Método de la ingeniería

Alejandro Arce
Duvan Cuero
Alexander Samaca

Universidad Icesi
Algoritmos y Estructuras de datos
Santiago de Cali
Noviembre del 2020

ETAPA 1: Contexto del problema

Después del excelente rendimiento en sus trabajos académicos previos, sus profesores de algoritmos los han recomendado a diversos profesores de la Facultad de Ingeniería, para que trabajen en sus proyectos de investigación. Su equipo ha sido contratado para una monitoría en un proyecto de investigación interna de la Universidad, como parte del Equipo VIP de Simulación[1].

El sub-proyecto que le ha sido asignado, consiste en el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD (Create, Read, Update y Delete) sobre una base de datos de personas de nuestro continente. La población del continente americano se estima, en 2020, en poco más de mil millones de personas[2], representando cerca del 13% del total mundial. Por tanto, usted debe simular la creación de (generar) un número similar de registros de personas, para este continente, con los siguientes datos: código (autogenerado), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía.

Definición del problema

El sub-proyecto requiere gestionar eficientemente las operaciones CRUD en una base de datos de personas de nuestro continente de más o menos mil millones de personas.

Especificación de requerimientos

- **Requerimientos funcionales**

  El sistema está en la capacidad de:

  - ➢ Generar un registro de personas del continente americano con los siguientes datos: código (generado automáticamente), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografia. El programa cuenta con un campo de texto donde se pone el número de registros que se desea generar, este número de registros tiene un máximo que es de más o menos mil millones que corresponde al número de personas del continente americano. Además se debe mostrar una barra de progreso en el momento de generar los registros

  - ➢ Agregar un nuevo registro de una persona con los datos : código (generado automáticamente), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografia. Y cuenta con una opción guardar de manera persistente

  - ➢ Actualizar la información de una persona en todos los campos editables menos el código que es autogenerado.

1

2

- ➢ Buscar una persona en la base de datos dado uno de los siguientes criterios: nombre, apellido, nombre completo y código. Para el nombre y el nombre completo se despliega una lista emergente con maximo 100 busquedas que cuenten con los primeros caracteres en orden al momento de digitar.

- ➢ Eliminar un registro de una persona en la base de datos.

- ➢ De guardar todos los cambios en el mismo programa, es decir, si se hace algún cambio en la base de datos ya sea actualizar información, agregar personas o eliminar, esta base de datos se guardará aunque se cierre el programa y se vuelva abrir.

Requerimientos no funcionales

- ➢ Al momento de almacenar todos los registros de personas se debe utilizar árboles binarios de búsquedas autobalanceados serializados.

- ➢ Para generar los nombres completos se debe tomar los nombres de este dataset dataset de nombres de data.world. Para los apellidos se debe usar este dataset dataset de apellidos de data.world.

- ➢ Para generar la fecha de nacimiento suponemos que la distribución de edad de toda América es igual a esta distribución de edad de Estados Unidos**.**

- ➢ Para generar la estatura de los registros debe ser de manera aleatoria en un rango coherente.

- ➢ Para generar la nacionalidad de las personas debe seguir los porcentajes relativos de población que corresponde a cada país, se debe tomar estos datos de población por países como base para generar las nacionalidades. Se debe tener en cuenta que si existe una diferencia en el total, esta diferencia ya sea negativa o positiva se agregara al país con mayor población.

- ➢ La barra de progreso al momento de generar los registros demora más de 1 segundo se debe mostrar el tiempo que demoró cada operación.

ETAPA 2 : *Recopilación de información:*

Para resolver este problema lo fundamental es reunir toda la información sobre operaciones CRUD y bases de datos. Como vamos a usar nuestra propia estructura de datos un árbol autobalanceado de búsqueda(BST AVL) también se definirán estos conceptos.

CRUD

En informática, CRUD es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

Base De Datos

Una **base de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta (https://es.wikipedia.org/wiki/Base_de_datos)

¿Qué es un Árbol como estructura de datos?

Es una estructura de datos de tipo árbol que permite la recuperación de información (de ahí su nombre del inglés re**TRIE**val). La información almacenada en un trie es un conjunto de claves, donde una clave es una secuencia de símbolos pertenecientes a un alfabeto. Las claves son almacenadas en las hojas del árbol y los nodos internos son pasarelas para guiar la búsqueda. El árbol se estructura de forma que cada letra de la clave se sitúa en un nodo de forma que los hijos de un nodo representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre. Por tanto la búsqueda en un trie se hace de forma similar a como se hacen las búsquedas en un diccionario:

> Se empieza en la raíz del árbol. Si el símbolo que estamos buscando es A entonces la búsqueda continúa en el subárbol asociado al símbolo A que cuelga de la raíz. Se sigue de forma análoga hasta llegar al nodo hoja. Entonces se compara la cadena asociada a el nodo hoja y si coincide con la cadena de búsqueda entonces la búsqueda ha terminado en éxito, si no entonces el elemento no se encuentra en el árbol.

