

Informe Proyecto Blockchain

1. Introducción

Se dará a conocer el proceso de trabajo de Blockchain, su importancia, su aplicación en el mundo real. Se explica el desarrollo de una aplicación implementada en C++ usando las principales estructuras de datos para el proceso de cadena de bloques con transacciones bancarias.

2. Descripción del caso de estudio “Transacciones Bancarias”

En las entidades bancarias y financieras, la seguridad y privacidad de los datos son aspectos críticos para las transacciones bancarias. Los sistemas tradicionales de almacenamiento y búsqueda de datos pueden presentar vulnerabilidades, y el acceso no autorizado a la información confidencial puede tener consecuencias graves, como el robo de identidad o el fraude financiero. Existe la necesidad de desarrollar una solución confiable y segura que garantice la protección de los datos transaccionales y, al mismo tiempo, permita búsquedas eficientes para su procesamiento. Debido a esa necesidad se propone la implementación de una solución basada en blockchain.

3. Importancia del Blockchain en la realización de Transacciones Bancarias

Las entidades bancarias son las encargadas de proporcionar la **liquidez monetaria inmediata** que se necesita para realizar operaciones comerciales y financieras entre personas, entre personas y empresas, entre empresas.

A fin de contar con dicha disposición de dinero a cualquier hora, lugar y momento, en la actualidad se intensificado el uso de transferencias en línea, que son procesos informáticos que descuentan o incrementan el dinero de una cuenta bancaria o monedero electrónico inmediatamente después de realizada alguna operación comercial o financiera como compra, venta, transferencia de dinero entre cuentas de personas, pagos de planillas, etc.

Con la finalidad de asegurar que el proceso de Transacción Bancaria se confiable y seguro, éste se apoyado de la herramienta de Blockchain.

Aquí las razones importantes para el uso de Blockchain en el sector financiero:

- **Seguridad:** El blockchain ofrece una capa adicional de seguridad para las transacciones financieras. Al utilizar criptografía avanzada y técnicas de consenso distribuido, como la prueba de trabajo o la prueba de participación, el blockchain garantiza la integridad de los datos y evita la posibilidad de manipulación o alteración de la información financiera.
- **Transparencia:** La naturaleza transparente del blockchain permite a todas las partes involucradas en una transacción financiera ver y verificar la información. Esto ayuda a aumentar la confianza entre los participantes y a reducir la necesidad de intermediarios, ya que cada transacción queda registrada de forma inmutable y accesible para todos los participantes autorizados.
- **Eficiencia y rapidez:** Las transferencias bancarias a través del blockchain pueden agilizar significativamente el proceso, ya que eliminan la necesidad de verificaciones constantes. Al utilizar una red

distribuida y descentralizada, las transacciones pueden realizarse en tiempo real y sin demoras significativas.

- **Reducción de costos:** El uso de blockchain en las transferencias bancarias puede reducir los costos asociados con los intermediarios y los procesos de liquidación. Al eliminar la necesidad de terceros de confianza y automatizar las transacciones, se pueden lograr ahorros significativos en comisiones y tarifas.
- **Acceso global:** El blockchain permite transferencias bancarias más accesibles a nivel global, especialmente en regiones donde el acceso a servicios financieros tradicionales es limitado. Las personas sin cuentas bancarias pueden participar en transacciones seguras y económicas utilizando billeteras digitales basadas en blockchain.
- **Innovación y nuevos modelos de negocio:** El blockchain ha estimulado la innovación en el sector financiero, permitiendo el desarrollo de nuevos modelos de negocio y servicios financieros. Las tecnologías basadas en blockchain, como los contratos inteligentes, han facilitado la creación de soluciones financieras descentralizadas y otras aplicaciones disruptivas.

En resumen, el blockchain ha transformado el sector financiero y las transferencias bancarias al proporcionar seguridad, transparencia, eficiencia y reducción de costos. Su capacidad para agilizar las transacciones, eliminar intermediarios y aumentar la confianza entre los participantes lo convierte en una herramienta poderosa para mejorar los servicios financieros a nivel global.

Así mismo su aplicación se ha visto expandida en otros sectores tales son los casos de:

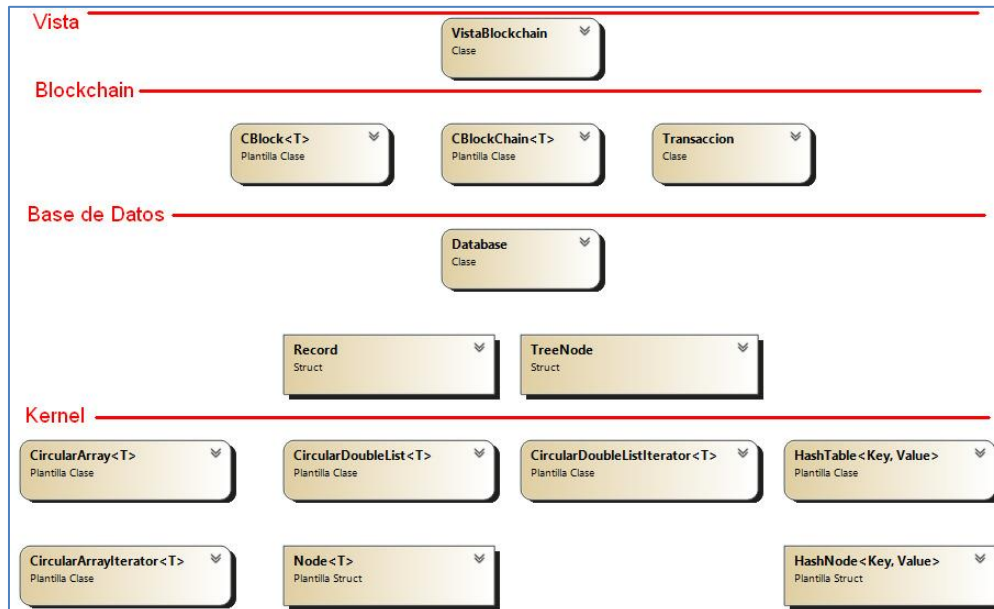
- **Cadena de suministro:** El blockchain se utiliza para rastrear y auditar el movimiento de productos a lo largo de la cadena de suministro. Permite la verificación de la autenticidad, la trazabilidad de los productos y la reducción de fraudes en sectores como la alimentación, la moda y la industria farmacéutica [1].
- **Salud:** En el ámbito de la salud, el blockchain se utiliza para mantener registros médicos electrónicos seguros y compartidos, facilitando el intercambio de información entre proveedores de atención médica. También se emplea para la gestión de ensayos clínicos y el seguimiento de la cadena de frío en el transporte de medicamentos [2].
- **Energía:** El blockchain se utiliza para la gestión descentralizada de redes eléctricas, el seguimiento y certificación de energías renovables y la facilitación de transacciones peer-to-peer de energía. Ayuda a aumentar la eficiencia energética y a fomentar la adopción de energías limpias [3].
- **Gobierno:** El blockchain se ha explorado para mejorar la transparencia, la trazabilidad y la seguridad en los procesos gubernamentales, como la votación electrónica, la gestión de identidad y la administración de registros públicos. Ofrece la oportunidad de reducir la corrupción y aumentar la confianza ciudadana [4].
- **Propiedad intelectual:** El blockchain se utiliza para registrar y proteger la propiedad intelectual, como derechos de autor y patentes. Permite una prueba de existencia y autenticidad de los activos intangibles, agilizando los procesos de registro y protección [5].

