

# Libreta interactiva de contactos

Universidad del Rosario

Algoritmos y estructuras de datos

Carlos Andrés Galán Pérez

**Resumen** –El presente documento busca presentar la aplicabilidad del uso de árboles binarios y tablas hash como estructura eficiente para gestionar una libreta de contactos que debe recolectar y ordenar su información. Además, se hace un análisis comparando la mejora que se tiene por hacer uso de estas estructuras de tipo “map” en vez estructuras lineales. **Key words:** Árbol binario, búsqueda binaria (inorder), Tabla hash, Mapas ordeados.

## I. INTRODUCCION

**Introducción** –Dado un problema que busca la mejor manera para gestionar contactos, se desarrolló de un sistema de gestión de contactos implementado en C++. Este programa proporciona una interfaz interactiva para agregar, buscar, eliminar y visualizar información detallada de contactos almacenados en una libreta, generando una copia de seguridad para almacenar cada registro. Además, integra estructuras de datos como árboles binarios y tablas hash para visualizar ordenadamente la información y maneja mapas ordenados para consultas rápidas.

## II. OBJETIVOS

La implementación de esta libreta se estableció como objetivo general del proyecto, de donde surgen varios objetivos específicos:

1. Creación de una interfaz de inicio buscando la interacción con el usuario. En donde se presenten todas las posibles opciones dentro de la libreta.
2. Agregación de contactos a la libreta, añadiendo los nombres de los contactos en un árbol binario que ingresa cada nombre de forma ordenada (Por el orden del alfabeto).
3. Creación de mapas ordenados que mantengan las relaciones que necesitamos a la hora de consultar información específica de los contactos.
4. Búsqueda de contactos por nombre y en el caso donde haya duplicidad se consulte por número de teléfono.
5. Eliminación de contactos en la libreta usando los métodos proporcionados por las clases árbol y tabla hash. (teniendo en cuenta cuando hay contactos repetidos)

6. Visualización de contactos presentados en dos formas, en primer lugar, en una lista ordenada y en segundo lugar en una tabla hash que muestre los contactos agrupados según su inicial (cada posición en la tabla hash representa una letra en el abecedario).
7. Desarrollar una copia de seguridad al estilo de una base de datos que guarde la información de cada ejecución en un archivo de texto (txt) para no perder el registro de la libreta y además se encuentre de forma ordenada.

## III. PROBLEMA

El usuario está solicitando una herramienta en la que pueda ser capaz de administrar y personalizar su libreta de contactos como lo desee. Además, que toda la información la tenga de forma inmediata y pueda encontrar copias de seguridades de sus movimientos en la plataforma.

## IV. ALCANCE

Los objetivos de este proyecto son el alcance esperado, se busca que el usuario logré estar satisfecho con su producto y no tenga forma de presentar errores que le impidan disfrutar su producto. Además, se aplican varios conceptos de la materia teniendo en cuenta las limitaciones del lenguaje, que por ejemplo no permiten un desarrollo de una interfaz tan atractiva o de una conexión a una base de datos por ejemplo con postgres.

## V. METODOLOGÍA

### 1. Análisis

#### Entradas:

#### Interfaz de inicio:

El usuario elige entre varias opciones proporcionadas por el menú, como agregar contactos, buscar, eliminar, mostrar la lista

completa, realizar copias de seguridad, entre otros.

#### Creación de información de contactos:

- Se crea una estructura contacto que tiene los atributos de: nombre, número de teléfono, dirección, redes sociales (Instagram, Github) y deben ser ingresados por el usuario al agregar un contacto, casi todos los atributos son de tipo string para facilitar su ingreso.

```
struct contacto{
    string nombre;
    string telefono;
    string redes_sociales;
    string Instagram;
    string Github;
    string direccion;
    string calle;
    string numero;
    string indicativo; //(-)
    int veces_repetido;
    int veces_visitado; //# de
};
```

