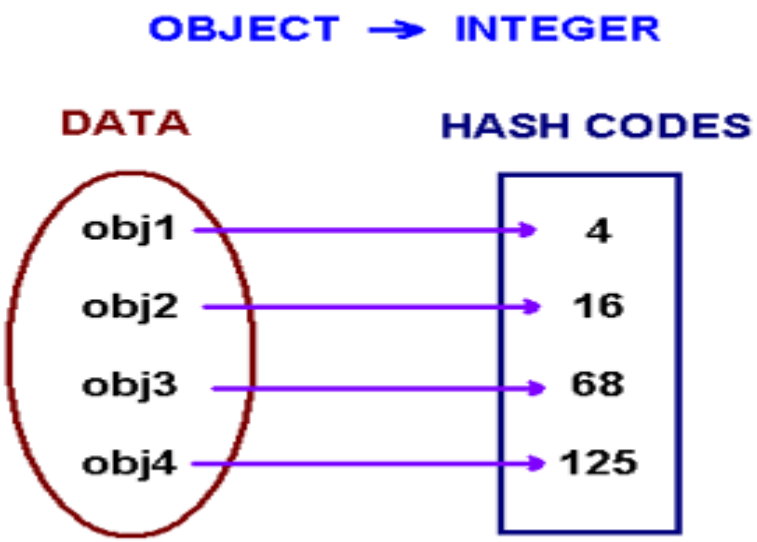
Concept of Hashing

Introduction to Hashing

- A hash function is function which when given a key, generates an address in the table.
- Hash functions are used to speed up searching.
- For unsorted data, the worse-case complexity is O(n).
- It is O(ln n) for sorted data if binary search is applied.
- We could search even faster if we have a function that would tell us the index for a given value/key.
- This gives us a constant runtime (1).

Hash function

- The example of a hash function is a *book call number*.
- Each book in the library has a *unique* call number.
- A call number is like an address: it tells us where the book is located in the library.



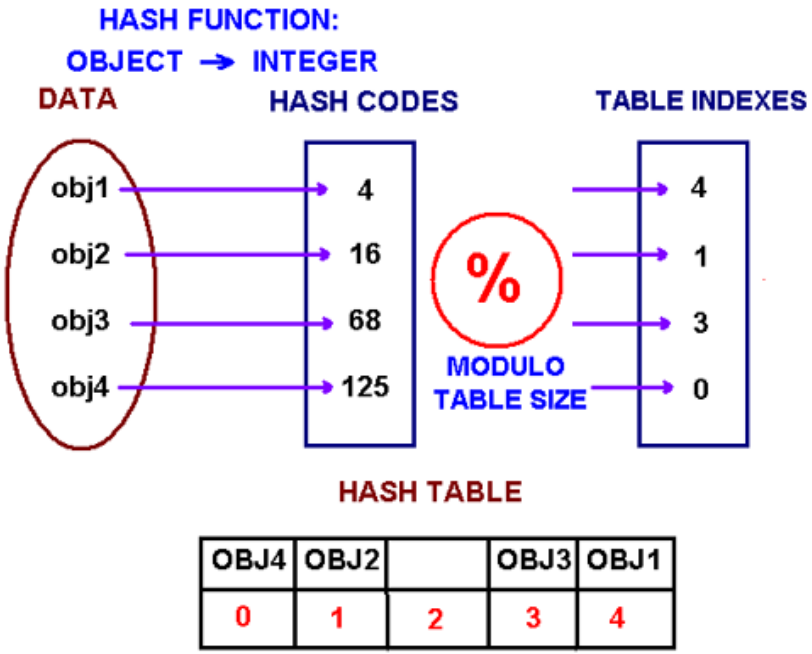
Hash function

- A hash function that returns a unique hash number is called a **universal hash function**.
- In practice it is extremely hard to assign unique numbers to objects.
- The later is always possible only if you know (or approximate) the number of objects to be processed.