4. Implementación

4.1. Arquitectura

Para lograr el cumplimiento de los requerimientos se ha desarrollado 4 capas dentro de la arquitectura de programas en C++.

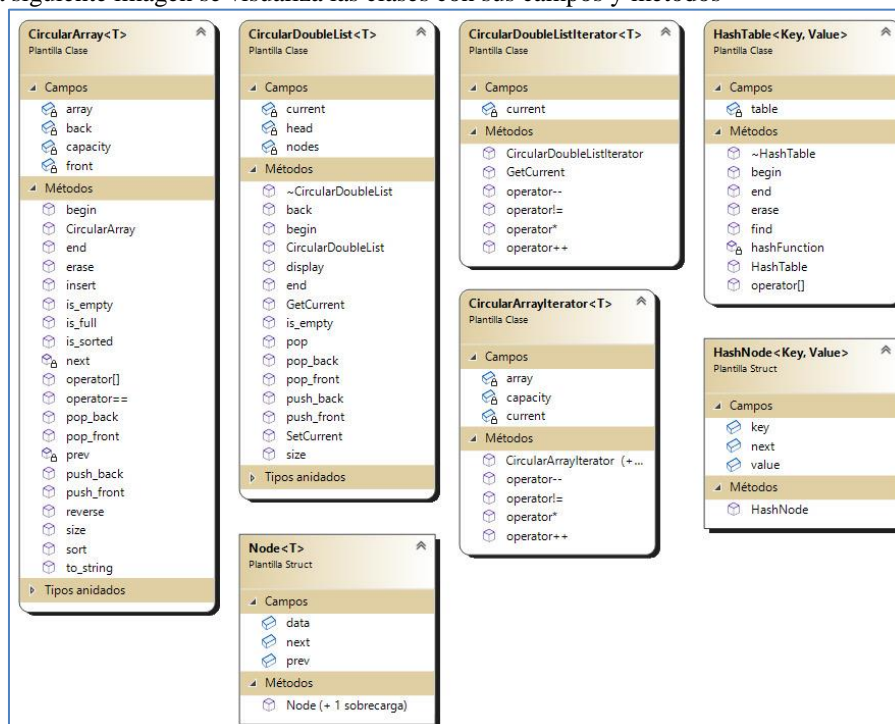
Se explica en orden de los procesos de construcción del Blockchain: Capa Kernel, Capa Base de Datos, Capa Blockchain y Capa Vista.



1.1. Kernel

Aquí se han implementado y probado todas las estructuras de datos que se usarán en el proyecto, su importancia es por ser el soporte de la información que se gestiona en las capas superiores.

En la siguiente imagen se visualiza las clases con sus campos y métodos



1.1.1. Clase *CircularArray.h*: Arreglo Circular

Clase implementada para definir los métodos necesarios para gestionar el arreglo circular. A continuación, se detallan estos métodos:

Método	Descripción
void push_front(T data)	Método para ingresar un dato antes del puntero front del array circular
void push_back(T data)	Método para ingresar un dato después del puntero back del array circular
void insert(T data, int pos)	Método que ingresa un elemento data, en una posición pos
T pop_front()	Método que elimina el elemento anterior al puntero front
T pop_back()	Método que elimina el elemento posterior al puntero back
bool is_full()	Método que determina si el arreglo circular está lleno o no.
bool is_empty()	Método que devuelve un valor de 1 si el arreglo circular está vacío; es decir, si front y back son iguales a -1
int size()	Método que devuelve el tamaño del conjunto de datos desde el front hasta el back.
void sort()	Método que ordena los datos del arreglo circular en forma ascendente.
bool is_sorted()	Método que determina si los datos del arreglo circular están ordenados o no.
void reverse()	Método que invierte el orden de los datos del arreglo circular.
string to_string(string sep = " ")	Método para convertir los datos del arreglo circular a cadena.
int next(int)	Método que sirve de apoyo para apuntar al siguiente elemento de un índice ingresado.
int prev(int)	Método que sirve de apoyo para apuntar al elemento anterior de un índice ingresado.
void erase(int pos)	Método que elimina un dato del arreglo circular de acuerdo a su posición.

1.1.2. Clase *CircularDoble.h*: Lista circular doblemente enlazada

Clase implementada para definir los métodos necesarios para gestionar una lista circular doblemente enlazada. A continuación, se detallan estos métodos:

Método	Descripción
void push_front(T data)	Método para ingresar un dato antes del puntero head
void push_back(T data)	Método para ingresar un dato después del puntero head
T pop_front()	Método para eliminar el dato antes del puntero head
T pop_back()	Método para eliminar el dato después del puntero head
void pop(Node<T>* pos)	Método para eliminar un nodo de acuerdo a su posición
bool is_empty()	Método que determina si hay algún nodo o no.
void display()	Método que muestra los valores de los nodos de la lista circular doble.
int size()	Método que retorna la cantidad de nodos existentes en la lista circular doble.

T back()	Método que retorna el valor del nodo anterior al head.
----------	--

1.1.3. HashTable.h: Clase Tabla Hash

Método	Descripción
void insert(const Key& key, const Value& value)	Método de inserción al hashtable a partir de una llave key y el valor a ingresar. Para esto, primero se verifica si la clave existe o no en el bucket, luego, si la clave ya existe se añade el valor al vector correspondiente a dicha clave, caso contrario se añade la clave y el valor de forma normal al bucket. Dentro de este método también se aplica el método rehashing en caso de que el fillFactor sobrepasa a maxFillFactor, para lo cual se duplica el tamaño del bucket.
size_t bucket_size(size_t index)	Método para obtener el tamaño del bucket a través de su índice. Si este índice es mayor o igual tamaño del bucket entonces se retorna 0.
size_t bucket_count()	Método para obtener la cantidad de buckets.
typename Bucket::iterator begin(size_t index)	Método que determina el inicio del iterador para recorrer los elementos de un bucket para lo cual se ingresa su índice.
typename Bucket::iterator end(size_t index)	Método que determina el final del iterador para recorrer los elementos de un bucket para lo cual se ingresa su índice.
void rehashing(size_t newSize)	Método para aplicar el rehashing cuando se sobrepasa el maxFillFactor. Para esto se crea un nuevo arreglo de buckets con el nuevo tamaño, luego se recorren los pares clave-valor en cada uno de los buckets, posteriormente se calcula el nuevo índice del bucket utilizando la nueva función de hash y se inserta cada par clave-valor en el bucket que le corresponde. Finalmente, se actualiza el arreglo de buckets con el nuevo arreglo

