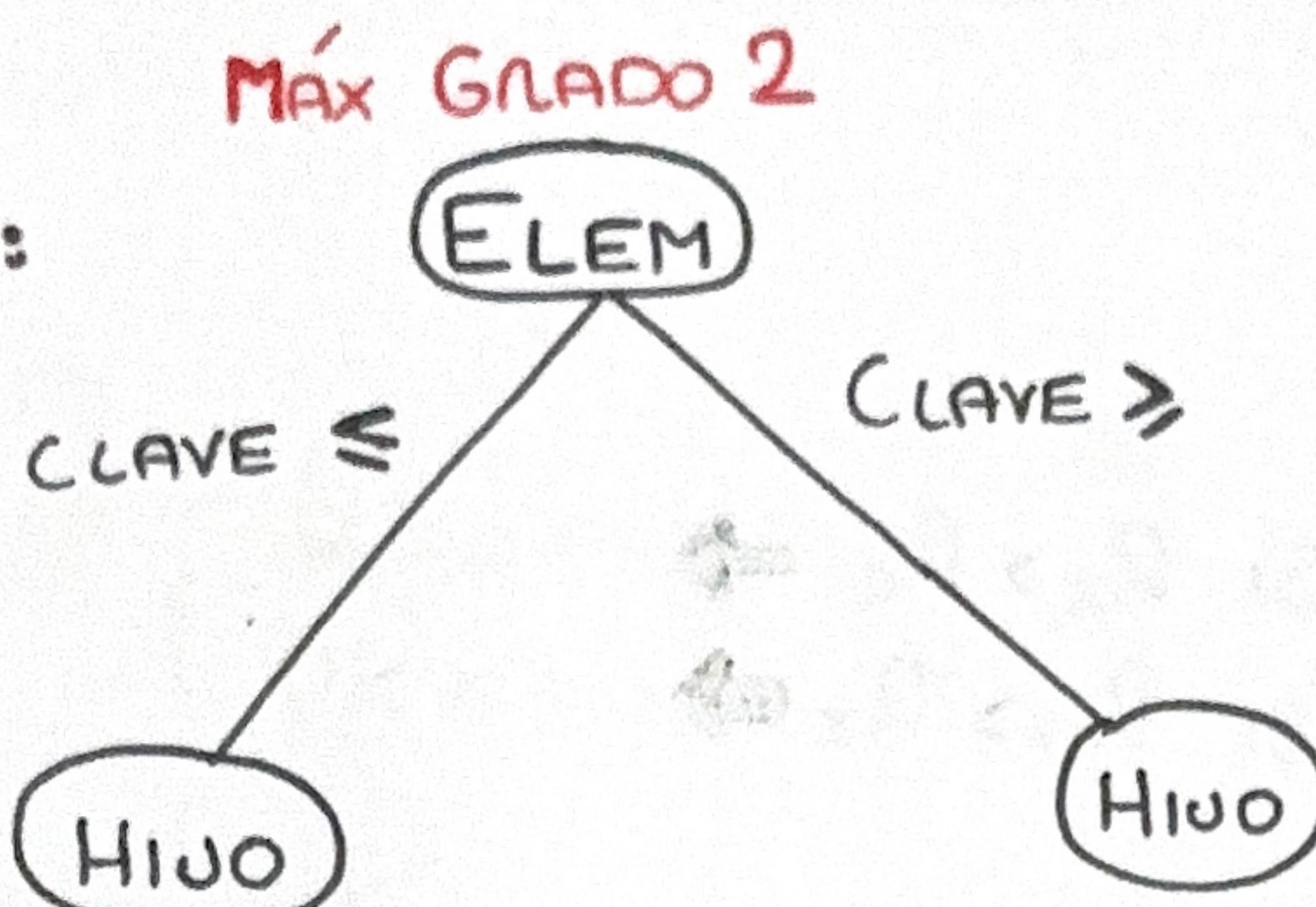


AED

ÁRBOLES BINARIOS:



UTILIZACIÓN: ÁRBOL DE EXPRESIÓN, BOOLEANOS (PROBLEMA NOCHILA)

PESO: N° NODOS HOJA

NIVEL: EMPIEZA EN 1 EN LA RAÍZ.

ÁRBOL LLENO

$N \rightarrow N(N-1)$ GRADO 2

NIVEL N GRADO 0

ÁRBOL COMPLETO

LLENO DESDE 1 A N-1

NIVEL N IZQUIERDA A DERECHA

ÁRBOL EQUILIBRADO

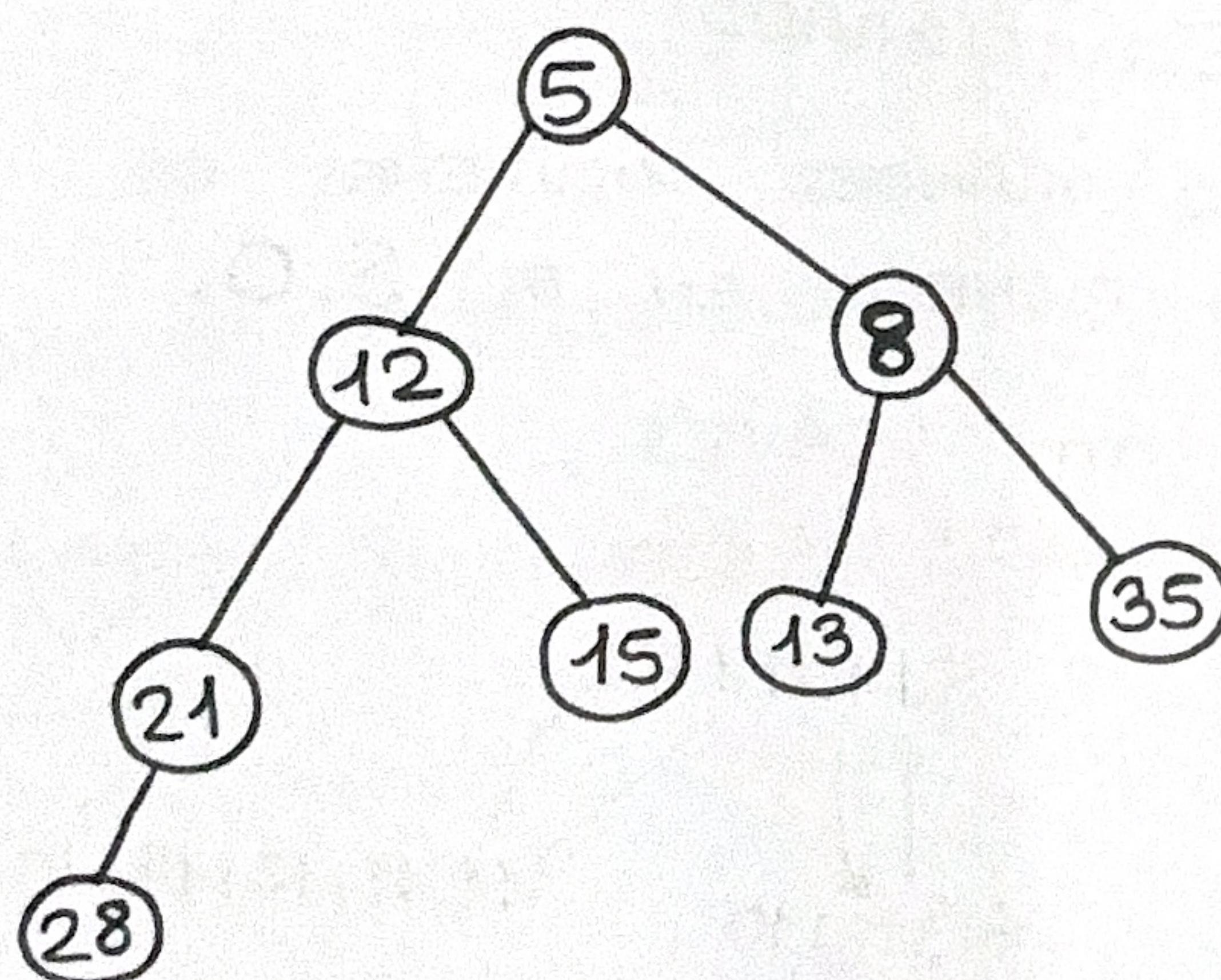
✓ NODO SU FACTOR DE EQUILIBRIO E[1,1]

