15CSE374
INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

Sarath tv

Last Lecture

- Maps and dictionaries.
- Interfaces for dictionaries.

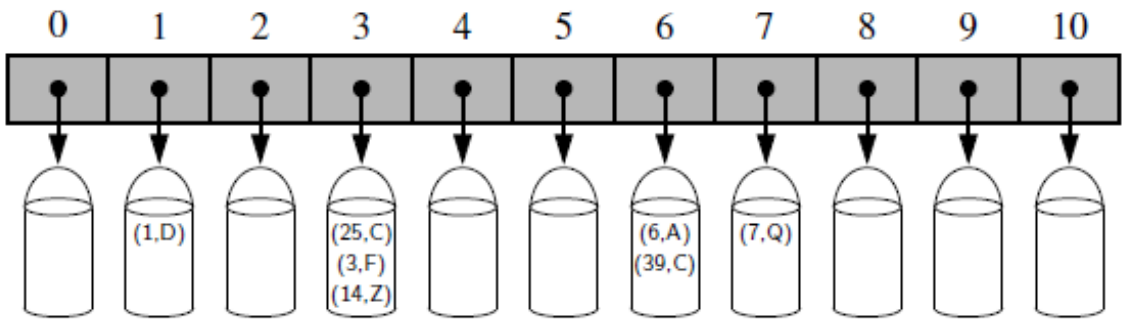
Hashing

- Hash Table- A data structure for implementing a map.
- Python uses for dict class.
- Consider a Map M uses keys as indices – M[k]
- Assumption-
  - map with n items.
  - Keys are integers .Range 0 to N-1, for some N>= n.
- Look up table of length N.

0	1	2	3	4	5	6	7	8	9	10
	D		Z			C	Q			

Hash function

- Logic: Store the value associated with key k at index k of table.
- Challenges in extending this for general setting of a map.
  - If N>>> n, we may not wish to devote an array of length N.
  - Maps keys may not be integers.
- Hash function.
  - Maps general keys to corresponding indices in a table.
- Keys to be distributed in the range 0 to N-1. but some times two or more keys get mapped to same index.
- Bucket array

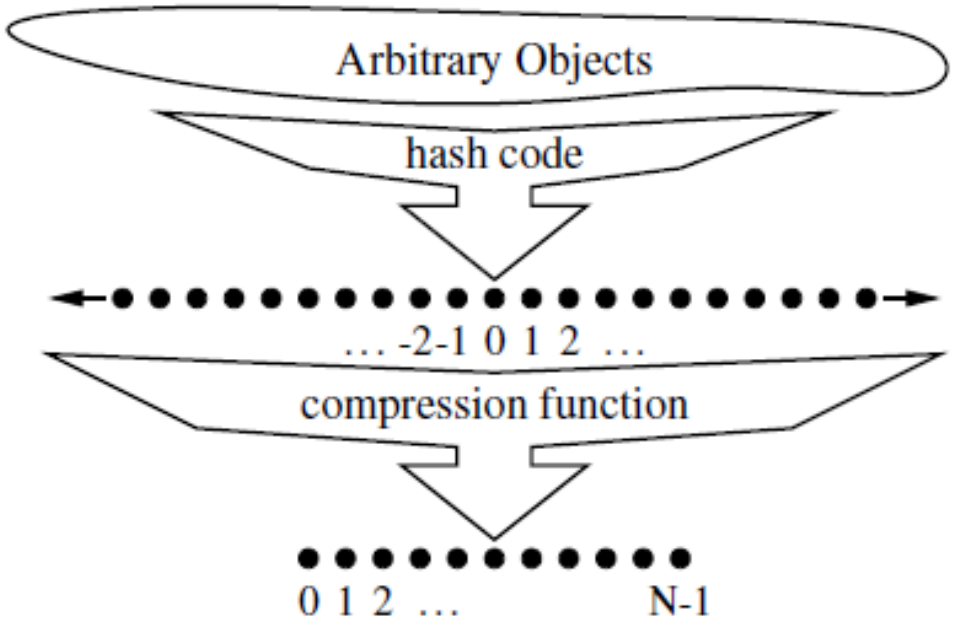


Hash function

- Goal – Map each key k to an integer in the range [0,N-1].
- Hash function – h.
- Key –k
- Use hash function value h(k) as index into our bucket array.
- An item (k,v) is stored in bucket array A[h(k)]
- V is the value for key k.
- Two or more key with same hash value→ different items will be mapped to the same bucket – Collision.
- A good has function minimize collisions sufficiently.

