****

**MiniProyecto**

**Estudiantes:**

|  |  |
| --- | --- |
| **Nombre** | **Cuenta** |
| **Josue Eliezer Castillo Soto** | **32151059** |
| **Douglas Daniel Ramirez Canales** | **32151049** |

**Asignatura: Estructura de Datos**

**Ingeniero: Oscar Fernando D`Cuire Galeano**

**Semana 6**

**Sección: 1168**

**Fecha y Lugar: Tegucigalpa M.D.C 01/06/2025**

**Sede: Ceutec Tegucigalpa**

**Link de GitHub del Proyecto:**

[**https://github.com/DaniRam17/MiniProyecto**](https://github.com/DaniRam17/MiniProyecto)

**Link del Video:**

[**https://www.youtube.com/watch?v=GWWEuY0txTQ**](https://www.youtube.com/watch?v=GWWEuY0txTQ)

**Descripción del Algoritmo de Dispersión y Funcionamiento**

El sistema implementa una **tabla hash con sondeo cuadrático** para almacenar y buscar usuarios de manera eficiente a partir de su ID. La función hash personalizada se basa en la constante de Knuth para dispersión uniforme:

const unsigned int A = 2654435761;

int hashBase = (id \* A) % TAMANIO\_TABLA;

return (hashBase + intento \* intento) % TAMANIO\_TABLA;

**Funcionamiento:**

1. Se calcula el valor base con una multiplicación por A, luego se aplica módulo con el tamaño de la tabla.
2. Si hay colisión (la posición ya está ocupada), se aplica **sondeo cuadrático**, aumentando intento hasta encontrar una posición libre.
3. Si no hay espacio disponible o se detecta sinónimo (colisión sin posibilidad de resolver), no se inserta el usuario.

**Ejemplos de Llaves Sinónimas o Colisiones en este algoritmo**

Se dice que hay **llaves sinónimas** cuando dos ID diferentes generan la misma dirección hash:

hashBase1 = (id1 \* A) % TAMANIO\_TABLA

hashBase2 = (id2 \* A) % TAMANIO\_TABLA

Si hashBase1 == hashBase2 **son sinónimos**.

**Ejemplo 1:**

id1 = 1 → (1 \* A) % 67 = 2654435761 % 67 = 29

id2 = 68 → (68 \* A) % 67 = (180501628, mod 67) = 29

Ambos dan **posición 29** por lo tanto genera una colisión

**Ejemplo 2:**

id1 = 7 → (7 \* A) % 67 = (18581050327) % 67 = 31

id2 = 74 → (74 \* A) % 67 = (19643024514) % 67 = 31

**Condición de conflicto**: Si tras aplicar sondeo cuadrático no se encuentra una celda libre sin colisiones con claves diferentes, se considera una colisión irresoluble.

**TDA utilizado para los usuarios**

**1. Tabla Hash (Hash Table)**

**Clasificación:**

* Estructura de acceso directo
* No lineal
* No ordenada

**¿Cómo funciona?**

* Se usa para almacenar usuarios según su ID mediante una función hash personalizada.
* Cada usuario se inserta en una posición calculada, y si hay colisión, se usa sondeo cuadrático para buscar otra celda.

**Operaciones principales:**

|  |  |
| --- | --- |
| Operación | Descripción |
| insertarUsuario() | **Inserta un nuevo usuario según el ID.** |
| buscarUsuario() | **Busca un usuario por ID.** |
| eliminarUsuarioPorId() | **Elimina un usuario con ID específico.** |
| eliminarUsuarioPorPosicion() | **Elimina un usuario por su posición de inserción.** |
| buscarPorPosicion() | **Retorna el usuario en una posición dada.** |
| obtenerUsuariosComoLista() | **Retorna una lista ordenada para visualización.** |

**Archivos: HashTable.h / HashTable.cpp**

**Lista Secuencial (Vector como historial)**

**Tipo:**

* Estructura lineal dinámica
* Ordenada por inserción

**¿Dónde se usa?**

* La clase HashTable también mantiene un vector<Usuario\*> historialInsercion, que preserva el orden de los usuarios tal como fueron agregados.
* Esto permite acceder por posición (índice), recorrer usuarios ordenadamente, y mostrar la lista visual en el mismo orden.