(DIFERENCIA DE ALTURAS SUBÁRBOL DERECHO E IZQUIERDO)

MONTÍCULOS: COMPLETOS ✓

PARCIALMENTE ORDENADOS
(PADRE > HIJOS O PADRE < HIJOS)
MÁXIMO NIÑOS

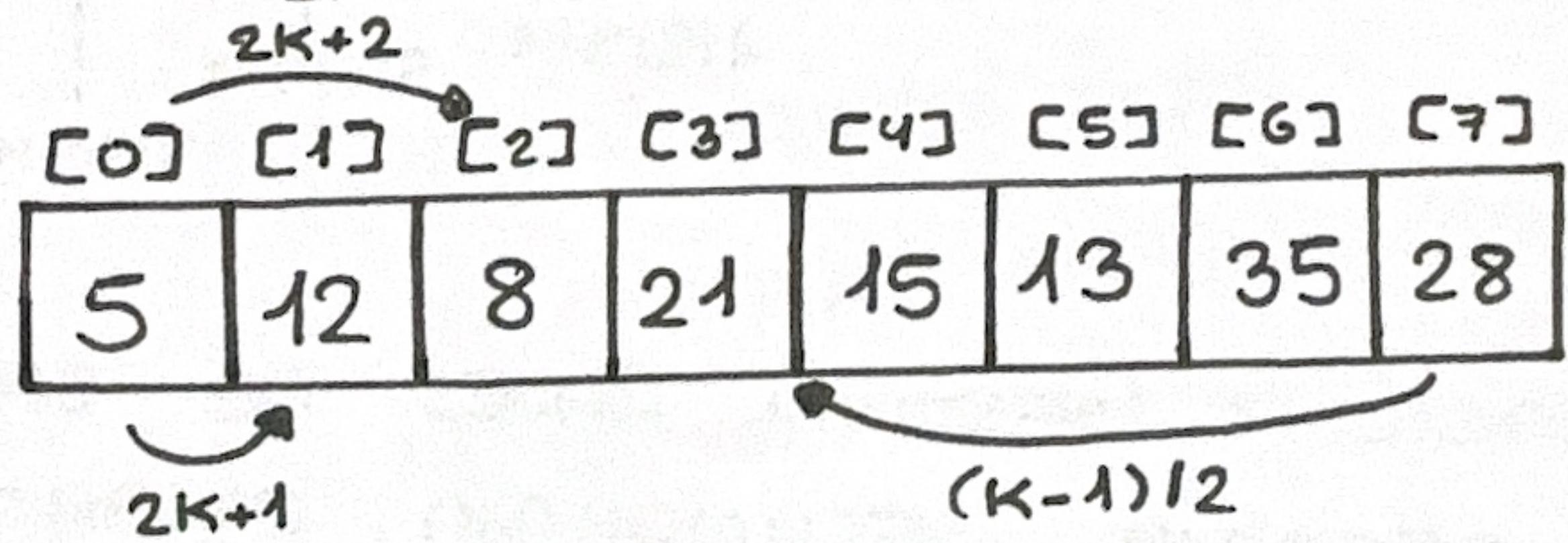
REPRESENTACIÓN ÁRBOL



+ MÁS FÁCIL MODIFICAR ✓

+ NO SE NECESITA TANANO MÁXIMO ✓

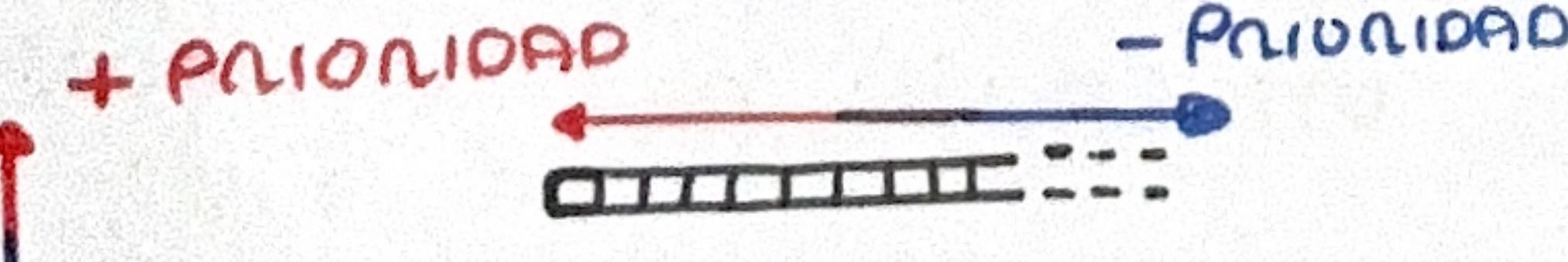
REPRESENTACIÓN ARRAY



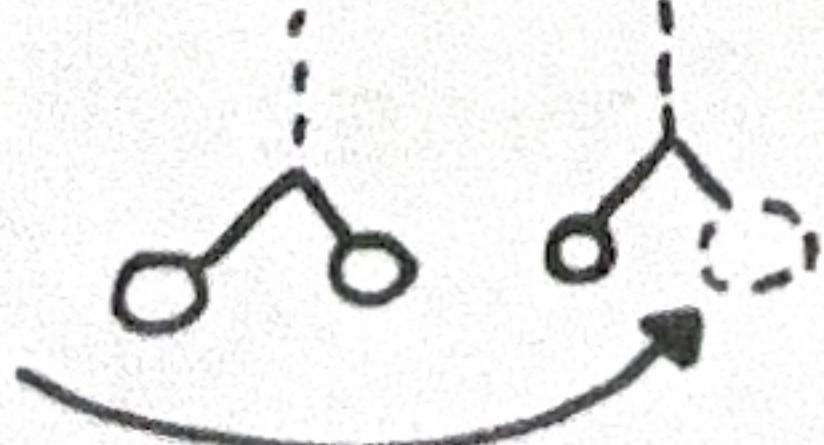
+ ACCESO MEDIANTE ÍNDICE ✓

+ AHORRO PUNTEROS ESTAN COMPLETO ✓

IDEAL PARA IMPLEMENTAR COLAS DE PRIORIDAD (CLAVE LA PRIORIDAD)

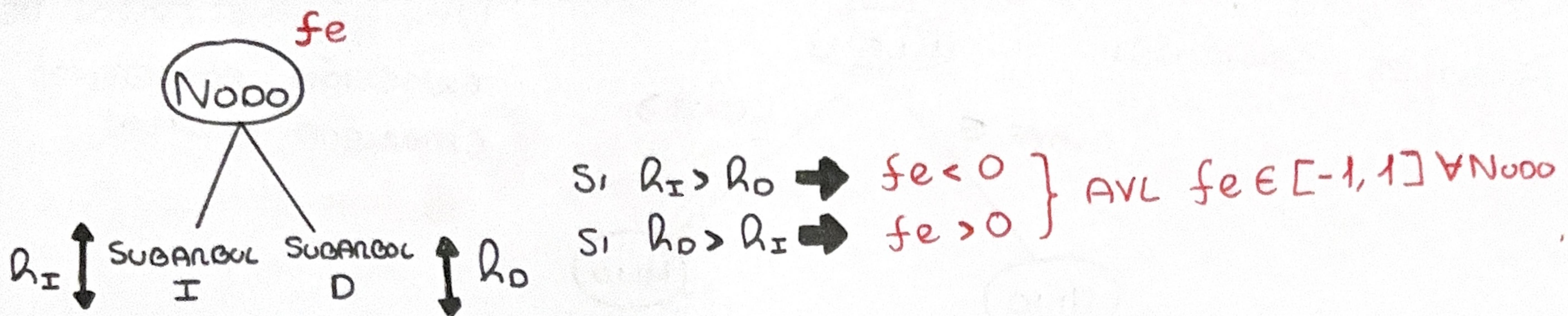


SIEMPRE SE INSERTA ÚLTIMO HUECO:



SE HACE FLOTAR: CLAVE HIJO > PADRE
(MONTÍCULO MÁXIMO)

ÁRBOLES AVL

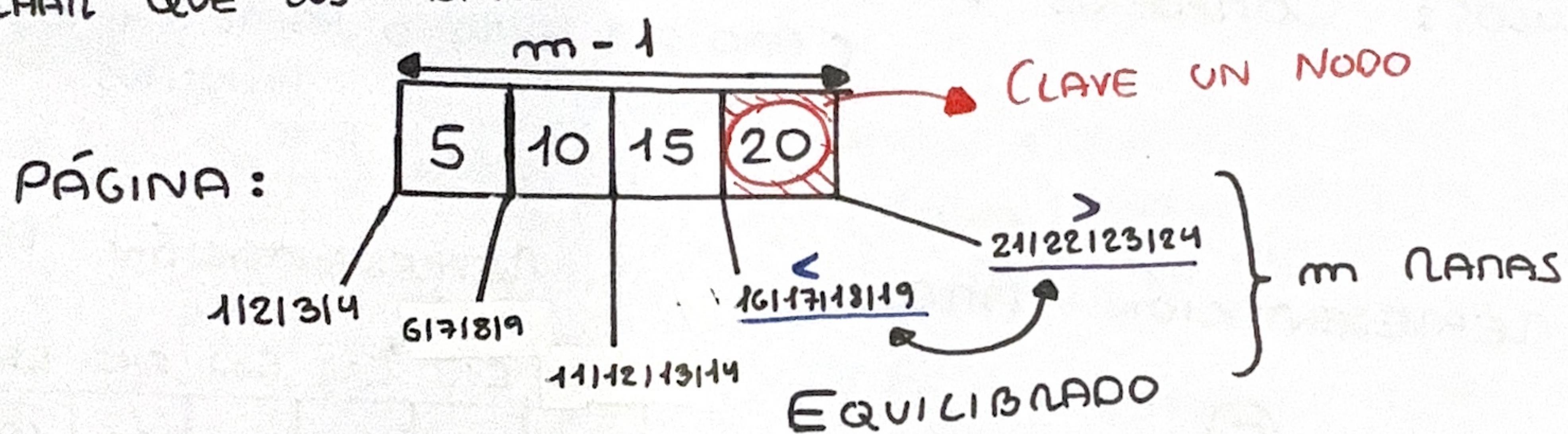


COMPONENTES CON ÁRBOL BINARIO

- + COSTO OPERACIONES INSERCIÓN (BUSCAR CAMINO NODO-RAÍZ CALCULANDO fe)
- + COSTO OPERACIONES ELIMINACIÓN (SUSTITUIR MAYOR NODO SUBARBL IZQUIERDO O MENOR SUBARBL DERECHO SI EXISTEN)
- COSTE BÚSQUEDA ÁRBOLES MUY ALTOS ($O(N)$ EN EL PEOR CASO BINARIO A $O(\log(N))$ EN EL PEOR CASO (MEJOR DE BINARIO))

