TOTALIZADOR PROGRAMACION II

Serán considerados al calificar este examen la eficiencia de las soluciones y del uso de las características del lenguaje C y de la programación estructurada.

Para aprobar es necesario obtener al menos 5 puntos

Ej 1) a) *(Utilizar TDA Cola)* Se tiene una matriz T de NxN naturales que representa un AAM construido sobre un grafo. Desarrollar una solución recursiva sobre la matriz que almacene en la cola los vértices y su grado en el AAM, sólo para aquellos vértices que grado > 1 en el AAM, retornar además el costo del AAM.

- Si la solución tiene algún ciclo el puntaje será 0
- Cada elemento de la cola será un registro con dos campos: *vértice* y *grado*.
- Mostrar la invocación de la solución desarrollada

b) Suponer que la Cola está implementada en memoria estática, definir el tipo de la Cola y desarrollar el operador PoneC utilizado en la solución del inciso **a)**.

Ej 2) Una lista simplemente enlazada se generó luego de haber analizado un texto, en cada nodo:

- ✓ Letra (Ordenada, pueden no estar todas las letras del alfabeto)
- ✓ SubLista de palabras que se hallaron en el texto
  - Palabra (Ordenada)
  - Apariciones (>0)

Se pide, definir tipos y desarrollar subprogramas para:

a) Dada una palabra que fue incorrectamente incluida en lista, eliminarla.

b) Generar una lista circular que contenga en cada nodo, sólo las palabras que comienzan con vocal y aparecieron más de 5 veces con el siguiente contenido en cada nodo:

- ✓ Palabra (Ordenada, comienza con vocal)
- ✓ Apariciones (>5)

Ej 3) (*Utilizar TDA N.Ario*) Dado un árbol n-ario de enteros, determinar mediante una función int cuántas hojas en el nivel K contienen un 0.

- K es dato
- Mostrar la invocación de la función desarrollada

Modulo Provisorio

1-a) Matriz T de N×N naturales que representa un AAM construido sobre un grafo. Desarrollar una solucion recursiva sobre la matriz que almacene en la cola los vertices y su grado, solo para aquelos vertices de grado > 1. Retornar el costo del AAM

```
void Almacena(unsgned int T[][MAX], int i, int j, int N, TCola *C, int*costo, int grado)
 if(i < N){                    // Telementoc V;
   if (j == N){
     if (grado > 1){
       V.Vertice = i+1;
       V.grado = grado;
       PoneC(C, V);
     }
     Almacena(T, i+1, 0, N, C, costo, 0);
   }
   else{
     if(T[i][j] != 0){
       grado++;
       if(j > i)
         *Costo += T[i][j];
     }
     Almacena(T, i, j+1, N, C, costo, grado);
   }
 }
}
```

b) Suponer que la cola esta implementada en memoria estatica, definir el tipo de la cola y desarrollar el operador PoneC

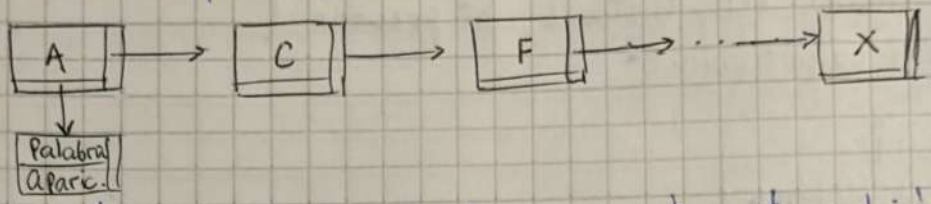
TDACola.h

```
#define MAX 100

typedef struct{
  int Vertice;
  int grado;} Telementoc;

typedef struct{
  Telementoc datos[MAX];
  int pri, ult;} TCola;
```

TDACola.C

```
#include <stdio.h> #include "TDACola.c"

void PoneC(TCola *C, Telementoc x){
   if ((*C).ult != MAX-1){
      (*C).datos[++(*C).ult] = x;
      if ((*C).pri == -1)
         (*C).pri = 0;
   }
}
```

2- Lista simple con sublistas de las palabras presentes en un texto



a) Dada una palabra que fue incorrectamente incluida, eliminarla

```
void EliminaPal(Tlista L, char Palabra[], int *eliminada){
    Tlista actL;
    TSub antP, actP;
    actL = L;
    while (actL != NULL && Palabra[0] > actL->Letra)
        actL = actL->sig;
    if (actL != NULL && Palabra[0] == actL->Letra){
        actP = actL->SubPalabras
        while (actP != NULL && strcmp(Palabra, actP->Palabra)>0){
            antP = actP;
            actP = actP->sig;
        }
        if (actP!=NULL && strcmp(Palabra, actP->Palabra)==0){
            if (actP == actL->SubPalabras)
                actL->subPalabras = actP->sig;
            else
                antP->sig = actP->sig;
            free(actP)
            *eliminada = 1;
        }
    }
}
```

b) Generar una lista circular que contenga en cada nodo solo las pal. que comienzan con vocal y aparecieron mas de 5 veces

```
void GeneraListaC(TlistaC *LC, Tlista L, int frecmin){
   TlistaC palnueva, palant, palact;
   TSub auxs;
   While(L!=NULL){
      auxs=L->SubPalabras;
      While(auxs!=NULL){
         if(auxs->apariciones > frecmin && EsVocal(auxs->Palabra[0]){
            palnueva=(TlistaC) malloc(sizeof(nodoC));
            Palnueva->apariciones=auxs->apariciones;
            strcpy(Palnueva->palabra, auxs->Palabra);
            if(*LC==NULL){
               palnueva->sig=palnueva;
               *LC=palnueva;
            }
            else
               if(strcmp(Palnueva->Palabra, (*LC)->Palabra)>0){
                  (*LC)->sig=palnueva;
                  Palnueva->sig=(*LC)->sig; *LC=palnueva;
               }
               else{
                  palant=*LC;
                  palact=(*LC)->sig;

                  while(strcmp(Palnueva->palabra, palact->palabra)>0){
                  palant=palact;
                  palact=palact->sig;
                  }
                  palant->sig=palnueva;
                  palnueva->sig=palact;
               }
         }
         auxs=auxs->sig;
      }
      L=L->sig;
   }
}
```

3) Dado un NArio de enteros determinar cuantas hojas en el nivel K contienen un 0

```
int CuentaCeros(ArbolN AN, Posicion P, int nivel){
   int incr;
   if(!Nulo(P))
      if(Nivel==K && NuloHijoMasIzq(P,AN) && Info(P,AN)==0)
         return 1 + CuentaCeros(AN, HnoDer(P,AN), nivel);
      else
         return CuentaCeros(AN, HijoMasIzq(P,AN), nivel+1) + CuentaCeros(AN, HnoDer(P,AN), nivel);
   else
      return 0;
```