**Ventajas combinadas:**

* La tabla hash permite acceso rápido por ID (O(1) en promedio).
* El vector permite recorrido ordenado y visualización.

**Visualmente:**

**Usuario1 → Usuario2 → Usuario3 ... (orden de inserción - historial)**

**↘ ↘ ↘**

**tabla[29] tabla[43] tabla[57] (ubicación en la tabla hash)**

**En resumen:**

|  |  |
| --- | --- |
| Propiedad | Valor |
| TDA principal | **Tabla hash con sondeo cuadrático** |
| TDA auxiliar | **Vector (historial de inserción)** |
| Clasificación | **No lineal (hash), lineal ordenada (vector)** |
| Acceso por ID | **O(1) promedio (hashing)** |
| Acceso por orden | **O(1) acceso, O(n) recorrido (vector)** |
| Uso | **Usuarios registrados en el sistema** |

**TDA utilizado para los accesos**

**Lista Doblemente Enlazada (Tipo Pila - LIFO)**

**Clasificación:**

|  |  |
| --- | --- |
| Propiedad | Valor |
| Tipo de estructura | Lista doblemente enlazada |
| Estructura | Dinámica, encadenada |
| Clasificación lógica | TDA lineal con acceso secuencial |
| Dirección del crecimiento | Hacia adelante (insertar al inicio) |
| Comportamiento | Tipo **Pila (LIFO)** |

**¿Cómo se implementa?**

* Cada usuario tiene una **instancia de lista propia** llamada ListaAccesos.
* Esta lista se define con:
  + struct Acceso (nodo con fechaHora, \*siguiente, \*anterior)
  + class ListaAccesos que gestiona los nodos

Archivos: Acceso.h / Acceso.cpp

**Operaciones permitidas**

|  |  |
| --- | --- |
| Operación | Descripción |
| registrarAcceso() | Inserta un nuevo acceso al inicio de la lista |
| eliminarAccesos(n) | Elimina los primeros n accesos desde la cabeza |
| contarAccesos() | Retorna la cantidad de accesos registrados |
| buscarPorPosicion(pos) | Devuelve el acceso en la posición pos |
| mostrarAccesos() | Recorre todos los accesos e imprime/visualiza |

**Comportamiento tipo Pila (LIFO)**

Aunque sea una lista, la lógica de inserción y eliminación implementada es **tipo pila**:

* **Último acceso** se inserta en la cabeza.
* **Eliminación** comienza desde la cabeza.
* Accesos más recientes se procesan primero.

**Visualmente:**

Acceso más reciente → Acceso anterior → ... → Acceso más antiguo

↑

cabeza

* Cada nodo apunta al anterior y siguiente.
* Se permite recorrer tanto hacia adelante como hacia atrás si se requiere.

**En resumen:**

|  |  |
| --- | --- |
| Característica | Valor |
| TDA utilizado | Lista doblemente enlazada |
| Comportamiento | Tipo pila (LIFO) |
| Estructura por usuario | Una lista independiente |
| Acciones típicas | Agregar al inicio, eliminar desde el inicio |
| Uso | Control de accesos por fecha y hora |

**Resumen de la Interfaz Gráfica Utilizada En Gestión de Accesos**

**Tecnología utilizada:**

* **Librería:** [SFML](https://www.sfml-dev.org/)
* **Lenguaje:** C++
* **Modo:** Gráfico 2D por ventanas (sf::RenderWindow)