1.2. Capa Base de Datos

Los aspectos que sobresalían de los requerimientos era la capacidad de búsqueda y consulta de diversos datos de Blockchain, así mismo tenía que tener la capacidad de importar los datos de Bloques y Transacciones desde un archivo en formato CSV, de la misma forma lo tenía que exportar

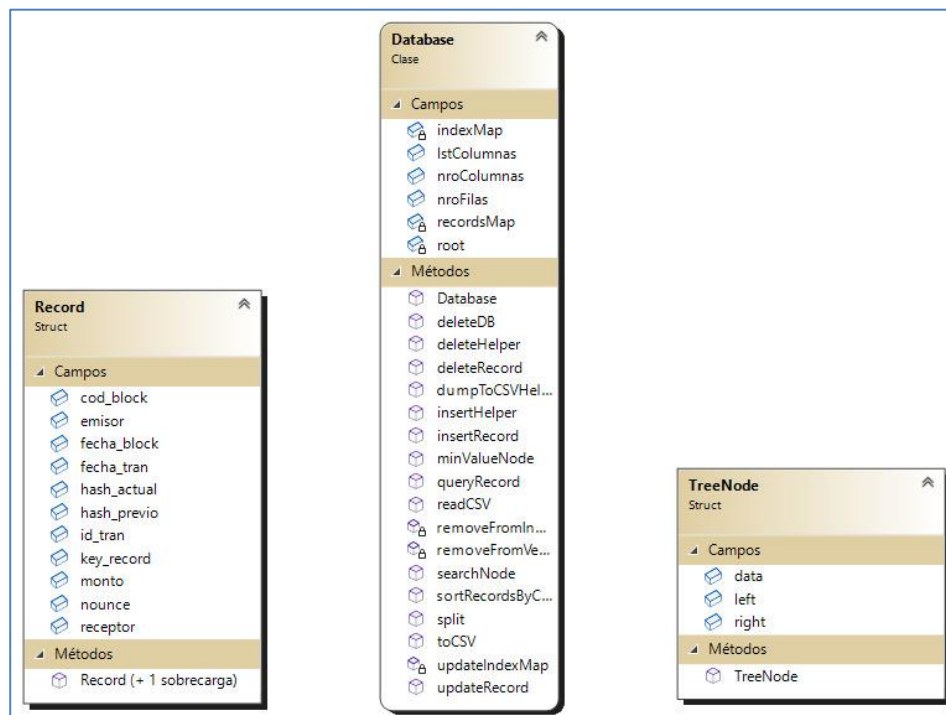


Figura10. Estructura de clases de la capa BD.

1.2.1. Clase Database: BD.h

Esta clase se encarga mantener la persistencia de la información generada por el Blockchain, su construcción es a base de un árbol BTS (Binary Tree Search), se usó como referencia una tesis sobre implementación de Tabla Hash¹:

- Permite realizar las consultas por diversas columnas o atributos de las transacciones,
- Permite mostrar listados ordenados por alguna columna,
- Permite importar información de bloques desde un archivo en formato CSV, permite exportar información de bloques en un archivo CSV.
- Cada vez que se guarda un Bloque, inmediatamente de su procesos de Proof of Work o minado, se guarda esta información en la Base de datos

Método	Descripción
int gen_key_record(int cod_block, int id_tran)	Método para la generación de una clave única para cada registro. Retorna un código de tipo entero basado en la multiplicación del código o id del bloque por 1 millón, y se suma con el id de la transacción. La finalidad de esto es lograr la inserción de todas las transacciones, aunque tengan datos parecidos o iguales.
std::string* split(const std::string& line, char delimiter)	Esta función permite obtener de cada línea con datos separados por comas, cada dato en un arreglo de tipo string.
bool readCSV(const std::string& filename, char delimiter)	Método para la lectura de los datos del archivo texto en formato CSV.
void dumpToCSVHelper(TreeNode* node, std::ofstream& file)	Esta función vuelca los datos de la Base de datos a un archivo CSV.
bool toCSV(std::string& nameFile)	Con este método se lee un archivo texto en formato CSV y se almacena en la base de datos.
void insertRecord(const Record& record)	Método para guardar un registro de una transacción en la estructura de tipo árbol binario de búsqueda o Base de Datos.
TreeNode* insertHelper(TreeNode* node, TreeNode* newNode)	Todos los métodos que contienen Helper son los que a bajo nivel realizarán gestiones de información directamente con el árbol Database. En este caso, este método inserta un nodo al árbol binario de búsqueda, este nuevo nodo se agregará a la izquierda o derecha del primer nodo ingresado dependiendo de si es menor o mayor que esta. Si el nuevo nodo es nulo, se inserta en la raíz.
void deleteRecord(int id_block, int id_tran = -1)	Método para eliminar un registro o transacción de la base de la base de datos.
void deleteDB(TreeNode* node)	Método para borrar toda la base de datos.
TreeNode* deleteHelper(TreeNode* node, TreeNode* deleteNode)	Método para eliminar el nodo ubicado a la izquierda o a la derecha dependiendo de la condición cumplida.
TreeNode* minValueNode(TreeNode* node)	Método para hallar el hijo de menor valor de un nodo
void updateRecord(const Record& updatedRecord)	Método para la modificación de un Registro
Record queryRecord(int id_block, int id_tran = -1)	Método que retorna la data de una transacción o del bloque si el parámetro id_tran es -1.
TreeNode* searchNode(int id)	Método que busca un nodo por su id del bloque, si este no se encuentra, se devuelve un valor nulo.
void sortRecordsByColumn(const std::string& column)	Método para ordenar los datos por alguna columna
void updateIndexMap(TreeNode* node)	Esta BD tiene dos partes, el guardado de información de las transacciones en cada nodo del BTS y asociación de cada registro a un ID para realizar su localización o búsqueda inmediata mediante la clase implementada HashTable
void removeFromIndexMap(TreeNode* node)	Método para eliminar el dato del HashTable

¹ [3] Trabajo Fin de Grado - Implementación Alternativa de una Tabla Hash en C++

1.2.2. Estructura Record

Esta estructura es la base para la gestión de la información de cada transacción del bloque de cadenas del Blockchain, contiene el código del bloque y el Id de cada transacción que tenga este bloque, contiene además una clave única Key_record que le permite registrarse en la Base de datos, permitiendo que exista otros registros que en el resto de columnas tengan datos parecidos o iguales.

