, \ldots, w_n$ tal que $(w_i, w_{i+1}) \in E$.
- Un recorrido es una lista ordenada de nodos.
  - *Longitud*: número de ramas en el recorrido.
  - *Costo*: suma de los pesos de las ramas del recorrido
  - *Ciclo*: es un recorrido que vuelve al nodo de partida.

Más conceptos (2)

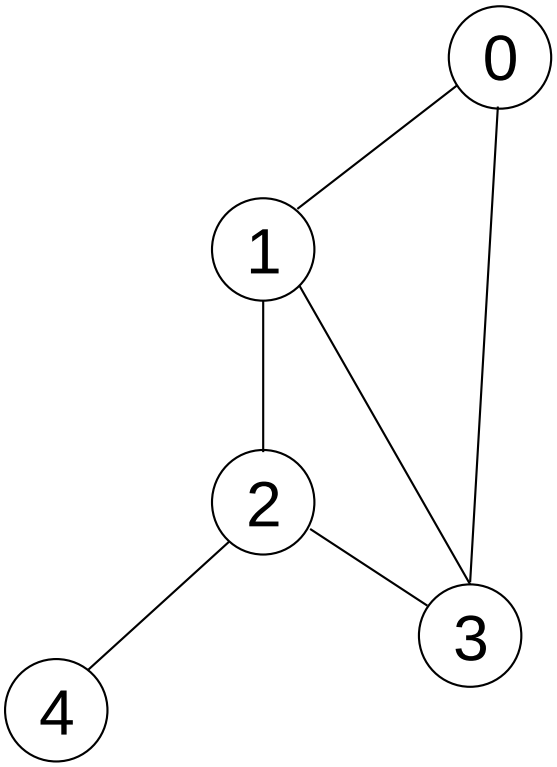


- Se dice que un grafo *no dirigido* es conectado si existe una trayectoria entre cada par de nodos.
- Por su parte, un grafo *dirigido* puede ser denso o disperso:
  - Denso: la razón entre el número de ramas y el numero nodos es grande.
  - Disperso: está razón es pequeña.

Dos representaciones

- Matriz de adyacencia
  - Tabla A de tamaño V x V, en que la que $a[i][j]$ tendrá valor 1 si existe la rama (i, j). En caso contrario, el valor será 0.
  - Cuando se trata de grafos ponderados en lugar de 1 se considera el peso de la rama.
- Lista de adyacencia
  - Se define una lista enlazada para cada nodo, que contendrá sus nodos adyacentes.
  - Si el grafo es ponderado, entonces se incluirá un peso para cada elemento de la lista.

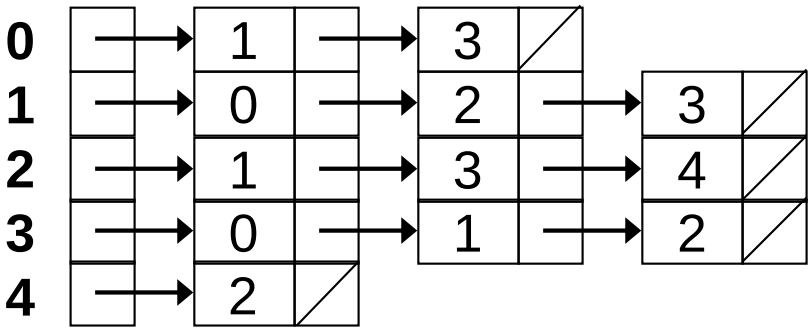
Representación de un grafo no-ponderado no-dirigido



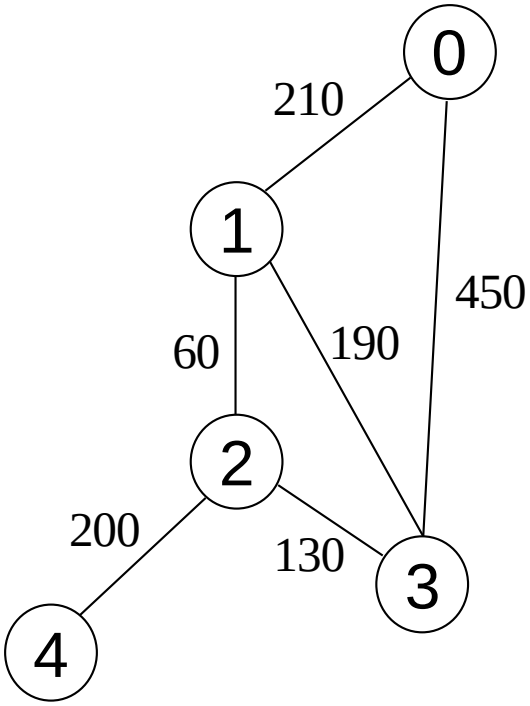
Matriz de adyacencia

	0	**1**	**2**	**3**	**4**
0	0	1	0	1	0
1	1	0	1	1	0
2	0	1	0	1	1
3	1	1	1	0	0
4	0	0	1	0	0

