HASHING

Hashing

- Hashing
- Hash Tables
- Hash Functions
- Collision Resolution
- Implementing a Hash Table

Hashing

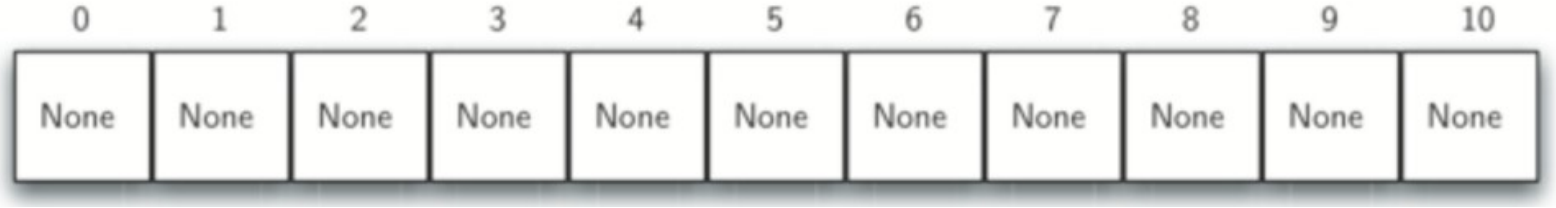
- We’ve seen how to improve search by knowing about structures beforehand.
- We can build a data structure that can be searched in O(1) time.
- This concept is referred to as **hashing.**

Hash Tables

- A **hash table** is a collection of items which are stored in such a way as to make it easy to find them later.
- Each position of the hash table, **slots**, can hold an item and is named by an integer value starting at 0.
- For example, we will have a slot named 0, a slot named 1, a slot named 2, and so on.
- Initially, the hash table contains no items so every slot is empty.

Hash Tables

- We can implement a hash table by using a list with each element initialized to the special Python value None.
- Here is an empty hash table with size m=11



Hash Tables

- The mapping between an item and the slot where that item belongs in the hash table is called the **hash function**.
- The hash function will take any item in the collection and return an integer in the range of slot names, between 0 and *m*-1.
- So how should we use hash functions to map items to slots?

Hash Function – Remainder Method

- One **hash function** we can use is the remainder method.
- When presented with an item, the hash function is the item divided by the table size, this is then its slot number.
- Let’s see an example!

Hash Function – Remainder Method

- Assume that we have the set of integer items 54, 26, 93, 17, 77, and 31.
- We've preassigned an empty hash table of m=11
- Our remainder hash function then is: **h(item)=item%11**
- Let's see the results as a table

Hash Function – Remainder Method

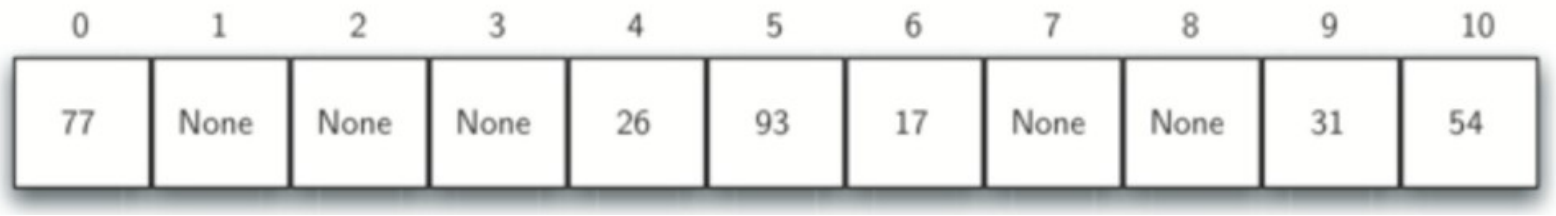
Item	Hash Value
54	10
26	4
93	5
17	6
77	0
31	9

Hash Function – Remainder Method

- We're now ready to occupy 6 out of the 11 slots.
- This is referred to as the **load factor**, and is commonly denoted by $\lambda = \frac{\text{number of items}}{\text{tablesize}}$.
- For this example, $\lambda = 6/11$.

Hash Function – Remainder Method

- Our hash table has now been loaded.



Hash Function – Remainder Method

- When we want to search for an item, we simply use the hash function to compute the slot name for the item and then check the hash table to see if it is present.
- This searching operation is O(1), since a constant amount of time is required to compute the hash value and then index the hash table at that location.

Hash Function – Remainder Method

- You might be thinking, what if you have two items that would result in the same location?
- For example 44%11 and 77%11 are the same.
- This is known as a **collision** (also known as a clash).
- We'll learn how to deal with them later on.
- Let's learn about hash functions in general!

Hash Functions

- A hash function that maps each item into a unique slot is referred to as a **perfect hash function**.
- Our goal is to create a hash function that minimizes the number of collisions, is easy to compute, and evenly distributes the items in the hash table.
- Let’s discuss a few techniques for this!

