Hash Tables, Dictionaries, and the Art of O(1) Lookup

n. a presentation by Matt Zhang for Algorithm Group

Dictionary: (n) an unordered and mutable collection of items composed of (key, value) pairs.

These slides are shamelessly ripped off from https://just-taking-a-ride.com/inside_python_dict/chapter1.html.
Take a look, it's interactive!

A Python dictionary is a keyword-based data organization method.

Bella = {"species":"dog", "age":1, "breed":"pit_bull", "weight":46}

Keywords can be used to reference, add, remove, or retrieve data.

Bella["species"] → "dog"

Bella["n_legs"] = 4
Bella["n_legs"] → 4

Bella.pop("breed")

How do we make a database that is rapidly searchable via keyword?

If you were stupid like me, this is how you would have done it:

```
keys = ["species", "age", "breed", "weight"]
values = ["dog", 1, "pit_bull", 46]

def find(my_key):
  for i, key in enumerate(keys):
    if key == my_key:
      return values[i]
```

O(n) search!

Hash Tables

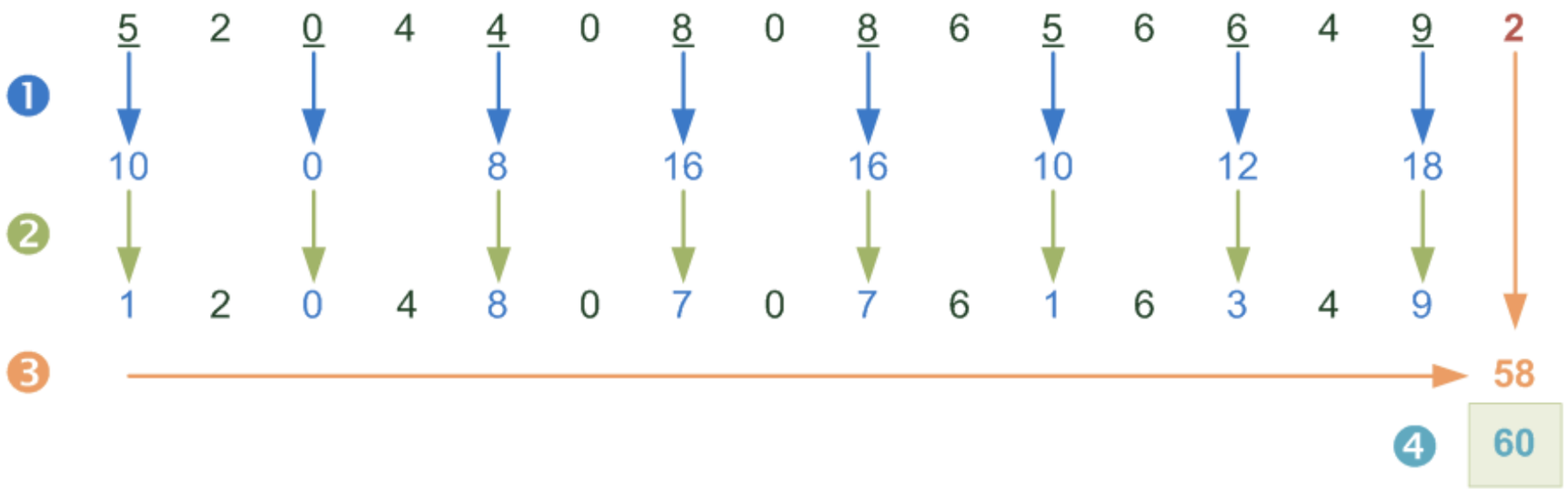
A hash function maps data of any arbitrary size onto data of a fixed size.

The value produced by a hash function is called the checksum.

A hash function is not one-to-one. You may have "collisions", but the chances of two arbitrary pieces of data colliding when hashed is very low.

Can be used for quickly comparing two pieces of data.

The Luhn algorithm is an example hash function for determining the validity of a credit card number.

