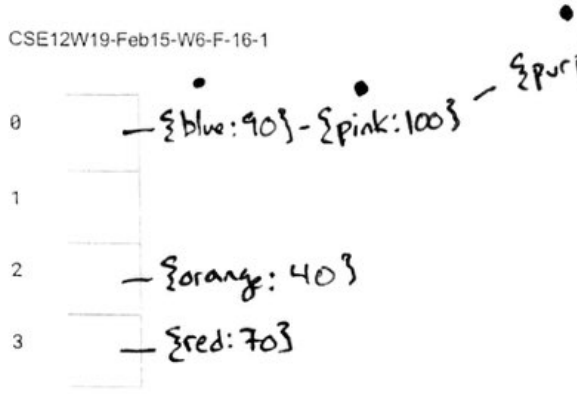


Example:
Start buckets array with size 4
Use string length as the hash function

```
add("red", 70)
add("blue", 90)
add("pink", 100)
add("orange", 40)
add("purplish", 30)
```

→ How many elements in bucket 0?
A: 0 B: 1 C: 2 (D: 3) E: more than 3

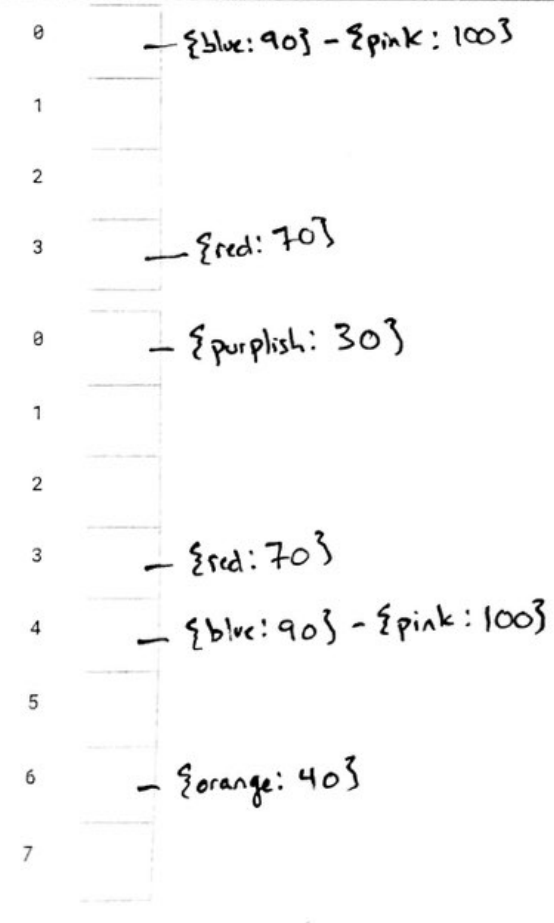
→ How many elements in bucket 1?
(A: 0) B: 1 C: 2 D: 3 E: more than 3

→ How many elements in bucket 2?
A: 0 (B: 1) C: 2 D: 3 E: more than 3

★ → How many **entries** are checked for get("purplish")?
A: 1 B: 2 C: 2 (D: 3) E: more than 3

```
A HashTable<Key, Value> using Separate Chaining has:
   •   size: an int
   •   buckets: an array of lists of Entries
   •   hash: a hash function for the Key type

An Entry is a single {key: value} pair.


void set(key, value):

→  hashed = hash(key)
   index = hashed % this.buckets.length ←
   if this.buckets[index] contains an Entry with key:
     update that Entry to contain value
   else:
     increment size
     bucket = buckets[index]
     add {key: value} to end of bucket


Value get(key):
   hashed = hash(key)
   index = hashed % this.buckets.length ←
★  if this.buckets[index] contains an Entry with key:
     return the value of that entry
   else:
     return null/report an error
```

$$\text{LoadFactor} = \frac{\#\text{elements}}{\#\text{buckets}}$$ Java 0.5 Python 0.75




```
void set(key, value):
   if LoadFactor > 0.5 : expandCapacity()

   ... as before ...


void expandCapacity():
   newBuckets = new List[this.buckets.length * 2];
   oldBuckets = this.buckets
   this.buckets = newBuckets
   this.size = 0
   for each list of entries in oldBuckets:
     for each {k: v} in the list:
       this.set(k, v)
```

Example:
Start buckets array with size 4
Use string length as the hash function

A add("red", 70) 1/4
B add("blue", 90) 1/2
C add("pink", 100) 3/4
D add("orange", 40) 1/2
E add("purplish", 30)

Which call will expandCapacity?

```
int hash1(String s) {
  return s.length();
}


int hash2(String s) {
  int hash = 0;
  for(int i = 0; i < s.length(); i += 1) {
    hash += Character.codePointAt(s, i);
  }
  return hash;
}



public int hash3(String s) {
  int h = 0;
  for (int i = 0; i < s.length(); i++) {
    h = 31 * h + Character.codePointAt(s, i);
  }
  return h;
}
```

"dog" "dim"

hash2 4 15 7 = 26 4 9 13 = 26

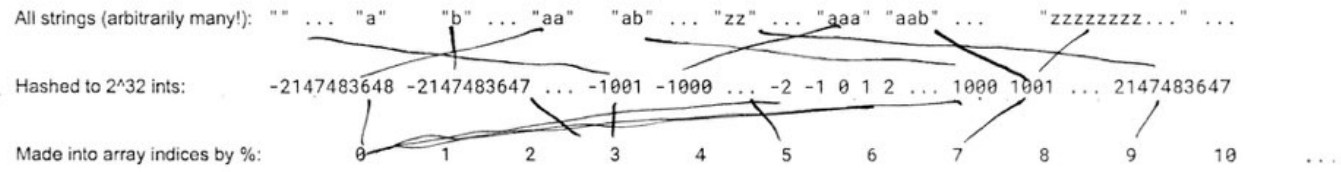
hash3 (((31*0)+4)*31+15)*31+7

(((31*0)+4)*31+9)*31+13

"dog" "log"

"ok" "ALM"

"unheavenly" "hypoplankton"



Goals:

- Evenly Distributed
- Fast (not related to # elements in hash table)
- Deterministic