**Universidad Peruana de Ciencias Aplicadas**

**UPC**



PROYECTO

Trabajo Final

CURSO DE ALGORITMOS Y ESTRUCTURAS DE DATOS

Carrera de Ingeniería de Software

Sección SW32

Alumno:

201618549 Alejandro Gray Hidalgo

201713987 Francisco Javier Orive Pais

Monterrico, Junio 2020

**CONTENIDO**

1. INTRODUCCIÓN

2. PROBLEMA

3. OBJETIVOS

4. ALCANCE DE PROYECTO

5. MARCO CONCEPTUAL

6. DIAGRAMA DE CLASES (ENTIDADES)

7. ASIGNACIÓN DE RECURSOS

8. CRONOGRAMA DE TRABAJO

9. DEFINIR REQUISITOS

10. DISEÑO DE INTERFAZ DE USUARIO

11. TIPOS DE DATOS ABSTRACTOS

12. SELECCIÓN E IMPLEMENTACIÓN DE ESTRUCTURAS DE DATOS

13. DIAGRAMA DE CLASES (SEGUNDO NIVEL)

14. IMPLEMENTACIÓN DE LAS FUNCIONALIDADES

15. DISEÑO DE ARCHIVOS

16. ARQUITECTURA DE LA APLICACIÓN

17. CONCLUSIONES

18. REFERENCIAS

**1.** **INTRODUCCIÓN**

El proyecto realizado consiste en un mini motor de Base de Datos que almacenará archivos en una estructura de datos tabular bidimensional a la que llamaremos Tabla. Dicha estructura organiza los datos de manera tabular en filas numeradas y columnas etiquetadas. De esta manera la tabla está formada por tres componentes principales: los datos que figuran en las celdas, las filas y las columnas.

**2.** **PROBLEMA**

Se dispone de una cantidad muy amplia de datos sin organizar. Para resolver esta problemática, se ha realizado un mini motor de Base de Datos que organiza dichos datos en Tablas. El proyecto hace un extenso uso de árboles AVL y de mapas para su correcto desarrollo.

**3.** **OBJETIVOS**

* Aplicar los conocimientos adquiridos en el curso de Algoritmos y Estructuras de Datos; en especial aquellos conocimientos referentes a: Programación Orientada a Objetos, diseño de tipos de datos abstractos y plantillas.
* Crear un mini motor de Base de Datos operado por medio de una serie de tablas que han sido construidas haciendo uso de una biblioteca de clases implementadas usando el lenguaje C++.

**4.** **ALCANCE DEL PROYECTO**

El producto final basa toda su funcionalidad en la biblioteca de clases que hemos creado. A continuación, se detallan todas las acciones disponibles:

* Creación de Tablas e Inserción de columnas.
* Inserción de datos.
* Indexado de datos por columnas.
* Selección de datos por columnas (genera nueva Tabla).
* Filtrado de datos por columnas (Mayor, Menor, Igual a, Inicia con, Finaliza con, Está contenido en, No Está contenido en).
* Ordenamiento de datos por columnas.
* Exportación de datos a archivos planos con diferente formato.

Además, para que el producto sea lo más versátil posible, ha sido diseñado haciendo uso de tipos de datos abstractos, plantillas y lambdas (para el filtrado). Asimismo, para mayor facilidad y rapidez de uso por parte del usuario, hemos optado por usar la consola como interfaz.

**5.** **MARCO CONCEPTUAL**

* Programación Orientada a Objetos (POO): es un paradigma de programación que usa objetos y sus interacciones con el objetivo de diseñar aplicaciones y programas informáticos. Entre las técnicas más usadas destacan la, herencia, abstracción, polimorfismo y encapsulamiento.

* Ordenamiento mediante árboles: es un algoritmo de ordenamiento, cuyo principio consiste en construir poco a poco un árbol binario introduciendo cada uno de los elementos y ordenarlos. Después, se obtiene una lista de los elementos ordenados.

