Hash Tables

(Hash Maps)

- Constant time search. O(1)

- independent of the data size.

There n elements,
1st element: fixed
kth element: fixed. (k < n)
nth element: fixed.

Key[Address]
Value [Its contents., single valued object, or, complex object.]

Key = Function(Hash) of Value.
We store the **Value** in the **Key**.

>> simple

: periodic operation

Assume: m keys, for example m = 10.

Whatever data comes,

we convert it to the period of m.

23 => 23 % 10 = 3 (its location.)

47 => 47 % 10 = 7 (its location.)

999 => 999 % 10 = 9 (its location)

key of 23 = 3, value = 23

key of 47 = 7, value = 47

key of 999 = 9, value = 999

key of 19 = 9, ???

>> complex
=> complex mathematical operations are used
to generate key.
=> password hasing.
=> abc => ???

Linear Searching O(n)

- we look all the elements.

Binary Searching O(logn)

- ordered data
- partition in 2 parts,

Hashing

- Calculation applied to a key to transform it into an address.

- For Numeric Keys, divide the key by the number of available addresses, m, and take the remainder

>> address = key % m

- For Alphanumeric keys, divide the sum of ASCII codes in a key by the number of available addresses, m, and take the remainder.

“ab12” => 97 + 98 + 49 + 50
= 294 =

294 % 10 = 4

- Folding method divides key into equal parts then adds the parts together.

>> The telephone number 01452 834 5654, becomes 01 + 45 + 28 + 34 + 56 + 54 = 218

218 % 10 = 8 = Key

>> Depending on the size of table, may then divide some constant and take the remainder.

>> Sometimes mid-square method could be utilized for hash value generation.

Data 234 =>

mid-value = 3

square = 9

9 % 9 => 9 Key.

List of data:

hash function,

list keys and their associated values in the table.

Hash Collision

10 locations
2 data items.

88, 38 = apply hash function of mod operation

key => 8, it means, collides.
Collision happens.

Alternative ??

Collision Resolution

- Closed Addressing
  - Chaining

m = 10

hash function = data % m

Input: 45(key 5), 88(key 8), 22(key 2), 32(key 2), 38 ..

Key	Data [Linked List]
0	
1	
2 address →	[22]→[32]→
3	
4	
5 address →	[45]→
6	
7	

8 address →	[88]→[38]→
9	

Search Operation

38 ??

key = 8,
I start traversing the list to find 38.

best case: O(1)
worse case: O(n)

- Open Addressing

Linear Probing

>> h(key, i) = (key + i) % m

>> i: 0 to m-1, that is the probe number.

Data => 88, 38, 33, 43

Locations => 0 to 9

88 → 8,
38 → 8, collision, 9

index = 0, index 1, ??, whether location 9 is free or not, Yes, I store 38 to 9.
probe => 1

33 → 3, free.

43 → 3, not free,

i = 1, location = 4

63 → 3, not free,

i = 1, not free,

i = 2, free, location 5.

Quadratic Probing

>> h(key, i) = (h(key)+i^2) % m

33 → 3, free.
43 → 3, not free,
   i = 1, i^2 = 1, location = 4
63 → 3, not free,
   i = 1, i^2 = 1, not free,
   i = 2, i^2 = 4,
      free, location 7.

Double Hashing

>> $h(key, i) = [h_1(key)+ih_2(key)] \% m$

Where,

$h_1(key) = key \% m,$

$h_2(key) = key \% m$

i = 1,

m = 10, number of locations,

43 => key % m + 1 x key % m

=> 43 % 10 + 1 x 43 % 10

=> 3 + 3

=> 6

Summary

- Good Hash Function
  >> Minimize Collision

  >> Uniform distribution of values
  all keys probe, equally distributed.

  >> Easy to calculate and involved suitable collision resolution technique.