

def busqueda_binaria (aneglo, inicio, fin, elem): Close 3

if inicio > fin: return -1

medio = (inicio + fin) // 2

if (aneglo[medio] == elem): return medio

if (aneglo[medio] > elem):

return busqueda_binaria (aneglo, medio + 1, fin, elem)

else:

return busqueda_binaria

(aneglo, inicio, medio - 1, elem)

Complejidad \rightarrow Notación Big Oh

Tiempo: medida cualitativa?

$T(n) = \mathcal{O}(f(n))$ si $\exists c \in \mathbb{R}, m \in \mathbb{N} / T(n) \leq c \cdot f(n)$

= $\Omega(f(n))$ "mínimo" $\forall n \geq m$

= $\Theta(f(n))$ " $T(n) = \mathcal{O}(f(n)) \wedge T(n) = \Omega(f(n))$ "

Por una forma cualitativa a medida que va creciendo

$\mathcal{O}(1)$

def foo (int a, int b): operaciones de
 + constante
 return a+b; } $\mathcal{O}(1)$ es $\mathcal{O}(1)$

↓
declarac

de var,

sumas

def foo () :

int a = 10;) $\mathcal{O}(1)$ {
int b = 5; , $\mathcal{O}(1)$ } $3\mathcal{O}(1) = \mathcal{O}(1)$
return a+b;) $\mathcal{O}(1)$

def factorial (int m):

$\mathcal{O}(1)$ result = 1

$n \mathcal{O}(1)$ for (; $m > 0$; $m--$):

$\mathcal{O}(1)$ { result *= m;

$\mathcal{O}(n)$ return result; } $\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(1)$

$\mathcal{O}(n+2) = \mathcal{O}(n)$

$\mathcal{O}(n)$ { for () }

$\mathcal{O}(n)$ { for () }

$2 \mathcal{O}(n) = \mathcal{O}(n) \rightarrow$ Búsqueda lineal

↓
buscan el máximo

$\mathcal{O}(n^2)$ { for (-> i:m)
for (-> j:k)

- Selection sort
- Insertion sort
- Bubble sort.

$\mathcal{O}(n)$ Sumar elem
de un arreglo
buscar máx
de arreglos (desordenados
u ordenados)

$\mathcal{O}(m \log n)$ vs $\mathcal{O}(n^2)$

↑
estáse bien
que acceso
+ lento

$\mathcal{O}(1)$ $\mathcal{O}(\log(n))$ $\mathcal{O}(n^2)$

$\mathcal{O}(n \log(n))$ $\mathcal{O}(n^3)$

$\mathcal{O}(\exp(n))$

Encontrar la parte entera de un n°.

def raiz(n):

i = 1

while (true):

if $i * i > n$:

exit

return i-1

$\mathcal{O}(\sqrt{n})$

Selección $\mathcal{O}(n^2)$ vs MergeSort

$$T(n) = T(m-1) + \mathcal{O}(1)$$

↓ ↓ ↗
tanda + tanda - caso base
equación de recurrencia

Llamo recursivamente y luego comparo.

$$\begin{aligned} &= (T(m-2) + \mathcal{O}(1)) + \mathcal{O}(1) \\ &= T(m-2) + 2\mathcal{O}(1) \\ &= T(1) + (m-1)\mathcal{O}(1) \\ &= \mathcal{O}(1) + (m-1)\mathcal{O}(1) \\ &= m\mathcal{O}(1) = \mathcal{O}(n) \end{aligned}$$

$$\begin{aligned} F(m) &= F(m-1) + F(m-2) + \mathcal{O}(1) \\ &= \mathcal{O}(2^m) \rightarrow \text{tpo de orden exponencial.} \end{aligned}$$

Búsqueda Binaria

~~m log n~~

División y Conquistar

$$T(n) = 1 \cdot T(m/2) + \mathcal{O}(1)$$

Cuando $m = 2^K \Rightarrow K = \log_2(n)$
entre llamas

$\alpha \in [\# \text{ de llamadas recursivas}]$

$$T(n) = T(2^K/2) + \mathcal{O}(1) = T(2^{K-1}) + \mathcal{O}(1)$$

Sabemos que: $T(2^{K-1}) = T(2^{K-2}) + \mathcal{O}(1)$

$$\begin{aligned} &= (T(2^{K-2}) + \mathcal{O}(1)) + \mathcal{O}(1) \\ &= T(2^{K-2}) + 2\mathcal{O}(1) \\ &= T(2^0) + K\mathcal{O}(1) = \\ &= T(1) + \mathcal{O}(K) = \mathcal{O}(K) \end{aligned}$$

$$\mathcal{O}(K) = \mathcal{O}(\log m) = \mathcal{O}(\log m)$$

No importa
el logaritmo

$$T(m) = 2_0 T(m/2) + (\mathcal{O}(m))$$

use $\lambda = \log_2 n$

$\therefore T(n) = 2_0 T(2^K/2) + \mathcal{O}(n)$

$$\Rightarrow T(n) = 2T(2^{K-1}) + \mathcal{O}(2^K)$$

Además

$$T(2^{K-1}) = 2 T(2^{K-2}) + \mathcal{O}(2^{K-1})$$

$$T(m) = 2 \sum_{i=0}^{K-1} 2^i \mathcal{O}(2^{K-(i-1)})$$

generalizar

Merge Sort

$$= 2 \sum_{i=0}^{K-1} \mathcal{O}(2^{K-i}) + \mathcal{O}(2^K)$$

$$= \sum_{i=0}^{K-1} \mathcal{O}(2^K) + \mathcal{O}(2^K) = \sum_{i=0}^K \mathcal{O}(2^K) = K \mathcal{O}(2^K)$$

$$= \mathcal{O}(2^K K) = \mathcal{O}(m \log m)$$

Práctica

Erros

Compilación → Compiladores

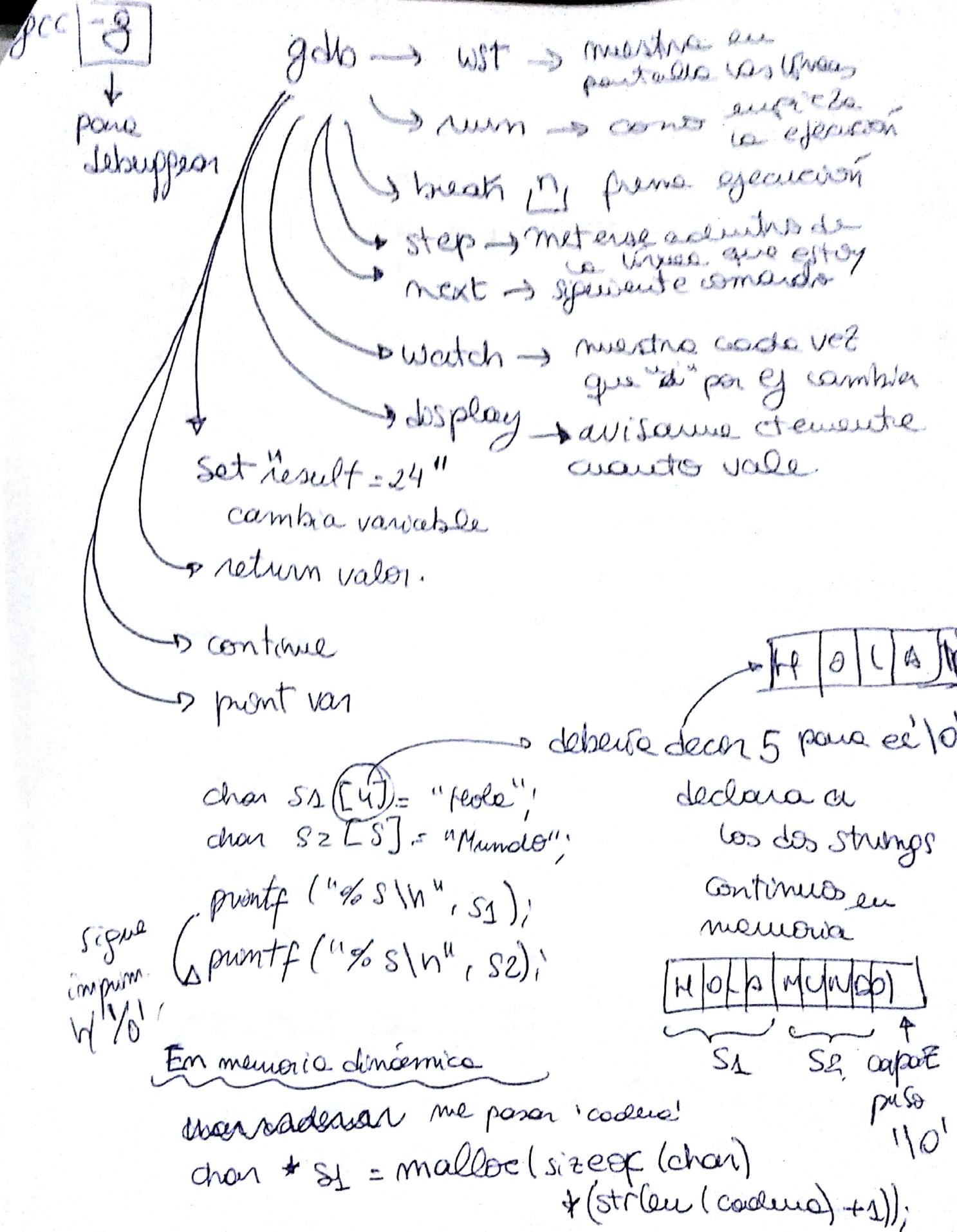
Memoria → Memcheck

Bugs → Debuggers } even no esperado y buscan por testeos,

GDB



El factorial de 4 es 0.



* Se pone el typecast para obviamente escribir

'struct persona' a code nota y simplemente poner
 (persona_t). persona_t p = { . . . }; se puede usar se admis
 del code.

~~Solo~~ Idea

→ perdí → Valgrind

persona_t * crear_persona (int edad, char * nombre,
int dni)

{ persona_t * persona = malloc (sizeof(persona_t));

Naciendo
le mismo
(*) persona).edad = edad;

(*) persona).nombre = nombre;

persona -> dni = dni;

return persona;

ofree

Pueden con
Valgrind

mes 1º parcial
No / porque
Sino no obtengo
un puntero a
la estructura
Personas ☺

Teorema mestizo \Rightarrow es para D&C

$$T(m) = A T(m/B) + \mathcal{O}(m^c)$$

llam
recursivas

proporción
de cada
subproblema

cuanto nos cuesta

"partir y juntar"

(todo lo que nos tiene
que ver con
recursividad)

$$\Rightarrow T(m) = \mathcal{O}(m^c)$$

$$\Rightarrow T(m) = \mathcal{O}(m^{\log_B A})$$

$$\Rightarrow T(m) = \mathcal{O}(m^{\underbrace{\log_B A}_c})$$

dice: $m^c \log m$

$\log_B A / C$	
<	$\mathcal{O}(m^c)$
=	$\mathcal{O}(m \log m)$
>	$\mathcal{O}(n^{\log_B A})$

Mergesort

$$T(m) = 2(T(m/2)) + \mathcal{O}(n)$$

$$A = B = 2$$

$$C = 1$$

$$\log_2 2 = 1 = C$$

$$\Rightarrow T(m) \sim \mathcal{O}(m \log m)$$

Búsqueda binaria

(números ordenados)

$$T(m) = 1 \cdot T(m/2) + \mathcal{O}(1)$$

$$\begin{aligned} A &= 1 \\ B &= 2 \\ C &= 0 \end{aligned}$$

$$\log_2 1 = 0 = C \Rightarrow T(m) \sim \mathcal{O}(\log n)$$

Búsqueda (mejor recursiva) (secuencial)

$$T(m) = 1 \cdot T(m-1) + O(1) \Rightarrow$$

está
en el 1º

sino todo

lo demás

No usar
Tres Maestros

$O(n)$

B debe ser > 1

$A > 0$



Escribirlo como D&C

Obs:

ESc con

D&C a

pesar de
tener SC
iterativa

defk búsqueda (an, inicio, fin, elem):

if inicio > fin:

return false

mitad = (inicio + fin) // 2

if (an[mitad] == elem): return True

return búsqueda (an, inicio, medio - 1, elem)

or búsqueda (an, medio + 1, elem)
if fin

$T(m) = 2 T(m/2) + O(1)$

$A=2=B$, $C=0$

$\log_2 2 = 1 > C \rightarrow O(m^{\log_B A}) = O(m)$

Algunas soluciones: no puede ser - $O(n)$ si
esta desordenado.

• no descarta mitades ($A=2$)

Calcular m^k en $\mathcal{O}(\log k)$

$$m^k = \begin{cases} m^{k/2} \cdot m^{k/2} & \text{si } k \text{ es par} \\ m^{k/2} \cdot m^{k/2} \cdot m & \text{si } k \text{ es impar} \end{cases}$$

def potencia(m, k)

if $k == 0$: return 1

pot_mitad = potencia(m, $k/2$)

if $k \% 2 == 1$:

restante = m

else: restante = 1

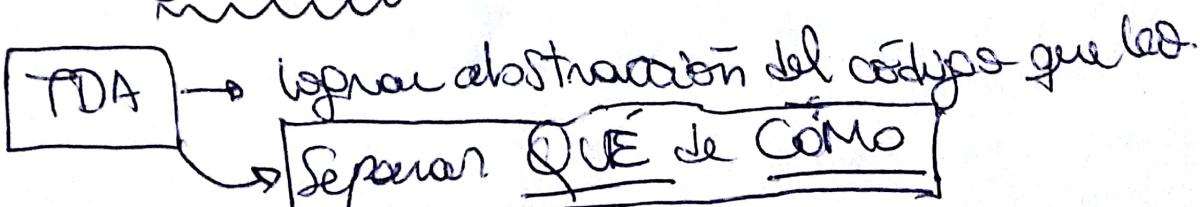
return pot_mitad * pot_mitad * restante

$$T(k) = T(k/2) + \mathcal{O}(1)$$

$$\Rightarrow \mathcal{O}(\log k)$$

misma ec de
recurrencia que
desq binaria

Fase = orden



* + trabajo con las primitivas que expone

funciones que expone

poseen invariantes

* No debe depender de la implementación

Invariantes: comparten pre y pos cond. Especifican al TDS y son parte de la descripción de sus primitivas.

Caso de prueba → Prueban todos los invariantes de la
primitiva

Tipo de dato abstracto

Clase 5

typedef struct pila pila_t;

pila_t pila;



No compila,

No desreferencia.

pila_t * pila



a qué tipo de dato

referencia.

Básicamente no
puedes tener

una

pila en
el stack

- No tiene sentido:

pila_t clear();

Vive en el stack

pila_t p = &clear();

Pila [Estructura LIFO]



Guardo cosas y
las saco en orden
inverso que las metí.

→ Blockchain

→ Stack

Simulan retroceso.

Balanceo de parentesis

Colas [Estructura FIFo]

→ ej de
uso

→ prop de
estabilidad

de met.s

de ordenamiento.

Pruebas: • Prueban códigos de errores

• Caso de uso R

• Software viejo, actualizaciones.

Consideran casos de volumen

Prácticas

gcc tp0.c -e -o tp0.o
compilar → nombre
↓
para ese
.c a un
.o.

gcc tp0.o testeng.o tp0.c
el que tiene el main

Para que no ande compilando cosas repetidas
dijo los .o.

Makrfile

objetivo: dependencias
dependencias 2.

CC = gcc

CFLAGS = -std=99 -Wconversion -Wtype-limits
-pedantic

Uname \$ (CC)

<Va a subir un makefile al drive>

PILA

struct pila {
 void **datos;
 size_t cantidad;
 size_t capacidad;
};

puntero.

amplio de punteros

La pila solo almacena punteros

invavante

→ condensación que siempre se cumple

Cases de
Prueba

{
función constante
casos borde
Volumen
Mem. dim.

La inserción O tamb es valiosa para el espacio.

Clase 6

La div. y C. ocupa log(n).

Heurística O(1)

Si tiene 1 y dops va llamando a las demás O(n)

CANT = 2

CAP = 6

A	B			
---	---	--	--	--

- Si empiezas a poner elementos al pplo se vuelve O(n) queremos que sea O(1)

- Si agrego de a 1 hayo realloc a cada resto y es una op. constante O(1). Aplicarse hace O(n)

Si apelmisionamos en el caso de duplicar \Rightarrow

log n \rightarrow redimensionar

$n \log_2 n \rightarrow$ ^{n veces} apilar o desapilar

en promedio
va a ser O(1)

\downarrow
usamos que
es O(1)

Faltamos de ver si una adic
puede ser O(1)

A	B	C	D	E	F	G
---	---	---	---	---	---	---

pos-ini = 2
CANT =

C	D	E	F	G	H	I
---	---	---	---	---	---	---

tiempo que hacer malo
que es una op de O(n)
pero como lo hago pocas
veces no afecta.

	B	C
--	---	---

an [(pos-ini + CANT) * CAP]

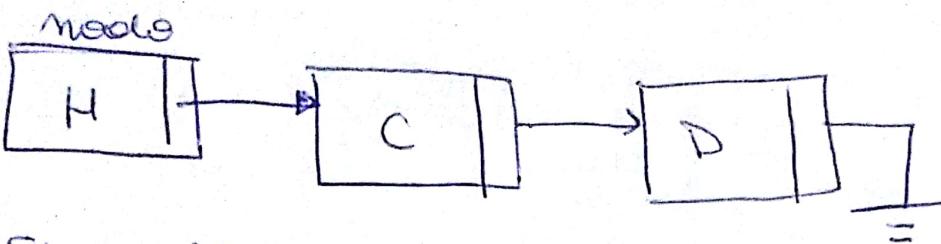
pos-ini = 1

CANT = 2

CAP = 3

Estructuras enlazadas

↳ Conj de nodos

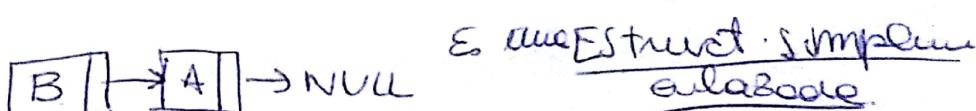


- Si pedimos memoria para un nodo nuevo es $O(1)$
- Meter algo en el nodo es $O(1)$

Definir una colección con una estr. enlazada.

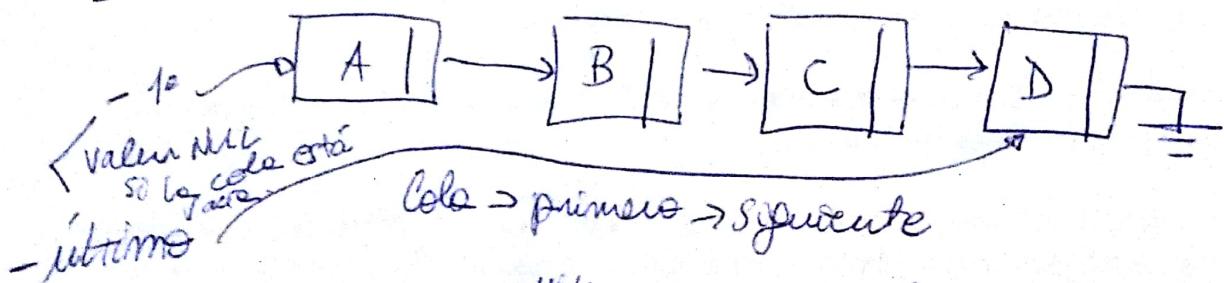


Agrego otro a la izq para que sea $O(n)$



ES + eficiente
trabajan con
arreglos que
con estruct.
enlazadas.
(a bajo nivel
se hacen
optimizaciones)

Implementar una colección con estr. enlazada.



Cola cuando encolo & cuando desencolo
& está vacía. & q posea a estas
mismas.

~~Notar~~ no tienen un sentido

- No importa implementación
- importa el comportamiento y es función de los algoritmos. (2)

Vamos a implementar colección enlazada.

Lista insertamos en cualquier lado.

Estr. deblemente enlazada (conoces el anterior)

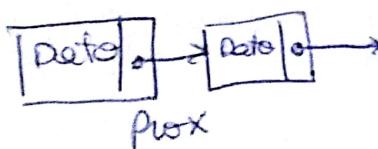
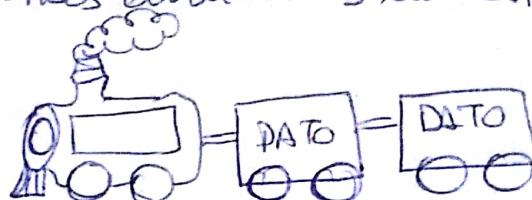
- ↳ Buscar el índice es $O(1)$
- ↳ me muevo adelante o atrás con iterator
- ↳ hay que mantener 2 punteros.

ArrayList → vector dinámico de rededor.

