



TOTALIZADOR PROGRAMACION II

7.8.2023

Nombre Apellido: Ricardo Gigliotti Calificación:

Serán considerados al calificar este examen la legibilidad, eficiencia y modularización de las soluciones y la utilización adecuada de las características del lenguaje C y de la programación estructurada.

Para aprobar es necesario obtener al menos 5 puntos. Y al menos el 25% de cada uno de los ejercicios 2, 3 y 4. Y al menos 4,25 entre los ejercicios 2, 3 y 4.

Cuando este examen está aprobado, la nota FINAL se obtiene así: CURSADA * 0.3 + TOTALIZADOR * 0.7

En todos los ejercicios que corresponda, mostrar las invocaciones (incluyendo su contexto: declaraciones de variables y tipos, inicializaciones y acciones posteriores) de las soluciones desarrolladas.

Ej 1 (1,5 p)	Ej 2 (4 p)	Ej 3 (2,5 p)	Ej 4 (2 p)	NOTA	CURSADA	FINAL (*)
0,75	3,5	1,25	0,75	6,75	6,75	7

Ej 1.- Indicar V o F, justificando o exemplificando adecuadamente (de lo contrario tendrá puntaje cero)
 a) Un árbol AVL es un árbol binario en el cual se verifica que la longitud de todas sus ramas es la misma o difieren en, a la sumo, 1.

b) No podría implementarse una pila en memoria estática iniciando el tope en un valor coincidente con el tamaño del arreglo.

Ej 2.- (Utilizar TDA Pila) Se tiene una lista L doblemente enlazada [ordenada de forma ascendente], que contiene en cada nodo una cadena de caracteres. Se tiene además, una Pila P que contiene en cada elemento dos datos: un entero E (es 0) y un carácter C.

Se pide:

a) Utilizando el TDA Pila, desarrollar en C subprogramas para:

i) Actualizar P de forma tal que, en cada elemento de P, E contenga la cantidad de elementos de la lista que tienen a C como inicial. El recorrido sobre P debe ser recursivo.

ii) Eliminar de L los nodos que contengan cadenas cuya inicial no esté en P almacenando en un archivo binario RESUMEN.DAT de structs la siguiente información para las cadenas eliminadas:
Cadena, CantVocales, CantConsonantes

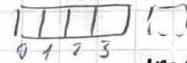
b) Suponer que P es estática. Definir los tipos asociados y desarrollar *SacaP()* y *PoneP()*. Indicar en que archivo(s) están las definiciones y funciones.

Ej 3.- Se tiene una lista de adyacencia que representa un digrafo de N vértices con aristas ponderadas, y un ABB con datos reales. Se pide obtener la cantidad de vértices del digrafo que cumplen que: no tienen bucle y el mínimo costo de las aristas de salida no está en el ABB.

Ej 4.- (Utilizar TDA N.Ario) Dado un árbol N-ario de enteros, verificar mediante una solución no void que haya exactamente un nodo de grado K (K es dato de entrada).

M

- 1) A) Verdadero. Si la longitud entre ranas varía, los F.e. serían > 2 , porque no son un AVL. Es balanceando justamente porque sus ranas están 'balanceadas' la longitud.
- B) Verdadero. En C, los arrays comienzan en 0. Es decir que, si el máx de logros es 50, podemos marcarlos del 0 al 49. Si inicializamos el topo en un valor coincidente con el tamaño del arreglo, estaremos ingresando en este caso en `vec[50]`, lo cual NO existe, estaremos accediendo a BASURA!

`vec[4] = BASURA!`

especie num NO puedo acceder,

ACARACÍN

↳ en TODOS los mains, irá

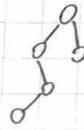
```
[ #include <stdlib.h>
  #include <stdio.h> ] :D :O → me olvidé de
                                     apoyarlo en algunos!
```

Al poner declaraciones
topo
e incrementarlas al sacar!