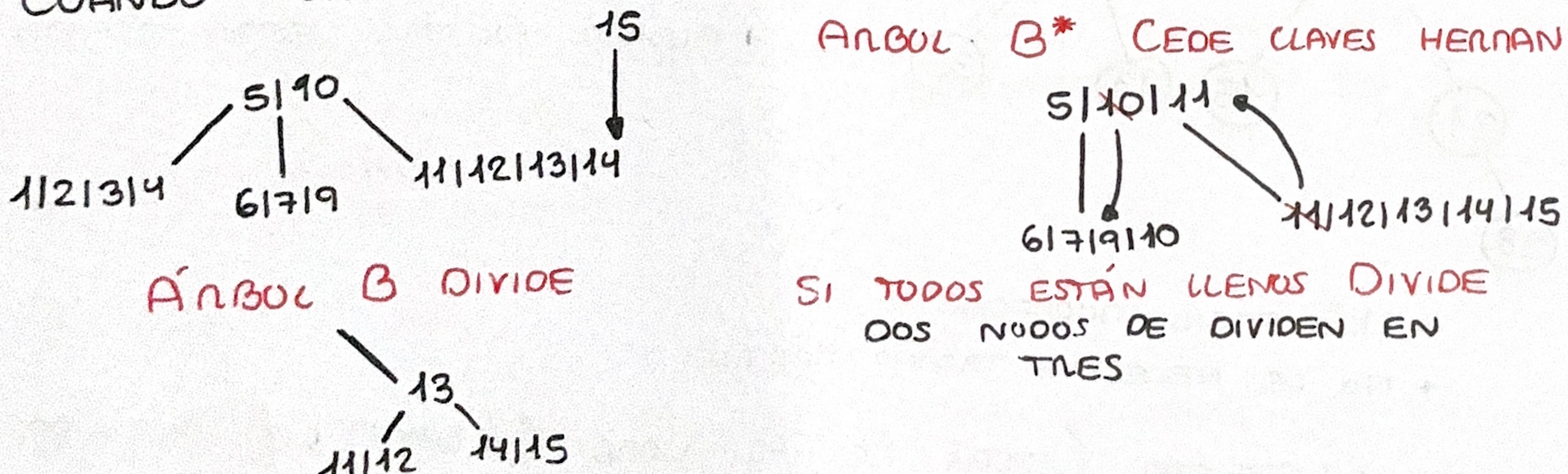
ÁRBOLES B

IDEA PRINCIPAL : AGRUPOAMOS DATOS EN BLOQUES DE HASTA m DATOS PARA APROVECHAR QUE LOS DATOS EN DISCO SE GUARDAN EN BLOQUES



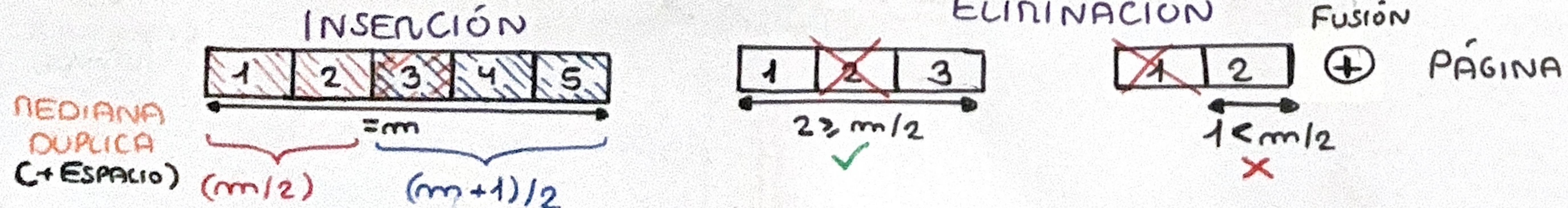
UTILIZACIÓN: GESTIÓN BASES DE DATOS O GRANDES VOLÚMENES DE DATOS EN MEMORIA SECUNDARIA, ORGANIZACIÓN FICHEROS EN EL S.O.

CUANDO UN NODO LLEGA A m CLAVES.



ÁRBOLES B*: + PROPORCIÓN UTILIZACIÓN DE LOS NODOS.

ÁRBOLES B+: OPTIMIZACIÓN TODAS LAS CLAVES ESTÁN EN LAS HOJAS Y LAS INTERNAIAS SOLO SON ÍNDICES.



EN B^+ NO HAY QUE REESTRUCTURAR EL ÁRBOL NI ACTUALIZAR CLAVES INTERNAIAS, SOLO FUSIONAN PÁGINAS Y ELIMINAN CLAVES SEPARADORAS.