• Práctica •

Estructuras enlazadas

Varios elem repetidos enlazados entre si. (Vayendo de un tren)



```
typedef struct nodo{  
    void * dato  
};
```

```
struct nodo * pnt;  
} nodo_t;
```

```
nodo_t * crear_nodo  
(void * val){
```

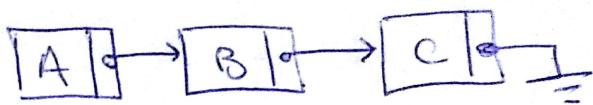
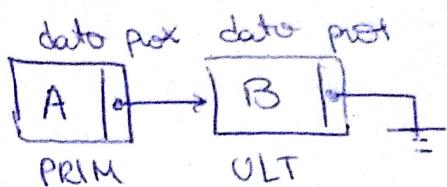
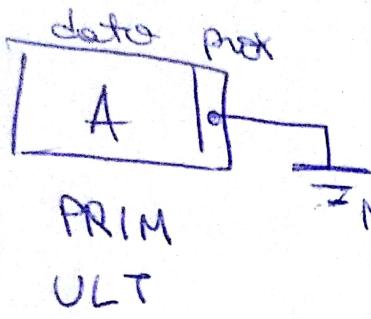
```
nodo_t * nodo  
= malloc();  
if (!nodo) ret NULL  
nodo->dato=val  
nodo->pnt=NULL  
ret nodo
```

Clase Cole

```
typedef struct cole{  
    nodo_t * prim;  
    nodo_t * ult;  
} cole_t;
```

primitivas importantes

- encolar
- desencolar

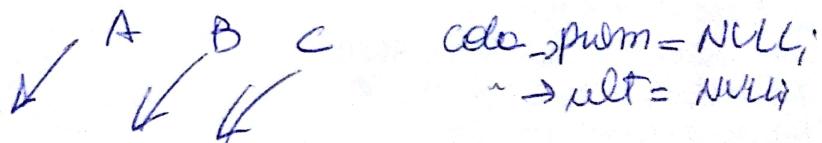


$\text{cole} \rightarrow \text{prim}$
 $= \text{node_a};$

$\text{node_b} \rightarrow \text{prox} = \text{node_c};$

$\text{cole} \rightarrow \text{ult} = \text{node_c};$

- cuando deseas devolver el valor.



$\text{const} \rightarrow$ ese que no hace modif en la cole.

Multiplican x 2 el伦 del vector.

```

size_t tam = vector_obtener_tamano(v);
for (size_t i=0; i < tam; i++) {
    int tmp;
    vector_obtener(v, i, &tmp);
    vector_guardar(v, i, tmp * 2);
}

```

Alan lo implementa.

```

void vector_mul_2(vector_t *v) {
    for (size_t i=0; i < v->tam; i++) {
        v->datos[i] *= 2;
    }
}

```

Incluimos la operación en vector_dinamico.h

```

void vector_mul_2(vector_t *v); // MUY ESPECIFICA

```

Alon decide generalizar. Multiplicar por n.

```

void vector_mul_n(vector_t *v, int n) {
    for (size_t i=0; i < v->tam; i++) {
        v->datos[i] *= n;
    }
}

```

Operación en vector_dinamico.h

```

void vector_mul_n(vector_t *v, int n); // OTRAS OPERACIONES

```

Alan decide generalizar: cualquier operación.

```

enum operacion { ADD, SUB, MUL, DIV };
void vector_arit(vector_t *v, enum operacion op, int n)
{
    for (size_t i=0; i < v->tam; i++) {
        if (op == ADD) v->datos[i] += n;
        else if (op == SUB) v->datos[i] -= n;
        else if (op == MUL) v->datos[i] *= n;
        else if (op == DIV) v->datos[i] /= n;
    }
}

```

Resumen de la situación

- Barbara tiene necesidades específicas aplicar una operación en cada elemento.
- Barbara ya lo puede hacer usando las primitivas públicas → pero quiere escribir menos código.
- Alan sabe cómo aplicar una operación op con código muy conciso.
- Alan no sabe qué operaciones quieren aplicar sus usuarios en su código.
- Alan intenta ayudar ofreciéndole un "conjunto genérico" de operaciones → pero no puede generalizar lo suficiente.

Solución: encapsular la operación
// Alan define un tipo encapsulado "vector_op":

```

// vector_dinamico.c
void vector_aplicar(vector_t *v, vector_op op) {
    for (size_t i=0; i < v->tam; i++) {
        v->datos[i] = op(v->datos[i]); // "op" actúa como
                                         // una función.
    }
}

```

Ejemplo del lado Bárbara

```

int por17(int elem) {
    return elem * 17;
}

Vector_aplicar(v1, por17);

```

Aplicar es una función de orden superior, porque recibe por parámetro una función.

con & no pasa
modifica no se rompe
entiendo que
es la dirección
de memoria.

Definición de vector_aplicar en C

// Como es un vector de enteros, el tipo "vector_op" tiene la forma:

```

int operacion(int elem);

Por tanto:
void vector_aplicar(vector_t *v,
                     (int)(*operacion)(int elem)) {
    for (size_t i=0; i < v->tam; i++) {
        v->datos[i] = operacion(v->datos[i]);
    }
}

```

- Alan: hace TDA
Bárbara: usa TDA
- Versión A**
void Vector_aplicar
(vector_t *v;
(int)(*operacion)
(int elem);
 - Versión B**
typedef (int)(*vector_op_t)
(int elem);
void vector_aplicar (vector_t *v;
vector_op_t op)

Tipo de ordenamiento
no comp

Insertion con un arreglo "ordenado" es $\mathcal{O}(n)$
si no es $\mathcal{O}(n^2)$

$$\mathcal{O}(n) \rightarrow \mathcal{O}(n \cdot k) \quad k \approx \log n \\ \rightarrow \mathcal{O}(n \log n) \\ k \ll \log n$$

Si tengo un alg. de $\mathcal{O}(1)$
me conviene usar insertion

Ordenar nros del 0 al 9

[1, 5, 7, 3, 4, 1, 1, 3, 9, 0, 0, 5, 7]

Pedir ordenando con merge sort si: grande

Counting sort

[0 0 0 0 0 0 0 0 0] → d: long del arreglo
 ✓ ✓ ✓ 1 ✓ ✓ 1 ↗ inicio en 0 es
 2 3 2 2 2 2 2 $\mathcal{O}(d)$

[0, 0, 1, 1, 1, 3, 3, 4, 5, 5, 7, 7, 9] $+ \mathcal{O}(n)$

recorrer el arreglo $\mathcal{O}(d + n)$

pasar $\mathcal{O}(d + n)$

Si $d \ll n \Rightarrow \mathcal{O}(n)$

este es (solo applicable para mas)

Counting Sort generalizado

[8E, 7B, 3C, 8B, 7E, 1C, 7B, 30, 20, 10, 2C, 8C, 12E]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

1	2	3	4
X X	X X	X X	X X
2 2	2 2	3	

1º [0, 2, 4, 6, 6, 6, 6, 9, 12, 12, 12, 12]

1	2	3	4
X	X	X	X
2	3	3	4

2º [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

1	2	3	4
X	X	X	X
2	3	4	6

2º [10, 20, 2C, 30, 7B, 7E, 7B, 8E, 8B, 8C, 12E]

$$\mathcal{O}(d) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n+d) = \mathcal{O}(n)$$

crean 1º crean 2º contar en el 2º
 + $\mathcal{O}(n)$
 ↗ ↗ ↗
 leer n.

Alg Estable: el orden rel de elem = pse orden
 en si se mantiene en la salida resp. a la
 entrada.

Selección es estable si ponemos $>=$

Inserción es estable (nunca superpuso elem \Rightarrow)

Mergesort y QSort no son estables

In place

→ Usan otras estructuras (auxiliar)

No \rightarrow Insert, Sele.

Quick Sort \rightarrow Q sort

Merge Sort

Counting Sort

En cada lugar una colección

Versión simplificada
de counting sort

1	2	3	4	5	6	7	8	9	10	11	12	12E
1C	20	3C			7B	8E						
10	2C	30			7E	8B						

[1C 10 20 2C 3C 30 7B 7E 7B 8E 8B 8C 12E]

$$\mathcal{O}(n) + \mathcal{O}(m+d) = \mathcal{O}(n+d)$$

$$+ 000 = \mathcal{O}(n)$$

$d \ll n$

Radix sort

(es estable)

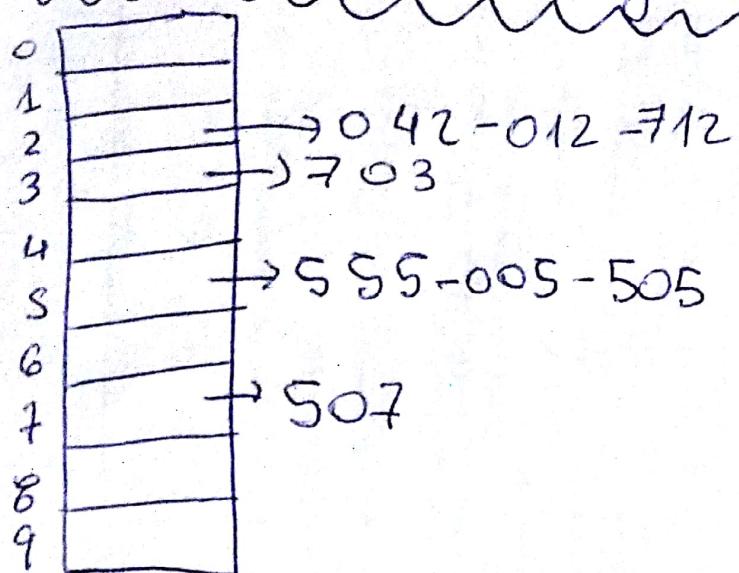
[703 042 012 712 507 555
 005 505]

Cada lugar tiene 10 valores posibles. $\square \square \square$

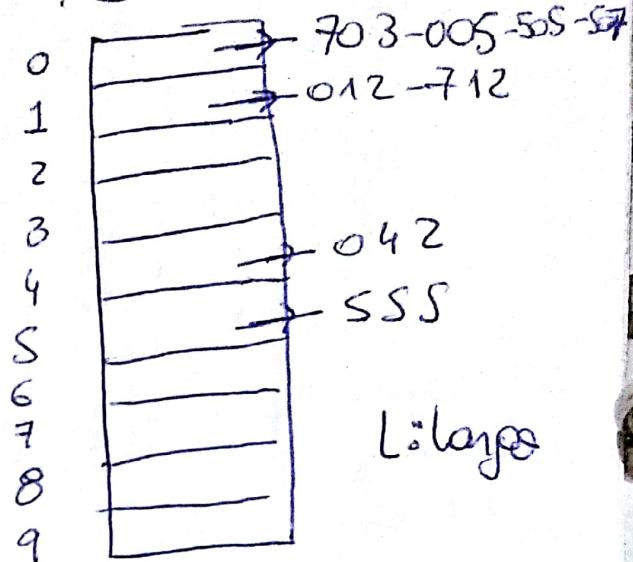
Se necesita ordenamiento estable para poder ordenar.

Empezamos ordenando con Counting Sort.

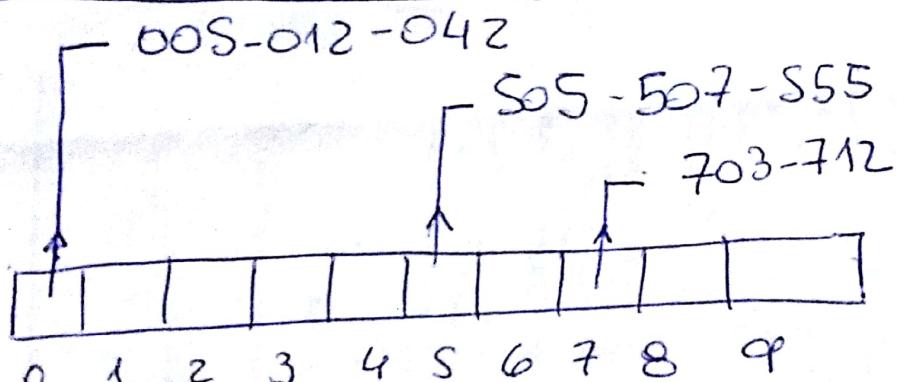
Vemos el último dígito



Volvemos a hacerlo con el 2º dígito



Vamos con el 1º dígito



Práctica

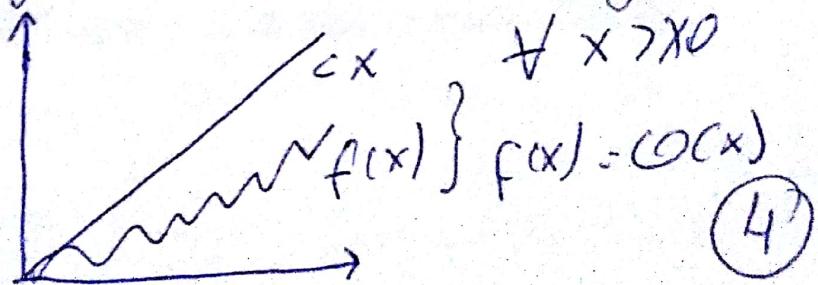
[005, 012, 042,
505, 507, 555,
703, 712]

$\mathcal{O}(L \cdot (m + d))$
aplicando
Counting
Sort

Temas

- » GDB Challenge
- » Demostr. Gráf
- » Teo del Maestro
- » BL vs (Ord + BB)

• Recordan: $f(x) = \mathcal{O}(g(x))$ si $f(x) \leq c g(x)$



(4)

int Sumatoria (int n) {

 int suma=0; → 1 op

 for (int i=0; i≤n; i++) { → 3n op}

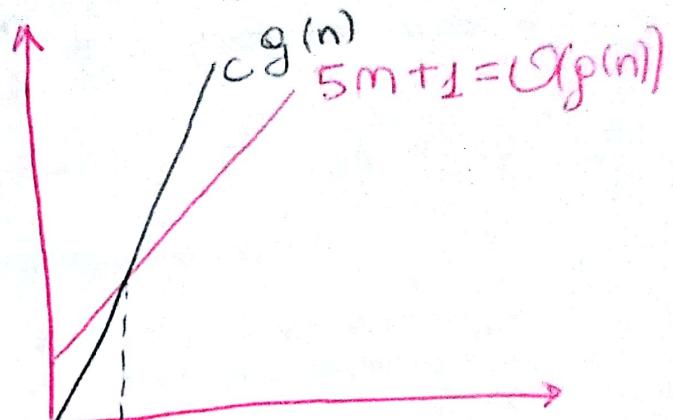
 suma+=i; → n·2 (suma y asign)

}

 return suma;

}

$$5m+1 = \mathcal{O}(n)$$



Otro ejemplo pero recursivo:

int sumatoria (int n) {

 if (m==1) return 1;

1

 int suma = n + sumatoria(m-1); → 2 + T(m-1)

 ↑
 asign
 y suma.

 return suma;

 → Ecuación de recurrencia

$$T(n) = 2 + T(n-1) + 1 = 3 + T(n-1) = T(1) + 3n$$

→ es $\mathcal{O}(n)$

Problema de búsqueda binaria

Recordar n^c si $\log_B a < c$

$n^c \log n$ si $\log_B a = c$

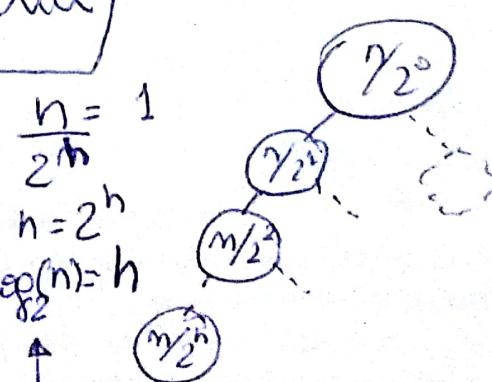
$n^{\log_B a}$ si $\log_B a > c$

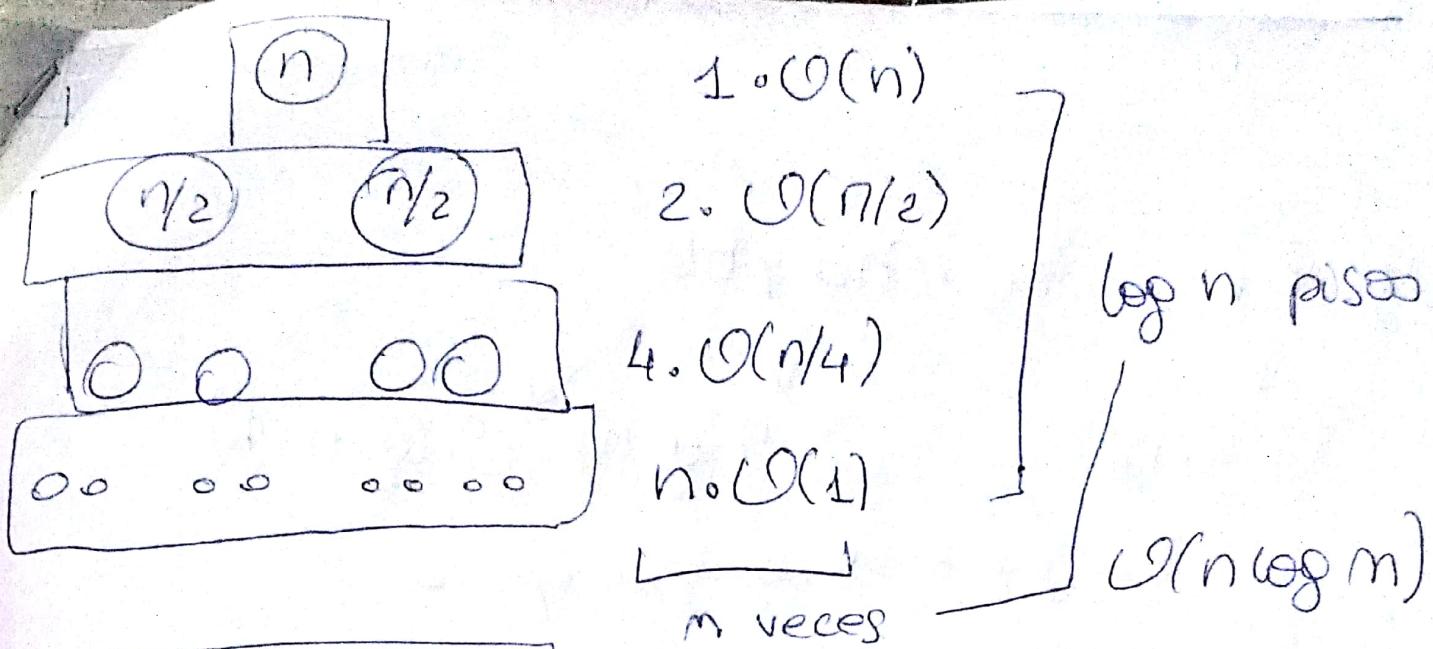
$$\begin{aligned} n &= 1 \\ 2^n & \\ n &= 2^h \\ \log_2(n) &= h \end{aligned}$$

↑
No a terminan siendo

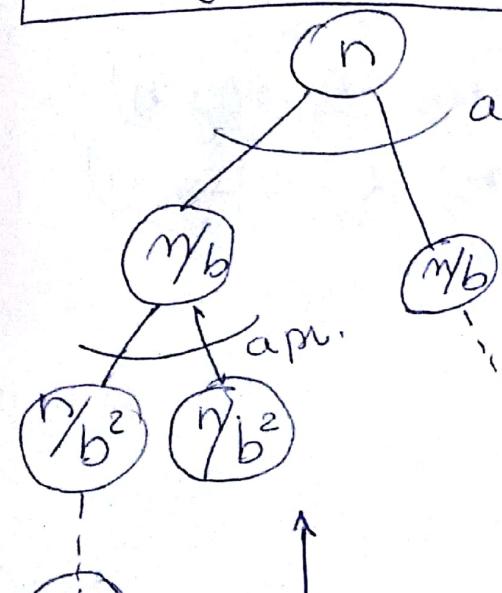
$\log(n)$

Optimización (S)

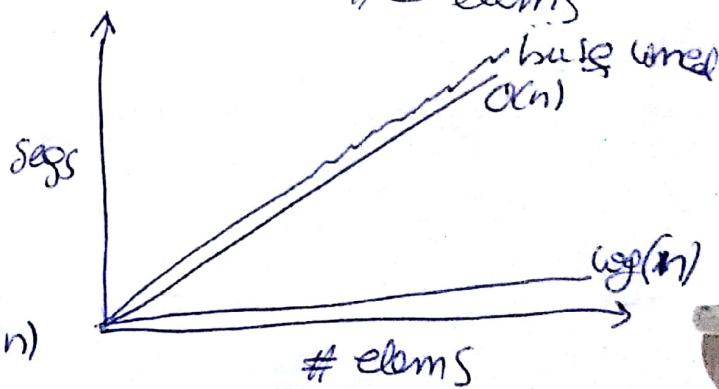
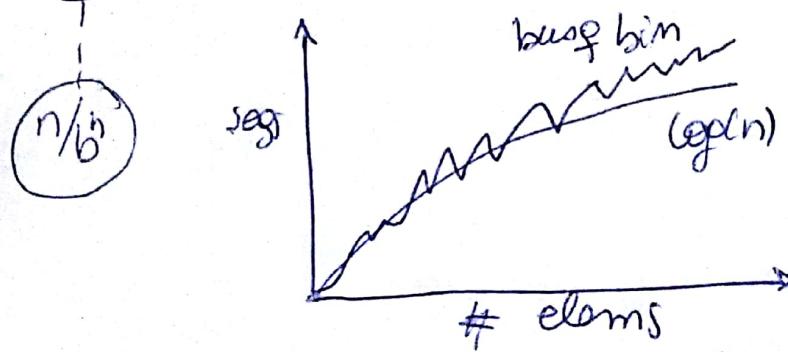
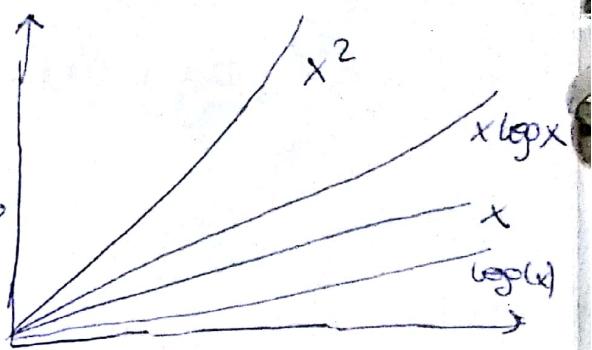