Hash function

- Hash function h(k)- Two portions
  - Hash code
  - Compression function.
- Hash code – Maps key k to an integer.
- Compression function – Maps the hash code to an integer within a range of indices for the bucket array.



- Separating the two components.
- Hash code portion –Independent of a specific hash table size.
- Development of general hash code to be used for any size.
- Only compression function depends on the table size.

Hash functions

- Get a numeric value which represents the string.
- Ordinal value for character.
- Sum of ordinal numbers of each character in string.

h	e	l	l	o		w	o	r	l	d	
104	101	108	108	111	32	119	111	114	108	100	= 1116

h	e	l	l	o		w	o	r	l	d	
104	101	108	108	111	32	119	111	114	108	100	= 1116

- Same hash for different strings

g	e	l	l	o		x	o	r	l	d	
103	101	108	108	111	32	120	111	114	108	100	= 1116
-1						+1					

Trade off

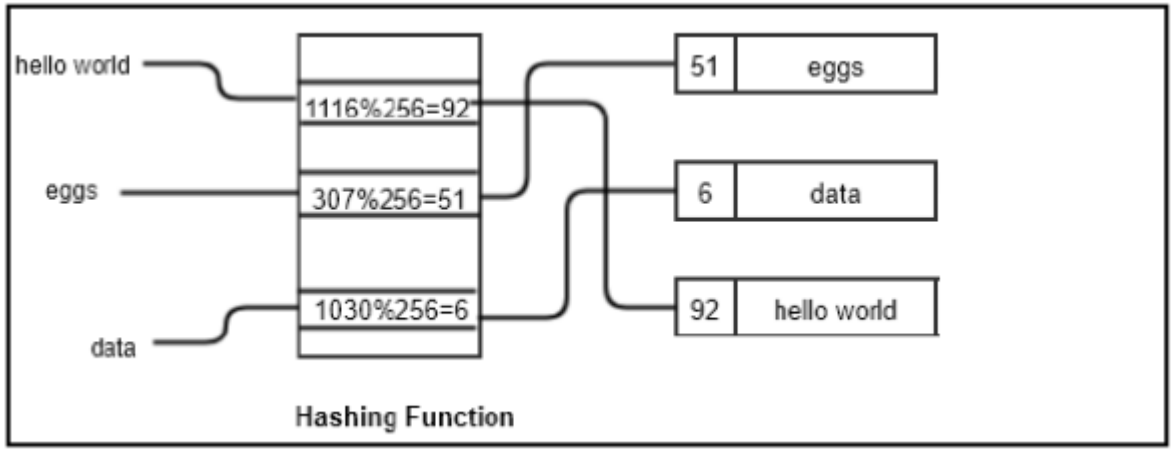
- Perfect hashing function- Unique hash value for a given string.
- Most hashing functions are imperfect and face collisions.
- Hash functions need to be very fast –so we accept that we may get some collisions.
- Rather than finding the perfect hash function ,try to resolve collisions

- To avoid collisions
- Add a multiplier- ordinal value of each character is multiplied by value that continuously increase as we progress in the string.

h	e	l	l	o		w	o	r	l	d	
104	101	108	108	111	32	119	111	114	108	100	= 1116
1	2	3	4	5	6	7	8	9	10	11	
104	202	324	432	555	192	833	888	1026	1080	1100	= 6736

HASH TABLE

- Data structure where elements are accessed by a keyword.
- Uses hashing function in order to find index position where the element should be stored and retrieved.
- Each position in the hash table DS –slots /buckets.
- Each data item –form – {key,value} pair



Compression function

- Size of hash table –Total number of slots
- Count of a hash table –number of slots filled.
- Example 256 slots-size
- So our hashing function needs to return a value in the range 0-255
  - One Solution –return reminder of dividing has value by size of the table.