B^+

- COSTE COMPUTACIONAL
- + ESPACIO OCUPADO

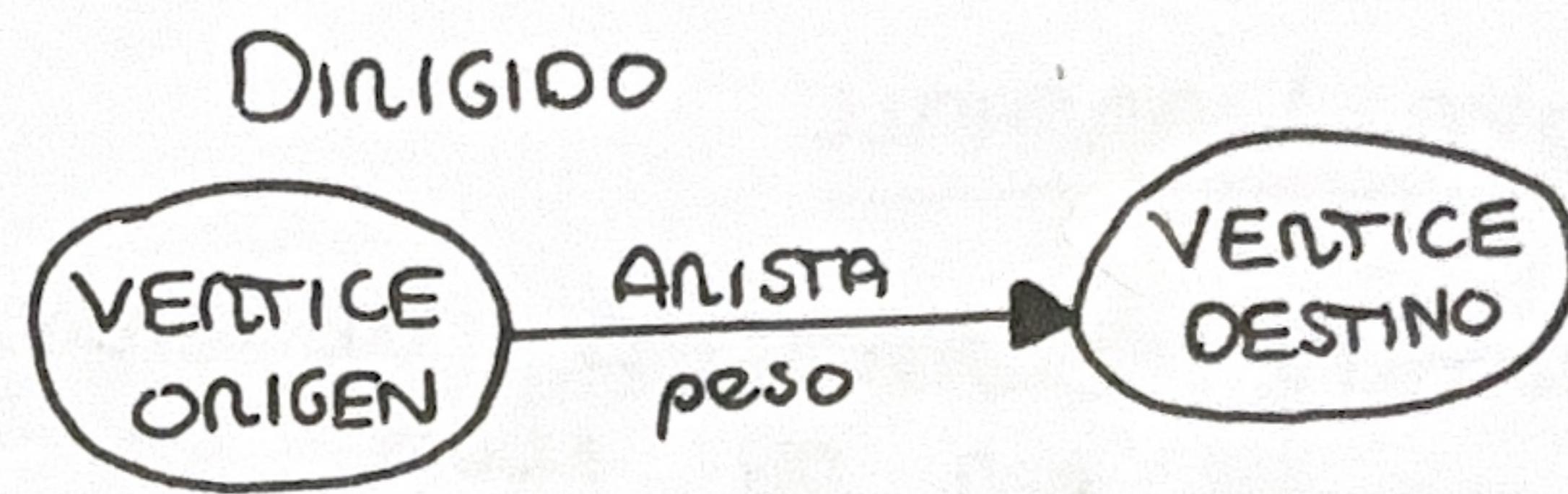
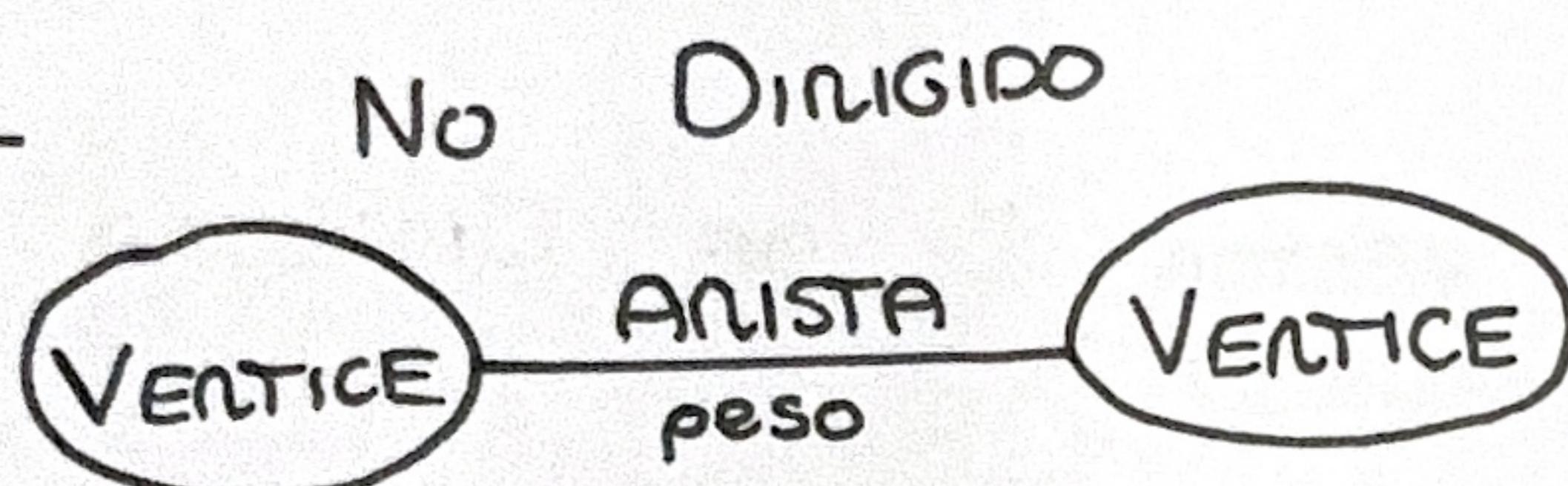
B^*

- ACCESOS A DISCO (PAPA UTILIZACIÓN)

VS
 B (CUALQUIER TIPO) / AVL o ABB