Lista de adyacencia

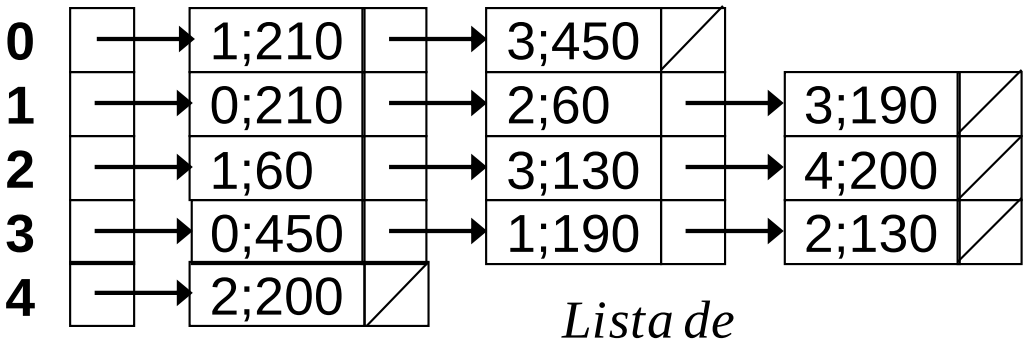


Representación de un grafo ponderado no-dirigido



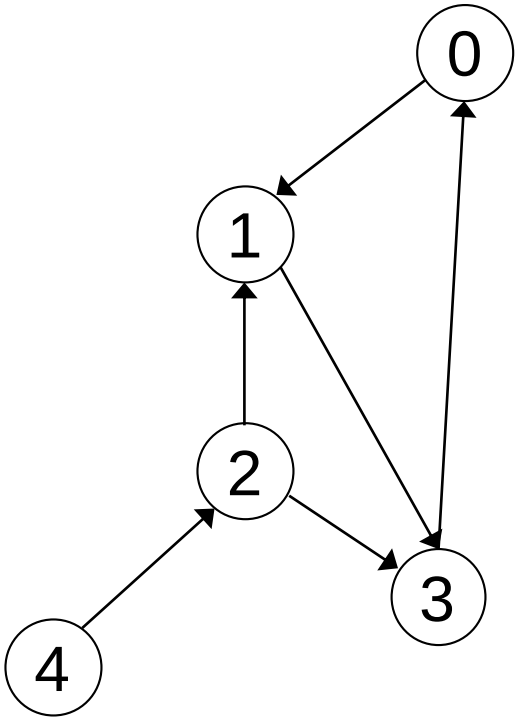
	0	1	2	3	4
0	0	210	0	450	0
1	210	0	60	190	0
2	0	60	0	130	200
3	450	190	130	0	0
4	0	0	200	0	0

Matriz de adyacencia



Lista de adyacencia

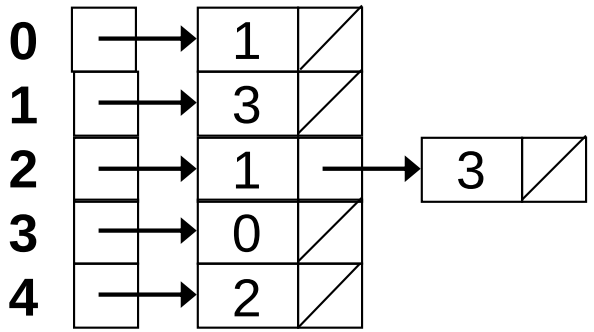
Representación de un grafo no-ponderado dirigido



Matriz de adyacencia

	0	**1**	**2**	**3**	**4**
0	0	1	0	0	0
1	0	0	0	1	0
2	0	1	0	1	0
3	1	0	0	0	0
4	0	0	1	0	0