En general



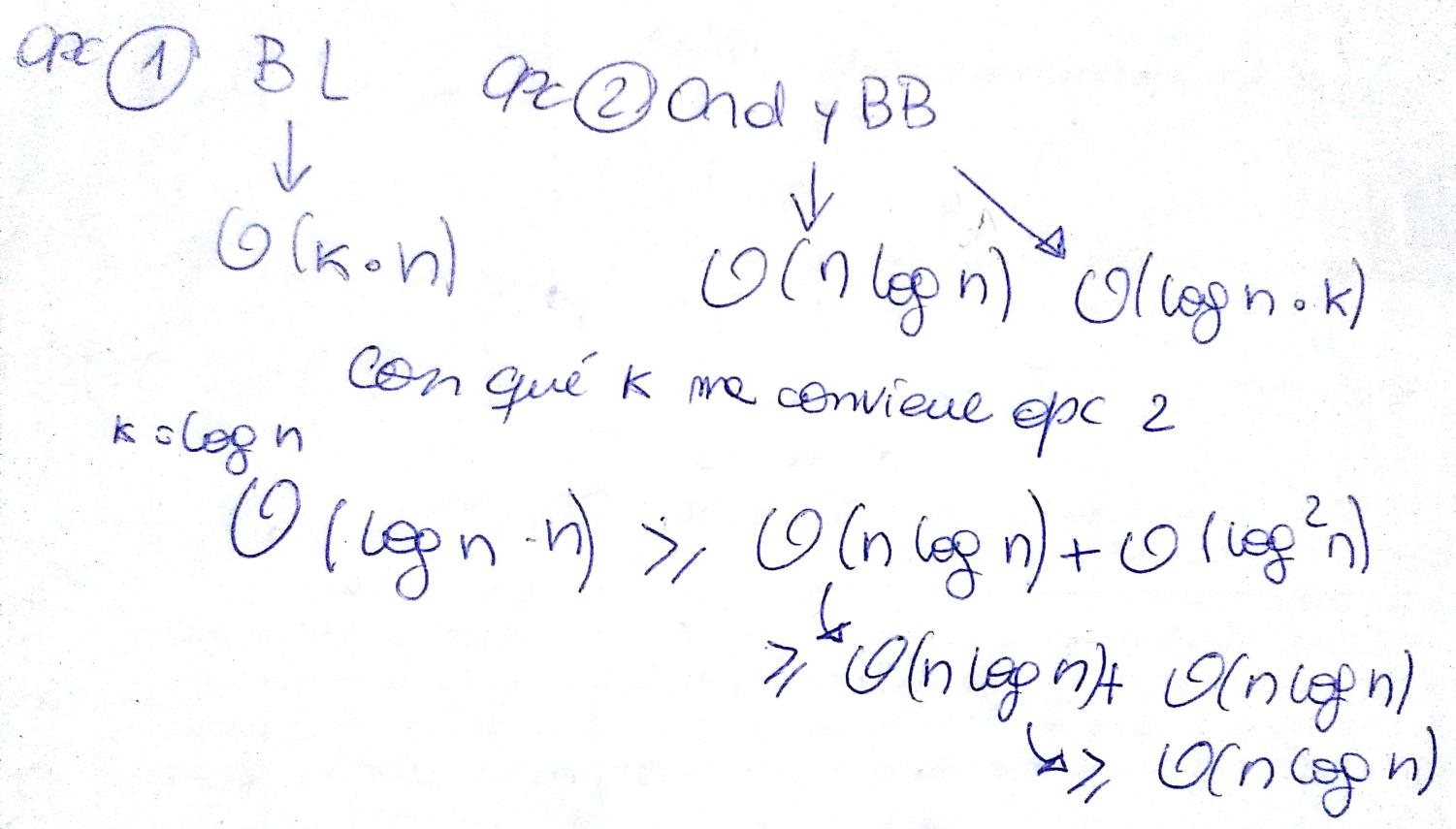
a problemas teórico



An desord
Bisq lineal

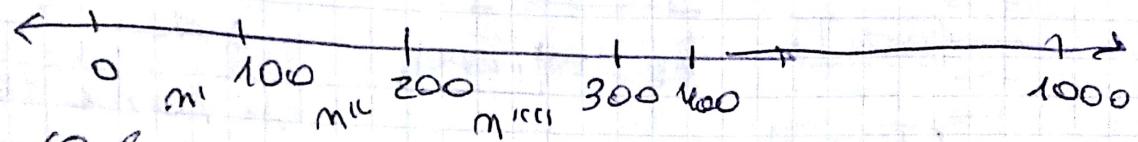


- An des
- Ordena en $n \log n$
- bucle binaria



Clase 8

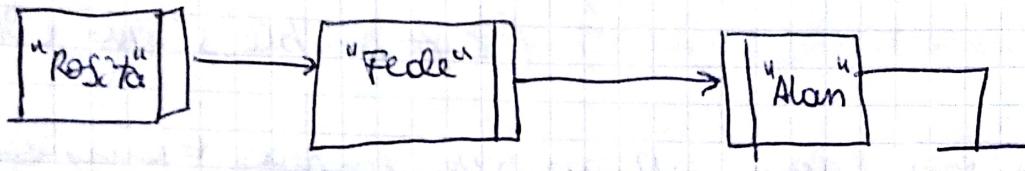
Bucket Sort



$O(n)$ $\rightarrow O(n+b)$ los intervalos no tienen que ser iguales.

TDA Conj y Diccionario

<clave, valor>



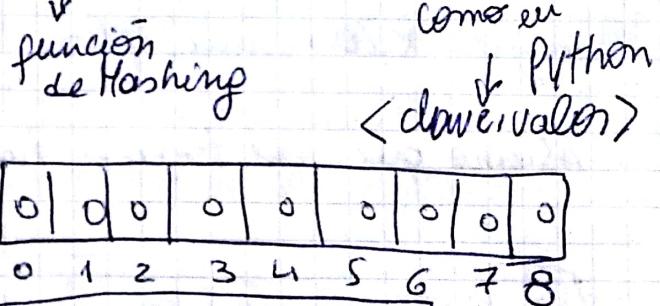
$O(n)$

Tablas de Hashing

→ Cadena

$$f: \text{Cadena} \rightarrow [\emptyset, L]$$

te devolverá pos
y vos dispone
fijos si
esta libe



Pertenece: $O(1)$

Guardado: $O(1)$

Borrar: $O(1)$

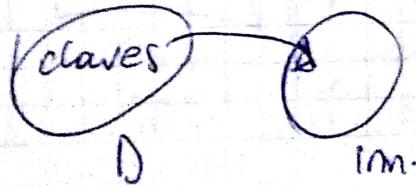
Importante

Si está bonado voy
al sgte.

Autor colisiones: Ample
grande.

La f te tiene que devolver siempre el mismo valor,
la misma pos. lo tengo que poder encontrar.

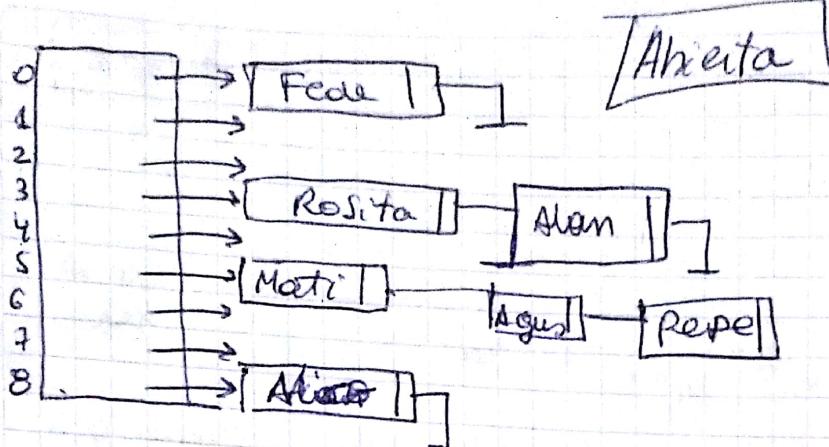
función determinística.



1 solo Im.

- Reutiliz le bonados depende
de implementación. ①

Es circular, si
encuentre ce
8 ocupado la
se la 1.



Es + molesto
redimensionar esto

$\mathcal{O}(1)$ siempre

y cuando sea bueno.

y bien out de redimensionamiento

($< \#$ posibles si son $\mathcal{O}(n)$)

Listas ord x dave explican como mejor + simple.
Mantenerlo ordenado es peor pero no cambia nada,
las listas. Son muy costos. Es $\mathcal{O}(1)$.

El tam inicial y finales importan. Si son primos
mejor x los múltiplos.

Asumo que en todas hay $m \rightarrow \#$ de elementos de la lista.

$M \rightarrow$ Largo de lista

Se pretende buscar funciones de hashing

Problemas de puebas de volumen -

Siempre vamos a trabajar con cadenas como
claves. Los vals que guardamos son void*.

La cierre se lo puede mejorar.

Taller: Miércoles 17-20 hs

Cole: Lunes 24/9

Si no redimensionas
se vuelve muy
rápido ($\mathcal{O}(n)$)
es un anejo de
costos.

Se borras tienen
+ elem.

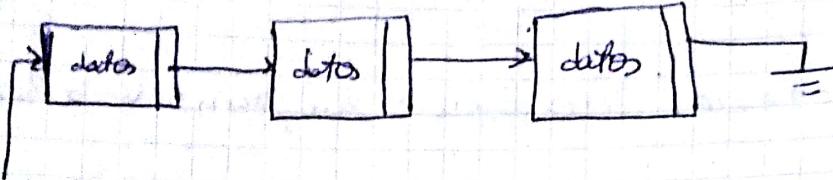
$m \rightarrow \#$ de elementos de la lista.

Avisos.

Práctica

Lista enlazada

Insertar/Borrar: $O(1)$
Perder largo: $O(n)$



Si tengo una ref al út. elem. $\Rightarrow O(1)$ ins y Borrar

Si queres borrarlo es $O(n)$ porque no tenés ref al anterior.

Iteradores

TAD para iterar

Forma genérica de recorrer los elem sin dep de cómo estén guardados

- ↳ forma genérica p/ acceder a los TDA
- Abstr de implement del TDA.
- Si cambia TDA el usuario del iterador no se entera.

Antes:

while p/ iteración xq no sabíamos donde termina

N { implementación primitiva → Alan
I } " función → Bárbara

Tipos de iteradores internos y externos.

↳ los que vemos hoy → adentrarse memoria.

Cómo se usa iterador externo?

list_iter_t *iter = lista_iter_crear(mi_lista);
while (!lista_iter_al_final(iter)) {

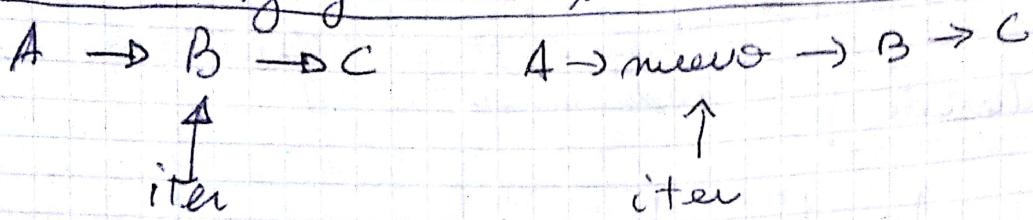
void *dato = lista_iter_ver_actuel(iter);

hacer algo(dato);

lista_iter_avanzar(iter); } lista_iter_destruir(iter);

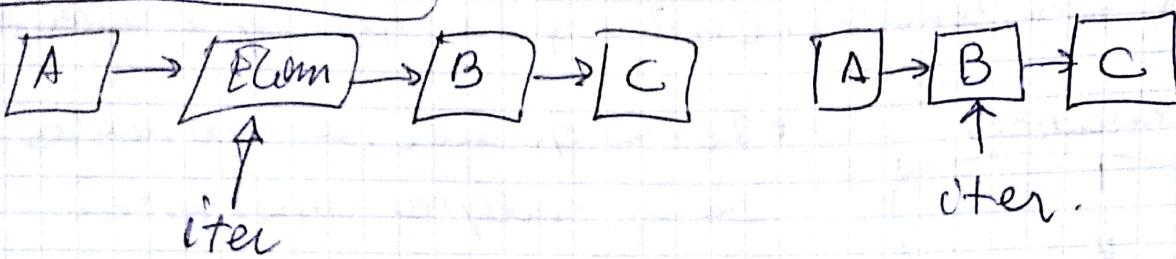
②

Iterador - Agregado de un elemento.



Tendríamos que tener disponible un puntero al anterior y al actual.

Eliminar elem



- si borras el 1º elem pierde la ref en la lista.

No se puede eliminar un elem de la lista x su primitiva mientras estoy recorriendo con un iterador (Debece dejar de recorrer si hace eso).

El iterador lo hace Alan y lo uso Borbore.

Implementar primitiva: \Rightarrow Alan \circ Ej de parabola

```
bool lista_eliminar (lista_t *lista, void *valor);
```

con `typedef struct nodo_lista {`

struct nodo_lista * prox;

void * dato;

} nodo_lista_t;

iterador No

y `typedef struct lista {`

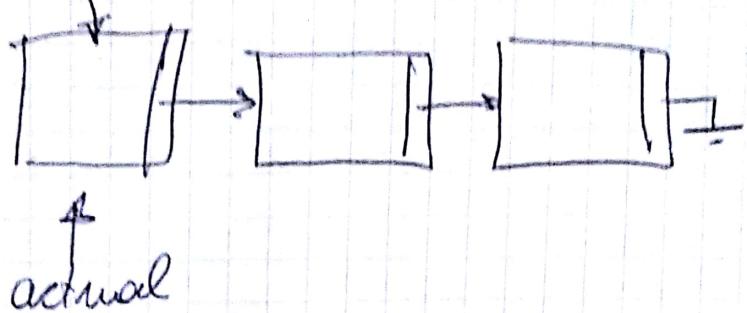
nodo_lista_t * prim } lista_t;

```

bool esta_elemento(Lista* lista, void * valor)
if (lista->encabezado(lista) == null) return false;
bool encontro=false;
nodo_t * anterior=null;
nodo_t * actual=lista->prim;

```

lista.prim



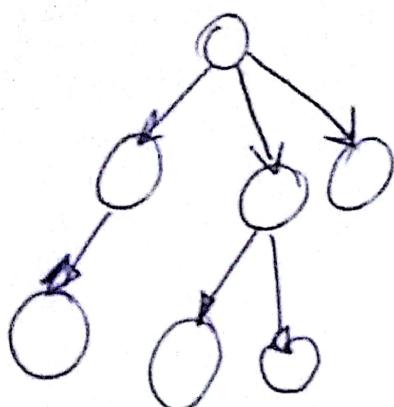
```

while (actual != null) {
    if (actual->dato == valor)
        if (anterior == null) {
            nodo_t * siguiente = actual->prox; prox
            if (anterior == null) {
                lista->prim = siguiente;
            }
        } else { anterior->prox = siguiente; }
        encontro = true;
        free (actual); free que quitar el
        nodo.
        actual = siguiente;
    } else { anterior = actual;
        actual = actual->prox; }
    return encontro;
}

```

Árbol: Estructura en la que cada nodo tiene uno o varios hijos.

Clase 9

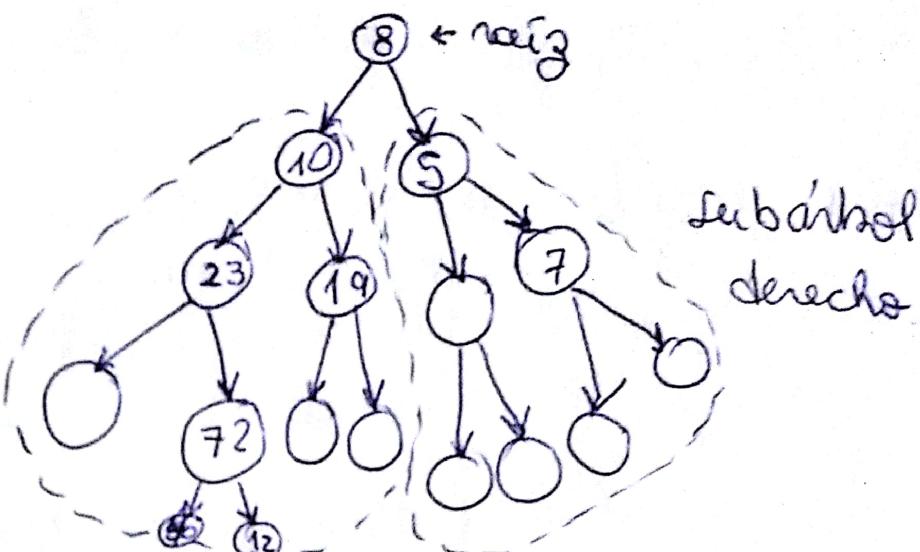


Es recursiva

sub
árbol
izq

Árbol binario:

Cada nodo tiene 2 hijos hasta



Ejemplos de árboles binarios



pre-orden,
pos si el
2 esto
señal.

struct ab{

 struct ab* izq;

 struct ab* der;

} i

```
size_t ab_altura (const ab* ab){
    if (ab == NULL) return 0;
    return 1 + max(ab_altura(ab->izq),
                    ab_altura(ab->der));
}
```

Orden?: Pasa x todos los nodos
solo 1 vez y en c/u hace cosas
 $O(1)$.

$$T(m) = 2 T(m/2) + O(1)$$

$$\rightarrow O(n \log_2^2) = O(n)$$

①

```
def preorden(ab, visitar):  
    if ab == NULL: return  
    visitar(ab.datos)  
    preorden(ab.hijoizq, visitar)  
    preorden(ab.hijoder, visitar)
```

```
def inorder(ab, visitar):  
    if ab == NULL: return.  
    inorder(ab.hijoizq, visitar)  
    visitar(ab.datos)  
    inorder(ab.hijoder, visitar)
```

```
def postorden(ab, visitar):  
    if ab == NULL: return  
    postorden(ab.hijoizq, visitar)  
    postorden(ab.hijoder, visitar)  
    visitar(ab.datos)
```

» Pre:
8 - 10 - 23 - 72 - 40 - 12 - 19 - 5 - 7 → 1º nñz
uso: reconstruir
el árbol
con el in-order.

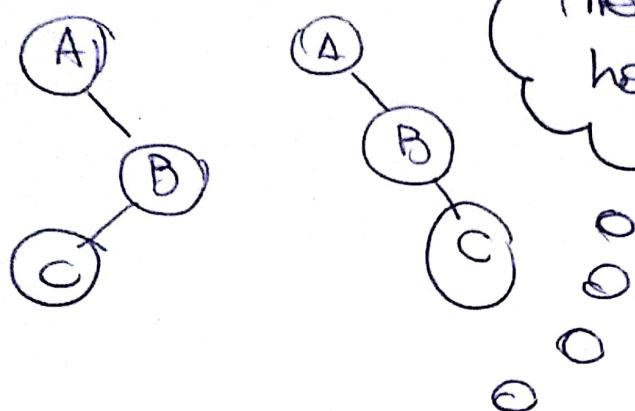
» In: elem que fa va izq está

{ 3, - 40 - 72 - 12 - 10 - 19 - 8 - 5 - 7 }

» Post: 40 - 12 - 72 - 23 - 19 - 10 - 7 - 5 - 8 → última raíz.
uso
destruir el árbol

» Niveles:
8 - 10 - 5 - 23 - 19 - 7 - 72 - 40 - 12

Pre: A-B-C



¿Cómo obtenemos la ~~ft~~ de nodos internos que no son hojas?

