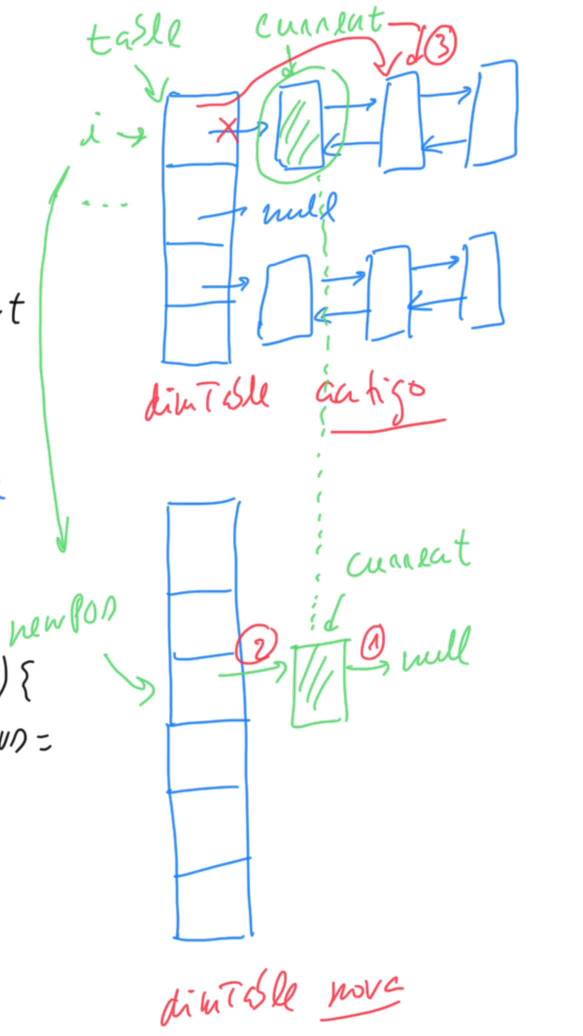
```
private fun resize() {
    // Aumentar a dimensão da tabela e
    // recalcular a posição dos elementos na nova
    // tabela
    dimTable = getLengthMersenne(dimTable) → devolve próximo nº
                                              primo de Mersenne
                                              em função de $t_k$ e $\delta_k$
                                              (ver livro)
    val newTable = arrayOfNulls<Node<Any?>>
                   (dimTable) as
                       Array<Node<E>?>

    for (i in table!!.indices) {
        var current = table!![i]
        while (current != null) {
            val newPos = index(current.
                               elem)
            table!![i] = table!![i].next
            // Inserir current à
            // cabeça da nova
            // lista, na nova tabela
 (1)        current.next = newTable
                           [newPos]
            if (newTable[newPos] != null) {
                newTable[newPos]!!.previous =
                        current
            }
 (2)        newTable[newPos] = current
 (3)        current = table!![i]
        } // while
    } // for
    table = newTable
}
```



2.

```
class HashSet<E>: MutableSet<E> {
  ...
  override fun iterator(): Iterator<E> {
        return MyIterator()
  }

  private inner class MyIterator : Iterator<E> {
      var currentPos : Int = -1            // Índice na
                                           // tabela
      var nodeIt : Node<E>? = null         // nó corrente
                                           // na lista a
      var currentElement : Node<E>?        // ser iterada
                            = null
      override fun hasNext() : Boolean {
          // se o hasNext for chamado várias
          // vezes antes de avançar o iterador
          // (chamando next()), o hasNext() deve
          // retornar sempre o mesmo resultado
          if (currentElement != null)
              return true
          while (currentPos < table!!.size) {
              if (nodeIt == null) {
                  currentPos ++
                  if (currentPos < table!!.size)
                      nodeIt = table!![currentPos]
              }
              else {
                  currentElement = nodeIt
                  nodeIt = nodeIt!!.next
                  return true
              }
          }
          return false
      }
```

next() não foi ainda invocado e, como tal, não mudou o estado

```
    override  fun next(): E {
        if (!hasNext())
              throw NoSuchElementException()
        val aux = currentElement.element
        currentElement = null
        return aux!!
    }
} // HashSet
```