* Archivos: Un archivo o fichero informático es un conjunto de bits que se almacenan en un dispositivo. Los archivos se pueden identificar ya que constan de un nombre y la descripción de la carpeta o directorio que lo contiene. Se hace uso de esta terminología ya que son los equivalentes digitales de los archivos escritos del entorno de oficina convencional.

**6.** **DIAGRAMA DE CLASES DE ENTIDADES PRINCIPALES**

Captura de pantalla con letras y números

Descripción generada automáticamente

**7.** **ASIGNACIÓN DE RECURSOS**

Para el presente proyecto, se han dividido las tareas según las habilidades de cada uno de los integrantes. La investigación necesaria para escoger las estructuras de datos, así como la biblioteca de clases a desarrollar ha sido una tarea conjunta. Lo mismo ha ocurrido a la hora de plantear el diseño de la interfaz. Asimismo, la implementación del código ha sido desarrollada en mayor medida por el señor Gray, mientras que el sustento por escrito ha sido elaborado por el señor Orive. No obstante, ambas partes revisan tanto el presente informe como el código fuente para asegurar que todo estuviese correcto. Para información más detallada visite el cronograma que se encuentra a continuación.

**8.** **CRONOGRAMA DE TRABAJO**

URL al cronograma: <https://tinyurl.com/y9o7f9u2>

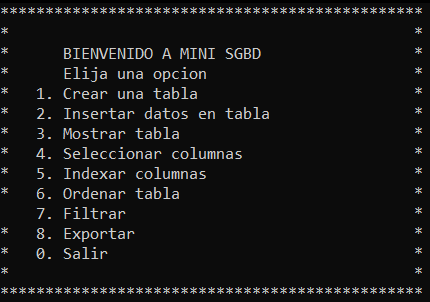
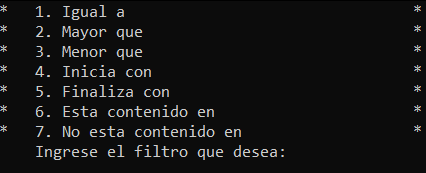
**9. DEFINIR REQUISITOS**

|  |  |
| --- | --- |
| Requerimiento | Descripción |
| RF01 | Se requiere una operación de creación para generar la tabla con las columnas deseadas. |
| RF02 | Se requiere una operación de inserción para colocar los datos en el campo deseado. |
| RF03 | Se requiere una operación de indexado para poder ordenar y filtrar los datos según los criterios: mayor, menor, igual a, inicia y, finaliza con, está contenido en, no está contenido en. |
| RNF01 | Debe construirse el proyecto mediante Programación Orientada a Objetos (POO). |
| RNF02 | Debe realizarse el proyecto haciendo uso del lenguaje C++. |
| RNF03 | Los datos deben exportarse en formato .txt |

**10. DISEÑO DE INTERFAZ DE USUARIO**

Hemos optado por el desarrollo en entorno consola.

En primer lugar, el usuario visualiza el menú, donde podrá escoger la opción deseada. Para exportar la tabla principal a un archivo de texto se escoge la opción 8. Sin embargo, dado que las opciones 4, 6 y 7 generan nuevas tablas también incluyen un pequeño apartado donde se pregunta al usuario si desea exportar la tabla.

Menú Principal Menú de filtros

**11. TIPOS DE DATOS ABSTRACTOS**

Un Tipo de Dato Abstracto (TDA) consiste en una estructura algebraica compuesta por un conjunto de objetos abstractos cuyo propósito es modelar elementos del mundo real y una serie de operaciones para poder ser manipulados. Estas estructuras son cerradas y autosuficientes, por lo que no requieren de un contexto específico para ser empleadas en un programa. De esta manera se garantiza la reutilización del software.

