



95.12 - ALGORITMOS Y PROGRAMACION II

Trabajo práctico 1: algoritmos y estructuras de datos

2^{do} CUATRIMESTRE DE 2020

Alumnos:

Burna Lucas	99608	lburna@fi.uba.ar
Perczyk Francisco	99631	fperczyk@fi.uba.ar
Sobico Carla	99738	csobico@fi.uba.ar

Entregas realizadas:
03/12/2020

Índice

1	Objetivos	2
2	Desarrollo	2
2.1	Diseño	2
2.2	Clases, estructuras y funciones	3
2.2.1	Clase list	3
2.2.2	Clase block	4
2.2.3	Comandos	5
2.3	Pruebas	6
2.3.1	Desempeño	7
3	Conclusiones	11
4	Referencias	11
5	Códigos fuente	12
5.1	Códigos provistos por la cátedra	63
6	Enunciado	71

1 Objetivos

El objetivo de este trabajo es desarrollar una algochain compuesta por los bloques creados anteriormente. En esta algochain se podrán realizar transacciones entre clientes y la creación de los mismos mediante de una interfaz operativa.

2 Desarrollo

Se desarrollan una serie de comandos para realizar las distintas operaciones sobre la algochain, estos comandos son

Comando	Descripción	Retorno	Argumentos
init	Genera un bloque genesis para inicializar la algochain	Hash del bloque génesis	usuario-valor-dificultad
transfer	Genera una nueva transacción en la que el usuario transfiere fondos a una cantidad N de usuarios.	Hash de la transacción	usuario-usuario destino-valor
mine	Ensambla y agrega a la algochain las transacciones de la mempool	Hash del bloque	dificultad
balance	Consulta el saldo disponible en la dirección del usuario	Saldo disponible	usuario
block	Consulta la información del bloque	Los campos del bloque	Hash id del bloque
txn	Consulta la información de la transacción	Los campos de la transacción	Hash id de la transacción
load	Lee la algochain serializada en un archivo	Hash del último bloque de la cadena	nombre del archivo
save	Guarda una copia de la algochain en un archivo	OK	nombre del archivo
exit program	Cierra el programa	-	-

Tabla 2.1: Caption

Existen dos tipos bloques especiales, el bloque génesis y mempool. El bloque génesis es el primer bloque de la Algochain, este bloque introduce un saldo inicial para un usuario. Este bloque tiene <prev block> con el hash nulo y un único <input> y un único <output>, este <input> referencia a un <outpoint> nulo. La mempool es un espacio de memoria donde se alojarán las transacciones de los usuarios que aún no fueron cargadas a la algochain.

2.1 Diseño

Se optó diseñar la **Algochain** como una lista doblemente enlazada con punteros al primer nodo y al último, donde los nuevos bloques se agregan al final de la lista. Se toma esta decisión debido a la facilidad para minar nuevos bloques cargados de transacciones en la mencionada lista.

En cuanto a las transacciones sin confirmar (no *minadas*), se guardarán en la **mempool**, se decide realizarla como un tipo «block» pues luego, para minarlo en la algochain, sólo basta con crear el header con la dificultad deseada y el bloque previo de la algochain.

Dado que se estará modificando la algochain y la lista de usuarios constantemente, se implementó un método de la clase «list» que pudiera buscar tanto el dato de un nodo, como un usuario o bloque enteros, también sobrecargada para buscar ciertos datos dentro de estos objetos más complejos. Se utiliza además en la sobrecarga del método *find* un diccionario que utiliza ciertas

funciones denominadas *finders* de acuerdo con lo que se desee buscar dentro del nodo de la lista.

Se tiene una lista auxiliar de clase «user» donde se almacenan los usuarios creados con su balance y transacciones realizadas. Esta lista será útil para el comando balance y para verificar si un usuario tienen fondos suficientes para realizar una transacción y de donde surgen estos fondos.

Para procesar las directivas ingresadas por línea de comandos una vez iniciado el programa se crea un diccionario mediante el cual se relacionan los mencionados comandos con las funciones que los procesan, mediante punteros a función. Por lo tanto, luego de procesar la entrada se llama a la función correspondiente con estos punteros.

Para el diseño se tuvo en cuenta las interrelaciones que podrían tener las clases y estructuras que poseen las variables globales representadas por la mempool, la lista de usuarios y la lista que conforma la algochain, y la clase nodo anidada a la clase lista. Dado que la lista debe poder modificar los atributos del nodo, se declara como *friend* dentro de la clase nodo. Las interrelaciones entre clases podrían generar problemas de inclusión entre archivos a la hora de compilar el programa, lo que llevó a la decisión de generar nuevos archivos que contengan a los prototipos de las funciones de cada clase, pudiendo de esta manera incluir los prototipos a utilizar en cada ocasión.

2.2 Clases, estructuras y funciones

Dado que el operador de salida se sobrecargó para la clase lista para poder imprimir por el flujo de salida deseado algún nodo o la lista en sí. Por lo tanto, todas las clases creadas para el TP0 también requirieron tener una sobrecarga del operador de salida, reemplazando en su uso a los métodos de clase *getAsString()* utilizados previamente, si bien se mantuvieron para mantener coherencia con el trabajo anterior.

El programa está compuesto por un main.cpp, cmdline.h, sha256 y block.h donde se definen las variables globales para poder manejar el flujo de entrada y salida y sus funciones de lectura. Además contiene un archivo tools.h con funciones auxiliares necesarias para algunos procedimientos, tales como: verificar si el atributo de un objeto es un número o un hash; o la conversión de hexadecimal a binario para el procesamiento de la dificultad especificada.

2.2.1 Clase list

Fue escrita en base a los códigos provistos por la cátedra. La clase lista está subdividida en dos archivos: uno con sus declaraciones y otro con las implementaciones, en las cuales se utilizaron templates. Como se mencionó en la sección 2.1, se mantuvo esta decisión para todas las clases y para evitar problemas de inclusión, por la dependencia de archivos existente.

La clase «list» cuenta con los métodos básicos de una clase: constructores básico y a partir de otro objeto de la clase, y un destructor; además de los específicos de la clase, tales como *insert()* que agrega un nodo al principio de la lista; *append()* que agrega un nodo al final; *removeElement()* que elimina un elemento de la lista específico; *empty()* que verifica si la lista está vacía; *size()* que verifica el tamaño de la lista; *toString()* que devuelve específicamente la lista como una cadena de caracteres; *getFirstNode()* que obtiene el primer nodo de la lista; *getLastNode()* que obtiene el último nodo de la lista; *deleteFirstNode()* que elimina el primer nodo de la lista; *deleteLastNode()*; y finalmente *contains* que busca si la lista contiene el dato. Es necesario analizar los métodos *find()* en otro apartado. La forma básica de éste método

```
T find(const T& t)
```

simplemente busca el parámetro T en la lista; mientras que la sobrecarga de *find()* dada declarada como

```
string find(const string& ref,const string& d)
```

tiene un diccionario mediante el cual se accede a la función que se requiera. Encuentra el dato

'd' de tipo 'ref' en su última aparición en la lista, donde ref determina el puntero a función que se utilizará y "d.^{el} dato que se quiere buscar.

2.2.2 Clase block

A diferencia del diseño pensado inicialmente para un bloque, se decide usar el atributo *value* como una cadena de caracteres para evitar inconvenientes a la hora de calcular los hashes involucrados en la construcción y/o minado del bloque.

Por otra parte, se agregó un método a la clase «block» para añadir una transacción individual al bloque bajo tratamiento. Esto es particularmente útil al momento de procesar la mempool, puesto que se agregan transacciones constantemente a través del flujo de entrada.

```
void txnsArrRedim(const size_t n )
```

Nuevamente, como se mencionó en la sección 2.1, se sobrecarga el operador de salida para todas las clases. Además se agregaron algunos métodos sobrecargados para la construcción de ciertos atributos del bloque, como los inputs y outputs de una transacción como

```
string setTxOut(Array <string>, Array<string>)
```

```
bool setTxIn(Array<inpt>&)
```

cuyas funcionalidades son las de un constructor, basado en dos arreglos de cadenas para setTxOut(), que se corresponden con los atributos de addr y value respectivamente; a partir de un arreglo de inputs para setTxIn().

Otro cambio significativo realizado en esta clase es el introducido por el cálculo del *header*, más específicamente el txn hash, dado por los árboles de Merkle. El *hash* de este atributo se calcula como el *hash* recursivo de a pares a partir de los *hashes* iniciales de cada transacción, es decir: primero se calculan los *hashes* de cada transacción, llamados *hashes* de nivel 1. Tomando los 2 primeros se los concatena y se calcula un *hash* que pertenecerá al nivel 2. De esta manera se completan todos los *hashes* de nivel dos tomando *hashes* de nivel 1 de a pares concatenados. Siguiendo esta lógica se calculan los *hashes* de nivel superior hasta quedar sólo uno, que será el valor de txn hash. Un procedimiento a modo de ilustración se muestra en la figura ??

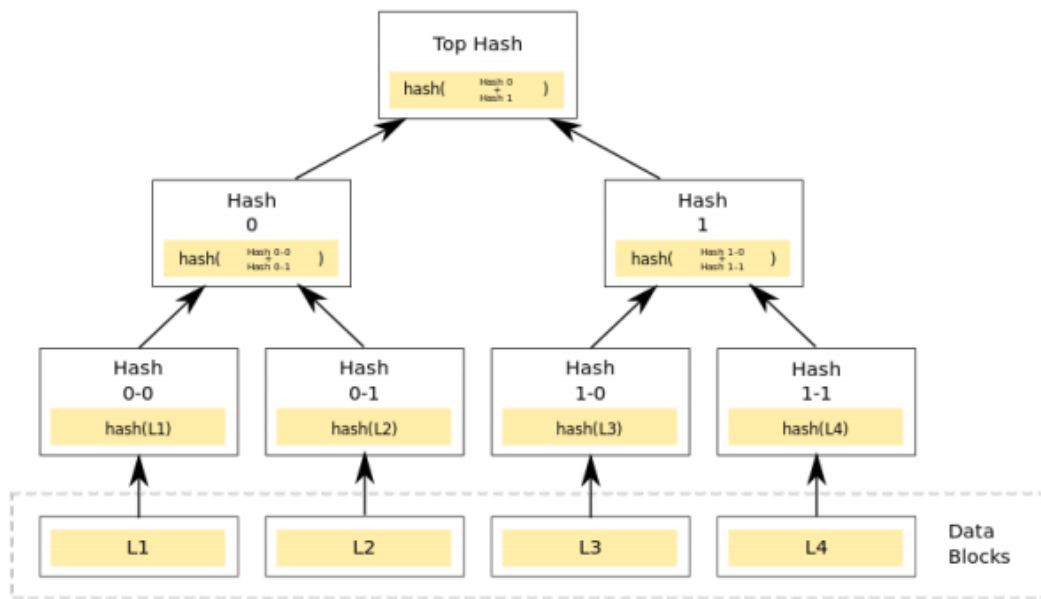


Figura 2.1: Ilustración de cálculo de un árbol de Merkle

Esta clase permite recorrer y buscar información de un usuario de manera eficiente, sin tener que buscar en la algochain y/o mempool las transacciones y el dinero disponible del usuario.

Cuenta con atributos que identifican al usuario mediante su nombre (atributo *name*); el dinero disponible que posee para realizar transacciones (atributo *balance*); y todas las transacciones en las que el usuario se vio involucrado como un output. Esto es: las transacciones en las que el usuario recibió dinero o tuvo UTXO's (atributo *transactions* implementado como una lista de txn).

Cuenta con los métodos fundamentales de clase: constructor base, constructor basado en una cadena de caracteres y destructor, además de *getters* y *setters* de sus atributos.

También se contempla el agregado de una transacción en la cual el usuario se ve involucrado, mediante el método

```
void addTxn(const txn)
```

que agrega la transacción como un nodo a la lista transactions.

Por otro lado se considera el caso de recibir mediante un archivo de texto las transacciones, por lo que se utiliza

```
void loadTxn(txn tran);
```

2.2.3 Comandos

cmdInit() En este comando se reciben tres argumentos para inicializar el bloque génesis. Se recibe el user, la cantidad de algocoins iniciales y la dificultad del minado. De estar inicializadas las listas algochain y users se reinician. Por último inicializa el *header* y el *body* de la manera determinada y se agrega a la algochain y el nuevo usuario al users.

cmdTransfer() Recibe por argumentos el usuario de origen y el usuario de destino con la cantidad de algocoins a transferir. Esta función verifica la existencia del usuario de origen y si tiene fondos suficientes para realizar la transacción. Luego carga a la lista de users el nuevo usuario.

cmdMine() Este comando recibe la dificultad con la que se minará el bloque y verifica que tenga el formato correcto. Luego arma el *header* de la mempool y lo agrega a la algochain.

cmdBalance() Se recibe el usuario (el nombre original sin *hashes* realizados), verifica si se encuentra en la lista de usuarios y devuelve el valor del balance del usuario.

cmdBlock() Recibe el *hash* del bloque que se busca, verifica que se encuentra y lo devuelve.

cmdTxn() Recibe el *hash* de la transacción, primero verifica si se encuentra en la algochain y luego en la mempool.

cmdLoad() Recibe el nombre del archivo a leer y verifica poder abrirlo y leer correctamente. Luego se valida que lo ingresado sea *hash* o *double* según corresponda. A medida que guarda los bloques en la algochain crea los nuevos usuarios y verifica que las transferencias se puedan realizar con los fondos y usuarios correspondientes.

cmdSave() Por argumentos recibe el nombre del archivo donde guardará la información, si no existe lo crea. Luego de esto imprime en este archivo la algochain completa.

2.3 Pruebas

Para compilar este programa se realizó el siguiente *Makefile*:

```
CCFLAGS= -Wall
CC= g++

all: algochain

algochain: main.o cmdline.o sha256.o
$(CC) $(CCFLAGS) -o algochain main.o cmdline.o sha256.o

main.o: main.cc cmdline.cc sha256.cpp Array.h block.h cmdline.h dictionary.h finders.h
      sha256.h tools.h user.h prototypeLista.h Lista.h main.h
$(CC) $(CCFLAGS) -c main.cc -o main.o

cmdline.o: cmdline.cc cmdline.h
$(CC) $(CCFLAGS) -c cmdline.cc -o cmdline.o

sha256.o: sha256.cpp sha256.h
$(CC) $(CCFLAGS) -c sha256.cpp -o sha256.o

clean:
$(RM) *.o algochain
```

Con la versión de g++: g++ (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0 Luego para correrlo por consola se escribe la siguiente línea

```
./algochain
```

Con el programa corriendo, se ingresan los comandos deseados.

Para obtener ayuda sobre los comandos disponibles del programa se escribe la siguiente línea

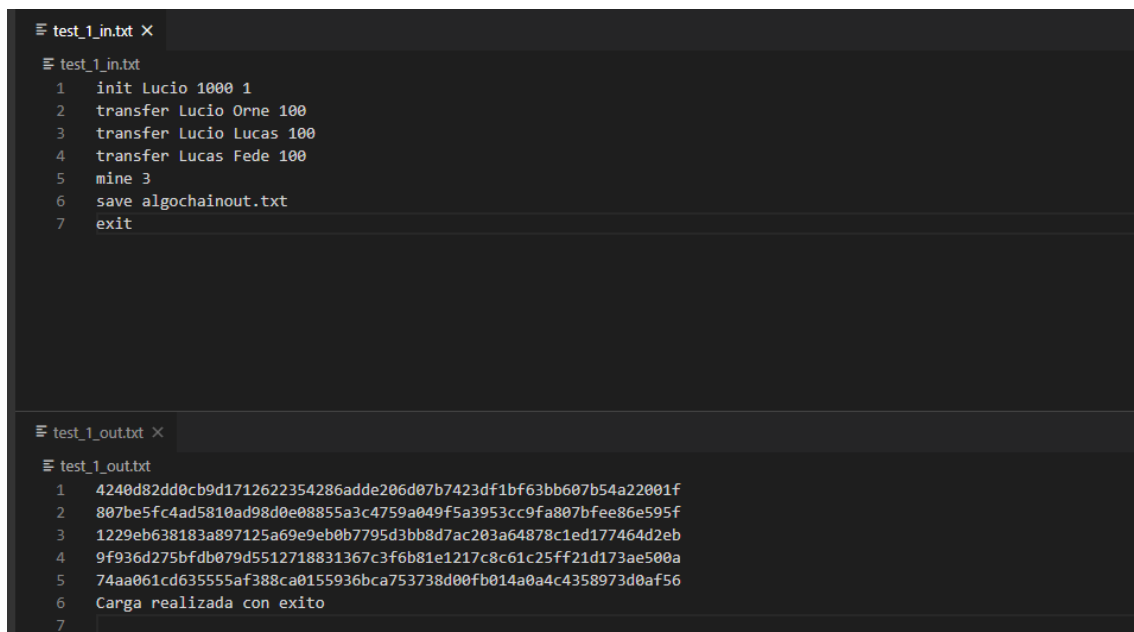
```
./algochain -h
```

Los argumentos que puede utilizar el programa no son obligatorios. Se añade la opción de ayuda mediante *-h*. Se puede especificar el flujo de entrada mediante un archivo de texto con el comando *-i*, así como el archivo de texto con el comando *-o*.

2.3.1 Desempeño

Se considera que cuando se realiza el comando *load* el archivo debe tener correcto los campos del *header*, es decir que el campo *prev block* se el *hash* del bloque anterior, y las demás verificaciones.