Por eficiencia se suelen eliminar los nodos intermedios que sólo tienen un hijo, es decir, si un nodo intermedio tiene sólo un hijo con cierto carácter entonces el nodo hijo será el nodo hoja que contiene directamente la clave completa.

Es muy útil para conseguir búsquedas eficientes en repositorios de datos muy voluminosos. La forma en la que se almacena la información permite hacer búsquedas eficientes de cadenas que comparten prefijos.

(https://es.wikipedia.org/wiki/Trie)

¿Entonces que es un Árbol Autobalanceado?

En ciencias de la computación, un **árbol binario de búsqueda auto-balanceable** o **equilibrado** es un árbol binario de búsqueda que intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento, automáticamente. Esto es importante, ya que muchas operaciones en un árbol de búsqueda binaria tardan un tiempo proporcional a la altura del árbol, y los árboles binarios de búsqueda ordinarios pueden tomar

alturas muy grandes en situaciones normales, como cuando las claves son insertadas en orden. Mantener baja la altura se consigue habitualmente realizando transformaciones en el árbol, como la rotación de árboles, en momentos clave.

Tiempos para varias operaciones en términos del número de nodos en el árbol n:

Operación	**Tiempo en cota superior asintótica**
Búsqueda	O(log n)
Inserción	O(log n)
Eliminación	O(log n)
Iteración en orden	O(n)

Para algunas implementaciones estos tiempos son el peor caso, mientras que para otras están amortizados.

Estructuras de datos populares que implementan este tipo de árbol:

- Árbol AVL
- Árbol rojo-negro

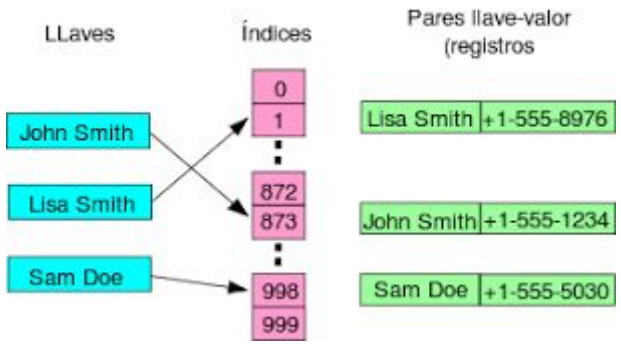
(https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda_auto-balanceable)

Tabla Hash

Una **tabla hash**, **matriz asociativa**, **hashing**, **mapa hash**, **tabla de dispersión** o **tabla fragmentada** es una estructura de datos que asocia *llaves* o *claves* con *valores*. La operación principal que soporta de manera eficiente es la *búsqueda*: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una **función hash** en un *hash*, un número que identifica la posición (*casilla* o *cubeta*) donde la tabla hash localiza el valor deseado.[1]

Las **tablas hash** se suelen implementar sobre *vectores* de una dimensión, aunque se pueden hacer implementaciones multi-dimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1),[2] sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(*n*), es decir, en función del número de elementos.

(https://es.wikipedia.org/wiki/Tabla_hash#:~:text=Las%20tablas%20hash%20almacenan%20la,est%C3%A1%20ordenada%20en%20todo%20momento.).

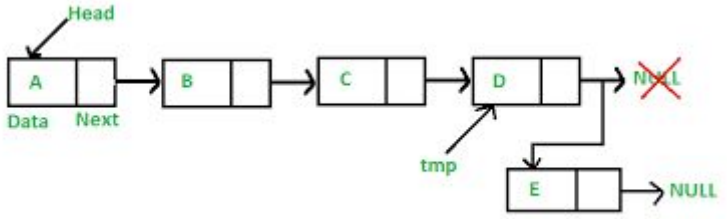


Linked List

En ciencias de la computación, una **lista enlazada** es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

Una lista enlazada es un tipo de dato autorreferenciado porque contienen un puntero o enlace (en inglés *link*, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

(https://es.wikipedia.org/wiki/Lista_enlazada)

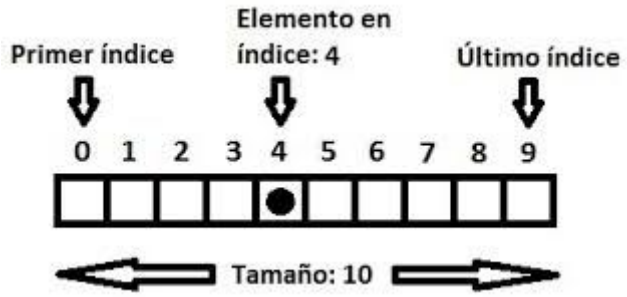


Array

Un array (arreglo) en Java es una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar luego durante la ejecución del programa, como sí puede hacerse en otros lenguajes.