Un TDA está compuesto por:

* Atributos (tipos de datos, identificadores, etc.)
* Funciones (que definen las operaciones necesarias para manipular los atributos

La forma de operar de manera cerrada los atributos y funciones dentro de un TAD genera un encapsulamiento.

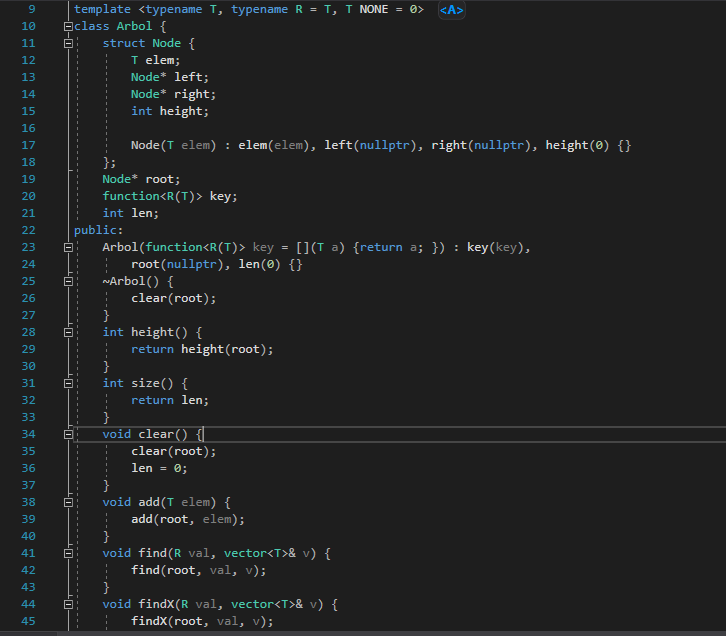
Para implementar un TDA primero se diseña las estructuras de datos que van a representar cada objeto abstracto. Acto seguido, se desarrolla una función, por cada operación del TDA, que simule el comportamiento del objeto abstracto, sobre las estructuras de datos seleccionadas.  
  
Entre los TDA's destacan las listas, pilas y colas, los árboles binarios de búsqueda y los grafos.

* Listas: son secuencias de elementos almacenados en una lista encadenada. En comparación con los vectores, son más rápidas en las operaciones de inserción y borrado, pero demoran más tiempo en acceder aleatoriamente. Además, poseen los [métodos](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Librer%C3%ADa_Est%C3%A1ndar_de_Plantillas/Listas#Lista_de_m%C3%A9todos) necesarios para insertar y borrar elementos al inicio, al final o en un punto específico de la lista.
* Pilas: son listas ordinales en las que el acceso a sus elementos es de tipo LIFO (Last In First Out, último en entrar, primero en salir). Para el manejo de los datos se utilizan dos operaciones: apilar (push), que coloca un elemento en la pila, y retirar (pop) que extrae el último elemento apilado.
* Colas: son listas ordinales en las que el acceso a sus elementos es de tipo FIFO (First In First Out, primero en entrar, primero en salir). Para el manejo de los datos se utilizan dos operaciones: añadir un elemento al final de la cola y eliminar el primer elemento de la cola.
* Árboles Binarios de Búsqueda: un árbol es una [estructura de datos](http://casicodigo.blogspot.com/search/label/Estructura%20de%20Datos) no lineal, ya que cada elemento apunta a uno o varios elementos pero con más de un camino a elegir. El elemento que apunta a otro es llamado padre, mientras que el elemento apuntado se conoce como hijo. En un árbol, todos los nodos poseen un padre salvo que dicho nodo se trate de la raíz. Destaca su recursividad ya que un árbol está formado por subárboles. La característica de un árbol binario es que cada elemento apunta como máximo a otros dos elementos (hijo izquierdo e hijo derecho). Asimismo, si se trata de un árbol binario de búsqueda, para todo elemento, aquellos que sean mayores a él se ubican en la rama derecha y en caso contrario se colocan en la rama izquierda. Dentro de los árboles binarios de búsqueda destacan los árboles AVL, cuya particularidad es su orden de complejidad O(log n) en todo momento. Esto se consigue equilibrando el árbol, de manera que, para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Cuando se realizan las operaciones de inserción y borrado, es necesario realizar una serie de rotaciones en los nodos para evitar que el árbol se desequilibre.
* Grafos: consiste en un conjunto de nodos (vértices) y un conjunto de arcos (aristas) que establecen relaciones entre ellos. Para implementar un grafo puede utilizarse una matriz de adyacencia o una lista de adyacencia. Por un lado, en una matriz de adyacencia se asocia cada fila y cada columna a cada nodo del grafo, siendo los elementos de la matriz la relación entre los mismos, tomando los valores de 1 si existe la arista o 0 de no ser el caso. Por otro lado, en una lista de adyacencia se asocia cada nodo del grafo a una lista que contiene todos aquellos elementos que son adyacentes a él.