- ACCESOS A DISCO
- ↑ NUEVOS VOLÚMENES GRANDES DATOS
- COMPACIÓNES

GRAFOS

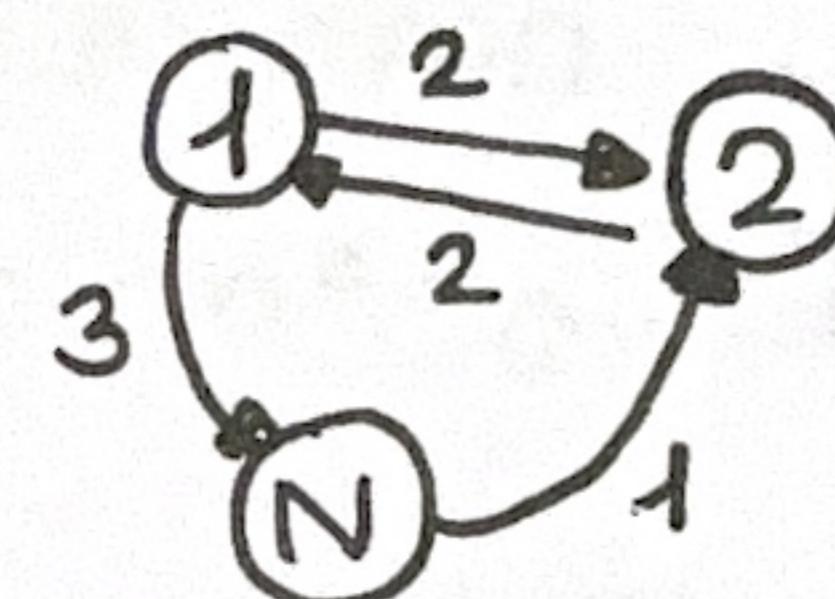


Dos posibles REPRESENTACIONES:

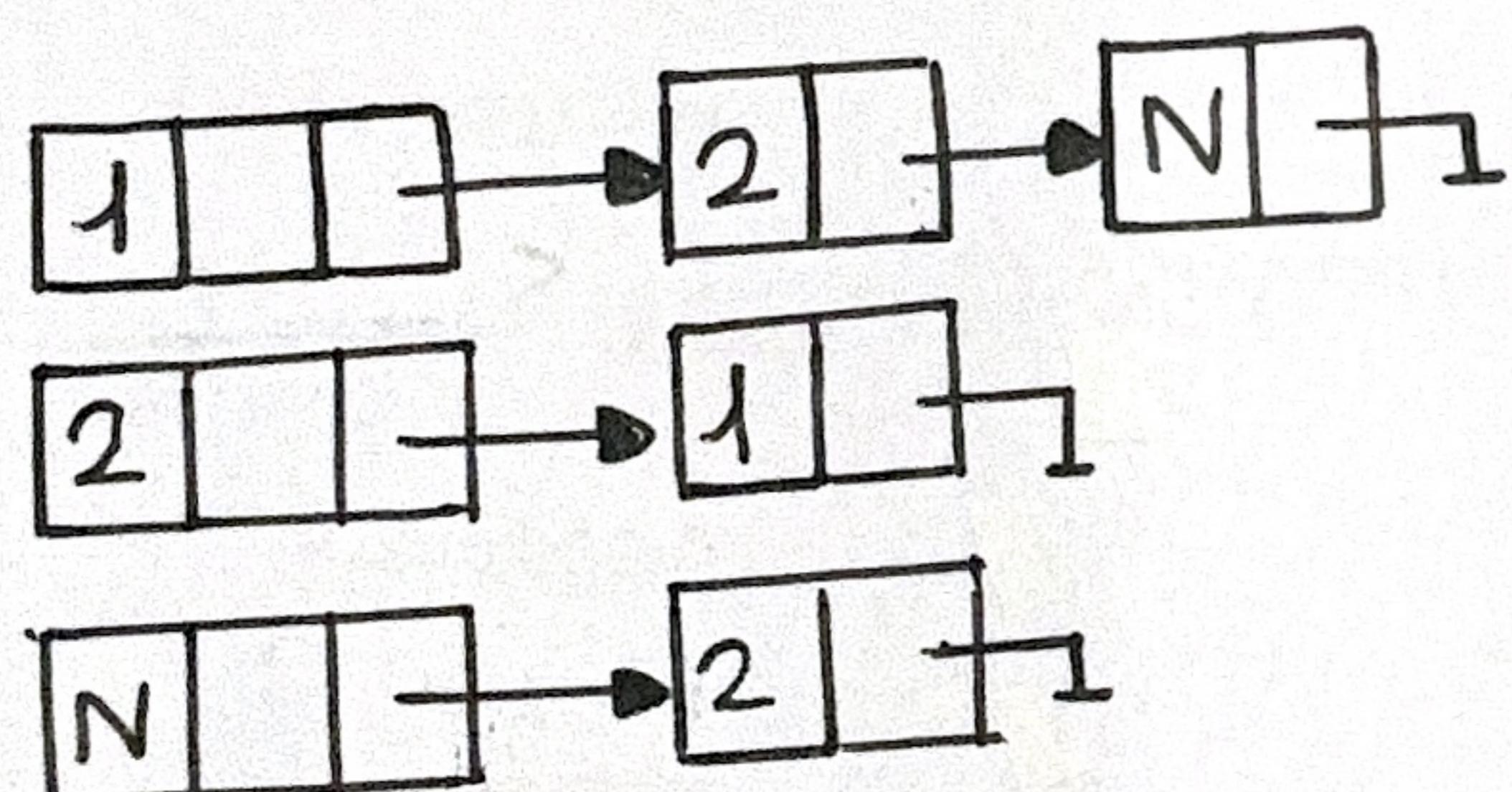
MATRIZ ADYACENCIA :