En las pruebas se especifica el flujo de entrada y salida mediante las opciones *-i* y *-o*.



```
test_1_in.txt X
test_1_in.txt
1  init Lucio 1000 1
2  transfer Lucio Orne 100
3  transfer Lucio Lucas 100
4  transfer Lucas Fede 100
5  mine 3
6  save algochainout.txt
7  exit

test_1_out.txt X
test_1_out.txt
1  4240d82dd0cb9d1712622354286adde206d07b7423df1bf63bb607b54a22001f
2  807be5fc4ad5810ad98d0e08855a3c4759a049f5a3953cc9fa807bfee86e595f
3  1229eb638183a897125a69e9eb0b7795d3bb8d7ac203a64878c1ed177464d2eb
4  9f936d275bfdb079d5512718831367c3f6b81e1217c8c61c25ff21d173ae500a
5  74aa061cd635555af388ca0155936bca753738d00fb014a0a4c4358973d0af56
6  Carga realizada con exito
7
```

Figura 2.2: Prueba de funcionamiento correcto del programa

En esta primer prueba se ingresan los comandos mediante el archivo de texto *test 1 in.txt* en donde se realizan transferencias válidas, y cuyos valores de retorno son los *hashes* dados en el archivo *test 1 out.txt*, tal como se especifica en la sección 2


```
test_2_in.txt X
test_2_in.txt
1  init Lucio 1000 1
2  transfer Lucio Orne 100
3  transfer Lucio Lucas 100
4  transfer Lucio Fede 100
5  mine 3
6  transfer Orne Leandro 100
7  mine 5
8  balance Orne
9  balance Leandro
10 balance Lucas
11 balance Fede
12 balance Lucio
13 save algochainout.txt
14 exit

test_2_out.txt X
test_2_out.txt
1  4240d82dd0cb9d1712622354286adde206d07b7423df1bf63bb607b54a22001f
2  807be5fc4ad5810ad98d0e08855a3c4759a049f5a3953cc9fa807bfee86e595f
3  1229eb638183a897125a69e9eb0b7795d3bb8d7ac203a64878c1ed177464d2eb
4  a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
5  c8455298f567877a8fd0c6e2d827151ef4c74da1091027f6db57d7e07381a68f
6  072c7e60720f694469aeb9e8dd98e3bbfd180cce4addc3734b2f8b726431a57
7  c18bc79aa1667c7bbf6f8e1caf4a2f711e66ddfadb8becc3ca96660b190a3d
8  0.000000
9  100.000000
10 100.000000
11 100.000000
12 700.000000
13 Carga realizada con exito
14
```

Figura 2.3: Prueba 2

La segunda prueba consiste en reiteradas transacciones, minándolas y verificando el balance de cada usuario al finalizarlas.

```
test_3_in.txt X
test_3_in.txt
48 transfer Lucio Fede 100
49 transfer Fede Lucio 100
50 transfer Lucio Fede 100
51 transfer Fede Lucio 100
52 transfer Lucio Fede 100
53 transfer Fede Lucio 100
54 transfer Lucio Fede 100
55 transfer Fede Lucio 100
56 transfer Lucio Fede 100
57 transfer Fede Lucio 100
58 balance Fede
59 balance Lucio
60 save algochainout.txt
61 exit

test_3_out.txt X
test_3_out.txt
48 a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
49 547cfe7dbcd22ebd77212950fadddc1a03bb5aba8d5a7fa021ecf2a413e2425e
50 a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
51 547cfe7dbcd22ebd77212950fadddc1a03bb5aba8d5a7fa021ecf2a413e2425e
52 a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
53 547cfe7dbcd22ebd77212950fadddc1a03bb5aba8d5a7fa021ecf2a413e2425e
54 a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
55 547cfe7dbcd22ebd77212950fadddc1a03bb5aba8d5a7fa021ecf2a413e2425e
56 a9fe12314e2d8800ecdaf83a9ba56969b39bc46df58cdd77428057f2a56876a5
57 547cfe7dbcd22ebd77212950fadddc1a03bb5aba8d5a7fa021ecf2a413e2425e
58 0.000000
59 800.000000
60 Carga realizada con éxito
61
```

Figura 2.4: Prueba de funcionamiento 3

En cuanto a la tercer prueba se realizan unas transacciones iniciales arbitrarias y luego reiteradas transacciones entre dos usuarios entre los cuales fluye la misma cantidad de dinero. Eventualmente se verifica el balance de los usuarios luego del vaivén de dinero entre los mismos.

```
test_4_in.txt X
test_4_in.txt
1  init Lucio 1000 1
2  transfer Lucio Orne 100
3  transfer Lucio Fede -100
4  save algochainout.txt
5  exit

test_4_out.txt X
test_4_out.txt
1  4240d82dd0cb9d1712622354286adde206d07b7423df1bf63bb607b54a22001f
2  807be5fc4ad5810ad98d0e08855a3c4759a049f5a3953cc9fa807bfee86e595f
3  FAIL
4  Carga realizada con exito
5  
```

Figura 2.5: Prueba de funcionamiento 4

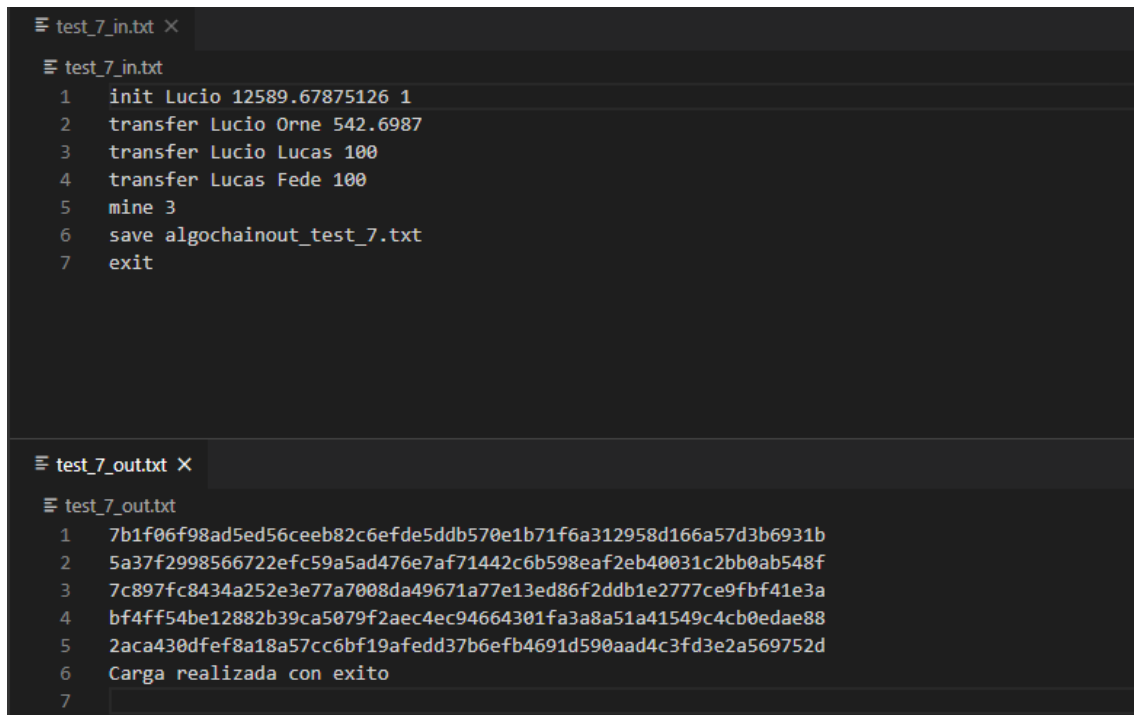
En esta prueba se intenta transferir dinero negativo, lo cual devuelve FAIL, y espera el próximo comando.

```
test_5_in.txt X
test_5_in.txt
1  init Lucio 1000 1
2  transfer Lucio Orne 100 Fede 100 Fran 100 Carla 100
3  transfer Lucio Orne 100 Fede 100 Fran 100 Carla 100
4  transfer Fede Carla 150
5  mine 3
6  save algochainout.txt
7  exit

test_5_out.txt X
test_5_out.txt
1  4240d82dd0cb9d1712622354286adde206d07b7423df1bf63bb607b54a22001f
2  0abc1a857d76e40ebbfefade88ab58654702098b172fd5bdd5a6ac30739f2fd3
3  07ac9392bfd60d5901a0f59c3d10a8d71908a40d36c4812e0a5c6947cf33de19
4  5c2a8d7e39bae84f0d5e9655feddc8927071a290c8fe9ce30fcb7527409c73b2
5  d85d28208b28676060199dbe4d80f883b97b5dd6b2c1899de9f705bd586803ec
6  Carga realizada con exito
7  |
```

Figura 2.6: Prueba de funcionamiento 5

Se realizan transferencias a mas de un usuario y en la ultima, el usuario debe obtener el dinero de dos inputs distintos.



```
test_7_in.txt
1  init Lucio 12589.67875126 1
2  transfer Lucio Orne 542.6987
3  transfer Lucio Lucas 100
4  transfer Lucas Fede 100
5  mine 3
6  save algochainout_test_7.txt
7  exit

test_7_out.txt
1  7b1f06f98ad5ed56ceeb82c6efde5ddb570e1b71f6a312958d166a57d3b6931b
2  5a37f2998566722efc59a5ad476e7af71442c6b598eaf2eb40031c2bb0ab548f
3  7c897fc8434a252e3e77a7008da49671a77e13ed86f2ddb1e2777ce9fbf41e3a
4  bf4ff54be12882b39ca5079f2aec4ec94664301fa3a8a51a41549c4cb0edae88
5  2aca430dfef8a18a57cc6bf19afedd37b6efb4691d590aad4c3fd3e2a569752d
6  Carga realizada con exito
7
```

Figura 2.7: Prueba de funcionamiento 6

Se realizan transferencias con números con decimales.

3 Conclusiones

En el presente trabajo se logra la realización del objetivo propuesto. Se demuestra en la sección 2.3.1 una serie de pruebas donde se muestra el correcto funcionamiento para distintas combinaciones de los comandos ingresados tanto de manera correcta o incorrecta.

Se aprendió el manejo de templates, variables globales, clases, listas y arreglos al poder implementarlo en su totalidad al buscar lograr el objetivo.

Por último, se implementó el algoritmo del árbol de Merkle de manera recursiva para el cálculo del hash de manera que se pueda comprobar que el hash sea correcto sin tener que acceder a información sensible.

4 Referencias

- Hash SHA256
<https://emn178.github.io/online-tools/sha256.html>
- “Bitcoin Wiki”, Wikipedia
<https://en.bitcoin.it/wiki/Mainpage>
- Fundamentos de la ingeniería de software
Ghezzi, C., Jazayeri, M., Mandrioli, D., Fundamentals of Software Engineering, Prentice-Hall International, Singapore, 1991

5 Códigos fuente

Listing 1: Array.h

```

1  #ifndef _ARRAY_INCLUDED_
2  #define _ARRAY_INCLUDED_
3  #include <iostream>
4  #include "arrayPrototype.h"
5
6  using namespace std;
7
8  #define INIT_SIZE_ARRAY 10
9
10 template <class T>
11 Array<T>::Array()
12 {
13     ptr = new T[INIT_SIZE_ARRAY];
14     rsize = INIT_SIZE_ARRAY;
15     vsize = INIT_SIZE_ARRAY;
16 }
17
18 template <class T>
19 Array<T>::Array(const size_t init_size)
20 {
21     ptr = new T[init_size];
22     rsize = init_size;
23     vsize = init_size;
24 }
25
26
27 template <class T>
28 Array<T>::Array(const Array<T> &init)
29 {
30     rsize = init.vsize;
31     vsize = init.vsize;
32
33     ptr = new T[init.vsize];
34     for(size_t i=0; i< init.vsize; i++)
35     {
36         ptr[i] = init.ptr[i]; //ASUMO QUE T tiene el operador =
37     }
38 }
39
40 template <class T>
41 Array<T>::~~Array() //Implementación del destructor de Array
42 {
43     if (ptr)
44         delete [ ] ptr; //Destructor de memoria dinámica.
45 }
46
47
48 template <class T>
49 size_t Array<T>::getSize()const { return vsize; } //Implementación del getter
50 // del tamaño del Array.
51
52
53 template <class T>
54 Array<T>& Array<T>::operator=(const Array<T> &right)
55 {
56     if(&right != this)
57     {
58         if(rsize != right.vsize)
59         {
60             T *aux;
61             aux = new T[right.vsize];
62             delete [ ] ptr;
63             rsize = right.vsize;
64             ptr = aux;

```

```

64     }
65
66     for (size_t i = 0; i < right.vsize; i++)
67     {
68         ptr[i] = right.ptr[i];
69     }
70     vsize = right.vsize;
71     return *this;
72 }
73 return *this;
74 }
75
76 template <class T>
77 bool Array<T>::operator==(const Array<T> &right) const
78 {
79     if(vsize != right.vsize)
80     {
81         return false;
82     }
83     else
84     {
85         for(size_t i = 0; i < right.vsize; i++)
86         {
87             if(ptr[i] == right.ptr[i])
88                 continue;
89             else
90                 return false;
91         }
92         return true;
93     }
94 }
95
96 template<class T>
97 T &Array<T>::operator[](size_t subscript)
98 {
99     if(subscript >= vsize)
100         std::abort();
101     return ptr[subscript];
102 }
103
104 template <class T>
105 std::ostream &operator <<(std::ostream &os, Array<T> &arr)
106 {
107     int i, size;
108     size = arr.getSize();
109     for(i=0; i < size; i++)
110     {
111         os << arr[i] << endl; //asumo que T tiene sobrecargado el <<
112     }
113     return os;
114 }
115
116
117 template<class T>
118 void Array<T>::ArrayRedim(size_t new_size)
119 {
120     if (new_size > rsize) {
121         T *aux = new T[new_size];
122         for (size_t i = 0; i < vsize; ++i)
123             aux[i] = ptr[i];
124         delete[] ptr, ptr = aux;
125         rsize = new_size;
126     }
127
128     vsize = new_size;
129 }
130
131 template <class T>
132 bool Array<T>::empty()

```

```
134 {
135     if(vsize)
136         return true;
137     else
138         return false;
139 }
140
141 template <class T>
142 Array<T> Array<T>::getSubArray(const size_t n1,const size_t n2)
143 {
144     Array<T> aux;
145     aux.vsize=0;
146     if(n1>this->getSize())
147         return aux;
148     else
149     {
150         if(n2<=this->getSize())
151         {
152             aux.ArrayRedim(n2-n1+1);
153             for(size_t i=n1-1; i<n2; i++)
154                 aux[i-n1+1]=this->ptr[i];
155         }
156         if(n2>this->getSize())
157         {
158             aux.ArrayRedim(this->getSize()-n1+1);
159             for(size_t i=n1-1; i<this->getSize(); i++)
160                 aux[i-n1+1]=this->ptr[i];
161         }
162     }
163     return aux;
164 }
165 #endif //ARRAY_INCLUDED
```

Listing 2: ArrayPrototype.h

```

1  #ifndef _ARRAYPROTOTYPE_H_
2  #define _ARRAYPROTOTYPE_H_
3
4  template<class T>
5  class Array
6  {
7  public:
8      Array(); //Constructor base
9      Array( const Array<T> & ); //Creador-Copiador
10     Array(size_t); //Creador mediante tamaño del array
11     ~Array(); //Destructor
12     void ArrayRedim(size_t); //Redimensionador de arrays.
13     size_t getSize() const; //Método: determina el tamaño del array
14     Array<T> getSubArray(const size_t n1, const size_t n2); //Obtiene un
15     subarreglo que consta de los mismos datos que el arreglo original entre
16     // las posiciones n1 y n2. Ejemplo: si n1=1 y n2
17     =4, devuelve un subarray de 4 elementos:
18     // los del arreglo original limitados por n1-1 (
19     cero) y n2-1 (tres).
20     // Si n2 es mayor al tamaño del subarreglo
21     original, se genera un subarreglo
22     // delimitado por n1 y el final del arreglo
23     original.
24     Array<T> &operator=( const Array<T> & ); //Operador asignación para una
25     array: A=B, donde A y B son arrays. Recibe como parámetro un array por
26     referencia constante, para no modificar lo que tiene dentro
27     bool operator==( const Array<T> & ) const; //Operador lógico para comprobar
28     si son iguales 2 arrays. Recibe como parámetro un array por referencia
29     constante, para no modificar lo que tiene dentro
30     T & operator[](size_t); //Operador indexación: Retorna un elemento del
31     vector (se puede cambiar, pues se retorna por referencia)
32     template <class TT>
33     friend std::ostream &operator<<(std::ostream&, Array <TT> &); //Operador de
34     impresion de salida
35     bool empty(); // Verifica si un arreglo esta vacio.
36
37 private:
38     size_t rsize; //Atributo que indica el tamaño del array
39     size_t vsize;
40     T *ptr; //Atributo que indica la dirección donde inicia el puntero
41 };
42
43 #endif //_ARRAYPROTOTYPE_H_

```



```

64 string inpt::getAddr(){return addr;}

66 void inpt::setInput(string aux_tx_id, size_t aux_idx, string aux_addr)
{
68     outpoint.tx_id=aux_tx_id;
69     outpoint.idx=aux_idx;
70     addr=aux_addr;
71 }

72 void inpt::show(ostream& oss)
73 {
74     if(outpoint.tx_id == "" || addr == "")
75     {
76         return ;
77     }
78     oss << outpoint.tx_id << " " << outpoint.idx << " " << addr;
79 }

80

82 bool inpt::operator==(const inpt & right) const
{
84     if(outpoint.idx == right.outpoint.idx && outpoint.tx_id == right.outpoint.
        tx_id && addr == right.addr)
        return true;
86     else
        return false;
87 }

90 bool inpt::operator!=(const inpt & right){return !(*this == right);}

92 string inpt::toString()
{
94     ostringstream ss;
95     ss << *this;
96     return ss.str();
97 }

98 //-----CLASE OUTPUT
-----

100 outpt::outpt() //Creador base
{
101 }

102

104 outpt::~outpt() //Destructor base
{
105 }

106

108 outpt & outpt::operator=(const outpt & right)
{
110     if(&right != this)
    {
112         value = right.value;
113         addr = right.addr;
114         return *this;
115     }
116     return *this;
117 }

118 outpt::outpt(string & str) //Creador mediante una string
119 {
120     istringstream ss(str);
121     string str_value, str_addr;

122     getline(ss, str_value, ' '); // Se recorren los campos. Si el formato es
        erroneo, se detecta como una cadena vacia.
123     getline(ss, str_addr, '\n');

124     if((isNumber<double>(str_value)==1) && (isHash(str_addr)==true))
125     {
126         this->value=str_value;

```

```

130     this->addr=str_addr;
131 }
132 else
133 {
134     this->addr=ERROR;
135 }
136 }
137 outpt::outpt(string& str_addr, string & str_value)
138 {
139     if((isNumber<double>(str_value)==1) && (isHash(str_addr)==true))
140     {
141         this->value=str_value;
142         this->addr=str_addr;
143     }
144     else
145     {
146         this->addr=ERROR;
147     }
148 }
149
150 string outpt::getAddr(){return addr;}
151
152 string outpt::getValue(){return value;}
153
154 void outpt::show(ostream& oss)
155 {
156     if(addr == "")
157     {
158         return ;
159     }
160     oss << value << " " << addr;
161 }
162
163
164 bool outpt::operator==(const outpt & right) const
165 {
166     if(value == right.value && addr == right.addr)
167         return true;
168     else
169         return false;
170 }
171
172 string outpt::toString()
173 {
174     ostringstream ss;
175     ss << *this;
176     return ss.str();
177 }
178 //-----CLASE TXN
179 -----
180
181
182 txn::txn()
183 {
184     n_tx_in=0;
185     n_tx_out=0;
186     tx_in.ArrayRedim(0);
187     tx_out.ArrayRedim(0);
188 }
189
190 txn::txn(Array<string>& txn_str_arr)
191 {
192     size_t i;
193     this->setNTxIn(stoi(txn_str_arr[0]));
194     for(i=1;i<(this->getNTxIn()+1;i++)
195     {
196         inpt in(txn_str_arr[i]);
197         tx_in[i-1] = in;