Atributo	Descripción
int key_record	Clave única de cada registro
int cod_block	Código de un bloque en la cadena de bloques.
string fecha_block	Fecha de generación del bloque en formato “AAAA/MM/DD hh:mm” AAAA: Año MM: Mes DD: Día hh: hora en 24 horas mm: minutos
string hash_previo	Dato del hash del bloque anterior
string hash_actual	Dato del hash del bloque actual
int nounce	Nonce o valor que indica en que paso se generó el hash con el prefijo de 4 ceros “0000CVXCV87CV9XC0VX...”
int id_tran	Contador único de cada transacción por bloque.
double monto	Valor del dinero a transferirse
string emisor	Nombre de quien emite el dinero
string receptor	Nombre de quien recepciona el dinero
string fecha_tran	Fecha y hora de la transacción en formato “AAAA/MM/DD hh:mm” AAAA: Año MM: Mes DD: Día hh: hora en 24 horas mm: minutos

1.2.3. Estructura TreeNode

Esta contiene la estructura básica del nodo del árbol BTS Database

Atributo	Descripción
Record data	Estructura de una transacción, incluye datos de su bloque respectivo
TreeNode *left	Rama izquierda del árbol
TreeNode *right	Rama derecha del árbol

1.3. Capa Blockchain

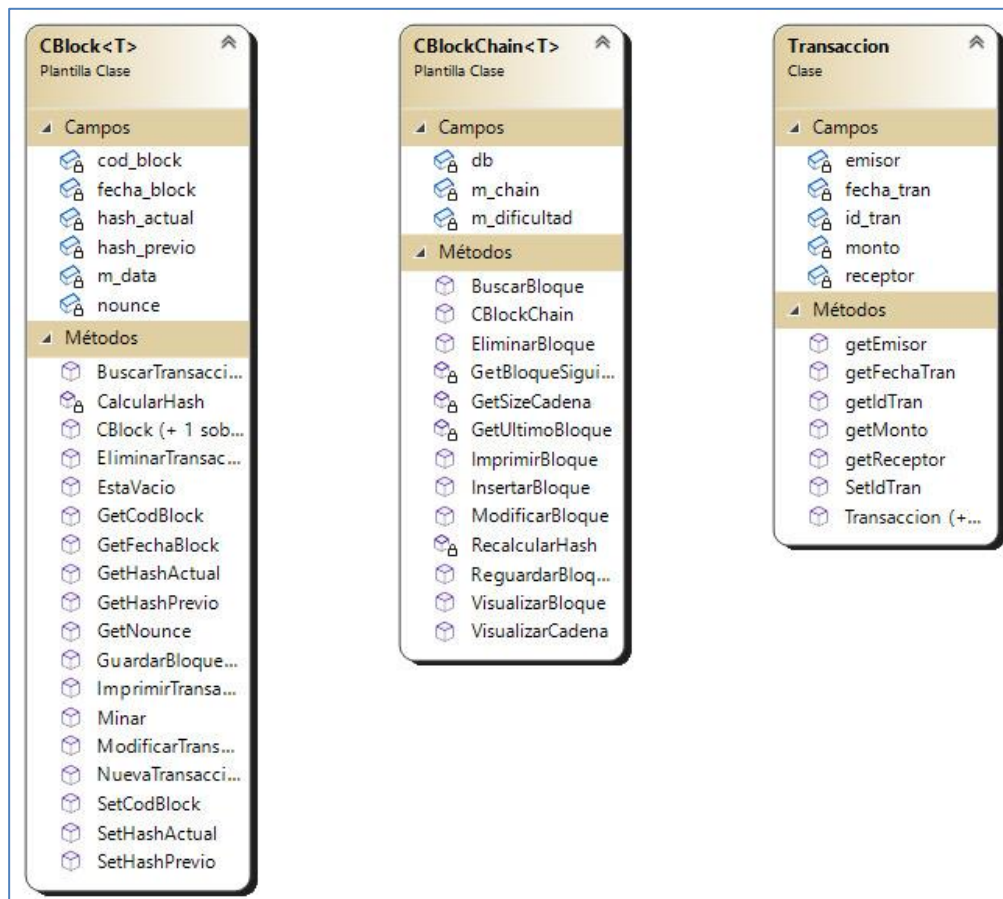


Figura 9. Estructura de clases de la capa Blockchain.

- Explicación de la estructura de datos del Blockchain y la estrategia para asegurar la integridad de su contenido. Además, indicar cómo se implementó el **proof of work**.

1.3.1. Clase Block.h

Librería implementada para definir los métodos requeridos para gestionar el uso de los bloques que componen el Blockchain. A continuación, se detallan estos:

Método	Descripción
string CalcularHash()	Método que verifica si el array circular donde se alojan los datos está vacío o no, y si contiene datos, hace un recorrido con un iterador para obtener los datos de la transacción y hace uso del método SHA256 para generar un hash.
void Minar(int difficulty)	Método para minar el bloque mediante un bucle que incrementa el valor de nonce y recalcula el valor del hash a partir de este con el método CalcularHash().
void NuevaTransaccion(T& data)	Método para verificar si el tamaño del array circular que almacena los datos no sobrepasa el máximo de transacciones por bloque y partir de esto hace uso del método push_back para agregar una nueva transacción.
void ModificarTransaccion(T data)	Método para modificar una transacción de acuerdo a la búsqueda de su id con un recorrido que se hace mediante una iteración en el array circular m_data
bool BuscarTransaccion(int IdTran)	Método para buscar una transacción, de acuerdo a su id, mediante un recorrido por el array circular m_data con una iteración.
void EliminarTransaccion(int idTran)	Método para eliminar una transacción de acuerdo a la búsqueda de su id con un recorrido que se hace mediante una iteración en el array circular m_data
void ImprimirTransacciones()	Método que muestra los datos de todas las transacciones almacenadas en los bloques mediante un recorrido que se hace con una iteración en el array circular m_data.

bool EstaVacio()	Método para determinar si el array circular m_data está vacío o no.
void GuardarBloqueDB(Database &db)	Método que almacena todos los registros en el array circular m_data mediante el recorrido que se hace con una iteración, de tal manera que se guarda un bloque en la base de datos.

1.3.2. Clase Blockchain.h

Clase implementada para definir los métodos de inserción, modificación, eliminación y visualización del bloque hacia la cadena de bloques. A continuación se detallan.

Metodo	Descripcion	Entrada	Salida
void InsertarBloque(T& nuevoBloque)	Metodo para agregar un nuevo bloque al blockchain (cadena de bloques). Se le asigna un código y el hash previo, se invoca al metodo Minar() para el calculo del hash actual, usamos el metodo push_back(T data) de nuestra estructura de datos Lista Circular Doblemente enlazada y finalmente guardamos el bloque en BD.	Bloque Nuevo	Agregar el nuevo bloque
T BuscarBloque(int IdBloque);	Metodo para buscar el bloque por el codigo.	Codigo del Bloque	Bloque buscado
void ModificarBloque(T nuevoBloque);	Metodo que actualiza la informacion del bloque y realiza lel recalcu lo del hash del bloque y sus subsiguientes.	Bloque modificado	Recalculo de los hash del bloque y los bloques dependientes
void VisualizarBloque(int IdBloque);	Metodo para visualizar la informacion de un bloque en especifico.	Codigo del Bloque	Hash previo, hash actual, nonce, fecha bloque y la informacion de cada una de sus transacciones
void VisualizarCadena();	Metodo para visualizar la informacion de todos los bloques.		El listado de bloques con su respectiva informacion
void RecalcularHash(T bloque)	Metodo que realiza el recalcu lo del bloque y los bloques subsiguientes para actualizar el hash actual y el hash previo de cada bloque.	Bloque	Hash actualizado del bloque y los bloques subsiguientes
CBlockChain()	Metodo constructor que asigna la dificultad de la prueba de trabajo y genera el bloque genesis.	Dificultad de la prueba de trabajo	Bloque genesis agregado al blockchain

Proof of work (Prueba de Trabajo): La implementación del proof of work inicia con el constructor CBlockChain<T>::CBlockChain(), que establece la dificultad de prueba de trabajo (m_dificultad) en el valor de DIFICULTAD_PRUEBA_DE_TRABAJO. Esta dificultad determina cuántos ceros consecutivos se requieren al principio del hash de un bloque para que se considere válido. Cuanto mayor sea la dificultad, más cálculos computacionales se requerirán para encontrar un hash válido.