#### Se incluyen 3 mapas:

- 1- Para cuando a la hora de buscar un nombre, no se encuentre repetido,
- 2- Para cuando se trata de buscar un nombre que esta repetido y se debe usar el teléfono para seguir consultando
- 3- Para mirar cuantas veces esta repetido un nombre y así conocer en cual caso de los anteriores dos nos encontramos.

```
map<string,contacto>datos_no_rep; //si solo
map<string,contacto>datos_rep2; // se nece
map<string,int>datos_rep; // mirar cuantas
```

- Inicialización de hash como una pareja de strings (por defecto la clave del hash es de tipo string y en el main se determina el tipo de dato del valor). Además, se inicializa un btree de tipo string y se define algunas variables que nos van a permitir navegar entre las opciones y salir del bucle while del programa.

```
Btree<string> mi_arbol;
HashMap<string> hash;
int exit=1;
int opciones;
int salir=1;
```

#### Procesos:

#### Agregar Contacto:

1. Una vez recopilada la información ingresada por el usuario, se verifica si el nombre está repetido y se actualiza la cantidad de repeticiones que se encuentra ese contacto.
2. Se establecen relaciones de tipo (llave, valor) en mapas para tener acceso inmediato dependiendo el caso en el que nos encontremos. (caso nombre repetido, caso nombre no repetido)
3. Se añade a un árbol binario el nombre ingresado y se inserta de forma ordenada, si el elemento ingresado es menor que el elemento existente en el árbol se añade a la izquierda y caso contrario se añade a la derecha. (para recorrer el árbol sin perder el orden se debe usar el algoritmo inOrder o usar el método display).

#### Búsqueda de Contacto:

1. Se verifica la condición de que si el nombre ingresado está presente en la libreta de contactos usando el mapa “datos repetidos”
2. En caso de nombres repetidos, se solicita información adicional (número asociado) para obtener detalles específicos.
3. Se muestra información detallada del contacto encontrado y se suma 1 al número de visitas del usuario.

#### Eliminación de Contacto:

- Se busca y elimina el contacto seleccionado por nombre o número.

#### Dos opciones de visualización de libreta:

1. Mostrar en una lista completa de contactos ordenada alfabéticamente, para esto se usa el método display que recorre el árbol de forma ordenada y recursiva de izquierda a derecha, siguiendo siempre el proceso (left,root,right), cabe resaltar que el método display utiliza la misma lógica del método de recorrido inOrder.

```
template <typename T>
void Btree<T>::display_node(Node<T> *node, int level) {
    if(node != nullptr){
        level++;
        display_node(node->left, level);
        std::cout << node->key << "(" << level-1 << " ) ";
        display_node(node->right, level);
    }
}
```

- Mostrar en una tabla los contactos agrupados por su inicial, para esto se modificó el método hash dentro de la clase hash\_table que permite asociar a cada nombre a la posición que se desea, dependiendo la inicial de cada nombre (Se utiliza lógica modular) lo cual se muestra a continuación:

```
template <typename VT>
unsigned HashMap<VT>::hash(std::string k) {
    unsigned int hashVal = 0;

    if (!k.empty()) {
        // Se obtiene el primer carácter de la cadena
        char firstChar = tolower(k[0]);
        if (firstChar >= 'a' && firstChar <= 'z')
            hashVal = firstChar - 'a'; // El hash
    }

    return hashVal % table_size; // modulo para as
}
```

#### Consultas adicionales:

- Mostrar la cantidad total de contactos presentes en la libreta, para esto se utilizó el método size() de la clase tabla\_hash.
- Mostrar la frecuencia con la que se ha buscado cada contacto. Para ello se recorre el vector que

Copia de seguridad:

- Creación de una copia de seguridad de la información de cada ejecución en un archivo de texto.

#### Salidas:

Información Detallada de Contactos:

Si se busca la información de un contacto, es posible conocer toda su información.

Estadísticas de la libreta

- Cantidad total de contactos presentes en la libreta.
- Numero de veces que fue consultado cada contacto

Tabla hash y historial

- Tabla ordenada alfabéticamente de contactos (cada fila representa una letra) en donde se muestra la relación entre el nombre y teléfono
- Árbol representado en una lista que muestra el historial de los contactos que se han añadido

## VI. PLANEACIÓN

El desarrollo del programa se compone de 4 etapas.

