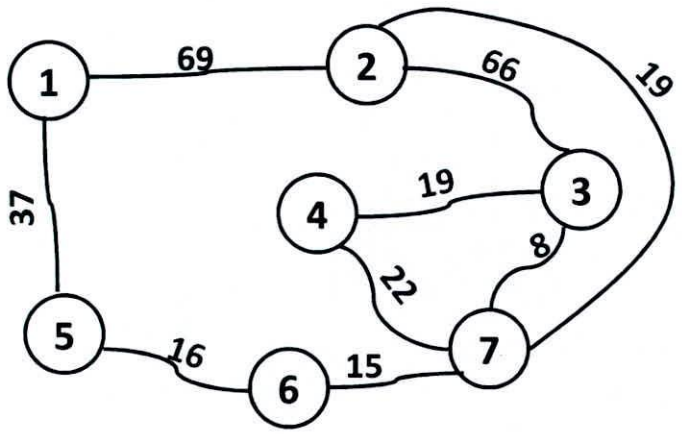
Algoritmos y Programación II, cátedra Calvo.

Final tomado en julio-17 (otro)

1. Dado el siguiente grafo, hacer otro con recorridos mínimos. ¿qué algoritmo utilizó? Describirlo y decir con qué estructura se implementaría.



2. Definir AVL. Realizar el alta de la secuencia:

   3 4 16 32 10 25 92

   Definir balance en un árbol binario. Definir un algoritmo que indique si dos AVL son idénticos.

3. Definir heap. Dar el alta de heap máximo de:

   21 77 96 21 47 1 89 71 53 30

   ¿Cuál es el algoritmo más eficiente para el alta? ¿Cuál es su coste? ¿cómo sería una baja?

4. Dada la ecuación

   $$T_{(n)} = T_{(n-1)} + n \qquad con\ T_{(0)} = 1$$

   Hallar su coste y mostrar cómo se llegó a tal resultado. Definir un algoritmo que tenga ese coste.

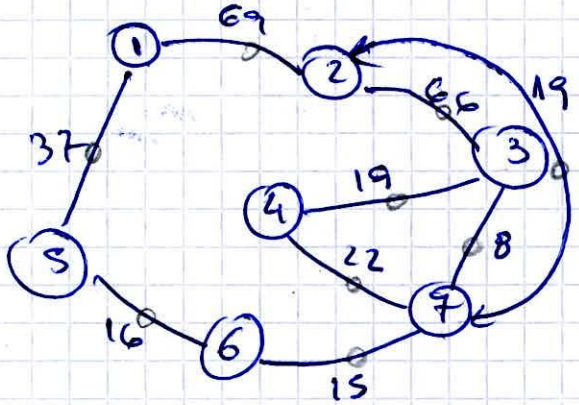
5. Definir hashing. ¿Qué procedimiento utilizar para la búsqueda si se usa un método de direccionamiento abierto para solución de colisiones?

Para aprobar es necesario tener correctos y completos el 60% de cada ítem propuesto.
En la nota se ponderan, también, los resultados de los parciales y trabajos prácticos.
Duración del examen: 2 horas.

Final 2

Julio 17

① Dado el grafo, hacer otro con recorridos mínimos. ¿Qué algoritmo utilizó? Describirlo y decir con qué estructura se implementaría

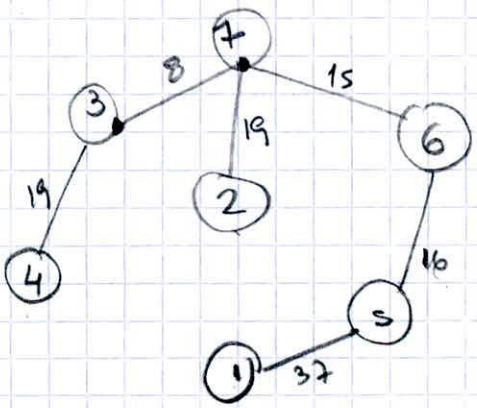


Lo resolvería con un heap de mínima, ordenado por clave = peso arista

KrusKal:
- Armar listado de aristas por peso (en forma creciente)
- Mientras $A < N-1$

{ elegir arista peso mínimo,
si arista no forma ciclo (u, r)
entonces
incorporar vértice/s menor al árbol
}

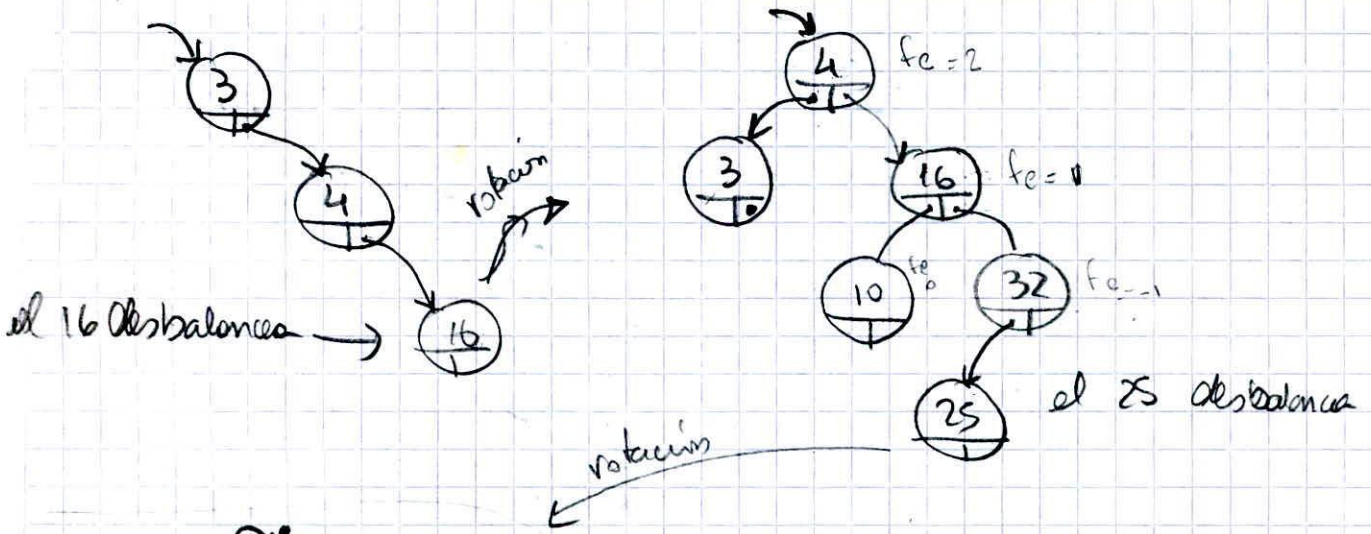
arista	peso
(3,7)	8
(6,7)	15
(5,6)	16
(3,4)	19
(2,7)	19
(4,7)	22 ciclo
(1,5)	37
(2,3)	66 ciclo
(1,2)	69

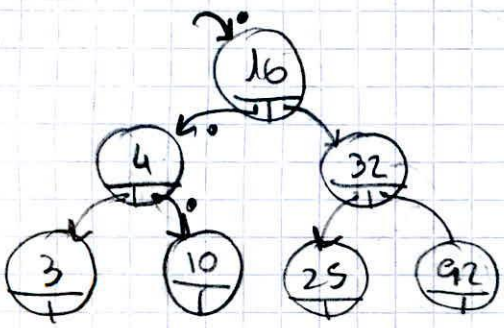


② Definir AVL. Realizar el alta de la secuencia 3 4 16 32 10 25 92.
Definir balance en un árbol binario.
Definir un algoritmo que indique si dos AVL son idénticos

AVL = árbol binario cuyos nodos tienen, a lo sumo, 2 hijos.
La clave de cada nodo es mayor que toda clave del subárbol izq y menor que toda clave del subárbol derecho

Este árbol tiene la particularidad de que está siempre balanceado, con dif de alt entre subárb. izq. y derecho por 0 o 1. Al ingresar una clave que produce que el balance tenga una dif mayor que 1, se hacen rotaciones (dobles o simples, según el caso) para balancearse





Balance de árbol binario:

es cuando la dif. de alturas entre subárbol izq. y derecho es menor que a (un n dado según el balance que se quiera tener)

```
bool igualdadAVL ( NodoArbol* arbol1, arbol2) {
  bool esIgual = verdadero
  Mientras (arbol1 && arbol2 && esIgual) {

    Recorrer SubArbIzq ()
    Reco
```

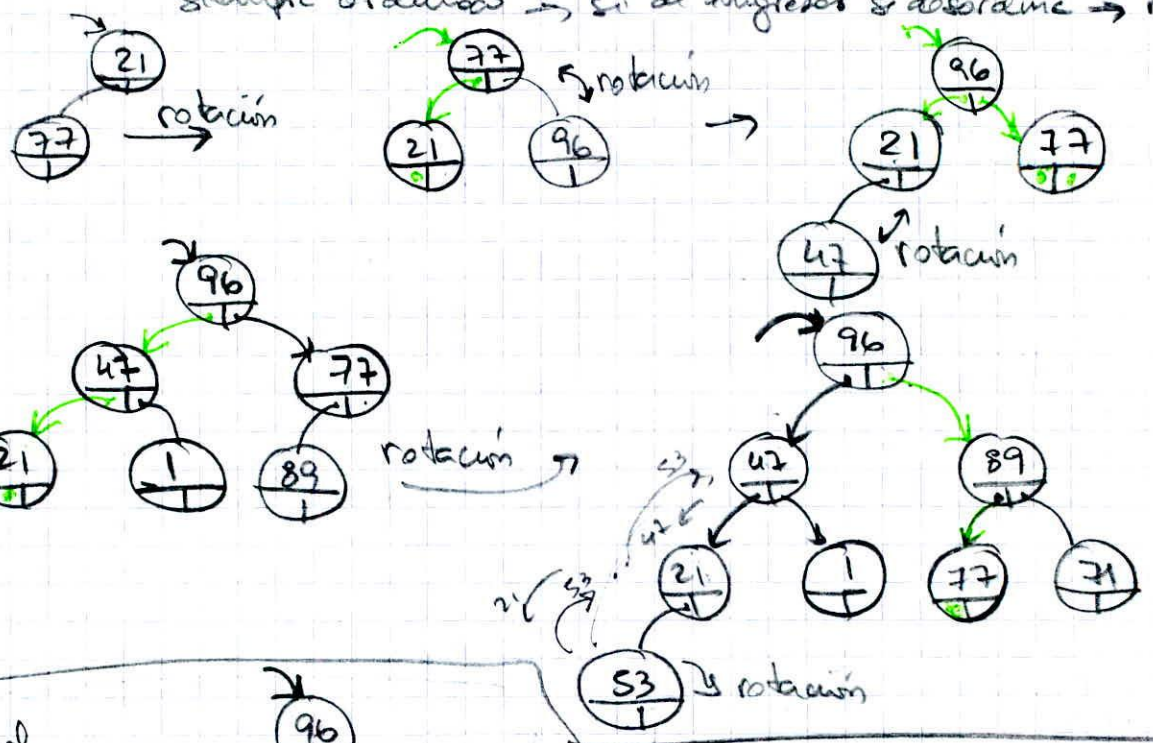
Acolar: • insertar elemento al final del heap.
• subir(elemento)

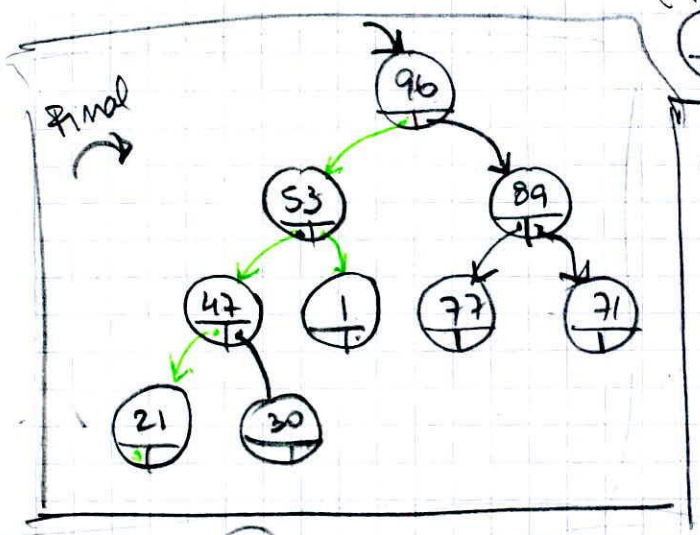
↳ subir(elemento): mientras (elem no raíz) y (clave(elem) > clave(padre(elem)))
{ intercambiar(elem, padre(elem)) }

