**Ejercicio 1 ( 0,75 de 1,5)**

a) Falso.En una lista circular si el nodo a eliminar es el primero se considera un caso particular. Para eliminar un nodo ubicado en la primer posición , si existen más elementos en la lista, debemos hacer que el nodo anterior al primero apunte al siguiente del primero(ant->sig=(\*LC)->sig) y luego el primero pasará a ser el anterior del primero ((\*LC)=ant) ,luego se procede a liberar el espacio del nodo que antiormente era el primero. La respuesta es VERDADERO, NO es un caso particular, a menos que sea el unico

b) Falso. El algoritmo de Dijkstra permite obtener caminos mínimos con un sólo origen, el algoritmo de Floyd es aplicar Dijkstra pero con distintos origenes(caminos mínimos entre todos los pares de vértices).En cambio,Prim es un método de obtención de árboles abarcadores de costos mínimos(AAM) y se aplica sólo para grafos sin sentido, a diferencia de Dijkstra y Floyd que se aplican en digrafos. BIEN

**Ejercicio 2 (3,25 de 4)**

#include <stdlib.h>

#include "cola.h"

void entrada(int [][],int ,int,int,int,Tcola \*); // debe especificar cantidad de columnas

int main(){

int A[20][20];

int n;

Tcola C;

printf("ingrese cantidad de vertices N");

scanf("%d",&n);

entrada(A,n-1,0,n-1,n-1,&c);

return 0;

}

void entrada(int A[][n],int n, int grado,int i,int j,Tcola \*c){ // debería ser constante (20 para este caso)

Tcola auxC;

TelementoC x;

if(j>=0){

if(i>=0){

if(A[i][j]!=0)

entrada(A,n,grado+1,i-1,j,c);

else

entrada(A,n,grado,i-1,j,c);

}

else{

iniciaC(&auxC); //debería modularizar

while(!vaciaC(\*c) && x!=grado){ // no tiene valor en la 1ra. iteración

sacaC(c,&x);

if(grado!=x)

poneC(&auxC,x);

}

while(!vaciaC(auxC){

sacaC(&auxC,&x);

poneC(c,&x);

}

entrada(A,n,0,n,j-1,c);

}

}

}

ii)TDA cola estática cola.h

typedef int TelementoC;

typedef struct{

TelementoC datos[50];

int pri,ult;

}Tcola;

void iniciaC(Tcola \*c);

void poneC(Tcola \*c,TelementoC x);

void sacaC(Tcola \*c,TelementoC \*x);

TelementoC consultaC(Tcola c);

int vaciaC(Tcola c);

cola.c

// No se piden los operadores del TDA

#include <stdlib.h>

#include "cola.h"

void iniciaC(Tcola \*c){

(\*c).pri=-1;

(\*c).ult=-1;

}

void poneC(Tcola \*c,TelementoC x){

if((\*c).ult!=49){

if((\*c).pri==-1)

(\*c).pri=0;

(\*c).datos[++(\*c).ult;

}

}

void sacaC(Tcola \*c,TelementoC \*x){

if((\*c).pri!=-1){

(\*x)=(\*c).datos[(\*c).pri];

if((\*c).ult==(\*c).pri)

iniciaC(c);

else

(\*c).pri+=1;

}

}

TelementoC consultaC(Tcola c){

if(c.pri!=-1)

return c.datos[c.pri];

}

int vaciaC(Tcola c){

return c.pri==-1;

}

**Ejercicio 3 (2,5 de 4,5)**

typedef struct nodoA{

int dato;

struct nodoA \*izq,\*der; // el árbol no es binario!

}nodoA;

typedef nodoA \*Tarbol;

typedef struct nodoD{

int nivel,clave;

struct nodoD \*sig,\*ant;

}nodoD;

typedef nodoD \*PnodoD;

typedef struct{

PnodoD pri,ult;

}TlistaD;

void generarlistaD(Tarbol ,posicion,int , TlistaD \*);

void eliminarK(TlistaD \*,int );

int main(){

Tarbol A;

TlistaD LD;

int K;

LD.pri=NULL;

LD.ult=NULL;

generarlistaD(A,raiz(A),1,&LD);

printf(" ingrese el nivel K para eliminar de la lista aquellas claves que se encontraba en dicho nivel");

scanf("%d",&K);

eliminarK(&LD,K);

return 0;

}

void generarlistaD(Tarbol A,posicion p,int niv,TlistaD \*LD){

posicion c;

PnodoD actD,auxD;

if(!nulo(p)){

c=hijomasizq(p,A);

if(nulo(c) && niv%2==1){

auxD=(PnodoD)malloc(sizeof(nodoD)); //debe modularizar

auxD->nivel=niv;

auxD->clave=info(p,A);

if((\*LD).pri==NULL || (\*LD).pri->nivel>auxD->nivel){ //podría ser >=

auxD->sig=(\*LD).pri;

if((\*LD).pri==NULL)

(\*LD).ult=auxD;

else

(\*LD).pri->ant=auxD;

(\*LD).pri=auxD;

// falta poner ant del nuevo nodo en nulo

}

else

if(auxD->nivel>(\*LD).ult->nivel){ //podría ser >=

(\*LD).ult->sig=auxD;

auxD->ant=(\*LD).ult;

(\*LD).ult=auxD;

// falta poner sig del nuevo nodo en nulo

}

else{

actD=(\*LD).pri;

while(actD && auxD->nivel>actD->nivel){//actD nunca puede ser ni volverse NULL

actD=actD->sig;

}

auxD->sig=actD; // [1]

actD->ant=auxD;

actD->ant->sig=auxD; // enlaza de nuevo lo hecho en [1]. Debe tener en cuenta el orden de los enlaces..

auxD->ant=actD->ant; // enlaza consigo mismo?

}

}

generarlistaD(A,hijomasizq(p,A),niv+1,LD);

generarlistaD(A,hermanoder(p,A),niv,LD);

}

}

void eliminarK(TlistaD \*LD,int K){

PnodoD elim,actD;

actD=(\*LD).pri;

while(actD && K>actD->nivel){

actD=actD->sig;

}

if(actD){

printf(" CLAVES ELIMINADAS \n");

while(actD && K==actD->nivel){

elim=actD;

actD=actD->sig;

printf(" %d \n",elim->clave); // se pide informar la cantidad, no cuales

if(elim==(\*LD).pri)

(\*LD).pri=(\*LD).pri->sig; // falta poner ant del nuevo primero en nulo

else

if((\*LD).pri==elim && (\*LD).ult==elim){ // debería preguntarlo primero! If mal anidados (no entra nunca!)

(\*LD).pri=NULL;

(\*LD).ult=NULL;

}

else{

elim->ant->sig=elim->sig;

elim->sig->ant=elim->ant;

}

// no tiene en cuenta caso de que sea el último

free(elim);

}

}

else

printf(" no existe el nivel \n");

}