0	1	2	.....	255
empty	empty	empty	.....	empty

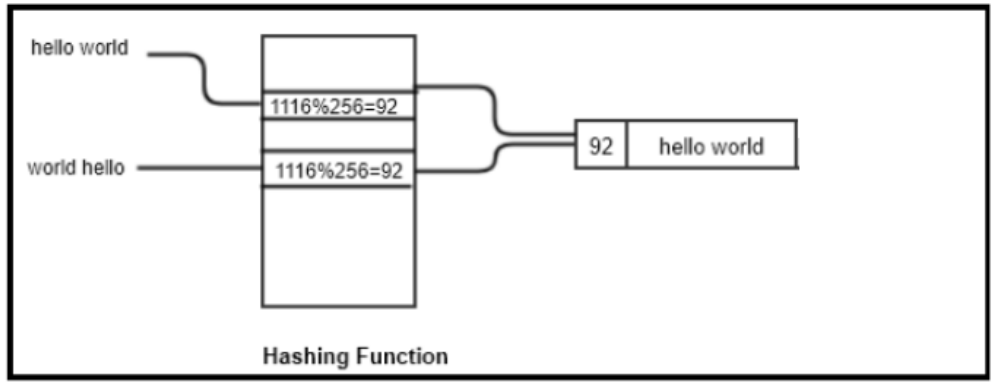
used slots = 0

Storing elements in a hash table

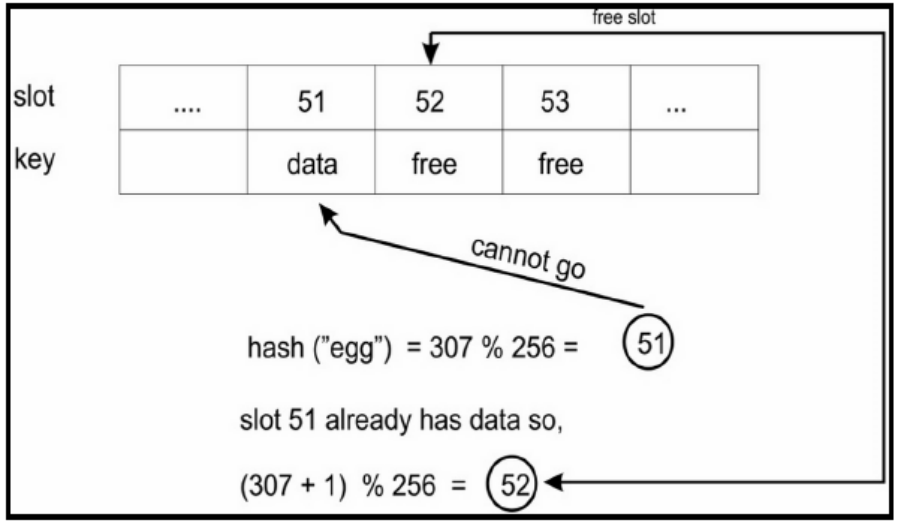
- Interface
  - put() – add item to table.
  - get () – retrieve item to the table.
- Put() – Embed key and value.
- Compute the hash value for the key.
- With the hash value –find the position where the element should be stored in the hash table.
- Look at the slot corresponding to the hash value of the key.
  - If empty –data item is added there.
  - If not empty- we have a collision.
- How to handle conflict.

Open addressing

- Way to resolve collision- find another free slot from the position of the collision.
- Linearly look for the next available slot.