- Análisis del problema sugerido realizando una lluvia de ideas de lo que puede hacerse para implementar el problema.
- A partir de la división del problema en partes pequeñas, se codificó la libreta al mismo tiempo que se iba teniendo una **retroalimentación** constata por medio de videos de YouTube y otros recursos.
- Encontrar las dificultades de cada tarea y las enseñanzas, para así buscar posibles propuestas de mejora.
- Documentar los procesos en Trello y seguir con el siguiente problema.

Como se indicó anteriormente para la planeación y seguimiento de tareas se utilizó la plataforma Trello, en este caso se quiso crear tres tableros cada uno representa las tareas, dificultades y enseñanzas de cada adelanto.

Primer adelanto:

<https://trello.com/invite/b/j8BmeyqY/ATTI2ec1cd36dd4fef445d4a1bf210208994D0648354/primer-adelanto>

Segundo Adelanto:

<https://trello.com/invite/b/yhOb4R1R/ATTI63e3a96c430a3fa533f75c9e5c9bb58211446456/segundo-adelanto>

Tercer Adelanto:

<https://trello.com/invite/b/cuVjiKs9/ATTI2431e45a474d4b639fcc5a3d133ce0b09F5CED9B/tercer-adelanto>

## VII. EVIDENCIAS DE EJECUCIÓN:

- Interfaz de inicio:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
-----Libreta de contactos-----
1:Agregar contacto
2:Buscar contacto
3:Eliminar contacto
4:Mostrar lista completa de contactos
5:Mostrar contactos ordenados por letra
6:Mostrar cantidad de contactos en la lista
7:Mostrar lista de contactos con su frecuencia de busqueda
8:Realizar copia de seguridad en archivo de texto
9:Salir
```

- Insertar un contacto
- Buscar nombre cuando no está repetido

```
1
Ingrese su nombre
Carlos
Ingrese su telefono3163881131
Ingrese su direccion cada dato de su direccion:
Ingrese calle
137
Ingrese numero(#)
55
Ingrese (-)
32
Usuario de Instagram
carlos.galanp
Usuario de Github
carlygp
```

```

2
Ingrese nombre que quiere buscar
Carlos
Perfecto, quiere saber info de Carlos sobre: (1:telefono)(2:direccion)(3:redes sociales)(4:salir)
1
3163881131
Perfecto, quiere saber info de Carlos sobre: (1:telefono)(2:direccion)(3:redes sociales)(4:salir)
2
calle137#55-32
Perfecto, quiere saber info de Carlos sobre: (1:telefono)(2:direccion)(3:redes sociales)(4:salir)
3
Instagram: carlos.galanp Github: carlygp
Perfecto, quiere saber info de Carlos sobre: (1:telefono)(2:direccion)(3:redes sociales)(4:salir)

```

- Busca información de un nombre cuando esta repetido:

```

2
Ingrese nombre que quiere buscar
Carlos
Se necesita informacion adicional, ingrese el numero del contacto
3163881131
Perfecto, quiere saber info de Carlos sobre: (1:direccion)(2:redes sociales)(3:salir)

```

- Lo mismo sucede con la opción de eliminar para cuando hay nombres repetidos.

```

Ingrese el nombre del contacto que quiere eliminar
Carlos
se necesita informacion adicional, ingrese el numero del contacto
3163881131

```

- Historial de contactos (ejemplo que muestra el nombre y el nivel en el árbol en que se encuentra)

```
camilo(1) carlos(0)
```

- Tabla hash (key : nombre , value : telefono)

```

5
0:
1:
2: (Camilo,3172992204) (Carlos,3003908274)
3:
4:
5:
6:

```

- Tamaño de libreta