La raíz es int
si no es hoja

Size_t ab_internos(const abt * ab){
if (ab == NULL) return 0;
Size_t summa = 0;
if (ab->izq != NULL || ab->der != NULL){
summa = 1; }
return summa + ab_internos(ab->izq)
+ ab_internos(ab->der);
}

	Merge	Quick	Insertion	Counting	Radix Sort	Bucket
Worst Case	$n \log n$	n^2	n^2	$n+k$	$(n+k)d$	$n+k$
Average Case	$n \log n$	$n \log n$	n			
Stable	✓	✗	✓	✓	✓	✓
Counting Sort						

I gotta feeling Get Lucky Taboo Crimen Te bote Upbeat Funk
 2009 2013 2011 2006 2017 2014

La Argentinita Fuegs Yeah! Teams Gangnam Bad
 al Palo 2005 2004 2008 Style Romana
 2004 2005 2006 2007 2008 2012 2008

Contador [0 1 2 3 4 5 6 7 8 9 10 11 12 13]
 [2 1 1 0 2 1 0 1 1 1 1 0 0 1]
 [04 05 06 07 08 09 10 11 12 13 14 15 16 17]

Semana [2 3 4 5 6 7 8 9 10 11 12]
 acumulado [0 → 2 3 4 5 6 7 8 9 10 11 12]
 [04 05 06 07 08 09 10 11 12 13 14 15 16 17]

Filme [LA MIA | Te (Bad) | IGP | Gang (GL) | UPR | TE
 AL PALO | Yeah! Fuego | CRIMEN amor Rom | TABOO | Style | PUNK BOTE
 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11]

Anf al
 Palo está
 antes de Yeah,
 es estable

Radix Sort con Counting Sort

Violeta (128, 0, 128) A Zul (0, 0, 255)

Lima (0, 255, 0) Beige (145, 245, 220)

Turquesa (64, 224, 208) Chocolate (220, 105, 30)

Amarillo (255, 255, 0) Agua (0, 255, 255)

Rojo (255, 0, 0) Cyan (0, 255, 255)

1º B

Lima

Amarillo

Rojo

Chocolate

Violeta

Turquesa

Beige

Azul

Aqua

Cyan

2º G

Rojo

Violeta

Azul

Chocolate

Turquesa

Beige

Lima

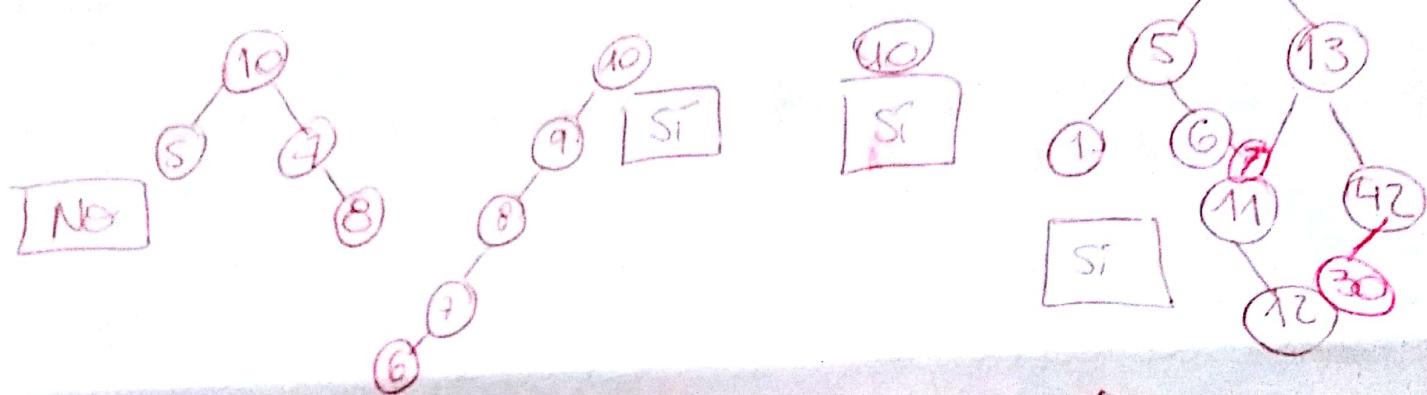
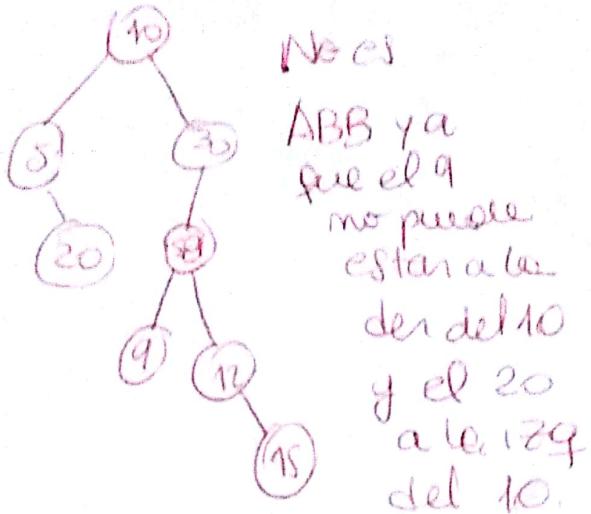
Amarillo

Aqua

Cyan

AVL Árbol Binario de Búsqueda

Clase 11



Es básicamente un TDS que se comporta como bien e inserta como quicksort

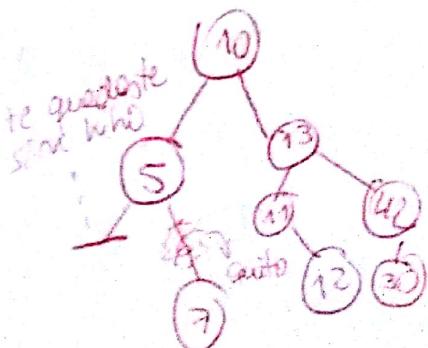
Pre 10 - 5 - 1 - 6 - 7 - 13 - 11 - 12 - 42 - 30

Post 1 - 7 - 6 - 5 - 12 - 11 - 30 - 42 - 13 - 10

In 1 - 5 - 6 - 7 - 10 - 11 - 12 - 13 - 30 - 42
+ grande a la izq + chico a la der.

Acá con el Preorden se puede reconstruir

Bonito → 0 hijos
→ 1 hijo
→ 2 hijos

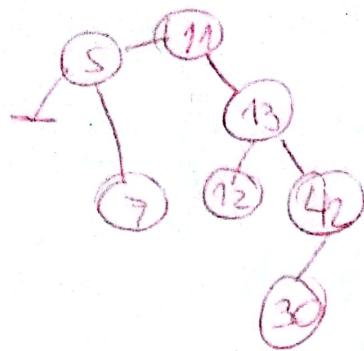


dejalo mantener
el In orden
o lo reempl.
por el 7 o
por el 11.

①

Dame el 11 que es

borrar el 10



Dame el 11
en el plazante y
le cambie el
valor al
nodo

Otra implem.

$11 \leftrightarrow 12$ swaps } definir
datos. } reutilizar
 } codigos

Luego Swaps. B con 11 y bors el modo que
tenia el 11.

Del TPS)

Todo como estos? $\rightarrow n \{ O(n*m)$
"como" $\rightarrow m$

10 5 8 + *

10 13 *

130

2 8 /

4

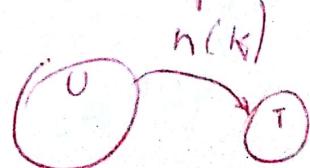
3 8 /

5 30 ?

2

Flash

- Indexan por valores y clave.
- Para una misma clave siempre tienes que devolver el mismo valor.
- Distr. Uniforme.

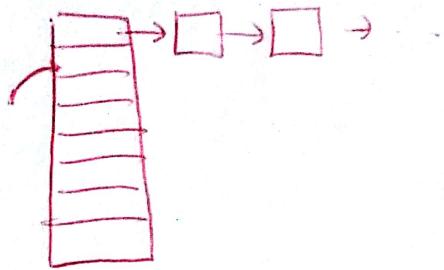


1ul >> (T)

Colisiones → Hash Abierto
 ↳ Hash Cerrado

Abierto

colección,
clave y valor



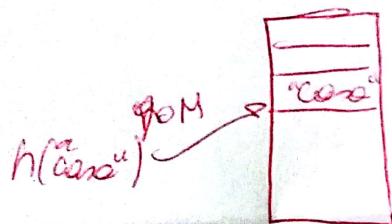
$$\text{"caso"} \rightarrow h(\text{"caso"}) = n+1$$

$$n+1 \% n$$

$$\text{factor de escala } \alpha = \frac{n}{m} \rightarrow \begin{array}{l} \# \text{ de vals que tiene} \\ \text{tam de mi tabla} \end{array}$$

Realloc es $O(n)$ porque tiempo que reubican los m elem de cada lista.

Cerrado



ket (Probing)

estados vacíos
y estados llenos.

estados bonos

clave, valor,
es fijo

guardar copias de claves (no deben ser mutables)

Árbol + ancho \rightarrow por un tema de espacio de memoria:

Estos árboles se usan para BDD

Árbol B

Características

- K : # MAX de claves x modo
- M : # MAX de hijos x modo ($K+1$)
- # MIN de claves

x modo: $\lfloor K/2 \rfloor$ (salvo raíz)

• # MIN de hijos: $\lfloor M/2 \rfloor$ (salvo raíz y hojas)
x modo

Ejemplo: $K = 4 \rightarrow M = 5$

MIN de claves x modo: 2

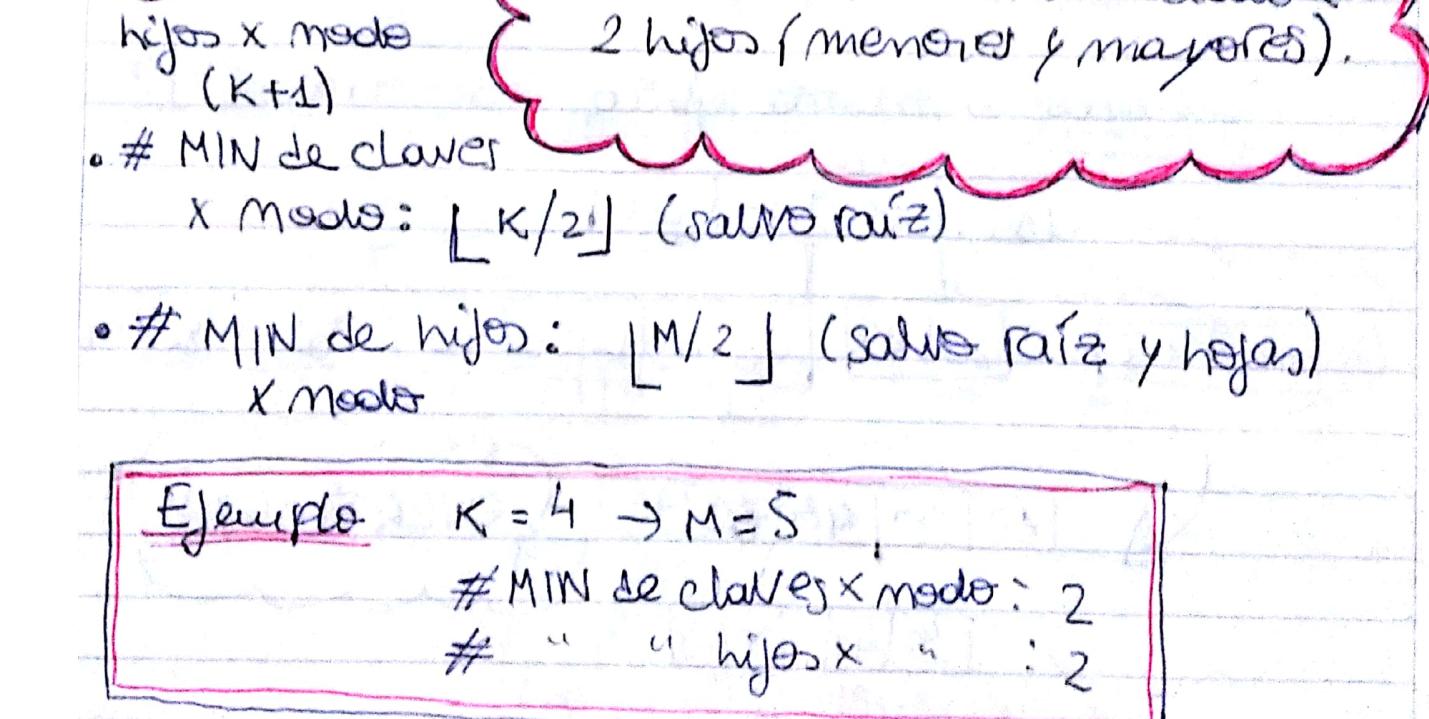
" " " hijos x " : 2

Axiomas del árbol B

① Todas las ramas tienen la misma profundidad

② Si un modo tiene K claves, debe tener $K+1$ hijos (salvo raíz y hojas)

③ Cada clave está asociada a 2 hijos (menores y mayores).

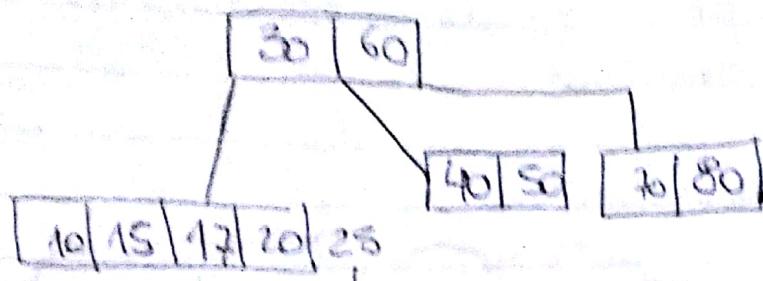


En el peor caso
hago $O(K \log m)$
comparaciones

este lo inserté

(1)

Insertemos el 17 y el 25

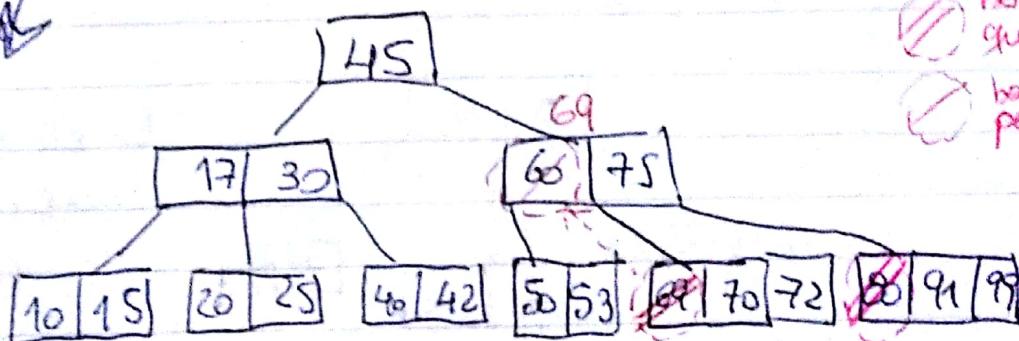
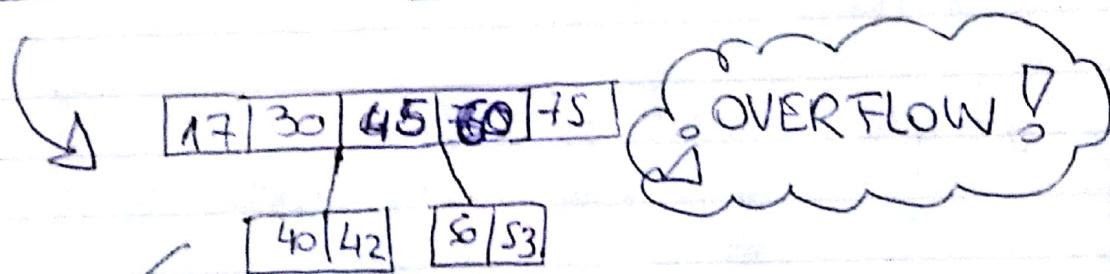
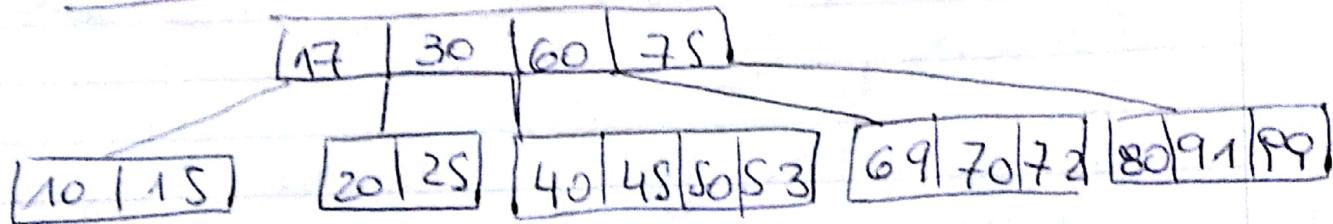


¿Cómo se inserta? → Produce overflow



comprueba el del medio en proche
y divide los "hijos"

Siguientes la misma lógica, insertaremos el 42



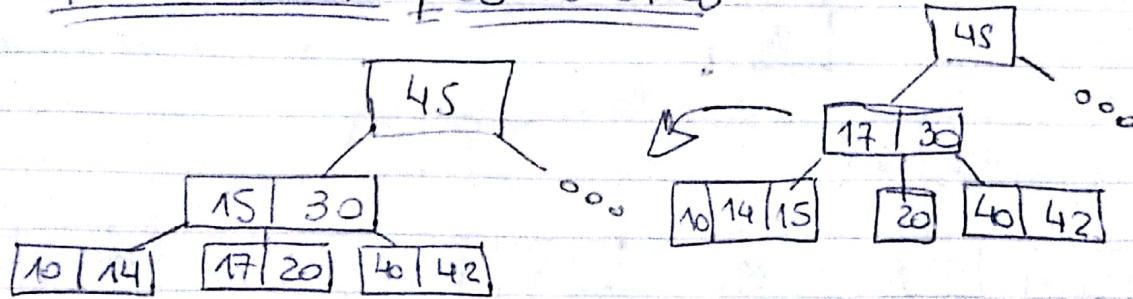
no hay
guito

no
pue cambar

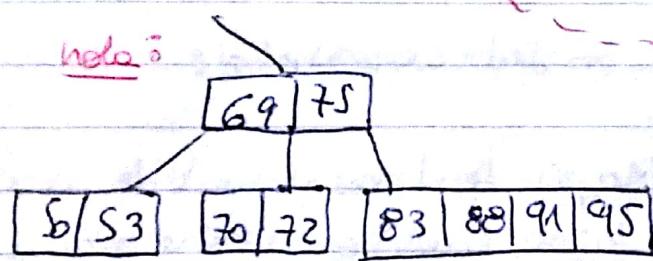
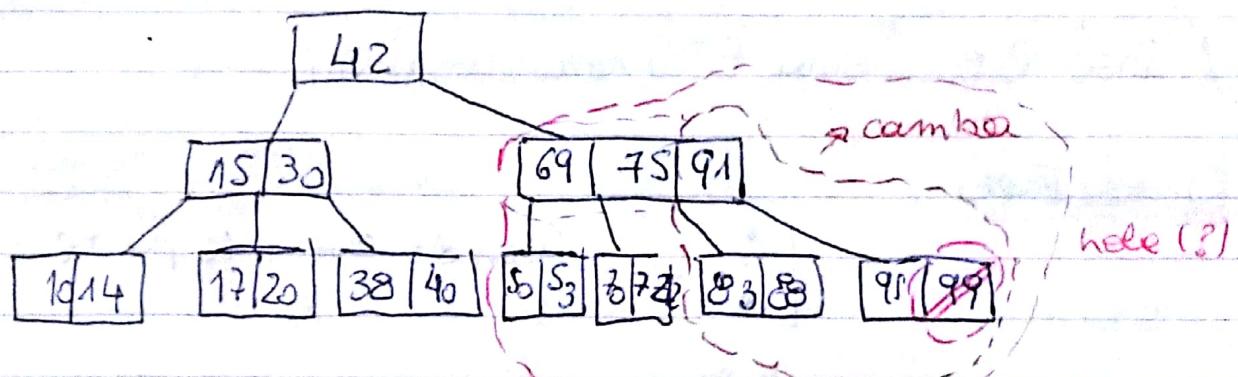
Si queremos borrar el 60

bien, no hay
underflow,
sigue :)

Inserto el 14 y boro el 25

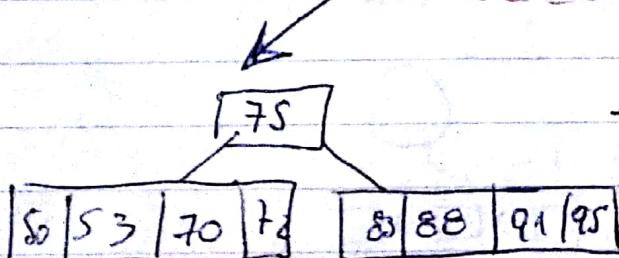
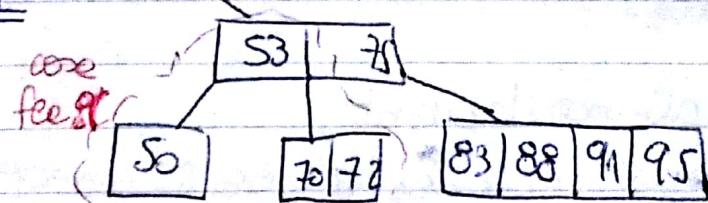


Insertan 95, 83, 88 (pero en fin es un árbol nuevo)

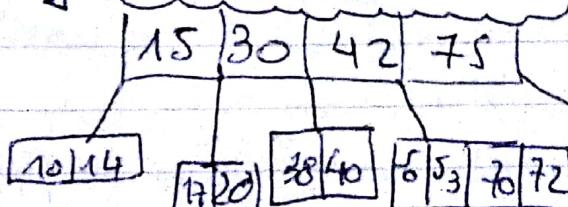


ideal sería
redibujar el árbol

Boro el 69



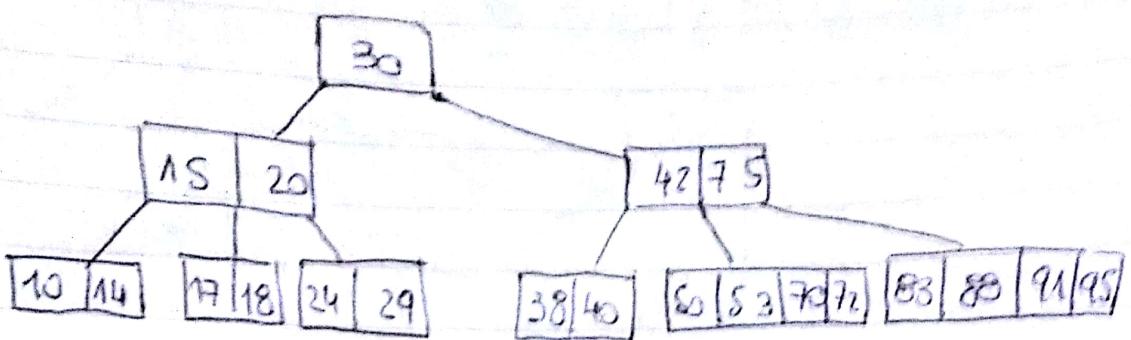
Con esto tenemos lo



83 88 91 95

(2)

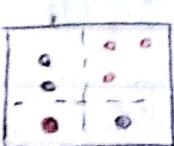
Volvemos al árbol htree y vemos que comienza



Hash Cerrado \rightarrow optimiz. de memoria, está todo en una tabla.