**Organización general de la interfaz**

La interfaz principal se estructura en **3 secciones principales**:

|  |  |
| --- | --- |
| Sección | Descripción |
| Entrada de datos | Campos para ID, nombre, correo, eliminar accesos, buscar/eliminar por posición |
| Acciones (botones) | Registrar usuario, registrar acceso, eliminar, buscar, visualizar |
| Mensajes del sistema | Zona inferior que muestra resultados, alertas y errores |

**Funciones visuales destacadas**

|  |  |
| --- | --- |
| Función visual | Descripción |
| Ver lista de accesos | Muestra los accesos del usuario en ventana gráfica tipo nodos conectados |
| Ver usuarios registrados | Despliega usuarios en orden de inserción como nodos encadenados |
| Scroll visual (parcial) | Permite ver múltiples nodos si hay muchos datos, expandiendo espacio |
| Retroalimentación inmediata | Cada acción del usuario (registro, búsqueda, error) actualiza el mensaje inferior con color específico |

**Ventanas gráficas adicionales**

Se agregaron dos **ventanas dedicadas** para visualizar estructuras de datos:

* mostrarListaAccesos: Dibuja los accesos de un usuario como una lista enlazada horizontal.
* mostrarListaUsuarios: Dibuja todos los usuarios en orden de inserción como nodos.
* Implementadas en: ListVisualizer.cpp / ListVisualizer.h

**Durante el desarrollo del MiniProyecto, se presentaron diversas dificultades técnicas y de integración, entre las cuales destacan:**

**1. Configuración de la interfaz gráfica en C++ (SFML)**

Uno de los primeros retos fue recordar cómo configurar correctamente SFML para poder compilar y enlazar las bibliotecas necesarias en Visual Studio Code y MinGW. Se presentaron errores como referencias indefinidas y rutas incorrectas de inclusión. Este problema fue solucionado investigando la [documentación oficial de SFML](https://www.sfml-dev.org/tutorials/) y ajustando las rutas de los archivos .lib y .dll.

**2. Errores de compilación y vinculación (linker)**

Durante la integración de nuevos archivos (como ListVisualizer.cpp), el compilador arrojaba errores del tipo undefined reference, debido a que no se había incluido el archivo fuente en la línea de compilación. Este error fue resuelto al revisar cuidadosamente el comando de build y asegurarse de que todos los .cpp necesarios estuvieran presentes.

**3. Orden visual y superposición de elementos en la interfaz**

A medida que se agregaban más funcionalidades (nuevos botones e inputs), la organización visual se volvió confusa, con elementos encimados o mal alineados. Esto fue corregido reorganizando las posiciones de los elementos en secciones: ingreso, acciones y resultados, y aplicando espaciado consistente.

**4. Manejo de colisiones en la función hash**

En la implementación inicial de la tabla hash, no se contempló adecuadamente el manejo de sinónimos (colisiones), lo cual causaba que algunos usuarios no se insertaran correctamente. Fue necesario implementar sondeo cuadrático y condicionar correctamente la detección de colisiones para asegurar que cada ID sea único.

**5. Persistencia de datos en archivo binario**

Hubo confusión al principio sobre cómo serializar estructuras con punteros, como Usuario y Acceso, al guardar en archivo binario. La solución fue separar los datos persistentes de los datos volátiles y usar structs planos para serialización en Archivo.cpp.

Bibliografía

*Class Index (SFML / Learn / 2.6.1 Documentation)*. (s/f). Sfml-dev.org. Recuperado el 29 de mayo de 2025, de https://www.sfml-dev.org/documentation/2.6.1/classes.php

*compare the sizes of two boxes - C++ Forum*. (s/f). Cplusplus.com. Recuperado el 29 de mayo de 2025, de https://cplusplus.com/forum/beginner/223467/

*No title*. (s/f). Cplusplus.com. Recuperado el 29 de mayo de 2025, de https://cplusplus.com/doc/tutorial/files/

*Reading from a txt file to linked list - C++ Forum*. (s/f). Cplusplus.com. Recuperado el 29 de mayo de 2025, de https://cplusplus.com/forum/beginner/270514/