Tabela de Dispersão

Fator de carga (Load Factor) = $\frac{N}{M}$ → nº elementos a inserir

↓ nº de entradas (buckets) da tabela

ex: load factor = 0.75 (75%)

Se $\frac{N}{M} > 0.75$ fazer um redimensionamento da tabela

A dimensão M pode ser escolhida como sendo um nº primo de Mérsenne ($2^t - 1$)

Implementação dum Hash set

permite guardar chaves (ou elementos onde se pode selecionar um campo como chave)

- Não podem existir chaves repetidas

```
class HashSet<E> : MutableSet<E> {
    private data class Node<E>(val elem: E,
                               var next: Node<E>?,
                               var previous: Node<E>?)

    private var table: Array<Node<E>?>? = null
    override var size: Int = 0
    private var dimTable: Int = 0

    constructor() {
        table = arrayOfNulls<Node<Any>?>(10) as
                Array<Node<E>?>?
        dimTable = 10
    }
```

Tabela de dispersão com encadeamento externo usando listas duplamente ligadas, não circulares, com sentinela

```
private fun index(e: E) : Int {
    var pos = e.hashCode() % dimTable
    if (pos < 0)
        pos += dimTable
    return pos
}
```

É o objeto completo, ex: Student, e não uma chave

é feito hash usando uma chave

hash(key, M) dos slides

Programador pode fazer override de hashCode herdado do tipo Any

encolhida, ex: número de Student

```
private fun search(e: E, idx: Int): Node<E>? {

    var curr: Node<E>? =
        table!!.get(idx)

    while (curr != null) {
        if (curr.elem!!.equals(e))
            return curr

        curr = curr.next
    }
    return curr
}
```

idx: posição na tabela onde "e" pode eventualmente ocorrer.

equals: método herdado de Any que deve ser redefinido (override)

Nota: Se uma data class, os métodos equals, hashCode já estão redefinidos.

NOTA:

Se s1.equals(s2) == true

⇒ s1.hashCode() == s2.hashCode()

Programador deve garantir que isto acontece quando redefine os métodos equals e hashCode (alternativa: uma data class)

Caso não o faça, irão ocorrer problemas na operação com a tabela de hash.

```
override fun add(elem: E): Boolean {
    var pos = index(elem)
    val node = search(elem, pos)
    if (node != null)
        return false          → chave repetida
    if ((size / dimTable).toDouble() > 0.75)
        resize()  → o dimTable vai ser incrementado (ex: usando nº primos de Mérsenne)
    pos = index(elem)
    val newNode = Node<E>(elem)
    newNode.next = table!![pos]   // liga next ao head existente
    if (table!![pos] != null)
        table!![pos]!!.previous = newNode   // liga prev do antigo head
    table!![pos] = newNode   // atualiza head
    ++size
    return true
}
```

inserir à cabeça da lista