



TOTALIZADOR PROGRAMACION II

7.8.2023

Nombre Apellido: Rosio 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	6,7	7

Ej 1.- Indicar V o F, justificando o ejemplificando 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).

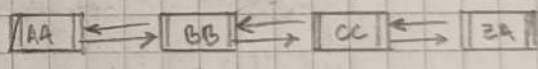
Agosto 2023

1 - V o F

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 lo sumo 1.
FALSO. Un árbol AVL es aquel en el que se verifica tanto para el árbol como para todos sus subárboles que la longitud entre su rama izquierda y derecha es la misma o difieren, a lo sumo, en 1.

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

FALSO. Si podría implementarse si el tope representa un único elemento y la pila se llena cuando el tope se encuentre en la posición 0 en el arreglo.

2- Lista doble de cadenas (ordenada) 

Pila de registros $\begin{cases} \text{Entero (0)} \\ \text{Character} \end{cases}$

a) Subprogramas Para:

i) Actualizar P de forma tal que en cada elemento de P, E contenga la cantidad de elementos de la lista que tengan a C como inicial

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

b) Suponer que P es estática. Definir los tipos asociados y desarrollar Sacar y Poner.

TDAPilaest.h

```
#define MAX 100
```

```
typedef struct {
```

```
    int E;
```

```
    char c; } TElementoP;
```

```
typedef struct {
```

```
    TElementoP datos[MAX];
```

```
    int tope; } TPila;
```

```
void PonP(TPila *, TElementoP);
```

```
void SacP(TPila *, TElementoP *);
```

TDAPilaest.c

```
#include <stdio.h>
```

```
#include "TDAPilaest.h"
```

```
void PonP(TPila *P, TElementoP x) {
```

```
    if ((*P).tope != MAX - 1)
```

```
        (*P).datos[++(*P).tope] = x;
```

```
}
```

```
void SacP(TPila *P, TElementoP *x) {
```

```
    if ((*P).tope != -1)
```

```
        *x = (*P).datos[(*P).tope--];
```

```
}
```

main.c

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

```
#define LONG 50; #define LET 27
```

```
typedef struct nodoD {
```

```
    char cad[L];
```

```
    struct nodoD *sig, *ant; } nodoD;
```

```
typedef nodoD *PnodoD;
```

```
typedef struct {
```

```
    PnodoD pri, ult; } TListaD;
```

```
typedef struct {
```

```
    char cad[L];
```

```
    int cantVoc, cantCons; } Fbin;
```

i) void Actualiza (TPila *P, int VF[]) {

	A	B	C				Y	Z
VF:	5	7	0	0	2

TElementoP r;
 if (!VacíaP(*P)) {
 Sacar(P, &r);
 Actualiza(P, VF);
 r.E = VF[r.C - 'A'];
 Poner(P, r);
 }
}

void CuentaYActualiza (TPila *P, TListaD LD) {

int VF[LET] = {0};
 PnodoD auxD;
 auxD = LD.pri;
 while (auxD != NULL) {
 VF[(auxD->Cad)[0] - 'A']++;
 auxD = auxD->sig;
 }
 Actualiza(P, VF);
}

ii) void EliminaDel (TPila *P, TListaD *LD) {

TPila Paux; TElementoP r; rbin rcad;
 FILE *ArchB;
 PnodoD auxD;
 if ((ArchB = fopen("RESUMEN.DAT")) == NULL)
 printf("No pudo abrirse el archivo");

else {
 while (!VacíaP(*P)) {
 Sacar(P, &r);
 auxD = (*LD).pri;
 while (auxD != NULL && r.C < (auxD->Cad)[0])
 auxD = auxD->sig;
 while (auxD != NULL && r.C == (auxD->Cad)[0])
 info(r.Cad, auxD->Cad);
 EliminaCad(LD, auxD);
 fwrite(r.Cad, sizeof(rbin), 1, ArchB);
 auxD = auxD->sig;
 }
 Poner(&Paux, r);
 }
 while (!VacíaP(Paux)) {
 Sacar(&Paux, &r);
 Poner(P, r);
 }
 fclose(ArchB);
}

3- Lista de adyacencia de digrafo de N vertices con aristas pond, y un ABB con datos reales. Obtener la cantidad de vertices del digrafo que cumplen que: no tienen bucle y el mínimo costo de las aristas de salida no esta en el ABB

```

int CantCumplen (TLista VL[], int CantV, arbol A) {
    TLista LAd;
    int Cont = 0, i, Bucle, MinGS;
    arbol aux;
    for (i = 0; i < CantV; i++) {
        LAd = VL[i];
        Bucle = 0;
        MinGS = 999;
        while (LAd != NULL && !Bucle) {
            if (LAd->Vertice == i+1)
                Bucle = 1;
            if (LAd->Costo < MinGS)
                MinGS = LAd->Costo;
            LAd = LAd->sig;
        }
        Cont += !Bucle && !(Esta(A, MinGS)) && MinGS != 999;
    }
    return Cont;
}

Esta (arbol A, int Buscado) {
    if (A != NULL)
        if (A->dato == Buscado)
            return 1;
        else
            if (A->dato < Buscado)
                return Esta(A->der, Buscado);
            else
                return Esta(A->izq, Buscado);
    else
        return 0;
}

```

4- Dado un árbol N-ario de enteros, verificar (no void) que haya exactamente un nodo de grado k

```
int Verifica(ArbolN A, Posicion P, int k) {
    Posicion c;
    int grado;
    if (!Nulo(P)) {
        c = HijoMasIzq(P, A);
        grado = 0;
        while (!nulo(c) && grado <= k) {
            grado++;
            c = HnoDer(c, A);
        }
        if (grado == k && nulo(c))
            return !Verifica(A, HijoMasIzq(P, A), k) && !Verifica(A, HnoDer(P, A), k);
        else
            if (Verifica(A, HijoMasIzq(P, A), k))
                return !Verifica(A, HnoDer(P, A), k);
            else
                return Verifica(A, HnoDer(P, A), k);
    }
    else
        return 0;
}
```