TOTALIZADOR PROGRAMACION II

19.2.2024

Apellido, Nombre: Biasucci Matias

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 4p 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 los ejercicios de desarrollo, mostrar las invocaciones (incluyendo su contexto: declaraciones de variables y tipos, inicializaciones y acciones posteriores) de las soluciones desarrolladas.

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

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

a) Si el valor de la arista (i,j) de G cambia cuando se está ejecutando el algoritmo de Dijkstra sobre G, el valor de los caminos mínimos ya calculados no cambiarán si i y j ya estén en S.

b) Dado G un grafo simple conexo con |V| = |E| = N, existen sobre G al menos N árboles abarcadores.

Ej 2.- Un sistema de reproducción de música administra las *playlists* públicas en una lista doblemente enlazada LP con la siguiente estructura:

- Id Playlist (Numérico, ordenada)
- Sublista de canciones
  - Id Canción (cadena de 15, los primeros 4 caracteres determinan el intérprete)
  - Título canción (Cadena de 20)
  - Duración (en segundos)

Se tiene una *playlist C* que ha tomado a su vez canciones de LP, implementada mediante una cola, en la que cada elemento contiene *Id Playlist (ordenada por este criterio), Id Canción.*

Se pide: utilizando el TDA COLA

a) En un único recorrido de la cola resolver (sin perder la cola ni cambiar el orden de la misma):

i) Hallar el tiempo en horas y minutos de reproducción de la playlist C (sin considerar las que se eliminan en ii))

ii) Eliminar de C aquellas canciones que estén interpretadas por "DUKI" y sean de la playlist X (X es dato de entrada)

b) Dado un Id Interprete (cadena de 4), eliminar sin recorrer LP más de una vez todas las canciones disponibles de él en las playlists X e Y de LP (X e Y datos validados) y listarlas (con el formato siguiente)

```
Intérprete: XXXX

PLAYLIST 99999
   Id Canción        Titulo Cancion
    xxxxxxxxxx     xxxxxxxxxxxxxxxxxxx
    xxxxxxxxxx     xxxxxxxxxxxxxxxxxxx
```

c) Suponiendo la cola circular definir su tipo y desarrollar los operadores *PoneC()* y *VaciaC()*. Indicar donde estarían estas definiciones.

Ej 3.- (*Utilizar TDA N-Ario*) Dado un árbol A N-Ario de enteros, determinar si hay exactamente K nodos no hoja que tienen grado igual al nivel en el que se encuentran (K es dato)

Ej 4.- Dado un grafo de N vértices implementado en una matriz de adyacencia, determinar mediante una función recursiva entera si los primeros K vértices (K dato) tienen el mismo grado.

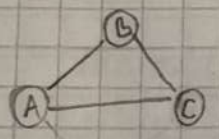
Febrero 2024

1- V o F

a) Si el valor de la arista (i, j) de G cambia cuando se está ejecutando el algoritmo de Dijkstra sobre G, el valor de los caminos mínimos ya calculados no cambiaran si i y j ya están en S VERDADERO

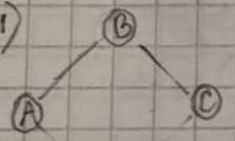
La arista (i, j) se analiza cuando se incluye i o j en el conjunto. Si ambos vertices ya están incluidos, por mas de que (i, j) cambie, los caminos mínimos ya calculados no lo harán

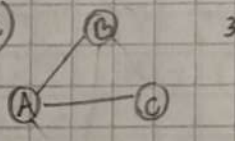
b) Dado G un grafo simple conexo con |V| = |E| = N, existen sobre G al menos N árboles abarcadores VERDADERO

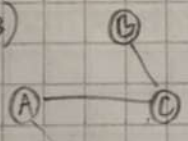


Para que un grafo G sea conexo necesita al menos N-1 aristas siendo N = |V|

Para |V| = 3 Como |E| = N, para cualquier |V| nos sobrará una arista

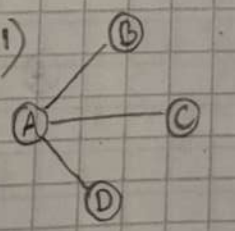
1) 

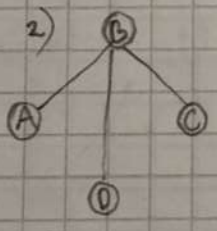
2) 

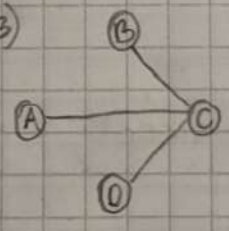
3) 

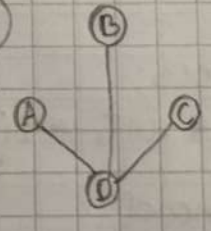
Para |V| = 4

Los posibles arboles abarcadores serían:

1) 

2) 

3) 

4) 

Para |V| = 2 no se cumple la hipotesis

Es de esperar que existan N arboles abarcadores por lo menos para cualquier grafo simple conexo con |V| = |E| = N.