Lista de adyacencia



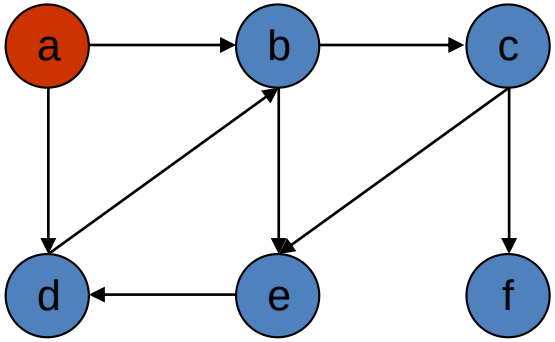
Comparación de representaciones

- Cuantitativamente (complejidad en espacio):
  - La matriz de adyacencia es $O(|V|^2)$
  - La lista de adyacencia es $O(|V| + |E|)$
- Cualitativamente:
  - La matriz es una representación estática: el grafo se construye en una sola oportunidad y es difícil de alterar.
  - La lista es una representación dinámica: el grafo se construye incrementalmente, y es muy fácil de modificarlo en tiempo de ejecución.

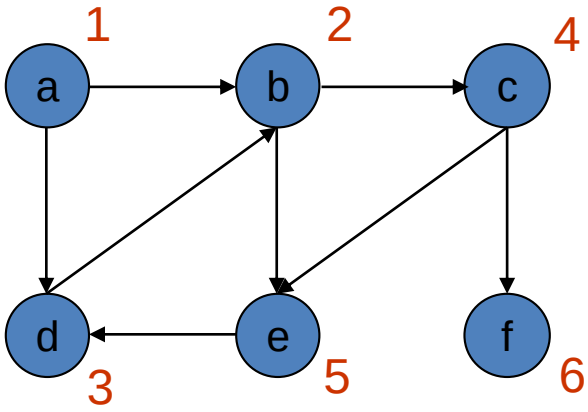
Recorridos de grafos

- Recorrer un grafo significa visitar todos sus nodos partiendo de un nodo de salida.
- Es muy importante asegurarnos de no ir en círculos (i.e., caer en un ciclo).
- Dos tipos básicos de recorridos:
  - **En anchura**: recorrer el grafo en niveles (de los nodos más cercanos a los más lejanos).
  - **En profundidad**: buscar caminos que parten desde el nodo de salida hasta que ya no es posible avanzar más, después volver atrás en busca de caminos alternativos inexplorados.

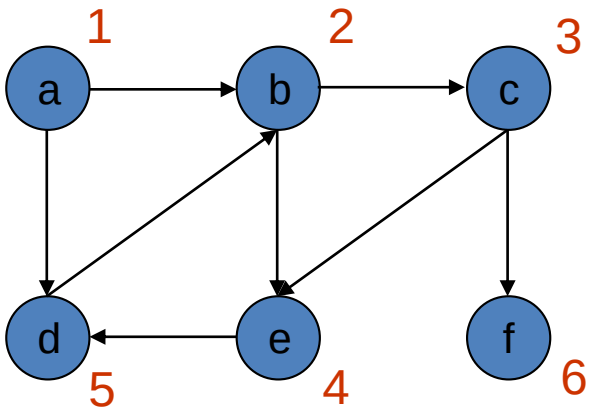
Ejemplo de recorridos



En anchura



En profundidad



Algoritmo de búsqueda en anchura

1. Para todo $v \in V[G]$ hacer $t_v \leftarrow$ infinito
2. Seleccionar un nodo de partida s
3. Hacer $t_s \leftarrow 1$
4. Meter en cola: $Q \leftarrow \{s\}$
5. Mientras $Q \neq \varnothing$
6. Sacar de cola: $u \leftarrow Q$
7. Para todo $w \in Adj[u]$ tal que $t_w =$ infinito
8. Hacer $t_w \leftarrow t_u + 1$
9. Meter en cola: $Q \leftarrow \{w\}$

t_v indica el turno de visita del nodo v

Algoritmo de búsqueda en profundidad

BúsquedaProfundidad(G, s)

- Para todo $v \in V[G]$ hacer $t_v \leftarrow$ infinito
- $turno \leftarrow 0$
- Hacer *VisitaProfundidad*(s)
- Para todo $v \in V[G]$ tal que $t_v =$ infinito
- Hacer *VisitaProfundidad*(v)

VisitaProfundidad(v)