$$\begin{pmatrix} v_1 & v_2 & \dots & v_m \\ v_1: & 0 & 2 & \dots & 3 \\ v_2: & 2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_m: & 0 & 1 & \dots & 0 \end{pmatrix}_{N \times N}$$

- ✓ El TAMAÑO DE $\{v\}$ ES cte
- ✗ HAY MUCHOS 0 POR EN MEDIO



LISITAS ADYACENCIA



- ✗ DIFÍCIL ENCONTRAR ARISTAS (RECORRER LISTA)
- +/- $O(N+\alpha)$ CONSUMO MEMORIA DONDE $\alpha = |A|$ VS $O(N^2)$ DE MATRIZ

$$\left(\frac{E}{V^2} \right) \rightarrow \text{DENSIDAD GRAFO}$$

- SI ES PEQUEÑA, DISPERSO LISTA ✓
- SI ES GRANDE, NO DISPERSO LISTA ✗

PUNTOS DE ARTICULACIÓN



CON A: 1 COMPONENTE CONEXA
SIN A: 2 COMPONENTES CONEXAS

→ CALCULA ÁRBOL REBALIZANDO A TRAVÉS DE UN RECORRIDO RECURSIVO EN PUNTO FUNDIDO

ALGORITMOS

WANSHALL: OBTENCIÓN MATRIZ CAMINOS EFICIENTE $P_K = P_{K-1} \times \dots \times P_0$
 ADYACENCIA

$$\hookrightarrow P_K(i,j) = P_{K-1}(i,j) \text{ OR } [P_{K-1}(i,K) \text{ AND } P_{K-1}(K,j)]$$

DIJKSTRA: CAMINO MÁS CORTO DE ORIGEN A LOS OTROS VERTICES GRAFO

↪ ALGORITMO Voraz Clásico

C = CONJUNTO VERTICES CANDIDATOS (CONVIENZA SIENDO $G - \{\text{ORIGEN}\}$)

S = CONJUNTO DE VERTICES SELECCIONADOS

D = VECTOR DISTANCIAS ORIGEN → VERTICE DE G / P = VECTOR PREDECESORES EN EL CAMINO DISTANCIA MÍNIMA PARA LLEGAR VERTICE DESTINO i.

A = MATRIZ DE ADYACENCIA

SELECCIONA i VÉRTICE CON $D(i)$ MÍNIMO. $i \in S$, $i \notin C$, ACTUALIZA $D(j)$ PARA $j = i$ COMO $\min(D(j), D(i) + A(i, j))$.
IN DIRECTAMENTE PASAR POR i

SEGUIMOS $m-1$ VECES SACANDO VÉRTICES DE C HASTA QUE $S = G$ Y $C = \emptyset$
(SUPONGO QUE SI UN VÉRTICE NO TIENE ENTRANTES LA DISTANCIA HASTA EL SERÁ INFINITA AL FINAL DEL ALGORITMO)

FLOYD-WARSHALL

- MATRIZ DISTANCIAS GENERALIZACIÓN VECTOR D DE DIJKSTRA PARA CUALQUIER PAR DE NODOS SIN FIJAR ORIGEN.
- APLICACIÓN MATRICES BOOLEANAS FLOYD A VALORES NUMÉRICOS.
 $D(i,j) = \min[D(i,j), D(i,k) + D(k,j)]$

FORD-FULKERSON

UTILIZACIÓN: CALCULAR Y DISTRIBUIR FLUJO COMUNICACIONES INTERNET, TRÁFICO EN LA VÍA PÚBLICA...