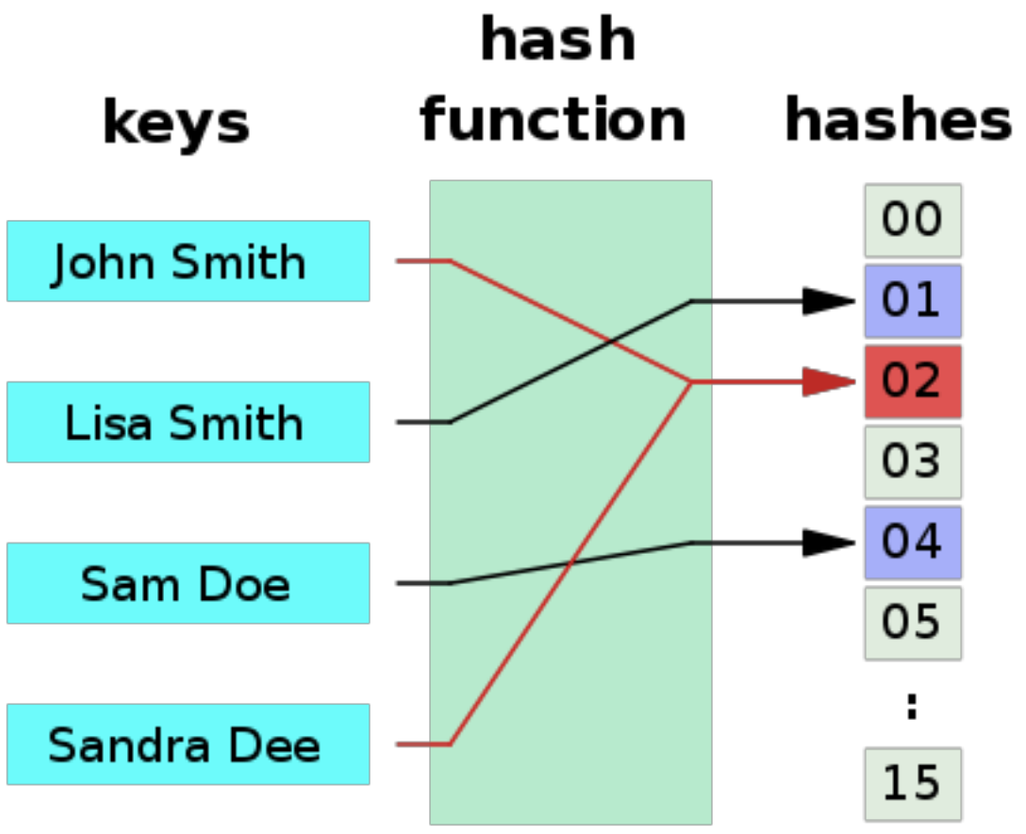


Instead of looking through a list to find a key, we can convert the key to an index via hashing.

Example:

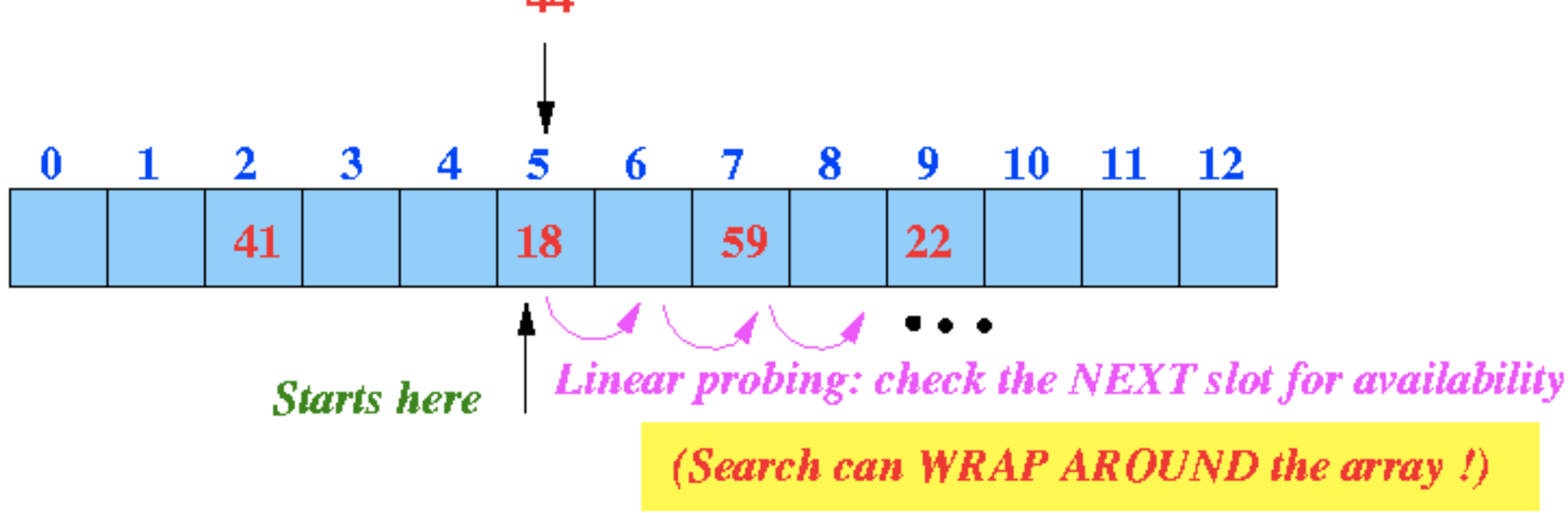
list_length = 10
key = "breed"

