TOTALIZADOR PROGRAMACION II
20.12.2021

Apellido, Nombre: [redacted]

Serán considerados al calificar este examen la eficiencia 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, inicializaciones y acciones posteriores) de las soluciones desarrolladas.

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

Ej 1.- Indicar Verdadero o Falso, justificando o ejemplificando adecuadamente (de lo contrario tendrá puntaje cero)

a) En la matriz final tras ejecutar Floyd (A_N) todos los elementos de la diagonal principal son 0, a menos que haya bucle.

b) Una cola implementada dinámicamente de modo circular, permite aprovechar aquellos nodos que ya no tienen datos significativos.

Ej 2.- La Municipalidad gestiona las infracciones realizadas por cada cámara de seguridad y radar mediante una lista doblemente enlazada en la que cada nodo contiene:

- Zona (1 a 10, se puede repetir)
- Cod de Cámara/Radar (cadena de 4 : si comienza con C es cámara y si comienza con R es Radar)
- Pila de infracciones (podría estar vacía), donde cada elemento contiene:
  - FechaHora (cadena de 10, formato MMDD-HH:MM)
  - Patente (cadena de 7)

Definir tipos y desarrollar funciones para:

a) Obtener la cantidad de radares que en zonas pares han hecho infracciones el 19/12. [Las pilas no pueden perderse. No pueden usarse estructuras auxiliares.]

b) Generar una lista circular (ordenada por el total de infracciones de la zona) que contenga en cada nodo, sólo para aquellas zonas que tengan infracciones por Radar y por Cámara (ambas). [Las pilas pueden perderse]

- Zona (no se repite)
- Cant infracciones Radares
- Cant infracciones Camaras

c) Definir el tipo de la pila asumiendo que es estática. Desarrollar los operadores SacaP y PoneP.

Ej 3.- (*Utilizar TDA N-Ario*) Se tiene un árbol NArio de cadenas, mediante una función entera, determinar cuántas cadenas de longitud impar que comiencen y terminen en vocal se encuentran en niveles impares.

Ej 4.- Se tiene un digrafo G = (V,E) con V=|N| y aristas ponderadas, almacenado en una matriz de adyacencia y un ABB de enteros. Determinar de forma recursiva sobre la matriz, si para cada vértice de G que tenga bucle se verifica que su grado de entrada se encuentra en el ABB. Definir tipos.

Diciembre 2021

1 - V o F

a) En la matriz final tras ejecutar Floyd, todos los elementos de la diagonal principal son 0 a menos que haya bucle FALSO

La matriz de Floyd se inicializa con 0's en la diagonal, los cuales no varían sin importar que el vértice tenga o no un bucle

b) Una cola implementada dinamicamente de modo circular permite aprovechar aquellos nodos que ya no tienen datos significativos. VERDADERO

Serían aprovechables aquellos nodos que siguen en la cola y ya no es de interes su contenido

2- Municipalidad. Infracciones realizadas (en cada camara/radar)

TDAPila.h

```
#define FEC 11   #define PAT 8   #define MAX 100

typedef struct{
   Char FechaHora[FEC];
   Char Patente[PAT]; } TelementoP;

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

TDAPila.c

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

void SacaP(TPila *P, TelementoP *x){
    if ((*P).tope != -1)
       *x = ((*P).datos[(*P).pri--]
}
void PoneP(TPila *P, TelementoP x){
    if ((*P).tope != MAX-1)
      (*P).datos[++(*P).tope] = x
}
```

main.c

```
#include <stdio.h> #include <stdlib.h> #include <string.h> #include "TDAPila.h"
#define COD 5 #define

typedef struct nodoD{
   unsigned int zona;
   Char Codigo[COD];
   TPila PilaInf;
   struct nodoD *ant, *sig; } nodoD;
typedef nodoD *PnodoD;
typedef struct{
   PnodoD pri, ult; } TlistaD;
// Lista doble de camaras/radares

typedef struct nodoC{
   unsigned int zona;
   unsigned int CantInfC, CantInfR;
   Struct nodoC *sig; } nodoC;
typedef nodoC *TlistaC;
// Lista circular de zonas c infracciones

typedef struct{
   unsigned int CantC;
   unsigned int CantE; } TregE;
```