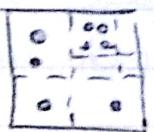
Árbol B+ (Bne BDD relacionales)

Quad Tree



Cuando son 4 se parte

Ahora si hay 4 subcuadrantes:

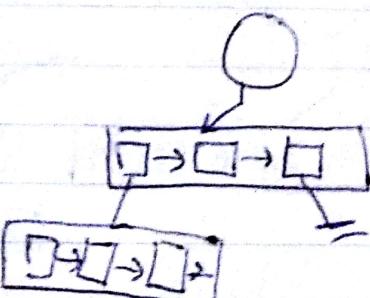


Maplo bidimensional de cosas,
juegos (colisiones entre elementos)

→ Práctico →

- Hay 2 implementaciones
- Con la implem de raíz si o sí necesito una func wrapper p/ la # de hojas, simb mo.

Árbol m-anis



```
typedef struct arbol arbol_t;  
struct arbol {
```

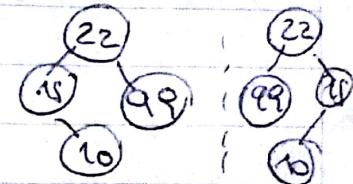
```
    arbol_t *izq;
```

```
    arbol_t *der;
```

```
    void *dato;
```

```
}
```

Enunciado: Espejar el sgte árbol



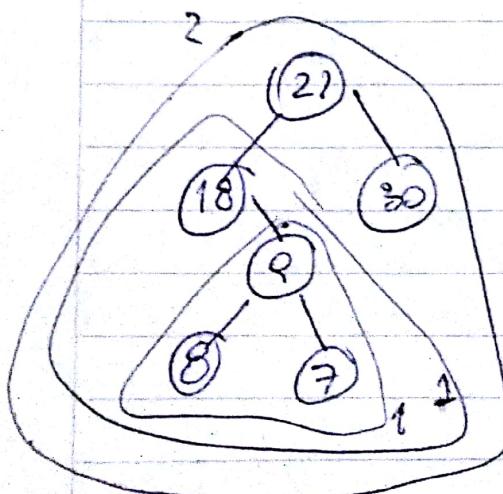
```
void espejar_arbol(arbol_t *ab) {  
    if (!ab) return;
```

~~if (!ab \Rightarrow izq && ab \Rightarrow der) return;~~

```
swap { arbol_t *aux = ab  $\rightarrow$  izq;  
       (ab  $\rightarrow$  izq, { ab  $\rightarrow$  izq = ab  $\rightarrow$  der  
                     ab  $\rightarrow$  der); } ab + der = aux;  
espejar_arbol(ab  $\rightarrow$  izq);  
espejar_arbol(ab  $\rightarrow$  der);
```

```
}
```

Enunciado: Devolver la # de nodos que tienen 2 hijos.



```
int contar_nodos_2_hijos(ab_t *ab) {
```

```
    if (!ab) return 0;
```

```
    int sumo_ac = Cn2h(ab  $\rightarrow$  izq)  
                + Cn2h(ab  $\rightarrow$  der);
```

```
    if (ab  $\rightarrow$  izq && ab  $\rightarrow$  der) sumo_ac++;  
    return sumo_ac;
```

```
}
```

(3)

```
int cn2h (abt* ab) {  
    int sum=0;  
    - cn2h (ab->left, &sum);  
    then sum;  
}  
void -cn2h (abt* ab, int *sum) {  
    if (ab->izq && ab->dch) *sum++;
```

Chequear que 2 palabras son anagramas

argentino

a,1
f,1

chequear que todos los
val en el hash sean 0.



Heap sort \rightarrow esto no es.

Clase 15

7

[7 8 2 3 5 7 4 6]

Ordenar un heap es $O(n \log n)$.

Es in place.

up-heap

Si le hacemos heap a c/u [8 7 7 6 5 2 4 3]

desde el 1º al último $O(n \log n)$ hacerlo p/todos

Si le hago del último al 1º no ande.

(no puedo decir que las cosas se arreglen anden)

No le pido que esté ordenado.

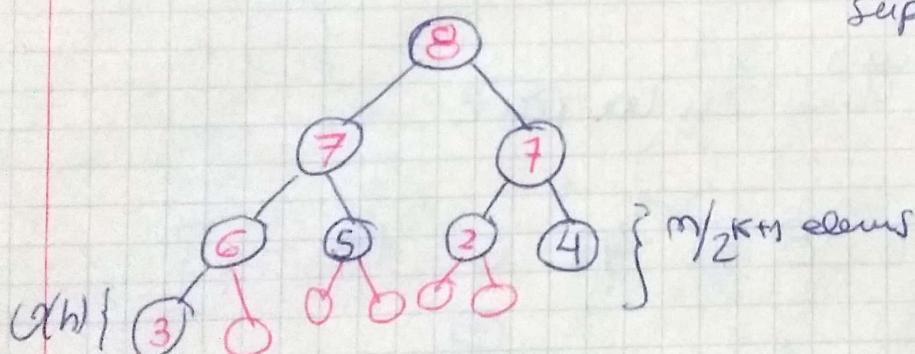
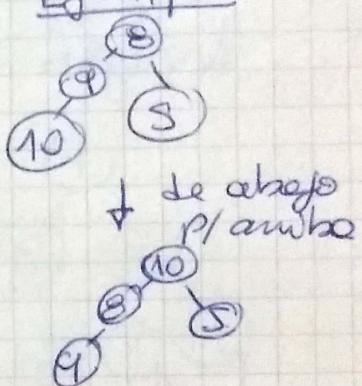
Down heap: Yo puedo estar
rotos pero mis hijos son heap.
De adelante para atrás no
ande. $O(n \log n)$ hacerlo p/todos

Si de atrás p/adelante.

cota superior

me salga
el arreglo con
prop de heap

Ejemplo



elems de arreglos, $m = 2^k \rightarrow h = k$

los de arreglos me cuestan $\frac{m}{2^{K+1}} \cdot O(h)$

$$\Rightarrow T(m) = \Theta\left(\sum_{h=0}^{K-1} \frac{m}{2^{h+1}} O(h)\right)$$

$$T(m) = O\left(m \sum_{h=0}^{K-1} \frac{1}{2^{K+1}} h\right) = O\left(\frac{m}{2} \sum_{h=0}^{K-1} \frac{h}{2^h}\right)$$

$$= O\left(m \sum_{h=0}^{K-1} \left(\frac{1}{2}\right)^h h\right) = O\left(m \sum_{n=0}^{\infty} \left(\frac{1}{2}\right)^h h\right)$$

①

$$= \mathcal{O}\left(n \cdot \frac{1/2}{(1-1/2)^2}\right) = \mathcal{O}(n)$$

Añadir un amigo y darle prop de heapify
heap

Nb conviene Up heap ppio al final $\mathcal{O}(m \log m)$
Down heap final al ppio $\mathcal{O}(m)$
 (peor caso)

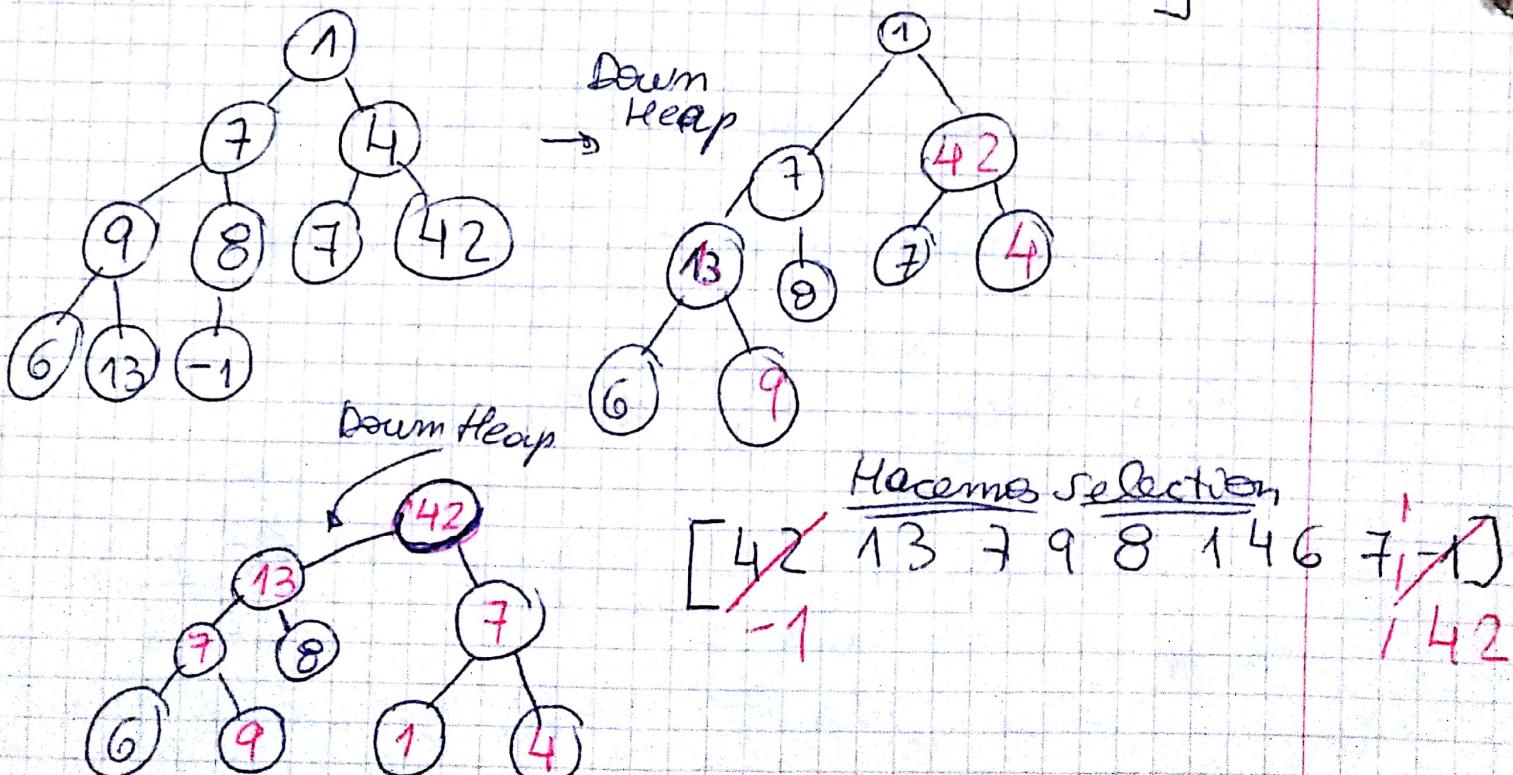
Subir un nivel cuesta 1, 2 niveles me cuesta 2.

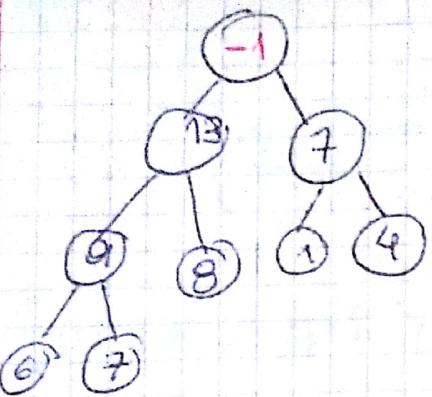
Este es mi punto rápido de crear un heap.

Amigo \rightarrow H \rightarrow Amigo cambiado

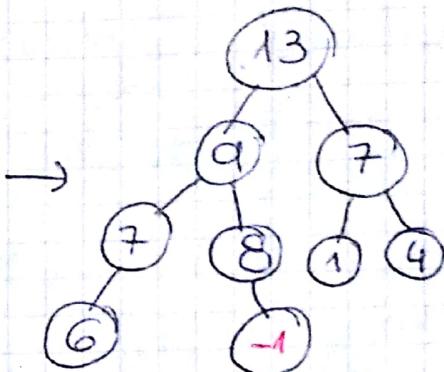
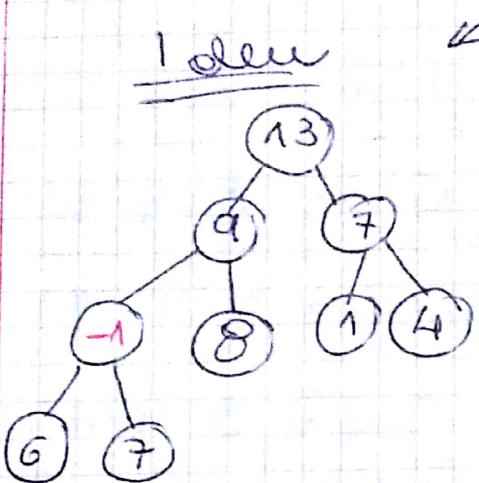
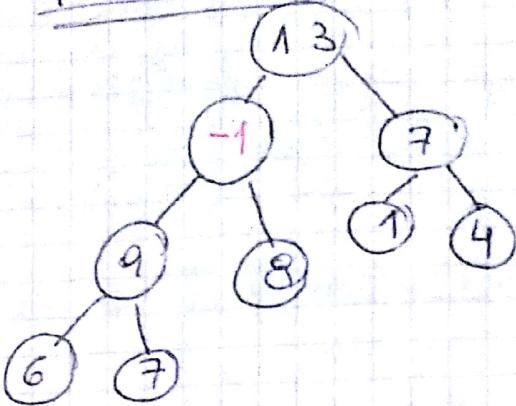
tiempo el más en la raíz.

[1 7 4 9 8 7 42 6 13 -1]



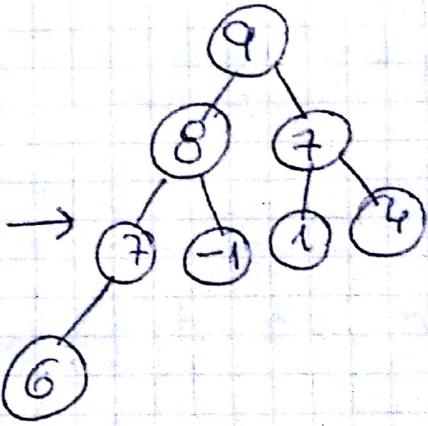
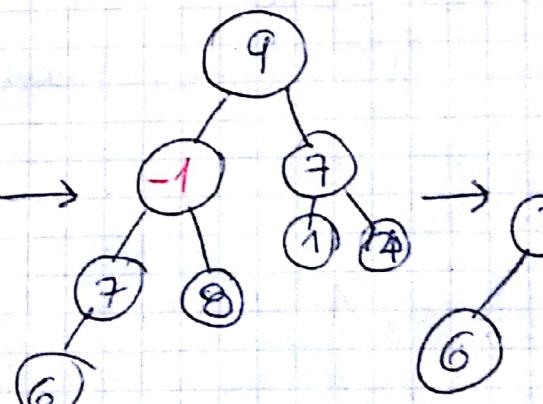
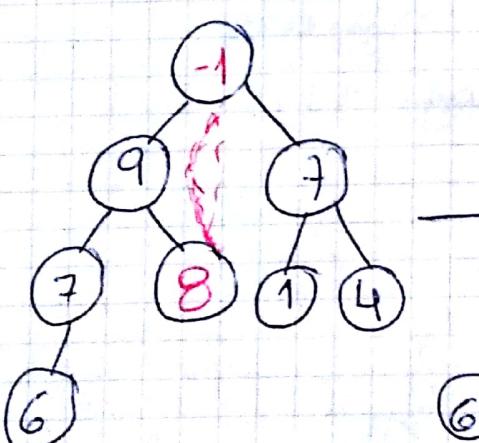


Splittarnos Down-Heap

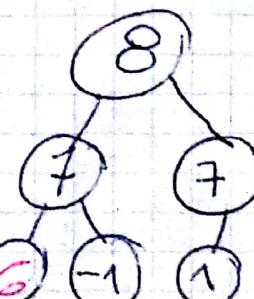
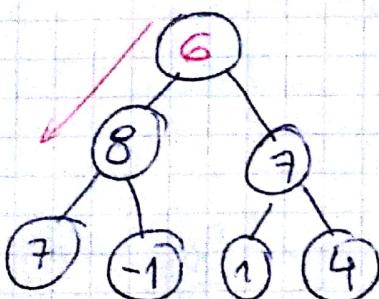


A negado con pop heap.

