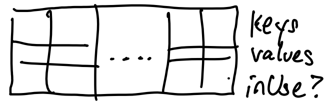
Hash Tables

- an implementation of Dictionary (unique keys)
- int hash(K key)
- hash, modulus → pick a location in an array
- collision: multiple mappings want in same slot
  - linear probing: use next empty slot



  - forward chaining

Dictionaries

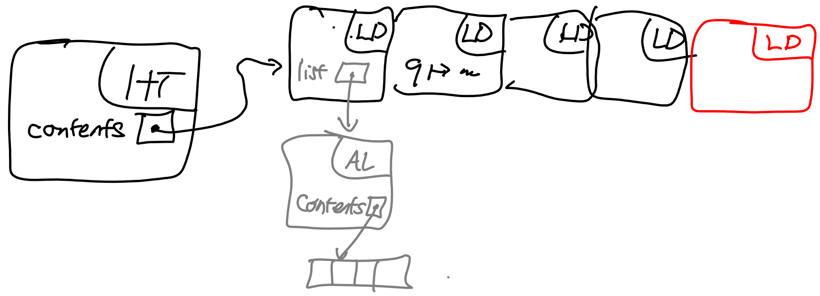
	AVL trees	Hash tables	Linear Dictionary
get insert update remove	$O(\log n)$	average $O(1)^*$	$O(n)$ using a list

```
Method get(K key):
    For i in 0 up to contents.size()-1:
        If contents[i].key == key:
            Return contents[i].value
        EndIf
    EndFor
    :(
EndMethod
```

```
Method insert(K key, V value):
    For i in 0 up to contents.size()-1:
        If contents[i].key == key:
            :(
        EndIf
    EndFor
    contents.insertLast(pair(key, value))
EndMethod
```

Forward chaining hash table:

- array of LinearDictionary

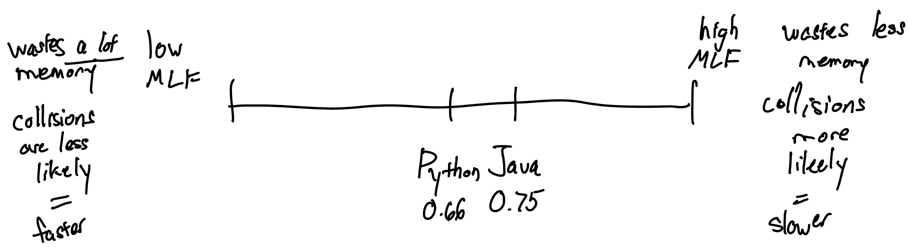


n LDs ⟶ m/n mappings per LD

- array of LDs ("buckets")
- key/value pairs insert into LDs based on key hash
- when I have "too many" k/v pairs for the LDs I have, expand capacity of HT
    - make new LD array of twice the size
    - rehash and add <u>all</u> k/v pairs to the new array.

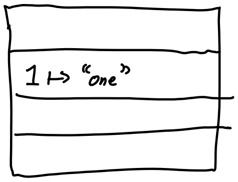
$$\frac{\#\text{ of mappings}}{\#\text{ of buckets}} = \text{load factor}$$

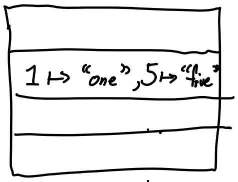
max load factor (MLF)

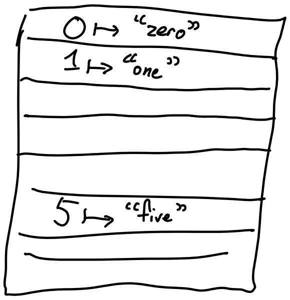


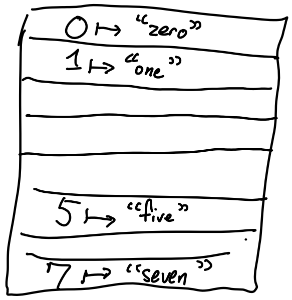
insert (1, "one")
insert (5, "five")
insert (0, "zero")
insert (7, "seven")

MLF = 0.6









average $O(1)^{*}$

* if your hash function is any good

- amortization
- averaging over all possible sets of user input