Trabajo Práctico

TD3: Algoritmos y Estructuras de Datos

Introducción

Se cuenta con un prototipo de implementación de una billetera virtual que trabaja sobre una precaria blockchain[1]. El prototipo posee la funcionalidad completa pero sufre de severos problemas de performance.

En vistas a una salida a producción, se han determinado los requisitos de complejidad que debe cumplir la aplicación para proveer una experiencia de usuario de primer nivel, y se ha confiado en los talentosísimos estudiantes de TD3 la desafiante tarea de adaptar la implementación actual para cumplir dichos requisitos.

Descripción del sistema

A modo de introducción, se brinda continuación un breve resumen del funcionamiento deseado y el diseño actual del sistema. **Esto complementa los comentarios en archivos .h del código base, que contienen la documentación del comportamiento de todas las clases y funciones junto con sus complejidades asociadas.**

Funcionalidad

En este sistema, se trabaja con una blockchain (instancia de la clase `Blockchain`) que tiene registradas distintas billeteras. La funcionalidad de la blockchain consiste en:

- almacenar las transacciones
- abrir billeteras nuevas
- realizar transacciones entre billeteras registradas

El proceso de creación de una billetera se realiza a través del método `blockchain.abrir_billetera()`, que se encarga de:

1. Crea la billetera, agregándola al registro interno
2. Realiza una transacción "semilla" por 100 unidades de moneda.
3. Retornar un puntero a la billetera creada para su posterior uso.

Esta clase provee una interfaz muy rudimentaria, no pensada para ser consumida directamente por usuarios finales. La clase `Billetera` es la encargada de proveer funcionalidad avanzada que será utilizada luego para programar la aplicación mobile para clientes:

- obtener el saldo actual
- obtener las últimas transacciones realizadas
- obtener el saldo histórico al final de cualquier día
- obtener un listado de las billeteras a las que más frecuentemente se les envía dinero

[1] El uso de la palabra "blockchain" en este trabajo es extremadamente liberal, a modo de setear un dominio de problema de interés. La implementación de la cadena de transacciones provista no asemeja en lo más mínimo a cómo trabaja una blockchain en la realidad.

Organización del código

Se recomienda comenzar por inspeccionar los archivos .h de las clases `Blockchain` y `Billetera`. Cada uno de estos tiene un archivo .cpp asociado con la implementación de su interfaz.

Adicionalmente, se provee la clase calendario (descrita en la sección siguiente) y el archivo `lib.h` donde se encuentran algunas definiciones utilizadas en todo el programa:

- `struct Transaccion`, con la información que compone una transacción
- definicion de varios alias de tipos (usando `typedef`) para mejorar la legibilidad

Por último, dentro del directorio `tests` se encuentran la suite de tests asociados a la clase `Blockchain` y `Billetera`. Se recomienda fuertemente leer los tests para ver ejemplos del uso de estas interfaces y su comportamiento esperado.

Trabajando con timestamps

Con *timestamp* nos referimos a la representación de un instante de tiempo, utilizado para marcar el momento en que ocurrió un evento (en este caso, el momento en que se realizó una transacción). En este sistema, se representan como números utilizando el sistema de **POSIX time**[2].

La clase `Calendario` debería permitir trabajar con dichos valores abstrayéndose de los detalles de implementación, proveyendo la siguiente funcionalidad:

- `timestamp Calendario::tiempo_actual()`
  - provee un valor que representa el tiempo actual[3].
- `timestamp Calendario::principio_del_dia(timestamp t)`
  - devuelve un timestamp correspondiente a la medianoche anterior t.
- `timestamp Calendario::fin_del_dia(timestamp t)`
  - devuelve un timestamp correspondiente a la medianoche siguiente a t.
- `timestamp Calendario::dia_siguiente(timestamp t)`
  - devuelve un timestamp correspondiente a la misma hora de `t`, pero del día siguiente

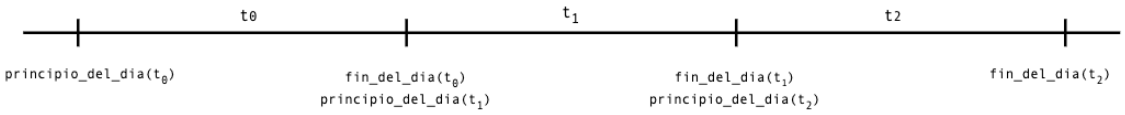

Figura 1: Representación gráfica de timestamps y sus operaciones

Tareas a cumplir

Se deben cumplir las siguientes tareas:

1. Modificar la estructura de representación y algoritmos de la clase `Billetera` para cumplir los requisitos de complejidad descritos a continuación.
2. Justificar que el código provisto cumple con las complejidades solicitadas en el **peor caso**.
3. Escribir el invariante de representación de la clase Billetera

[2]La convención es que estos números se interpretan como la cantidad de segundos que ocurrieron desde las 00:00hs del 1 de enero de 1070 UTC.