Cuando creamos un array de nombre “a” y de dimensión “n” (int[ ] a = new int[n]) estamos creando n variables que son a[0], a[1], a[2], ..., a[n-1]. Los arrays se numeran desde el elemento cero, que sería el primer elemento, hasta el n-1 que sería el último elemento. Es decir, si tenemos un array de 5 elementos, el primer elemento sería el cero y el último elemento sería el 4. Esto conviene tenerlo en cuenta porque puede dar lugar a alguna confusión. Disponer de un valor con índice cero puede ser de utilidad en situaciones como considerar cada variable asociada a una hora del día, empezando a contar desde la hora

cero hasta la 23 (total de 24 horas), cosa que es habitual en algunos países. En lugar de 1, 2, 3, ..., 24 estaríamos usando 0, 1, 2, ..., 23.



(https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=168:repaso-arrays-o-arreglos-unidimensionales-en-java-tipos-de-inicializacion-ejemplos-de-codigo-cu00903c&catid=58&Itemid=180#:~:text=Un%20array%20(arreglo)%20en%20Java,puede%20hacerse%20en%20otros%20lenguajes.)

Árbol Binario de Búsqueda

Un árbol binario de búsqueda (abb) se basa en la propiedad de que las claves que son menores que el padre se encuentran en el subárbol izquierdo, y las claves que son mayores que el padre se encuentran en el subárbol derecho. Llamaremos esto la **propiedad abb**. La Figura 1 ilustra esta propiedad de un árbol binario de búsqueda, mostrando las claves sin ningún valor asociado. Observe que la propiedad es válida para cada padre e hijo. Todas las claves del subárbol izquierdo son menores que la clave de la raíz. Todas las claves en el subárbol derecho son mayores que la raíz.

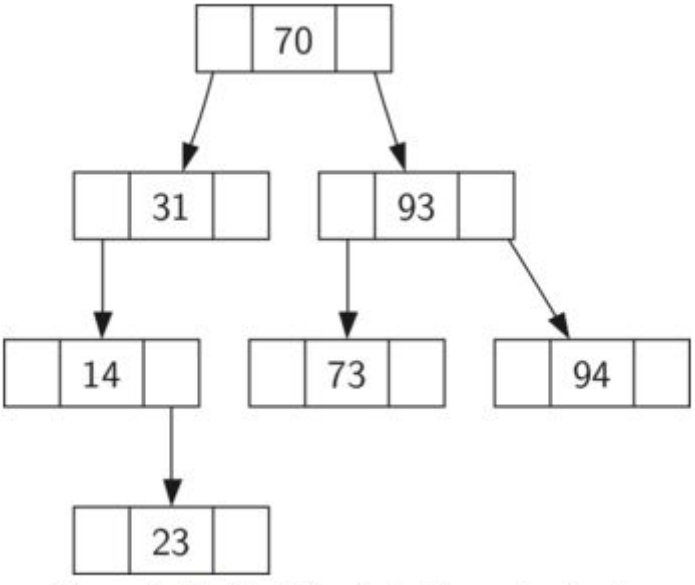


Figura 1: Un árbol binario de búsqueda simple

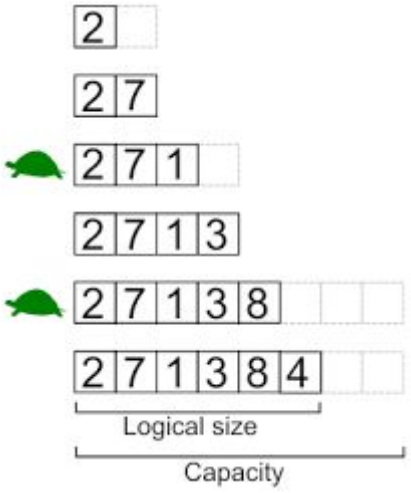
Array List

En programación, un **arreglo dinámico** o **array dinámico**, también llamado inapropiadamente matriz dinámica o tabla dinámica, es un array de elementos que crece o mengua dinámicamente conforme los elementos se agregan o se eliminan. Se suministra como librerías estándar en muchos lenguajes modernos de programación.

Un array dinámico no es lo mismo que un array asignado dinámicamente, que es un array de tamaño fijo, pero cuyo tamaño se fija cuando se asigna por primera vez.[1]

Los arrays dinámicos tienen un rendimiento similar a una array estático, con la adición de nuevas operaciones para añadir y eliminar elementos al final:

- Al obtener o establecer el valor de un índice en particular: Θ(1) (tiempo constante)
- Recorrer sus elementos en orden: Θ(*n*) (tiempo lineal, buen uso del caché de lectura)
- Insertar o eliminar un elemento no al final del array: Θ(*n*) (tiempo lineal)
- Insertar o eliminar un elemento al final del array: Θ(1) (tiempo constante amortizado)
- Espacio desperdiciado: Θ(*n*)[5]



(https://es.wikipedia.org/wiki/Array_din%C3%A1mico)

ETAPA 3: Búsqueda de soluciones creativas.