- Hacer $t_v \leftarrow turno \leftarrow turno + 1$
- Para todo $w \in Adj[v]$ tal que $t_w =$ infinito
- Hacer *VisitaProfundidad*(v)

Problema de la "ruta más corta"

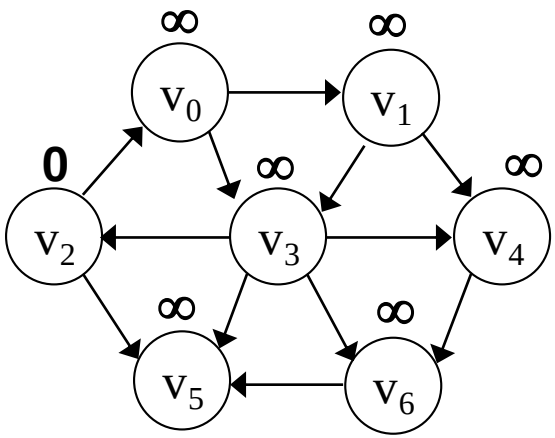
- Dado un grafo $G=(V,E)$ y un vértice s, encuentre el camino más corto desde s a todos los demás nodos en V.
- Existen dos variantes:
  - Las ramas NO tienen peso. En este caso, la longitud de una ruta es simplemente la cantidad de ramas incluidas en la trayectoria.
  - Las ramas tienen asociado un PESO. Este caso es más complicado en particular si se permiten distancias negativas. En caso de ciclos con costo negativo no existe solución.

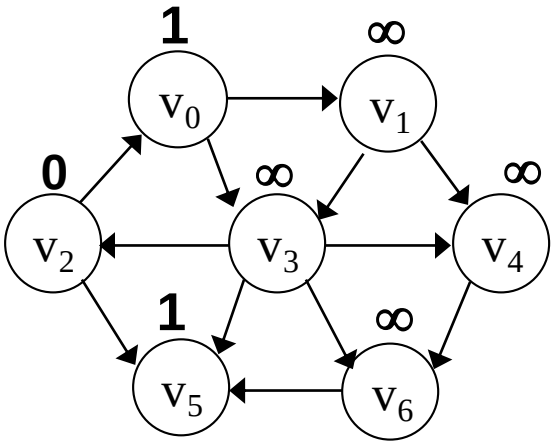
Algoritmo simple - grafo no ponderado

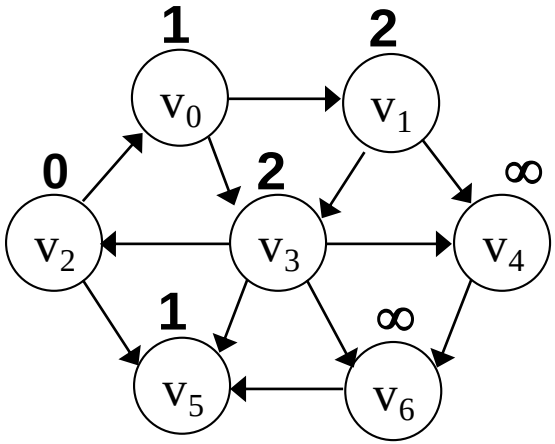
1. Para todo $v \in V[G]$ hacer $d_v \leftarrow$ infinito
2. Seleccionar un nodo de partida s
3. Hacer $d_s \leftarrow 0$ y $MasCorto = 0$
4. Para todo $v \in V[G]$ tal que $d_v = MasCorto$
5. Para todo $w \in Adj[v]$ tal que $d_w =$ infinito
6. Hacer $d_w \leftarrow MasCorto + 1$
7. $MasCorto \leftarrow MasCorto + 1$
8. Repetir *paso 4* hasta que se visiten todos los nodos

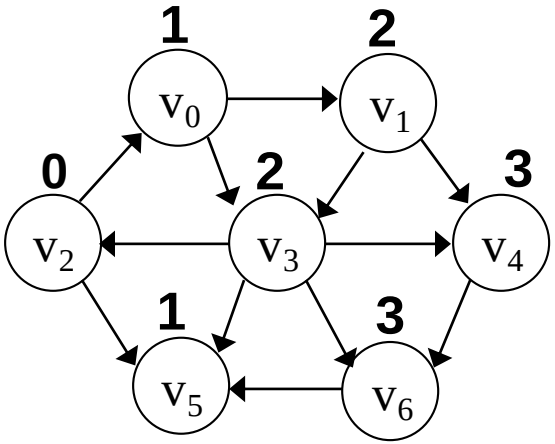
d_v indica ladistancia más corta entre s y el nodo v

