UNMDP - Facultad de Ingeniería **Recuperatorio 2do Parcial 2022** Algoritmos y Estructura de Datos

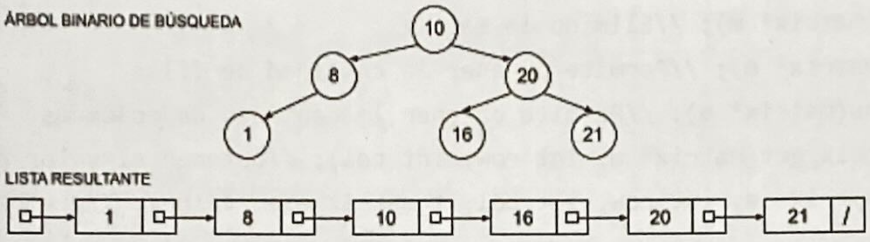
Nombre y apellido: Nazareno Montesoro

Ejercicios (puntaje)			Nota
1 (4 pts)	2 (4 pts)	3 (2 pts)	
B-	B	M-	6 (SEIS) -

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.**

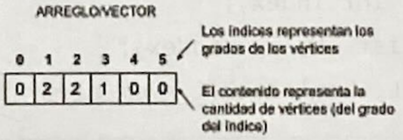
Ejercicio 1: **SIN USAR TDA**, desarrollar una función (y todas las necesarias) para **crear**, del modo más eficiente, **una lista dinámica** simplemente enlazada (SLL) con el valor (entero) **de todos los nodos un árbol binario de búsqueda** (SBT), teniendo en cuenta que la lista debe quedar **ordenada en forma ascendente**.
Ejemplo:



Encabezado de la función: `void crea_lista_de_sbt (sll** head, btn* root);`

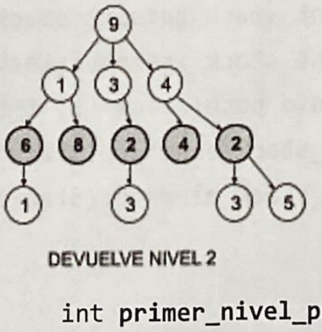
Ejercicio 2: Dado un **digrafo** (con implementación a elección, que puede o no utilizar TDAs), desarrollar una función (y todas las necesarias) para **completar un arreglo** (o vector) que contenga **la cantidad de vértices agrupados por grado**, que son **alcanzados por un vértice origen**. Es decir que el índice del arreglo/vector indicará el grado de los vértices, y el contenido la cantidad de vértices, siempre que los vértices puedan ser alcanzados directa o indirectamente por un vértice de origen.

Ejemplo: en el siguiente grafo, con origen en "*a*", se completa en el arreglo/vector en el índice 1: 2 por los nodos "*a*" y "*e*" de grado 1, en el índice 2: 2 por los vértices "*b*" y "*d*" de grado 2 y en el índice 3: 1 por el vértice "*c*" de grado 3 y "*f*" si bien es de grado 2 **NO SUMA** en el índice 2 porque **no es alcanzado** por "*a*".

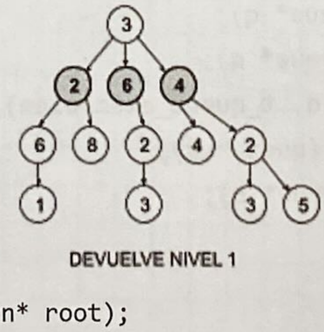


Encabezado de la función: `void vertices_por_grado (graph* g, int origen, <t_struct>* a);`

Ejercicio 3: Desarrollar una función (y todas las necesarias) para que **dado un árbol n-ario devuelva el primer nivel en el cual todos los nodos son números pares**, o -1 si no se encuentra. Ejemplos:



DEVUELVE NIVEL 2



DEVUELVE NIVEL 1

Encabezado de la función: `int primer_nivel_par (ntn* root);`

HOJA N°

① Nazareno Monteroro

FECHA

```
#include <stdlib.h>

typedef struct sll {
  int value;
  struct sll *next;
} sll;

typedef struct btn {
  int value;
  struct btn *left;
  struct btn *right;
} btn;

void add-node (sll **head, int value) {

  if (head [illegible]) {

    sll *aux = *head;

    if (aux == NULL) { [illegible]

      aux = calloc (1, sizeof(sll));     } NO INSERTA
      aux -> value = value;
    }
    else
    {
      while (aux->next) { aux = aux->next; }    } INEFICIENTE, RECORRE
      aux->next = calloc(1, sizeof (sll));         TODA LA LISTA EN
      aux->next->value = value;                    CADA LLAMADO
    } FALTA ASIGNAR AUX→NEXT → NEXT
  }
}

void crea-lista-de-sbt (sll **head, btn *root) {
  if (head [illegible] && root) {

    crea-lista-de-sbt (head, root->left);
    add-node (head, root->value);
    crea-lista-de-sbt (head, root->right);
  }
}
```

NOTA

HOJA N°

② Nazareno Montesoro

FECHA

```
#include <stdbool.h>
#define t-matrix-elem  int

typedef struct graph {
                                   → Vértices?
   matrix *adj;                ✓
   int count;
} graph;

int degree (graph *g, int index) {

   int d = 0;

   if (g && index >= 0 && index < g->count) {
      for (int j = 0; j < g->count; j++) {                    ✱ ciclo 1
         if (matrix-get (g->adj, index, j)) { d++; }       ✓
      }
   }
   return d;
}

void helper (graph *g, int origen, int **a, [illegible] bool **visited) {
   int d = 0;
   if (!visited [origen]) {    ✓

      visited [origen] = true;      ✓
      d = degree (g, origen);        ✓

? ✱  a [d] += 1;

      for (int j = 0; j < g->count; j++) {        ✱ ciclo 2 → Recorre 2
                                                          veces cada
         if (matrix-get (g->adj, origen, j)) {  ✓        vertice.-
            helper (g, origen, a, visited);
         }
      }
   }
   return d;                                            No contiene el
}                                                          tamaño

Void vertices-por-grado graph *g, int origen, void **a) {
                                                 → porque no int?
   if (g && a) {                                 ✓
      *a = calloc (g->count, sizeof (int));           ✓
      bool ** visited = calloc (g->count, sizeof (bool));
      helper (g, origen, (int **) *a, visited);       ✓
      free (visited)                   ?
   }
}
```

NOTA