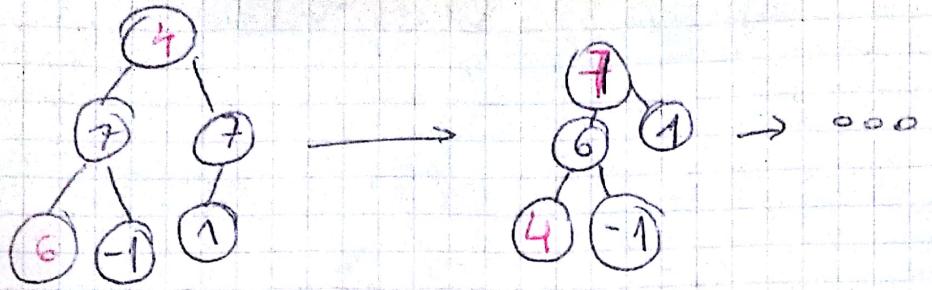
Selection [~~13~~ 9 7 7 8 ~~15~~ 4 6 ~~-1~~ ~~42~~]
-1



Selection [~~9~~ 8 7 7 -1 14 ~~6~~ ~~9~~ ~~13~~]
6 9 42



Selection [~~8~~ 7 7 6 -1 14 ~~9~~ ~~13~~ ~~42~~]
4 8 2



Selection ~~[7 6 14 -1 7 8 9 13 42]~~
~~-1~~
~~7~~

selection [6 4 1 -1; 7 7 8 9 13 42]
~~000~~

~~[4 -1 1; 6 7 7 8 9 13 42]~~
~~000~~

~~[1 -1; 4 6 7 7 8 9 13 42]~~
~~000~~

~~[-1 1 4 6 7 7 8 9 13 42]~~
~~000~~

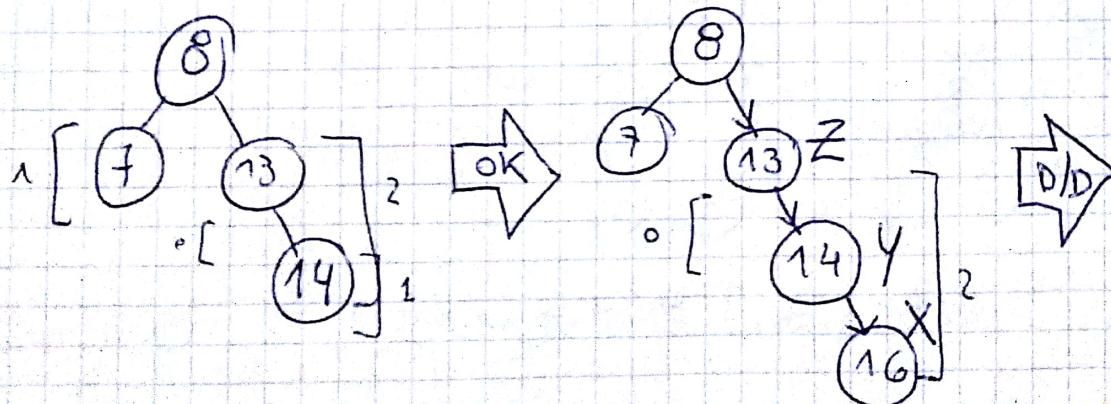
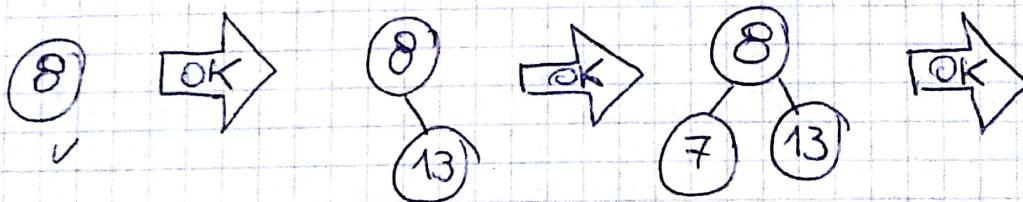
Al final ordenado \rightarrow [-1 1 4 6 7 7 8 9 13 42]

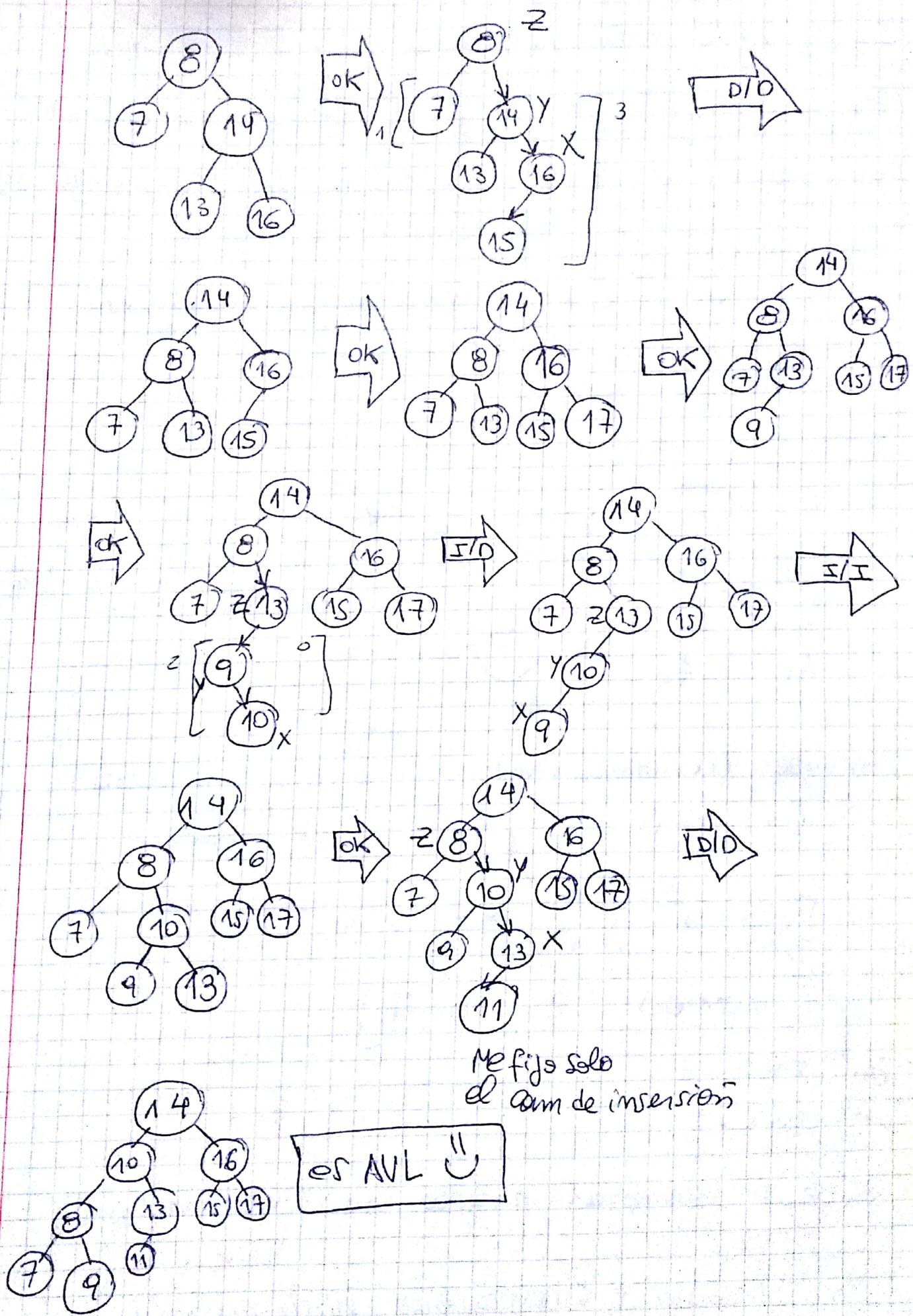
Q sort serio
+ rápido.

Respetando no es estable
pero si es in place.
es. $O(n \log n)$

Práctica AVL

8, 13, 7, 14, 16, 15, 17, 9, 10, 11





3

es AVL !!

In Order: mnos ordenados

Pre orden: 1 4 - 10 - 8 - 7 - 9 - 13 - 11 - 16 - 15 - 17

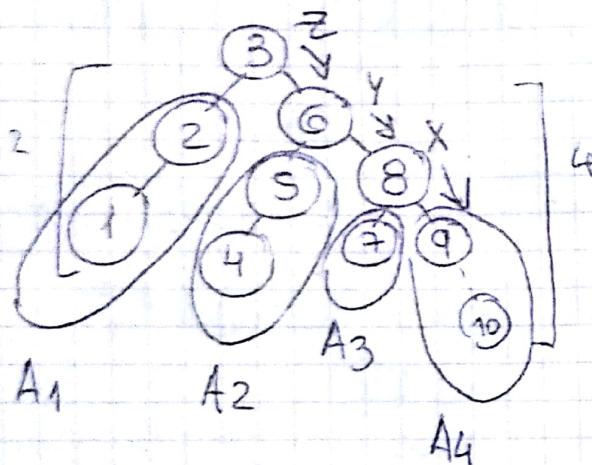
Post orden: 7 - 9 - 8 - 11 - 13 - 10 - ~~15~~ - 17 - 16 - 14

Rotación de raíz

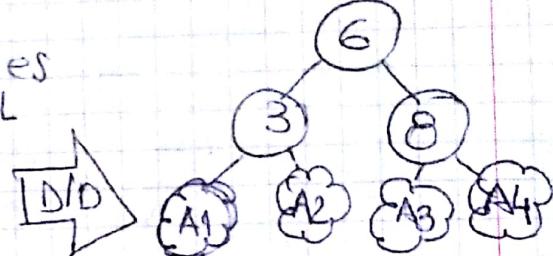
Dado el sgte AVL, cuya raíz es 6:

3 → 2 → 1 → 6 → 5 → 4 → 8 → 7 → 9

Insertar el elem 10.



No es
AVL



1 Armar función pqº

2 [1 3 8 8] \xrightarrow{f}

{
1:1
3:1
8:2 \leftarrow max

3 [5 5 3 2 88 5 4 2] \xrightarrow{f}

[10000 0 10000]

int max = 1

int dove = 5

$$4 - 2 = 2$$

Size_t eliminar_ancho_min(int ancho[],

size_t m) {

hash_t * contadores = hash_crear();

int max = ancho[0];

```

for (size_t i=0; i<m; i++) {
    if (!hash_pertenece(contadores, arreglo[i])) {
        hash_guardar(contadores, arreglo[i], 0);
    }
    int frec = hash_obtener(contadores,
                           arreglo[i]) + 1;
    hash_guardar(contadores, arreglo[i], frec);
    if (frec > hash_obtener(contadores, max)) {
        max = arreglo[i];
    }
}
size_t q_destruir = m - hash_obtener
                    (contadores, max);
hash_destruir(contadores);
return q_destruir;
}

```

Ejercicios 1º
cadenas enl x ddmmaaaa Radix n.
2º ↑ 4º 3º ↑ de bits
LSB MSB

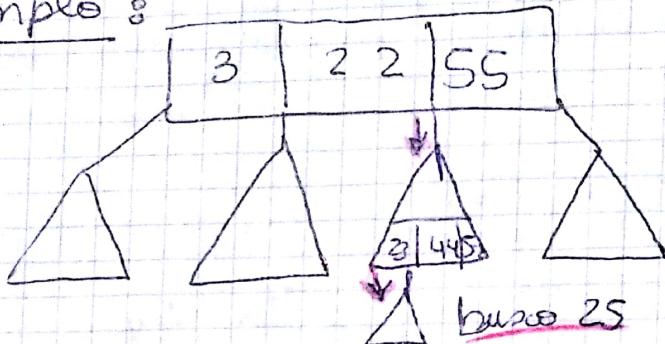
Separar días de meses de año

$$\frac{dd}{10} \quad \frac{mm}{2^\circ} \quad \frac{aa}{3^\circ}$$

Árbol B → generalización del ABB
 ↳ muchas claves x noder,
 muchos rangos.

Cáuse 16

Ejemplo :



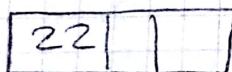
Si busco el
25 voy al
3º árbol.

Insertar en árbol B:

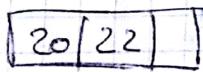
$$t=2$$

esta mén 1 } # claves
n máx 5 } nodos

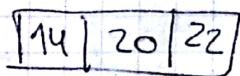
Inserto
22



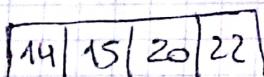
Inserto
20



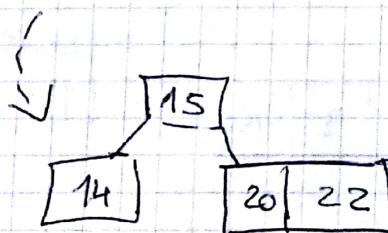
Inserto
14



Inserto
15

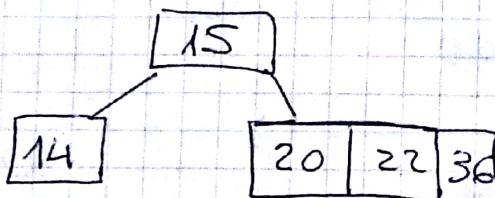


Mediana: 15

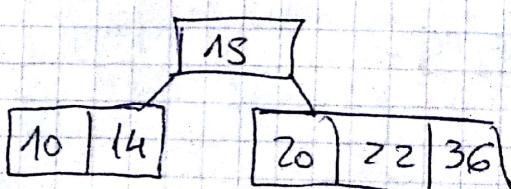


Siempre tratamos de
insertar en las hojas.

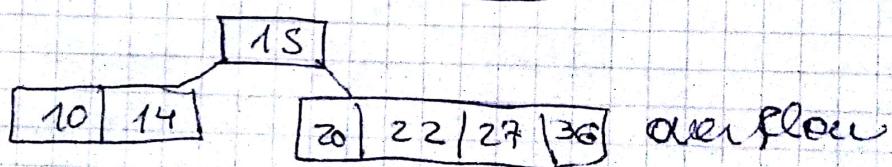
Inserto
36



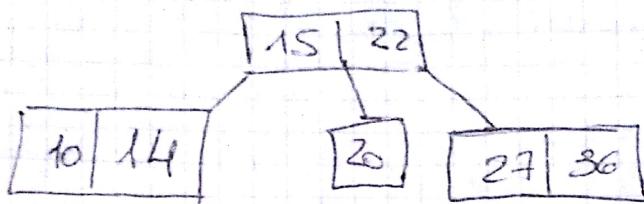
Inserto
10



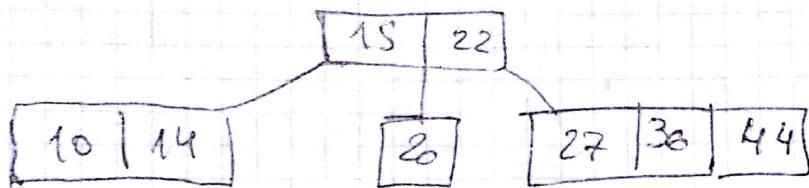
Inserto
27



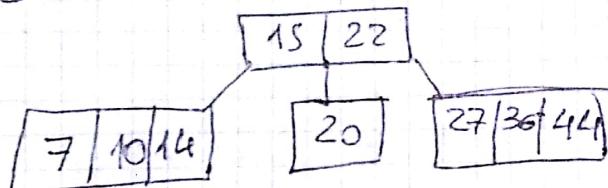
Árbol B es un árbol balanceado, para que no crezca hacia abajo solamente:



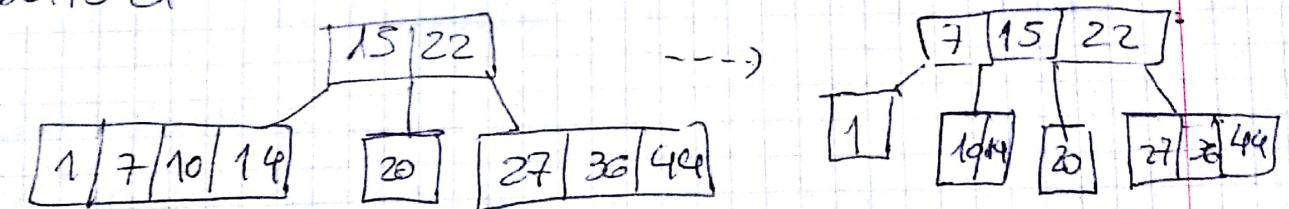
Inserto 44



Inserto 7

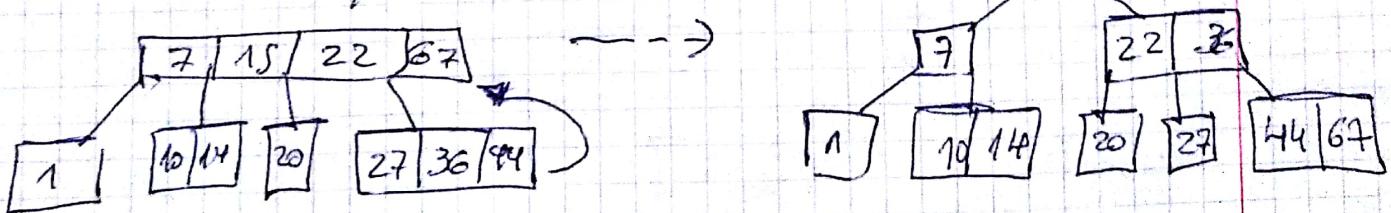


Inserto el 1

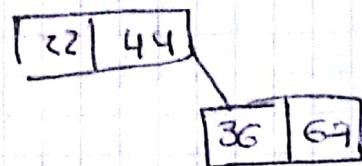


overflow

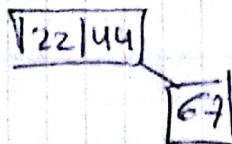
Inserto 67 y lo salgo



Swap



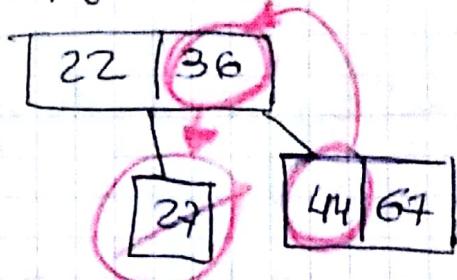
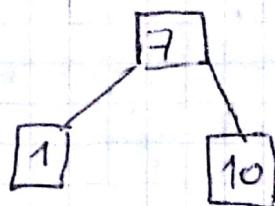
→



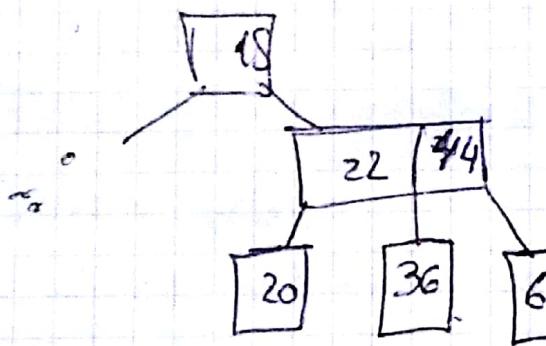
Bono el 14

Del árbol final anterior

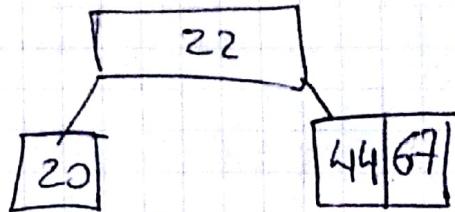
Bono 27°



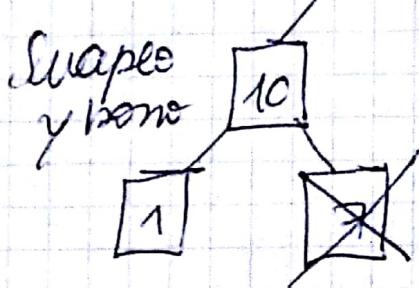
Quiero eliminar 36



→ bajo 44 y bono 36

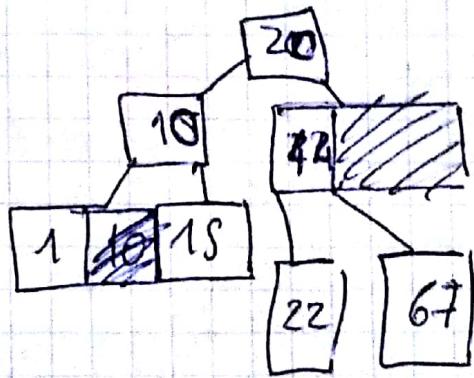


quiero borrar el 7°



pero me quedo
desbalanceado

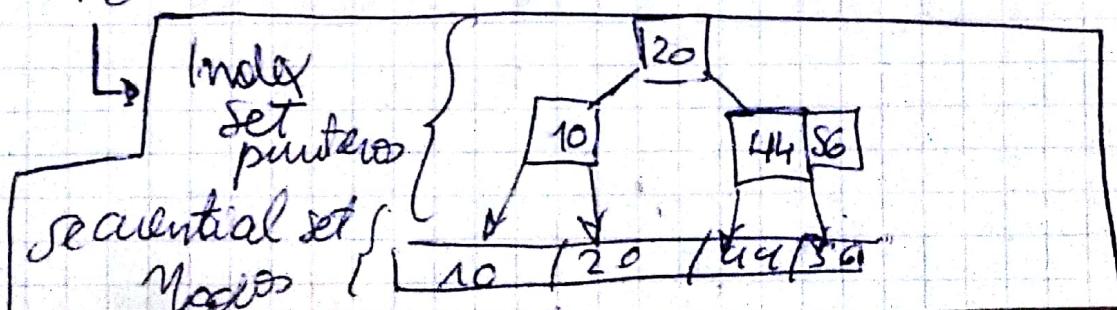
→



Variantes AB:

AB* → aprovechar mejor el espacio (modo entre 75 y 100%)

AB+ → Leer secuencialmente los datos



AB+
es esta
estructura

búsqueda: $\mathcal{O}(\log(n))$

funciones auxiliares:

nodes \rightarrow buscar-nodo (char & clave); {

nodes \rightarrow buscar-padre (char & clave); }

nodes-t * buscar-nodo-y-padre (char & clave, {meyer
nodos-t & *padre}); } }

buscar la unión entre el padre y el hijo.

Grafos → tienen → vértices

↓
→ aristas

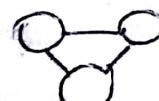
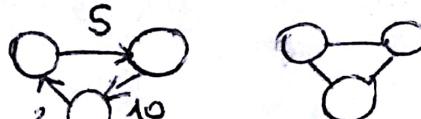
• Todos
los TDA
nos sirven
Salvo el
hash"

Características

→ Dirigidos / No dirigidos (o una u otra)



→ Pesados / No pesados



→ Simple / Compuestos



→ Admite bucles / No admite bucles



Aplicaciones de grafos:

Redes Sociales Sist. de recomendación

Red de actores Mapa para ciudad
(IMDB por ej.) Mapa de mi juego

Cambios de Monedas Modelos de Lewis

Cotaciones de Papelis Análisis de datos

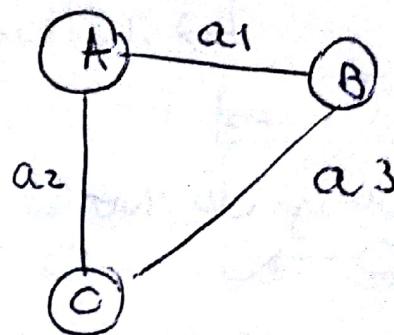
Operaciones sobre grafos

- Agregar / Sacar vértices / aristas
- Obtener vecinos "
- Ver si un vértice \exists
- Ver si 2 vértices están unidos
- Iterar sobre vértices (simetral)

Matriz de incidencia: Es como una tabla que me muestra las relaciones entre vértices

	A	B	C
a ₁	1	1	
a ₂	1	0	1
a ₃	0	1	1

no dirigido,
no pesado

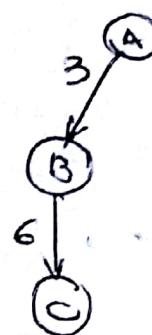


Si fuera pesado

Vamos a tener el peso en lugar de 1.