OOS NODOS: $T \rightarrow S$
FUENTE ORIGEN SUMIDERO DESTINO

BUSCA CAMINOS $T \rightarrow S$ V i, j VÉRTICES INTERMEDIOS

CADA ARCO

- REDUCIBLE
- INCREMENTABLE
- NO MODIFICABLE

INCREMENTABLE
 $F_{i,j} < U_{i,j}$
 FLUJO CAPACIDAD

REDUCIBLE
 $F_{i,j} > 0$
 EL FLUJO PUEDE BAJAR

PRIM Y KRUSKAL



OBJETIVO: ENCONTRAR EL ÁRBOL EXPANSIÓN MÍNIMO

PRIM

$G(V, A)$; $W = \{i\} / i$ VÉRTICE NADIE

SELECCIONA $v \in V$ ANO $v \notin W$
DONDE ARCO (u, v) MÁS CORTO $u \in W$

FIN: $V = W$

Solo PARA GRAFOS NO DIRIGIDOS

KRUSKAL

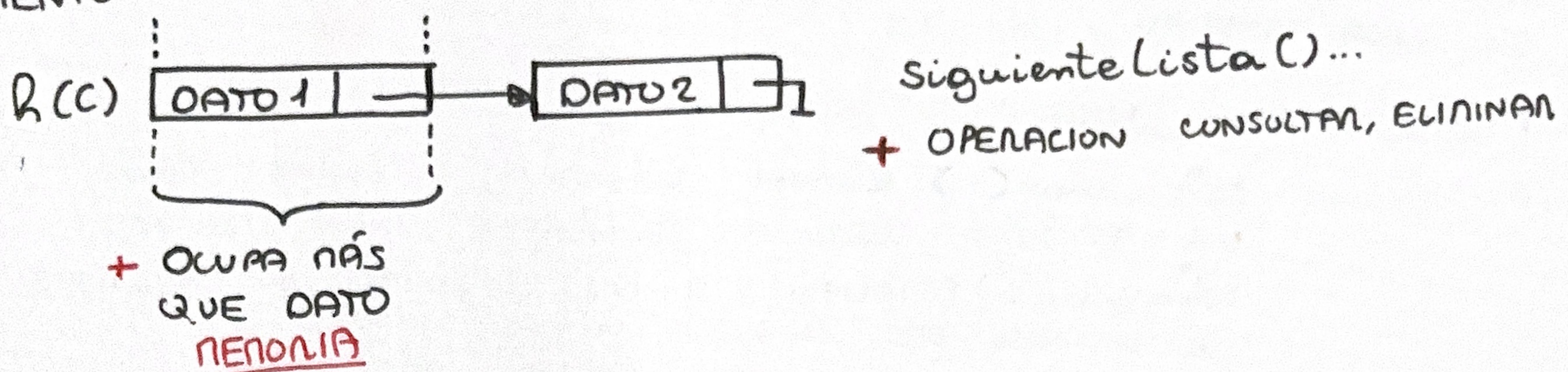
$G = (V, A)$ $T = V$

ORDENA A DE MÍNIMO A MÁXIMO
SE AÑADE $e_{i,j}$ SI une OOS COMPONENTES CONEXAS DISTINTAS

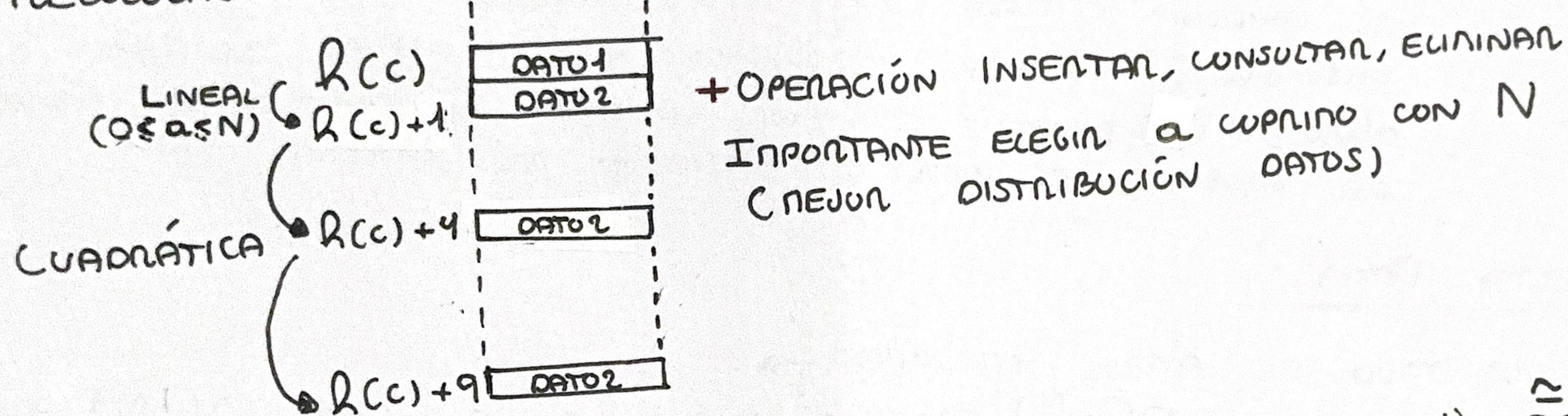
FIN: T solo 1 COMPONENTE CONEXA

TABLAS HASH

ENCADENAMIENTO



RELOCACIÓN

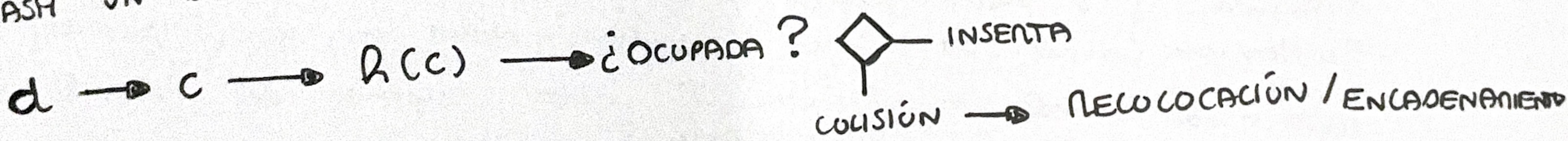


PROPIEDADES: INSERCIÓN CASI $O(1)$, ACCESO ALEATORIO TIEMPO "CONSTANTE"

FUNCIÓN HASH IMPORTANTE: COMPROMISO:

MUY SOFISTICADA	NADA SOFISTICADA
+ COSTE COMPUTACIONAL	+ RÁPIDA Y SENCILLA
- COLISIONES	+ COLISIONES

HASH: VALOR IDEALMENTE ÚNICO UTILIZADO PARA INDEXAR EN UNA TABLA. HASH UN DATO EN LA POSICIÓN IDENTIFICADA POR EL HASH DE LA CLAVE.



FACTOR DE CARGA $L = m/n$ ≈ "CANTIDAD DATOS POR POSICIÓN EN LA TABLA HASH" → DEFINE QUE TAN SUSCEPTIBLE VA A SER A LAS COLISIONES.