```

```

198     }
199     this->setNTxOut(stoi(txn_str_arr[i]));
200     for( size_t j=0;j<(this->getNTxOut());j++,i++)
201     {
202         outpt out(txn_str_arr[i]);
203         tx_out[j] = out;
204     }
205 }
206 txn::txn(string txn_str)
207 {
208     istringstream ss(txn_str);
209     string aux;
210     getline(ss, aux, '\n');
211     size_t i;
212     n_tx_in = stoi(aux);
213     tx_in.ArrayRedim(n_tx_in);
214     for(i=0;i<n_tx_in;i++)
215     {
216         getline(ss, aux, '\n');
217         inpt in(aux);
218         tx_in[i] = in;
219     }
220     getline(ss, aux, '\n');
221     n_tx_out = stoi(aux);
222     tx_out.ArrayRedim(n_tx_out);
223     for(i=0 ;i<n_tx_out;i++)
224     {
225         getline(ss, aux, '\n');
226         outpt out(aux);
227         tx_out[i] = out;
228     }
229 }
230
231 txn::~txn()
232 {
233 }
234
235 txn & txn::operator=(const txn &right)
236 {
237     if(&right != this)
238     {
239         n_tx_in = right.n_tx_in;
240         n_tx_out = right.n_tx_out;
241         tx_in = right.tx_in;
242         tx_out = right.tx_out;
243         return *this;
244     }
245     return *this;
246 }
247 void txn::setNTxIn(const size_t n)
248 {
249     n_tx_in=n;
250     if(tx_in.getSize() == 0)
251     {
252         tx_in.ArrayRedim(n);
253     }
254 }
255
256 void txn::setNTxOut(const size_t n)
257 {
258     n_tx_out=n;
259     if(tx_out.getSize() == 0)
260     {
261         tx_out.ArrayRedim(n);
262     }
263 }
264
265

```

```
268 string txn::setTxIn(const size_t n, istream *iss) //Se modifica el retorno del
    setter por defecto (void) por
    {
        // necesidad. Verifica si el setteo pudo realizarse
        correctamente.
270     string aux_s;
        tx_in.ArrayRedim(n);
272     for (size_t i = 0; i < n; i++)
    {
274         getline(*iss, aux_s, '\n');
        if(isHash(aux_s)==true)
276     {
            return aux_s;
278     }
        inpt in(aux_s);
280     if(isError(in.getAddr())==false)
        return "ERROR: adres invalida";
282     tx_in[i] = in;
    }
284     return "OK";
}

286 bool txn::setTxIn(const size_t n, Array<string>& tx_in_str_arr)
288 {
    //Se modifica el retorno del setter por defecto (void) por
290     // necesidad. Verifica si el setteo pudo realizarse correctamente.
    string aux_s;
292     tx_in.ArrayRedim(n);
    for (size_t i = 0; i < n; i++)
294     {
        inpt in(tx_in_str_arr[i]);
296         if(isError(in.getAddr())==false)
            return false;
298         tx_in[i] = in;
    }
300     return true;
}

302 bool txn::setTxIn(Array<inpt>& arr)
304 {
    size_t n = arr.getSize();
306     tx_in.ArrayRedim(n);
    for (size_t i = 0; i < n; i++)
308     {
        if(isError(arr[i].getAddr())==false)
310         return false;
        tx_in[i]=arr[i];
312     }
    return true;
314 }

316 string txn::setTxOut(const size_t n, istream *iss) //Se modifica el retorno del
    setter por defecto (void) por
318     // necesidad. Verifica si el setteo pudo realizarse
    correctamente.
    {
320     string aux_s;
        tx_out.ArrayRedim(n);
322     for (size_t i = 0; i < n; i++)
    {
324         getline(*iss, aux_s, '\n');
        if(isHash(aux_s)==true)
326     {
            return aux_s;
328     }
        outpt out(aux_s);
330     if(isError(out.getAddr())==false)
        return "ERROR: adres invalida";
332     tx_out[i] = out;
    }
}
```

```

334     return "OK";
335 }
336
337 string txn::setTxOut(Array<string> dst, Array<string> dst_value)
338 {
339     size_t n=dst.getSize();
340     tx_out.ArrayRedim(n);
341     for(size_t i=0; i< n;i++)
342     {
343         outpt aux_output(dst[i],dst_value[i]);
344         if(isError(aux_output.getAddr())==false)
345             return "ERROR: address invalida";
346         tx_out[i]=aux_output;
347     }
348     return "OK";
349 }
350
351 bool txn::setTxOut(const size_t n, Array<string>& tx_in_str_arr) //Se modifica
352 el retorno del setter por defecto (void) por
353 // necesidad. Verifica si el seteo pudo realizarse
354 correctamente.
355 {
356     tx_out.ArrayRedim(n);
357     for (size_t i = 0; i < n; i++)
358     {
359         outpt out(tx_in_str_arr[i]);
360         if(isError(out.getAddr())==false)
361             return false;
362         tx_out[i] = out;
363     }
364     return true;
365 }
366
367 size_t txn::getNTxIn(){return n_tx_in;}
368
369 size_t txn::getNTxOut(){return n_tx_out;}
370
371 Array<inpt>& txn::getInputs(){return tx_in;}
372 Array<outpt>& txn::getOutputs(){return tx_out;}
373
374 void txn::show(ostream& oss)
375 {
376     size_t i;
377     if(n_tx_in == 0)
378         return ;
379     oss << n_tx_in << endl;
380     for (i = 0; i < n_tx_in; i++)
381     {
382         oss << tx_in[i] << endl;
383     }
384
385     oss << n_tx_out << endl;
386     for (i = 0; i < n_tx_out-1; i++)
387     {
388         oss << tx_out[i] << endl;
389     }
390     oss << tx_out[i];
391 }
392
393 bool txn::operator==(const txn & right) const
394 {
395     if(n_tx_in == right.n_tx_in && tx_in == right.tx_in && n_tx_out == right.
396         n_tx_out && tx_out == right.tx_out )
397         return true;
398     else
399         return false;
400 }
401
402 string txn::toString()

```

```

{
402     ostreamstream ss;
403     ss << *this;
404     return ss.str();
405 }
406 //-----CLASE BODY
407
408 bdy::bdy()
409 {
410 }
411
412 bdy::~~bdy(){}
413
414 bdy & bdy::operator=(const bdy & right)
415 {
416     if(&right != this)
417     {
418         txn_count = right.txn_count;
419         txns = right.txns;
420         return *this;
421     }
422     return *this;
423 }
424
425 void bdy::setTxnCount(const size_t n)
426 {
427
428     txn_count = n;
429     if(this->txns.getSize() == 0)
430     {
431         txns.ArrayRedim(n);
432     }
433 }
434
435 string bdy::setTxns(istream *iss)
436 {
437     string str, error_string;
438     size_t aux, i = 0;
439     bool err=false;
440     while(getline(*iss, str, '\n'))
441     {
442         if(isHash(str)==true || str == "")
443         {
444             txn_count = i;
445             return str;
446         }
447         if(i >=txns.getSize())
448         {
449             txns.ArrayRedim(txns.getSize()*2); // Dependiendo de cuantos datos haya
450             // que analizar se puede modificar // la estrategia de crecimiento del arreglo.
451         }
452
453         // Se verifica n_tx_in
454         if(isNumber<size_t>(str)==0 || (str[0] == '\0'))
455         {
456             err=true;
457             break;
458         }
459
460         aux = stoi(str);
461         txns[i].setNTxIn(aux);
462         // Se verifican las entradas
463
464         if(txns[i].setTxIn(aux, iss)!="OK")
465         {
466             err=true;
467             break;
468         }
469     }

```

```

468      // Se verifica n_tx_out
470      getline(*iss, str, '\n');
472      if(isNumber<size_t>(str)==0 || (str[0]) == '\0')
474      {
476          err=true;
478          break;
480      }
482
484      aux = stoi(str);
486      txns[i].setNTxOut(aux);
488
490      // Se verifican las salidas
492      str=txns[i].setTxOut(aux, iss);
494      i++;
496
498      if(isHash(str)==true)
500      {
502          txn_count = i;
504          return str;
506      }
508      else if(str=="OK")
510      {
512          if(i < txn_count)
514              continue;
516          txn_count = i;
518          return str;
520      }
522      else
524      {
526          err=true;
528          break;
530      }
532      }
534      txn_count = i;
536      if(err==true)
538      {
540          error_string.append("Error en la transaccion ");
542          error_string.append(to_string(i+1));
544          error_string.append("\n");
546          error_string.append("Carga de datos interrumpida");
548          error_string.append("\n");
550          error_string.append("Vuelva a cargar los datos del bloque");
552          return error_string;
554      }
556
558      if(str == "")
560      {
562          return str;
564      }
566      return "OK";
568      }
570
572      size_t bdy::getTxnCount(){return txn_count;}
574      Array<txn> &bdy::getTxns(){return txns;}
576
578      bdy bdy::getBody(){return *this;}
580
582      void bdy::txnsArrRedim(const size_t n ){txns.ArrayRedim(n);}
584
586      void bdy::show(ostream& oss)
588      {
590          size_t i;
592          if(txn_count == 1)
594              oss << txn_count << endl;
596          else

```



```

538     oss << txn_count << endl;
540 for (i = 0; i < txn_count - 1; i++)
542 {
544     oss << txns[i];
546     oss << endl;
548 }
550 if(txn_count !=0)
552 {
554     oss << txns[i];
556 }
558 }
560 string bdy::toString()
562 {
564     ostringstream ss;
566     ss << *this;
568     return ss.str();
570 }
572 //-----CLASE HEADER
574 -----
576
578
580
582
584
586
588
590
592
594
596
598
600
602

```

```

560 hdr::hdr()
562 {
564     prev_block = "\0";
566     txns_hash = "\0";
568     bits = 0; //Se podría hacer que los bits y el nonce fueran ints para
570             detectar errores haciendo que estos valgan -1 (por ej)
572     nonce = 0;
574 }
576
578
580
582
584
586
588
590
592
594
596
598
600
602

```

```

560 hdr::~hdr(){}
562
564
566
568
570
572
574
576
578
580
582
584
586
588
590
592
594
596
598
600
602

```

```

570 hdr & hdr::operator=(const hdr & right)
572 {
574     if(&right != this)
576     {
578         prev_block = right.prev_block;
580         txns_hash = right.txns_hash;
582         bits = right.bits;
584         nonce = right.nonce;
586         return *this;
588     }
590     return *this;
592 }
594
596
598
600
602

```

```

570 bool hdr::setPrevBlock(const string & str)//Se modifica el retorno del setter
572 por defecto (void) por
574 // necesidad. Verifica si el setteo pudo realizarse
576 correctamente.
578 {
580     if(isHash(str) == false)
582         return false;
584     else
586     {
588         prev_block = str;
590         return true;
592     }
594 }
596
598
600
602

```

```

596 void hdr::setTxnsHash(const string & str)
598 {
600     if(isHash(str)==true)
602     {
604         txns_hash = str;
606     }
608     else
610

```

```

    txns_hash = sha256(sha256(str));
604 }

606 void hdr::setTxnsHash(Array<txn> & txns)
{
608     Array<string> hashes(txns.getSize());
    for (size_t i = 0; i < txns.getSize(); i++)
610     {
        hashes[i] = sha256(sha256(txns[i].toString()));
612     }
    txns_hash = merkle_hash(hashes, hashes.getSize());
614 }

616 void hdr::setBits(const size_t n){bits = n;}
618 void hdr::setNonce(const size_t n){nonce = n;}
620

622 string hdr::getPrevBlock(){return prev_block;}

624 string hdr::getTxnHash(){return txns_hash;}
626

628 size_t hdr::getBits(){return bits;}
630
632 size_t hdr::getNonce(){return nonce;}

634 void hdr::setNonce(const string prev_block, const string txns, const size_t
    bits) // Setea el header con el nonce que verifica que el hash del header
    cumpla con los primeros d bits en cero
{
636     size_t out = 0; // inicializo d_aux que contara el nivel de dificultad y out
    que es un flag para el for
    size_t nonce_aux = 0; // inicializo el nonce, para mi hay que hacerlo double
    porque se puede hacer muy grande pero hay que cambiar struct
638
    string header_str; // defino un header_aux auxiliar para hacer la string antes
    de hashearla
640 string hash_header; // para guardar el hash del header_aux

642 int j, i, aux;

644 int cant_char = bits/4; // me da la cantidad de char en 0 que necesito
    int cant_bit = bits%4; // me da la cantidad de bits del ultimo char en 0
646

648 for (nonce_aux = 0; out == 0 ; nonce_aux++) // aumento el nonce hasta que el flag
    out sea 1, igualmente tambien hay un break, es por las dudas que el break
    no funcione como espero
    {
650         header_str.clear();
        nonce = nonce_aux;
652         header_str = toString();
        header_str.append("\n");
654         hash_header = sha256(sha256(header_str)); // calculo el hash del header_aux
        i=0;
        aux=0;
        while (i<cant_char)
656         {
            if(hash_header[i] != '0')
660             {
                aux=1;
                break;
662             }
            i++;
664         }
    }
}

```

```

666     if(aux==1)
667     {
668         continue;
669     }
670     else
671     {
672         hash_header = Hex2Bin(hash_header.substr(i,i+1));
673
674         j=0;
675         aux=0;
676         while (j<cant_bit)
677         {
678             if(hash_header[j] != '0')
679             {
680                 aux=1;
681                 break;
682             }
683             j++;
684         }
685         if(aux == 0)
686         {
687             nonce = nonce_aux; //guardo el nonce en el header_aux
688             out = 1; // Para verificar que termine el for
689             break; //como cumple la cantidad de bits necesarias y ya esta guardado y
690             hasheado salgo
691         }
692     }
693     return ;
694 }
695
696 void hdr::show(ostream& oss)
697 {
698     oss << prev_block << endl;
699     oss << txns_hash << endl;
700     oss << bits << endl;
701     oss << nonce << endl;
702 }
703
704 string hdr::toString()
705 {
706     ostringstream ss;
707     ss << *this;
708     return ss.str();
709 }
710 //-----CLASE BLOCK
711 -----
712
713 hdr block::getHeader()
714 {
715     return header;
716 }
717
718 bdy block::getBody()
719 {
720     return body;
721 }
722
723 void block::setHeader(hdr h){header = h;}
724
725 void block::setHeader(const string& prev_block_str, const size_t diffic)
726 {
727     string aux;
728     header.setPrevBlock(prev_block_str);
729     header.setTxnsHash(body.getTxns());
730     header.setBits(diffic);
731     header.setNonce(header.getPrevBlock(), header.getTxnHash(), header.getBits());
732 }
733

```

```

block & block::operator=(const block & right)
734 {
736     if(&right != this)
738     {
740         header = right.header;
742         body = right.body;
744         return *this;
746     }
748     return *this;
750 }

string block::setBody(istream *iss)
752 {
754     string str;
756     getline(*iss, str, '\n');
758
760     size_t txn_count = stoi(str);
762     body.setTxnCount(txn_count);
764     str=body.setTxns(iss);
766     if (isHash(str)==true)
768     {
770         return str;
772     }
774     else if (str=="")
776     {
778         return str;
780     }
782     else if(str!="OK")
784     {
786         cerr<< "ERROR: set txns fallo \n"<< str << endl;
788         exit(1);
790     }
792     return "OK";
794 }

block::block()
796 {
798     header.setPrevBlock(NULL_HASH);
800     header.setTxnsHash(NULL_HASH);
802     header.setBits(0);
804     header.setNonce(0);
806     body.setTxnCount(0);
808 }

block::block(const string str,const size_t diffc, istream *iss)
810 {
812     setBody(iss);
814     setHeader(str, diffc);
816 }

block::block(const string block_str)
818 {
820     istringstream ss(block_str);
822     string aux;
824     int aux_int;
826     getline(ss, aux, '\n');
828     this->header.setPrevBlock(aux);
830     getline(ss, aux, '\n');
832     this->header.setTxnsHash(aux);
834     getline(ss, aux, '\n');
836     aux_int=stoi(aux);
838     this->header.setBits(aux_int);
840     getline(ss, aux, '\n');
842     aux_int=stoi(aux);
844     this->header.setNonce(aux_int);
846     getline(ss, aux, '\n');
848     aux_int=stoi(aux);
850     body.setTxnCount(aux_int);
852     body.setTxns(&ss);
854 }

```

```
    }  
804  
    block::~~block()  
806    {  
    }  
808  
810    void block::addTxn(txn aux_txn)  
    {  
812        body.setTxnCount(body.getTxnCount()+ 1);  
        body.txnsArrRedim(body.getTxnCount());  
814        body.getTxns()[body.getTxnCount()-1]=aux_txn;  
    }  
816  
818    void block::show(ostream& oss)  
    {  
820        oss << header;  
        oss << body;  
822    }  
  
824    string block::toString()  
    {  
826        ostringstream ss;  
        ss << *this;  
828        return ss.str();  
    }  
830 #endif // _BLOCK_H_
```

Listing 4: BlockPrototype.h

```

1  #ifndef _BLOCKPROTOTYPE_H_
2  #define _BLOCKPROTOTYPE_H_

3
4  #include <iostream>
5  #include <string.h>
6  #include "Array.h"

7
8  using namespace std;

9
10 //-----ESTRUCTURA OUTPOINT
11 -----
12 struct outpnt
13 {
14     string tx_id; //Es un hash de la transaccion de donde este input toma fondos
15     size_t idx; //Valor entero no negativo que representa un indice sobre la
16                 //secuencia de outputs de la transaccion con hash tx id
17 };
18 //-----CLASE INPUT
19 -----
20 class inpt
21 {
22     outpnt outpoint;
23     string addr; //La direccion de origen de los fondos (que debe coincidir con la
24                 //direccion del output referenciado)
25 public:
26     inpt(); //Creador base
27     inpt(string&); //Creador mediante una string
28     ~inpt( ); //Destructor
29     inpt & operator=(const inpt &);
30     //Si hay getters deberian haber setters. Si no se usan, eliminarlos.
31     string getAddr();
32     outpnt getOutPoint();
33     void setInput(string aux_tx_id, size_t aux_idx, string aux_addr);
34     void show(ostream&);
35     friend ostream& operator<<(ostream& oss, inpt& in)
36     {
37         in.show(oss);
38         return oss;
39     }
40     bool operator==(const inpt &) const;
41     bool operator!=(const inpt &);
42     string toString();
43 };
44 //-----CLASE OUTPUT
45 -----
46 class outpt
47 {
48     string value; //La cantidad de Algocoins a transferir en este output
49     string addr; //La direccion de origen de los fondos (que debe coincidir con la
50                 //direccion del output referenciado)
51 public:
52     outpt(); //Creador base
53     outpt(string&); //Creador mediante una string
54     outpt(string&, string &);
55     ~outpt( ); //Destructor
56     outpt & operator=(const outpt &);
57     string getValue();
58     string getAddr();
59     void show(ostream&);
60     friend ostream& operator<<(ostream& oss, outpt& out)
61     {
62         out.show(oss);

