

Hashing and Collision

Unit#5



Marwadi
University

Department of
Computer Engineering

Data Structure
01CE0301 / 3130702

Ravikumar R N

Highlights

- Hashing Concepts and methods.
- Hash Table Methods - Introduction, Hashing Functions.
- Collision and its understanding.
- Discuss different Collision-Resolution Techniques with examples.



Why Hashing?

- Internet has grown to millions of users generating terabytes of content every day.
- With this kind of growth, it is impossible to find anything in the internet, unless we develop new data structures and algorithms for storing and accessing data.



Why not traditional data structures?

- Traditional data structures like Arrays and Linked Lists?
- Amount of time required to look up an element in the array is either $O(\log n)$ or $O(n)$.
 - If array is sorted then binary search or searched linearly.
- Either case may not be desirable if we need to process a very large data set.



Why Hashing?

- Therefore we discuss a new technique called hashing that allows us to update and retrieve any entry in constant time $O(1)$.
- The constant time or $O(1)$ performance means, the amount of time to perform the operation does not depend on data size 'n'.



Hash Function

- Pair is of the form (key, value), where for given a key, we can find a value using some kind of a “function” that maps keys to values.
- The key for a given object can be calculated using a function called a hash function.
- For example, given an array A, if i is the key, then we can find the value by simply looking up A[i].



HashTable

- The concept of a hash table is a generalized idea of an array where key does not have to be an integer.
- We can have a name as a key, or for that matter any object as the key.
- The trick is to find a hash function to compute an index so that an object can be stored at a specific location in a table such that it can easily be found.



Example

- Suppose we have a set of strings {"abc", "def", "ghi"} that we would like to store in a table.
- Our objective here is to find or update them quickly from a table, actually in $O(1)$.
- We are not concerned about ordering them or maintaining any order at all.
- Suppose we assign "a" = 1, "b"=2, ... etc to all alphabetical characters, then simply compute a number for each of the strings.
 - "abc" = $1 + 2 + 3 = 6$
 - "def" = $4 + 5 + 6 = 15$
 - "ghi" = $7 + 8 + 9 = 24$



Example

- If we assume that we have a table of size 5 to store these strings,
- we can compute the location of the string by taking the sum mod 5.
- So we will then store “abc” in $6 \bmod 5 = 1$, “def” in $15 \bmod 5 = 0$, and “ghi” in $24 \bmod 5 = 4$.
- So locations 1, 0 and 4 as follows.

0	1	2	3	4
def	abc			ghi



Example

- We can immediately compute the location using a simple hash function, which is **sum of the characters mod Table size**.
- Using this hash value, we can search for the string.
- This seems to be great way to store a Dictionary.
- Therefore the idea of hashing seems to be a great way to store pairs of (key, value) in a table.



Problem with Hashing

- The method discussed seems too good as we begin to think more about the hash function.
- First of all, the hash function we used, that is the sum of the letters, is a bad one.
- In case we have permutations of the same letters, "abc", "bac" etc in the set, we will end up with the same value for the sum and hence the key.
- In this case, the strings would hash into the same location, creating what we call a "collision".



Problem with Hashing

- Question 1: How do we pick a good hash function?
- Question 2: How do we deal with collisions?



Question 1 : hash function

- Picking a “good” hash function is key to successfully implementing a hash table.
- What we mean by “good” is that the function must be easy to compute and avoid collisions as much as possible.



Question 1 : hash function

- Properties of Good Hash Function
 - Low Cost – Binary search find value with $\log n$, so hash function must cost less than this
 - Determinism – same hash value for same input
 - Uniformity – uniform hash values minimize the number of collisions



Functions or Methods in Hashing

1. Division Method
2. Midsquare Method
3. Folding Method
4. Multiplication Method
5. Algebraic Method
6. Digit Analysis
7. Length Dependent Method



Division Method

- This method divides Key K by Prime number M and then uses the remainder thus obtained.
- $h(K) = K \bmod M$
- Example:
 $h(12345) = 12345 \% 95 = 90$



Midsquare Method

- Square the value of the Key
- Extract the middle r bits of the result
- Example:
if $K = 12345$
 $\text{Square}(K) = 152399025$
middle 3 bits: 399
middle 2 bits: 39



Folding Method

- Divide the Key value into a number of parts.
 - K into $k_1, k_2, k_3, \dots, k_n$. where each part has same number of digits except the last one.
- Add the individual parts.
 - Obtain $k_1+k_2+k_3+\dots+k_n$. the hash value is obtained by ignoring the last carry.
- Example:
K = 12345: $k_1=12, k_2=34, k_3=5$
 $k_1+k_2+k_3 = 51$