```
Hay 2 Contactos en la libreta disponibles
```

- Frecuencia de visitas

```
{Nombre: Camilo,frecuencia de visitas del perfil: 0}
{Nombre: Carlos,frecuencia de visitas del perfil: 1}
```

## VIII. EVALUACIÓN.

1. Para verificar que el programa es consistente y que siempre haya forma de finalizar el programa se hicieron pruebas de ingresarle datos no válidos y el programa los maneja de manera satisfactoria. Para ello se añadieron condicionales que no permitieran ingresar datos no deseados.
2. Cuando hay una búsqueda se propuso que si el dato que se está buscando es repetido pedirle el teléfono al usuario para poder acceder a su contacto por medio de mapas ordenados.
3. Cuando se busca eliminar un nodo en un árbol, hay que tener presente que si se pretende borrar un nodo padre se va a perder el acceso a los nodos hijos. Por lo que se dejó el árbol como un historial de todos los contactos que han entrado a la libreta y solo eliminar el contacto de la tabla hash, la cual tiene un método que permite borrar elementos por su llave.
4. La libreta no acepta el ingreso de contactos repetidos, pero sí de nombres repetidos.
5. Realizar copias de seguridad en un archivo de texto asegura la persistencia de los datos.

## IX. ANÁLISIS Y RESULTADOS:

- Primero es importante resaltar que la complejidad de **búsqueda** en árboles binarios balanceados está dada por  $O(\log_2 n)$  [1]. Sin embargo, en este caso el árbol solo tiene la funcionalidad de ingresar nombres con el fin de ordenarlos, ya que nuestra clase árbol inserta los nodos de manera ordenada. Además, permite visualizarlos de forma también ordenada y en este caso tener un historial de los contactos que se han ingresado a la libreta. Para esta visualización se recorre el árbol con el método inorder[3] y con un “cout” que va mostrando el árbol de forma ordenada y recorriéndolo de izquierda a derecha.
- Ahora bien, para consultas a partir de un nombre o un teléfono(keys) , utilizamos realmente mapas ordenados que tienen complejidad también  $O(\log n)$  [2] , siendo n el número de elementos en el mapa justo antes de la operación que se desee ejecutar, en

este caso, buscar si se encuentra un elemento dada una clave (nombre o teléfono). Lo cual es más eficiente que si utilizáramos búsquedas lineales para consultar los datos del contacto.

- La tabla hash al igual que el árbol se utiliza como método para que el usuario tenga una forma familiar de visualizar los contactos en orden alfabético.

## X. CONCLUSIONES:

- Aunque el programa satisface las necesidades básicas de gestión de contactos, se podrían considerar mejoras adicionales. Estas podrían incluir la implementación de una interfaz gráfica de usuario (GUI) para una experiencia más amigable con el usuario.
- La utilización de estructuras de datos como árboles binarios y mapas ha mejorado la eficiencia en la búsqueda y almacenamiento de contactos. El uso de un árbol binario facilita la visualización ordenada de los contactos, mientras que los mapas permiten un acceso rápido mediante claves únicas. Además, el peor caso que se presente en las consultas de los mapas tiene complejidad  $O(\log_2(n))$  lo cual es una mejora considerable frente a algún tipo de estructura que se tuviera que hacer búsqueda lineal.
- También es importante resaltar que la tabla hash fue una herramienta excelente para la visualización de datos, sin embargo, su uso en el ámbito laboral tiene un enfoque diferente y se debe buscar la menor cantidad de colisiones entre datos que se ingresen.
- Por último, se observa que implementación de esta aplicación fue exitosa y demuestra la utilidad de estructuras de datos eficientes y prácticas para crear un sistema de gestión de contactos. El uso de herramientas proporcionadas por C++ permitió una mejor administración de memoria e hizo que este programa sea efectivo para organizar y acceder a la información de contactos.

## XI. REFERENCIAS

- [1] [https://ccia.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb\\_BB.htm](https://ccia.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_BB.htm)
- [2] <https://aprende.olimpiada-informatica.org/node/374>
- [3] <https://www.youtube.com/watch?v=tBaOQeyXYqg&t=321s>  
<https://hhmosquera.wordpress.com/arbolesbinarios/>  
[https://www.it.uc3m.es/pbasanta/asng/course\\_notes/ch07.html](https://www.it.uc3m.es/pbasanta/asng/course_notes/ch07.html)  
<https://www.youtube.com/watch?v=9tZsDJ3JBUA>