hash(key) = -8837423875198100574
hash(key) % list_length = 6

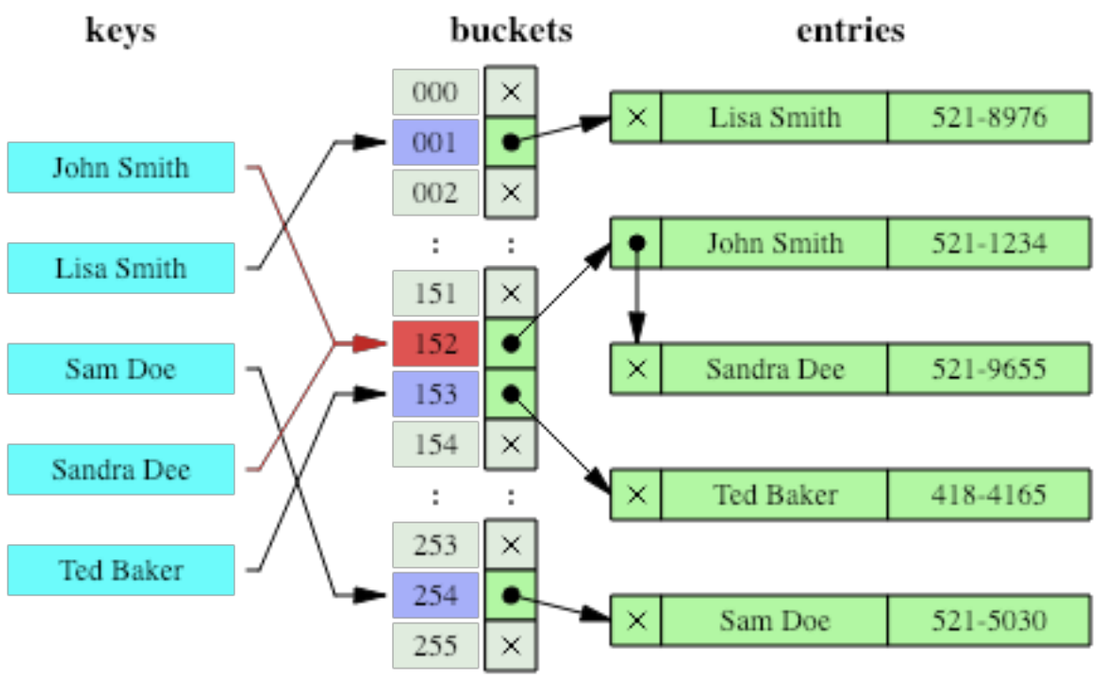

keys
hash function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

Probing Functions

When we have a collision, we need to find an empty space in the list via a probing function.