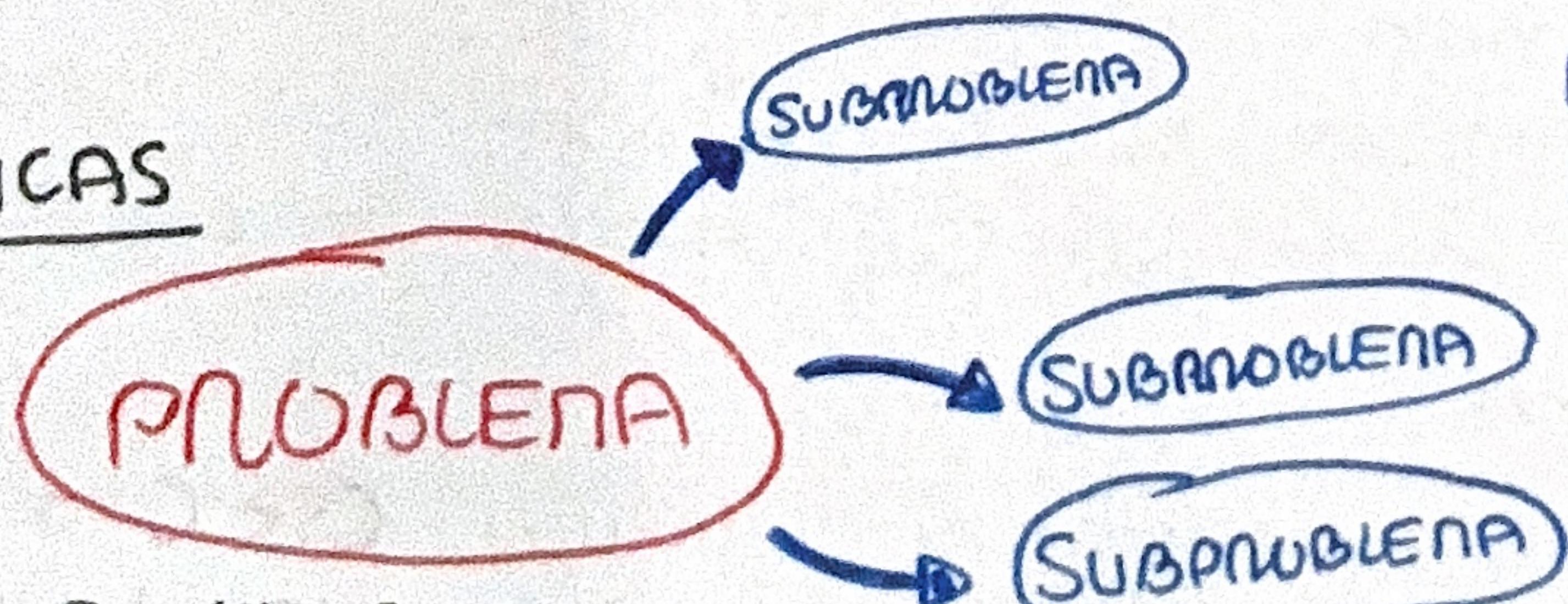
LÓPTIMOS: 0,75 (ENCADENAMIENTO), 0,5 RELOCACIÓN

- + TABLA MÁS PEQUEÑA
- + SUSCEPTIBLE COLISIONES

ESTRATEGIAS ALGORÍTMICAS

1. DIVIDE Y VENCERÁS

EJEMPLO: PRÁCTICA PROCESOS E HIJOS DE SISTEMAS OPERATIVOS I.



FÁCILES DE RESOLVER
FÁCILES DE AGUARDAR
DIFÍCIL DE RESOLVER
FÁCIL DE DIVIDIR

2. VORAZ

ENCUENTRA LA SOLUCIÓN PRODUCTO DE ELEGIR LAS MEJORES SOLUCIONES PARCIALES EN CADA MOMENTO, NO LA MEJOR SOLUCIÓN

EJEMPLOS: PRIN, DIJKSTRA, KRUKSAL

FUNCIONES: seleccionar(C) EXTRAE LA MEJOR SOLUCIÓN PARCIAL A TRAVÉS DE LOS CANDIDATOS RESTANTES, DEVUELVE EL MEJOR CANDIDATO.

solución(S) COMPRUEBA SI UN CONJUNTO DE CANDIDATOS CUMPLE LAS RESTRICCIONES DEL PROBLEMA Y ES SOLUCIÓN

factible(S, x) INDICA SI EL CONJUNTO DE CANDIDATO S AL AÑADIRLE EL CANDIDATO x PUEDE LLEGAR A SER SOLUCIÓN.

insertar(S, x) AÑADE EL CANDIDATO x AL CONJUNTO SELECCIONADO S .

objetivo(S) DEVUELVE EL VALOR ASIGNADO AL CONJUNTO DE CANDIDATOS DEL PROBLEMA, ES DECIR LA ASIGNACIÓN DEL PROBLEMA.

3. VUELTA ATÍAS

- EXPLORA TODO EL ÁRBOL SOLUCIONES → ENCUENTRA SOLUCIÓN ÓPTIMA.
- coste computacional $O(m^n)$ PARA ÁRBOLES m -ANOS, COSTE $O(!n)$ PARA PERMUTACIONALES Y COSTE $O(b^d)$ ÁRBOLES COMBINATORIOS.
- VECTOR USADAS (PROBLEMA ASIGNACIÓN NO PERMITE REPETICIÓN TAREAS)
- FUNCIONES
 - generar(nivel, S): modifica la asignación de S para la variable nivel → crea un nuevo nodo en dicho nivel
 - solución(nivel, S): comprueba si la solución S es una solución completa al problema y es válida.
 - criterio(nivel, S): comprueba si el nodo existe en el árbol de soluciones y S puede llegar a ser una solución posible.
 - noHermanas(nivel, S): indica si quedan nodos por generar en un cierto nivel.
 - retroceder(nivel, S): retrocede al nivel superior del árbol al que se encontraba S .

4. PLANIFICACIÓN PODA

- ELEGIR ESTRATEGIA
 - MAXIMIZACIÓN (PB) O MINIMIZACIÓN (LB)
 - NODES + PROFUNDOS PRIMERO (LIFO) + SUPERFICIALES PRIMERO (FIFO)
- DEFINIR PRECISIÓN COTAS
 - PERMITE ACOTAR EL ÁRBOL SOLUCIONES A UN SUBÁRBOL MENOR.
 - PLANIFICAR UN NODO SI $CS < \min(S, CI_i)$ PARA MINIMIZACIÓN O $CI > \max(S, CS_i)$ DONDE S ES LA SOLUCIÓN MÁS PROFUNDORA HASTA EL MOMENTO Y CS_i O CI_i ES LA COTA SUPERIOR O INFERIOR DEL NODO MÁS PROFUNDOR. ESE FACTOR SE LLAMA VARIABLE DE COTA DE PODA (C).
- LOS NODOS PLANIFICADOS SI CUMPLEN $CS < C$ O $CI > C$ PARA MINIMIZACIÓN O MAXIMIZACIÓN SE INSERTAN A LA LNV, UNA COLA CON PRIORIDAD ORDENADA POR BE (ASCENDIENTES EN MIN Y DESCENDIENTES EN MAX)

ACABA CUANDO LNV ESTÁ VACÍA