Multiplication Method

- Choose a constant A such that $0 < A < 1$
- Multiply Key K by A
- Extract the fractional part of KA
- Multiply the result by m and take floor
- $h(K) = \text{floor}[m (KA \bmod 1)]$
- Example:
$$\begin{aligned} h(12345) &= \text{floor}[100 (12345 * 0.357840 \bmod 1)] \\ &= \text{floor}[100 (4417.5348 \bmod 1)] \\ &= \text{floor}[100 (0.5348)] \\ &= \text{floor}[53.48] \\ &= 53 \end{aligned}$$



Algebraic Method

- Suppose we need to store a dictionary in a hash table.
- A dictionary is a set of Strings and we can define a hash function as follows.
- Assume that S is a string of length n and

$$S = S_1 S_2 S_3 \dots S_n$$

$$H(S) = H("S_1 S_2 S_3 \dots S_n")$$

$$= S_1 + pS_2 + p^2S_3 + \dots + p^{n-1}S_n$$
- where p is a prime number. Obviously, each string will lead to a unique number



Collisions

- It is difficult to find a “perfect” hash function, that is a function that has no collisions.
- One problem with hashing is that it is possible that two strings can hash into the same location.
- This is called a collision.



Collision Resolution/ Types of Hashing

- If two keys hash to the same index, the corresponding records cannot be stored in the same location.
- So, if it's already occupied, we must find another location to store the new record, and do it so that we can find it when we look it up later on.
- There are number of collision resolution techniques, but the most popular are,
- **Open/External Hashing (Closed Addressing)**
 - Chaining
- **Closed/Internal Hashing (Open Addressing)**
 - Linear Probing (Default)
 - Quadratic Probing
 - Double Hashing



Open Addressing

- Open addressing hash tables can store the records directly within the array.
- A hash collision is resolved by probing, or searching through alternate locations in the array (the probe sequence) until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table.



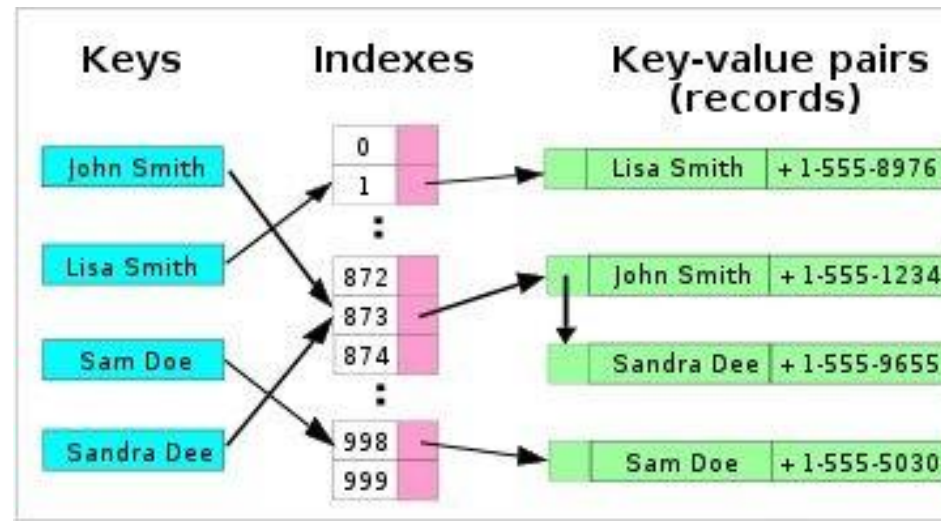
Open Addressing

- Linear probing: looking for the next available location $i+1$, $i+2$, etc. from the hashed value i .
- Quadratic probing: same as linear probing, except we look for available positions $i+1$, $i+4$, $i+9$, etc from the hashed value i .



Chaining

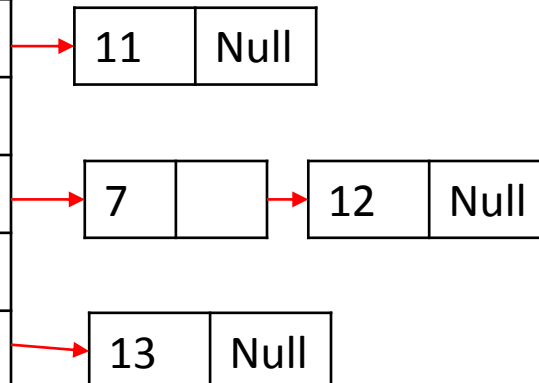
- Each slot in the array references a linked list of inserted records that collide to the same slot.
- Insertion requires finding the correct slot, and appending to either end of the list in that slot; deletion requires searching the list and removal.



