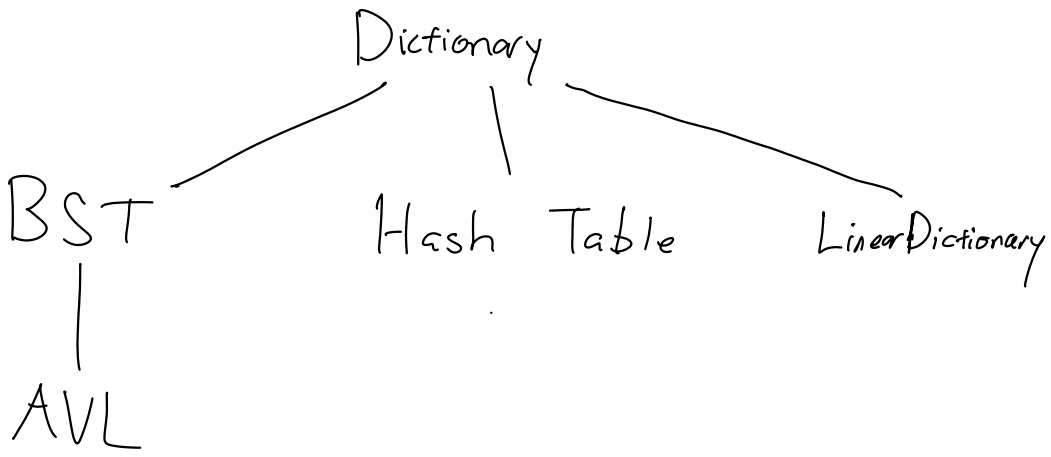
What operations do Dictionaries support?

V get(K)
void insert(K, V)
void update(K, V)
V remove(K)



Hash Table is a kind of dictionary based on an array

↳ hash function: K input, int output in a fixed range

collision: two keys have same hash

linear probing —— okay: check next slot

forward chaining —— each hash associated w/ many K,V pairs

LinearDictionary: O(n) on all operations.

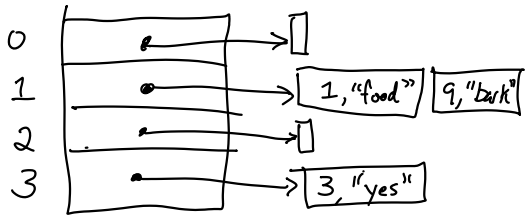
```
get(key):
    for each k,v in contents:
        if key == k:
            return v
    //
```

```
insert(key, value):
    for each k, _ in contents:
        if key == k:
            //
    contents.insertAtTail(pair(key, value))
```

Forward Chaining.

array of "buckets": each bucket has KV for one hash

Forward Chaining Hash Table



insert (1, "food")
insert (9, "bark")
insert (3, "yes")

Problem: for any constant # of buckets B and n (K, V) pairs, each bucket has about $n/B = O(n)$ items

Solution: increase B as n increases

New Problem: increasing B changes hash, so changes which bucket things should be in: need to rehash, move everything ($O(n)$)

Old Solution: (from ArrayList) Every time the list of buckets is too small, double its size

Hash Tables' terms

* capacity: # buckets
* load factor: size/capacity
* maximum load factor: after load factor is larger than MLF: grow # buckets

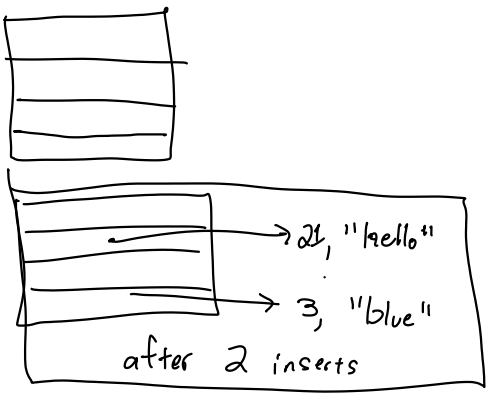
small MLF (close to 0)
uses more memory
fewer collisions

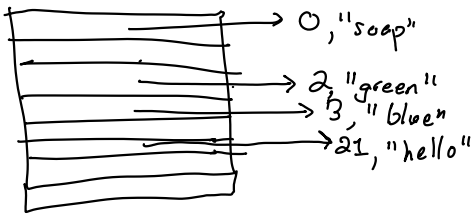
0.66 0.75
Python Java

big MLF (close to 1)
use less memory
more collisions

average $O(1)$ on all operations

IF hash function distributes keys evenly over the hash range





MLF = 0.5

insert (21, "hello")
insert (3, "blue")
insert (0, "soap")
insert (2, "green")
insert (3, "yellow")

Hash Table: How to handle collision

- Linear Probing —— MUST have MLF < 1
  - High MLF worse than w/ fe
  - Good for HDD
- Forward Chaining —— collisions do not make more collisions more complex (fine if you have random access)