

CSC508 Data Structures

Topic 12 : Hashing

Recap

- ▶ Sequential Search
- ▶ Binary Search

Topic Structure

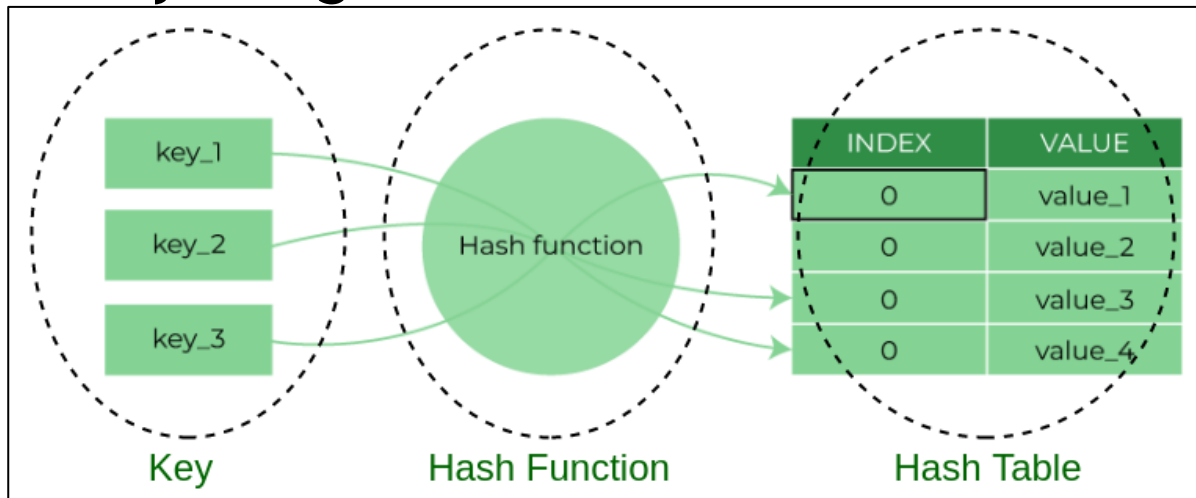
- ▶ Hashing
- ▶ Hash Method
- ▶ Collision Resolution
- ▶ Load Factor

Learning Outcomes

- ▶ At the end of this lesson, students should be able to:
 - ▶ Explain the concept of hashing and hash function
 - ▶ Describe available hashing methods
 - ▶ Describe collision resolution methods

Hashing

- ▶ Hashing is process of mapping keys, and values into the hash table by using a hash function.



<https://www.geeksforgeeks.org/hashing-data-structure/>

- ▶ Good for storing and retrieving key-value pairs
 - ▶ Not good for iterating through a list of items

Why Hashing?

- ▶ The need of data structure in which searching can be done faster.
 - ▶ Best case - $O(1)$
 - ▶ Average case - $O(1)$
 - ▶ Worst case - $O(n)$
- ▶ Objects have unique keys
 - ▶ A key may be a single property/attribute value
 - ▶ Or may be created from multiple properties/values

Hashing vs. Array vs. Linked List

► Array

- not space efficient (assumes we leave empty space for keys not currently in the structure)
- Sequential search or Binary search

► Linked List

- space efficient (node create when needed)
- Insert(), Delete() and Search()/Find() not too efficient. Sequential search.

► Hashing

- Relatively better than the above in terms of space and efficiency

Hash Value / Index

- ▶ A hash value or hash index is used to index the hash table (array)
- ▶ A hash function takes a key and returns a hash value/index
 - ▶ The hash index is a integer (to index an array)
- ▶ The key is specific value associated with a specific object being stored in the hash table
 - ▶ It is important that the key remain constant for the lifetime of the object

Hash Function

- ▶ A Hash function is a function that converts a given numeric or alphanumeric key to a small practical integer value
- ▶ A good hash function should:
 - ▶ be easy to compute
 - ▶ minimize the number of collisions
- ▶ Input for hash function can be integer key values, string key values and multipart key values (Multipart fields, and/or Multiple fields)

Hash Function (cont.)

- ▶ The performance of the hash table depends on having a hash function that evenly distributes the keys: uniform hashing is the ideal target
- ▶ Choosing a good hash function need to consider the kind of data that will be used.
 - ▶ E.g., Choosing the first letter of a last name will likely cause lots of collisions depending on the nationality of the population.
- ▶ Most programming languages (including java) have hash functions built in.