Luego, se crea un objeto **BloqueGenesis** de tipo T, que representa el bloque génesis de la cadena. A continuación, se llama al método Minar() de la clase Block.h, pasando la dificultad de prueba de trabajo como argumento (this->m_dificultad).

En el método **Minar**(int difficulty) de la clase Block.h, se realiza el proceso de minado. El minado implica encontrar un hash que cumpla con ciertas condiciones, como tener una cierta cantidad de ceros consecutivos al principio.

El proceso de minado se realiza en un bucle while (true) que continúa hasta que se encuentre un hash válido. En cada iteración del bucle, se incrementa un valor llamado nonce, y se calcula el hash actual del bloque llamando al método CalcularHash(). Luego, se verifica si el hash actual cumple con la dificultad de prueba de trabajo establecida, es decir, si tiene los ceros consecutivos requeridos al principio.

Si se encuentra un hash válido, se sale del bucle y se muestra el mensaje "Bloque minado" junto con el código del bloque y el hash actual.

```

// Constructor
template<class T>
CBlockChain<T>::CBlockChain() {
    this->m_dificultad = DIFICULTAD_PRUEBA_DE_TRABAJO;
    T BloqueGenesis;
    BloqueGenesis.Minar(this->m_dificultad);
    this->m_chain.push_back(BloqueGenesis); // El BlockChain nace con el bloque Génesis
}

// Metodo Minar
template<class T>
void CBlock<T>::Minar(int dificultad) {
    char* cstr = new char[dificultad + 1];

    for (uint32_t i = 0; i < dificultad; ++i) {
        cstr[i] = '0';
    }
    cstr[dificultad] = '\0';

    string str(cstr);

    while (true) {
        this->nounce++;
        this->hash_actual = CalcularHash();

        if (this->hash_actual.substr(0, dificultad) == str) {
            break;
        }
    }
    if(this->cod_block != 0)
        cout << "Bloque " << this->cod_block << " minado: " << this->hash_actual <<
endl;
}

```

Así mismo definimos el análisis de la complejidad de los métodos inserción y búsqueda.

Metodo	Codigo	Analisis de Complejidad
void InsertarBloque(T& nuevoBloque)	<pre> template<class T> void CBlockChain<T>::InsertarBloque(T& nuevoBloque) { nuevoBloque.SetCodBlock(GetUltimoBloque().GetCodBlock() + 1); nuevoBloque.SetHashPrevio(GetUltimoBloque().GetHashActual()); nuevoBloque.Minar(this->m_dificultad); this->m_chain.push_back(nuevoBloque); nuevoBloque.GuardarBloqueDB(db); } </pre>	La complejidad del metodo InsertarBloque es constante $O(1)$
T BuscarBloque(int IdBloque);	<pre> template<class T> T CBlockChain<T>::BuscarBloque(int IdBloque) { for (typename CircularDoubleList<T>::iterator it = m_chain.begin(); it != m_chain.end(); ++it) { if ((*it).GetCodBlock() == IdBloque) { m_chain.SetCurrent(it.GetCurrent()); return *it; } } T BloqueVacio; return BloqueVacio; } </pre>	El código realiza un bucle lineal para buscar un bloque en la lista m_chain. La complejidad total del código es $O(N)$, donde N es el número de elementos en la lista m_chain

1.3.3. Clase Transaccion.h

Clase implementada para definir los datos de la transacción tales como: Código de la transacción, monto a transferir, Emisor (usuario que realiza la transferencia), Receptor (Usuario que recibe la transferencia) y Fecha de la transacción. Está definida por los get() y set() de cada campo.

1.4. Capa Vista

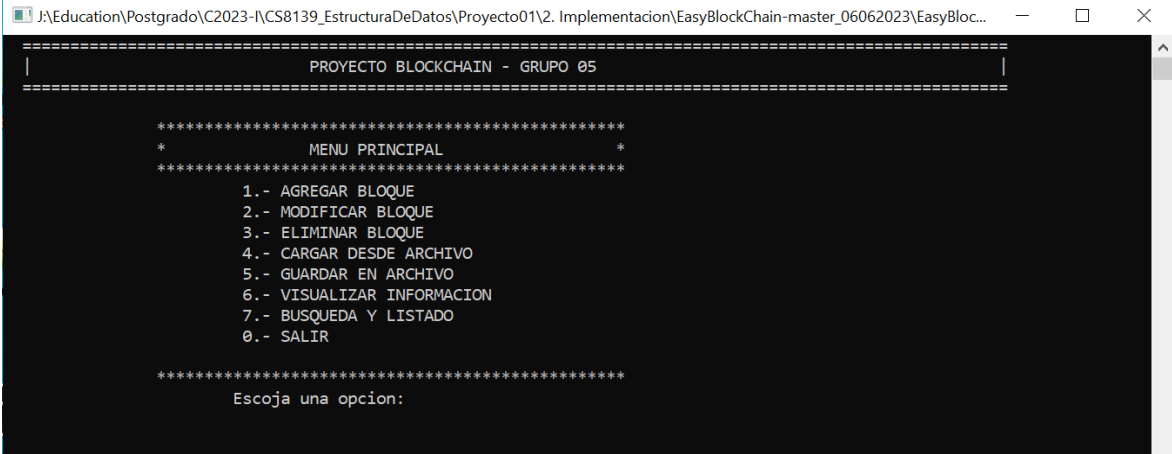
La vista se ha trabajado desde la misma consola, y lo hemos estructurado con un menú de opciones, para ello se ha generado la clase VistaBlockchain.h encargada de la gestión de las funcionalidades.

1.4.1. Clase VistaBlockchain.h

Clase implementada para definir los métodos requeridos para la Vista de la aplicación, a continuación se detallan.

