Nombre y apellido: ALVAREZ Julián Manuel ENTREGO 3 HOJAS

Ejercicios (puntaje)			Nota
1 (3 pts)	2 (4 pts)	3 (3 pts)	4 – (CUATRO menos)
M	MB	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);`

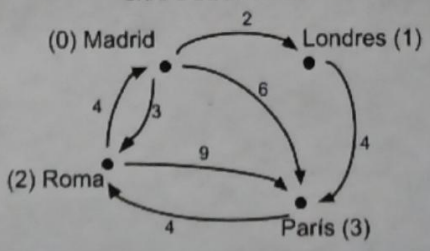
Se cuenta con la implementación de las siguientes funciones (que no deben ser desarrolladas):

```
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

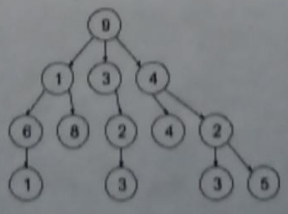
origen = "Londres" destino = "Roma"

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);`

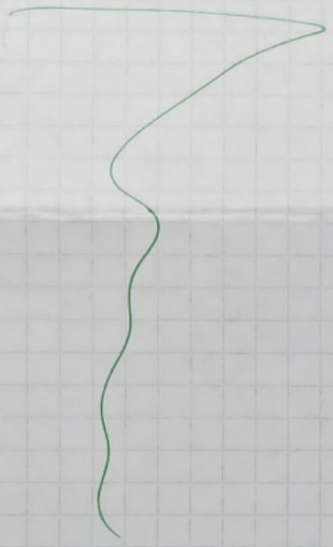
NOTA: NO debe asumir que el cada nodo contiene calculado el nivel como atributo, si se define como atributo **debe calcularlo**.

Alvarez Julian
14173

```
#define t_elem_btm int
```

(1)
```
Typedef struct _btm {
    t_elem_btm value;
    struct _btm * left;
    struct _btm * right;
} btm;
```

```
Void ConvertABBtoAVL (btm* raiz_abb, btm** raiz_avl)
{ if (raiz_abb) return;
  Copy_btm_in_AVL (raiz_abb, &raiz_avl);
  While (|balanceFactor (*raiz_avl)| < 2)
```



NO RESOLVE

```
Void Copy_btm_in_AVL (btm* root_btm, btm** AVL)
{ If (root_btm == Null) return
  *(*AVL) -> left = Copy_btm_in_AVL (root->left, &AVL);
  *AVL = CreateNode (root->value);
  *(*AVL) -> right = Copy_btm_in_AVL (root->right, &AVL);
}
```

(*AVL) == NULL

define T_list_elem int
define t_vector_elem int
define T_element_mtm int

Alvarez Julian 14173
23/06/23

③
```
typedef struct _mtm{
    t_element_mtm value;
    t_list* sons;
} MTM;
```

```
typedef struct _tlist{
    _mtm* node;
    struct _t_list* next;
} t_list;
```

```
Vector* level_count(MTM* root)
{
    if(!mtm) return;
Vector* vec_result = Vector_New();
    Vector_add(vec_result, 1);
t_list* L = (MTM*) root -> node;
While(L != NULL)
{
    Vector_add(vec_result, list_length(L->sons));
    L = L->next;
}
return(vec_result);
}
```

- NO RESUELVE
Agrega LOS Hijos DE ~~LA~~ RAIZ ~~NODO~~, NO cuenta ~~NIVEL~~ NODOS x NIVEL