

Programación III

Trabajo Obligatorio

Licenciatura en Informática

Ingeniería en Informática

Profesor: Pablo Denis

Horario: Lunes y Miércoles de 18 a 20 horas

Victoria Aloy. Cédula: 4.713.341-6

Juan Pablo Rosa. Cédula: 4.610149-6

Total hojas: 8

Índice

[**Modelado del problema de las previaturas mediante un grafo**](#_f47e6ucqltoq) **3**

[**Análisis de T.A.D para esta realidad y estructuras de datos más adecuadas para representarlos**](#_7m52j8ptg4kh) **4**

[**Esquema de módulos para los T.A.D elegidos**](#_r0vcni4vtf0f) **5**

[**Implementación del sistema en C++**](#_6l14z34t4i6u) **5**

[**Estructura y encabezados de las operaciones de los T.A.D**](#_m5wdzsmw4bgl) **6**

[AvlAlumno](#_pipsf1rrmp3l) 6

[ListaAprob](#_wodjj4abuv6p) 7

[MapeoAsig](#_u326bhx653jl) 7

## 

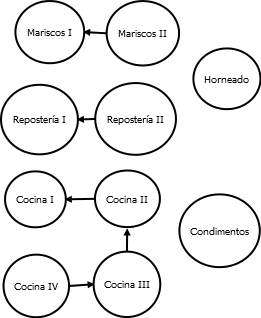
## 

## Modelado del problema de las previaturas mediante un grafo

Las 10 materias que posee la carrera y sus previas se representan en la siguiente tabla.

|  |  |
| --- | --- |
| **Materias** | **Previas** |
| Mariscos I | No tiene |
| Mariscos II | Mariscos I |
| Repostería I | No tiene |
| Repostería II | Repostería I |
| Cocina I | No tiene |
| Cocina II | Cocina I |
| Cocina III | Cocina II |
| Cocina IV | Cocina III |
| Horneado | No tiene |
| Condimentos | No tiene |

El grafo se ve representado en la siguiente imagen.



Los vértices del grafo representan las materias de la carrera y aristas representan el sistema de previaturas.

Es un grafo es dirigido ya que lo creemos conveniente debido a la realidad planteada (previaturas) y una materia tiene, por ejemplo: Mariscos I no tiene previa y Mariscos II tiene a Mariscos I como previatura, pero no al revés.

El orden, osea la cantidad de vértices de G es 10, ya que hay 10 materias en la carrera. El grafo tiene 5 artistas. Esto último implica que sea finito.

Es simple ya que todos los elementos tienen cardinal 2. Es decir, si no existen lazos en el grafo. No es un grafo completo ya que no están unidos todos los vértices por una arista.

Hay vértices aislados. El grado mínimo del grafo es 0 y el máximo es 2, lo que implica que no es un grafo k regular.

El grafo tiene 5 caminos y no hay ciclos. Dos caminos tienen largo 0, otros dos largo 1 y el último largo 4. Los caminos son independientes.

La distancia del grafo es 0 (largo del camino más corto). El diámetro es 3.

El grafo es no conexo, ya que hay vértices que no los une ningún camino.

## Análisis de T.A.D para esta realidad y estructuras de datos más adecuadas para representarlos

Las asignaturas tendrán claves de identificación, no se admitirán elementos repetidos y se almacenarán en orden de identificador de menor a mayor, empezando por 0.

El T.A.D para esta estructura será del tipo Diccionario, dado que son idóneas para elementos que poseen una clave que los identifica. La estructura de Mapeo puede ser usada si se asume un diccionario acotado, según la letra la carrera está compuesta por una cantidad fija de asignaturas (en nuestro caso serán 10). Es posible usar la estructura Mapeo ya que los códigos de las asignaturas pertenecen a un subrango de los naturales, según la definición de la estructura mapeo esto aplica.

Los alumnos también tendrán claves de identificación, no se admitirán elementos repetidos y se almacenarán en orden de cédula de menor a mayor.

El tipo abstracto de datos para esta estructura también será del tipo Diccionario, por el mismo motivo que la anterior estructura: son adecuados para elementos que poseen una clave que los identifica. La estructura de AVL es ideal ya que no existe cota alguna para la cantidad de alumnos que se pueden inscribir en la carrera. Esta estructura es más eficiente cuando interesa recorrer los elementos del diccionario ordenados por clave de menor a mayor.

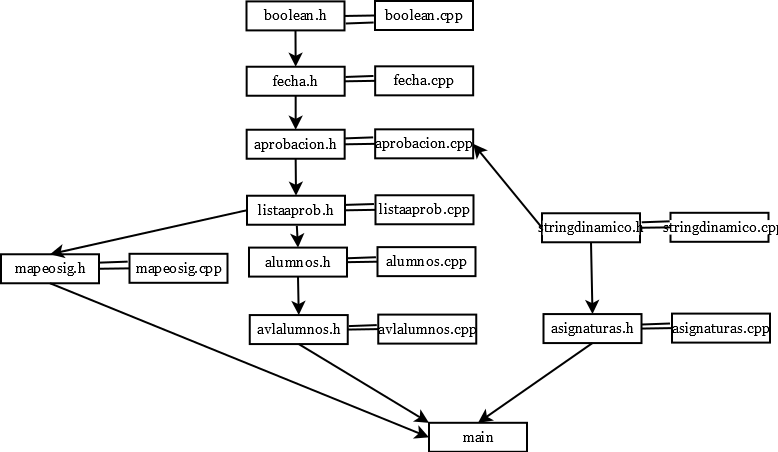
Ya que el AVL es un tipo especial de árbol binario de búsqueda, almacena los alumnos en orden y se le suma el plus que balancea el árbol. El balanceo hace que sea rápido en búsquedas ya que serán de O(log(n)) en el peor caso. Con esta implementación ganamos en la velocidad cuando se trata de buscar alumnos en el AVL.