Algoritmo simple - ejemplo



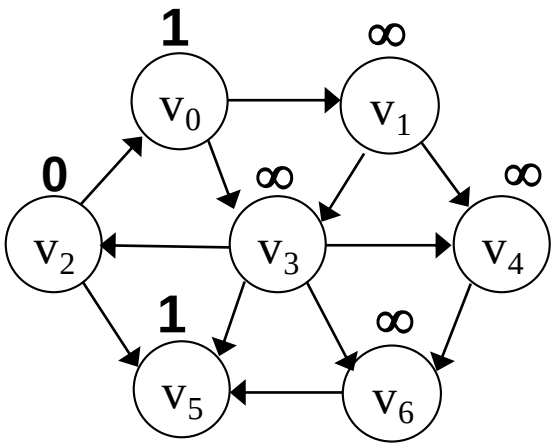




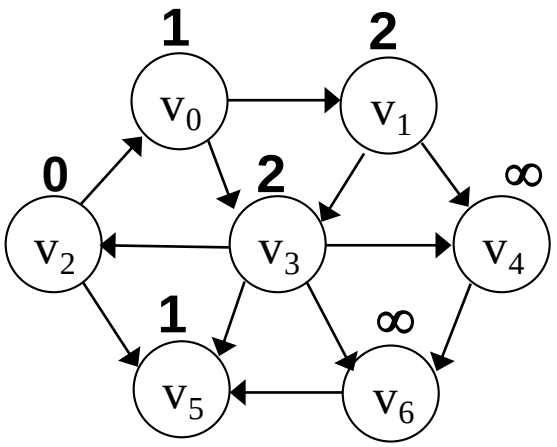


Algoritmo simple usando una cola

- En el paso 4 se busca en el grafo completo
  - Para todo $v \in V[G]$ tal que $d_v = TrayectoriaCorta$
- Una mejor solución es buscar sólo los nodos adyacentes a los nodos previamente visitados.



Queue: v_0, v_5



Queue: v_5, v_1, v_3

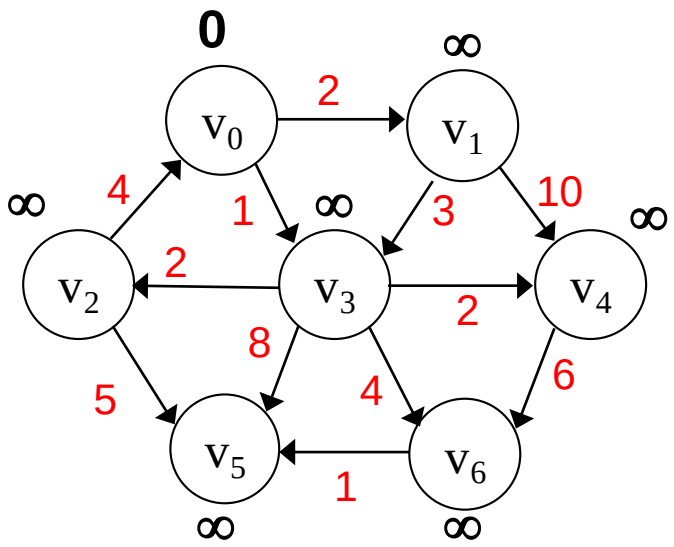
Ruta más corta en grafo ponderado

- Descubrir la trayectoria con menos costo entre un nodo de partida y todos los demás.
- El algoritmo más popular es el de Dijkstra
- Sólo trabaja con grafos cuyos pesos son positivos
- También puede aplicarse a grafos no ponderados. En este caso se considera que el peso de las ramas es 1.

