Nombre y apellido: Jose Z...

Ejercicios (puntaje)			Nota
1 (3 pts)	2 (4 pts)	3 (3 pts)	RECUPERA
M	R	M	

CONSIDERACIONES: RESOLVER CADA EJERCICIO EN UNA HOJA DIFERENTE. LOS EJERCICIOS QUE NO ESTÉN CORRECTOS EN UN 50% NO SUMARÁN PUNTOS PARA LA NOTA FINAL. PARA CADA EJERCICIO DEBE **DEFINIR EL TIPO DE DATO** DE CADA TDA UTILIZADO. **EN TODOS LOS CASOS QUE UTILICE ESTRUCTURAS** QUE NO SEAN TDAs "ESTÁNDAR" **DEBE DEFINIR LOS STRUCTS**.

Ejercicio 1: SIN USAR TDA, desarrollar una función (o varias) para **crear un Árbol AVL a partir de un árbol binario de búsqueda**.

Encabezado de la función principal: `void convertABBToAVL(btn* raiz_ABB, btn** raiz_AVL);`

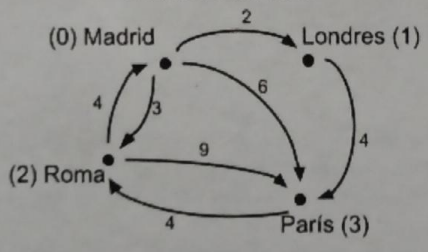
Se cuenta con la implementación de las siguientes funciones (que no deben ser desarrolladas):

```
btn* createNode(int); //Crea un nodo del árbol binario.
int height (btn*); // Devuelve la altura de un nodo de ab.
int balanceFactor (btn*); // Devuelve el factor de balanceo del nodo.
int leftRotation (btn**); // Rota a izquierda un nodo de ab.
int rightRotation (btn**); // Rota a izquierda un nodo de ab.
```

Ejercicio 2: Dado un **digrafo ponderado** (TDA *Graph*) cuyos vértices son nombres de ciudades, el nombre de una ciudad origen, el nombre de una ciudad destino y las matrices (TDA *Matrix*) devueltas por el algoritmo de Floyd de distancias y precedencias, desarrollar una función (y todas las necesarias) para crear y **devolver en una estructura que contenga un valor entero con el costo mínimo de origen a destino y una queue de strings (TDA *queue*) con la secuencia de las ciudades con el camino más corto desde el origen al destino**.

Ejemplo:

GRAFO DE ENTRADA



DISTANCIAS

	0	1	2	3
0	0	2	3	6
1	12	0	8	4
2	4	6	0	9
3	8	10	4	0

PRECEDENCIAS

	0	1	2	3
0	0	1	2	3
1	3	1	3	3
2	0	0	2	3
3	2	2	2	3

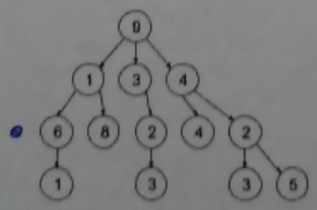
origen = "Londres" destino = "Roma"

Salida | 8 | → queue = ("Londres", "París", "Roma")

Encabezado de la función principal:

```
<definir struct>* get_path (graph* g, char* origen, char* destino,
                            matrix* distancias, matrix* precedencias);
```

Ejercicio 3: Desarrollar una función (y todas las necesarias) para que **dado un árbol n-ario devuelva un TDA *vector* con la cantidad de nodos por nivel**, donde cada índice del vector es el nivel del árbol. Ejemplos:



Árbol de entrada

0	1	2	3
1	3	5	4

TDA *Vector* de salida

Encabezado de la función principal: `vector* level_count (ntn* root);`

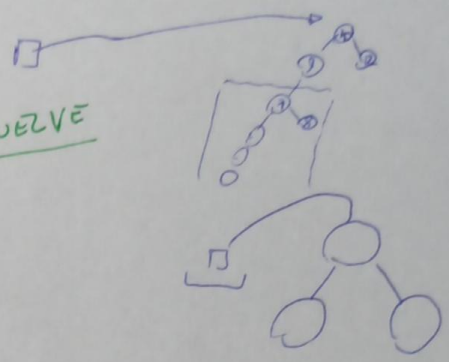
NOTA: NO debe asumir que el cada nodo contiene calculado el nivel como atributo, si se define como atributo debe calcularlo.

① void convertABBToAVL (btn* raiz_ABB, btn** raiz_AVL){

if (balanceFactor (raiz_ABB) <= 1) {
*raiz_AVL = raiz_ABB; // ESTÁ BALANCEADO
}

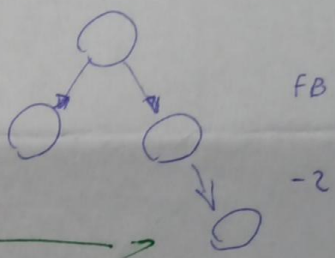
typedef struct _btn{
t_elem_btn value;
struct btn *izq;
struct btn *der;
} btn;

if (balanceFactor (raiz_ABB->der) > 0){



NO RESUELVE

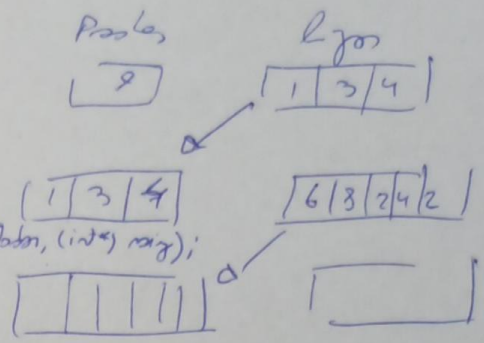
if (balanceFactor (raiz_ABB->izq) < 0){
if (height (raiz_ABB->der) > height (raiz_ABB->izq)