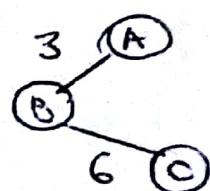
Y si es
pesado
y dirigido:

	A	B	C
a ₁	-3	3	0
a ₂	0	-6	6



y no
dirigido

	A	B	C
a ₁	3	3	0
a ₂	0	6	6



Mínimo de vértices y aristas de un grafo: 0

Esto también
es un grafo: A B

mat aristas (E) = $\frac{V(V-1)}{2}$ donde V es # vértices
no dirigido

en el peso de los pesos $\mathcal{O}(E) \approx \mathcal{O}(V^2)$

- Crear la matriz $\mathcal{O}(V \cdot E)$

- Agregar un vértice $\mathcal{O}(V \cdot E)$
o arista

- Ver si 2 vértices son adyacentes $\mathcal{O}(E)$

- Obtener adyacentes de un vértice

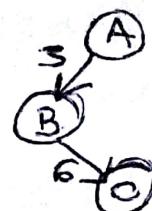
vertice
lista de aristas crea la $\mathcal{O}(E)$

agregar vértice/arista $\mathcal{O}(1)$
adyacentes $\mathcal{O}(E)$

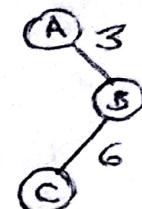
• Matriz de adyacencia

	A	B	C
A	0	3	0
B	0	0	6
C	0	0	0

dengadas



$$\begin{matrix} & \begin{matrix} A & B & C \end{matrix} \\ \begin{matrix} A \\ B \\ C \end{matrix} & \begin{pmatrix} 0 & 3 & 0 \\ 0 & 0 & 6 \\ 0 & 6 & 0 \end{pmatrix} \end{matrix}$$



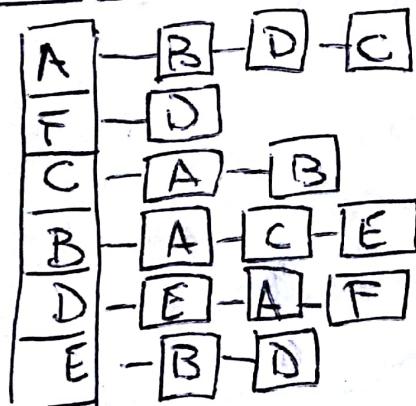
crear $\mathcal{O}(V^2)$, agregar vértice $\mathcal{O}(V^2)$

agregar arista $\mathcal{O}(1)$

Ver 2 vért son ady $\mathcal{O}(1)$

obt ady de un vértice $\mathcal{O}(V)$

• Lista de Adyacencia



En espacio
 $\mathcal{O}(V+E)$

Agregar vértice
 $\mathcal{O}(1)$ si no recorre
y no tiene hijos sumo
 $\mathcal{O}(E)$

Se implementa con:

- >> lista de lista (más usual)
- >> dict de listas (común)
- >> dict de diccionarios (común)
- >> lista de " (no es usual)
 - agregar arista en dict
 - Suponemos que es $\mathcal{O}(1)$

Agregante arista $\mathcal{O}(V)$

Ver vért ady $\mathcal{O}(V)$
y obt ady vert

(2)

Ciclo: Debe tener al menos 2 aristas, me permite
y volver al mismo vértice.

Camino simple: Camino donde no se repiten vértices
(solo init. y fin)

Comp. conexa: Grupos de vértices que se conectan
todas con todas.

grado de salida / entrada:

de aristas que salen/entran

lista_deady
lista_de_list
 $O(V)$

def grado_salida (grafo, v):

Só es mo
diferido es
de adyac.

return len(grafo.adyacentes(v)),

en el no dirigido es grado de vértice, no de
salida o salida ##

matriz de ady $O(V^2)$
matriz de incid $O(E)$

def grado_entrada (grafo, v):

entrada = 0

for u in grafo:

if v in grafo.adyacentes(u):

entrada += 1

return entrada

lista_vistas $O(V^2)$

dicc de listas/lista de dict $O(V \cdot E)$

matriz de incidencia $O(V \cdot E)$

Práctica heap

función recursiva:

```
bool es_heap(int arr[], size_t n) {
    return _es_heap(arr, n, 0)
```

```
}
```

bool _es_heap(int arr[], size_t n, size_t actual){

if (actual >= n)

return true;

int izq = 2 * actual + 1;

int der = 2 * actual + 2;

if (izq <= n && arr[izq] < arr[actual])

return false;

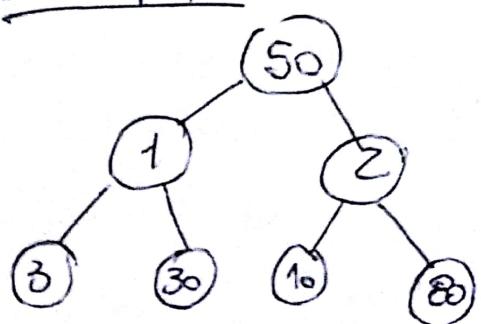
if (der <= n && arr[der] < arr[actual])

return false;

return _es_heap(arr, n, izq)

& & _es_heap(arr, n, der);

Heapify



- Se completa por niveles, de izq a derecha.

$$\begin{aligned} \text{hijo-izq} &= p \times 2 + 1 \\ \text{hijo-der} &= p \times 2 + 2 \\ \text{padre} &= (p-1)/2 \end{aligned}$$

0	1	2	3	30	10	80
---	---	---	---	----	----	----

Heapify:

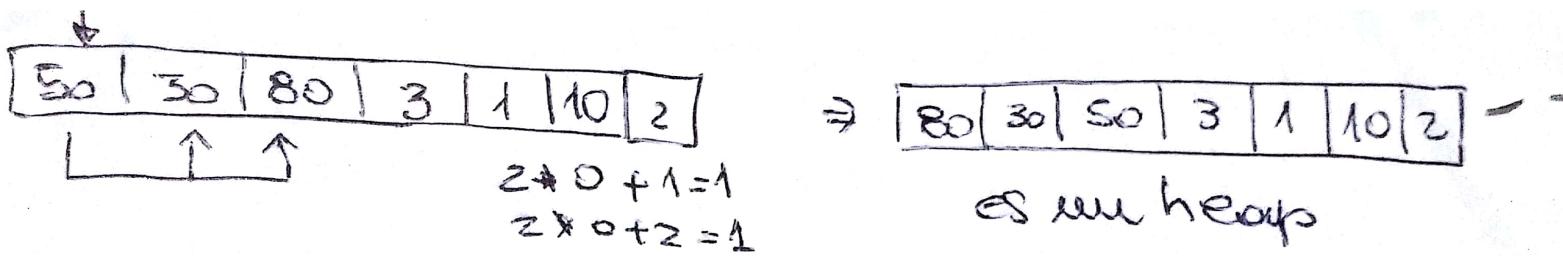
- se las hojas el descenso
- realice cambios ya que no tienen hijos.

50	12	3	30	10	80
$2 \times 2 + 1 = 5$				$2 \times 2 + 2 = 6$	sweeps el + grande
↓	↑↑				

50	1	80	3	30	10	2
----	---	----	---	----	----	---

(3)

$2 \times 1 + 1 = 3$, $2 \times 1 + 2 = 4$



HeapSort (ver dsps)

Ranking: Mete los k elems.

Para los restantes, encola y desencola
el pse ya no se define.

Clases faltadas

18 y 19

BFS

Breadth First Search

Se recorre x niveles

usa TDA Cola

DFS

Depth First Search

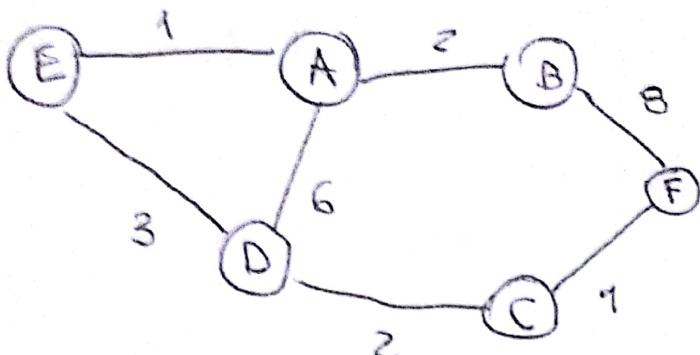
Recorre en profundidad

usa TDA pila

Camino mínimo, Algoritmo de Dijkstra y Centralidad

BFS give
grafo no
pesado → Si es pesoado no
sirve p/cam.
mínimo

Close 20



→ sirve que
cumplimos colo x
heap
(Dijkstra)

def camino_minimo(grafo, origen):

dist = {}
padre = {}

for v in grafo: dist[v] = ∞

dist[origen] = 0

padre[origen] = None

q = Heap()

q.encolar(origen, dist[origen]),

$O(V+E)$

while not q.esta_vacia():

v = q.desencolar()

ace^r → //if v == destino: return reconstruct_caminho(
destino, padre)
destino ← para cada w em adjacents(v):
if dist[v] + grafo.peso_arco(v,w) < dist[w]:
dist[w] = dist[v] + grafo.peso_arco(v,w)

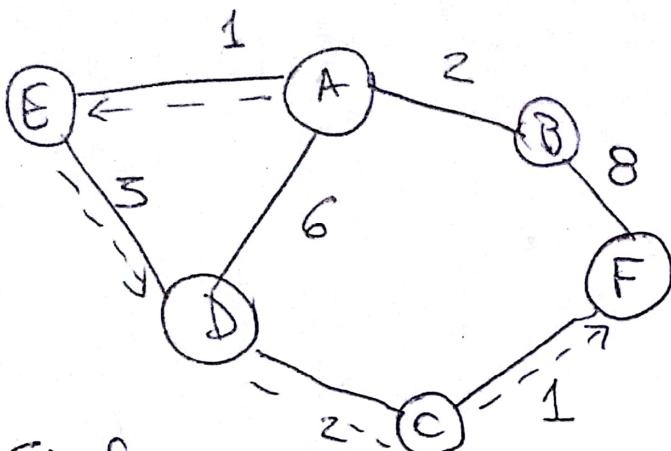
padre[w] = v

q.encolar(w, dist[w]) → actualiza
(w, dist[w])

return padre, dist

①

Camino mínimo
desde A a F



Si el grafo es más denso y tiene alguno con 1 peso neg
⇒ ya no camina.

Padre

A: Name

dist

A = 0

B = ∞

C = ∞

D = ∞

E = ∞

F = ∞

Padre

A: Name

E = A

dist

A = 0

0

0

E = 1

0

0

Heap

E: 1

H

P

E = 1

A: Name

A = 0

D = 6

E = A

B = 2

B = 2

D = A

C = ∞

D = A

E = 6

B = A

F = ∞

E = 1

F = 00

P

A: Name

dist

A = 0

E = A

0

D = A

D = 6

HM

E = 1

E = 1

D = 6

0

H

P

D = 4

A: Name

dist

A = 0

D = 6

D = E

B = 2

B = 2

B = A

C = ∞

D = 4

E = 1

F = ∞

H

P

D = 4

A: Name

dist

A = 0

D = 6

E = A

B = 2

F = 10

D = E

C = ∞

B = A

O = 4

D = 4

E = 1

E = 1

F = B

F = B

F = B

F = 10

H

P

C = 6

A: Name

dist

A = 0

D = 6

E = A

B = 2

F = 10

D = E

C = 6

B = A

D = 4

F = B

E = 1

C = D

F = 10

H

P

F = 7

A: Name

dist

0

F = 10

B = 2

0

F = C

C = 6

0

desp del heap queda vacío

(2)

$\mathcal{O}(E \log E)$

$\mathcal{O}(E) \leq \mathcal{O}(V^2)$

$$\rightarrow \mathcal{O}(E \log E) = \mathcal{O}(E \log V^2) = \mathcal{O}(2E \log V) = \mathcal{O}(E \log V)$$

Centralidad → medida de import del vértice dentro
↓ de un grafo

4 tipos pero solo los vamos a tratar juntos con 1.

- De grado: "Cuán pesado es q' un vértice recibe
distr. de info" en el m' denigrado peso.

• Cercanía: $\mathbb{E}[\text{dist}]$

• De caminos y caminos: De matrices de adyacencia.

- De intermedio: La fuerza q' aparece el
(Betweenness Centrality)

A priori:

1 def centralidad(grafos):

2 cent = {}

3 for v in grafos: cent[v] = 0

4 for v in grafos:

5 for w in grafos:

6 if v == w: continue

7 padre, dist = columnas_minimas(grafos, v, w)

8 if w not in padre: continue

9 actual = padre[w]

10 while actual != v:

11 cent[actual] += 1

12 actual = padre[actual] (3)

13 return cent

$\mathcal{O}(V^2 \cdot \text{columnas_min}) \rightarrow \mathcal{O}(V^2 E \log V)$ (3)

Ordenar de $a \leq x$ dist. \rightarrow 1º hjo recorre el perde

def centralidad (grafo):

cent = {}

$\mathcal{O}(v)$ [for v in grafo: cent[v] = 0

for v in grafo:

$\mathcal{O}(\text{cam_min})$ | perde, dist = camino_minimo(grafo, v)

cent_aux = {}

$\mathcal{O}(v)$ | for w in grafo: cent_aux[v] = 0

$\mathcal{O}(v)$ | filtrar_infinito(dist)

$\mathcal{O}(\text{ord})$ [vertices_ordereds = ordenar_vertices(grafo, dist)

$\mathcal{O}(v)$ [for w in vertices_ordereds:

cent_aux[perde[w]] += 1

cent_aux[perde[w]] += cent_aux[w]

$\mathcal{O}(v)$ [for w in grafo

if $w == v$: continue

cent[w] += cent_aux[w]

return cent

no pesos

orden?

$\mathcal{O}(V+E)$ BFS
(no pesos)

$\begin{bmatrix} 2 & 5 \\ A & B & C & D & E & F \end{bmatrix}$

$\mathcal{O}(V)$ (cam_min + ord))

$|V|$ dist max

si es cl

pesos (sin peso)

mismo centro

$\mathcal{O}(d+n) = \mathcal{O}(V)$

$\begin{bmatrix} F & E & D & B & A & C \end{bmatrix}$

perde: C - perde: D

y le sigue

tmb

a todos

los q

ns de

inten

orden pesos

$\mathcal{O}(V \cdot cam_ord)$

$\mathcal{O}(E \log V)$

$\mathcal{O}(V+E \cdot \log V)$

$\mathcal{O}(V \cdot \log V)$

$\mathcal{O}(V \cdot (V+E) \cdot \log V)$

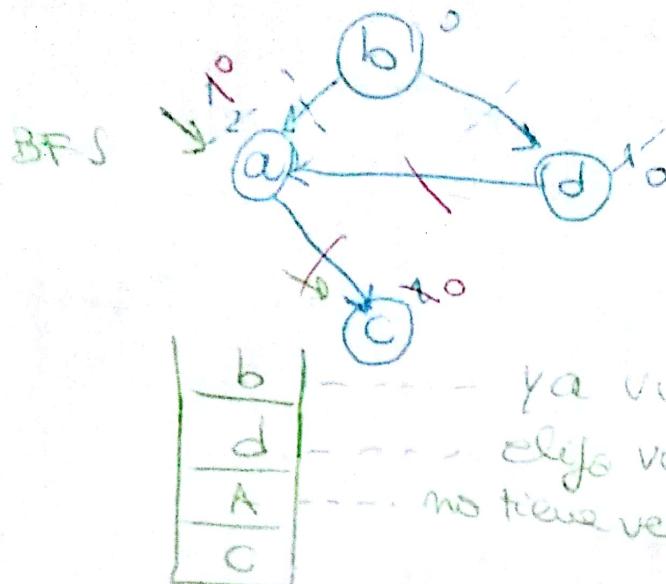
$\mathcal{O}(V \cdot (V+E))$ si es grafo
no pesos

$(V(V+E) \log V)$
(E grupos conexos)

vs $O(V^2 E \log V)$

• Práctica →

[baa, abcd, abc, a, cab, cad]



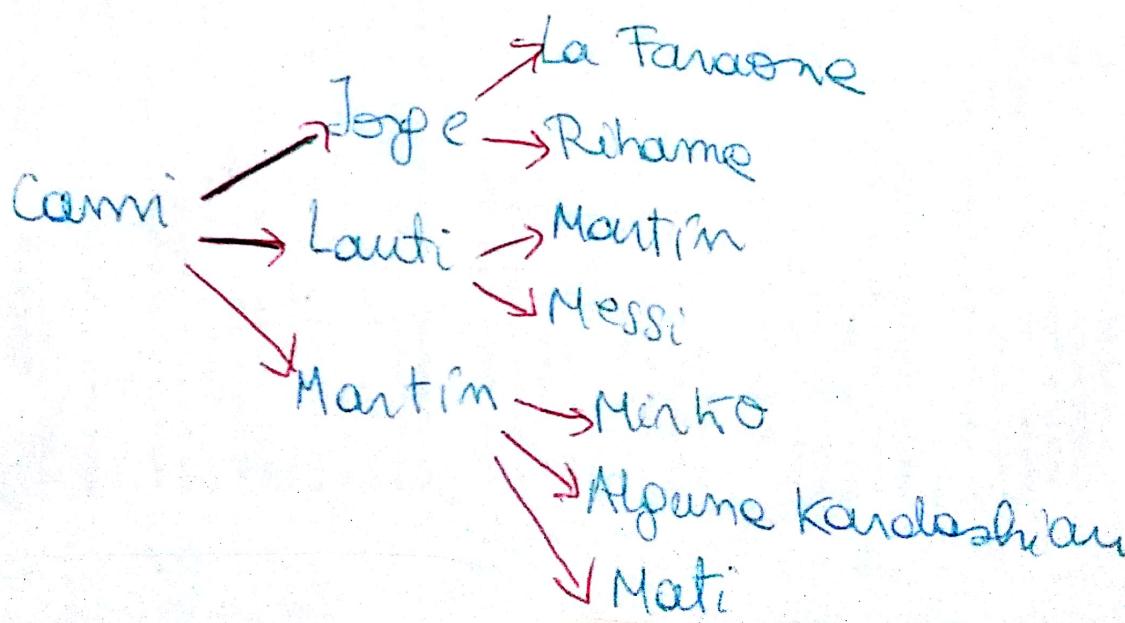
b | d | a | c
est
(V+E)

ya visité sus amigos.
elijo visitar b. a ✓ evitó
no tiene vecinos cuando visitó c.

Recomendar

Cami → Lauti → Mati

recomendación:



5

Diccionarios a usar:

~~Postre~~, orden y visitados

Va a recomendar a todos menos a Martín xq ya
↳ visitaste.

```
def instagram_recomendaciones(grafos, origen):
    visitados = {}
    recomendaciones = {}
    orden = {}
    q = cole()
    visitados[origen] = true
    orden[origen] = 0
    q.append(origen)
```

while not q.esta_vacio():

v = q.desencolar()

for w in grafos.adyacentes(v)

if w not in visitados:

orden[w] = orden[v] + 1

if (orden[w] == 2)

recomendaciones.agregar(w)

else

q.encolar(w)

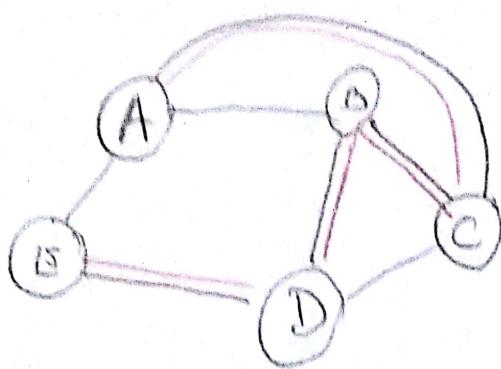
visitados[w] = true

return recomendaciones

Cami	Martín	Martín
	Lauti	Lauti
	Jorge	Xxgg

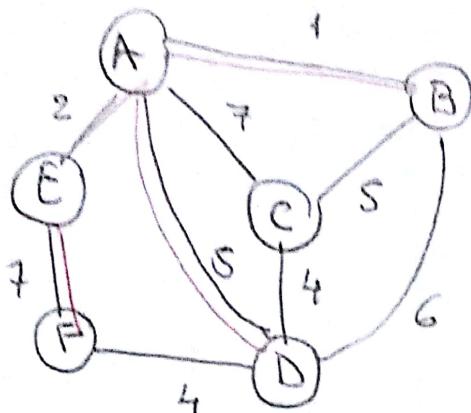
Árboles de Tamano Mínimo (MST)

Clase 21



$$V \\ E = V - 1$$

grafo indirectamente conexo



- 1) Suma = 24
- 2) Suma = 19

Obtener su ciclo con alg. BFS/DFS.