- Thus, we say that our hash function has the following properties
  - it always returns a number for an object.
  - two equal objects will always have the same number
  - two unequal objects not always have different numbers

Hash function Procedure

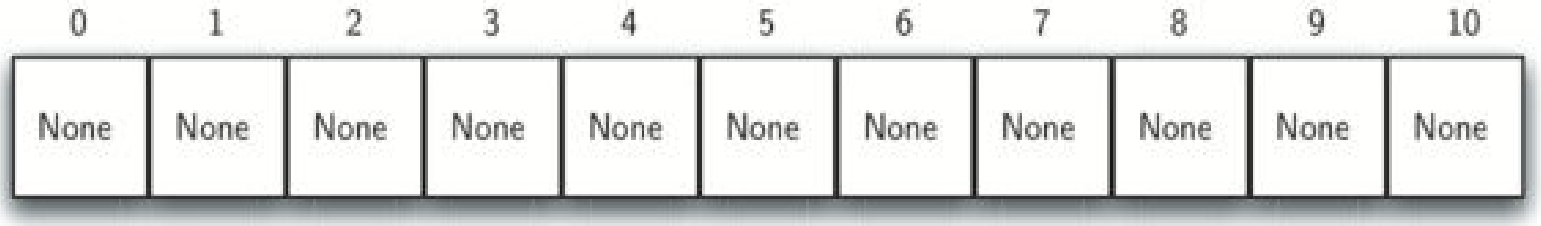
- Create an array of size *M*.
- Choose a hash function *h*, that is a mapping from objects into integers *0, 1, ..., M-1*.
- Put these objects into an array at indexes computed via the hash function *index = h(object)*.
- Such array is called a **hash table**.



Hash Table

- 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, called a **slot**, can hold an item and is named by an integer value starting at 0.

- Initially, the hash table contains no items so every slot is empty.
- Figure 4 shows a hash table of size m=11. In other words, there are *m* slots in the table, named 0 through 10.



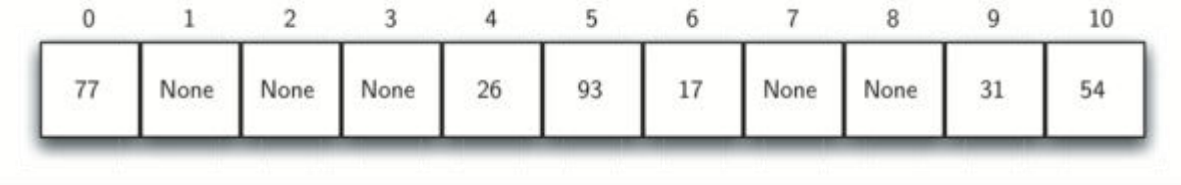
Hash Function (Remainder Method)

- Assume we have the set of integer items 54, 26, 93, 17, 77, and 31.
- Remainder method simply takes an item and divides it by the table size, returning the remainder as its hash value (h(item)=item%11).
- Table 4 gives all of the hash values for our example items.

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

Hash Function (Remainder Method)

- The result hash table is shown below.



- **Load factor** denoted by λ is =number of items /table size. For this example, λ=6/11.

- 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.

- When two items have a same hash value then the situation is called collision.
- In our case 44 and 77 has the same hash value of 0.

Hash Function

- One way to always have a perfect hash function is to increase the size of the hash table so that each possible value in the item range can be accommodated.

- Although this is practical for small numbers of items, it is not feasible when the number of possible items is large.

- For example, if the items were nine-digit Social Security numbers, this method would require almost one billion slots. If we only want to store data for a class of 25 students, we will be wasting an enormous amount of memory.

- There are a number of common ways to extend the simple remainder method. We will consider a few of them here.

Hash Function (Folding Method)

- Divide 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.
- For example, 436-555-4601, if we divide them into groups of 2 (43,65,55,46,01). After the addition, 43+65+55+46+01, we get 210.
- In this case 210 % 11 (11 slots) is 1, so the phone number 436-555-4601 hashes to slot 1.
- Some folding methods go one step further and reverse every other piece before the addition.
- For the above example, we get 43+56+55+64+01=219 which gives 219 % 11=10.