③

```
vector* level_count (ntn* root) {
if (root == NULL) return NULL;
int cant_hijos = 1;
queue* padres = NULL;  queue_new();
queue* hijos = NULL;   queue_new();
vector* resultado = vector_new();
int* aux = malloc (sizeof(int));  int ?
*aux = cant_hijos;
int level_contador = 0;
enqueue (padres, root); * No creado padres.
resultado[level_contador] = aux;  vector_insert(resultado, level_contador, (int*) aux);
cant_hijos = 0;
while (! queue_isempty (padres)) {
auxntn = dequeue (padres);
aux = auxntn->hijos;
cant_hijos = vector_size (aux);
for(int i=0; i<cant_hijos; i++){
enqueue (hijos, (ntn*) vector_get (aux, i));
}
if( queue_isempty (padres)){
level_contador = level_contador+1;
padres = hijos;
resultado[level_contador] = cant_hijos;
cant_hijos = 0;
level_contador = level_contador++;  int* aux = malloc(sizeof(int));
*aux = cant_hijos;
resultado[level_contador] = cant_hijos;  vector_insert(resultado, level_contador, (int*)cant_hijos);
cant_hijos = 0
}
}
queue_free (padres);
queue_free (hijos);
return resultado;
}
```

```
#define t_queue_elem ntn*
#define t_vector_elem void*
```



- NO RESUELVE EL PROBLEMA
- VARIOS ERRORES BASICOS
- FALTA ORDEN EN LOS CICLOS (IDENTACION)

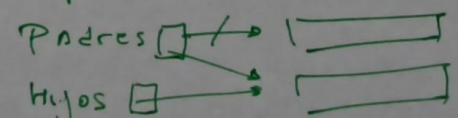
// ME QUEDÉ SIN PADRES => LEVEL PASADO

④ Padres = hijos; ← Asigno punteros? Pierdo lo queque original de padres

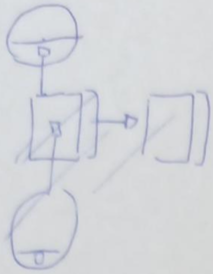
// PARA GUARDAR

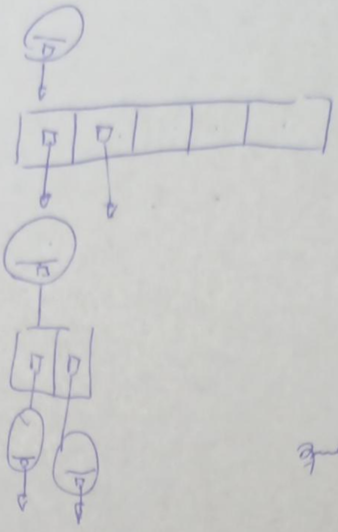
// SUPONGO QUE VECTOR DEVUELVE "0" COMO LONGITUD DE UN NULL

④ Luego del primer ciclo del While "padres" y "hijos" quedan apuntando a lo mismo queue.



③

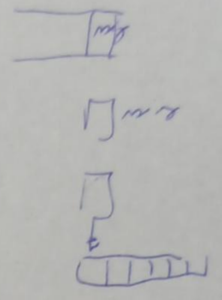




cantidad hijos = longitud vector

cola ptros vector

queue

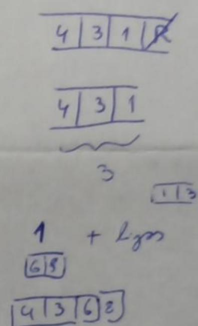


③

```
#define t_elem_ntn int

typedef struct ntn {
t_elem_ntn value;
vector* hijos;
} ntn;
```

```
vector* level_count (ntn* root) {
  if (root == NULL) return NULL;
vector* resultado = vector_new();
int cont_hijos=0;
vector* aux = NULL;   (~~root->hijos~~)
ntn * auxntn = NULL;
queue* cola = queue_new();
enqueue (cola, root);
int level_contador=0;
vector* hijosdehijos=NULL;
while (! queue_isempty (cola)) {
auxntn = dequeue (cola);
  aux = auxntn->hijos;
  cont_hijos = vector_size(aux);
int* crestes = malloc (sizeof(int));
  *crestes = cont_hijos;
    for (int i=0; i< cont_hijos ; i++){
      enqueue (cola, (ntn*) vector_get(aux,i));
          auxntn = vector_get(aux,i);
      if (auxntn->hijos != NULL) {
        hijosdehijos = aux->hijos
        cont_hijos = cont_hijos + vector_size(hijosdehijos);
      }
~~auxntn = dequeue(cola);~~
~~enqueue(cola.hijos, (ntn*) auxntn->~~
    }
*crestes = cont_hijos;
resultado[level_contador] = cont_hijos;
```



1 + hijos

// ver los hijos de sus hijos = hijos nivel