```

```

60     return oss;
61 }
62 bool operator==( const outpt &) const;
63 string toString();
64 };
65
66 //-----CLASE TXN
67 -----
68
69
70 class txn
71 {
72 private:
73     size_t n_tx_in; //Indica la cantidad total de inputs en la transaccion
74     size_t n_tx_out; //Indica la cantidad total de outputs en la transaccion
75     Array<inpt> tx_in; //Datos de entrada para las transacciones
76     Array<outpt> tx_out; //Datos de salida para las transacciones
77
78 public:
79     txn(); //Creador base
80     txn(Array<string>&); //Creador en base a un array de cadenas. El array debe
        // contener todos los campos necesarios
        // para crear la transaccion.
81     txn(string txn_str); //Creador en base a una cadenas que contiene toda la
        // informacion para crear la transaccion.
82     ~txn(); //Destructor
83     txn &operator=( const txn & );
84
85     void setNTxIn(const size_t) ;
86     void setNTxOut(const size_t);
87     string setTxIn(const size_t n, istream *iss); // Seteador que valida los datos
        // y devuelve un booleano para el error
88     bool setTxIn(Array<inpt>&);
89     bool setTxIn(const size_t n, Array<string>& tx_in_str_arr);
90     string setTxOut(Array<string>, Array<string>);
91     bool setTxOut(const size_t, Array<string>&);
92     string setTxOut(const size_t n, istream *iss);
93     string setTxOutFile(const size_t n, istream *iss);
94
95     size_t getNTxIn();
96     size_t getNTxOut();
97     Array<inpt>& getInputs();
98     Array<outpt>& getOutputs();
99
100     string validateTxn();
101     void show(ostream&);
102     friend ostream& operator<<(ostream& oss, txn& tx)
103     {
104         tx.show(oss);
105         return oss;
106     }
107     string toString();
108     bool operator==( const txn &) const;
109 };
110
111 //-----CLASE BODY
112 -----
113
114 class bdy
115 {
116     friend class block;
117     size_t txn_count;
118     Array<txn> txns;
119 public:
120     bdy();
121     ~bdy();
122     bdy & operator=(const bdy &);

```

```

124 bdy getBody();
125 size_t getTxnCount();
126 Array<txn> &getTxns();
127 // void setTxns(Array <txn> txns);
128 string setTxns(istream *iss);
129 void setTxnCount(const size_t n);
130 void txnsArrRedim(const size_t );
131 void show(ostream&);
132 friend ostream& operator<<(ostream& oss, bdy& body)
133 {
134     body.show(oss);
135     return oss;
136 }
137 string toString();
138 };
139
140 //-----CLASE HEADER
141
142 class hdr
143 {
144     friend class block;
145     string prev_block; //El hash del bloque completo que antecede al bloque actual
146     // en la Algochain.
147     string txns_hash; //El hash de todas las transacciones incluidas en el bloque.
148     size_t bits; // Valor entero positivo que indica la dificultad con la que
149     // fue minada este bloque.
150     size_t nonce; // Un valor entero no negativo que puede contener valores
151     // arbitrarios. El objetivo de este
152     // campo es tener un espacio de prueba modificable para poder generar
153     // hashes sucesivos hasta
154     // satisfacer la dificultad del minado.
155 public:
156     hdr();
157     ~hdr();
158
159     hdr & operator=(const hdr &);
160     bool setPrevBlock(const string&);
161     void setTxnsHash(const string&);
162     void setTxnsHash(Array<txn> & txns);
163     void setBits(const size_t n);
164     void setNonce(const string prev_block, const string txns, const size_t bits);
165     void setNonce(const size_t nonce);
166     string getPrevBlock();
167     string getTxnHash();
168     size_t getBits();
169     size_t getNonce();
170     void show(ostream&);
171     friend ostream& operator<<(ostream& oss, hdr& header)
172     {
173         header.show(oss);
174         return oss;
175     }
176     string toString();
177 };
178
179 //-----CLASE BLOCK
180
181 class block{
182 private:
183     hdr header;
184     bdy body;
185
186 public:
187     block(); //Constructor base
188     block(const string, const size_t, istream*); //Constructor en base al hash del

```



```
    bloque previo, al nivel de
                                //dificultad y un flujo de entrada por el que se reciben
    las
                                // transacciones.
186 block(const string ); //Constructor en base a una string.
188 ~block( ); //Destructor
    block & operator=(const block &);
190 void setHeader(const string&,const size_t);
    string setBody(istream *iss);
192 void setHeader(hdr header);
    void setBody(bdy body);
194 void setBlockFromFile(istream *iss);
    hdr getHeader();
196 bdy getBody();
    void addTxn(txn aux_txn);
198 void show(ostream&);
    friend ostream& operator<<(ostream& oss, block& block)
200 {
        block.show(oss);
202     return oss;
    }
204 string toString();
};
206 #endif // _BLOCKPROTOTYPE_H_
```

Listing 5: dictionary.h

```

1 #ifndef _DICTIONARY_H_
2 #define _DICTIONARY_H_
3
4 // dictionary.h tiene elementos que traducen los comandos para la interfaz con
   la algochain
5 // ingresados por linea de comandos.
6 #include <iostream>
7 #include <string.h>
8 #include <cstring>
9
10 #include "Array.h"
11 #include "tools.h"
12 #include "Lista.h"
13 #include "block.h"
14 #include "sha256.h"
15 #include "Lista.h"
16 #include "main.h"
17 #include "user.h"
18
19 //-----MACROS
20
21 //Para definir las referencias de comandos
22 #define STR_INIT "init"
23 #define STR_TRANSFER "transfer"
24 #define STR_MINE "mine"
25 #define STR_BALANCE "balance"
26 #define STR_BLOCK "block"
27 #define STR_TXN "txn"
28 #define STR_LOAD "load"
29 #define STR_SAVE "save"
30
31 //Para definir hashes en un estado inicial
32 #define NULL_HASH "
   0000000000000000000000000000000000000000000000000000000000000000"
33
34 //Para contar la cantidad de punteros a funciones se tiene. Se utiliza para
   encontrar el que se necesita en
35 //cada situacion
36 #define MAXCMD 8
37
38 //Para ocurrencias de error
39 #define MSG_FAIL "FAIL"
40
41 //Para aumentar el tamaño de un arreglo dinamico
42 #define CHOP_SIZE 5
43
44 //-----VARIABLES GLOBALES
45
46 using namespace std;
47
48 //-----PUNTEROS A FUNCION
49
50 //Los punteros a funcion ejecutan el comando ingresado y devuelven lo
   especificado por el comando (como un
51 // hash que represente cierto objeto o un mensaje de error)
52 typedef string (* p_func)(Array <string>);
53
54 string cmdInit(Array <string> args);
55 string cmdTransfer( Array <string> args);
56 string cmdMine(Array <string> args);
57 string cmdBalance(Array <string> args);
58 string cmdBlock(Array <string> args);
59 string cmdTxn(Array <string> args);
60 string cmdLoad(Array <string> args);
61 string cmdSave(Array <string> args);

```

```

62 struct cmd_option_t
63 {
64     string name;
65     p_func comand;
66     int num_param;
67 };
68
69 static cmd_option_t dictionary_cmd[] = {
70     {STR_INIT, cmdInit, 3},
71     {STR_TRANSFER, cmdTransfer, 0},
72     {STR_MINE, cmdMine, 1},
73     {STR_BALANCE, cmdBalance, 1},
74     {STR_BLOCK, cmdBlock, 1},
75     {STR_TXN, cmdTxn, 1},
76     {STR_LOAD, cmdLoad, 1},
77     {STR_SAVE, cmdSave, 1},
78 };
79
80 p_func dictCmds( string cmd, int &num_param)
81 {
82     string aux;
83     int i = 0;
84     while(cmd != dictionary_cmd[i].name && i < MAXCMD) i++;
85     if(i == MAXCMD)
86     {
87         cerr << "El comando no es valido" << endl;
88         exit(1);
89     }
90     num_param = dictionary_cmd[i].num_param;
91     return dictionary_cmd[i].comand;
92 }
93
94 string cmdInit(Array <string> args)
95 {
96
97     string user_;
98     string STR_TXNing;
99     size_t bits;
100     if(algochain.empty()==false)
101     {
102         list <block> list_empty;
103         algochain = list_empty;
104     }
105     if(users.empty()==false)
106     {
107         list <user> list_empty;
108         users = list_empty;
109     }
110     user_ = sha256(args[0]);
111
112     if(isNumber<double>(args[1]) == false || args[1][0] == '-')
113     {
114         cerr << "El user no es valido" << endl;
115         exit(1);
116     }
117
118
119
120     if(isNumber<int>(args[2]) == false || args[2][0] == '-')
121     {
122         cerr << "El bits no es valido" << endl;
123         exit(1);
124     }
125
126     bits = stoi(args[2]);
127     STR_TXNing.append("1"); // txn count
128     STR_TXNing.append("\n");
129     STR_TXNing.append("1"); // txn in
130     STR_TXNing.append("\n");
131     STR_TXNing.append(NULL_HASH);

```

```

132 STR_TXNing.append(" ");
133 STR_TXNing.append("0");
134 STR_TXNing.append(" ");
135 STR_TXNing.append(NULL_HASH);
136 STR_TXNing.append("\n");
137 STR_TXNing.append("1");
138 STR_TXNing.append("\n");
139 STR_TXNing.append(args[1]);
140 STR_TXNing.append(" ");
141 STR_TXNing.append(user_);
142
143 istream iss(STR_TXNing);
144 block genesis_block(NULL_HASH, bits, &iss);
145 if (refreshUsersFromBlock(genesis_block) == false)
146 {
147     cerr << "ERROR: No se puede cargar user" << endl;
148     exit(1);
149 }
150 algochain.append(genesis_block);
151 return sha256(sha256(genesis_block.toString()));
152 }
153
154 string cmdTransfer( Array <string> args)
155 {
156     //El comando se invoca asi:
157     // transfer src dst1 value1 dst2 value 2 ... dstN valueN
158
159     //Se debe buscar la ultima aparicion de ese usuario primero en la MEMPOOL y si
160     //no se encuentra nada, en la
161     //ALGOCHAIN; conseguir su value y verificar que dicho valor (su dinero
162     //disponible) no sea menor a la suma de
163     //las cantidades a transferir.
164
165     double src_balance,aux;
166     string src=sha256(args[0]); //El primer elemento se condice con el usuario de
167     //origen.
168
169     if(users.find("checkUser",src)=="Findnt")
170     {
171         cerr<< "ERROR usuario: " << args[0] << " inexistente" << endl;
172         exit(1);
173     }
174     src_balance=stod(users.find("balance",src)); //Se busca el dinero disponible
175     //de el usuario src, que aporta el dinero en la transaccion.
176     //Precondicion: la lista global con los balances debe
177     //estar actualizada en todo momento.
178     aux=src_balance;
179     size_t dim_array_aux=(args.getSize()-1)/2;
180     Array<string> dst(dim_array_aux); //Arreglo de usuarios destino.
181
182     //Al saber la cantidad de argumentos que se reciben se puede calcular el tamaño
183     //o de los arreglos auxiliares, pues
184     // args.getSize()-1 es la cantidad de argumentos relacionados a los usuarios
185     //de destino. Como vienen de a pares
186     //(una vez validados) el resultado de (args.getSize()-1)/2 sera siempre entero
187     //el valor del tamaño del arreglo.
188
189     Array<string> dst_value_str(dim_array_aux); //Arreglo de valores(en strings) a
190     //transferir a usuarios destino.
191     Array <double> dst_value(dim_array_aux); //Arreglo de valores(en doubles) a
192     //transferir a usuarios destino.
193
194     size_t n=dim_array_aux; // args.getSize();
195
196     for(size_t i=2,j=0; j < n ;i+=2,j++)
197     {
198         //Se consiguen los hash de los usuarios destino y los valores a transferir
199         dst[j]=sha256(args[i-1]);

```

```

192     args[i-1]=dst[j];
194     dst_value_str[j]=args[i];
195     dst_value[j]=stod(dst_value_str[j]);
196
197     if(dst_value[j]<0)
198         return MSG_FAIL;
199     src_balance-=dst_value[j];
200     if(src_balance<0) //Si en algun momento los fondos del usuario fuente se
        terminan, se devuelve error.
        return MSG_FAIL;
202 }
203
204 //Al salir del for ya se tienen cargadas las estructuras con las addresses y
    los valores a transferirles
    //por lo que se crea un arreglo con la informacion de la transaccion.
206     txn aux_txn;
207
208     //Construccion del arreglo de inputs
    // Se debe buscar en la lista de transacciones del user src todos los outputs
    necesarios hasta completar la
210     //cantidad que se desea transferir: buscar en los outputs hasta conseguir aux-
        src_balance.
212     string str_user=users.find("user",src);
    user aux_user(str_user);
214
    Array<inpt> aux_arr_inputs;
216     aux_arr_inputs = aux_user.trackMoney(aux-src_balance);
    aux_txn.setNTxIn(aux_arr_inputs.getSize());
218     aux_txn.setTxIn(aux_arr_inputs);
    string aux_str=to_string(src_balance);
220     //Construccion del arreglo de outputs
    if(src_balance == 0)
222     {
        aux_txn.setNTxOut(dim_array_aux);
224     }
    else
226     {
        aux_txn.setNTxOut(dim_array_aux+1);
228         dst.ArrayRedim(dim_array_aux+1);
        dst[dst.getSize()-1]=src;
230         dst_value_str.ArrayRedim(dim_array_aux+1);
        dst_value_str[dst_value_str.getSize()-1]=aux_str;
232     }
    //Se agrega a los arreglos auxiliares el output necesario para el UTXO de src.
234
236     aux_txn.setTxOut(dst,dst_value_str);
237
238     mempool.addTxn(aux_txn);
239
240     //Se carga la transaccion a la lista de usuarios.
    for(size_t i=0; i< dim_array_aux;i++)
242     {
        str_user=users.find("user",dst[i]);
244
        if(str_user==FINDNT)
246        {
            user new_user;
248            new_user.setName(dst[i]);
            new_user.addTxn(aux_txn);
250            users.append(new_user);
        }
        else
252        {
            user aux_user(str_user);
254            users.removeElement(aux_user);
            aux_user.addTxn(aux_txn);
256            users.append(aux_user);

```

```

258     }
259 }
260 str_user=users.find("user",src);
261 if(str_user==FINDNT)
262 {
263     user new_user;
264     new_user.addTxn(aux_txn);
265     users.append(new_user);
266 }
267 else
268 {
269     user aux_user(str_user);
270     users.removeElement(aux_user);
271     aux_user.addTxn(aux_txn);
272     users.append(aux_user);
273 }
274 return sha256(sha256(aux_txn.toString()));
275 }
276
277 string cmdMine(Array <string> args)
278     //Ensambla y agrega a la Algochain un nuevo bloque a partir de todas las
279     transacciones en la mempool.
280     //La dificultad del minado viene dada por args
281 {
282     /*Valor de retorno.
283     Hash del bloque en caso de exito; FAIL en caso de falla por invalidez.
284     */
285     size_t bits = stoi(args[0]);
286     block aux_save;
287     if(bits<0)
288     {
289         cerr << "ERROR: dificultad invalida"<< endl;
290         exit(1);
291     }
292     block aux = algochain.getLastNode();
293     string prev_block = sha256(sha256(aux.toString()));
294     if(isHash(prev_block)==false)
295     {
296         cerr << "ERROR: al convertir prev block"<< endl;
297         exit(1);
298     }
299     mempool.setHeader(prev_block,bits);
300     aux_save = mempool; //lo guardo para despues imprimirlo
301     algochain.append(mempool);
302     block empty_block;
303
304     mempool = empty_block;
305     return sha256(sha256(aux_save.toString()));
306 }
307
308 string cmdBalance(Array <string> args)
309 {
310     string name = sha256(args[0]), balance_str;
311     if((balance_str = users.find(STR_BALANCE, name)) == FINDNT)
312         return "0";
313     return balance_str;
314 }
315
316 string cmdBlock(Array <string> args)
317 {
318     string id = args[0], block_str;
319     if((block_str = algochain.find(STR_BLOCK, id)) == FINDNT)
320         return "FAIL";
321     return block_str;
322 }
323
324 string cmdTxn(Array <string> args)
325 {

```

```

328 string id = args[0], txn_str;
329 if((txn_str = algochain.find(STR_TXN_IN_BLOCK_BY_HASH, id)) == FINDNT)
330 {
331     if((txn_str = findTxnInBlockByHash(id, mempool.toString())) == FINDNT)
332         return "FAIL";
333     else
334         return txn_str;
335 }
336 return txn_str;
337 }
338
339 string cmdLoad(Array <string> args)
340 {
341     fstream ifs_load;
342     istream *iss_load = 0;
343     ifs_load.open(args[0].c_str(), ios::in);
344     iss_load = &ifs_load;
345     if (!iss_load->good()) {
346         cerr << "cannot open "
347             << args[0]
348             << "."
349             << endl;
350         exit(1);
351     }
352     if(algochain.empty()==false)//la vacio si es necesario
353     {
354         list <block> empty_list;
355         algochain = empty_list;//
356     }
357     if(setAlgochainFromFile(iss_load)==false)
358     {
359         cerr << "ERROR: no se pudo cargar el archivo " << endl;
360         exit(1);
361     }
362     ifs_load.close();
363     block aux ;
364     aux = algochain.getLastNode();
365     return sha256(sha256(aux.toString()));
366 }
367
368 string cmdSave(Array <string> args)
369 {
370     fstream ofs_save;
371     ostream *oss_save = 0;
372     ofs_save.open(args[0].c_str(), ios::out);
373     oss_save = &ofs_save;
374     if (!oss_save->good())
375     {
376         cerr << "cannot open "
377             << args[0]
378             << "."
379             << endl;
380         exit(1); // EXIT: Terminación del programa en su totalidad
381     }
382     else if (oss_save->bad())
383     {
384         cerr << "cannot write to output stream."
385             << endl;
386         exit(1);
387     }
388     *oss_save << algochain;//le agrego toString?
389     ofs_save.close();
390     return "Carga realizada con éxito";
391 }
392
393 Array <string> parseCmdArgs(string str, size_t N)//funcion para verificar la
394     correcta escritura de los argumentos,
395     // devuelve un array dinamico con los argumentos.