Metodo	Descripcion
void menuPrincipal(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que lista las funcionalidades del proyecto
void opcionAgregarBloque(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para Insertar el bloque e insertar transacciones
Transaccion ingresarNuevaTransaccion();	Metodo que solicita los datos de cada transaccion que se desea ingresar en el bloque.
void mostrarTransaccion(Transaccion trx);	Metodo para mostrar los datos de la transaccion
void opcionModificarBloque(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo con las subopciones de la opcion Modificar Bloque
void supOpcionNuevaTransaccionPorBloque(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para Insertar nueva transaccion a un bloque ya existente
void subOpcionModificarTransaccion(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para modificar una transaccion de un bloque ya existente
void subOpcionEliminarTransaccion(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para eliminar una transaccion determinada de un bloque ya existente
void opcionEliminarBloque();	Metodo que llama a los metodos para eliminar un bloque ya existente
void opcionCargarDesdeArchivo();	Metodo que llama a los metodos para cargar bloques de un archivo csv
void opcionGuardarEnArchivo();	Metodo que llama a los metodos para guardar los bloques generados en un archivo csv
void opcionVisualizar(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo con las subopciones de la opcion Visualizar
void subOpcionVisualizarBloque(CBlockchain<CBlock<Transaccion>>& cadena);	Metodo que llama al metodo para visualizar un bloque determinado
void opcionBusqueda();	Metodo con las subopciones de la opcion Busqueda y listado

menuPrincipal En la figura 1 se puede visualizar el menú principal.



```
=====
|                                PROYECTO BLOCKCHAIN - GRUPO 05                                |
|=====|
*****
*                                MENU PRINCIPAL                                *
*****
1.- AGREGAR BLOQUE
2.- MODIFICAR BLOQUE
3.- ELIMINAR BLOQUE
4.- CARGAR DESDE ARCHIVO
5.- GUARDAR EN ARCHIVO
6.- VISUALIZAR INFORMACION
7.- BUSQUEDA Y LISTADO
0.- SALIR

*****
Escoja una opcion:
```

Figura 1. Menú Principal.

▪ void opcionAgregarBloque(CBlockchain<CBlock<Transaccion>>& cadena);

- **Transaccion ingresarNuevaTransaccion();**
- **void mostrarTransaccion(Transaccion trx);**

Se usan estos métodos tres métodos para el acceso a la opción 1. Agregar Bloque. En la figura 2 se visualiza el ingreso de los datos de una primera transacción.

```

*****
                          AGREGAR BLOQUE
*****

                          NUEVA TRANSACCION
-----
- Ingrese Monto      : 2000
- Ingrese Emisor     : Miguel
- Ingrese Receptor   : Judith

                          INFORMACION DE LA TRANSACCION
-----
ID | Emisor | Receptor | Monto | Fecha Trx
-----
1  | Miguel | Judith  | 2000  | 2023/06/07 00:29
-----

Desea ingresar nueva transaccion? Si (S) / No (N) :
  
```

Figura 2. Opción 1. Agregar Bloque.

En la figura 3 se visualiza el ingreso de más transacciones, la confirmación del minado y finalmente el bloque creado con los datos de cada transacción ingresada.

```

*****
                          AGREGAR BLOQUE
*****

                          NUEVA TRANSACCION
-----
- Ingrese Monto      : 4000
- Ingrese Emisor     : Juan
- Ingrese Receptor   : Carlos

                          INFORMACION DE LA TRANSACCION
-----
ID | Emisor | Receptor | Monto | Fecha Trx
-----
2  | Juan   | Carlos  | 4000  | 2023/06/07 00:39
-----

Desea ingresar nueva transaccion? Si (S) / No (N) : n
Desea minar el Bloque? Si (S) / No (N) : s

Bloque [1] minado: 000fc80927f417694821ce69fb21e315b2de5398d74ed7b9f69815b16fc3d7cd

| BLOQUE [1] |
-----
Hash       : 000fc80927f417694821ce69fb21e315b2de5398d74ed7b9f69815b16fc3d7cd
Hash Previo : 0008fd5c4d08848411220115f6e6f45ba583a696cae0c2c5e222695e59a7f699
Nonce      : 6827
Fecha Bloque : 2023/06/07 00:38
-----
ID Trx | Emisor | Receptor | Monto | Fecha Trx
-----
1      | Miguel | Judith  | 2000  | 2023/06/07 00:39
2      | Juan   | Carlos  | 4000  | 2023/06/07 00:39
-----

Presione una tecla para continuar . . .
  
```

Figura 3. Bloque minado y creado.

- **void opcionModificarBloque(CBlockchain<CBlock<Transaccion>>& cadena);**

En la siguiente figura se muestra las sub opciones de la opción Modificar Bloque

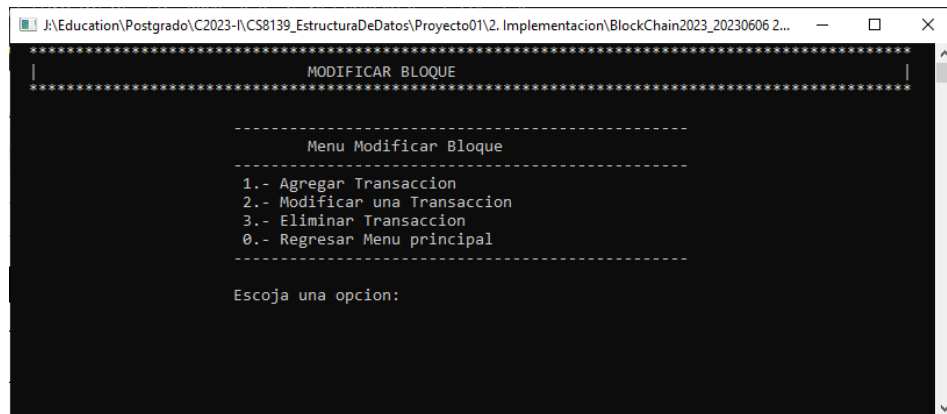
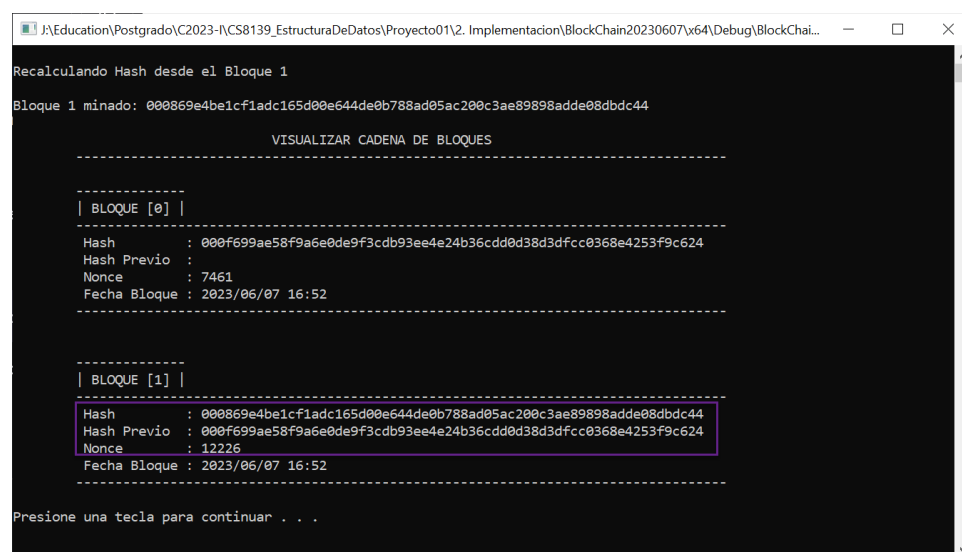
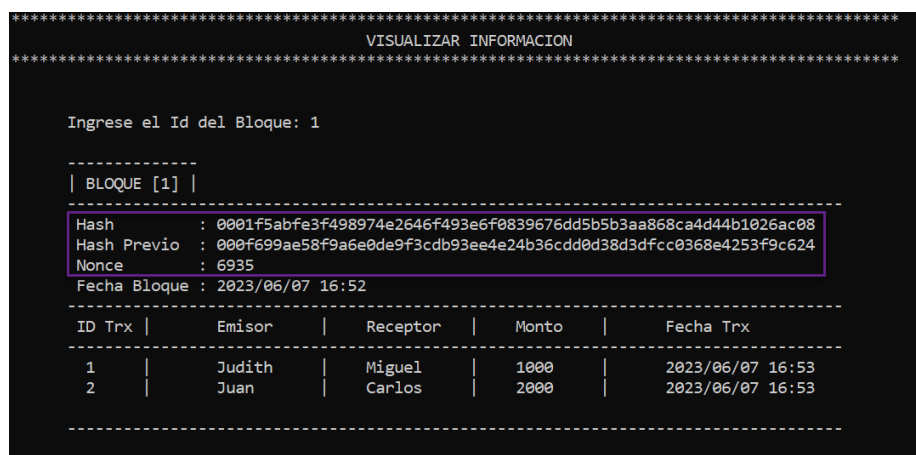