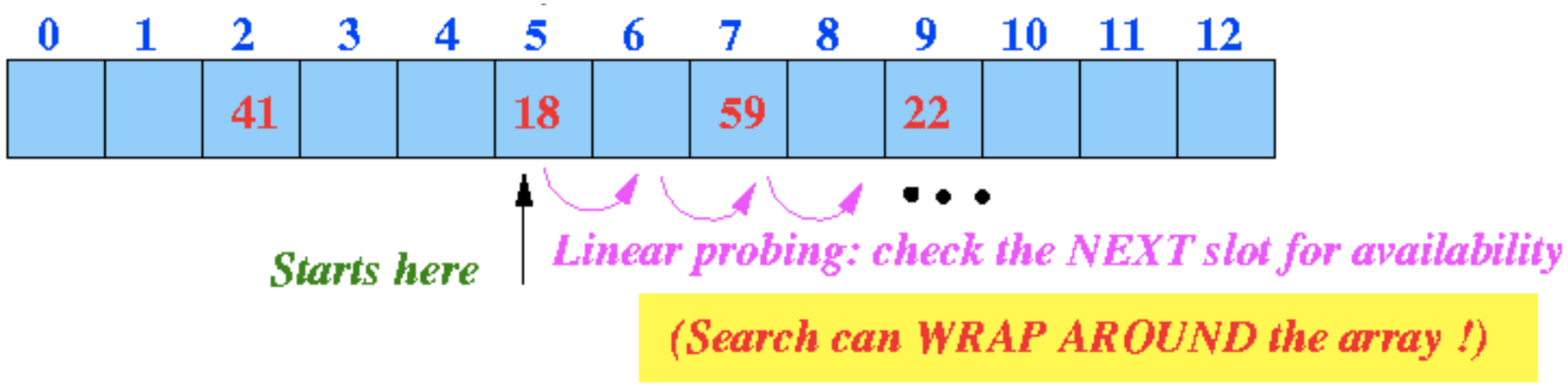


Separate chaining is another commonly used probing method.



When performing linear probing, at each probe we need to check if the existing key == the new key.

== ?



To prevent too much unnecessary probing, Python generally allocates a list with size 2x the number of keys when the hash table is created.

Since NONE is both hashable and a valid key, we need to create a special object to act as a null key.

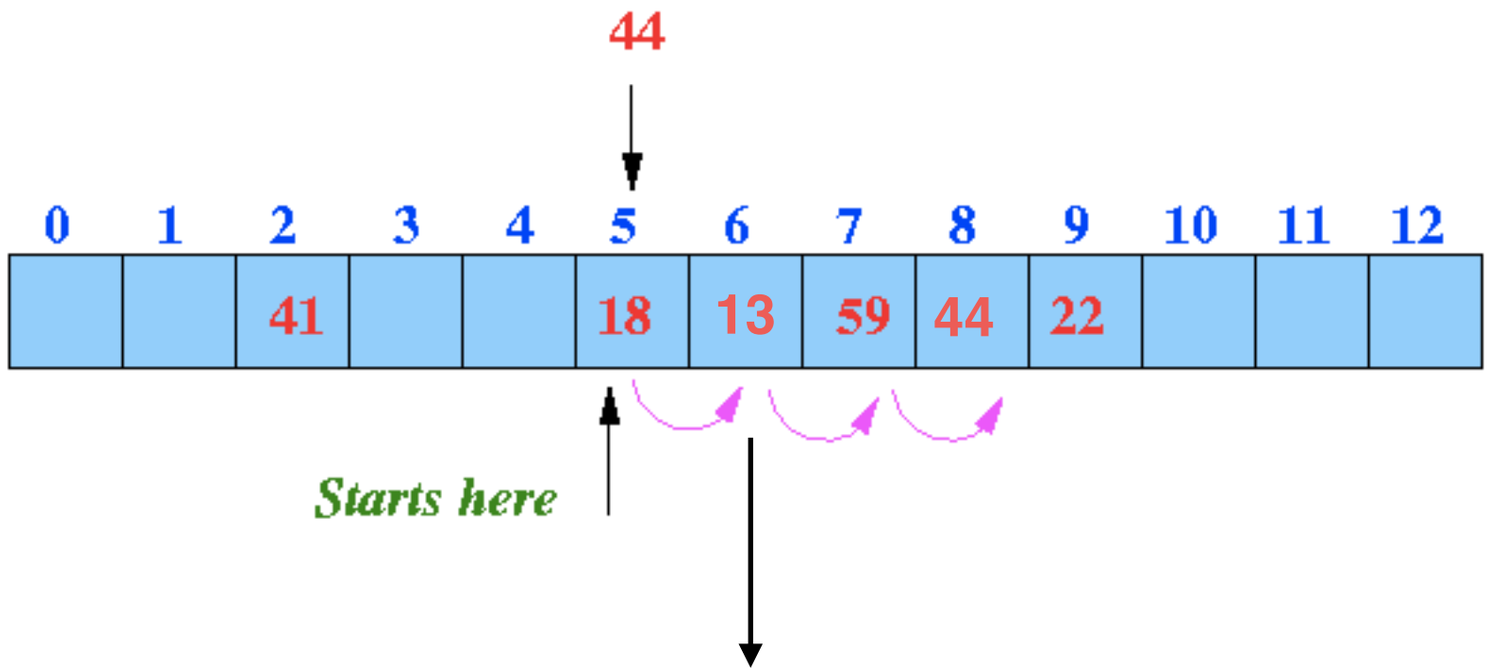
```
class EmptyEntry():
    pass

EMPTY = EmptyEntry()
```

