ALGORITMOS Y ESTRUCTURA DE DATOS

TP Integrador Bitcoin Fase I. Árboles.

Requerimiento

El programa posee una interfaz gráfica de usuario o **GUI** y le permite al usuario seleccionar un archivo JSON ubicado en el sistema de archivos, del cual deberá leer un arreglo de bloques o **Blockchain**. Deberá permitir ver una representación gráfica del **Merkle Tree** de cada bloque, navegar los bloques y realizar operaciones sencillas.

Nota: *La cátedra proveerá un archivo JSON de prueba con la Blockchain*

Funcionamiento del programa

El usuario ingresa la dirección del archivo o **path** de una carpeta en la GUI. El programa busca todos los archivos JSON con un arreglo de bloques dentro de ese path y le permite al usuario:

1. Elegir uno de los archivos.
2. Ver todos los bloques dentro del archivo.
3. Seleccionar / Deseleccionar un bloque.
4. Realizar las siguientes acciones sobre un bloque seleccionado:
    a. Ver la información del bloque:
        i. ID del bloque.
        ii. ID del bloque anterior.
        iii. Cantidad de transacciones.
        iv. Número de bloque.
        v. Nonce.
    b. Calcular el **Merkle Root**.
    c. Validar el **Merkle Root** del bloque (comparando el calculado con el del bloque).
    d. Ver el **Merkle Tree** (una representación gráfica del **Merkle Tree**).

Nota: *En los anexos se encuentran las descripciones necesarias para los algoritmos y las estructuras de datos a emplear.*

Recomendaciones

- Para la implementación de la GUI (**Graphic User Interface**) utilizar la librería de C++ **Dear ImGui**, pueden encontrar el repositorio aquí
- Pueden encontrar en este repositorio de la cátedra, una **breve introducción** al uso de la librería **Dear ImGui**

- En el repositorio de la cátedra, también encontraran un ejemplo útil donde se muestra como agregar múltiples widgets en una misma ventana en partes separadas del código

Anexo 1 – Estructuras de datos

Estructura JSON de un Bloque

```
{
    "tx": array,                // Arreglo de transacciones
    "nTx": number,              // Cantidad de transacciones
    "height": number,           // Posición del Bloque dentro de la Blockchain
    "nonce": number,
    "blockid": string,          // Identificador único del Bloque
    "previousblockid": string,  // Identificador único del Bloque que le precede
    "merkleroot”: string        // Identificador del Merkle Root
}
```

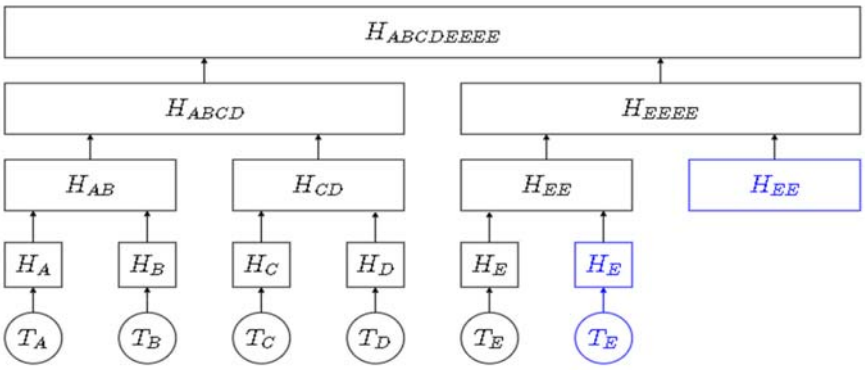
Estructura JSON de una Transacción

```
{
    "txid": "string",                   // Identificador único de la Transacción
    "nTxin": number,                    // Cantidad de Entradas
    "vin": [                            // Registro de Entradas de la Transacción
        {
            "blockid": "string",        // Bloque donde está la Entrada
            "txid": "string",           // Transacción donde está la Entrada
            "signature": "string",
            "outputIndex": number       // Salida utilizada como Entrada
        },
             (...)
    ],
    "nTxout": number,                   // Cantidad de Salidas
    "vout": [                           // Registro de Salidas de la Transacción
       {
            "publicid": "string",       // A quién se transfiere
            "amount": number            // Cantidad transferida
       },
             (...)
    ]
}
```

Anexo 2 – Algoritmos

Generación del Merkle Tree

El siguiente algoritmo de generación del **Merkle Tree** se aplica sobre un **Bloque**, para lo cual se asume la estructura de datos de los mismos, según se describe en Estructura JSON de un Bloque.



1. Generamos un identificador para cada transacción del bloque que llamaremos **id** que se obtiene concatenando los campos **txid** de cada una de las entradas de la transacción registradas en **vin**, respetando el orden de aparición en tal arreglo.
2. Generamos un nuevo identificador para cada transacción del bloque que llamaremos **nid**, que se obtiene de aplicar la función **generateID** sobre el **id**. La función se encuentra provista en Generador de IDs.
3. El **nid** de cada transacción se convierte a una representación denominada **Hex Coded ASCII**. Es decir, un string cuyos caracteres ASCII son la representación hexadecimal del identificador numérico **nid**. Esta nueva representación la llamaremos **nidstr**.

Si ID = 249346712_{10} = $0EDCBA98_{16}$ → HexCodedASCII = '0EDCBA98'

4. Se verifica que strlen(**nidstr**) = 8.
5. El conjunto o colección de **nidstr** para todas las transacciones son las hojas del árbol binario al cual denominaremos **Merkle Tree**.
6. Se verifica que la cantidad de **nidstr** en el conjunto sea par, y en caso de ser impar, se debe agregar a la colección una hoja que resulta de copiar el último elemento.
7. Se toma el conjunto de **nidstr** de a pares, respetando el orden en el que se encuentran las transacciones dentro del Bloque
8. Se concatenan los **nidstr** para cada par tomado, y sobre eso se aplica la función **generateID** y se convierte a su representación **Hex Coded ASCII**. Esto nos da como resultado, el **nodo padre** de cada par tomado.
9. El conjunto **nodos padres** conforma el **nivel superior** del **Merkle Tree**.
10. Se repite el algoritmo desde el paso 5, pero ahora utilizando los **nodos padres** como el conjunto o colección. Si la cantidad de **nodos padres** es uno, entonces se pasa al paso final.
11. El algoritmo termina cuando queda un único nodo. Este nodo se conoce como **Merkle Root**.

Anexo 3 – Funciones

Generador de IDs

La función **generateID** se utiliza para generar un nuevo identificador numérico. Es importante tener en cuenta que, esta función es provisoria y será modificada en futuras etapas del proyecto, y no es relevante la comprensión de su funcionamiento o propósito.

```
static unsigned int generateID (unsigned char *str)
{
	unsigned int ID = 0;
	int c;
	while (c = *str++)
	{
		ID = c + (ID << 6) + (ID << 16) - ID;
	}
	return ID;
}
```