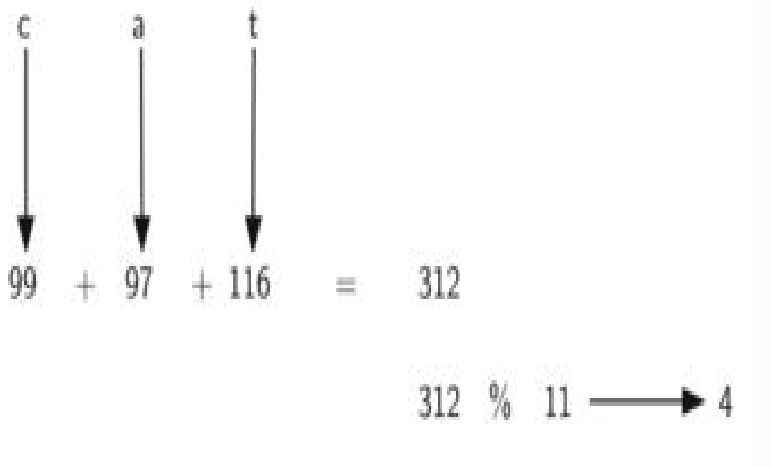
Hash Function (Mid Square Method)

- 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 (93 % 11) we get 5.
- Table below shows items under both the remainder method and the mid-square method.

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

Hash Function (Character-Based Method)

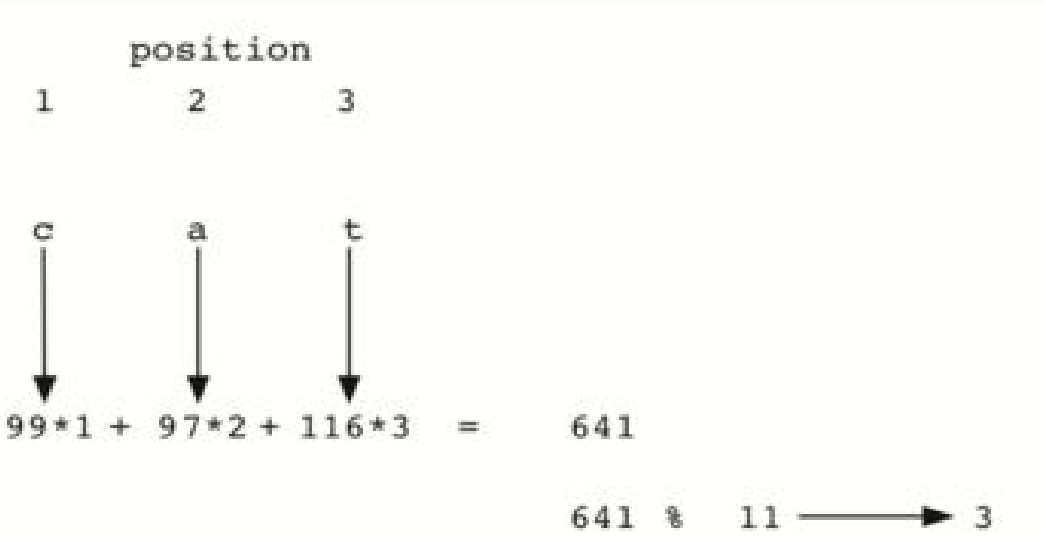
- For character-based items such as string “cat” can be thought of as a sequence of ordinal values.



Hash Function (Character-Based Method)

- It is interesting to note that when using this hash function, anagrams will always be given the same hash value.

- To remedy this, we could use the position of the character as a weight.



Collision Resolution(Open Addressing)

- When two items hash to the same slot, we must have a systematic method for placing the second item in the hash table. This process is called **collision resolution**.

- We have the following resolution process:
  1. Open Addressing
     a) Linear Probing.
     b) Plus 3 Probing.
     c) Quadratic Probing.
  2. Chaining.