```

```
396 // Le paso N para saber cuantos argumentos debo mirar para
    // que sea correcto
    // Si es N=0 entonces es argumentos variables impares minimo
    3
398 {
    istream iss(str);
400 size_t i=0;
    string aux;
402
    Array <string> args(1); // Lo dimensiono con 3 porque es el valor minimo,
        // despues lo resize
404 while(getline(iss, aux, ' ').good() || iss.eof() )//averiguar que pasa con el
        // ultimo
    {
406         if(aux == "")
        {
408             cerr << "Hay un espacio de mas" << endl; // poner error
            exit(1);
410         }
        if (i>=args.getSize())
412         {
            args.ArrayRedim(args.getSize()+1);
414         }
        if(iss.eof()==true)
416         {
            args[i]=aux;
418             break;
        }
        args[i]=aux;
420         i++;
422     }
    if(N!=(i+1) && N!=0)
424     {
        cerr << "Numero incorrecto de argumentos"<< endl;
426         exit(1);
    }
    return args;
428 }
430 #endif //_DICTIONARY_H_
```


Listing 6: finders.h

```

1  #ifndef _FINDERS_H_
2  #define _FINDERS_H_
3
4  #define FINDNT "Findnt"
5
6  //Para definir las referencias de busqueda
7  #define STR_BALANCE "balance"
8  #define STR_TRANSACTIONS "transactions"
9  #define STR_BLOCK "block"
10 #define STR_CHECK_USER "checkUser"
11 #define STR_USER "user"
12 #define STR_TXN_BY_HASH "txnByHash"
13 #define STR_TXN_IN_BLOCK_BY_HASH "findTxnInBlockByHash"
14
15 #include <iostream>
16 #include "user.h" //Necesario para buscar dentro de la lista de usuarios.
17 #include "block.h" //Necesario para crear la lista de transacciones de cada
    usuario.
18
19 using namespace std;
20 //-----MACROS
21
22 //Para contar la cantidad de finders que se tiene. Se utiliza para encontrar el
    que se necesita en
23 //cada situacion
24 #define MAXFINDER 7
25
26
27 //-----FINDERS
28
29 //Los finders buscan la informacion especifica pedida (como un id o un value de
    cierto
30 // user) y la devuelven en una string
31 typedef string (*finder)(string d,string str); //Buscan en un la string str dato
    d
32
33 //Recordar modificar la macro MAXFINDER al agregar nuevas funciones aqui
34
35 string findBalance(string d, string str);
36 string findTransactions(string d, string str);
37 string checkUser(string d, string str);
38 string findUser(string d, string str);
39 string findTxnByHash(string d, string str);
40
41 string findTransactions(string d, string str)
42 {
43     //str debe ser el contenedor de la transaccion. En este caso el user.
44
45     //Esta funcion devuelve todas las transacciones del usuario como una string.
46     user aux_user(str);
47     string result, aux;
48     aux = aux_user.getName();
49     if(d == aux)
50     {
51         return aux_user.getTransactions().toString();
52         // return (aux_user.getTransactions()).toString(); //AGREGAR ESTA LINEA
            CUANDO SE HAYA TERMINADO LA FUNCION
53     }
54     else
55     {
56         return "Findnt";
57     }
58 }
59 string findBalance(string d, string str)
60 {
61     //Se recorren todos los outputs de todas las transacciones realizadas buscando
    //la ultima aparicion del usuario especificado para devolver el valor que
    quedo en output.

```

```

        //Seria bueno agregar unos metodos mas en la clase outpt que sean
        getValueAsString()
62    //y getAddr()

64    user aux_user(str);
    string result, aux;
66    aux = aux_user.getName();
    if(d == aux)
68        return to_string(aux_user.getBalance());
    else
70        return "Findnt";
}

72
74    string findBlock(string d, string str)
{
    string aux;
76    aux=sha256(sha256(str));
    if(aux==d)
78        return str;
    else
80        return "Findnt";
}

82
84    string checkUser(string d, string str)
{
    user aux_user(str);
86    if(d == aux_user.getName())
        return "TRUE";
    else
88        return "Findnt";
}

90
92    string findUser(string d, string str)
{
    user aux_user(str);
94    if(d == aux_user.getName())
        return aux_user.toString();
    else
96        return "Findnt";
}

98
100    string findTxnByHash(string d, string str)
{
102    txn tran(str);
    if(d == sha256(sha256(tran.toString())))
104        return tran.toString();
    else
106        return "Findnt";
}

108
110    string findTxnInBlockByHash(string d, string str)
{
112    block blk(str);
    bdy body;
114    Array<txn> txns;
    body = blk.getBody();
116    txns = body.getTxns();
    for (size_t i = 0; i < body.getTxnCount(); i++)
118    {
        if(d == sha256(sha256(txns[i].toString())))
120        {
            return txns[i].toString();
122        }
    }

124    return FINDNT;
126}

128 //Completar con las definiciones de todos los finders necesarios
struct finder_option_t

```

```
130 {  
131     string reference;  
132     finder fndr;  
133 };  
134  
135 static finder_option_t dictionary_finder[] = {  
136     {STR_BALANCE, findBalance},  
137     {STR_TRANSACTIONS, findTransactions},  
138     {STR_BLOCK, findBlock},  
139     {STR_CHECK_USER, checkUser},  
140     {STR_USER, findUser},  
141     {STR_TXN_BY_HASH, findTxnByHash},  
142     {STR_TXN_IN_BLOCK_BY_HASH, findTxnInBlockByHash}  
143 };  
144  
145 finder finderParse( string ref)  
146 {  
147     string aux;  
148     int i = 0;  
149  
150     while(ref != dictionary_finder[i].reference && i < MAXFINDER) i++;  
151  
152     if(i == MAXFINDER)  
153     {  
154         cerr << "El finder no es valido" << endl;  
155         exit(1);  
156     }  
157     return dictionary_finder[i].fndr;  
158 }  
#endif //_FINDERS_H_
```

Listing 7: Main.cc

```
1  #include <fstream>
2  #include <iostream>
3  #include <sstream>
4  #include <cstdlib>
5  #include <iomanip>
6  using namespace std;
7
8  #include "Lista.h"
9  #include "cmdline.h"
10 #include "sha256.h"
11 #include "block.h"
12 #include "dictionary.h"
13 #include "main.h"
14
15 int main(int argc, char * const argv[])
16 {
17     cmdline cmdl(options);
18     cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline
19
20     if (iss->bad()) {
21         cerr << "cannot read from input stream."
22              << endl;
23         exit(1);
24     }
25
26     string aux, aux1, aux2;
27     Array <string> arr;
28     p_func cmd;
29     int num_param;
30
31     cout << "Empieza el programa" << endl;
32     while(aux1 != "exit")
33     {
34         getline(*iss, aux1, ' ');
35         if(aux1=="Exit" || aux1=="exit") break;
36         cmd = dictCmds(aux1, num_param);
37         getline(*iss, aux2, '\n');
38         arr = parseCmdArgs(aux2, num_param);
39         aux = cmd(arr);
40         *oss << aux << endl;
41     }
42     cout << "Termina el programa" << endl;
43
44     return 0;
45 }
```

Listing 8: Main.h

```

1  #ifndef _MAIN_H_
2  #define _MAIN_H_
3
4
5  #include <fstream>
6  #include <iomanip>
7  #include <iostream>
8  #include <sstream>
9  #include <cstdlib>
10 #include "cmdline.h"
11
12 #include "Lista.h"
13 #include "block.h"
14 #include "Array.h"
15 #include "user.h"
16
17 using namespace std;
18
19
20 /***** Elementos globales *****/
21 static void opt_input(string const &);
22 static void opt_output(string const &);
23 static void opt_help(string const &);
24
25 bool setAlgochainFromFile( istream *iss);
26 bool refreshUsersFromBlock(block );
27 block mempool;
28 list <block> algochain;
29 list <user> users;
30
31 // Tabla de opciones de línea de comando. El formato de la tabla
32 // consta de un elemento por cada opción a definir. A su vez, en
33 // cada entrada de la tabla tendremos:
34 //
35 // o La primera columna indica si la opción lleva (1) o no (0) un
36 // argumento adicional.
37 //
38 // o La segunda columna representa el nombre corto de la opción.
39 //
40 // o Similarmente, la tercera columna determina el nombre largo.
41 //
42 // o La cuarta columna contiene el valor por defecto a asignarle
43 // a esta opción en caso que no está explícitamente presente
44 // en la línea de comandos del programa. Si la opción no tiene
45 // argumento (primera columna nula), todo esto no tiene efecto.
46 //
47 // o La quinta columna apunta al método de parseo de la opción,
48 // cuyo prototipo debe ser siempre void (*m)(string const &arg);
49 //
50 // o La última columna sirve para especificar el comportamiento a
51 // adoptar en el momento de procesar esta opción: cuando la
52 // opción es obligatoria, deberá activarse OPT_MANDATORY.
53 //
54 // Además, la última entrada de la tabla debe contener todos sus
55 // elementos nulos, para indicar el final de la misma.
56 //
57
58 static option_t options[] = {
59     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
60     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
61     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
62     {0, },
63 };
64
65
66 static istream *iss = 0; // Input Stream (clase para manejo de los flujos de
67                          // entrada)
68 static ostream *oss = 0; // Output Stream (clase para manejo de los flujos de

```

```

        salida)
68 static fstream ifs;      // Input File Stream (derivada de la clase ifstream que
        deriva de istream para el manejo de archivos)
static fstream ofs;      // Output File Stream (derivada de la clase ofstream que
        deriva de ostream para el manejo de archivos)
70
static void
72 opt_input(string const &arg)
{
74     // Si el nombre del archivos es "-", usaremos la entrada
    // estándar. De lo contrario, abrimos un archivo en modo
76     // de lectura.
    //
78     if (arg == "-") {
        iss = &cin;    // Establezco la entrada estandar cin como flujo de entrada
80     }
    else {
82         ifs.open(arg.c_str(), ios::in); // c_str(): Returns a pointer to an array
        that contains a null-terminated
        // sequence of characters (i.e., a C-string) representing
84         // the current value of the string object.
        iss = &ifs;
86     }

88     // Verificamos que el stream este OK.
    //
90     if (!iss->good()) {
        cerr << "cannot open "
92             << arg
             << ". "
94             << endl;
        exit(1);
96     }
}

98
static void
100 opt_output(string const &arg)
{
102     // Si el nombre del archivos es "-", usaremos la salida
    // estándar. De lo contrario, abrimos un archivo en modo
104     // de escritura.
    //
106     if (arg == "-") {
        oss = &cout;    // Establezco la salida estandar cout como flujo de salida
108     } else {
        ofs.open(arg.c_str(), ios::out);
110         oss = &ofs;
    }

112     // Verificamos que el stream este OK.
    //
114     if (!oss->good()) {
        cerr << "cannot open "
116             << arg
             << ". "
118             << endl;
120         exit(1);    // EXIT: Terminación del programa en su totalidad
    }
122 }

124
static void
opt_help(string const &arg)
126 {
    cout << "Comands: \n"
128     << "init <user> <value> <bits> \n"
    << "transfer <src> <dst1> <value1> ... <dstN> <valueN> \n"
130     << "mine <bits> \n"
    << "balance <user> \n"
132     << "block <id> \n"
    << "txn <id> \n"

```

```

134     << "load <filename> \n"
135     << "save <filename> \n"
136     << "exit program \n"
137     << endl;
138     exit(0);
139 }
140
141 bool setAlgochainFromFile( istream *iss_load)
142 {
143     block block_aux, block_empty;
144     string str, str_aux;
145     size_t i = -1; //, aux = 0;
146     hdr header_aux;
147     size_t diff, nonce;
148     bdy body_aux;
149     getline(*iss_load, str, '\n');
150     if(str!=NULL_HASH)
151     {
152         cerr << "ERROR: No comienza con el genesis block" << endl;
153         exit (1);
154     }
155     while (str!="")
156     {
157         //seteo el header
158         i++;
159
160         if(isHash(str)==false)
161         {
162             cerr << "ERROR: no es un hash para prev block" << endl;
163             exit(1);
164         }
165         header_aux.setPrevBlock(str);
166         getline(*iss_load, str, '\n');
167         if(isHash(str)==false)
168         {
169             cerr << "ERROR: no es un hash para txns hash" << endl;
170             return false;
171         }
172         header_aux.setTxnsHash(str);
173         getline(*iss_load, str, '\n');
174         if(isNumber<size_t>(str)==0)
175         {
176             cerr<<"ERROR: no es un numero"<< endl;
177             return false;
178         }
179         diff = stoi(str);
180         header_aux.setBits(diff);
181         getline(*iss_load, str, '\n');
182         if(isNumber<size_t>(str)==0)
183         {
184             cerr<<"ERROR: no es un numero"<< endl;
185             return false;
186         }
187         nonce = stoi(str);
188         header_aux.setNonce(nonce);
189
190         block_aux.setHeader(header_aux); //guarda el header
191         //seteo el body
192         str_aux=block_aux.setBody(iss_load);
193         // chequeo que sea genesis
194         if(i==0)
195         {
196             body_aux = block_aux.getBody();
197             if (body_aux.getTxnCount() !=1)
198             {
199                 cerr << "ERROR: No comienza con el genesis block" << endl;
200                 exit (1);
201             }
202             Array <txn> txns_aux = body_aux.getTxns();
203             if(txns_aux[0].getNTxnIn() !=1 || txns_aux[0].getNTxnOut() !=1 )

```

```

204     {
205         cerr << "ERROR: No comienza con el genesis block" << endl;
206         exit (1);
207     }
208     Array <inpt> tx_in_aux = txns_aux[0].getInputs();
209     outpnt outpoint = tx_in_aux[0].getOutPoint();
210     if(outpoint.tx_id!=NULL_HASH && outpoint.idx!=0)
211     {
212         cerr << "ERROR: No comienza con el genesis block" << endl;
213         exit (1);
214     }
215 }
216
217 if(isHash(str_aux)==true || str_aux=="")//volver a los if separados y ver
218 si es break o continue se rompio mi crerbo
219 {
220     str=str_aux;
221     algochain.append(block_aux);
222     if(refreshUsersFromBlock(block_aux)==false)
223     {
224         cerr<< "ERROR: no se pueden cargar los users"<< endl;
225         exit(1);
226     }
227     break;
228 }
229 else if (str_aux=="OK")
230 {
231     algochain.append(block_aux);
232     if(refreshUsersFromBlock(block_aux)==false)
233     {
234         cerr<< "ERROR: no se pueden cargar los users"<< endl;
235         exit(1);
236     }
237     getline(*iss_load, str, '\n');
238     continue;
239 }
240 else
241 {
242     return false;
243 }
244 }
245 return true;
246 }
247
248 bool refreshUsersFromBlock(block blk)
249 {
250     bdy body = blk.getBody();
251     size_t txn_count = body.getTxnCount(), n_tx_in, n_tx_out;
252     Array <txn> txns = body.getTxns();
253     Array <inpt> inpts;
254     Array <outpt> outpts;
255     string addr;
256     list <string> address, empty_list;
257     for(size_t i =0 ; i < txn_count ; i++)
258     {
259         n_tx_in = txns[i].getNTxIn();
260         inpts = txns[i].getInputs();
261
262         for (size_t j = 0; j < n_tx_in; j++)
263         {
264             if(j==0)
265             {
266                 addr = inpts[j].getAddr();
267                 continue;
268             }
269             else
270             {
271                 if(inpts[j].getAddr() !=addr)
272                 {

```



```
                cerr << "ERROR: Addr en inputs es distinto" << endl;
274         return false;
        }
276     }
    }

278     if(users.find("checkUser", addr)!=FINDNT)
280     {
        string str_user=users.find("user", addr);
282         user aux_user(str_user);
        users.removeElement(aux_user);
284         aux_user.loadTxn( txns[i]);
        users.append(aux_user);
286     }
    else
288     {
        if (addr != NULL_HASH )
290        {
            user aux_user;
292            aux_user.setName(addr);
            aux_user.loadTxn(txns[i]);
294            users.append(aux_user);
        }
296    }
    //con los outputs
298    n_tx_out = txns[i].getNTxOut();
    outputs = txns[i].getOutputs();
300

    for (size_t j = 0; j < n_tx_out; j++)
302    {
        addr = outputs[j].getAddr();
304        if (addr ==inpts[0].getAddr() )
        {
306        }
        else if(address.contains(addr)==true)
308        {
            cerr << "ERROR: Addr en outputs repetidas" << endl;
310            return false;
        }
312        else if (address.contains(addr)!=true || users.find("checkUser", addr)==
        FINDNT)
        {
314            addr = outputs[j].getAddr();
            address.append(addr);
316            user aux_user;
            aux_user.setName(addr);
318            aux_user.loadTxn(txns[i]);
            users.append(aux_user);
320        }
        else
322        {
            string str_user=users.find("user", addr);
324            user aux_user(str_user);
            users.removeElement(aux_user);
326            aux_user.loadTxn(txns[i]);
            users.append(aux_user);
328        }
    }
330    address = empty_list;
}
332    return true;
}
334 #endif //MAIN_H
```

Listing 9: lista.h

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_

3
4  #include "math.h"
5  #include "prototypeLista.h"
6  #include "finders.h"

7
8  // Implementacion de list doblemente enlazada no circular utilizando
9  // templates e iteradores.

10
11 //Se aprovecha el hecho de que la lista tiene una referencia del principio y el
12 // final para reducir a la mitad
13 //el procesamiento de las altas,bajas y borrado de las listas.

14 //Los nodos son creados con memoria dinamica.