En el presente proyecto, haremos uso de árboles AVL.

**12. SELECCIÓN E IMPLEMENTACIÓN DE ESTRUCTURAS DE DATOS**

En esta ocasión optamos por un árbol AVL, puesto que es un árbol binario de búsqueda que de ser necesario equilibrará cada vez que se realice una inserción. La razón de esta elección se debe principalmente al tiempo de complejidad. En el peor de los casos, un árbol binario de búsqueda tiene un tiempo O(n), mientras que en un árbol AVL es O(log n). De esta manera se logra acceder a los datos en los nodos con mayor rapidez.

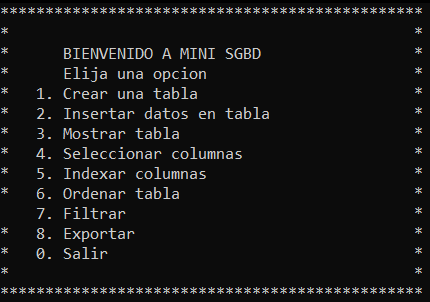
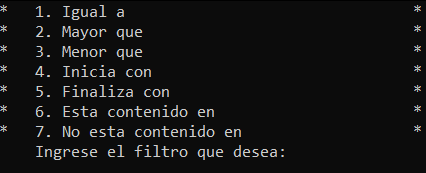


**13. DIAGRAMA DE CLASES DE SEGUNDO NIVEL**

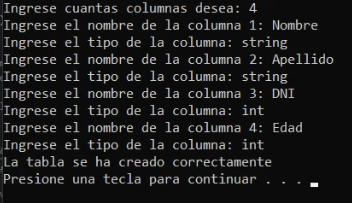
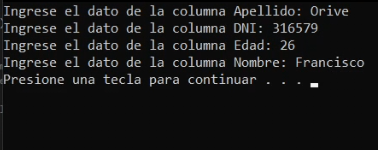
**Captura de pantalla de un celular con letras

Descripción generada automáticamente**

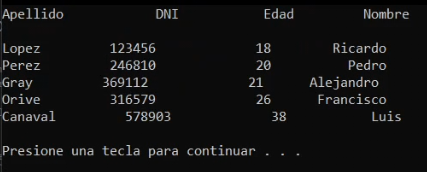
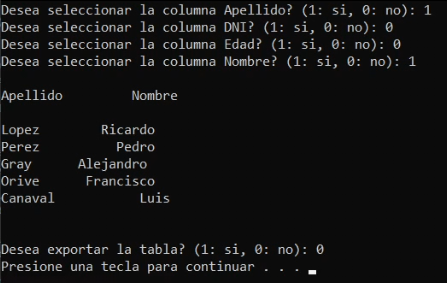
**14. IMPLEMENTACIÓN DE LAS FUNCIONALIDADES**