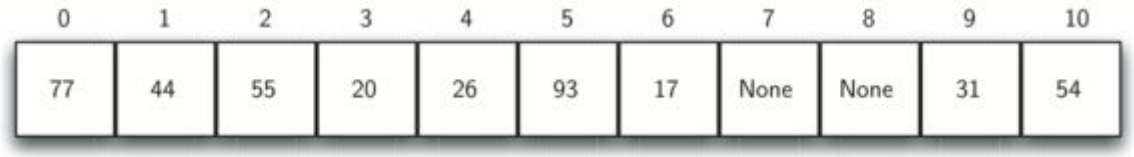
- In open addressing, we start at the original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty.

- We may need to go back to the first slot (circularly) to cover the entire hash table.

Collision Resolution(Linear Probing)

- Suppose we have (54,26,93,17,77,31,44,55,20). Table below shows the hash values.

Item	Remainder
54	10
26	4
93	5
17	6
77	0
31	9
44	0
55	0
20	9



Collision Resolution(Linear Probing)

- When we attempt to place 44 into slot 0, a collision occurs. Under linear probing, we look sequentially, slot by slot, until we find an open position. In this case, we find slot 1.

- Again, 55 should go in slot 0 but must be placed in slot 2 since it is the next open position. The final value of 20 hashes to slot 9. Since slot 9 is full, we begin to do linear probing. We visit slots 10, 0, 1, and 2, and finally find an empty slot at position 3.
- Once we have built a hash table using open addressing and linear probing, it is essential that we utilize the same methods to search for items.

- Assume we want to look up the item 93. When we compute the hash value, we get 5. Looking in slot 5 reveals 93, and we can return True.

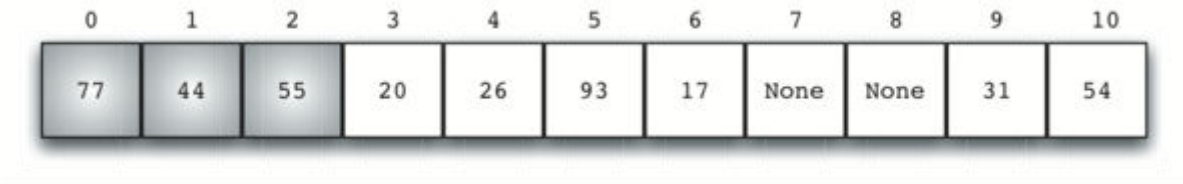
- What if we are looking for 20? Now the hash value is 9, and slot 9 is currently holding 31. We cannot simply return False since we know that there could have been collisions. We are now forced to do a sequential search, starting at position 10, looking until either we find the item 20 or we find an empty slot.

Collision Resolution(Linear Probing)

- A disadvantage to linear probing is the tendency for **clustering**; items become clustered in the table.

- This means that if many collisions occur at the same hash value, a number of surrounding slots will be filled by the linear probing resolution.

- This will have an impact on other items that are being inserted, as we saw when we tried to add the item 20 above. A cluster of values hashing to 0 had to be skipped to finally find an open position.
- For data: 54, 26, 93,17, 77, 31, 44, 55, 20



Collision Resolution(Plus 3 Probe)

- In “plus 3” probe, once a collision occurs, we will look at every third slot until we find one that is empty.

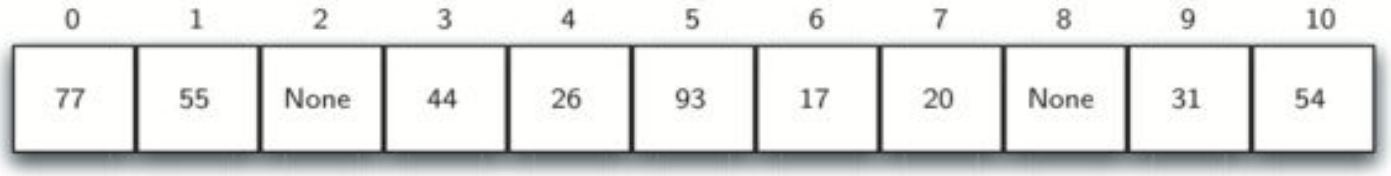
- The general name for this process of looking for another slot after a collision is **rehashing**. With simple linear probing, the rehash function is

  newhashvalue=rehash(oldhashvalue) where
  rehash(pos)=(pos+1)%sizeoftable.

