Gestión eficiente de bases de datos de personas.

Jhoan David Fiat, Julián Andrés Rivera Carrillo y David Jhun Kim.

Universidad ICESI.

Algoritmos y estructuras de datos.

Juan Manuel Reyes y Andrés Aristizábal.

07/11/2020

Contenido.

1. Contexto del problema.
2. Identificación de necesidades y síntomas.
3. Definición del problema.
4. Etapa #1 Requerimientos funcionales y no funcionales.
5. Etapa #2 Recopilación de información.
6. Etapa #3 Búsqueda de soluciones creativas.
7. Etapa #4 Transición de ideas a diseños preliminares.
8. Etapa #5 Evaluación y selección de la mejor solución.

Contexto del problema.

Los profesores del curso de algoritmos han recomendado a un equipo de estudiantes con un buen desempeño a diferentes profesores de la facultad de Ingeniería para que estos puedan trabajar junto a ellos en sus proyectos de investigación. Este equipo fue contratado para una monitoria en un proyecto de investigación interna de la universidad ICESI formando parte del equipo VIP de simulación.

A este equipo se le ha asignado el subproyecto que consiste en el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD (Create, Read, Update and Delete) sobre una base de datos del continente americano y la simulación de registros del total de personas que viven en él. Además, estos registros de las personas deben ser generados utilizando una serie de archivos cuya combinación de los datos dará un total aproximado de la población total del continente.

Identificación de necesidades y síntomas.

- El equipo VIP de simulación de la Universidad ICESI no cuenta con un programa para manejar grandes cantidades de información.
- La solución al problema debe ser capaz de generar información a partir de archivos externos.
- La solución al problema debe ser eficiente.
- La solución al problema debe hacer persistir la información.

Definición del problema.

El equipo VIP de simulación de la Universidad ICESI necesita un programa que les permita gestionar de manera eficiente grandes cantidades de datos de personas que viven en el continente americano. Este programa debe contener una estructura de datos eficiente que permita un manejo fluido de los datos por medio de operaciones de tipo CRUD.

Etapa #1. Requerimientos funcionales y no funcionales.

RF#1: Agregar una persona. Permite registrar a una nueva persona en el programa. Las entradas requeridas para esta operación son: un nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía. La salida para esta operación es el registro exitoso de la persona si no se encuentra ya registrada. De lo contrario, esta no podrá ser registrada debido a su existencia en el programa.

RF#2: Buscar una persona. Permite buscar una persona generada o agregada en el programa. Las entradas requeridas para esta operación pueden ser: un nombre, apellido, nombre completo o código. La salida para esta operación es la visualización exitosa de

los datos de la persona que estaba siendo buscada. En caso de no ser encontrada o de no existir, no se visualizará ninguna información en pantalla.

RF#3: Actualizar datos de una persona. Permite actualizar los datos de una persona, una vez que ha sido encontrada en el programa. Las entradas requeridas para esta operación son: una persona buscada. La salida para esta operación concluye con la modificación de los datos de la persona. En caso de no ser encontrada o de no existir, no se realizará cambio alguno.

RF#4: Eliminar una persona. Permite eliminar de manera permanente a una persona generada o agregada en el programa. Las entradas requeridas para esta operación son: una persona buscada. La salida para esta operación concluye con la eliminación de esta persona. En caso de no ser encontrada o de no existir, no se realizará cambio alguno.

RF#5: Generar un registro. Permite generar el registro de una persona aleatoria en el programa. Las estradas requeridas para esta operación son: ninguna. La salida para esta operación es la generación de un nuevo registro de la persona.

RF#6: Mostrar tiempo de los registros. Muestra el tiempo total que tarda el programa en generar una cantidad de registros digitada por el usuario. Las entradas requeridas para esta operación son: ninguna. La salida para esta operación es la visualización en pantalla del tiempo total que tardo la operación.

RF#7: Interfaz grafica de usuario. El programa deberá contar con una interfaz gráfica para facilitar la interacción de las funcionalidades con el usuario final. Las entradas requeridas para esta operación son: ninguna. La salida para esta operación será la visualización de la interfaz gráfica de usuario.

RF#8: Componente menú. La interfaz gráfica del programa deberá contar con un botón de tipo “menú” que permita visualizar diferentes elementos de acuerdo con las opciones disponibles. Las entradas para esta operación son: ninguna. La salida para esta operación será la visualización del botón menú en la interfaz gráfica.

RF#9: Mostrar coincidencias. Muestra el número total de elementos que coinciden con los prefijos digitados por el usuario. Las entradas para esta operación son: una cadena. La salida para esta operación es un número entero que puede variar de acuerdo con las coincidencias que se encuentren.

RF#10: Mostrar lista de coincidencias. Muestra una lista de todos los elementos que coinciden con la búsqueda del usuario. Las entradas requeridas para esta operación son: una cadena. La salida para esta operación será la visualización de la lista de los elementos que coinciden con la búsqueda. En caso de no hallarse coincidencias, no se mostrará la lista.

RNF#1: Todos los registros del programa deben ser almacenados de manera persistente.

RNF#2: Debe haber un directorio “/data” donde se guardarán los archivos con los que se generarán los registros de las personas.

RNF#3: Los nombres y apellidos de las personas en los registros generados deben ser tomados de un archivo almacenado en el directorio “/data”.

RNF#4: La fecha de nacimiento de las personas en los registros generados puede ser aleatoria, pero, debe basarse en una distribución de edad de estados unidos.

RNF#5: La estatura de las personas en los registros puede ser generada de manera aleatoria en intervalos que tengan sentido.

RNF#6: La nacionalidad de las personas en los registros debe respetar los porcentajes poblacionales de cada país en el continente americano.