Hash Functions – Folding Method

- The **folding method** for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size).
- These pieces are then added together to give the resulting hash value.

Hash Functions – Folding Method

- If our item was the phone number 436-555-4601
- We would take the digits and divide them into groups of 2 (43,65,55,46,01).
- After the addition, 43+65+55+46+01, we get 210.
- If we assume our hash table has 11 slots, then we need to perform the extra step of dividing by 11 and keeping the remainder.
- **210 % 11** is **1**, so the phone number 436-555-4601 hashes to slot 1.

Hash Functions – Mid Square Method

- For the **mid-square method** we first square the item, and then extract some portion of the resulting digits.
- For example, if the item were 44, we would first compute $44^2=1,936$.
- By extracting the middle two digits, 93, and performing the remainder step, we get **93%11 =5**

Hash Functions

- Comparison Table

Item	Remainder	Mid-Square
54	10	3
26	4	7
93	5	9
17	6	8
77	0	4
31	9	6

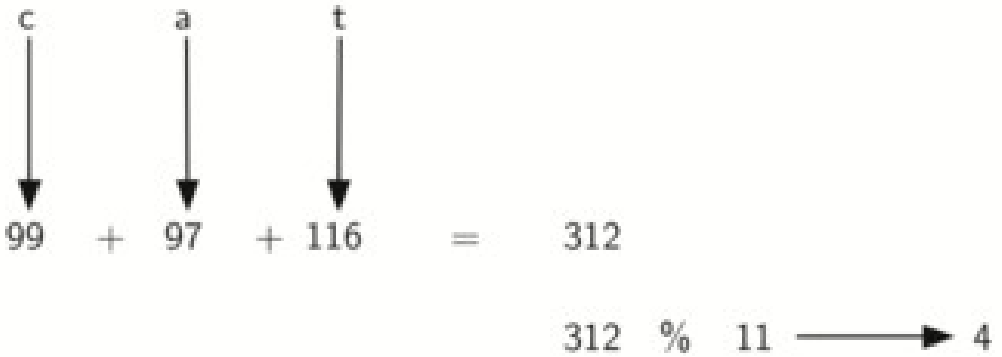
Non-integer elements

- We can also create hash functions for character-based items such as strings.
- The word “cat” can be thought of as a sequence of ordinal values.

Non-integer elements

- The word “cat” can be thought of as a sequence of ordinal values.

```
>>> ord('c')
99
>>> ord('a')
97
>>> ord('t')
116
```



Collision Resolution

- One method for resolving collisions looks into the hash table and tries to find another open slot to hold the item that caused the collision.
- We could start at the original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty.
- This collision resolution process is referred to as **open addressing** in that it tries to find the next open slot or address in the hash table.

Collision Resolution

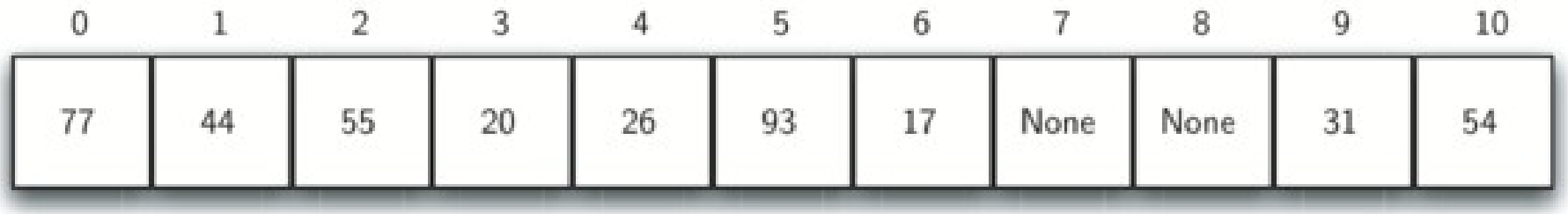
- By systematically visiting each slot one at a time, we are performing an open addressing technique called **linear probing**.

Collision Resolution

- Consider the following table:
- What if we had to add 44,55, and 20?

Collision Resolution

- With linear probing we keep moving down until we find an empty slot!



Collision Resolution

- One way to deal with clustering is to skip slots, thereby more evenly distributing the items that have caused collisions.

Collision Resolution

- The general name for this process of looking for another slot after a collision is **rehashing**.
- A variation of the linear probing idea is called **quadratic probing**.
- Instead of using a constant “skip” value, we use a rehash function that increments the hash value by 1, 3, 5, 7, 9, and so on.