a) Obtener la cantidad de radares que en zonas pares han hecho infracciones el 19/12

```
void BuscaEnPila (TPila *P, Char DiaInf[], int *Hizo){
    TElementoP Inf;
    if (!VaciaP(*P)){
        SacaP (P, &Inf);
        if ((strnstr(Inf.FechaHora, DiaInf, 4)) != NULL)
            *Hizo = 1;
        else
            BuscaEnPila (P, DiaInf, Hizo);
        PoneP(P, inf);
    }
}
                                    Radar o camara, Dia de interes
void CuentaInfracciones (TListaD LD, Char Disp, Char DiaInf[], int *Contador){
    PnodoD auxZona;
    int Hizo;
    auxZona = LD.pri;
    *Contador = 0;
    while (auxZona != NULL){
        if (auxZona->Zona % 2 == 0 && auxZona->Codigo[0] == Disp){
            BuscaEnPila(&(auxZona->PilaInf), DiaInf, &Hizo)
                                                    0
            *Contador += Hizo;
        }
        auxZona = auxZona->sig;
    }
}
int main(){
    TListaD LD; LD.pri = NULL; LD.ult = NULL;
    int CantInf;
    Char DiaInf[5], Dispositivo;
    :
    CargaLista(&LD);
    :
    Printf("Ingrese el dispositivo de interes: \n");
    scanf("%c", &Dispositivo);
    Printf("Inarese el dia (MMDD): \n");
    scanf("%s", DiaInf);
    CantInf = 0;
    CuentaInfracciones(LD, Dispositivo, DiaInf, &CantInf);
    Printf("La cantidad de infracciones en zonas pares realizadas por
        un %c en el dia %s fueron %d", Dispositivo, DiaInf, CantInf);
    return 0;
}
```

b) Generar una lista circular (ordenada por el total de infracciones de la zona) que contenga (solo para aquellas zonas que tengan infracciones por radar y camara), en cada nodo (zona, cantR, cantC).

```
void GeneraVec(TListaD LD, TregZ VecZ[], int zonas){
    ProdoD auxZona, TelementoP Inf; int CantInf;
    InicializaVec(VecZ, zonas);
    auxZona = LD.pri;
    While(auxZona != NULL){
        CantInf = 0;
        While(!VaciaP(auxZona->PilaInf))
            SacaP(&(auxZona->PilaInf), &Inf);
            CantInf ++;
        }
        if (auxZona->Codigo[0]==C)
            VecZ[auxZona->Zona].CantC = CantInf;
        else
            VecZ[auxZona->Zona].CantR = CantInf;
        auxZona = auxZona->sig;
    }
}
void CreaLista(TListaC *LC, TregZ VecZ, int zonas){
    TListaC ant, act, nuevo;
    int i, Total;
    For (i=0; i<zonas; i++){
        if (VecZ[i].CantC != 0 && VecZ[i].CantR != 0){
            Total = VecZ[i].CantC + VecZ[i].CantR;
            nuevo = (TListaC) malloc(sizeof(nodoC));
            nuevo->CantInfC = VecZ[i].CantC; nuevo->CantInfR = VecZ[i].CantR;
            if (*LC = NULL){
                nuevo->sig = nuevo; *LC = nuevo;}
            else{
                if (Total > (*LC)->CantInfR + (*LC)->CantInfC){
                    nuevo->sig = (*LC)->sig, (*LC)->sig = nuevo; (*LC) = nuevo;}
                else{
                    ant = *LC; act = (*LC)->sig;
                    while(Total > act->CantInfR + act->CantInfC){
                        ant = act; act = act->sig;}
                    ant->sig = nuevo; nuevo->sig = act;}
    }}}
}
```

3- Se tiene un arbol NArio de cadenas, mediante una funcion entera, determinar cuantas cadenas de longitud impar que comiencen y terminen en vocal se encuentran en niveles impares

```
int CantCadenas(ArbolN A, posicion P, int nivel){
    int long, incr;
    Char Cadena[MAX];
    if (!Nulo(P)){
        if (nivel%2==0)
            incr=0;
        else{
            strcpy(Cadena, Info(P,A));
            long = strlen(Cadena);
            if (long%2==1 && EsVocal(Cadena[0]) && EsVocal(cadena[long-1]))
                incr=1;
            else
                incr=0;
        }
        return incr + CantCadenas(A, HijoIzq(P,A), nivel+1) + CantCadenas(A, HnoDer(P,A), n
    }
    else
        return 0;
}
```

4- Se tiene un digrafo G=(V,E) con V=|N| y aristas ponderadas almacenado en una matriz y un ABB. Determinar de forma recursiva sobre la mat, si para cada vértice de G que tenga bucle, su grado de E está en el ABB

```
int EstaEnABB(arbol ABB, int buscado){
    if (ABB != NULL)
        if (ABB -> dato == buscado)
            return 1;
        else
            if (ABB -> dato > buscado)
                return EstaEnABB(ABB -> Izq, buscado);
            else
                return EstaEnABB(ABB -> Der, buscado);
    else
        return 0;
}
int Verifican(int Mat[][MAX], int i, int j, int N, int GE, arbol ABB){
    if (j == N)
        return 1;
    else
        if (i == N)
            if (EstaEnABB(ABB, GE))
                return Verifican(Mat, 0, j+1, N, 0, ABB)
            else
                return 0;
        else
            if (Mat[i][i] != 0)
                return Verifican(Mat, i+1, j, N, GE + Mat[i][i] != 0, ABB)
            else
                return Verifican(Mat, 0, j+1, N, 0, ABB);
}
int main(){
    int Mat[MAX][MAX], CantV, cumple; arbol ABB
    :
    CargaDigrafo(Mat, CantV); CargaArbol(&ABB);
    :
    cumple = Verifican(Mat, 0, 0, CantV, 0, ABB);
    if (cumple)
    :
    else
    :
    return 0;
}
```