15
16 template<typename T>
17 list<T>::list(){first=NULL;last=NULL;max_size=0;}

18
19 template<typename T>
20 list<T>::list(const list &orig) : first(0), last(0), max_size(orig.max_size)
21 {
22     node *iter;
23     node *ant;

24
25     // Recorremos la secuencia original en sentido directo. En cada paso,
26     // creamos un nodo, copiando el dato correspondiente, y lo enganchamos
27     // al final de nuestra nueva lista.
28     //
29     for (iter = orig.last, ant = 0; iter != 0; iter = iter->next)
30     {
31         // Creamos un nodo, copiando el dato, y lo enganchamos en e
32         // final de nuestra lista.
33         //
34         node *new_node = new node(iter->data);
35         new_node->prev = ant;
36         new_node->next = 0;

37
38         // Si ésta no es la primera pasada, es decir, si no se trata
39         // del primer nodo de la lista, ajustamos el enlace sig_ del
40         // nodo anterior.
41         //
42         if (ant != 0)
43             ant->next = new_node;

44
45         // Además, tenemos que ajustar los punteros a los elementos
46         // distinguidos de la secuencia, primero y último. En el caso
47         // de pri_ (enlace al primer elemento), esto lo vamos a
48         // hacer una única vez; para el caso de ult_, iremos tomando
49         // registro del último nodo procesado, para ajustarlo antes
50         // de retornar.
51         //
52         if (first == 0)
53             first = new_node;
54         ant = new_node;
55     }

56
57     // Ajustamos el puntero al último elemento de la copia.
58     last = ant;
59 }

60
61 template<typename T>
62 bool list<T>::empty()
63 {
64     //Forma complicada de preguntar si la lista esta vacia
65     // if((this->first==this->last)&&(this->first==NULL))
66     if(!max_size)

```

```
68     return true;
69     return false;
70 }
71
72
73
74 template<typename T>
75 list<T>::~~list()
76 {
77     for (node *p = first; p; )
78     {
79         node *q = p->next;
80         delete p;
81         p = q;
82     }
83 }
84
85 template<typename T>
86 size_t list<T>::size(){return this->max_size;}
87
88 template<typename T>
89 void list<T>::append(const T& t)
90 {
91     node *aux= new node(t);
92
93     if( this->empty() )
94     {
95         aux->next=NULL;
96         aux->prev=NULL;
97         this->last=aux;
98         this->first=aux;
99     }
100    else
101    {
102        aux->next=NULL;
103        aux->prev=this->last;
104        this->last->next=aux;
105        this->last=aux;
106    }
107    this->max_size=this->max_size+1;
108 }
109
110
111 template<typename T>
112 void list<T>::insert(const T& t)
113 {
114     node *aux= new node(t);
115
116     if( this->empty() )
117     {
118         aux->next=NULL;
119         aux->prev=NULL;
120         this->last=aux;
121         this->first=aux;
122     }
123    else
124    {
125        aux->prev=NULL;
126        aux->next=this->first;
127        this->first->prev=aux;
128        this->first=aux;
129    }
130    this->max_size=this->max_size+1;
131 }
132
133
134 template <typename T>
135 bool list<T>::placeElement(const T& t, size_t n)
136 {
137     if(n>max_size)
```

```

138     return false;
140     node*aux=new node(t);
141     node* p_aux;
142
143     if(n==1)
144         this->insert(t);
145     else if(n==this->max_size)
146         this->append(t);
147     else if(n<=floor(this->max_size)/2)
148     {
149         aux=this->first;
150         aux->prev=NULL;
151         for(size_t i=1; i<n ;i++)
152         {
153             p_aux=aux;
154             aux=aux->next;
155             aux->prev=p_aux;
156         }
157     }
158     else if(n>floor(this->max_size)/2)
159     {
160         aux=this->last;
161         aux->next=NULL;
162         for(size_t i=1; i<n ;i++)
163         {
164             p_aux=aux;
165             aux=aux->prev;
166             aux->next=p_aux;
167         }
168     }
169     else
170         return false; //Si por algún motivo no pudo realizarse ninguna de las
acciones previas, devuelve error.
171     return true;
172 }
173
174 template<typename T>
175 T list<T>::find(const T& t)
176 {
177     node* prev_;
178     node* aux;
179
180
181     if(this->empty())
182     {
183         return NULL;
184     }
185     else
186     {
187         aux=this->last;
188         aux->next=this->last->next;
189         aux->prev=this->last->prev;
190
191         if(aux->data==t) //Si se encuentra en el ultimo, se devuelve el dato
contenido en el ultimo.
192         {
193             return aux->data;
194         }
195
196         for(size_t i=this->max_size; i>=1;i--)
197         {
198             prev_=aux->prev;
199             if(!prev_)
200             {
201                 return NULL;
202             }
203
204             prev_->next=aux;

```

```

206     prev_ -> prev = aux -> prev -> prev;
208     //Se comprueba si el dato buscado está en nodo anterior
209     if(prev_ -> data == t)
210     {
211         return prev_ -> data;
212     }
213     //Se retrocede en la lista
214     aux = prev_;
215     aux -> next = prev_ -> next;
216     aux -> prev = prev_ -> prev;
217 }
218 }
219 return NULL; //Si se hubo algun error se devuelve NULL.
220 }
221
222 template<typename T>
223 string list<T>::toString()
224 {
225     ostringstream ss;
226     ss << *this;
227     return ss.str();
228 }
229
230 template<typename T>
231 string list<T>::find(const string& ref, const string& d)
232 {
233     finder aux_finder;
234     aux_finder = finderParse(ref);
235     string result;
236
237     node* prev_;
238     node* aux;
239
240     if(this -> empty())
241     {
242         return "Findnt";
243     }
244     else
245     {
246         aux = this -> last;
247         aux -> next = this -> last -> next;
248         aux -> prev = this -> last -> prev;
249         if((result = aux_finder(d, (aux -> data).toString())) != "Findnt") //Si se
250             encuentra en el ultimo, se devuelve el dato contenido en el ultimo.
251         {
252             return result;
253         }
254
255         for(size_t i = this -> max_size; i >= 1; i--)
256         {
257             prev_ = aux -> prev;
258             if(!prev_)
259             {
260                 return "Findnt";
261             }
262
263             prev_ -> next = aux;
264             prev_ -> prev = aux -> prev -> prev;
265
266             if((result = aux_finder(d, (prev_ -> data).toString())) != "Findnt")
267             {
268                 return result;
269             }
270             aux = prev_;
271             aux -> next = prev_ -> next;
272             aux -> prev = prev_ -> prev;
273         }
274     }

```

```

    return "Findnt";
276 }

278
280 template<typename T>
281 void list<T>::removeElement(const T &t)
282 {
283     node *iter, *next_=0;
284
285     for (iter = first; iter != 0; iter = next_)
286     {
287         next_ = iter->next;
288         if (t == iter->data)
289         {
290             node *prev = iter->prev;
291             if (prev == 0)
292                 first = next_;
293             else
294                 prev->next = next_;
295             if (next_ == 0)
296                 last = prev;
297             else
298                 next_>prev = prev;
299             delete iter;
300
301             max_size--;
302             break;
303         }
304     }
305 }
306
307 template <typename T>
308 void list<T>::show(ostream& oss) {
309     if(this->empty())
310     {
311         oss << "Lista vacia" << endl;
312         return;
313     }
314     if(first == NULL){
315         oss << "NULL";
316         return;
317     }
318     node* now = first;
319     while(now->next != NULL){
320         oss << now->data << endl;
321         now = now->next;
322     }
323     oss << now->data << endl;
324 }
325 template<typename T>
326 T list<T>::getFirstNode()
327 {
328     return this->first->data;
329 }
330
331 template<typename T>
332 T list<T>::getLastNode()
333 {
334     return this->last->data;
335 }
336
337 template<typename T>
338 list<T> const &list<T>::operator=(list const &orig)
339 {
340     node *iter_;
341     node *next_;
342     node *prev_;
343
344     if (this != &orig)

```

```
346     {
348         for (iter_ = first; iter_ != 0; )
350         {
352             next_ = iter_->next;
354             delete iter_;
356             iter_ = next_;
358         }
360
362         first = 0;
364         last = 0;
366
368         for (iter_ = orig.first, prev_ = 0; iter_ != 0; iter_ = iter_->next)
370         {
372             node *new_node = new node(iter_->data);
374             new_node->prev = prev_;
376             new_node->next = 0;
378             if (prev_ != 0)
380                 prev_->next = new_node;
382             if (first == 0)
384                 first = new_node;
386             prev_ = new_node;
388         }
390         last = prev_;
392         max_size = orig.max_size;
394     }
396
398     return *this;
399 }
400
401 template<typename T>
402 bool list<T>::contains(const T &elem) const
403 {
404     node *iter;
405
406     for (iter = first; iter != 0; iter = iter->next)
407         if (elem == iter->data)
408             return true;
409     return false;
410 }
411
412 #endif // _LIST_H_
```

Listing 10: listaPrototype.h

```

1  #ifndef _PROTOTYPELISTA_H_
2  #define _PROTOTYPELISTA_H_
3  template<class T>
4  class list
5  {
6      class node
7      {
8          // La list puede tener acceso a los atributos privados del nodo para poder
9          // modificarlos, dado que la clase nodo es inherente a la clase list.
10
11          T data;
12          node* next;
13          node* prev;
14          friend class list;
15
16      public:
17          node(T const &data_){data=data_ ; next= NULL; prev=NULL;}; //Constructor a
18          // partir de dato.
19          // Se asume que siempre que se quiera crear un nodo se tendrá la información
20          // que va a contener.
21          node* getNext(){return this->next;}
22          node* getPrev(){return this->prev;}
23          T getData(){return this->data;}
24          ~node(){}; //Destructor.
25      };
26
27      node* first;
28      node* last;
29      size_t max_size; //Tamaño de la lista. Si la lista no esta vacia max_size>=1
30                        // Si la lista esta vacia max_size=0
31
32      public:
33          list(); //Constructor basico
34          list(const list &); //Constructor en base a otra list.
35          ~list(); //Destructor
36          void append(const T& t); //Agregar nodo al final de la list.
37          void insert(const T& t); //Agregar nodo al principio de la list.
38          bool placeElement(const T& t, size_t n=1); //Agregar nodo en la posición n de
39              // la list. Si no lo pudo agregar
40              // devuelve false. Por defecto lo agrega al principio.
41          bool empty(); //Verifica si la list está vacia.
42          T find(const T& t); //Encuentra el ultimo nodo que contiene el dato T. Si no
43              // lo encuentra, debe devolver NULL.
44              // DEBERIA devolver un puntero a un dato dentro de un nodo de la
45              // lista, DEBE devolver un puntero a un dato constante.
46          string find(const string& ref, const string& d ); //Encuentra el dato "d" de
47              // tipo "ref" en su última aparición en la lista.
48              // Ejemplo: ref=valor d=Carla. Devuelve una string con
49              // el el ultimo output de Carla
50              // Ejemplo: ref= id d=<valor del hash>. Devuelve el
51              // bloque como string.
52              // Si no lo encuentra, devuelve una cadena vacia.
53          void removeElement(const T& t); //Elimina el primer nodo que contiene al dato
54              // t. Devuelve false si no pudo eliminarlo.
55          size_t size(); //Obtiene el tamaño de la list
56          // list const &operator=(const list& other_list);
57          void show(ostream&);
58          friend ostream& operator<<(ostream& oss, list& l)
59          {
60              l.show(oss);
61              return oss;
62          }
63
64          list const &operator=(list const &);
65          string toString();
66          T getFirstNode();
67          T getLastNode();
68          bool deleteFirstNode();
69          bool deleteLastNode();

```



```
60 |     bool contains(const T &elem) const;  
    | };  
62 | #endif //_PROTOTYPELISTA_H_
```

Listing 11: user.h

```

1  #ifndef _USER_H_
2  #define _USER_H_

3
4  #include "finders.h"
5  #include "block.h"
6  #include "Lista.h"
7  #include "Array.h"
8
9
10 class user
11 {
12     string name;
13     double balance;
14     list<txn> transactions; //Solo se guardan las transacciones donde user
15     aparece como output.
16     public:
17     user();
18     user(string);
19     ~user();
20     string getName();
21     double getBalance();
22     list<txn>& getTransactions();
23
24     void setName(const string &);
25     void setBalance(const double &);
26     void setTransactions(const list<txn> &);
27     void show(ostream&);
28
29     string toString();
30     friend ostream& operator<<(ostream& oss, user& usr)
31     {
32         usr.show(oss);
33         return oss;
34     }
35
36     user const &operator=(user const &);
37     Array<inpt> trackMoney(const double);
38     void addTxn(const txn);
39     void loadTxn(txn tran);
40
41     bool operator==(const user &) const;
42 };
43
44 user::user()
45 {
46     name = "";
47     balance = 0;
48 }
49
50 user::~~user()
51 {
52 }
53
54 user::user(string str_user)
55 {
56     //Para crear el usuario a partir de una string, se considera que dicha string
57     contiene todos los campos de user separados
58     //por fines de linea (\n): name,balance y transactions.
59     istream ss(str_user);
60     bdy aux_body;
61
62     string aux_str;
63     txn aux_txn;
64     getline(ss, this->name, '\n');
65     getline(ss, aux_str, '\n');
66     balance = stod(aux_str);
67     aux_body.setTxns(&ss);
68 }

```

```

68     Array<txn> array_aux_txns=aux_body.getTxns();
69     for(size_t i=0; i<aux_body.getTxnCount(); i++)
70     {
71         this->transactions.append(array_aux_txns[i]);
72     }
73 }
74 string user::getName()
75 {
76     return name;
77 }
78 double user::getBalance(){return balance;}
79 list<txn>& user::getTransactions(){return transactions;}
80
81 void user::setName(const string & str){name = str;}
82 void user::setBalance(const double & n){balance = n;}
83 void user::setTransactions(const list<txn> & d){transactions = d;}
84
85 void user::show(ostream& oss)
86 {
87     if(name == "")
88         return ;
89     oss << name << endl;
90
91     oss << balance << endl;
92
93     oss << transactions;
94 }
95
96 string user::toString()
97 {
98     ostringstream ss;
99     ss << *this;
100     return ss.str();
101 }
102
103 user const &user::operator=(user const &right)
104 {
105     if(&right != this)
106     {
107         name = right.name;
108         balance = right.balance;
109         transactions = right.transactions;
110         return *this;
111     }
112     return *this;
113 }
114
115 Array<inpt> user::trackMoney(const double money)
116 {
117     txn aux_txn;
118     size_t inpt_iter = 0;
119     size_t i;
120     Array<outpt> aux_outputs;
121     Array<inpt> inputs;
122     double utxo = 0;
123     while(utxo < money)
124     {
125         aux_txn = transactions.getLastNode();
126         aux_outputs = aux_txn.getOutputs();
127         for (i = 0; i < aux_txn.getNTxOut(); i++)
128         {
129             if(aux_outputs[i].getAddr() == name)
130             {
131                 utxo += stod(aux_outputs[i].getValue());
132                 break;
133             }
134         }
135         inputs[inpt_iter].setInput(sha256(sha256(aux_txn.toString()))), i, name);
136         transactions.removeElement(aux_txn);

```

```
        inpt_iter++;
138    }
    balance -= utxo;
140    inputs.ArrayRedim(inpt_iter);
    return inputs;
142 }

144 void user::addTxn(txn tran)
{
146     Array<inpt> aux_inpt;
    transactions.append(tran);
148     if(name != tran.getInputs()[0].getAddr())
    {
150         for (size_t i = 0; i < tran.getNTxOut(); i++)
        {
152             if(name == tran.getOutputs()[i].getAddr())
            {
154                 balance += stod(tran.getOutputs()[i].getValue());
                break;
156             }
        }
158     }
    else
160     {
        double spent_value = 0;
162         for (size_t i = 0; i < tran.getNTxOut(); i++)
        {
164             if(tran.getOutputs()[i].getAddr() == name)
                continue;
166             else
            {
168                 spent_value += stod(tran.getOutputs()[i].getValue());
            }
170         }
        balance -= spent_value;
172     }
}

174 void user::loadTxn(txn tran)
176 {
    string aux_str_txn;
178     double source_value = 0, spent_value = 0, change;
    Array<inpt> inputs = tran.getInputs();
180     Array<outpt> outputs = tran.getOutputs();
    if(name == inputs[0].getAddr())
182     {
        for (size_t i = 0; i < tran.getNTxIn(); i++)
184         {
            if((aux_str_txn = transactions.find(STR_TXN_BY_HASH, inputs[i].getOutPoint()
            ().tx_id)) == "Findnt")
186             {
                cerr << "Error en la carga del user: < " << name << ">" << endl;
288                 exit(1);
            }
190             txn aux_txn(aux_str_txn);
            transactions.removeElement(aux_txn);

192             if(aux_txn.getOutputs()[inputs[i].getOutPoint().idx].getAddr() != name)
            {
194                 cerr << "Error en la carga del user: << " << name << ">>" << endl;
                exit(1);
196             }
            source_value += stod(aux_txn.getOutputs()[inputs[i].getOutPoint().idx].
            getValue());
        }
200         for (size_t i = 0; i < tran.getNTxOut(); i++)
        {
202             spent_value += stod(outputs[i].getValue());
            if(outputs[i].getAddr() == name)
204                 change = stod(outputs[i].getValue());
        }
    }
}
```

```
    }
206
    if(spent_value != source_value)
208    {
        cerr << "Error en la carga del user: <<< " << name << ">>>" << endl;
210        exit(1);
    }
212
    balance = balance - spent_value + change;
214
}
else
216
{
    for (size_t i = 0; i < tran.getNTxOut(); i++)
218    {
        if(outputs[i].getAddr() == name)
220        {
            balance += stod(outputs[i].getValue());
222            break;
        }
224    }
}
226
transactions.append(tran);
}
228

bool user::operator==( const user & right) const
230
{
232    if(name == right.name && balance == right.balance)
        return true;
234    else
        return false;
236
}
#endif //_USER_H_
```

Listing 12: tools.h

```

1  #ifndef TOOLS_H
2  #define TOOLS_H

4  #include <iostream>
5  #include <string.h>
6  #include <cstring>
7  #include <sstream>
8  #include <cstdlib>
9  #include <bitset>
10 #include "block.h"
11 #include "Lista.h"
12 #include "sha256.h"

14 using namespace std;

16 static const string ERROR="Error"; //Se usa a modo de macro.