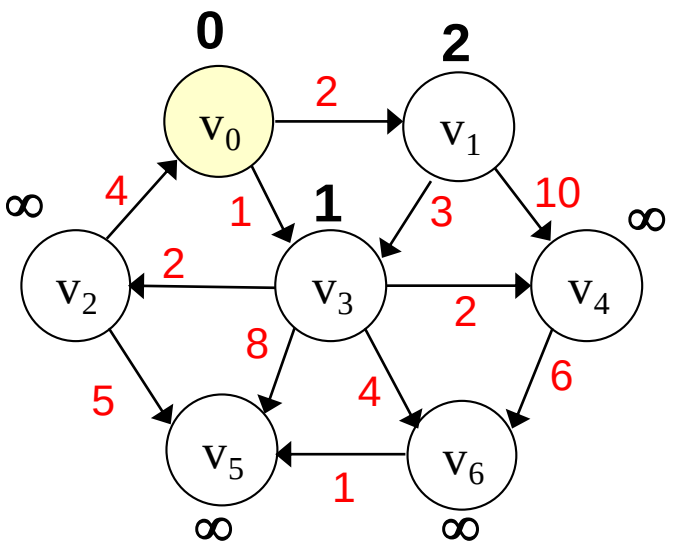
Algoritmo de Dijkstra

1. Para todo $v \in V[G]$ hacer d_v = infinito
2. Seleccionar un nodo de partida u
3. Repetir
4. Para todo $w \in Adj[u]$
5. Si costo(u) + peso(u,w) < costo (w)
6. Hacer costo(w) ← costo(u) + peso(u,w)
7. Seleccionar nuevo nodo u (con menor costo)
8. Hasta que se analicen todos los nodos

Algoritmo de Dijkstra: ejemplo (1)

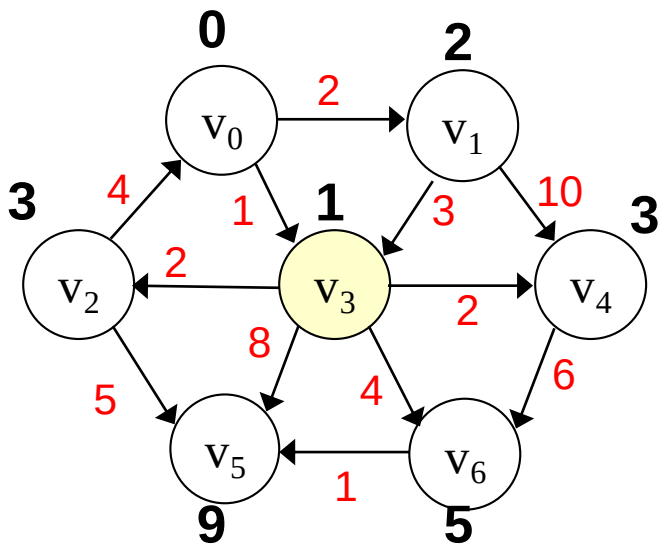


Queue: v_0

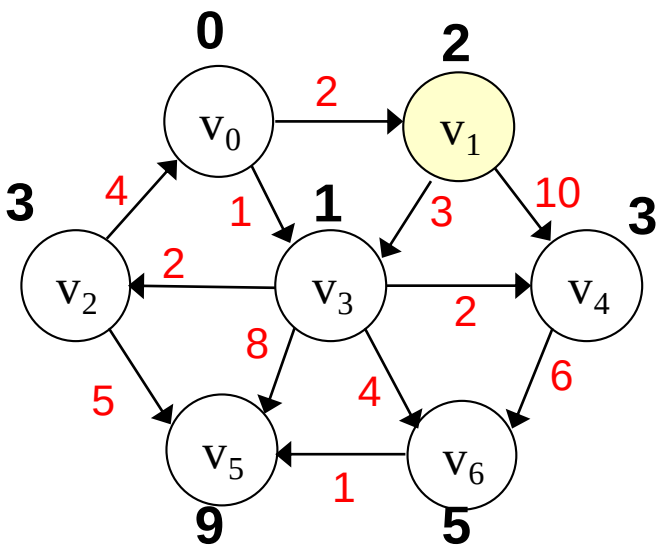


Queue: v_3, v_1

Algoritmo de Dijkstra: ejemplo (2)