Collision Resolution

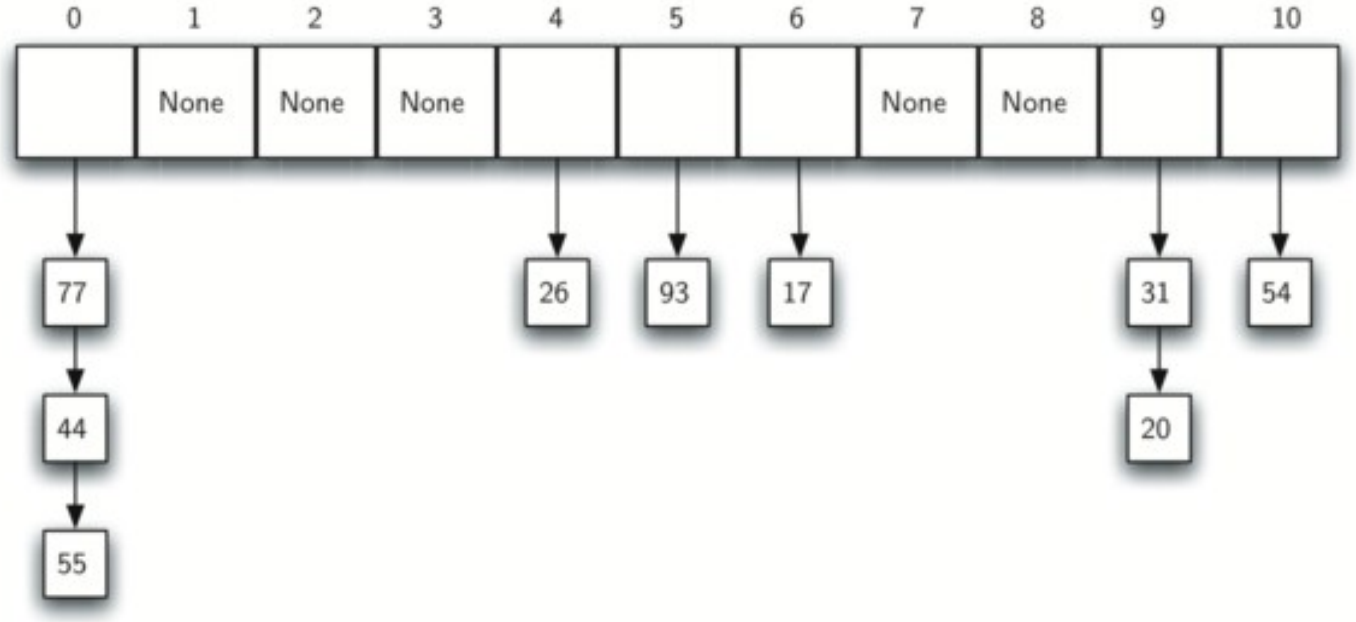
- The general name for this process of looking for another slot after a collision is **rehashing**.
- A variation of the linear probing idea is called **quadratic probing**.
- Instead of using a constant “skip” value, we use a rehash function that increments the hash value by 1, 3, 5, 7, 9, and so on.
- This means that if the first hash value is *h*, the successive values are h+1, h+4, h+9, h+16, and so on.

Collision Resolution

- An alternative method for handling the collision problem is to allow each slot to hold a reference to a collection (or chain) of items.
- **Chaining** allows many items to exist at the same location in the hash table.
- When collisions happen, the item is still placed in the proper slot of the hash table.

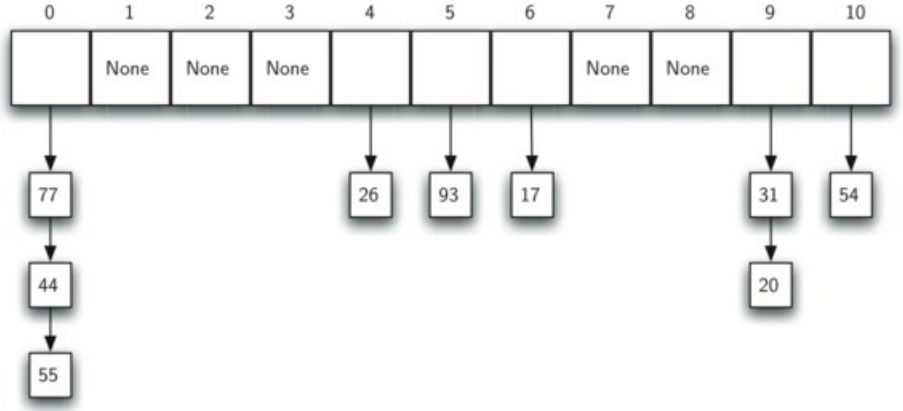
Collision Resolution

- Chaining



Collision Resolution

- As more and more items hash to the same location, the difficulty of searching for the item in the collection increases.



Review

- We’ve covered a lot!
  - Hashing
  - Hash Tables
  - Various Hash Functions
  - Collision Resolution Methods
- Up next, let’s implement our own HashTable class!