18 string Hex2Bin(const string& s); // Transforma una cadena de caracteres que
    contiene un numero en Hexa a una en binario
19 bool isHash(const string& str); // Confirma si str cumple con los requisitos
    minimos de un HASH
20 bool isNumber(const string& s); // Revisa si s es un numero, implementado como
    template para distinguir si es int o double, etc...
21 bool isError(const string& addr); // Se fija si en addr esta lo guardado en la
    variable ERROR
22 Array<string> _merkle_hash(Array<string>& hashes, size_t n);
23 string merkle_hash(Array<string>& hashes, size_t n);

24 bool setAlgochainFromFile( istream *iss);

26 bool isError(const string& addr)
27 {
28     if(addr==ERROR)
29         return false;
30     else
31         return true;
32 }

34 string Hex2Bin(const string& s) //transforma de hexa a binario una string, maximo
    4 bytes, 8 char y 32 bits
35 {
36     stringstream ss;
37     ss << hex << s;
38     unsigned n;
39     ss >> n;
40     bitset<4> b(n); //32 es el maximo por el unsigned
41
42     return b.to_string(); // .substr(32 - 4*(s.length()));
43 }
44 bool isHash(const string& str) //Devuelve false (0) si no es un hash, true (1)
    si lo es.
45 {
46     int n;
47     if(str.length()!=64) //El hash devuelve una string de 64 chars. Es siempre de
        largo fijo.
48         return false;
49     for (size_t i = 0; i < str.length(); i++)
50     {
51         n=(int)(tolower(str[i])-'0');
52         if(n<0 || (n>9&& n<17) || (n>22 && n<49) || n>63) //se verifica si el char de la
            string es un numero hexa
53             return false;
54     }
55     return true;
56 }
57
58 }
59
60 template<typename Numeric>

```

```
62 bool isNumber(const string& s) //Devuelve 1 si es true y 0 si es false
63 {
64     Numeric n;
65     return((istringstream(s) >> n >> ws).eof());
66 }
67
68 string merkle_hash(Array<string>& hashes, size_t n)
69 {
70     return _merkle_hash(hashes, n)[0];
71 }
72 Array<string> _merkle_hash(Array<string>& hashes, size_t n)
73 {
74     if(n == 1)
75         return hashes;
76
77     if(n%2 == 1)
78     {
79         if (hashes.getSize() <= n)
80         {
81             hashes.ArrayRedim(n+1);
82         }
83         hashes[n] = hashes[n-1];
84         n++;
85     }
86     Array<string> result(n/2);
87     size_t j = 0;
88     for (size_t i = 0; i < n/2; i++, j+=2)
89     {
90         result[i] = sha256(sha256(hashes[j] + hashes[j+1]));
91     }
92     return _merkle_hash(result, n/2);
93 }
94 #endif //TOOLS_H
```

5.1 Códigos provistos por la cátedra

Listing 13: cmdline.h

```
1  #ifndef _CMDLINE_H_INCLUDED_
2  #define _CMDLINE_H_INCLUDED_
3
4  #include <string>
5  #include <iostream>
6
7  #define OPT_DEFAULT    0
8  #define OPT_SEEN      1
9  #define OPT_MANDATORY 2
10
11 struct option_t {
12     int has_arg;
13     const char *short_name;
14     const char *long_name;
15     const char *def_value;
16     void (*parse)(std::string const &); // Puntero a función de opciones
17     int flags;
18 };
19
20 class cmdline {
21     // Este atributo apunta a la tabla que describe todas
22     // las opciones a procesar. Por el momento, sólo puede
23     // ser modificado mediante constructor, y debe finalizar
24     // con un elemento nulo.
25     //
26     option_t *option_table;
27
28     // El constructor por defecto cmdline::cmdline(), es
29     // privado, para evitar construir "parsers" (analizador
30     // sintáctico, recibe una palabra y lo interpreta en
31     // una acción dependiendo su significado para el programa)
32     // sin opciones. Es decir, objetos de esta clase sin opciones.
33     //
34
35     cmdline();
36     int do_long_opt(const char *, const char *);
37     int do_short_opt(const char *, const char *);
38 public:
39     cmdline(option_t *);
40     void parse(int, char * const []);
41 };
42
43 #endif
```


Listing 14: cmdline.cc

```

1 // cmdline - procesamiento de opciones en la línea de comando.
2 //
3 // $Date: 2012/09/14 13:08:33 $
4 //
5 #include <string>
6 #include <cstdlib>
7 #include <iostream>
8 #include "cmdline.h"
9
10 using namespace std;
11
12 cmdline::cmdline()
13 {
14 }
15
16 cmdline::cmdline(option_t *table) : option_table(table)
17 {
18     /*
19      - Lo mismo que hacer:
20
21     option_table = table;
22
23     Siendo "option_table" un atributo de la clase cmdline
24     y table un puntero a objeto o struct de "option_t".
25
26     Se estaría contruyendo una instancia de la clase cmdline
27     cargandole los datos que se hayan en table (la table con
28     las opciones, ver el código en main.cc)
29
30     */
31 }
32
33 void
34 cmdline::parse(int argc, char * const argv[])
35 {
36     #define END_OF_OPTIONS(p) \
37         ((p)->short_name == 0 \
38          && (p)->long_name == 0 \
39          && (p)->parse == 0)
40
41     // Primer pasada por la secuencia de opciones: marcamos
42     // todas las opciones, como no procesadas. Ver código de
43     // abajo.
44     //
45     for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
46         op->flags &= ~OPT_SEEN;
47
48     // Recorremos el arreglo argv. En cada paso, vemos
49     // si se trata de una opción corta, o larga. Luego,
50     // llamamos a la función de parseo correspondiente.
51     //
52     for (int i = 1; i < argc; ++i) {
53         // Todos los parámetros de este programa deben
54         // pasarse en forma de opciones. Encontrar un
55         // parámetro no-opción es un error.
56         //
57         if (argv[i][0] != '-') {
58             cerr << "Invalid non-option argument: "
59                  << argv[i]
60                  << endl;
61             exit(1);
62         }
63
64         // Usamos "--" para marcar el fin de las
65         // opciones; todo los argumentos que puedan
66         // estar a continuación no son interpretados
67         // como opciones.
68         //

```

```

70     if (argv[i][1] == '-')
71         && argv[i][2] == 0)
72         break;
73
74     // Finalmente, vemos si se trata o no de una
75     // opción larga; y llamamos al método que se
76     // encarga de cada caso.
77     //
78     if (argv[i][1] == '-')
79         i += do_long_opt(&argv[i][2], argv[i + 1]);
80     else
81         i += do_short_opt(&argv[i][1], argv[i + 1]);
82 }
83
84 // Segunda pasada: procesamos aquellas opciones que,
85 // (1) no hayan figurado explícitamente en la línea
86 // de comandos, y (2) tengan valor por defecto.
87 //
88 for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
89 #define OPTION_NAME(op) \
90     (op->short_name ? op->short_name : op->long_name)
91     if (op->flags & OPT_SEEN)
92         continue;
93     if (op->flags & OPT_MANDATORY) {
94         cerr << "Option "
95             << "-"
96             << OPTION_NAME(op)
97             << " is mandatory."
98             << "\n";
99         exit(1);
100     }
101     if (op->def_value == 0)
102         continue;
103     op->parse(string(op->def_value));
104 }
105
106 int
107 cmdline::do_long_opt(const char *opt, const char *arg)
108 {
109     // Recorremos la tabla de opciones, y buscamos la
110     // entrada larga que se corresponda con la opción de
111     // línea de comandos. De no encontrarse, indicamos el
112     // error.
113     //
114     for (option_t *op = option_table; op->long_name != 0; ++op) {
115         if (string(opt) == string(op->long_name)) {
116             // Marcamos esta opción como usada en
117             // forma explícita, para evitar tener
118             // que inicializarla con el valor por
119             // defecto.
120             //
121             op->flags |= OPT_SEEN;
122
123             if (op->has_arg) {
124                 // Como se trata de una opción
125                 // con argumento, verificamos que
126                 // el mismo haya sido provisto.
127                 //
128                 if (arg == 0) {
129                     cerr << "Option requires argument: "
130                         << "-"
131                         << opt
132                         << "\n";
133                     exit(1);
134                 }
135                 op->parse(string(arg));
136                 return 1;
137             } else {
138                 // Opción sin argumento.

```

```

140         //
141         op->parse(string(""));
142         return 0;
143     }
144 }
145
146 // Error: opción no reconocida. Imprimimos un mensaje
147 // de error, y finalizamos la ejecución del programa.
148 //
149 cerr << "Unknown option: "
150        << "--"
151        << opt
152        << ". "
153        << endl;
154 exit(1);
155
156 // Algunos compiladores se quejan con funciones que
157 // lógicamente no pueden terminar, y que no devuelven
158 // un valor en esta última parte.
159 //
160 return -1;
161 }
162
163 int
164 cmdline::do_short_opt(const char *opt, const char *arg)
165 {
166     option_t *op;
167
168     // Recorremos la tabla de opciones, y buscamos la
169     // entrada corta que se corresponda con la opción de
170     // línea de comandos. De no encontrarse, indicamos el
171     // error.
172     //
173     for (op = option_table; op->short_name != 0; ++op) {
174         if (string(opt) == string(op->short_name)) {
175             // Marcamos esta opción como usada en
176             // forma explícita, para evitar tener
177             // que inicializarla con el valor por
178             // defecto.
179             //
180             op->flags |= OPT_SEEN;
181
182             if (op->has_arg) {
183                 // Como se trata de una opción
184                 // con argumento, verificamos que
185                 // el mismo haya sido provisto.
186                 //
187                 if (arg == 0) {
188                     cerr << "Option requires argument: "
189                            << "--"
190                            << opt
191                            << "\n";
192                     exit(1);
193                 }
194                 op->parse(string(arg));
195                 return 1;
196             } else {
197                 // Opción sin argumento.
198                 //
199                 op->parse(string(""));
200                 return 0;
201             }
202         }
203     }
204
205     // Error: opción no reconocida. Imprimimos un mensaje
206     // de error, y finalizamos la ejecución del programa.
207     //
208     cerr << "Unknown option: "

```

```
210     << "-"
211     << opt
212     << "."
213     << endl;
214     exit(1);
215
216     // Algunos compiladores se quejan con funciones que
217     // lógicamente no pueden terminar, y que no devuelven
218     // un valor en esta última parte.
219     //
220     return -1;
221 }
```

Listing 15: sha256.h

```

1  #ifndef SHA256_H
2  #define SHA256_H
3  #include <string>
4
5  class SHA256
6  {
7  protected:
8      typedef unsigned char uint8;
9      typedef unsigned int uint32;
10     typedef unsigned long long uint64;
11
12     const static uint32 sha256_k[];
13     static const unsigned int SHA224_256_BLOCK_SIZE = (512/8);
14 public:
15     void init();
16     void update(const unsigned char *message, unsigned int len);
17     void final(unsigned char *digest);
18     static const unsigned int DIGEST_SIZE = ( 256 / 8);
19
20 protected:
21     void transform(const unsigned char *message, unsigned int block_nb);
22     unsigned int m_tot_len;
23     unsigned int m_len;
24     unsigned char m_block[2*SHA224_256_BLOCK_SIZE];
25     uint32 m_h[8];
26 };
27
28 std::string sha256(std::string input);
29
30 #define SHA2_SHFR(x, n) ((x >> n))
31 #define SHA2_ROTTR(x, n) ((x >> n) | (x << ((sizeof(x) << 3) - n)))
32 #define SHA2_ROTL(x, n) ((x << n) | (x >> ((sizeof(x) << 3) - n)))
33 #define SHA2_CH(x, y, z) ((x & y) ^ (~x & z))
34 #define SHA2_MAJ(x, y, z) ((x & y) ^ (x & z) ^ (y & z))
35 #define SHA256_F1(x) (SHA2_ROTTR(x, 2) ^ SHA2_ROTTR(x, 13) ^ SHA2_ROTTR(x, 22))
36 #define SHA256_F2(x) (SHA2_ROTTR(x, 6) ^ SHA2_ROTTR(x, 11) ^ SHA2_ROTTR(x, 25))
37 #define SHA256_F3(x) (SHA2_ROTTR(x, 7) ^ SHA2_ROTTR(x, 18) ^ SHA2_SHFR(x, 3))
38 #define SHA256_F4(x) (SHA2_ROTTR(x, 17) ^ SHA2_ROTTR(x, 19) ^ SHA2_SHFR(x, 10))
39 #define SHA2_UNPACK32(x, str) \
40 { \
41     *((str) + 3) = (uint8) ((x) >> 24); \
42     *((str) + 2) = (uint8) ((x) >> 16); \
43     *((str) + 1) = (uint8) ((x) >> 8); \
44     *((str) + 0) = (uint8) ((x) >> 0); \
45 }
46 #define SHA2_PACK32(str, x) \
47 { \
48     *((x) + 3) = ((uint32) *((str) + 3) << 24) \
49     | ((uint32) *((str) + 2) << 16) \
50     | ((uint32) *((str) + 1) << 8) \
51     | ((uint32) *((str) + 0) << 0); \
52 }
53 #endif

```

Listing 16: sha256.cpp

```

1  #include <cstring>
2  #include <fstream>
3  #include "sha256.h"
4
5  const unsigned int SHA256::sha256_k[64] = //UL = uint32
6      {0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
7        0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
8        0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
9        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
10       0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
11       0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
12       0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
13       0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
14       0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
15       0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
16       0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
17       0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
18       0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
19       0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
20       0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
21       0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};
22
23 void SHA256::transform(const unsigned char *message, unsigned int block_nb)
24 {
25     uint32 w[64];
26     uint32 wv[8];
27     uint32 t1, t2;
28     const unsigned char *sub_block;
29     int i;
30     int j;
31     for (i = 0; i < (int) block_nb; i++) {
32         sub_block = message + (i << 6);
33         for (j = 0; j < 16; j++) {
34             SHA2_PACK32(&sub_block[j << 2], &w[j]);
35         }
36         for (j = 16; j < 64; j++) {
37             w[j] = SHA256_F4(w[j - 2]) + w[j - 7] + SHA256_F3(w[j - 15]) + w[
j - 16];
38         }
39         for (j = 0; j < 8; j++) {
40             wv[j] = m_h[j];
41         }
42         for (j = 0; j < 64; j++) {
43             t1 = wv[7] + SHA256_F2(wv[4]) + SHA2_CH(wv[4], wv[5], wv[6])
+ sha256_k[j] + w[j];
44             t2 = SHA256_F1(wv[0]) + SHA2_MAJ(wv[0], wv[1], wv[2]);
45             wv[7] = wv[6];
46             wv[6] = wv[5];
47             wv[5] = wv[4];
48             wv[4] = wv[3] + t1;
49             wv[3] = wv[2];
50             wv[2] = wv[1];
51             wv[1] = wv[0];
52             wv[0] = t1 + t2;
53         }
54         for (j = 0; j < 8; j++) {
55             m_h[j] += wv[j];
56         }
57     }
58 }
59
60 void SHA256::init()
61 {
62     m_h[0] = 0x6a09e667;
63     m_h[1] = 0xbb67ae85;
64     m_h[2] = 0x3c6ef372;
65     m_h[3] = 0xa54ff53a;
66     m_h[4] = 0x510e527f;

```