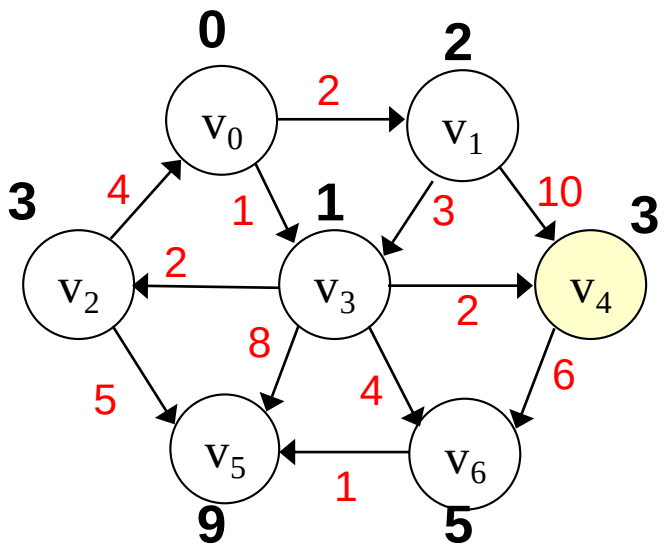


Queue: v_1, v_4, v_2, v_6, v_5

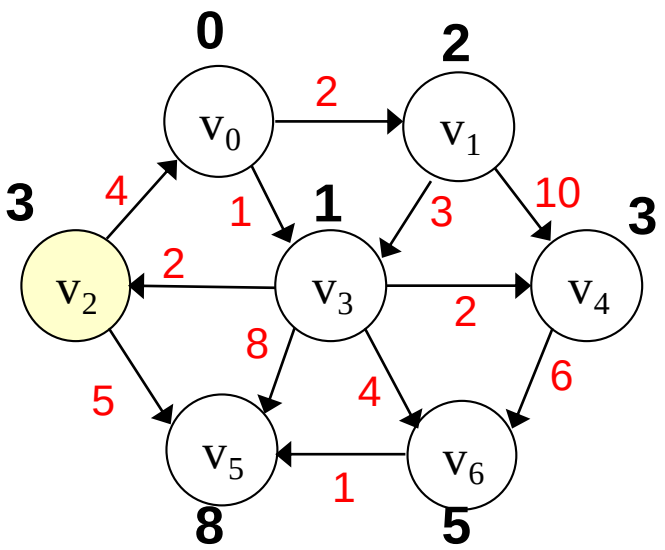


Queue: v_4, v_2, v_6, v_5

Algoritmo de Dijkstra: ejemplo (3)

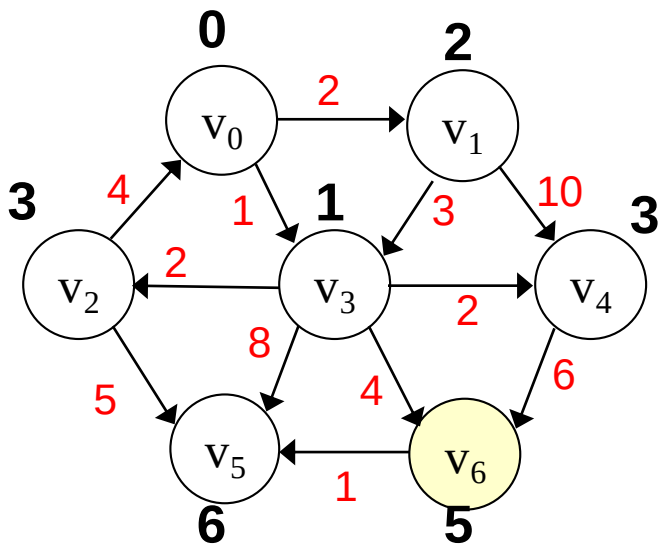


Queue: v_2, v_6, v_5

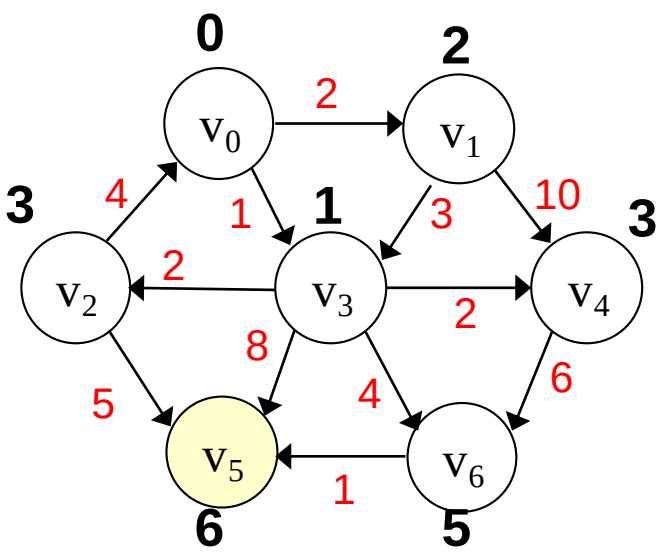


Queue: v_6, v_5

Algoritmo de Dijkstra: ejemplo (4)



Queue: v_5



Queue: