Apellidos, Nombre: ***DNI:***

Examen PED junio 2021
Modalidad 0

Normas:

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual. Este test vale 2 puntos (sobre un total de 10 de la nota de Teoría).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En C++, el puntero **this** se tiene que declarar en todos los constructores de la clase.	❑	❑	1	F

Para el siguiente fragmento de código para C++ de un posible método perteneciente a la conocida clase TCalendario, la línea "**delete [] a;**" es la instruccción más adecuada para liberar correctamente la memoria dinámica de a.
`void Funcion(void) {`
`  TCalendario *a =new TCalendario;`
`  TCalendario *b =new TCalendario [5];`
`  (. . . . . .)`
`  delete [] a;`
`}`
	❑	❑	2	F

La cota promedio de complejidad es el resultado de hacer la media entre la cota superior y la cota inferior.	❑	❑	3	F
concatenar(lista,lista), inscabeza(lista,item) y crear_lista( ) son operaciones constructoras modificadoras del tipo lista.	❑	❑	4	F
Existe un único árbol binario completo que se puede construir a partir del recorrido en postorden.	❑	❑	5	V
La complejidad temporal, en su peor caso, del recorrido por niveles en un árbol binario es la misma que las de los recorridos in-pre-post orden.	❑	❑	6	V
Un árbol binario completo es un AVL.	❑	❑	7	F
El número máximo de nodos en un árbol AVL de altura 10 sería de 1023.	❑	❑	8	V
El número mínimo de nodos que tiene un árbol AVL de altura 4 es 7.	❑	❑	9	V
Todo árbol completo es un árbol completamente equilibrado.	❑	❑	10	F
La estructura de un árbol 2-3-4 con 2^h-1 elementos, donde "h" es la altura del árbol, se corresponde con la estructura de un árbol binario de búsqueda lleno.	❑	❑	11	V
Tras la inserción en un árbol 2-3, la altura del árbol sólo aumenta cuando todos los nodos del árbol tienen dos hijos.	❑	❑	12	F
Dado un árbol 2-3, si la clave a borrar "x" está en un nodo hoja, "x" se tendría que sustituir por la clave siguiente a "x" en el recorrido en inorden del árbol.	❑	❑	13	F
La operación de borrar un elemento en un árbol 2-3-4 finaliza cuando el nodo "p" es el nodo que contiene al elemento que se desea borrar.	❑	❑	14	F
En los conjuntos representados como listas no ordenadas, la complejidad temporal de la operación "diferencia de conjuntos" es O(n), siendo n el número de elementos de cada conjunto.	❑	❑	15	F
El menor elemento de un Heap máximo siempre estará en el nivel de las hojas.	❑	❑	16	F
Al representar un grafo dirigido de N vértices y K aristas con una lista de adyacencia, la operación de hallar la adyacencia de entrada de un vértice, tiene una complejidad de $O(N^2)$.	❑	❑	17	V
La complejidad temporal del caso peor de la inserción en un Heap de altura h será de O(h).	❑	❑	18	V

Examen PED junio 2021

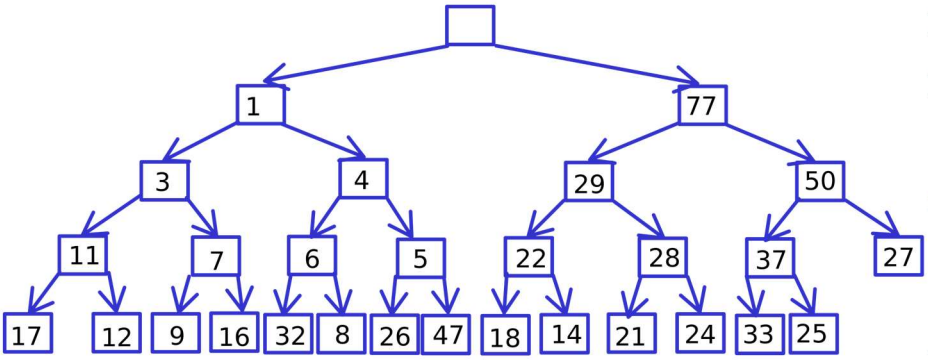
Normas:
- Tiempo para efectuar el examen: **2 horas**
- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: ***APELLIDOS, NOMBRE***.
- **Cada pregunta se escribirá en hojas diferentes**.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con **lápiz**, siempre que sea legible
- **Cada pregunta vale 1,5 puntos (sobre un total de 10 de la nota de Teoría).**
- **Las fechas de "Publicación de notas" y "Revisión del examen teórico" se publicarán en el Campus Virtual.**

1. Utilizando exclusivamente las operaciones constructoras generadoras del tipo vector, define la sintaxis y la semántica de la operación Examen que actúa sobre un vector de números naturales y elimina todas las ocurrencias de un elemento especificado excepto la primera aparición que se encuentre de dicho elemento.
Nota: se asume que está definido el TAD de los números naturales con todas las operaciones aritméticas. Es recomendable el uso de una operación auxiliar para facilitar el diseño de la recursión.

Ejemplo: sea el vector v=3,2,4,2,3,1,2 y la llamada Examen(v,**2**). La salida sería el siguiente vector v=3,**2**,4,3,1.

2. Dada la siguiente cola de prioridad doble implementada mediante la estructura de datos DEAP que se muestra a continuación, contesta las siguientes preguntas:

a) El árbol de la imagen, ¿es realmente un DEAP? Si lo es, enumera las propiedades que debe cumplir un árbol binario completo para ser un DEAP, y explica por qué el árbol de la imagen las cumple. Si no lo es, indica cuál(es) de las propiedades incumple (no es necesario que las enumeres todas; únicamente debes indicar las que incumple) y realiza sobre él el mínimo número de cambios necesarios para que sea un DEAP y contenga los mismos elementos que el original.



b) Sobre el DEAP obtenido en el apartado a), inserta el elemento 2. Sobre el resultado obtenido, inserta el 88.

c) Sobre el DEAP obtenido en el apartado a), realiza dos borrados consecutivos del elemento máximo.

d) Indica de manera justificada la complejidad temporal asintótica en el peor caso (por ejemplo, O(1), O(n), etc.), de obtener el menor elemento en las siguientes implementaciones del TAD cola de prioridad doble: DEAP, lista enlazada ordenada, lista enlazada desordenada, árbol binario de búsqueda y árbol AVL. Utiliza un ejemplo para justificar la complejidad de cada implementación.

3. a) Construye el **árbol AVL** con recorrido en inorden "20-25-30-32-35-100-140-150-160-170-180-190-199" y recorrido en preorden "100-30-20-25-35-32-160-150-140-180-170-190-199"

b) Construye el **árbol 2-3-4** con recorrido por niveles "**Nivel 1:** 75-157-197 **Nivel 2:** 30-50-95-115-137-175-215 **Nivel 3:** 10-20-40-60-85-105-125-145-165-185-205-225 **Nivel 4:** 5-15-25-35-45-55-65-80-90-100-110-120-130-140-150-160-170-180-190-200-210-220-230"

c) Sobre el **árbol 2-3-4** del apartado anterior, realiza el borrado de la clave 197, detallando las operaciones de reestructuración empleadas. Para ello, se sustituirá por el mayor del subárbol izquierdo en caso de estar en un nodo interior, y para las reestructuraciones se consultará el hermano de la izquierda.

4. Considera el **GRAFO NO DIRIGIDO**, representado por la lista de adyacencia que aparece a continuación:

a	→ f → i → h → g
b	→ c → f → e → g → h → i
c	→ f → i → g → h → e
d	→ f → h → i → c
e	→ f → a
f	→ h → i
g	→ h → i → j
h	→ i
i	→ j
k	→ m
l	→ k
n	→ k

a) Recorrido en PROFUNDIDAD. Obtener:

a.1) el recorrido DFS(a).

a.2) el árbol extendido en PROFUNDIDAD partiendo del vértice *a*.

a.3) la clasificación de las aristas para este recorrido.

b) Recorrido en ANCHURA. Obtener:

b.1) el recorrido BFS(a).

b.2) el árbol extendido en ANCHURA partiendo del vértice *a*.

b.3) la clasificación de las aristas para este recorrido.

c) ¿Es un grafo conexo? Justifícalo

NOTAS: ● Hay que tener en cuenta que la lista de adyacencia no está ordenada.

● La lista de adyacencia de cada vértice se recorre de menor a mayor alfabéticamente, para todos los casos del ejercicio (aunque las listas aparecen desordenadas).

Examen PED junio 2021. Soluciones

1.

sintaxis:

Examen: vector, natural -> vector
BorraOcurrencias: vector, natural -> vector

semántica:

var v: vector; i,x,Z:natural;

Examen(crear_vector(), Z) = crear_vector()
Examen(asig(v,i,x), Z) =
si (x == Z) entonces
asig(BorraOcurrencias(v, Z), i, x)
sino
asig(Examen(v, Z), i, x)

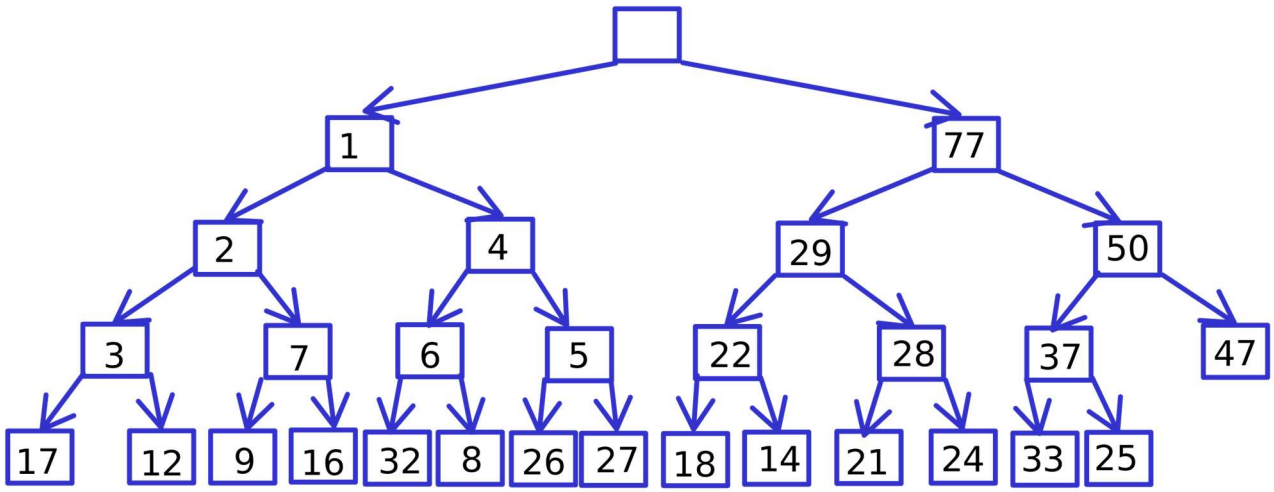
BorraOcurrencias(crear_vector(), Z) = crear_vector()
BorraOcurrencias(asig(v,i,x), Z) =
si (x == Z) entonces
BorraOcurrencias(v, Z)
sino
asig(BorraOcurrencias(v, Z), i, x)

2.

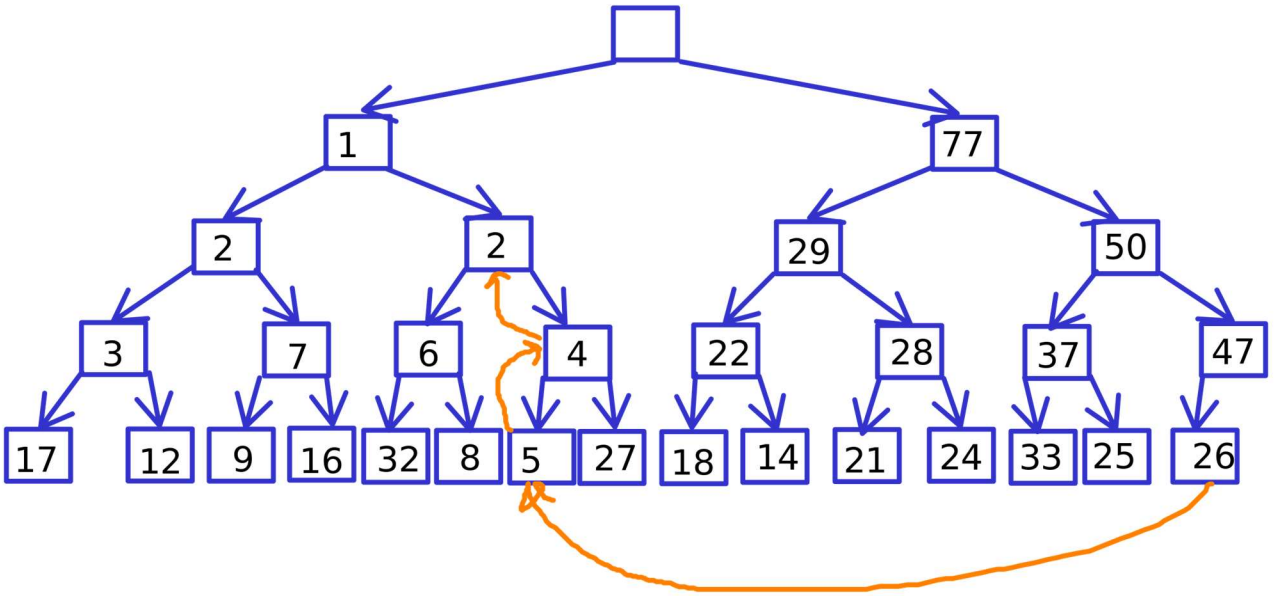
a) El árbol de la imagen no es un DEAP. Incumple la propiedad 4, que establece que, si el subárbol derecho no es vacío, para todo nodo *i* del subárbol izquierdo, su valor es menor que el de su nodo *j* correspondiente en el subárbol derecho.

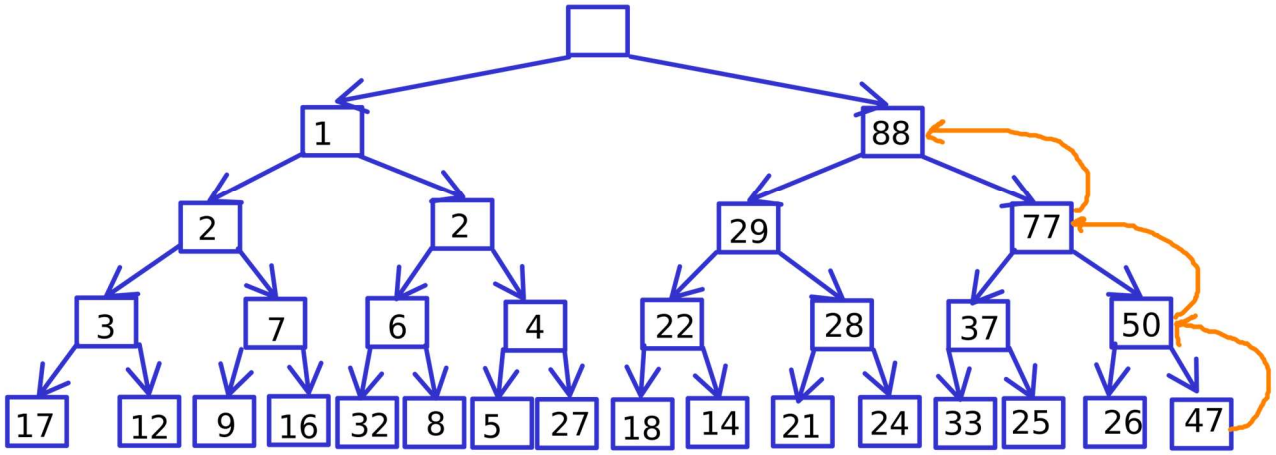
En particular, para el nodo con etiqueta 47 del subárbol izquierdo, su nodo correspondiente en el subárbol derecho contiene la etiqueta 27.

El mínimo conjunto de cambios necesarios para que el árbol sea un DEAP implica intercambiar las etiquetas 47 y 27.

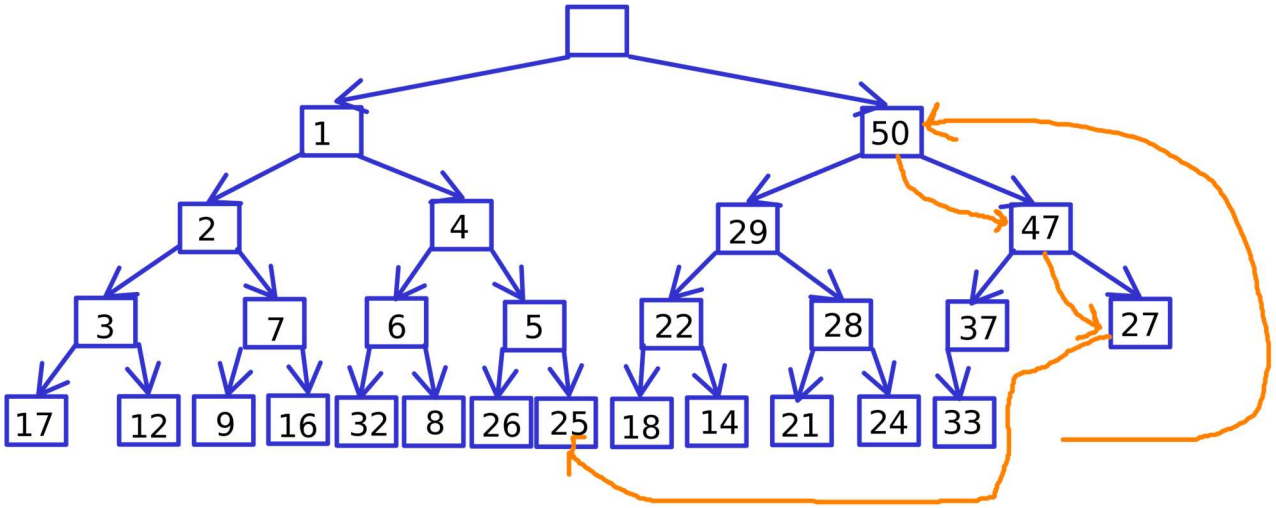


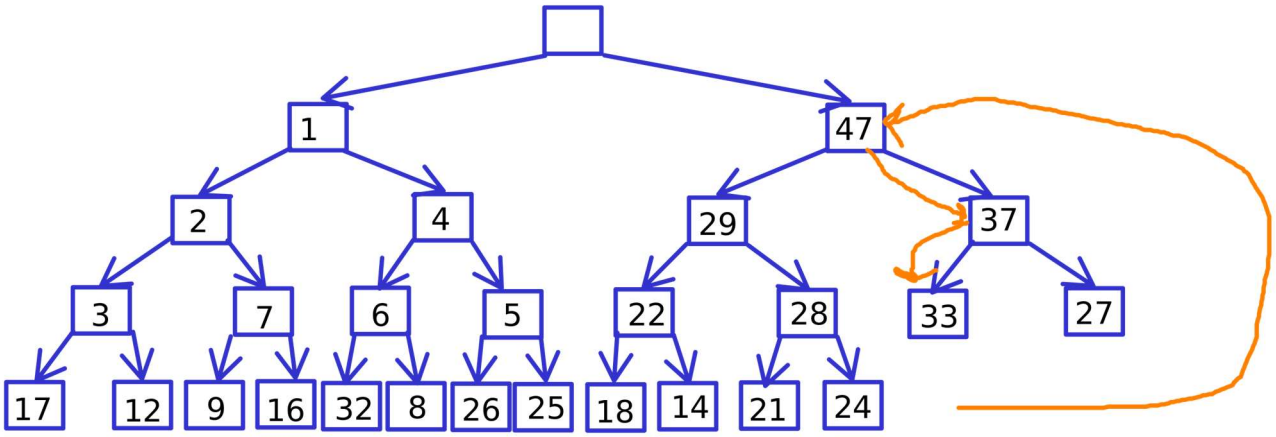
b)


1
77
2
2
29
50
3
7
6
4
22
28
37
47
17
12
9
16
32
8
5
27
18
14
21
24
33
25
26


1
88
2
2
29
77
3
7
6
4
22
28
37
50
17
12
9
16
32
8
5
27
18
14
21
24
33
25
26
47

c)


1
50
2
4
29
47
3
7
6
5
22
28
37
27
17
12
9
16
32
8
26
25
18
14
21
24
33


1
47
2
4
29
37
3
7
6
5
22
28
33
27
17
12
9
16
32
8
26
25
18
14
21
24

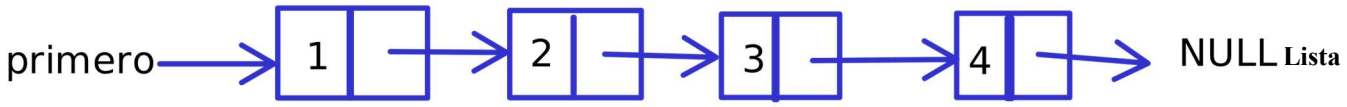
d)

DEAP: la complejidad temporal asintótica de obtener el elemento mínimo es O(1), es decir, constante. Sólo hay que consultar la raíz del subárbol izquierdo. Necesitaremos el mismo número de operaciones independientemente del número de elementos almacenados en el DEAP.

Un ejemplo podría ser el propio DEAP obtenido en el apartado a). Para acceder al menor elemento (1), sólo habría que acceder al hijo izquierdo de la raíz.

Lista enlazada ordenada: la complejidad temporal asintótica de obtener el elemento mínimo es O(1), es decir, constante. Si la lista está ordenada de menor a mayor, bastará con acceder al puntero que apunta al primer elemento de la lista. Si la lista está ordenada de mayor a menor, accederemos al puntero que apunta al último elemento de la lista. Necesitaremos el mismo número de operaciones independientemente del número de elementos almacenados en la lista.

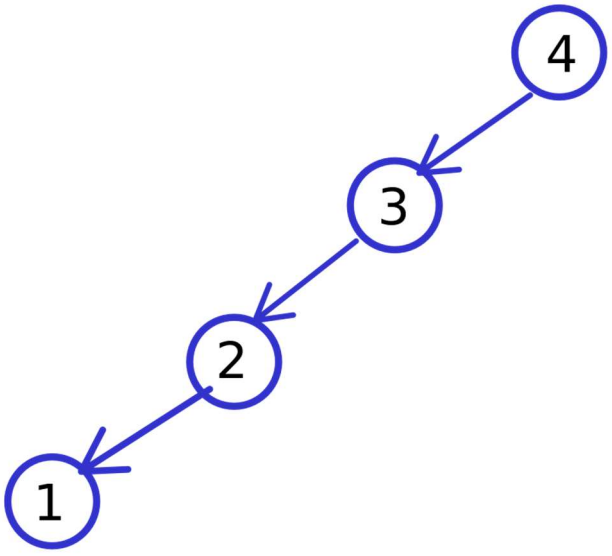
A continuación se muestra una lista de 4 elementos. La única acción necesaria para obtener el menor elemento es acceder al nodo apuntado por "primero".



enlazada desordenada: la complejidad temporal asintótica de obtener el elemento mínimo es O(n), es decir, lineal. n representa el número de elementos almacenados en la lista. Como la lista está desordenada, tendremos que recorrerla elemento a elemento y, hasta que no la hayamos recorrido entera, no podremos estar seguros de haber encontrado el menor elemento.

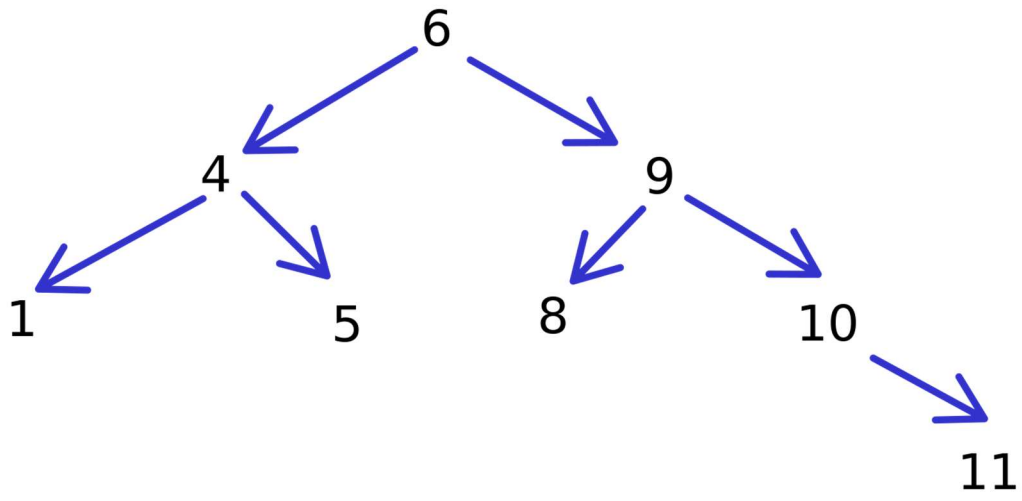
El mismo ejemplo del apartado anterior nos sirve para ilustrar la complejidad. Si no nos garantizan que la lista está ordenada, tendremos que recorrerla entera para estar seguros de que no hay ningún elemento menor que el 1.

Árbol binario de búsqueda: la complejidad temporal asintótica de obtener el elemento mínimo es O(n), es decir, lineal. n representa el número de elementos almacenados en el árbol. En el peor caso, el número de operaciones que tendremos que realizar será proporcional a la altura del árbol. Del mismo modo, en el peor caso la altura será equivalente al número de elementos almacenados en el árbol cuando el árbol está degenerado, como se muestra en el siguiente ejemplo:



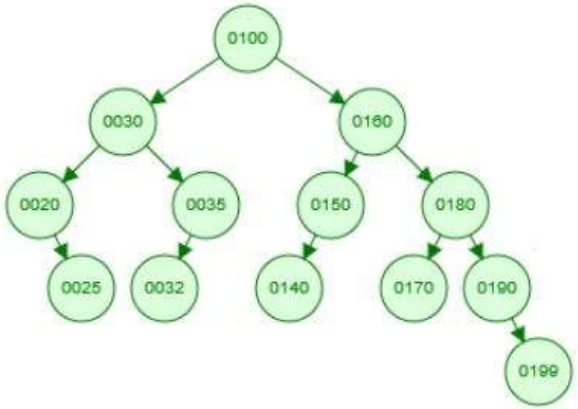
Árbol AVL: la complejidad temporal asintótica de obtener el elemento mínimo es O(log n), es decir, lineal. n representa el número de elementos almacenados en el árbol. En el peor caso, el número de operaciones que tendremos que realizar será proporcional a la altura del árbol, pero un árbol AVL nunca puede estar degenerado debido a que es un árbol equilibrado con respecto a la altura. La altura siempre será proporcional a $\log_2 n$.

El árbol AVL del siguiente ejemplo contiene 8 elementos, pero sólo es necesario acceder a 3 nodos (6, 4 y 1) para llegar al elemento mínimo.

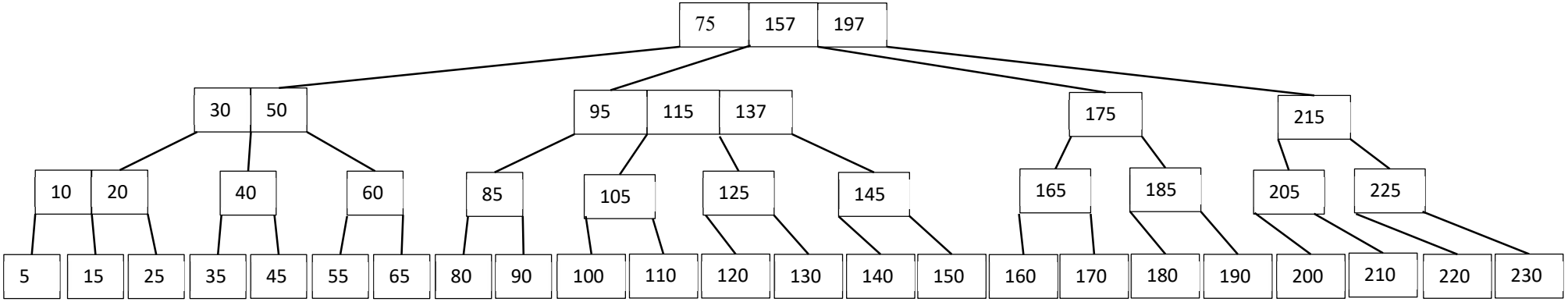

6
4
9
1
5
8
10
11

3.

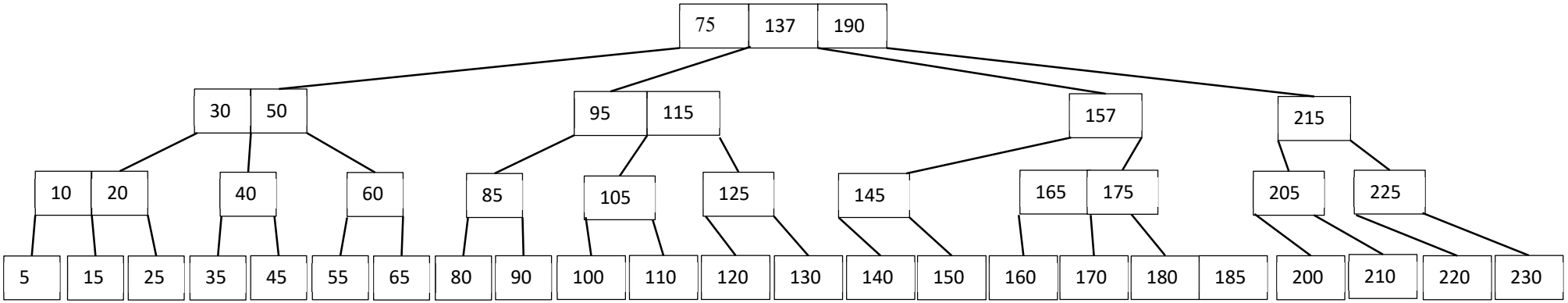
a


0100
0030
0160
0020
0035
0150
0180
0025
0032
0140
0170
0190
0199

b


75
157
197
30
50
95
115
137
175
215
10
20
40
60
85
105
125
145
165
185
205
225
5
15
25
35
45
55
65
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230

c Borrado 197. Se producen 1 rotación y 2 combinaciones:



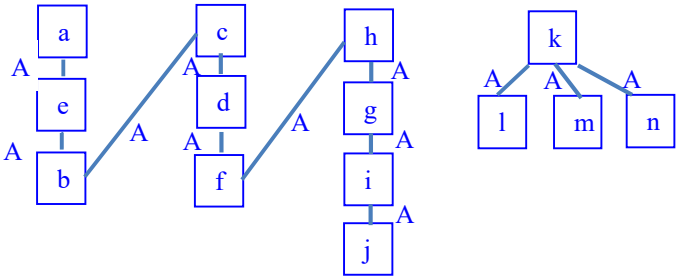
4.

a) **[0,65]** Recorrido en PRUFUNDIDAD. Obtener

a.1) [0,20] el recorrido DFS(a).

DFS(a) = a , e , b , c , d , f , h , g , i , j . Continuar por DFS(k) = k , l , m , n

a.2) [0,30] el árbol extendido en PROFUNDIDAD partiendo del vértice *a*.
a.3) [0,15] la clasificación de las aristas para este recorrido.

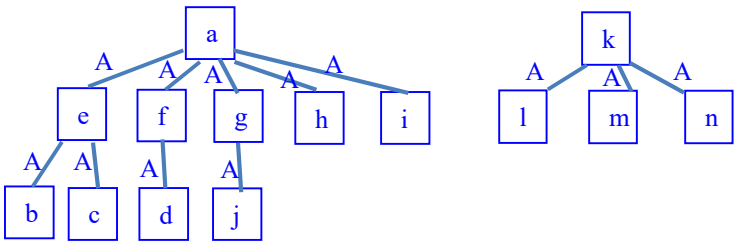


Las aristas "A" son de ÁRBOL.
El resto de aristas son de RETROCESO.

b) **[0,65]** Recorrido en ANCHURA. Obtener:
b.1) [0,20] el recorrido BFS(a).

BFS(a) = a , e , f , g , h , i , b , c , d , j . Continuar por BFS(k) = k , l , m , n

b.2) [0,30] el árbol extendido en ANCHURA partiendo del vértice *a*.
b.3) [0,15] la clasificación de las aristas para este recorrido. [0,1]



Las aristas "A" son de ÁRBOL.
El resto de aristas son de RETROCESO.

c) ¿Es un grafo conexo? Justifícalo. **[0,20]**
NO. Un grafo no dirigido se dice que es conexo: si PARA_TODO vi, vj PERTENECIENTE A V(G) existe un camino de vi a vj en G.

Este Grafo NO CUMPLE. Hay 2 COMPONENTES FUERTEMENTE CONEXAS:

C1={ **a,b,c,d,e,f,g,h,i,j** }

C2={ **k, l, m, n** }