Open Hashing – Chaining Example

- Consider $h(k)=2k+3$ and use open hashing and division method to store the following keys: 3,2,9,6,11,13,7,12 and $m=10$
- We know that division method is $h(k)=k \bmod m$

Address	Key
0	
1	9
2	
3	
4	
5	6
6	
7	2
8	
9	3

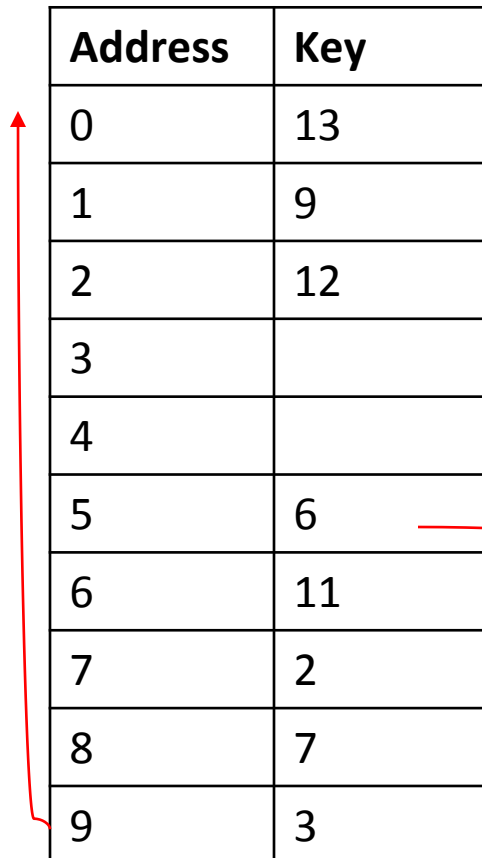


Key	Location(u)
3	$((2*3)+3) \% 10 = 9$
2	$((2*2)+3) \% 10 = 7$
9	1
6	5
11	5
13	9
7	7
12	7



Closed Hashing – Linear Probing Example

- Consider $h(k)=2k+3$ and use closed hashing and division method to store the following keys: 3,2,9,6,11,13,7,12 and $m=10$
- $(u+i)\%m$ – insert K_i at next free place
- We know that division method is $h(k)=k \bmod m$



Address	Key
0	13
1	9
2	12
3	
4	
5	6
6	11
7	2
8	7
9	3

Key	Location(u)	Probes
3	$((2*3)+3) \% 10 = 9$	1
2	$((2*2)+3) \% 10 = 7$	1
9	1	1
6	5	1
11	5	2
13	9	2
7	7	2
12	7	6



Closed Hashing – Quadratic Probing Example

- Consider $h(k)=2k+3$ and use closed hashing - quadratic probing and division method to store the following keys: 3,2,9,6,11,13,7,12 and $m=10$
- $(u+i^2)\%m$ – insert K_i at next free place, $i=0$ to 9
- We know that division method is $h(k)=k \bmod m$

Address	Key
0	13
1	9
2	
3	12
4	
5	6
6	11
7	2
8	7
9	3

Key	Location(u)	Probes
3	$((2*3)+3) \% 10 = 9$	1
2	$((2*2)+3) \% 10 = 7$	1
9	1	1
6	5	1
11	5	2
13	9	2
7	7	2
12	7	5

7 -> $(7+0^2)\%10=7$
 7 -> $(7+1^2)\%10=8$

12 -> $(7+0^2)\%10=7$
 12 -> $(7+1^2)\%10=8$
 12 -> $(7+2^2)\%10=1$
 12 -> $(7+3^2)\%10=6$
 12 -> $(7+4^2)\%10=3$

11 -> $(5+0^2)\%10 = 5$ //address 5 already occupied

11 -> $(5+1^2)\%10 = 6$ //free place so store 11 at address 6

13 -> $(9+0^2)\%10 = 9$ //address 9 already occupied

13 -> $(9+1^2)\%10 = 0$ //free place so store 13 at address 0



Closed Hashing – Double Hashing Example

- Consider $h_1(k)=2k+3$, $h_2(k)=3k+1$ and use closed hashing and division method to store the following keys: 3,2,9,6,11,13,7,12 and $m=10$
- $(u+v*i)\%m$ and $v=[h_2(k)\%m]$
- We know that division method is $h(k)=k \bmod m$

Address	Key
0	
1	9
2	
3	11
4	12
5	6
6	
7	2
8	
9	3

Key	Location(u)	V	Probes
3	$((2*3)+3) \% 10 = 9$		1
2	$((2*2)+3) \% 10 = 7$		1
9	1		1
6	5		1
11	5	4	3
13	9		X
7	7		X
12	7		6

11

$V=(3*11+1) \% 10=4$
 $5+4*0 \% 10 = 5$
 $5+4*1 \% 10 = 9$
 $5+4*2 \% 10 = 3 // \text{Store}$

13

$V=(3*13+1) \% 10=0$

$5+0*0 \% 10 = 9$
 13 cannot be stored



Review

- Hashing Concepts and methods.
- Hash Table Methods - Introduction, Hashing Functions.
- Collision and its understanding.
- Discuss different Collision-Resolution Techniques with examples.



Thank You.