Removing Items

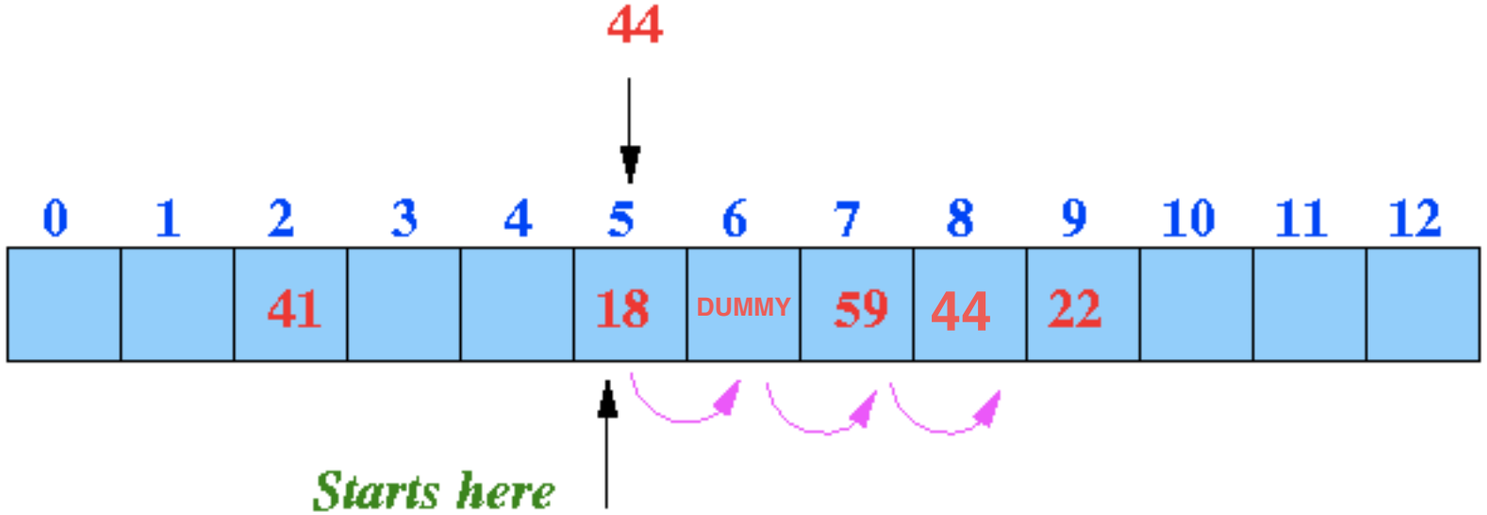
Q: If we want to remove a key, can we just find its position in the hash table and set it to EMPTY?

A: No way! Any key that probed past it could not be found if we did this!



If key 18, 13, or 59 were deleted, we would not be able to find key 44 again!

To safely remove a key, replace it with a DUMMY object.