Hash Methods

- ▶ Commonly used hash methods:
 - ▶ Division
 - ▶ Mid-square
 - ▶ Folding

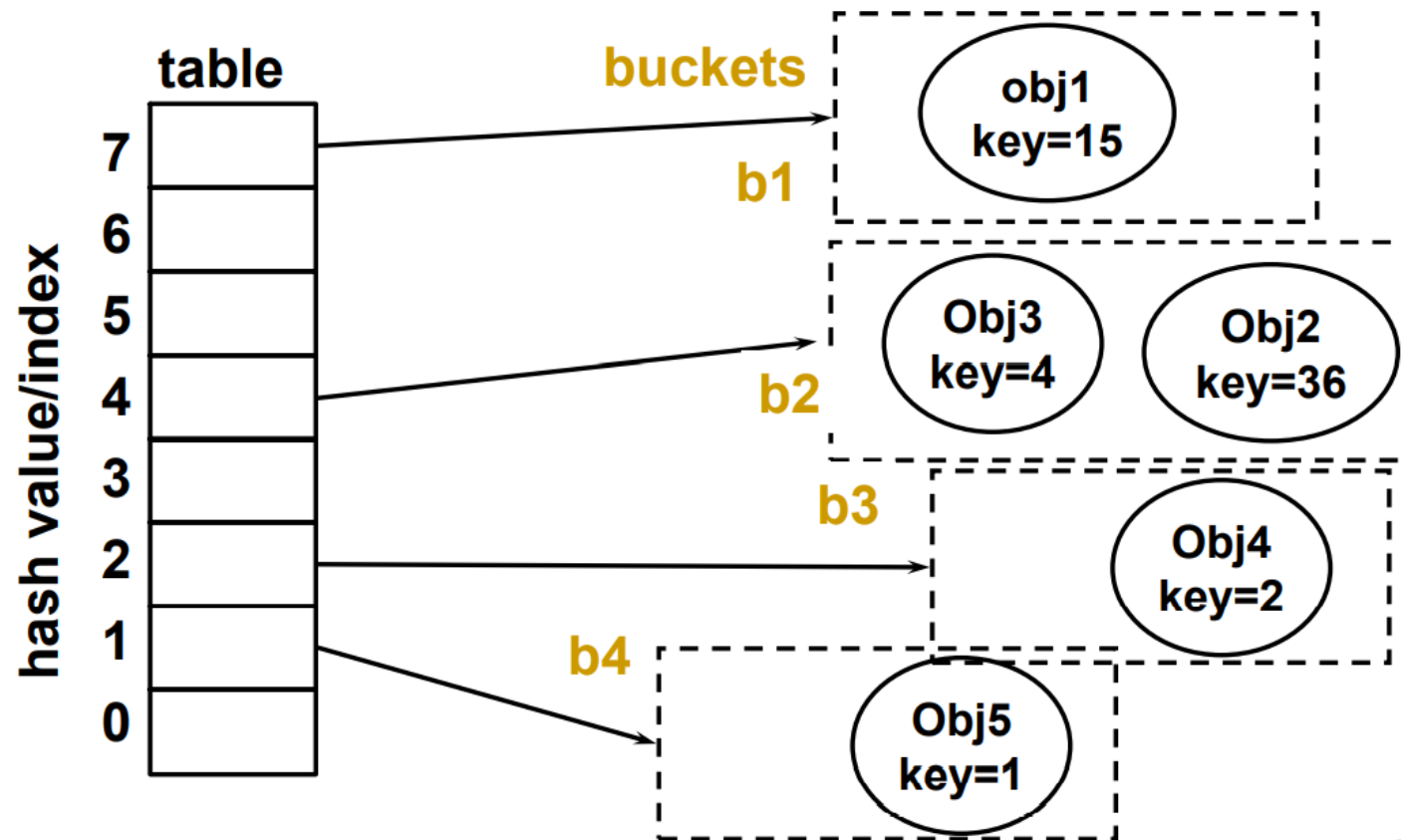
Division Method

- Divides the key (k) by the size of the array (M), and take the remainder (modulo)

$$h(K) = k \bmod M$$

- Preferable array with prime-number size.

$$h(\text{key}) = \text{key} \bmod 8$$



Division Method - String

- Suppose that each key is a string, sum the character's value in the string (based on ASCII code). Calculate hash index based on the sum.

```
int hashmethod(String insertKey) {  
    int sum = 0;  
    for(int j = 0; j <= insertKey.length(); j++)  
        sum = sum + (int) (insertKey.charAt(j));  
    return (sum % HTSize);  
}
```

Mid-square Methods

- ▶ Hash method, h , computed by squaring the identifier (key)
- ▶ Using appropriate number of bits from the middle of the square to obtain the bucket address
- ▶ Middle bits of a square usually depend on all the characters, it is expected that different keys will yield different hash addresses with high probability, even if some of the characters are the same

$$h(K) = k * k$$

Key (k)	$k * k$	$h(K)$
54	2916	91
65	4225	22
37	1369	36
89	7921	92

Folding Method

- ▶ Key X is partitioned into parts (k_1, k_2, \dots, k_n) such that all the parts, except possibly the last parts, are of equal length
- ▶ Parts then added, in convenient way, to obtain hash address

$$h(K) = k_1 + k_2 + \dots + k_n$$

Key (k)	parts	h(K)
765239	76, 52, 39	167
42641	42, 64	106
59823531	598, 235	833
98513	9,8,5,1,3	26

Hashing - Insert

Given that we use division, mod method $h(K) = k \bmod M$, with M equals to 5.