A) Gj =



BALANCEADO

DESBALANCEADO
(difiere en 2)!

BALANCEADO

EJ ② ③ ④

2) LISTA DOBLE

* cadena

↳ ORD ASC d.

PILA

* ultimo E

* cabeza C

typedef struct nodoD {

char cad[100];

struct nodoD *ant, *sig;

} nodoD;

typedef struct {

nodoD *pri, *ult;

} TListaD;

A) P

i) Recorrido RECURSIVO

CADA elemento E → cuenta la lista iniciando C.

int contarCar(TListaD LD, char caracter) { // recorre la LD para ver cuantos elem arrancan con 'C'

 nodoD aux;

 int count=0;

 while (aux != NULL) && (aux->cad[0] <= caracter) {

 if (aux->cad[0] == caracter)

 count++;

 aux = aux->sig;

 }

 return count;

}

void AlterarP(TPilaP, TListaD LD)

{ ElementoP Elemp;

if (!Vacia(&P))

 sacaP(&P, &Elemp);

 alteraP(&P, Elemp, LD);

 Elemp.E = contarCar(LD, Elemp.C);

 ponerP(&P, Elemp);

}

 #include "pilas.h" #include "stdlib.h" #include "stdio.h"

int main (void) { // TIPO PILA DESARROLLADO EN DIB

 TPila P;

 TListaD LD;

 iniciarP(&P);

 cargaLD(&LD);

 alteraP(&P, LD);

}

EJ

(2) (A)(ii) (MAIN) (1/3)

define MAX 25

```
#include < stdlib.h >
#include < stdio.h >
#include < string.h >
#include < pilas.h >

int main (void) {
    TlistaD LD;
    TPila P;
    int vec [25] = {0};
    FILE * arch;
```

iniciaP(&P); ✓

cargaDatosP(&P); /

cargaDatosLD(&LD); /

Abre Archivo

```
arch = fopen ('RESUMEN.DAT', 'wb');
if arch == NULL
```

printf ('Problema al abrir el archivo');

{/2}

inicializaVecIniciales(vec, &P);

elimSuc(&LD, vec, arch);

{/}

{ Felizos(arch);

```
typedef struct nodoD{
    char cad [MAX];
    struct nodoD * ant * sig;
} nodoD;
```

```
typedef struct {
    nodoD * pri, ult;
} TlistaD;
```

EJ ② ① ② 2/3 ④ ⑤ (Funciones Aux) ~~Muy~~

ii)

```
void inicializaVecInicial86(int vec[], TFla *P){ // PIDERON QUE NO SE PIERDA LA PILA, RECORRO RECURSIVAMENTE;
    ElementoSP ElemenP;
    if (!raizP(&P)) {
        sacaP(P, &ElemenP);
        if (ElemenP[0] - 'A1' == 1) { // comillas no
            inicializaVecInicial86(vec, P);
            poneP(P, ElemenP);
        }
    }
}
```

```
int esVocal (char letra){ // minúsculas
    return (letra == 'a') || (letra == 'e') || (letra == 'i') || (letra == 'o') || (letra == 'u'); // minúsculas
}
```

```
void analizoPalabra (char palabra[], int *vocales, int *consonantes)
{
    int i;
    for (i=0; i<strlen(palabra); i++) {
        if (esVocal(palabra[i])) // 
            (*vocales)++;
        else // 
            (*consonantes)++;
    }
}
```

(C) (2)(3) (PILA ESTÁTICA, poneP y sacaP)

2(b) #define MAX 50

```
typedef struct {  
    int E;  
    char C;  
} TElementoP;
```

ESTE
CÓDIGO!

```
typedef struct {  
    TElementoP datos[MAX];  
    int tope;  
} TPila;
```

```
void sacaP(TPila *P, TElementoP *x){  
    if ((*P).tope != -1)  
        *x = (*P).datos[(*P).tope--];  
}
```

```
void poneP(TPila *P, TElementoP x){  
    if ((*P).tope == MAX - 1)  
        (*P).datos[(*P).tope + 1] = x;  
}
```

