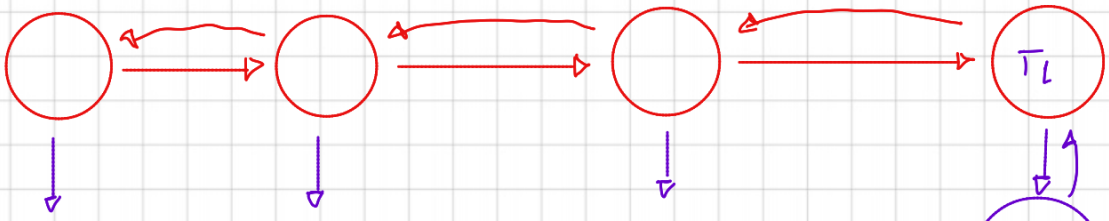
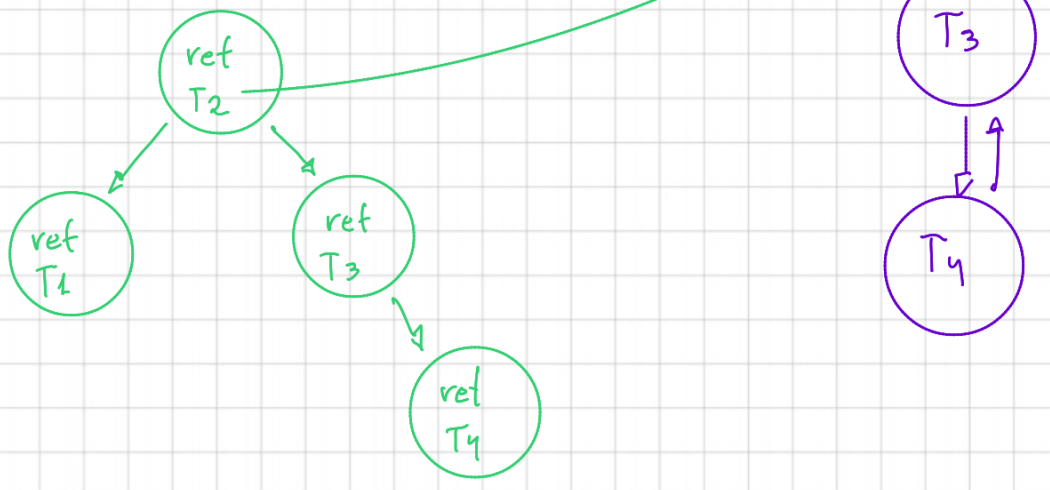
blockchain en listaEnlazada

bloque en listaEnlazada



---

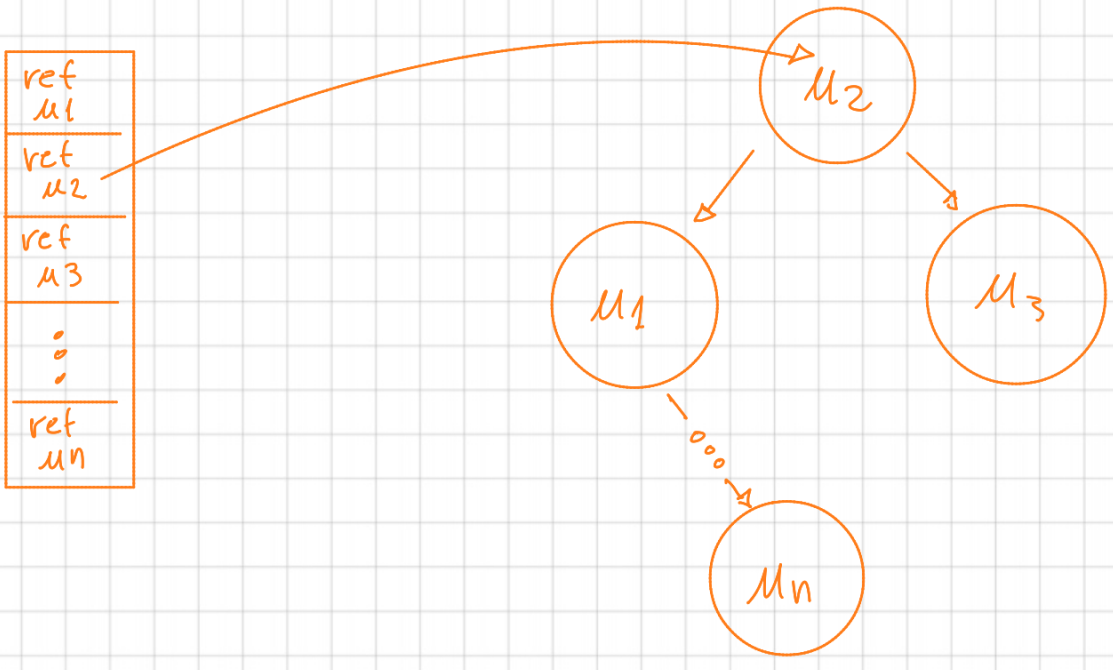
heap (ordenado por el monto de las transacciones)
compuesto por referencia a las transacciones



---

array de ref al heap de usuarios

heap ordenado por mayor tenedor



Nota: respecto de los ref, entendemos que ocupa entorno (de alguna manera.) el concepto de handle

Respecto de

1. nuevoBerretzacoin:

   se garantiza $O(n)$ pues hay que inicializar el array de ref y crear el heap de usuarios

2. agregarBloque:

   se garantiza $O(n_b \cdot \log P)$ pues se accede al final de la lista enlazada (blockchain) en $O(1)$, luego se agrega a la lista enlazada (bloque) transaccion por transaccion que es de $O(n)$ y, en

   ↳ ¿También podría ser arrayList o array fijo?

   cada "agregado" de transaccion accede en $O(1)$ al array de refs (pues el indice del array va a ser el nº de usuario, ya que la cant de usuarios es fija) que apunta al heap de Tenencia de usuario siendo que modificar un elem del heap y reacomodar es de $O(\log P)$.
   Luego heapify las transacciones del ult. bloque, siendo de $O(n_b)$
   Me queda complejidad $O(n_b \cdot \log P + n_b) = O(n_b \cdot \log P)$

3. txMayorValorUltimoBloque

   se garantiza $O(1)$ pues consulta la raiz del heap de transacciones.

4. txUltimoBloque

   se garantiza $O(n)$ pues se recorre la lista enlazada del ult. bloque para armar el array a devolver

5. maximoTenedor

   se garantiza $O(1)$ pues se consulta la raiz del heap de transacciones.

6. montoMedioUltimoBloque

   se garantiza $O(1)$ pues el valor se almacena en una variable que la voy calculando en "agregarBloque"

7. hackearTx: se garantiza $O(\log n_b + \log P)$ pues accedo, por ref de la raiz del heap Tx, al elem de la lista enlazada a eliminar ($O(1)$), reacomodo heapTx ($O(\log n_b)$), y luego, accedo a los dos usuarios involucrados usando el array de ref ($O(1)$) y reacomodo el heap usuarios ($2 \cdot \log P$).
   Me queda $O(\log n_b + 2 \cdot \log P)$ ✓