Insert **23**

[0]	
[1]	
[2]	
[3]	23
[4]	

Insert **36**

[0]	
[1]	36
[2]	
[3]	23
[4]	

Insert **70**

[0]	70
[1]	36
[2]	
[3]	23
[4]	

Insert **55**

[0]	70
[1]	36
[2]	
[3]	23
[4]	

Index 0 is
already occupied

COLLISION

Collision Resolution

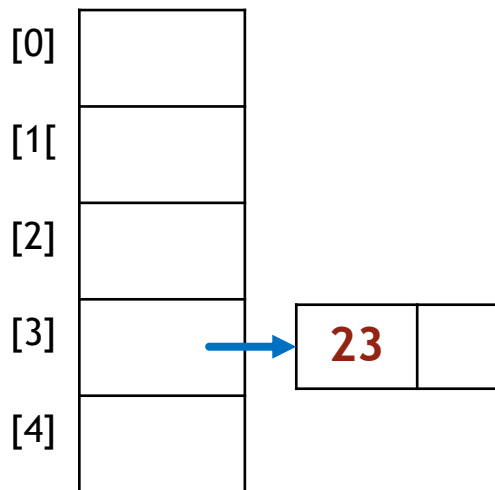
- ▶ Hash collision is a random match in hash values that occurs when a hashing algorithm produces the same hash value for two distinct pieces of data
- ▶ Two categories of collision resolution technique:
 - ▶ Chaining (open hashing)
 - ▶ Open addressing (closed hashing)

Chaining

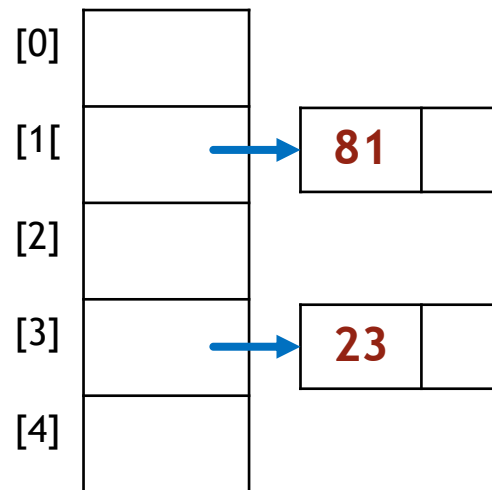
- ▶ Every hash table entry contains a pointer to a linked list of keys that hash in the same entry
- ▶ The collision is resolved by placing all keys that hash in the same hash table entry in a chain (linked list) or bucket (array) pointed by this entry.

Chaining - simulation

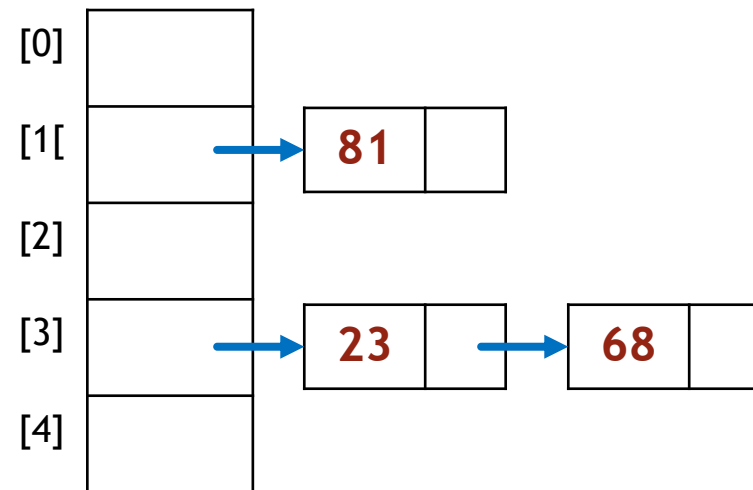
Insert **23**



Insert **81**



Insert **68**



Open Addressing

- ▶ Every hash table entry contains only one key. If a new key hashes to a table entry which is filled, systematically examine other table entries until you find one empty entry to place the new key
 - ▶ Collisions are resolved by systematically examining other table indexes, i_0 , i_1 , i_2 , ... until an empty slot is located.
- ▶ Four different algorithms to find (probe for) the next array cell
 - ▶ Linear probing
 - ▶ Quadratic probing
 - ▶ Random probing
 - ▶ Double hashing