- The “plus 3” rehash can be defined as

  rehash(pos)=(pos+3)% sizeoftable.

For data: 54, 26, 93,17, 77, 31, 44, 55, 20



Collision Resolution(Plus 3 Probe)

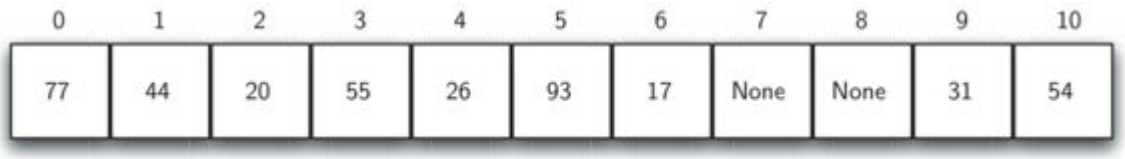
- In general, rehash(pos)=(pos+skip)%sizeoftable.
- It is important to note that the size of the “skip” must be such that all the slots in the table will eventually be visited. Otherwise, part of the table will be unused.
- To ensure this, it is often suggested that the table size be a prime number. This is the reason we have been using 11 in our examples.

Collision Resolution(Quadratic Probe)

- In **quadratic probing,** 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.

- For data: 54, 26, 93,17, 77, 31, 44, 55, 20

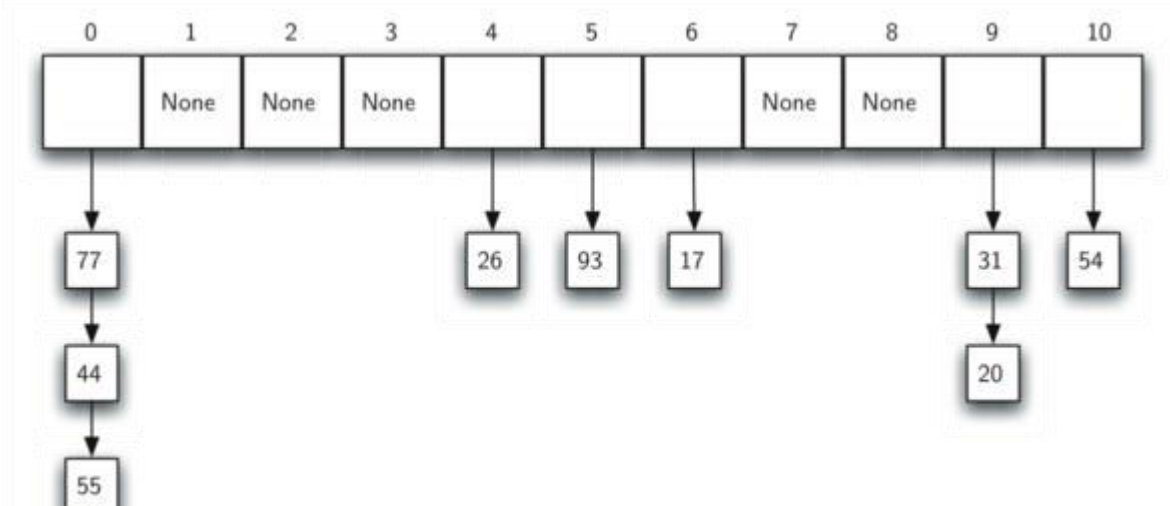


Collision Resolution(Chaining)

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

Collision Resolution(Chaining)

- When we want to search for an item, we use the hash function to generate the slot where it should reside.
- Since each slot holds a collection, we use a searching technique to decide whether the item is present.
- The advantage is that on the average there are likely to be many fewer items in each slot, so the search is perhaps more efficient.



For Hashing Document