Figura 4. Submenú de Modificar Bloque.

- **void supOpcionNuevaTransaccionPorBloque(CBlockChain<CBlock<Transaccion>>& cadena);**



- **void subOpcionModificarTransaccion(CBlockChain<CBlock<Transaccion>>& cadena);**
- **void subOpcionEliminarTransaccion(CBlockChain<CBlock<Transaccion>> & cadena);**
- **void opcionEliminarBloque();**

- **void opcionVisualizar(CBlockchain<CBlock<Transaccion>>& cadena);**

```

J:\Education\Postgrado\C2023-I\CS8139_EstructuraDeDatos\Proyecto01\2. Implementacion\BlockChain20230607\64\Debug\BlockChai...
*****
VISUALIZAR INFORMACION
*****

-----
Menu Visualizar
-----
1.- Visualizar un Bloque
2.- Visualizar los Bloques
0.- Regresar Menu principal
-----

Escoja una opcion:

J:\Education\Postgrado\C2023-I\CS8139_EstructuraDeDatos\Proyecto01\2. Implementacion\BlockChain20230607\64\Debug\BlockChai...
VISUALIZAR CADENA DE BLOQUES

-----
| BLOQUE [0] |
-----
Hash       : 000f699ae58f9a6e0de9f3cdb93ee4e24b36cdd0d38d3dfcc0368e4253f9c624
Hash Previo : 000f699ae58f9a6e0de9f3cdb93ee4e24b36cdd0d38d3dfcc0368e4253f9c624
Nonce      : 7461
Fecha Bloque : 2023/06/07 16:52
-----

-----
| BLOQUE [1] |
-----
Hash       : 0008c34eefaf21b59dadfcdccf65a7d1d69df9aed2698db0262c3afaFada6deb
Hash Previo : 000f699ae58f9a6e0de9f3cdb93ee4e24b36cdd0d38d3dfcc0368e4253f9c624
Nonce      : 17734
Fecha Bloque : 2023/06/07 16:52
-----

-----
| BLOQUE [2] |
-----
Hash       : 000599f2d811c87f4cabf13adff1b91001e3b516c05b5fe3af8d8e14d4578a92
Hash Previo : 0008c34eefaf21b59dadfcdccf65a7d1d69df9aed2698db0262c3afaFada6deb
Nonce      : 7443
Fecha Bloque : 2023/06/07 17:02
-----

Presione una tecla para continuar . . .

```

- **void subOpcionVisualizarBloque(CBlockchain<CBlock<Transaccion>>& cadena);**

```

*****
VISUALIZAR INFORMACION
*****

Ingrese el Id del Bloque: 1

-----
| BLOQUE [1] |
-----
Hash       : 0001f5abfe3f498974e2646f493e6f0839676dd5b5b3aa868ca4d44b1026ac08
Hash Previo : 000f699ae58f9a6e0de9f3cdb93ee4e24b36cdd0d38d3dfcc0368e4253f9c624
Nonce      : 6935
Fecha Bloque : 2023/06/07 16:52
-----

ID Trx | Emisor | Receptor | Monto | Fecha Trx
-----
1 | Judith | Miguel | 1000 | 2023/06/07 16:53
2 | Juan | Carlos | 2000 | 2023/06/07 16:53
-----

```

■ **CARGAR DESDE ARCHIVO:**

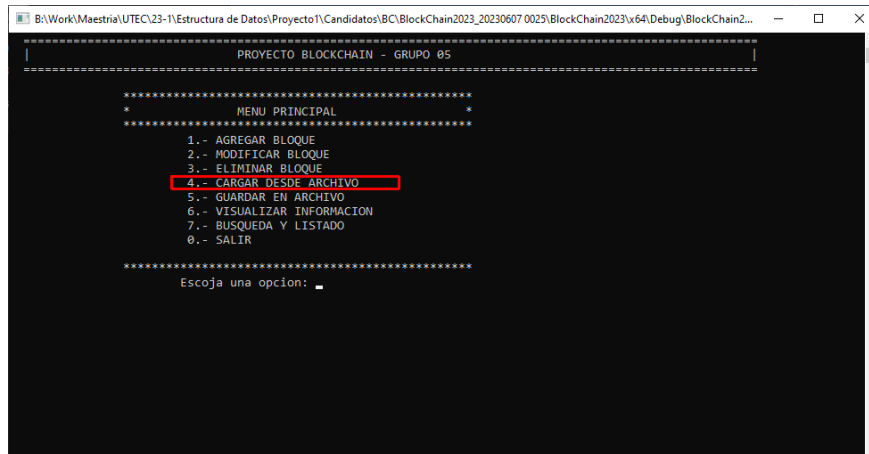
Se ha considerado un archivo con un Blockchain de 50 bloques y cada uno con sus transacciones:

Para esto tenemos un archivo en formato texto CSV llamado `dataset_grande_v1.csv` que se previsualiza en Excel:

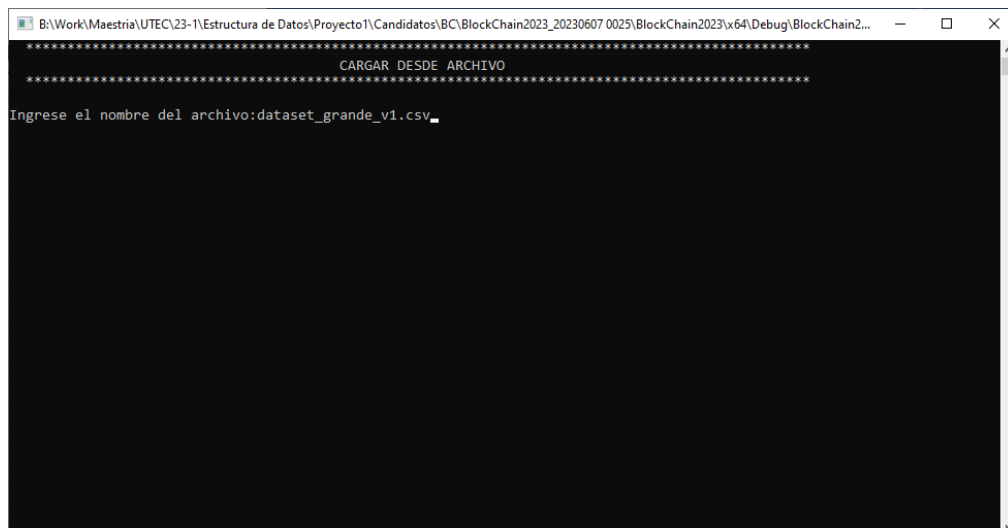
[illegible]

Fig. 11 – Datos importados

En el menú usamos la opción 4:



Ingresamos el nombre del archivo: dataset_grande_v1.csv



Aquí se obtiene el resumen del archivo:

```
El archivo dataset_grande_v1.csv tiene 10 columnas
El archivo dataset_grande_v1.csv tiene 551 filas
Presione una tecla para continuar . . .
```

Opción 7 para consultar los datos:

```
B:\Work\Maestria\UTEC\23-1\Estructura de Datos\Proyecto1\Candidatos\BC\BlockChain2023_20230607 0025\BlockChain2023\x64\Debug\BlockChain2...
=====
|                                PROYECTO BLOCKCHAIN - GRUPO 05                                |
=====
*          MENU PRINCIPAL          *
*****
1.- AGREGAR BLOQUE
2.- MODIFICAR BLOQUE
3.- ELIMINAR BLOQUE
4.- CARGAR DESDE ARCHIVO
5.- GUARDAR EN ARCHIVO
6.- VISUALIZAR INFORMACION
7.- BUSQUEDA Y LISTADO
0.- SALIR

*****
Escoja una opcion: _
```

Elegimos la opción 1. Buscar Bloque por ID de Bloque e ID de Transacción:

```
B:\Work\Maestria\UTEC\23-1\Estructura de Datos\Proyecto1\Candidatos\BC\BlockChain2023_20230607 0025\BlockChain2023\x64\Debug\BlockC...
=====
|                                BUSQUEDA Y LISTADO                                |
=====
-----
Menu Buscar y Listar
-----
1.- Buscar bloque por ID Bloque e ID Transacci n
2.- Listar ordenado por Receptor
3.- Listar ordenado por Monto
4.- Listar ordenado por Fecha de Transacci n
0.- Regresar Menu principal
-----
Escoja una opcion: _
```

Consultando por el Bloque 1 e ID de transacción 5, se tiene:

```
B:\Work\Maestria\UTEC\23-1\Estructura de Datos\Proyecto1\Candidatos\BC\BlockChain2023_20230607 0025\BlockChain2023\x64\Debug\BlockChain2...
Buscar bloque por ID Bloque e ID Transacci n
Ingrese el Id del Bloque: 1

Ingrese el Id de la Transacci n: 5

Consulta de Registro: cod_bloque=1, id_tran=5, desde=Emma, hasta=Liam, fecha transacci n=2011/04/12 11:50
Presione una tecla para continuar . . . _
```

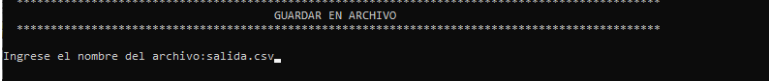
Ello se puede corroborar con la figura 11 (Datos importados y previsualizados con Excel)

Elegimos la opción 2. Listar ordenado por Receptor:

```
-----
Menu Buscar y Listar
-----
1.- Buscar bloque por ID Bloque e ID Transacci n
2.- Listar ordenado por Receptor
3.- Listar ordenado por Monto
4.- Listar ordenado por Fecha de Transacci n
0.- Regresar Menu principal
-----
Escoja una opcion: _
```


[illegible]

■ **GUARDAR EN ARCHIVO:**



```
B:\Work\Maestria\UTEC\23-1\Estructura de Datos\Proyecto1\Candidatos\BC\BlockChain2023_20230607 0025\BlockChain2023\64\Debug\BlockChain2023_20230607 0025>
*****
GUARDAR EN ARCHIVO
*****
Ingrese el nombre del archivo:salida.csv_
```

Nombre	Fecha de modificación	Tipo	Tamaño
salida.csv	8/06/2023 08:06	Archivo de valores...	98 KB
	8/06/2023 07:40	Per-User Project O...	1 KB
	7/06/2023 17:20	Archivo WinRAR Z...	59 KB

« Proyecto1 » Candidatos » BC » BlockChain2023_20230607 0025

1.4.2. Clase *Utiles.h*

Clase implementada para colocar métodos utilitarios y que serán usadas por las diferentes clases. Está definida con los siguientes métodos.

Metodo	Descripción	Entrada	Salida
string ObtieneFechaActual()	Metodo que hace la conversion del formato de la fecha	La fecha en formato time	La fecha en formato YYYY/MM/DD HH:MM

2. Conclusiones

- ✓ Implementación de una estructura de datos de cadena de bloques: Se logró implementar la clase "Blockchain.h", "Block.h" y "Transaccion.h", que permiten el almacenamiento seguro y la gestión de los bloques de datos en una cadena de bloques.
- ✓ Uso de las estructuras de datos vista en clases: Arreglo Circular, Lista circular doblemente enlazada, BST, HashMap.
- ✓ Interfaz de usuario y visualización de la cadena de bloques: La clase "VistaBlockchain.h" proporciona una interfaz de usuario amigable que permite a los usuarios realizar transacciones bancarias y visualizar la cadena de bloques.
- ✓ Seguridad y encriptación de datos: La clase "Crypto.h" junto con las implementaciones de "SHA256.cpp" y "SHA256.h" permiten la encriptación y protección de los datos almacenados en la cadena de bloques. Esto garantiza la confidencialidad e integridad de las transacciones bancarias almacenadas en la aplicación.

En general, el proyecto logra su objetivo principal de desarrollar una aplicación transaccional de almacenamiento seguro de datos utilizando una cadena de bloques y diferentes estructuras de datos para indexar los registros.

3. Referencias bibliográficas

- [1] Tapscott, D., & Tapscott, A. (2016). Blockchain revolution: How the technology behind bitcoin is changing money, business, and the world. Penguin.
- [2] Ekblaw, A., Azaria, A., Halamka, J. D., & Lippman, A. (2016). A case study for blockchain in healthcare: "MedRec" prototype for electronic health records and medical research data. In Proceedings of IEEE Open & Big Data Conference (pp. 1-8). IEEE.
- [3] Daniel García (2022) - Trabajo Fin de Grado - Implementación Alternativa de una Tabla Hash en C++ - Universidad Politécnica de Madrid.

4. Anexos

Para el desarrollo de las actividades del grupo se han usado las herramientas de trello, google meet y Drive