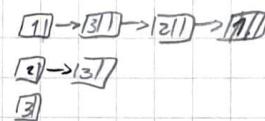
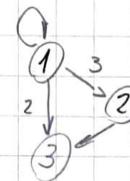
Definiciones
de tipo y
funciones

'Pilas.h'

desarrollo
de funciones
TOA

en 'main.c' ASOCIADA 'pilas.h'

EJ (3)

3) Lista de ADT \rightarrow DIGRAFO N vértices !,
Aristas ponderadasABB \rightarrow realesCANT. de vértices que cumplen: A) No bude
B) Min costo Ar. salidas ABB

```

int estaEnABB(arbol A, float buscado){
    if A == NULL
        return 0;
    else
        if (A->dato == buscado)
            return 1;
        else
            if (A->dato > buscado)
                return estaEnABB(A->iizq, buscado);
            else
                return estaEnABB(A->der, buscado);
}

```

```

int cantCompleta (Tlista L[], int N, arbol A) {
    Tlista aux;
    int i; no tiene Bucle, cont = 0;
    float min;
    for (i=0; i<N; i++) {
        aux = L[i];
        min = 999; no tiene Bucle = 1;
        while (aux != NULL) && (no tiene Bucle) {
            if aux->vertice == i + 1
                no tiene Bucle = 0;
            else
                if aux->costo < min
                    min = aux->costo;
        }
        if no tiene Bucle && !(estaEnABB(A, min))
            cont++;
    }
    return cont;
}

```

#define MAX 50

int main (void) {

int N;

Tlista L[MAX];

arbol A;

CargaL (&L, &N);

CargaArbol (&A);

printf ('La cant de vértices que cumplen con lo pedido es %d', cantCompleta (L, N, A))

No avanza!

Falta verificar que el min sea distinto de 999

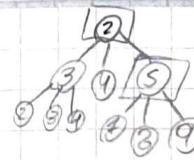
```

typedef struct nodo {
    float dato;
    struct nodo *izq;
    struct nodo *der;
} nodo;
typedef nodo * arbol;

```

EJ (4)

4) TDA N-Ario

EXACTAMENTE UN nodo
de grado K

(2) ! degr 3cl,

```

int grNodo (arbol A, posicion p) {
    int gr = 0; posicion aux;
    if Nulo (p)
        return 0;
    else {
        aux = HijoMasIzq (p, A);
        while !Nulo (aux) {
            gr++;
            aux = HnoDer (aux, A);
        }
        return gr;
    }
}
  
```

posible contar si $\rightarrow K$

NOTA = la solución mediante una función `int` es inefficiente = recorre TODO el árbol y cuenta todos los nodos de grado K. PERO (1) si pudiera hacer una función void llamaría un parámetro 'cant' por referencia y pasaría el cuento en segundo nodo con grado K.

```
int cantGradoK (arbol A, posicion p, int K) {
```

C: SIMPLEMENTE
Cuenta todos
los nodos
con grado
K!

```

if Nulo (p)
    return 0;
else
    return cantGradoK (A, HijoMasIzq (p, A), K) + cantGradoK (A, HnoDer (p, A), K) + (grNodo (A, p) == K);
}
  
```

```
#include <stdlib.h>
#include <stdio.h>
#include "TDAARIO.h"
```

```
int main (void) {
    arbol A;
    int K;
    cargoArbol (&A);
    printf ("Ingresé grado K");
    scanf ("%d", &K); // ASUMO DATO CORRECTO
}
```

```
if cantGradoK (A, Raiz (A), K) == 1
    printf ("Hay exactamente un nodo de grado K");
else
```

```
    printf ("Hay más de un nodo de grado K");
}
```

* Recorre los nodos multiples veces

* No termina si la cantidad es
mayor a 1.