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. 0, 1, 2,...
- A diferencia de la especificación anterior, esta vez la cantidad de transacciones por bloque **no está acotada**.
  (no hay un nro máximo de transacciones por bloque).

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

En rojo las que tienen un "return"

1. `nuevoBerretacoin(in n_usuarios: ℤ): $Berretacoin` $O(P)$
   - sabremos cuántos usuarios hay.
   - se "crean" monedas (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. crea bloque con transacciones (de monedas y usuarios preexistentes).

armo lista de transacciones de mayor cant. de coins y de ahí tomo la de id más grande.

3. `txMayorValorUltimoBloque(in berretacoin: $Berretacoin): ℤ` $O(1)$

   Devuelve la transacción de mayor valor del último bloque (sin extraerla). En caso de empate, devuelve aquella de mayor id. ← se refiere a la transacción "más reciente"? id de la transacción?

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.

   id de transacción?

5. `maximoTenedor(in berretacoin: $Berretacoin): ℤ` $O(1)$

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

   - habría que calcular la cantidad final de coins por vendedor
   - si hay empate tomar al vendedor de id más "reciente" (el menor número)

Podríamos armar una "lista" de aquellos con la mayor cant. de coins y de ahí agarramos al de nro de id más chico.

6. `montoMedioUltimoBloque(in berretacoin: $Berretacoin): 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.

Habría que armar una auxiliar que de la cantidad total de bloques vaya al último.

1. `nuevoBerretacoin(in n_usuarios: ℤ): $Berretacoin` $O(P)$ - sabremos cuántos usuarios hay. - se "crean" monedas (P).
Inicializa al sistema de criptomonedas con usuarios numerados de 1 a n_usuarios.

clase Berretacoin

atributos privados (observadores):
- bloque (lista de transacciones)
- transacción (struct con id, comprador, vendedor y monto)
- usuarios ("cardinal del cjto")

1) nuevoBerretacoin (nos inicializa la clase(?))
- bloque=null
- transaccion=null
- define cant. total de usuarios que harán transacciones (dice cuántos usuarios hay (P)).

#usuarios = P

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. crea bloque con transacciones (de monedas y usuarios preexistentes).

2) agregarBloque (crea nuevas listas de transacciones, sin límite específico)
parámetros de entrada: berretacoin, transacciones (secuencia de transacciones).
La función toma la secuencia de transacciones y las asigna como un nuevo bloque.
nuevo bloque = transacciones

3. `txMayorValorUltimoBloque(in berretacoin: $Berretacoin): ℤ` $O(1)$
Devuelve la transacción de mayor valor del último bloque (sin extraerla). En caso de empate, devuelve aquella de mayor id. ← se refiere a la transacción "más reciente"? id de la transacción?

3) txMayorValorUltimoBloque
valores entrada: berretacoin
returns: entero (el nro más grande de coins de alguna de las transacciones (tuplas) del último bloque.
- seleccionar el último bloque del total de bloques.
- comparar todas las transacciones (tuplas dentro del bloque) y tomar aquella cuyo monto (tupla[3]) sea el más grande de toda la secuencia de tuplas.

txMayorValorUltimoBloque([(0, 2, 1, 100), (1, 5, 1, 234), (2, 1, 4, 555), (3, 3, 1, 53)]) = 555
forma: (id, comprador, vendedor, monto)
bloque = lista de tuplas]
tupla = transacción
transacción = (id, comprador, vendedor, monto) (ejemplo!)

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 ← id de transacción?

4) txUltimoBloque (asumimos que se refiere al id de transacción)
valores entrada: seq(bloques) (de menor a mayor?)
devuelve: bloque

- tomamos el último bloque
- crea nuevo bloque con las transacciones del bloque_anterior en orden de id_transacción
- copia y pega transacciones en orden

id de transaccion?

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.

5) valor entrada: berretacoin
devuelve: entero

- armo una lista de usuarios y su cantidad final de monedas.
- tomo el de mayor monto y el de menor id en comparación a los otros de mayor monto.

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.

6) valor entrada: berretacoin
retorna: entero

- empezando de la transacción 1 (la 0 es de creación) sumamos todos los montos de todas las transacciones de 1 a n (n=cant. transacciones) y lo dividimos por la cant. de transacciones NO de creación (n-1)

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.

7) valor entrada: berretacoin

- vamos al último bloque y ELIMINAMOS la transacción con el monto más grande (como el 555 del ejemplo).