RNF#7: Se debe mostrar una barra de progreso en caso de que la operación para generar los registros tarde más de 1 segundo.

Etapa #2 Recopilación de información.

Con el objetivo de tener una mayor claridad en los conceptos que faciliten la solución al problema, se realizó una recopilación de información de algunas estructuras de datos y conceptos usados en programación y el desarrollo.

CRUD: El concepto CRUD está estrechamente vinculado a la gestión de datos digitales. CRUD hace referencia a un acrónimo en el que se reúnen las primeras letras de las cuatro operaciones fundamentales de aplicaciones persistentes en sistemas de bases de datos:

- Create (Crear registros)
- Read bzw. Retrieve (Leer registros)
- Update (Actualizar registros)
- Delete bzw. Destroy (Borrar registros)

Árbol binario de búsqueda: Un árbol binario de búsqueda también llamado *BST* (acrónimo del inglés *Binary Search Tree*) es un tipo particular de árbol binario que presenta una estructura de datos en forma de árbol usada en informática.

Árbol AVL: El árbol AVL es un árbol de búsqueda binaria (BST) auto equilibrado donde la diferencia entre las alturas de los subárboles izquierdo y derecho no puede ser más de una para todos los nodos.

Árbol rojo-negro: Un árbol rojo-negro es una especie de árbol de búsqueda binaria auto equilibrado donde cada nodo tiene un bit extra, y ese bit a menudo se interpreta como el color (rojo o negro). Estos colores se utilizan para garantizar que el árbol permanezca equilibrado durante las inserciones y eliminaciones. Aunque el equilibrio del árbol no es perfecto, es lo suficientemente bueno como para reducir el tiempo de búsqueda y mantenerlo alrededor del tiempo O (log n), donde n es el número total de elementos del árbol.

Debe tenerse en cuenta que, dado que cada nodo requiere solo 1 bit de espacio para almacenar la información de color, estos tipos de árboles muestran una huella de memoria idéntica al árbol de búsqueda binario clásico (sin colorear).

TRIE: Tries es un árbol que almacena cuerdas. El número máximo de hijos de un nodo es igual al tamaño del alfabeto. Trie admite operaciones de búsqueda, inserción y eliminación en tiempo O (L) donde L es la longitud de la clave.

Fuente:

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/

https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda

https://www.geeksforgeeks.org/binary-search-tree-data-structure/#basic

https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

https://github.com/williamfiset/Algorithms/tree/master/slides/datastructures

https://www.geeksforgeeks.org/advantages-trie-data-structure/?ref=lbp

https://www.youtube.com/watch?v=wjWSfRfiiIs

https://gist.github.com/jeanmw/9ce88093deafca3cf1ac5daed65d47eb

Etapa #3 Búsqueda de soluciones creativas.

Aunque se puede pensar en soluciones propias, decidimos realizar una lista de revisión para analizar la opinión de los miembros del equipo con el objetivo de tener diferentes puntos de vista sobre las posibilidades de diseño e implementación de algunas estructuras de datos. Con este análisis, llegamos a la conclusión de que estas estructuras de datos podrían ser útiles para la solución del problema:

1. Lista enlazada.
2. Árbol AVL.
3. Árbol rojo-negro.
4. TRIE.
5. Lista doblemente enlazada.
6. Arraylist.

Etapa #4 Transición de las ideas a los diseños preliminares.

En este punto, decidimos descartar las estructuras de datos cuyos algoritmos no tenían un tiempo adecuado. Por esto, descartamos las estructuras de datos: lista enlazada, lista

doblemente enlazada y arraylist. La principal razón por la cual decidimos descartar estas estructuras de datos fue por la complejidad temporal que tienen sus algoritmos. Por ejemplo, para la lista enlazada, lista doblemente enlazada y arraylist, sus algoritmos más usados tienen una complejidad temporal en el caso promedio de O(n). Por otro lado, las estructuras de datos árbol AVL y árbol rojo-negro tienen una complejidad temporal de sus algoritmos en el caso promedio de O(log n). En el caso de la estructura de datos TRIE, la complejidad de sus algoritmos puede variar en función de su entrada; O(h), donde h representa la entrada de caracteres.

Etapa #5 Evaluación y selección de la mejor solución.

Para terminar de decidir cual estructura de datos implementar, decidimos realizar diferentes criterios de evaluación. Estos criterios son:

Criterio A. Eficiencia. Los algoritmos utilizados por la estructura de datos son eficientes y poseen de una buena complejidad temporal.

- [2] Bueno
- [1] Regular
- [0] Malo

Criterio B. Dificultad en la implementación. Nivel de dificultad que requiere cada estructura de datos a la hora de ser implementada.

- [2] Alta
- [1] Media
- [0] Baja

Criterio C. Completitud. La estructura de datos brinda una solución completa a todas las necesidades planteadas.

- [2] Solución completa
- [1] Cumple con algunas necesidades
- [0] No cumple con las necesidades

	Criterio A	Criterio B	Criterio C	Total
Árbol AVL	2	1	2	5
Árbol rojo-negro	2	2	2	6
TRIE	1	2	1	4

La estructura de datos ganadora en este caso es la de árbol rojo-negro. Sin embargo, todas las estructuras de datos en la tabla de criterios anterior nos sirven para dar solución a las necesidades del cliente. Además, se implementarán las estructuras de datos de árbol AVL y árbol rojo-negro, con el objetivo de comparar que tan eficiente es la una de

la otra y cuáles son sus diferencias más notorias. Por otro lado, la estructura de datos TRIE se implementará porque da solución a las sugerencias que nacen ante la entrada del usuario.

DIAGRAMA DE CLASES