La gran problemática que evidenciamos al momento de crear una base de datos con más de mil millones de registros es qué estructura es la indicada para poder almacenar esa cantidad adecuada de manera eficiente de modo que cuando quiera realizar una búsqueda el software no tarde demasiado en encontrar el registro que quiero buscar.

Para esta ardua tarea se sugieren las siguientes alternativas que pueden ser candidatos para lograr almacenar esa cantidad de registros.

Alternativa 1:

Hash Table

Hash table o tabla hash es una buena alternativa al problema de almacenar grandes cantidades de datos, ya que tiene la capacidad de acceder a los datos de manera rápida esto es debido a la función hash que maneja cada dato.

Alternativa 2:

Linked List

Las listas enlazadas pueden ser una opción viable ya que como se pudo ver en la definición de las listas enlazadas cuando se recorren los nodos de la lista este orden es diferente al orden en la memoria o en el disco por lo que pueden ser eficientes en cuanto a memoria al momento de realizar las búsquedas.

Alternativa 3:

Arbol AVL

Los árboles AVL llegan a ser una excelente alternativa, ya que al ser árboles autobalanceados logrando que la altura del árbol sea la mínima, consecutivamente entre menor sea la altura del árbol más rápido se podrá llegar a los registros que se quiera encontrar.

Alternativa 4:

Array

Los arreglos son una alternativa sencilla de utilizar, ya que el propio lenguaje de Java utiliza esta estructura contenedora por lo que hacer uso de ella es de lo más fácil e intuitivo de entender como solo manejaremos un tipo de registro es conveniente tener en cuenta esta alternativa.

Alternativa 5:

Árbol Binario de Búsqueda

Los árboles binarios se destacan por ser muy eficiente al momento de buscar elementos ya que poseen un orden determinado por niveles por lo que en la mayoría de los casos no se necesita recorrer todos los elementos del árbol para encontrar el elemento a buscar.

Alternativa 6:

Array List

Dado que el tamaño de la base de datos puede cambiar dependiendo si queremos añadir o eliminar registros de personas del problema a tratar debemos tener una estructura que no tenga un tamaño predefinido de tal forma que se puedan ir agregando elementos periódicamente, por tanto los arreglos dinámicos pueden llegar a ser una opción viable.

ETAPA 4: Transición de la formulación de ideas a los diseños preliminares

Todas las alternativas anteriores son buenas y pueden ser perfectamente utilizadas para resolver el problema pero como solo podemos usar una debemos evaluar minuciosamente cada opción disponible y escoger así la que nos parezca más adecuada para resolver el problema, por esto hemos decidido descartar las alternativas 2,4 y 6 debido a que nos parecen muy simples como bases de datos y no son tan eficientes.

Alternativa 1:
Una tabla Hash es muy eficiente como diccionario o base de datos debido a su eficiencia para encontrar los datos almacenados en ella sin embargo su defecto es el uso excesivo de memoria que se presenta al usar inmensas bases de datos.

Alternativa 3:
Un Arbol AVL conserva un equilibrio entre eficiencia de uso de memoria y una eficiencia de búsqueda de datos constante, ya que independientemente de su tamaño el tiempo siempre sera el constante.

Alternativa 5:
Un árbol binario de búsqueda es más eficiente que una lista en términos de búsqueda pero su tiempo de búsqueda no es constante y puede variar a diferencia de un árbol de búsqueda autobalanceado.

ETAPA 5. Selección de mejor solución
Ahora seleccionaremos la mejor solución con base en unos criterios que vamos a definir, estos criterios son relevantes para la solución del problema.

Criterio A: Uso de memoria.
- **[1]** Uso de memoria excesivo o poco eficiente
- **[2]** Uso de memoria lineal

Criterio B: Velocidad para encontrar datos.
- **[1]** Poco eficiente

- **[2]** Medianamente eficiente o variable
- **[3]**Altamente eficiente y constante

Criterio C: Elegancia como solución creativa

- **[1]** Muy simple
- **[2]** Medianamente Elegante
- **[3]**Altamente elegante o compleja

Alternativa #	**Criterio A**	**Criterio B**	**Criterio C**	**Puntuación**
Alternativa 1	1	3	3	7
Alternativa 3	2	3	3	8
Alternativa 5	2	2	2	6

Después de evaluar más específicamente cada alternativa llegamos a la conclusión por puntaje que la mejor opción es la ***alternativa 3*** usar un árbol binario de búsqueda como base de datos.