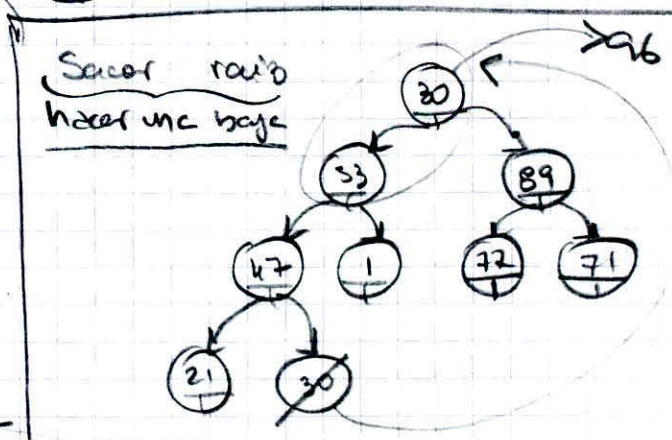
③ Definir heap. Dar el alta de heap máximo de
21 72 96 21 47 1 89 71 53 30

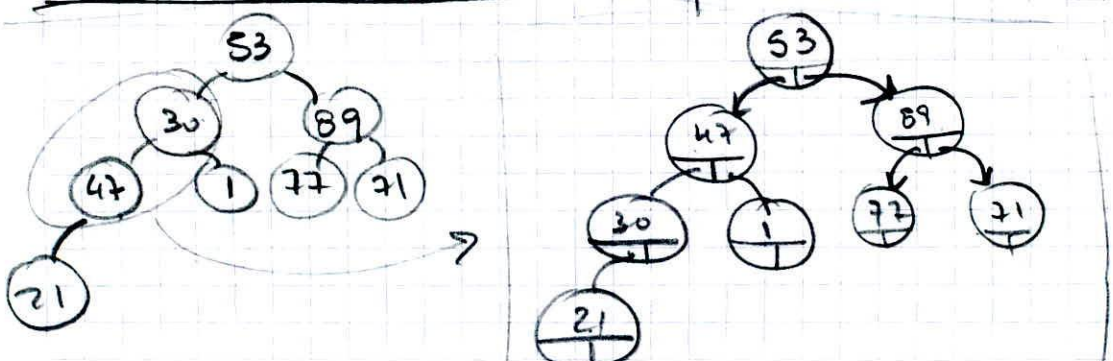
¿Cuál es el algoritmo más eficiente para el alta? ¿cuál es su costo?
¿Cuño sería una baja?

HEAP: Árbol binario completo (o casi completo) en donde la clave del nodo es siempre mayor (o menor) que las claves de sus hijos si es un heap máximo (↑ si es mínimo)
Se completan sus hojas desde la izquierda. Debe quedar siempre ordenado → si al ingresar se desordena → rotación









Alta: O(log(n)) pues depende de su altura

④ Dada la ec. $t(n) = t(n-1) + n$ hallar a qué O pertenece y mostrar cómo se llegó a tal resultado.
Definir un algoritmo que tenga ese orden

$$t(n) = t(n-1) + n$$

$$t(n-1) = t(n-2) + n-1$$

$$t(0) = 1$$

$$t(n) = t(n-2) + n-1 + n$$

$$t(n-2) = t(n-3) + n-2$$

$$t(n) = t(n-3) + n-2+n-1+n-0 = t(n-3) + 3n-3$$

$i$ésimo paso $\rightarrow t(n) = t(n-i) + in - i = t(n-i) + i(n-1)$

quiero llegar a $t(0) \rightarrow 0 = n-i \rightarrow n = i$

$\rightarrow t(n) = \underbrace{t(0)}_{1} + n(n-1) = 1 + n^2 - n \rightarrow \boxed{\in O(n^2)}$

```
void ejercicio Cuatro (int V[], int n) {   // n: largo del vector
  for (i=0, i<n, i++) {
      cout << V[i];
  }
  ejercicio Cuatro (V, n-1);
}
```

NOTA

Final 2

(5) Definir hashing. ¿Qué procedimiento utiliza para la búsqueda si se usa un método de direcc. abierto para solución de colisiones.

Es una alternativa a los algoritmos de búsqueda en donde se comparan claves. Se basa en la transformación de la clave en un número.

Si se utiliza un método de direccionamiento abierto, se debe conocer el procedimiento por el cuál se decidió ubicar en otro lugar al momento de encontrarse en una colisión

Colisión → es cuando la función que transforma la clave, genera que $h(x_1) = h(x_2)$

Redir. abierta: Se puede ubicar en el sig lugar disponible al corresp. x el hash, o se puede aplicar otra función → rehasheo. Con esto se conoce cómo se decidió ubicar esa clave y así poder buscarla