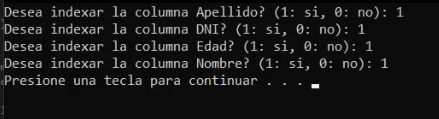
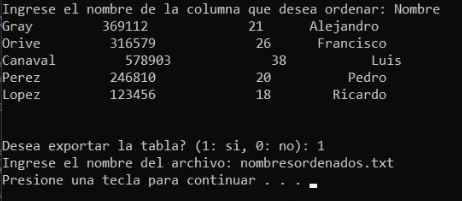
Menú Principal Menú de filtros

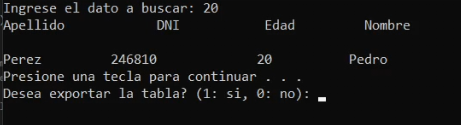
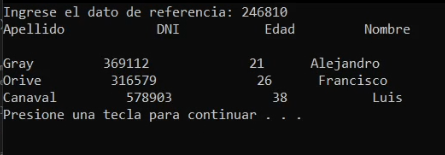
Opción 1 (Crear una tabla) Opción 2 (Insertar datos en tabla)

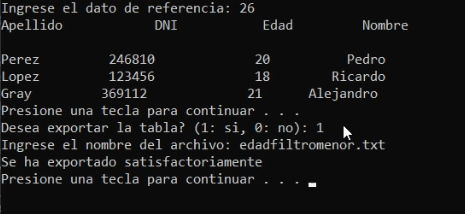
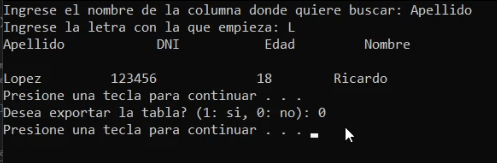
Opción 3 (Mostrar tabla) tras 5 inserciones de datos Opción 4 (Seleccionar datos)

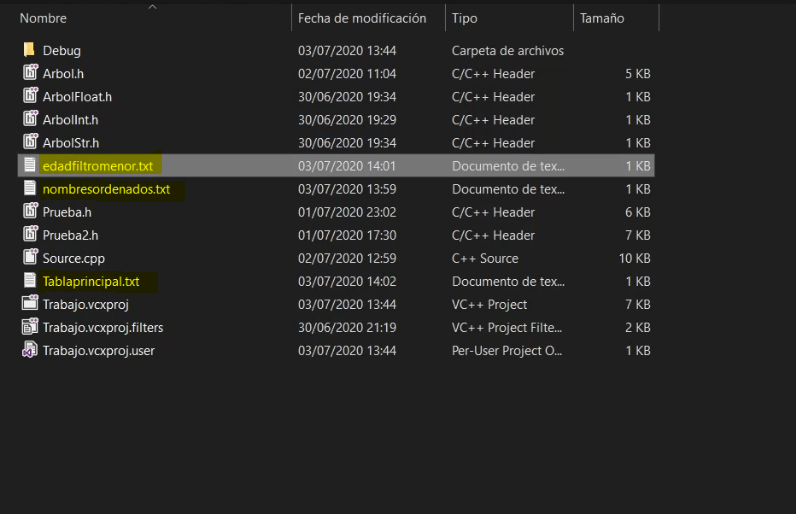
Opción 5 (Indexar columnas) Opción 6 (Ordenar datos) aplicada a col. Nombre (4)

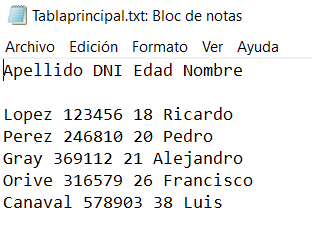
Opción 7 (Filtro Igual a) Opción 7 (Filtro Mayor que)

Opción 7 (Filtro Menor que) Opción 7 (Filtro Inicia con)