Linear Probing

- ▶ Suppose that an item with key X is to be inserted in HT
 - ▶ Use hash function to compute index $h(X)$ of item in HT
 - ▶ Suppose $h(X) = t$.
 - ▶ If $HT[t]$ is empty, store item into array slot.
 - ▶ If $HT[t]$ already occupied by another item; collision occurs
- ▶ Linear probing: starting at location t , search array sequentially to find next available array slot:
 - ▶ $(t + 1) \% HTSize, (t + 2) \% HTSize, \dots, (t + j) \% HTSize$
 - ▶ Be sure to wrap around the end of the array!
 - ▶ Stop when you have tried all possible array indices
 - ▶ If the array is full, you need to throw an exception or, better yet, **resize the array**

Linear Probing Example

Given that we use division, mod method $h(K) = k \bmod M$, with M equals to 8.

Insert 65

[0]	
[1]	65
[2]	
[3]	
[4]	
[5]	
[6]	
[7]	

Insert 12

[0]	
[1]	65
[2]	
[3]	
[4]	12
[5]	
[6]	
[7]	

Insert 49

[0]	
[1]	65
[2]	49
[3]	
[4]	12
[5]	
[6]	
[7]	

Insert 27

[0]	
[1]	65
[2]	49
[3]	27
[4]	12
[5]	
[6]	
[7]	

Insert 81

[0]	
[1]	65
[2]	49
[3]	27
[4]	12
[5]	81
[6]	
[7]	

Random Probing

- ▶ Uses a random number generator to find the next available slot
- ▶ i^{th} slot in the probe sequence is: $h(K) + ri) \% M$, where ri is the i^{th} value in a random permutation of the numbers 1 to $M - 1$
 - ▶ Suppose $M = 101$, for $h(K) = 26$, and $r1 = 2$, $r2 = 5$, $r3 = 8$.
 - ▶ The probe sequence of X has the elements 26, 28, 31, and 34
- ▶ All insertions and searches use the same sequence of random numbers

Quadratic Probing

- ▶ In Quadratic probing, starting at position t , check the array locations

$$(t + 1^2) \% M, (t + 2^2) \% M, \dots, (t + i^2) \% M$$

- ▶ We do not know if it probes all the positions in the table
- ▶ When M is prime, quadratic probing probes about half the table before repeating the probe sequence

Double Hashing

- ▶ Apply a second hash function after the first
- ▶ The second hash function, like the first, is dependent on the key
- ▶ Secondary hash function must:
 - ▶ Be different than the first
 - ▶ Not generate a zero
- ▶ Good algorithm:
 - ▶ $\text{arrayIndex} = (\text{arrayIndex} + \text{stepSize}) \% \text{arraySize};$
 - ▶ Where $\text{stepSize} = \text{constant} - (\text{key} \% \text{constant})$
 - ▶ And constant is a prime less than the array size

Load Factor

► Utilization of the hash table

$$\alpha = \frac{\text{number of occupied table element}}{\text{table size}}$$

Closed hashing

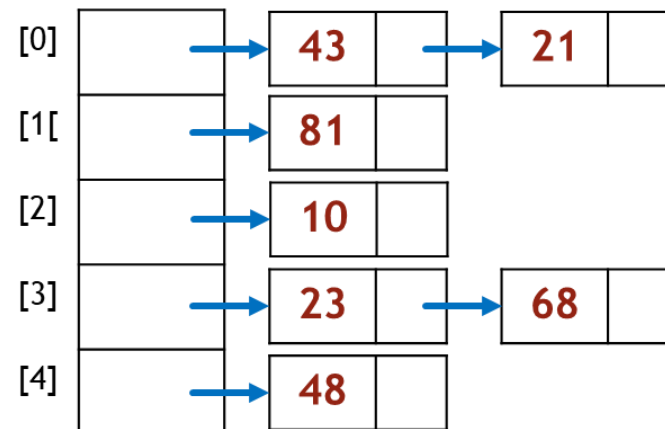
Insert **55**

[0]	70
[1]	36
[2]	
[3]	23
[4]	

$$0 < \alpha < 1$$

$$\alpha = 0.6$$

Open hashing



$$\alpha = 1.4$$

*load factor can be more than 1

Summary

- ▶ Hashing is process of mapping keys, and values into the hash table by using a hash function.
- ▶ Common hash methods : division, mid-square, and folding
- ▶ Collision happens when there is one hash value for more than one key
- ▶ Collision resolution approach : Chaining, Open addressing
- ▶ Open addressing probing : Linear probing, quadratic probing, random probing, double hashing

Next Topic...

► Graph

References

- ▶ Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- ▶ Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- ▶ Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.

TEST 2

- ▶ Date : 29 January 2023
- ▶ Time : 10:45 a.m. - 12:15 a.m.
- ▶ Topic Covered
 - ▶ Recursion
 - ▶ Sorting Algorithm
 - ▶ Searching
 - ▶ Hashing