Algoritmos y estructuras de datos

Universidad del Rosario

Carlos Andrés Galán Pérez

Santiago Peña

Introducción:

El presente documento aborda el desarrollo 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. Además, integra estructuras de datos como árboles binarios y tablas hash para optimizar la búsqueda y almacenamiento de información.

Objetivos:

General.

Implementar una libreta en la que se pueda buscar, eliminar y visualizar contactos teniendo en cuenta algunas consideraciones, haciendo uso de estructuras de datos y algoritmos de ordenamiento como lo son arboles binarios, tablas hash y árbol de búsqueda binaria. Adicionalmente se va a poder hacer una copia de seguridad de cada registro.

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. Eliminación de contactos en la libreta usando los métodos proporcionados por las clases árbol y tabla hash.
4. 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).
5. 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.

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.

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 en postgres.

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 veces que se ha visitado en cada ejecución un perfil
};
```

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 se necesita el nombre para acceder al contacto
map<string,contacto>datos_rep2;  // se necesita el numero porque el nombre esta repetido
map<string,int>datos_rep; // mirar cuantas veces esta repetido el nombre
```

- 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);
  }
}
```

2. 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:

```cpp
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 k y lo convierte a minúsculas utilizando la función tolower.
        char firstChar = tolower(k[0]);
        if (firstChar >= 'a' && firstChar <= 'z') { // verificar si la primera letra pertenece al conjunto de letras
            hashVal = firstChar - 'a'; // El hash va a ser el ascci de la letra menos el ascci de 'a'
        }
    }

    return hashVal % table_size; // modulo para asignar posicion correspondiente
}
```

Consultas adicionales:

1. Mostrar la cantidad total de contactos presentes en la libreta, para esto se utilizó el método size() de la clase tabla_hash.

2. 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

Planeación y metodología:

El desarrollo del programa se compone de 4 etapas.

1. Análisis del problema sugerido realizando una lluvia de ideas de lo que puede hacerse para implementar el problema.
2. A partir de la división del problema en partes pequeñas, se codificó la libreta al mismo tiempo que se iba teniendo una **retroalimentación** constate por medio de videos de YouTube y otros recursos.

3. Encontrar las dificultades de cada tarea y las enseñanzas, para así buscar posibles propuestas de mejora.
4. 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/ATTI2ec1cd36dd4fefd45d4a1bf210208994D0648354/primer-adelanto

Segundo Adelanto:

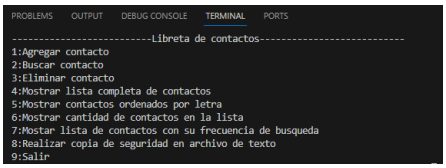
https://trello.com/invite/b/yhOb4R1R/ATTI63e3a96c430a3fa533f75c9e5c9bb58211446456/segundo-adelanto

Tercer Adelanto:

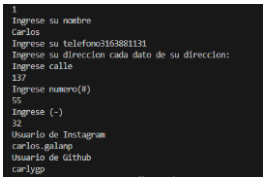
https://trello.com/invite/b/cuVjiKs9/ATTI2431e45a474d4b639fcc5a3d133ce0b09F5CED9B/tercer-adelanto

Evidencias de ejecución:

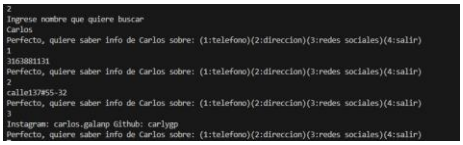
- Interfaz de inicio:



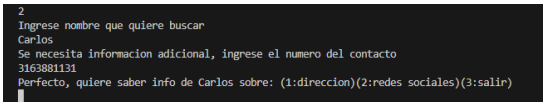
- Insertar un contacto



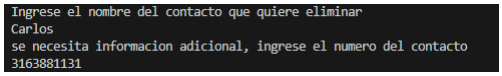
- Buscar nombre cuando no está repetido



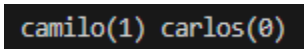
- Busca información de un nombre cuando esta repetido:



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

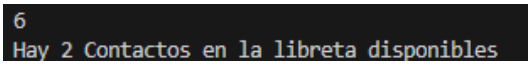


- Historial de contactos (ejemplo que muestra el nombre y el nivel en el árbol en que se encuentra)

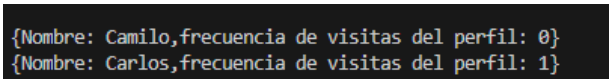


- Tabla hash (key : nombre , value : telefono)

- Tamaño de libreta



- Frecuencia de visitas



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.
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.

Análisis y resultados:

Primero es importante resaltar que la complejidad de búsqueda en árboles binarios está dada por O(log n) lo cual es más eficiente que si utilizáramos listas y búsquedas lineales. En donde el peor caso sería si el árbol queda muy desbalanceado a la hora de ingresar los datos. Por ejemplo, si solo

se ingresan nombres menores o mayores al anterior. Vamos a tener solo una rama y para recorrerlo debería costar más recursos. Ahora bien, el mejor caso no es realmente muy probable que suceda debido a que el usuario ingresa datos sin tener en cuenta que van a ser guardados en un árbol.

Además, la utilización de mapas permitió durante todo el proceso, mantener las relaciones que se necesitaban para por ejemplo mantener actualizado los datos de los contactos. Por lo que fueron muy útiles y se usaron bastante.

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.

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.

Por último, se observa que implementación de esta aplicación fue exitosa y demuestra la utilidad de estructuras de datos eficientes y prácticas de programación para crear un sistema de gestión de contactos. El uso de herramientas proporcionadas por c++ hizo que este programa sea efectivo para organizar y acceder a la información de contactos.