Resizing the Table

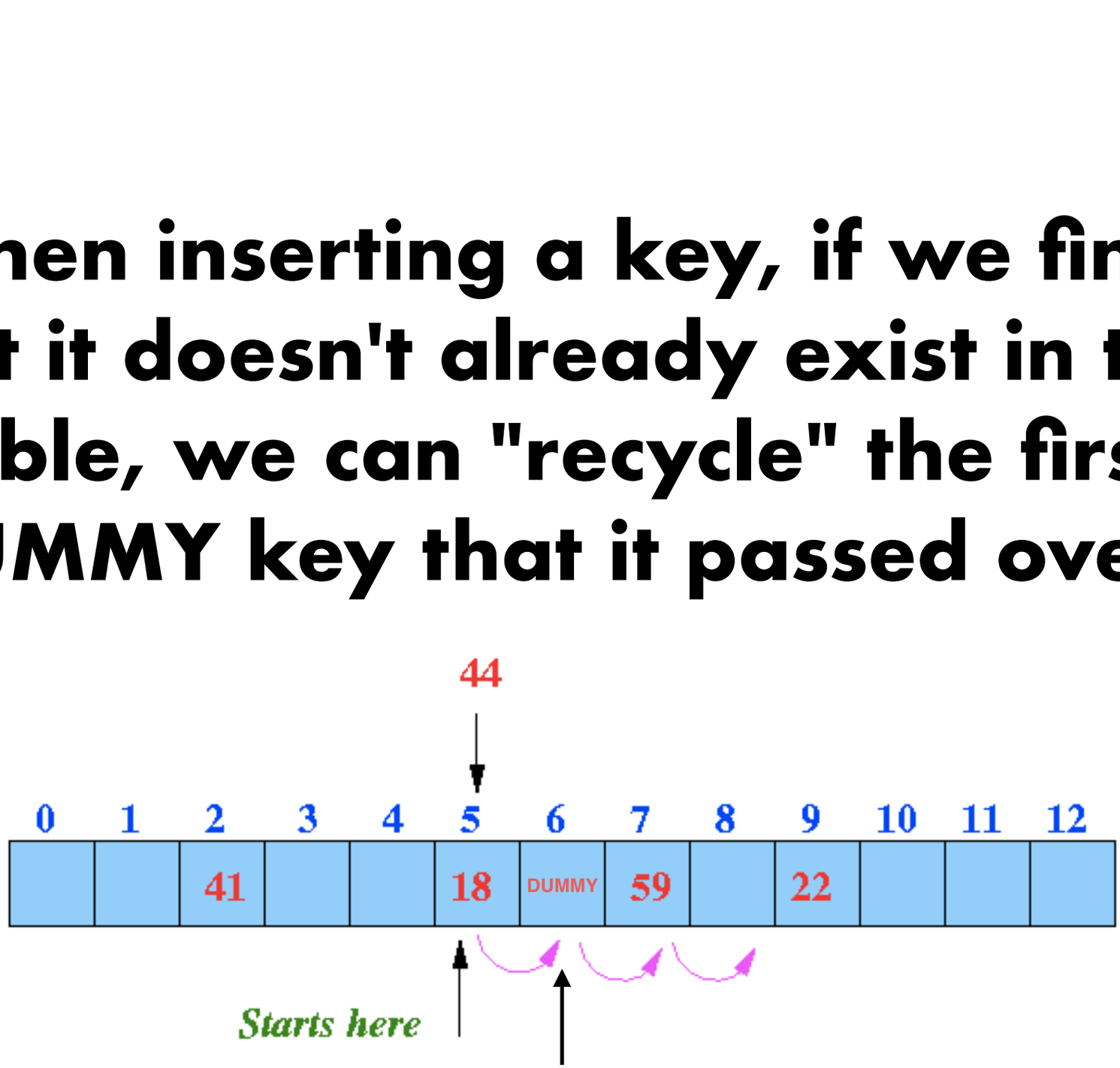
After a while, the dictionary can start getting pretty full. When the "load factor" reaches 66%, Python creates a new, larger table and enters the keys and values from the old table.

DUMMY keys can be discarded when this happens.

To prevent resizing too often, we make the new table have 2x as much space as necessary.

More Tricks

When inserting a key, if we find that it doesn't already exist in the table, we can "recycle" the first DUMMY key that it passed over.



The actual probing function in Python is not linear. Rather, it jumps around in order to prevent repeated lookups from related clusters of keys.

If a table is small, Python will quadruple its size when resizing rather than doubling.

Example Problem

3. Longest Substring Without Repeating Characters

Medium 4969 259 Favorite Share

Given a string, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
             Note that the answer must be a substring, "pwke" is a
subsequence and not a substring.
```

Using a sliding window approach, we can look at the substring from index i to j.

Deciding whether character j+1 is already in the window makes the problem O(n^2) if we check the naive way. Using a hash table makes this O(n).