Se deberá poder saber la cantidad de asignaturas aprobadas por alumno, también la fecha en el cual fueron aprobadas dichas asignaturas, para ello se agrego una lista de aprobaciones a cada alumno en la cual se podrá guardar qué asignatura fue aprobada, junto con una fecha y calificación respectiva.

El T.A.D que se optó para representar esta realidad es Secuencia, los elementos se almacenan linealmente (como en una fila). Dentro de la familia secuencia se eligió el tipo stack. La estructura elegida es una lista simple ya que nos permite agregar esta información sin necesidad de desperdiciar memoria cuando no sea necesario, y permitirá tener en orden cada materia que fuera salvada por dicho alumno.

## Esquema de módulos para los T.A.D elegidos

La siguiente imagen representa el diagrama de módulos del programa desarrollado.



## Implementación del sistema en C++

Siempre respetando la modularización y abstracción se desarrolló la tarea. Intentando no generar confusión y tener un código prolijo, mantenible.

Se desarrollaron las primitivas de los T.A.D y otras operaciones necesarias, para acceder a las estructuras se utilizaron sus selectoras y operaciones programadas.Los T.A.D fueron elegidos para que sean lo más convenientes para la realidad a desarrollar, evitando funciones, procedimientos innecesarios y maximizando la velocidad de tiempo en cada operación.

## Estructura y encabezados de las operaciones de los T.A.D

### AvlAlumno

typedef struct nodoA {  
 Alumno alum;  
nodoA \* hIzq;  
nodoA \* hDer;  
int altura;  
 } Nodo;  
typedef Nodo \* AvlAlum;  
  
int alturaAvlAlum(AvlAlum a);  
void balancearAvlAlum(AvlAlum &a);  
void rotacionSimpleIzquierda(AvlAlum &t);  
void rotacionSimpleDerecha(AvlAlum &t);  
void rotacionDobleDerecha(AvlAlum &t);  
void rotacionDobleIzquierda(AvlAlum &t);  
void crearAvlAlum(AvlAlum &a);  
boolean vacioAvlAlum(AvlAlum a);  
boolean perteneceAvlAlum(AvlAlum a, int cedula);  
void insertarAvlAlum(AvlAlum &a, Alumno alum);  
void postOrden (AvlAlum a);  
void orden (AvlAlum a);  
void preorden (AvlAlum a);  
Alumno obtenerMinimoAvlAlum(AvlAlum a);  
Alumno obtenerMaximoAvlAlum(AvlAlum a);  
Alumno obtenerAvlAlum(AvlAlum a, int cedula);  
void eliminarAvl(AvlAlum &a, int ci);  
AvlAlum unirAvl(AvlAlum izq, AvlAlum der);

### 

### ListaAprob

typedef struct nodo {

Aprobacion aprob;

nodo \* sig;

} NodoAprob;

typedef NodoAprob \* ListaAprob;

//crea lista vacia

void crearLista(ListaAprob &lis);

//dice si la lista esta vacia

boolean vaciaLista(ListaAprob lis);

//devuelve el primer elemento de la lista

//PRECOND: lista NO vacía

Aprobacion primero (ListaAprob lis);

//borra el primer elemento de la lista

//PRECOND: lista NO vacía

void resto(ListaAprob &lis);

//agrega un elemento al principio de la lista

void insFront (ListaAprob &lis, Aprobacion a);

//inserta en orden de fecha en la lista

void insOrdenFecha(ListaAprob &lis, Aprobacion a);

void mostrarListaAprob (ListaAprob l);

boolean perteneceLista (ListaAprob l, int codAsig);

void insBack (ListaAprob &lis, Aprobacion a);

### MapeoAsig

const int TAM=10;

typedef struct {

boolean existe;

Asignatura asig;

} T;

typedef T MapeoAsig [TAM];

boolean salvoPreviasMapeoAsig(MapeoAsig m, int codigo, ListaAprob l);

void crearMapeoAsig (MapeoAsig &m);

boolean perteneceMapeoAsig (MapeoAsig m, int pos);

void insertarMapeoAsig (MapeoAsig &m, Asignatura a, int pos);

int cantidadElemMapeo (MapeoAsig m);

//Precondición: Pertenece(m,pos)

Asignatura obtenerMapeoAsig (MapeoAsig m, int pos);

//Precondición: Pertenece(m,pos)

void eliminarMapeoAsig (MapeoAsig &m, int pos);

void mostrarMapeoAsig(MapeoAsig m);

void insertarMapeoAsigEntreDos(MapeoAsig &m);

void mostrarPreviasMapeoAsig(MapeoAsig m, int codAsig);

boolean completoMapeoAsig(MapeoAsig m);

boolean generaCicloMapeoAsig(MapeoAsig m, int codAsig);