```

68     m_h[5] = 0x9b05688c;
69     m_h[6] = 0x1f83d9ab;
70     m_h[7] = 0x5be0cd19;
71     m_len = 0;
72     m_tot_len = 0;
73 }
74
75 void SHA256::update(const unsigned char *message, unsigned int len)
76 {
77     unsigned int block_nb;
78     unsigned int new_len, rem_len, tmp_len;
79     const unsigned char *shifted_message;
80     tmp_len = SHA224_256_BLOCK_SIZE - m_len;
81     rem_len = len < tmp_len ? len : tmp_len;
82     memcpy(&m_block[m_len], message, rem_len);
83     if (m_len + len < SHA224_256_BLOCK_SIZE) {
84         m_len += len;
85         return;
86     }
87     new_len = len - rem_len;
88     block_nb = new_len / SHA224_256_BLOCK_SIZE;
89     shifted_message = message + rem_len;
90     transform(m_block, 1);
91     transform(shifted_message, block_nb);
92     rem_len = new_len % SHA224_256_BLOCK_SIZE;
93     memcpy(m_block, &shifted_message[block_nb << 6], rem_len);
94     m_len = rem_len;
95     m_tot_len += (block_nb + 1) << 6;
96 }
97
98 void SHA256::final(unsigned char *digest)
99 {
100     unsigned int block_nb;
101     unsigned int pm_len;
102     unsigned int len_b;
103     int i;
104     block_nb = (1 + ((SHA224_256_BLOCK_SIZE - 9)
105                     < (m_len % SHA224_256_BLOCK_SIZE)));
106     len_b = (m_tot_len + m_len) << 3;
107     pm_len = block_nb << 6;
108     memset(m_block + m_len, 0, pm_len - m_len);
109     m_block[m_len] = 0x80;
110     SHA2_UNPACK32(len_b, m_block + pm_len - 4);
111     transform(m_block, block_nb);
112     for (i = 0; i < 8; i++) {
113         SHA2_UNPACK32(m_h[i], &digest[i << 2]);
114     }
115 }
116
117 std::string sha256(std::string input)
118 {
119     unsigned char digest[SHA256::DIGEST_SIZE];
120     memset(digest, 0, SHA256::DIGEST_SIZE);
121
122     SHA256 ctx = SHA256();
123     ctx.init();
124     ctx.update((unsigned char*)input.c_str(), input.length());
125     ctx.final(digest);
126
127     char buf[2*SHA256::DIGEST_SIZE+1];
128     buf[2*SHA256::DIGEST_SIZE] = 0;
129     for (unsigned int i = 0; i < SHA256::DIGEST_SIZE; i++)
130         sprintf(buf+i*2, "%02x", digest[i]);
131     return std::string(buf);
132 }

```

6 Enunciado

75.04/95.12 Algoritmos y Programación II

Trabajo práctico 1: algoritmos y estructuras de datos

Universidad de Buenos Aires - FIUBA
Segundo cuatrimestre de 2020

1. Objetivos

Ejercitar conceptos relacionados con estructuras de datos, diseño y análisis de algoritmos. Escribir un programa en C++ (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

2. Alcance

Este Trabajo Práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado a través del campus virtual, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe de acuerdo con lo que mencionaremos en la Sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

El propósito de este trabajo es continuar explorando los detalles técnicos de Bitcoin y blockchain, tomando como objeto de estudio nuestra versión simplificada de la blockchain introducida en el primer trabajo práctico: la ALGOCHAIN.

En esta oportunidad, extenderemos el alcance de nuestros desarrollos y operaremos con cadenas de bloques completas. Para ello, nos apoyaremos en un protocolo sencillo que permite abstraer los aspectos técnicos de la ALGOCHAIN. Al implementar este protocolo, nuestros programas podrán actuar como clientes transaccionales de la ALGOCHAIN, simplificando la operativa de cara al usuario final.

4.1. Tareas a realizar

A continuación enumeramos las tareas que deberemos llevar a cabo. Cada una de estas será debidamente detallada más adelante:

1. Implementación de una interfaz operativa basada en un protocolo artificial para interactuar con la ALGOCHAIN.
2. Lectura, interpretación y pre-procesamiento de una ALGOCHAIN completa.
3. Nuevo algoritmo de cómputo del campo `txns_hash` basado en árboles de Merkle.

4.1.1. Protocolo operacional

El protocolo con el que trabajaremos consiste en una serie de **comandos** que permiten representar distintas operaciones sobre la ALGOCHAIN. Cada comando recibe una cantidad específica de parámetros, realiza cierta acción y devuelve un resultado al usuario final.

Conceptos preliminares Antes de detallar los comandos, es importante definir algunos conceptos preliminares:

- **Bloque génesis:** al igual que en blockchain, al primer bloque de toda ALGOCHAIN se lo conoce como *bloque génesis*. Esencialmente, este bloque introduce un saldo inicial para un usuario dado. Puesto que no existen bloques anteriores, el campo `prev_block` de un bloque génesis debe indicar un hash nulo (i.e., con todos los bytes en 0). Este bloque debe también contar con un único *input* y un único *output*. De igual modo, el *input* debe referenciar un *outpoint* nulo, mientras que el *output* hace la asignación del saldo inicial respetando el formato usual.
- **Mempool:** el protocolo de este trabajo permitirá que nuestros programas operen como mineros de la ALGOCHAIN. Emulando el comportamiento de los mineros de Bitcoin, nuestros programas contarán con un espacio en memoria donde se alojarán las transacciones de los usuarios que aún no fueron confirmadas (i.e., que no se agregaron a la ALGOCHAIN). Este espacio se conoce como *mempool*.

Descripción de los comandos

- `init <user> <value> <bits>`

Descripción. Genera un bloque génesis para inicializar la ALGOCHAIN. El bloque asignará un monto inicial `value` a la dirección del usuario `user`. El bloque deberá minarse con la dificultad `bits` indicada.

Valor de retorno. El hash del bloque génesis. Observar que es posible realizar múltiples invocaciones a `init` (en tales casos, el programa debe descartar la información de la ALGOCHAIN anterior).

- `transfer <src> <dst1> <value1> ... <dstN> <valueN>`

Descripción. Genera una nueva transacción en la que el usuario *src* transferirá fondos a una cantidad *N* de usuarios (al *i*-ésimo usuario, *dst_i*, se le transferirá un monto de *value_i*). Si el usuario origen no cuenta con la cantidad de fondos disponibles solicitada, la transacción debe considerarse inválida y no llevarse a cabo.

Consideraciones adicionales. Recordar que cada *input* de una transacción toma y utiliza la cantidad completa de fondos del *outpoint* correspondiente. En caso de que una de nuestras transacciones no utilice en sus *outputs* el saldo completo recibido en los *inputs*, la implementación de este comando debe generar un *output* adicional con el *vuelto* de la operación. Este vuelto debe asignarse a la dirección del usuario origen.

Valor de retorno. Hash de la transacción en caso de éxito; FAIL en caso de falla por invalidez.

■ `mine <bits>`

Descripción. Ensambla y agrega a la ALGOCHAIN un nuevo bloque a partir de todas las transacciones en la *mempool*. La dificultad del minado viene dada por el parámetro *bits*.

Valor de retorno. Hash del bloque en caso de éxito; FAIL en caso de falla por invalidez.

■ `balance <user>`

Descripción. Consulta el saldo disponible en la dirección del usuario *user*. Notar que las transacciones en la *mempool*, al no estar todavía confirmadas, no deben contemplarse para responder esta consulta.

Valor de retorno. Saldo disponible del usuario.

■ `block <id>`

Descripción. Consulta la información del bloque representado por el hash *id*.

Valor de retorno. Los campos del bloque siguiendo el formato usual. Las transacciones sólo deben mostrarse con el respectivo hash que las identifica (es decir, omitiendo sus campos). Debe devolver FAIL en caso de recibir un hash inválido.

■ `txn <id>`

Descripción. Consulta la información de la transacción representada por el hash *id*.

Valor de retorno. Los campos de la transacción siguiendo el formato usual. Debe devolver FAIL en caso de recibir un hash inválido.

■ `load <filename>`

Descripción. Lee la ALGOCHAIN serializada en el archivo pasado por parámetro.

Valor de retorno. Hash del último bloque de la cadena en caso de éxito; FAIL en caso de falla por invalidez de algún bloque y/o transacción. Observar que es posible realizar múltiples invocaciones a *load* (en tales casos, el programa debe descartar la información de la ALGOCHAIN anterior).

■ `save <filename>`

Descripción. Guarda una copia de la ALGOCHAIN en su estado actual al archivo indicado por el parámetro `filename`. Cada bloque debe serializarse siguiendo el formato usual. Los bloques deben aparecer en orden en el archivo, comenzando desde el génesis.

Valor de retorno. OK en caso de éxito; FAIL en caso de falla.

4.1.2. Lectura de la Algochain

Tal como se infiere del comando `load`, nuestros programas deberán tener la capacidad de leer e interpretar versiones completas de la ALGOCHAIN. Esto permitirá extender e interactuar con cadenas de bloques arbitrarias, permitiendo entre otras cosas el cruce de información entre grupos distintos.

En resumen, los programas deberán poder recibir una ALGOCHAIN serializada en un archivo de entrada y leer la información bloque a bloque, posiblemente organizando los datos en estructuras convenientes para facilitar las consultas y operaciones posteriores. El formato de entrada sigue los lineamientos detallados en el enunciado del trabajo práctico anterior: una ALGOCHAIN no es otra cosa que una concatenación ordenada de bloques.

4.1.3. Árboles de Merkle y hash de transacciones

Un *árbol de Merkle* [3] es un árbol binario completo en el que los nodos almacenan hashes criptográficos. Dada una secuencia de datos L_1, \dots, L_n sobre la que se desea obtener un hash, el árbol de Merkle se define computando primero los hashes $h(L_1), \dots, h(L_n)$ y generando hojas a partir de estos valores. Cada par de hojas consecutivas es a su vez hashado concatenando los respectivos hashes, lo cual origina un nuevo nodo interno del árbol. Este proceso se repite sucesivamente nivel tras nivel, llegando eventualmente a un único hash que corresponde a la raíz del árbol. Esto se ilustra en la Figura 1.

Una particularidad interesante de un hash basado en árboles de Merkle es que resulta muy eficiente comprobar que un dato dado forma parte del conjunto de datos representado por la raíz del árbol. Esta comprobación requiere computar un número de hashes proporcional al logaritmo del número de datos iniciales (cf. el costo lineal en esquemas secuenciales como el adoptado en el primer trabajo práctico).

Siguiendo los lineamientos del protocolo de Bitcoin, en este trabajo práctico computaremos los hashes de las transacciones de un bloque a partir de un árbol de Merkle. En otras palabras, el campo `txns_hash` del header de un bloque b arbitrario deberá contener el hash SHA256 correspondiente a la raíz del árbol del Merkle construido a partir de la secuencia de transacciones de b .

En caso de que la cantidad de transacciones no pueda agruparse de a pares, la última transacción debe agruparse consigo misma para generar los hashes del nivel superior del árbol. Esta estrategia debe repetirse en cada nivel sucesivo.

Ejemplo de cómputo Supongamos que queremos calcular el árbol de Merkle para una secuencia de tres cadenas de caracteres: $s_1 = \text{árbol}$, $s_2 = \text{de}$, $s_3 = \text{Merkle}$. El cómputo debería seguir los siguientes pasos:

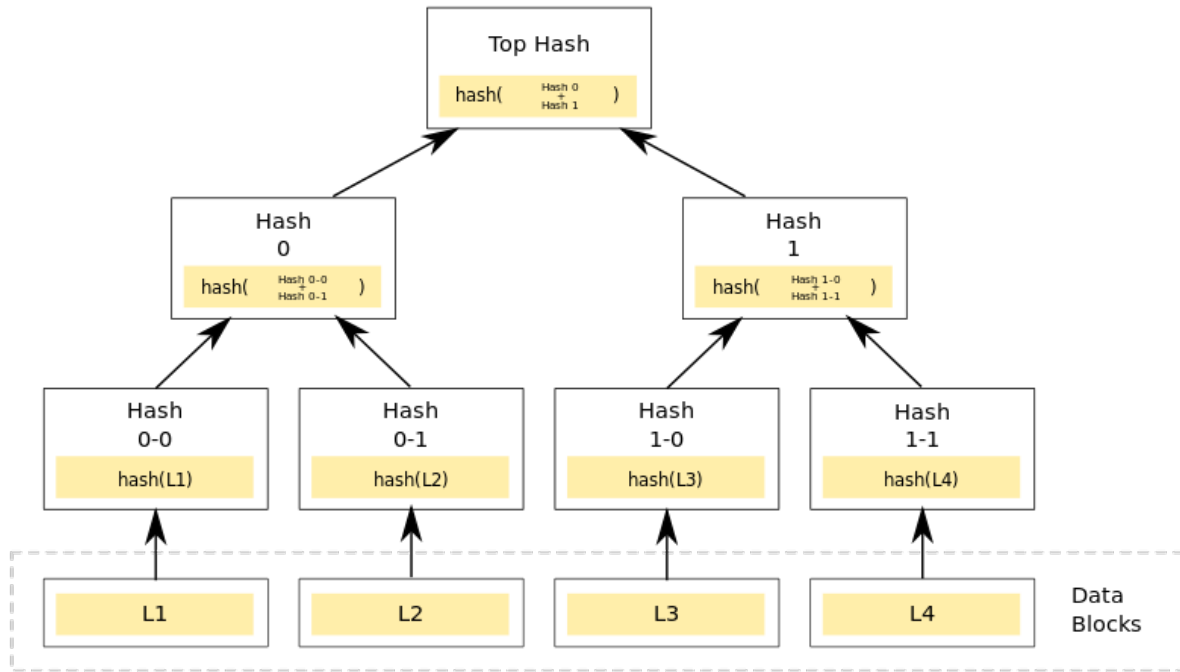


Figura 1: Esquema de un árbol de Merkle (cortesía Wikipedia). El operador +, en este contexto, indica concatenación de hashes y no suma numérica.

1. Para cada s_i , se calcula h_i , un doble hash SHA256 de dicha cadena.
2. Se agrupa h_1 con h_2 y h_3 consigo mismo. Esto da lugar a un nuevo nivel en el árbol formado por dos nuevas cadenas $s_{1,2} = h_1 + h_2$ y $s_{3,3} = h_3 + h_3$ con las respectivas concatenaciones de sus hashes.
3. Se vuelve a repetir el proceso anterior, en esta oportunidad a partir de los hashes $h_{1,2}$ y $h_{3,3}$ de $s_{1,2}$ y $s_{3,3}$, respectivamente.
4. De lo anterior surge un nuevo nivel del árbol con un único nodo, H . Este nodo es la raíz del árbol de Merkle para s_1, s_2 y s_3 .

Los hashes anteriores son los siguientes:

- $h_1 = \text{a225a1d1a31ea0d7eca83bcfe582f915539f926526634a4a8e234a072b2cec23}$
- $h_2 = \text{b2d04d58d202b5a4a7b74bc06dc86d663127518cfe9888ca0bb0e1a5d51e6f19}$
- $h_3 = \text{b96c4732b691beb72b3a8f28c59897bd58f618dbac1c3b0119bcea85ada0212f}$
- $h_{1,2} = \text{798f857ba2cdd63f03e22aa5aa52340f10da8fc8b5183dfe989ad366327d36fc}$
- $h_{3,3} = \text{af2b866e8ef21130a6ca55776f256a002215e72e99a711978534772af767fbf8}$
- $H = \text{abe24c1aeaf6f7358e1702009026c8ad146aa5321e91d36e1928bfc8e6e48896}$

4.1.4. Consideraciones adicionales

- Los detalles técnicos de la blockchain y el formato de transacciones y bloques de la ALGOCHAIN fueron deliberadamente omitidos en este enunciado. Sugerimos remitirse al enunciado del primer trabajo práctico para revisar preventivamente todos estos conceptos.
- El cálculo de hashes SHA256 puede realizarse mediante la misma librería provista por la cátedra en la instancia anterior.
- Es importante remarcar que toda estructura de datos (e.g., listas, arreglos dinámicos, pilas o árboles) **debe ser implementada**. La única excepción permitida son las tablas de hash. En caso de necesitar utilizarlas, sugerimos revisar la clase `std::unordered_map` de la STL de C++ [4].

4.2. Interfaz de línea de comandos

Al igual que en el primer trabajo práctico, la interacción con nuestros programas se dará a través de la línea de comandos. Las opciones a implementar en este caso son las siguientes:

- `-i`, o `--input`, que permite controlar el stream de entrada de los comandos del protocolo detallado en la Sección 4.1.1. Si este argumento es `"-"`, el programa deberá recibir los comandos por la entrada standard, `std::cin`. En otro caso, el argumento indicará el archivo de entrada conteniendo dichos comandos. Puede asumirse que cada comando aparece en una única línea dedicada.
- `-o`, o `--output`, que permite direccionar las respuestas del procesamiento de los comandos a un stream de salida. Si este argumento es `"-"`, el programa deberá mostrar las respuestas de los comandos por la salida standard, `std::cout`. En otro caso, el argumento indicará el archivo de salida donde deberán guardarse estas respuestas.

4.3. Ejemplos

En lo que sigue mostraremos algunos ejemplos que ilustran el comportamiento básico del programa ante algunas entradas simples. Tener en cuenta las siguientes consideraciones:

- Los hashes mostrados podrían no coincidir con los computados por otras implementaciones, puesto que dependen entre otras cosas de la elección de los nonces al momento de minar los bloques.
- Al igual que en los ejemplos del trabajo práctico anterior, por conveniencia resumiremos algunos hashes con sus últimos 8 bytes. Las entradas y salidas de nuestros programas deben, naturalmente, trabajar con los hashes completos.

4.3.1. Ejemplo trivial: entrada vacía

Si no hay comandos para procesar (i.e., el stream de entrada es vacío), el programa no debe realizar ninguna acción:

\$

4.3.2. Múltiples inits

azul y con un símbolo \triangleright al comienzo.

al usuario lucas una unidad de dinero:

\$

Se observa lo siguiente:

- Al pedir el bloque con hash fb08a246, vemos que el programa informa una falla. Esta

falla proviene de un hash de bloque inválido en la cadena actual: notar que dicho hash corresponde al bloque génesis de la primera cadena.

- El último comando solicita la información del bloque cuyo hash es 08b52667. En este caso, dicho hash coincide con el del nuevo bloque génesis, por lo que la operación es ahora exitosa.

Podemos también realizar una operatoria en modo *batch* copiando todos estos comandos en un archivo e invocando luego al programa con este archivo como entrada:

```
$ cat commands.txt
init satoshi 100 10
balance satoshi
balance lucas
init lucas 1 10
balance satoshi
balance lucas
block fb08a246
block 08b52667
$ ./tp1 -i commands.txt -o output.txt
$ cat output.txt
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
100.0
0
b40495bf172be3c172a41a85f72d13e8b2e8e7e582fc7e14b05e614408b52667
0
1.0
FAIL
0000000000000000000000000000000000000000000000000000000000000000
647bbe505403dca7a11d08269d02017c72eb0fc2e4398befe41cea620570e639
10
2535
1
1
00000000 0 00000000
1
1.0 f82e82dac113d37a21e2b3e0c37eab9e6fbc3657a38b0a8397d913abedab7605

$
```

Prestar especial atención a la última línea vacía de la salida (sugerimos remitirse al formato de transacciones y bloques detallado en el enunciado del primer trabajo práctico en caso de dudas).

4.3.3. Transferencias

El próximo ejemplo utiliza el comando `transfer` para generar transacciones y mover dinero entre distintos usuarios:


```

$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> transfer satoshi lucio 90
4ab0d8a4fdab846e9f28c1850fe06a73b446341ba7eab2cab8eae9948597e1e1
> transfer satoshi lucas 1
0a7e61b9b17c7e7e21aef8d5e65e3b036e949c7f398bd0692b5b704cf04e9b84
> balance lucio
0
> mine 10
5d4075e53f5cb51da5fffb3e68eef18046fc8c1327c4c4f787550b2e94e013806
> balance satoshi
9.0
> balance lucio
90.0
> balance lucas
1.0
> transaction f04e9b84
1
8597e1e1 1 ea55eb5c
2
1.0 f82e82dac113d37a21e2b3e0c37eab9e6fbc3657a38b0a8397d913abedab7605
9.0 5fe3f3a6faaef93165aff8d88e701f965b8b956ea77e3116c8c8b2cfea55eb5c

$

```

Es importante destacar lo siguiente:

- La primera invocación de transfer consume el UTXO del usuario satoshi en el bloque génesis. La transacción generada deposita 90 unidades de dinero en la dirección del usuario lucio y un vuelto de 10 unidades de dinero en la dirección de satoshi. El hash de esta transacción es 8597e1e1.
- La segunda invocación de transfer debe, necesariamente, consumir el UTXO de satoshi correspondiente a la transacción anterior (observar que el primer *output* en el bloque génesis ya fue consumido y no puede volver a utilizarse). Esta vez, se generará una nueva transacción que deposita una unidad de dinero en la dirección de lucas y un vuelto de 9 unidades de dinero en la dirección de satoshi.
- Puesto que el saldo debe calcularse a partir de las transacciones ya confirmadas en bloques, la primera invocación de balance nos dice que lucio no tiene saldo disponible.
- No obstante esto, luego de minar el nuevo bloque a partir de las transacciones anteriores, vemos que esta vez el saldo de lucio es 90. Por otro lado, satoshi tiene un saldo de 9 unidades de dinero, mientras que lucas sólo dispone de una unidad de dinero.
- Por último, el comando transaction solicita información sobre la transacción con hash f04e9b84. Vemos que este hash corresponde a la transacción derivada del segundo uso de transfer. Allí puede verse el vuelto de 9 unidades de dinero a la dirección de satoshi (el segundo *output* de dicha transacción).

4.3.4. Lectura y escritura de cadenas

Finalmente, veamos cómo leer y escribir cadenas completas con nuestros programas. El siguiente ejemplo guarda una cadena de dos bloques al archivo `algochain.txt`:

```
$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> transfer satoshi lucas 1 lucio 90
8b58f15e5c4408b30322daca6d14edd44ff3d067d8b1ea967dff89d5705f5ff3
> mine 10
3b44b8c5182097fa63c2e84aa27735f8cad40971c84266fb874d0bd993c15315
> save algochain.txt
OK
$
```

Notar que, esta vez, el comando `transfer` incluye múltiples destinatarios: la transacción depositará una unidad de dinero en la dirección de `lucas` y 90 unidades de dinero en la de `lucio` (esto se lleva a cabo definiendo dos *outputs* diferentes). Puesto que el saldo de `satoshi` consumido por la transacción es de 100 unidades de dinero, el vuelto que le corresponde es de 9 unidades.

En esta invocación posterior, cargamos la cadena anterior a partir del archivo generado. Observar que la información de la cadena inicial (la generada vía `init`) se descarta:

```
$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> load algochain.txt
3b44b8c5182097fa63c2e84aa27735f8cad40971c84266fb874d0bd993c15315
> balance satoshi
9.0
> balance lucio
90.0
> balance lucas
1.0
$
```

4.4. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

5. Informe

El informe deberá incluir, como mínimo:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante a los algoritmos y estructuras de datos involucrados en la solución del trabajo.
- El análisis de las complejidades solicitado en la sección 4.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++.
- Este enunciado.

6. Fechas

La última fecha de entrega es el **jueves 3 de diciembre de 2020**.

Referencias

- [1] Wikipedia, "Bitcoin Wiki." https://en.bitcoin.it/wiki/Main_Page.
- [2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.
- [3] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, pp. 369–378, Springer, 1987.
- [4] cplusplus.com, "Unordered Map." https://www.cplusplus.com/reference/unordered_map/unordered_map/.