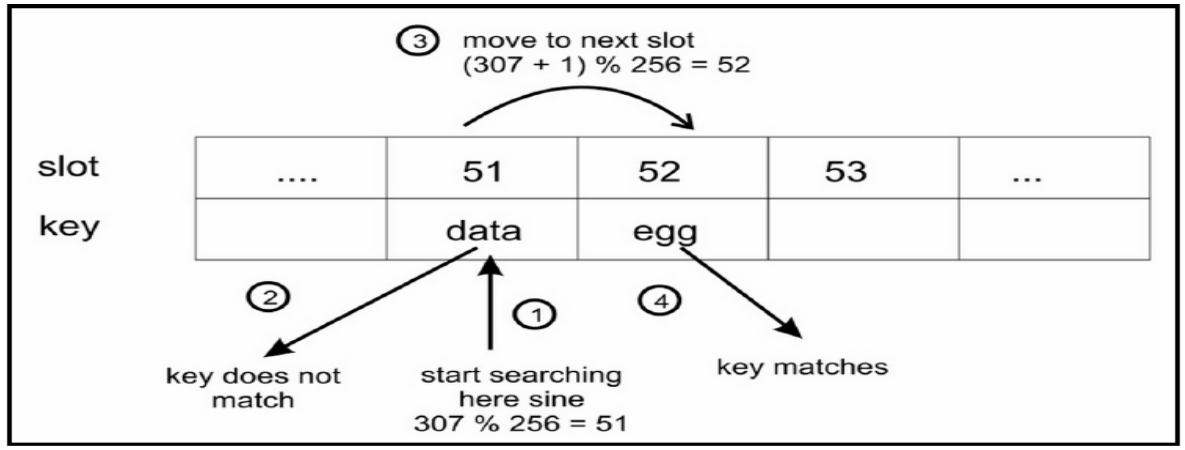


- Linear Probing



Retrieving elements from hash table

- Value stored corresponding to the key –returned.
- Compute the hash of the given key.
- Look up the hash table at the position of the hash value.
- If key matches with stored key – return value.
- Else add 1 then do the same(assuming linear probing)
- Keep looking until we get our key or we check all the slots.

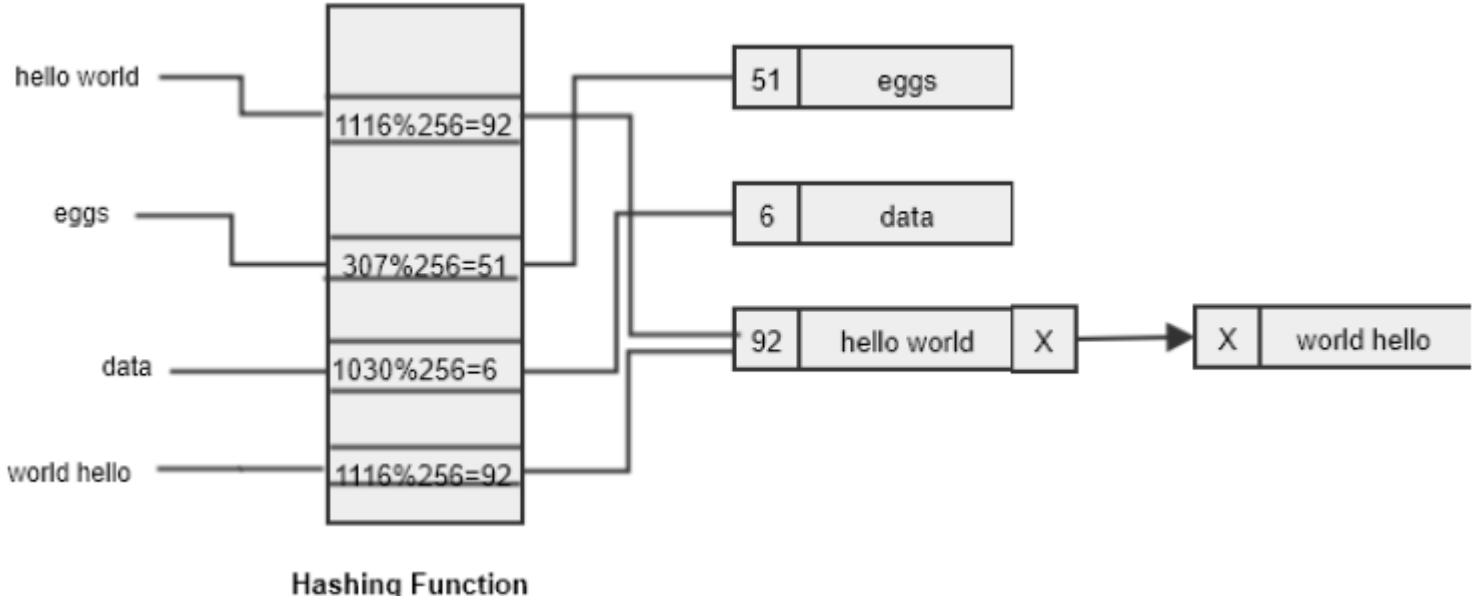


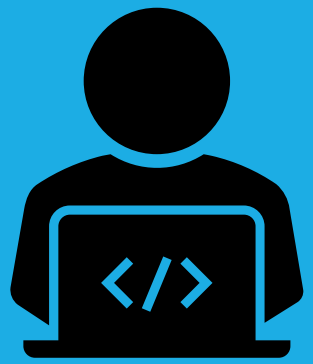
Discussions

- Growing hash table.
- Load factor.
- Open addressing.
- Chaining.

$$\text{load factor} = \frac{n}{k}$$

n is the number of used slots
k is the total number of slots





THANK YOU!!!!!