**TAD Vértice**

**Conjunto mínimo de datos:**

Contenido, tipo plantilla(T), representa la información guardada en el vértice.

**Operaciones:**

Vertice(), constructor, crea un vértice;

Vertice( pContenido), crea un vértice con contenido pContenido.

getContenido(), retorna el contenido del vértice.

void setContenido( nuevo), cambia el contenido del vértice por nuevo.

**TAD Arista**

**Conjunto mínimo de datos:**

Valor, representa el inicio de las aristas;

Adyacentes, representa una lista de vértices que son adyacentes al vértice valor.

**Operaciones:**

ListaAdyacencia(), crea una lista de ayacencias;

ListaAdyacencia(T contenido), crea una llsta de adyacencias para un vertice con contenido contenido;

getAdyancentes(), retorna la colección de vértices adyacentes;

getValor(), retorna el valor del vértce inicio;

agregarAdyacente(vertice), agrega el vértice vertice a la lsita de ayacentes.

buscar( contenidoVertice). Busca un vertice con contenido contenidoVertice en la lista de adyacentes.

**TAD Grafo**

**Conjunto mínimo de datos:**

Vértices, lista de vértices del grafo;

matrizAdyacentes, representa todas las aristas en el grafo.

**Operaciones:**

Grafo();

bool agregarVertice (nuevo), agrega un nuevo vertice a a lista de vértices del grafo.

bool agregarArista(inicio, fin), agrega a las aristas la arista con inicio inicio y fin fin.

**Código Grafo**

#ifndef \_\_GRAFO\_H

#define \_\_GRAFO\_H

#include <vector>

template<class T>

class Grafo{

private:

vector<Vertice<T>\*> vertices;

vector<ListaAdyacencia<T>\*> matrizAdyacentes;

public:

Grafo();

bool agregarVertice(Vertice<T>\* nuevo);

bool agregarArista(T inicio, T fin);

};

#include "Grafo.hxx"

#endif

**Grafo.hxx**

#include "Grafo.h"

template<class T>

Grafo<T>::Grafo(){

vertices= new vector<Vertice<T>\*>();

matrizAdyacentes= new vector<ListaAdyacencia<T> >();

}

template <class T>

bool Grafo<T>::agregarVertice(Vertice<T>\* nuevo){

/\*

\* falta verificar que no exista

\*/

vertices->push\_back(nuevo);

//agregar a la lista de adyacencias el nuevo espacio

ListaAdyacencia<T>\* temporal= new ListaAdyacencia<T>(nuevo->getContenido());

matrizAdyacentes->push\_back(temporal);

}

template<class T>

bool Grafo<T>::agregarArista(T inicio, T fin){

//primero busco el vertice fin en la lista de vertices del grafo.

//buscar lista de adyaencia en la matriz de adyacencia igual a inicio

//agregar el vertice a la lista de adyacencia.

Vertice<T>\* temporal=NULL;

for(int i=0; i< vertices->size();i++){

if((\*vertices)[i]->getContenido()==fin){

temporal = (\*vertices)[i];

break;

}

}

if(temporal!=NULL){

for(int i=0; i< matrizAdyacentes->size();i++){

ListaAdyacencia<T>\* temporal= (\*matrizAdyacentes)[i];

if(temporal->getValor()==inicio){

temporal->agregarAdyacente(temporal);

return true;

}

}

return false;

}else{

return false;

}

}

**ListaAdyacencia**

#ifndef \_\_LISTAADYACENCIA\_H

#define \_\_LISTAADYACENCIA\_H

#include <vector>

#include "Vertice.h"

template<class T>

class ListaAdyacencia{

private:

//hubiera sido mejor poner un vertice

T valor;

/\* falta cambiar el valor por un vertice\*/

std::vector< Vertice<T>\* >\* adyacentes;

public:

ListaAdyacencia();

ListaAdyacencia(T contenido);

std::vector<Vertice<T>\*>\* getAdyancentes();

T getValor();

void agregarAdyacente(Vertice<T>\* vertice);

bool buscar(T contenidoVertice);

};

#include "ListaAdyacencia.hxx"

#endif

**ListaAdyacencia.hxx**

#include "ListaAdyacencia.h"

using namespace std;

template<class T>

ListaAdyacencia<T>::ListaAdyacencia()

{

adyacentes= new vector<Vertice<T>\*>();

}

template<class T>

ListaAdyacencia<T>::ListaAdyacencia(T pContenido)

{

valor= pContenido;

adyacentes= new vector<Vertice<T>\*>();

}

template<class T>

std::vector<Vertice<T>\*>\* ListaAdyacencia<T>::getAdyancentes()

{

return adyacentes;

}

template<class T>

void ListaAdyacencia<T>::agregarAdyacente(Vertice<T>\* vertice)

{

/\*

\*revisar el caso en que no exista, falta agregarlo

\*/

adyacentes->push\_back(vertice);

}

template<class T>

bool ListaAdyacencia<T>::buscar(T contenidoVertice){

for(int i=0; i<adyacentes->size(); i++){

Vertice<T>\* temp= (\*adyacentes)[i];

if(temp->getContenido()==contenidoVertice){

return true;

}

}

return false;

}

template<class T>

T ListaAdyacencia<T>::getValor(){

return valor;

}

**Veritce**

#ifndef \_\_VERTICE\_H

#define \_\_VERTICE\_H

template<class T>

class Vertice

{

private:

T contenido;

public:

Vertice();

Vertice(T pContenido);

T getContenido();

void setContenido(T nuevo);

};

#include "Vertice.hxx"

#endif

**Vertice.hxx**

#include "Vertice.h"

template<class T>

Vertice<T>::Vertice(){

}

template<class T>

Vertice<T>::Vertice(T pContenido){

contenido=pContenido;

}

template<class T>

T Vertice<T>::getContenido(){

return contenido;

}

template<class T>

void Vertice<T>::setContenido(T pContenido){

contenido=pContenido;

}