$O(V + E)$

Algoritmo x fza bruta = $O(E(V+E))$

mejor aproximación al problema del viajante

def prim(grafo): // Algoritmo de Prim

vertice=grafo.vertice_aleatorio()
visitados=set()

visitados.agregar(vertice)

q=heap.create()

for w in grafo.adyacentes(vertice):
q.encolar((vertice,w),grafo.peso_arista(vertice,w))

arbol=grafo.clear(grafo.obtener.visitados())
while not q.esta_vacia():

if (v,w)=q.desencolar():

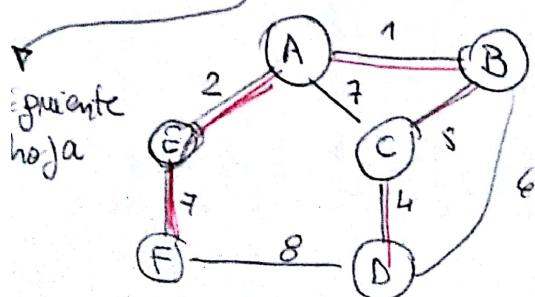
if w in visitados: continue

Partimos de un vértice al azar y nos movemos x la pescada.

Quindamos todos los vecinos y metemos el heap de mínimos este

Vacío, descolamos.

No genera un ciclo



Visitados: B, A, E, C, D, F

busca anistas + chicas



Más o menos

Voy segundo Visitados: B
las W's son chicas y chicas lo
Visitados Heap (B, C, S), (B, D, G) más
(A, E, Z), (A, C, 7), la letra como
(E, F, 7) (A, D, S) ① visitas

No busque obtener los caminos mínimos.

dentro del while

arbol_agregar_arista(v, w, grafo_pesos_arista(v, w))
visitados_agregar(w)

for x in grafo.adyacentes(w):

if x not in visitados: q_en_arbol((w, x),
grafo_pesos_arista
(w, x))

return arbol.

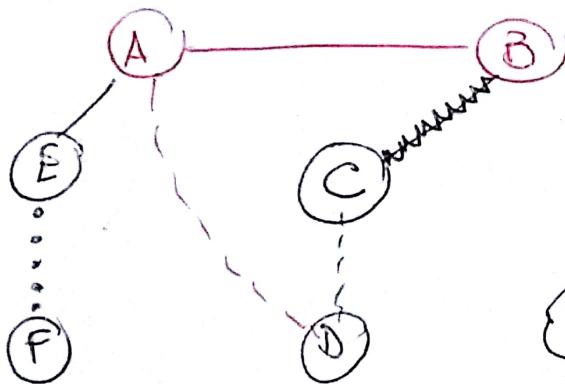
en el peor caso encolo todos los aristas

$$E \in \mathcal{O}(E \log E) = \mathcal{O}(E \log V)$$

Algoritmo de Kruskal

Agarré E → ordeno → me fijo de no
meter ciclos en la WST

~~(A, B, 1)~~ ~~(A, E, 2)~~ ~~(C, D, 4)~~ ~~(A, D, 5)~~ ~~(C, B, 5)~~ ~~(B, D, 6)~~
~~(E, F, 7)~~ (A, C, 7) (F, D, 8) saltos



Va a encontrar algún
árbol de tendido mínimo.

¿Cómo saber que 2 vért.
están en la misma componente?

Unión Final

	1	2	3	4	5
A	0	X	2	3	4
B		X			
C			2	3	4
D				3	4
E					5
F					

(A, B, 1)

Find: $\mathcal{O}(1)$

Union: $\mathcal{O}(1)$

Andar: $\mathcal{O}(E \log V)$

Implementación 2

tu comp es
la misma que E
con B

0	1	2	3	4	5
0	1	2	3	4	5

A B

Find: " $\mathcal{O}(n)$ " $\sim \mathcal{O}(1 \log n)$ Disjoint Set
Union: $\mathcal{O}(1)$

ef Kruskal (grafo) : $\rightarrow O(E \log V)$

$O(V)$ (conjuntos = conjuntos disjuntos _crear (grafo, obtener vertices))

aristas = sort (obtener_aristas(grafo)) $\rightarrow O(E \log E)$
 $= O(E \log V)$

arbol = grafo._crear (grafo, obtener_vertices())

for a in aristas:

^{no fin} $v, w, peso = a$

if conjuntos._find(v) == conjuntos._find(w):
 continue

arbol._agregar_arista(v, w, peso) $\rightarrow O(1)$

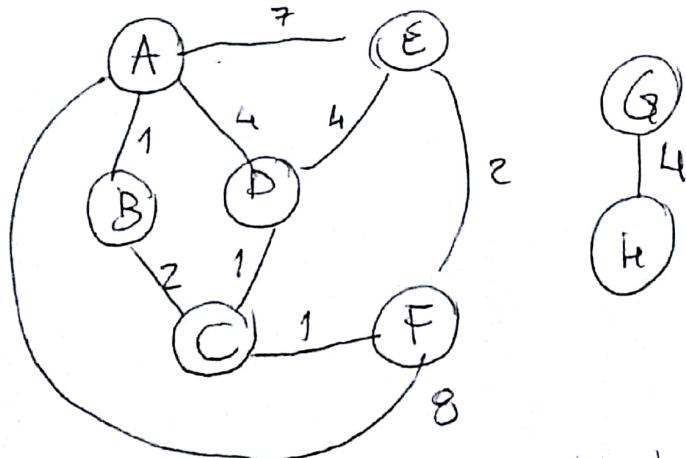
conjuntos._union(v, w) $\rightarrow O(1)$

return arbol.

— Práctica —

Dijkstra usa heap

BFS usa cole



A	B	C	D	E	F	G	H
A	∞						
B	1A	∞	∞	7A	8A	∞	∞
C	1A	2	1A	7A	8A	9	0
D	1A	2	1A	7A	4C	0	0
F	1A	2	1A	7A	4G	0	0
E	1A	2	1A	6F	4G	6	0

↓ desorden

empate me
quedo con
histórico

V	Padre	d(A, v)
A	None	0
B	A	1
C	B	3
D	A	4
E	F	6
F	C	4
G	—	∞
H	—	∞

$O(E \log V)$

VS
BFS
 $O(V+E)$

Centralidad → qué tan importante es un vértice en un grafo

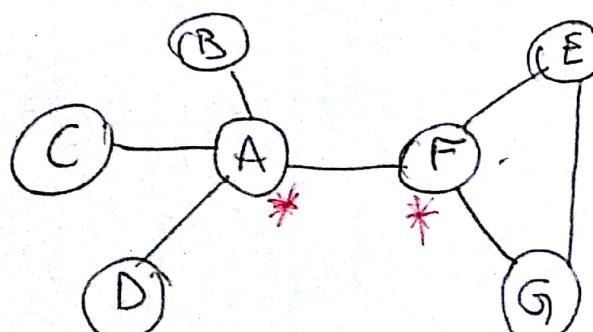
$C(x) = \sum_{y \neq x} \frac{\text{camino min}}{N-1}$

→ Al que le entran + es un centro.

→ Closeness: Tiempo un nodo y me fijo todos los caminos mínimos y me fijo cual es la dist. prom.

$B(x) = \sum_{y \neq x} C(y, x)$

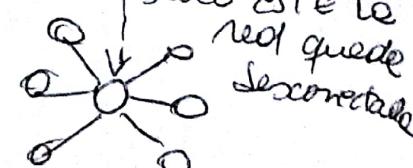
→ Betweeness: Nodo que es vital y me acelita la comunic. en la red.



Lista de diccionarios de padres

¿De dónde es la ejec

	A	B	C	D	E	F	G
A	∅	A	A	A	F	A	F
B	B	∅	A	A	F	A	F
C	C	A	∅	A	F	A	F
D	D	A	A	∅	F	A	F
E	F	A	A	A	∅	E	E
F	F	A	A	A	F	∅	F
G	F	A	A	A	G	G	∅



$(V \# E \log V)$
 para ver la ruta
 el orden
 Si BFS
 $(E + V)$

def caminos_minc(graf, v)
 ↑
 Matriz bono

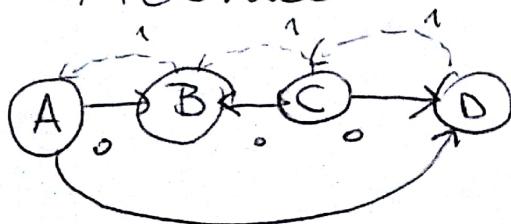
Con el dicc de padres puedes ir
 reconstruyendole el camino.

$A \rightarrow F \rightarrow G$
 contador_apen = {}
Condición de corte: \emptyset no tiene

cont_apen[G] ++
 cont_apen[F] ++,
 cont_apen[A] ++

```

def reconstruir_caminos(padres_orig, orig, dest):
    recorrido = [dest]
    while padres_orig[dest] != None:
        dest = padres_orig[dest]
        recorrido.append(dest)
        cont_apen[dest] += 1
    return recorrido.
  
```



pesar los aristas
 nuevas.

↓
 Luego función Dijkstra → minimiza

Algoritmos Greedy → Busca que una sucesión de óptimos locales nos lleven a uno global

Deontologismo vs Utilitarismo

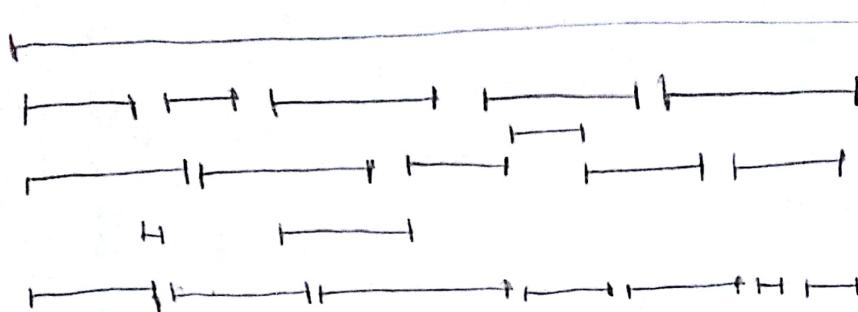
Pueden no llegar al óptimo pero van a tener ser óptimos.

Clase 24

- Fáciles de entender, usar, busca los cosas rápido.
- Prim, Dijkstra, Kruskal.

Scheduling

Busco el que termina antes y me fijo si se solapan, si no lo agrego.



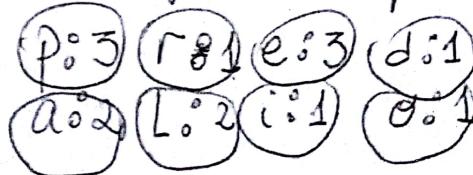
def scheduling(horarios)

$O(mn)$ { horarios_ordenados = ordenar_por_tiempo_fim(horarios)
charlas = []

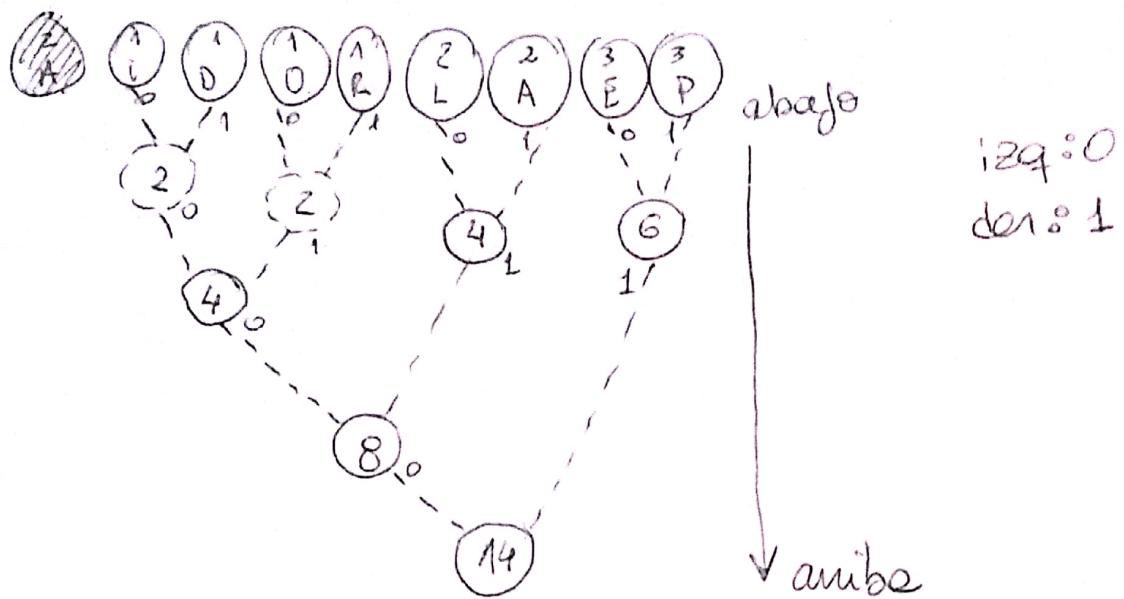
for horario in horarios_ordenados:
 if len(charlas) == 0 or not hay_intersección(
 charlas[-1], horario):
 charlas.append(horario)
return charlas

Cambiar lo que ocupan estas cosas que aparecen seguidas comprimir

Algoritmo de Huffman para comprimir los palabras paralelepípedo.



creamos nodos y los dejaremos en un heap de mínimos ①



11 010 011 011 010 10 010 10 11 0000 11 10 000
 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
 P A R A L E L E P I P E D

5 bytes.

0 0 0 0
 .

Comprender y Descomprender generan el mismo árbol.

def huffman(texto):

frecuencias = calcular_frecuencias(texto) } $\mathcal{O}(m)$

q = heap_crear()

for carácter in frecuencias:

q.encolar(Noja(carácter, frecuencia))

while q.cantidad() > 1:

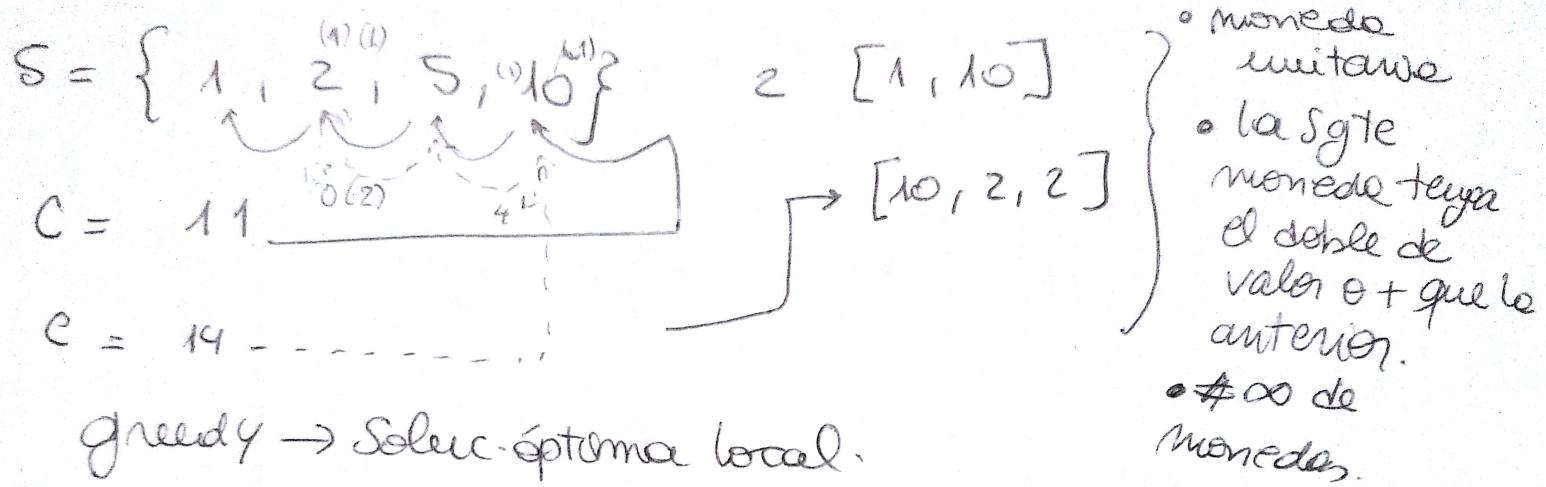
t1 = q.desencolar() } $\mathcal{O}(\log(m))$

t2 = q.desencolar() } $\mathcal{O}(\log(m))$

$\mathcal{O}(\log m)$ { q.encolar(Arbol(t1, t2, t1.frecuencia + t2.frecuencia))}

return codificar(q.desencolar()).

(2)



greedy \rightarrow Soluc. óptima local.

Clase 25

Fuerza Bruta y Backtracking

"Poder el árbol"

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

ej: ubicar
nuevas de
tal forma
que no
se conocen entre si.

0	0	0	0
0	0	0	0
0	0	0	0

↓
no es
SC

↓
... sigue así,

no es
SC

Me pongo en mi lugar,
y trato de generar
todas las posibilidades,
del resto se encarga
la recursividad.

```
def reinas (tablero, fila, m):  
    if fila == m:  
        return es_compatible (tablero, m)  
    if not es_compatible (tablero, m):  
        return False; //poder
```

FOR col in range(m):

tablero [fila] [col] = 1

if reinas (tablero, fila + 1, m):

return True

else:

tablero [fila] [col] = 0

return False

"N" dados y un n. "S" dan todos los comb de dados que tienen
caso base: no tengo + dados S.

def dodos(m, s, solucion_parcial):

if m == 0:

if sum(solucion_parcial) == s:

return [solucion_parcial].

else: return []

Ranote:
Backtracking.

(faltante = s - sum(solucion_parcial))

if (m > faltante or 6 * m < faltante): return []

Soluciones = []

for i in range [1, 7]:

Soluciones.append(dodos(m-1, s,
solucion_parcial + [i]))

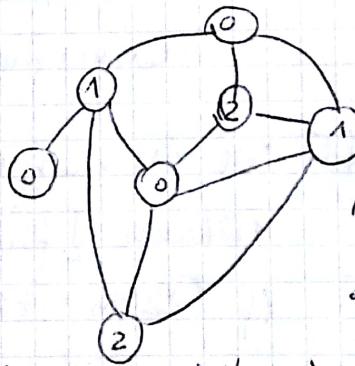
return soluciones.

4	2	1	3
3	1	4	2
1	4	3	.
2	3	.	1

→ No.

Ejercicios de colores

colores = [Rojo, Azul,
Negro]



Voy y vuelvo hasta
conseguir el error

Algoritmo q
hace backt. ~

• Crea los vértices

• Puede

• avanzar en
caso de poder.

• deshacer.

def colores(G, colores, origen, visitados):

if len(G) == len(visitados):

return True.

for color in colores:

if se_puede(G, color, origen, visitados)

visitados[origen] = color

if colores(G, colores, G.ady(origen), visitados)

visitados.delete(origen)

return False

funcion
se_puede
que nos
dice si

debe volver
en vert
no visit

def colores(G, colores)

origen = G.vertex()

visitados = {}

for vertice in grafos:

if not vertice in visitados:

if colores(G, colores, origen, visitados)

return False.

return True

def se_puede(G, colos, origen, visitados):

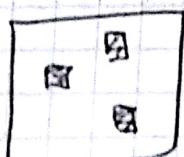
for w in G.adyacentes(origen):

if visitados[w] == visitados[origen]:

return False

return True

laberinto → 2 if → me muevo p/derecha o p/abajo



0: puedo pasar
1: no puedo

Programación Dinámica → Top-Down

$f(n) = f(n-1)$ → repite muchas veces lo mismo → Memorization

def fibonacci(m):

 if $m \leq 1$: return 1

 else: return fib(m-1) + fib(m-2)

def fib(m, Mem):

 if m in Mem:

 return Mem[m]

 if m < 2

 valor = 1

 else:

 valor = fib(m-1, Mem) + fib(m-2, Mem)

 Mem[m] = valor

 return valor

Vamos a hacerlo de forma iterativa

def fibonacci(m):

 v = [None] * (m+1)

 v[0] = v[1] = 1

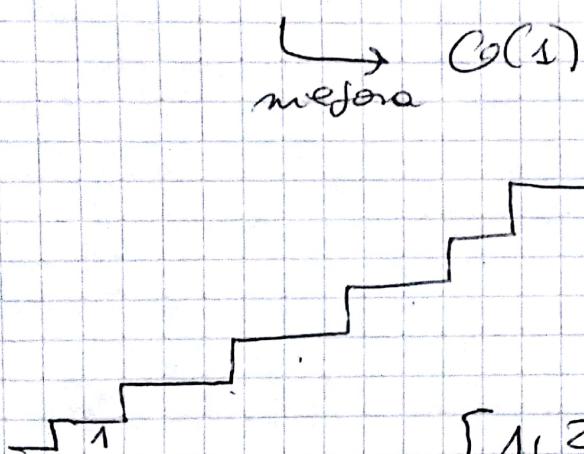
 for i in range(2, m+1):

 v[i] = v[i-1] + v[i-2]

 return v[m-1]

$$V[m] = \begin{cases} 1 & \text{si } m \leq 1 \\ V[m-1] + V[m-2] & \text{eoc} \end{cases}$$

Espacio: $\mathcal{O}(m)$



$$\text{Escalones } (m) = \begin{cases} 1 & \text{si } m=1 \\ 2 & \text{si } m=2 \\ 4 & \text{si } m=3 \\ \text{Esc}(m-1) + \text{Esc}(m-2) & \text{eoc} \\ + \text{Esc}(m-3) \end{cases}$$

$$\{1, 2, 4, 7, 13, 24, 44, 81, 149, 274\} \quad (\mathcal{O}m)$$