2- Sistema de reproducción de música → Lista doble de playlist

LP { Id Playlist (numérico, ordenada)
Sublista de canciones { Id Cancion, Titulo cancion, Duración }

C { Id playlist, Id cancion }

a) En un único recorrido de la cola resolver

i) Hallar el tiempo en horas y minutos de reproduccion de la playlist c sin considerar las que se eliminan en ii)

ii) Eliminar de C aquellas canciones que estén interpretadas por "DUKI" y sean de la playlist x (x dato de entrada)

main.c

```
#include <stdio.h>  #include <stdlib.h>  #include <string.h>
#include "TDACola.h"
#define INTER 5  #define CAN 16  #define TIT 21
typedef struct nodito{
    Char IdC[CAN];
    Char TitC[TIT];
    Unsigned int Duracion;
    Struct nodito *sig;} nodito;
typedef nodito *Tsub;
typedef struct nodoD{
    int IdP;
    Tsub SubCanciones;
    struct nodoD *ant, *sig;} nodoD;
typedef nodoD *PnodoD;
typedef struct{
    PnodoD pri,ult;} TlistaD;
```

TDACola.h

```
#define CAN 16   #define MAX 100
typedef struct{
   int IdP;
   Char IdC; } TelementoC;
typedef struct{
   TelementoC datos[MAX];
   Int pri, ult; } TCola;
int VaciaC(TCola);
Void PoneC(TCola *, TElementoC);
```

TDACola.C

```
#include <stdio.h>  #include <stdlib.h>
#include "TDACola.h"
int VaciaC(TCola C){
   return C.pri==-1;
}
void PoneC(TCola *C, TElementoC x){
   if (!((*C).pri==0 && (*C).ult==MAX-1 || (*C).ult==(*C).pri-1)){
      if ((*C).ult==MAX-1)
         (*C).ult=0;
      else
         (*C).ult+=1;
      (*C).datos[(*C).ult]=x;
      if ((*C).pri==-1)
         (*C).pri=0;
   }
}
```

a) void HallayElimina(TCola *C, TlistaD LP, int playlistx, char Elim[]){

```
    TElementoC RegC;                    Char Interprete[INTER]
    PnodoD auxP; TSub auxS;
    Unsigned int HorasC, MinC, SegC, n = INTER;
    RegC.IdP = -1; SegC = 0;
    PoneC(C, RegC);
    SacaC(C, &RegC);
    While(RegC.IdP != -1){
        strncpy(Interprete, RegC.IdC, n);
        If (!(strcmp(Interprete, Elim)==0) && RegC.IdP != Playlistx){
            auxP = LD.pri;
            while(auxP -> IdP != RegC.IdP)
                auxP = auxP -> sig;
            auxS = auxP -> SubCanciones;
            while(strcmp(auxS -> IdC, RegC.IdC) != 0)
                auxS = auxS -> sig;
            SegC += auxS -> Duracion;
            PoneC(C, RegC);
        }
        SacaC(C, &RegC);
    }
    HorasC = SegC/3600;
    SegC = SegC % 3600;
    MinC = SegC/60;
    SegC = SegC % 60;
    Printf("Duracion de la playlist: %d horas %d minutos", HorasC, MinC);
}
```

b)
```
void Elimina(TListaD LP, char interprete[], int x, int y){
    PnodoD auxD;
    Tsub acts, ants, elims; int listadas;
    Printf("Interprete: %s \n", interprete);
    auxD = LP.pri;
    listadas = 0
    while (listados < 2){
        if (auxD -> idP == x || auxD -> idP == y){
            printf("\n PLAYLIST %s \n", auxD -> idP); acts = auxD -> subC;
            Listadas++;
            while(acts != NULL)
            if (strstr(acts -> idC, interprete)){
                printf("%s %s", acts -> idC, acts -> Titc);
                elim = acts;
                acts = acts -> sig;
                if (elim == auxD -> subCanciones)
                    auxD -> subCanciones = acts;
                else
                    ants -> sig = acts;
                free(elim);
            }
            else{
                ants = acts;
                acts = acts -> sig;
            }
        }
        auxD = auxD -> sig;
    }
}
```

3- Dado un arbol A N-Ario de enteros, determinar si hay exactamente K nodos no hoja que tienen grado igual al nivel en el que se encuentran

```
int Cuenta(ArbolN A, posicion P, int K, int nivel){
    Posicion c;
    int grado, cont = 0;
    if (!Nulo(P)){
       C = HijoMasIzq(P,A);
       grado = 0;
       While (!Nulo(c) && cont<=K){
           grado++;
           Cont += Cuenta(A, C, K, nivel+1);
           C = HnoDer(C,A);
       }
       return Cont + grado == nivel;
    }
    else
       return 0;
}
```

4- Dado un grafo de N vértices implementado en una matriz de ady, determinar mediante una función recursiva entera si los primeros K vertices tienen el mismo grado

```
int PrimerosK(int Mat[][MAX], int i, int j, int N, int K, int grado, int gradoant){
    if (i==K)
       return 1;
    else
       if (j==N)
          if (i==0 || grado==gradoant)
             return PrimerosK(Mat, i+1, 0, N, K, 0, grado);
          else
             return 0;
       else
          if (i!=0 && grado>gradoAnt)
             return 0;
          else
             return PrimerosK(Mat, i, j+1, N, K, grado+Mat[i][j], gradoAnt);
}
```