Algoritmos y Estructuras de Datos - DC - UBA

Primer cuatrimestre de 2025

Trabajo práctico 2: Diseño e implementación de estructuras

Normativa

Revisión de mitad de TP: viernes 06 de Junio, en el horario de la clase.
Límite de entrega: domingo 15 de Junio a las 23:59.
Defensa del TP: viernes 27 de Junio, en el horario de la clase.
Límite de re-entrega: domingo 13 de Julio a las 23:59.
Forma de entrega: Subir el archivo `Berretacoin.java` junto con *todos* los archivos de otras clases que hagan dentro de un único ZIP al campus.

Enunciado

En un oscuro rincón del mundo cripto, un consorcio conocido como "Los Berreteros" ha estado trabajando en secreto para desarrollar su propia criptomoneda con fines poco éticos. Este grupo, formado por estafadores expertos en tecnología, contrató a varios programadores bajo la falsa promesa de estar creando una "revolución financiera descentralizada".

En el trabajo práctico anterior, especificamos el tipo de datos abstracto **$Berretacoin**, una criptomoneda libre de controles criptográficos que, sin saberlo, estaba siendo diseñada para los Berreteros. Ahora, en este segundo trabajo práctico, revelaremos parte de sus intenciones ocultas mientras implementamos este TAD utilizando estructuras de datos apropiadas.

Entre los requisitos específicos de los Berreteros, destaca la función `hackearTx`, un método malicioso diseñado para extraer el valor máximo (MEV) de las transacciones, permitiéndoles manipular la cadena de bloques y realizar estafas sistemáticas a los desprevenidos tenedores de la moneda. Este método es parte integral de su plan para realizar esquemas *pump-and-dump* donde artificialmente elevan el valor de la moneda para luego extraer las transacciones más valiosas, dejando a los inversores con pérdidas significativas.

Aclaraciones

- Los usuarios se identifican con números enteros positivos consecutivos.
- A diferencia de la especificación anterior, esta vez la cantidad de transacciones por bloque **no está acotada**.
- Al hackear una transacción, se debe restaurar el monto de la transacción al comprador y al vendedor.

Operaciones a implementar

Las operaciones se deben implementar respetando las complejidades temporales indicadas. Usaremos las siguientes variables para expresar las complejidades:

- P: cantidad total de usuarios
- n_b: cantidad de transacciones en el bloque

1. `nuevoBerretacoin(in n_usuarios: ℤ): $Berretacoin` $O(P)$
   Inicializa al sistema de criptomonedas con usuarios numerados de 1 a n_usuarios.
2. `agregarBloque(inout berretacoin: $Berretacoin, in transacciones: seq<Transaccion>)` $O(n_b * \log P)$
   Agrega un nuevo bloque con la secuencia de transacciones, que vienen ordenadas por su identificador, a la cadena de bloques.
3. `txMayorValorUltimoBloque(in berretacoin: $Berretacoin)`: Transaccion $O(1)$
   Devuelve la transacción de mayor valor del último bloque (sin extraerla). En caso de empate, devuelve aquella de mayor id.
4. `txUltimoBloque(in berretacoin: $Berretacoin)`: `seq<Transaccion>` $O(n_b)$
   Devuelve una copia de la secuencia de transacciones del último bloque, ordenadas por su identificador.

5. `maximoTenedor(in berretacoin: $Berretacoin)`: $\mathbb{Z}$ $O(1)$

   Devuelve al usuario que posee la mayor cantidad de $Berretacoin. En caso de empate, devuelve aquél de menor id.

6. `montoMedioUltimoBloque(in berretacoin: $Berretacoin)`: $\mathbb{Z}$ $O(1)$

   Devuelve el monto promedio de todas las transacciones en el último bloque de la cadena, sin considerar las "transacciones de creación". En caso de que no haya transacciones, devuelve 0.

7. `hackearTx(inout berretacoin: $Berretacoin)`: $O(\log n_b + \log P)$

   Extrae del último bloque de la cadena la transacción de mayor monto. No importa si después de la extracción queda una transacción dentro del bloque donde el comprador no tiene fondos suficientes.

La entrega debe incluir *al menos* una clase `Java` que implemente la solución al problema. Recomendamos (y valoraremos) modularizar la solución en varias clases.

IMPORTANTE: No se manden a programar sin pensar antes una solución al problema. La idea es que piensen "en papel" las estructuras de datos que permitirán cumplir con las complejidades especificadas y las consulten con su corrector durante la clase. Una vez que su corrector les dé el OK, pueden comenzar a programar la solución.

Condiciones de entrega y aprobación

Durante la clase del día viernes 06 de Junio deberán hablar con su corrector asignado para la **revisión obligatoria** de mitad de TP. En esta, deben mostrarle la estructura de representación propuesta para resolver el problema. Al finalizar esta clase, deben salir con una estructura que funcione para cumplir las complejidades pedidas.

La entrega deberá incluir el archivo `Berretacoin.java`, el cual implementará la solución al problema. Las demás clases diseñadas para la solución se deben incluir en otros archivos `.java`. También los archivos de testeo que desarrollen. Todos estos archivos deben ser subidos al campus antes de la fecha y hora límite de entrega.

Para la aprobación del trabajo práctico, la implementación debe superar todos los tests provistos y, además, debe cumplir con las complejidades temporales especificadas en la sección anterior. Se debe dejar comentado (de forma breve y concisa) la justificación de la complejidad obtenida. Solamente pueden utilizar las estructuras vistas en la teórica hasta ahora (arreglo, arreglo redimensionable, lista enlazada, ABB, AVL, heap), implementadas por ustedes mismos. Pueden usar el código de las estructuras que hicieron para los talleres. **No se puede usar ninguna clase predefinida en la biblioteca estándar de Java, con la excepción de ArrayList, String y StringBuffer**.

Si bien les proveemos una base de tests, **esperamos que agreguen sus propios tests**.

También evaluaremos:

- la claridad del código y de las justificaciones de las complejidades.
- que se respeten los principios de abstracción y encapsulamiento.
- la modularidad del código: crear nuevas clases y funciones auxiliares cuando sea necesario.
- el testeo correcto de sus implementaciones.