[3]Esta funcionalidad es utilizada por la clase `Blockchain` para marcar el timestamp de cada transacción nueva. No es necesario utilizarla en `Billetera`.

Requerimientos de complejidad

Para justificar la complejidad, se pide anotar la complejidad de cada línea, y luego un comentario al final justificando con álgebra de órdenes la complejdiad final. No es necesario anotar la cantidad exacta de operaciones de cada línea.

Se pide respetar las siguientes cotas de complejidad en el peor caso:

`notificar_transaccion`	$O(D \log(D) + C)$
`saldo`	$O(1)$
`ultimas_transacciones`	$O(k)$, donde k es la cantidad de operaciones a listar
`saldo_al_fin_del_dia`	$O(log(D))$
`destinatarios_mas_frecuentes`	$O(k)$, donde k es la cantidad de destinatarios a listar

Donde:

T	cantidad total de transacciones en la blockchain
D	máxima cantidad de días que una billetera estuvo activa
C	máxima cantidad de destinatarios totales a los que una billetera envió dinero

Escribiendo el invariante de representación

Se pide escribir el invariante de representación de la clase *Billetera* **tanto en castellano como en lógica de primer orden** (predicado *Rep(e : Billetera)*).

A tener en cuenta:

- La versión en lenguaje natural debe entregarse en un comentario claramente identificado en el archivo `billetera.h`.
- Se deben capturar las condiciones que deben cumplir todos los campos que agreguen a la clase Billetera.
- Se pueden ignorar los campos ya definidos `_id` y `_blockchain`.
- Se puede asumir como definida la igualdad para el tipo *Transaccion*, que será verdadera si todos los campos de dos transacciones son iguales.

También cuentan con los siguientes predicados y funciones queno es necesario definir:

- *transaccionesBlockchain*(*billetera* : *Billetera*) : `list<Transaccion>`
  - denota el listado de **todas** las transacciones en la blockchain asociada a la billetera (incluyendo aquellas en las que la billetera no haya participado y transacciones semilla).
- *EsPrincipioDeDia*(t : `int`)
  - será verdadero si y sólo si t es el timestamp asociado a el principio de algún día.
- *Principio*(t : `int`) : `int`
  - denota el timestamp asociado al principio del día de t (es decir, la medianoche previa a t para valores que no son ya medianoche).
- *Fin*(t : `int`) : `int`
  - denota el timestamp asociado el fin del día de t (es decir, la medianoche siguiente a t para valores que no son ya medianoche).

Trabajando sobre el código

- El código provisto contiene la configuración necesaria para cargar el proyecto en VSCode dentro de un container.
- Todas las modificaciones al código deben limitarse a los archivos `billetera.h` y `billetera.cpp`, modificando sólo la parte privada de la clase (NO su interfaz).
- Se debe respetar el comportamiento documentado.
- Este proyecto no tiene un ejecutable interactivo. Deben enfocarse en que los tests sigan pasando a medida que cambian la implementación.
- Se evaluarán tanto la corrección técnica de la solución como la claridad del código escrito.
- Se recomienda escribir tests adicionales si desean probar otros casos bordes no cubiertos o hacer pruebas y utilizar el debugger para inspeccionar el comportamiento del código si es necesario.

Condiciones de entrega

- La fecha límite de entrega es el **lunes 23 de junio a fin del día**.
- El TP debe realizarse en grupos de 3 personas. Los grupos se deben registrar en el formulario provisto por el cuerpo docente junto con este enunciado.
- Se deben entregar vía campus tres archivos:
  - `billetera.h` y `billetera.cpp` (siguiendo las indicaciones previas)
  - Una fotografía **clara** con la especificación formal del Rep (en caso de entregarse varias páginas, los archivos deben estar debidamente numerados).