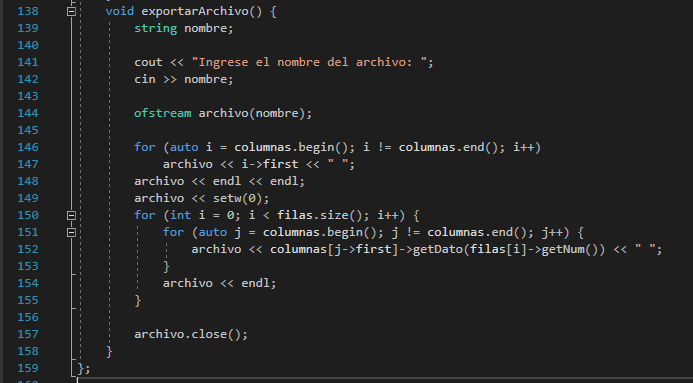
Archivos creados en el directorio correspondiente (sombreados en amarillo).



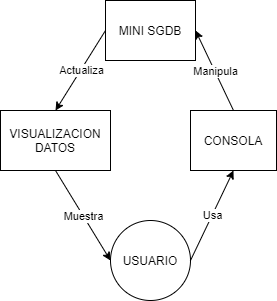
Contenido de Tablaprincipal.txt

**15. DISEÑO DE ARCHIVOS**

Para poder leer o escribir en archivos hemos incluido la librería fstream. En esta ocasión hemos optado por exportar los datos a un archivo en formato .txt. Aunque otra posibilidad que se contemplo fue en formato .cvs para su visualización en programas como Excel.



**16. ARQUITECTURA DE LA APLICACIÓN**



**17. CONCLUSIONES**

Este proyecto ha servido para reforzar los conocimientos adquiridos a lo largo del curso. Con el concepto de archivos hemos aprendido todo lo necesario para escribir o modificar los datos contenidos en cualquier archivo. Asimismo, hemos visto la lectura y escritura de varios tipos de archivos.

De igual manera, el conocimiento sobre arboles fue esencial a la hora de escoger el árbol para este proyecto. Finalmente, optamos por el árbol AVL ya que, a diferencia de un árbol binario de búsqueda, este se encuentra balanceado y al mantener su complejidad en O (log n) permite realizar los procedimientos con mayor rapidez.

Crear este motor de búsqueda con todas sus funciones, nos ha ayudado a mejorar los conocimientos previos que teníamos de cursos anteriores sobre el manejo y uso de la memoria, gracias a la implementación de estructuras de datos como el árbol AVL. Todo ello ha permitido que esta aplicación funcione lo más optimo posible.

Dicho motor de búsqueda ha sido uno de los proyectos con mayor funcionalidad que hemos desarrollado a lo largo de nuestra vida académica, ya que puede aplicarse en múltiples campos: registrar pacientes de un hospital, base de datos de algún videojuego, como herramienta de consulta de inventario de un almacén, etc.

**18. REFERENCIAS**

* *Map In C++* *| STL C++.* *(2019, 2 de Octubre).* [Video]. YouTube. <https://www.youtube.com/watch?v=nPSDR5nZzHA>
* USF Computer Science Department. (n.d.). *AVL Tree Visualzation*. Myusf.Usfca.Edu. Accedido 22 de Junio, 2020, a través de <https://www.cs.usfca.edu/%7Egalles/visualization/AVLtree.html>
* Canaval Sánchez, L. M. (n.d.). *Ejemplos de Algoritmos y Estructura de Datos*. GitHub. Accedido 24 de Junio, 2020, a través de <https://github.com/pcsilcan/aed>
* *map::erase - C++ Reference*. (n.d.). Cplusplus.Com. Accedido 23 de Junio, 2020, a través de <http://www.cplusplus.com/reference/map/map/erase/>
* Khaled, A. (n.d.). *Sorting Algorithms Animations*. Toptal®. Accedido 28 de Junio, 2020, a través de <https://www.toptal.com/developers/sorting-algorithms>