Hashing

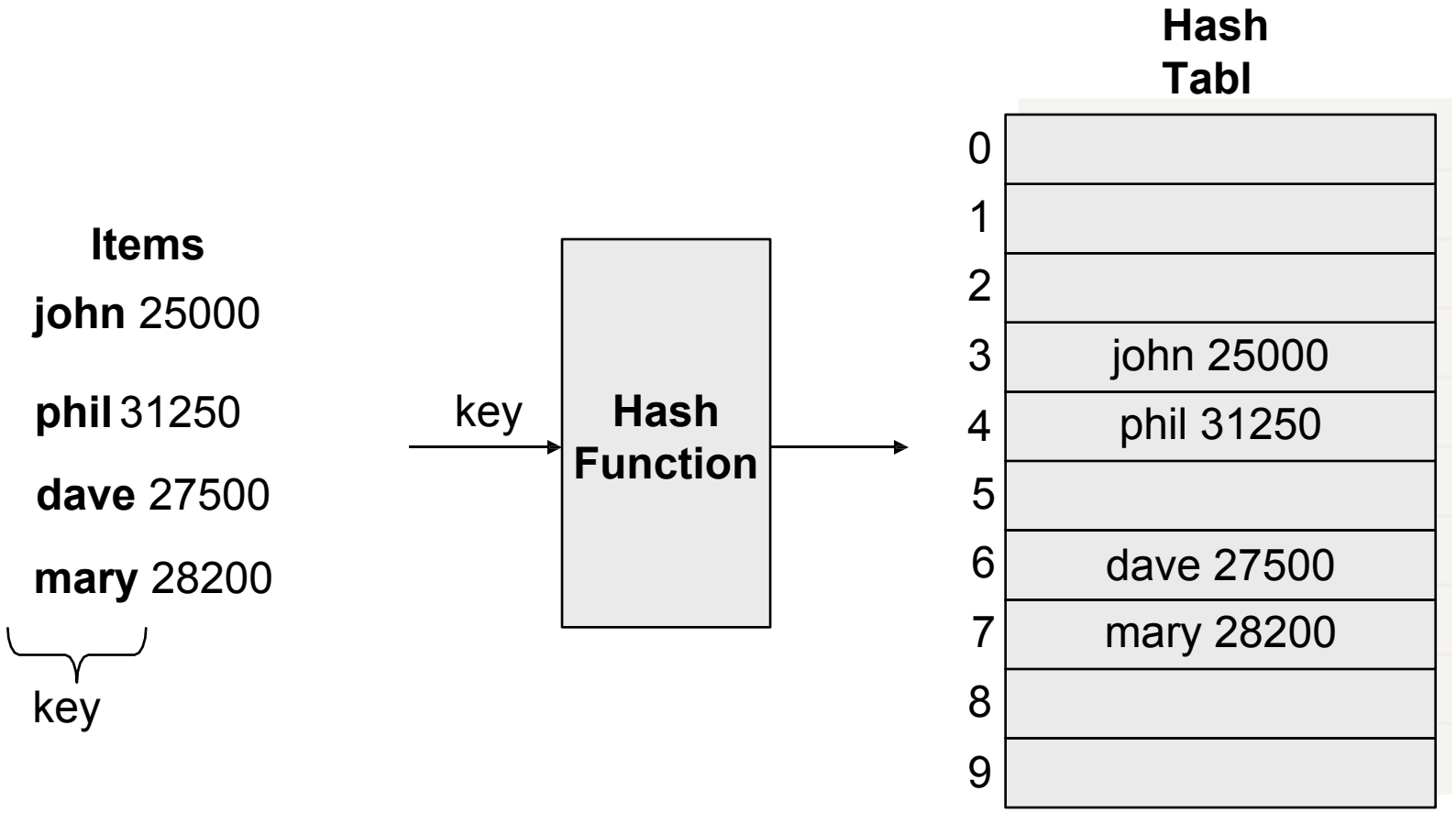
Overview

- Hashing
- Hash function
  - Characteristics of Hash function
- Insert, Update, Delete, and Search operations
- Collision Resolution
  - Separate chaining
  - Open Addressing

Hashing

- Hashing is a technique used for performing the search operation in constant average time (i.e. O(1))
- This data structure, however, is not efficient in operations that require any ordering informatio among the elements, such as findMin, findMax and printing the entire table in sorted order.

Working Principle



Hash Function

- The hash function:
  - must be simple to compute.
  - must distribute the keys evenly among the cells.
- If we know which keys will occur in advance we can write *perfect* hash functions, but we don't.

Hash function

Problems:

- Keys may not be numeric.
- Number of possible keys is much larger than the space available in table.
- Different keys may map into same location
  - Hash function is not one-to-one => collision.
  - If there are too many collisions, the performance of the hash table will suffer dramatically.

Hash Functions

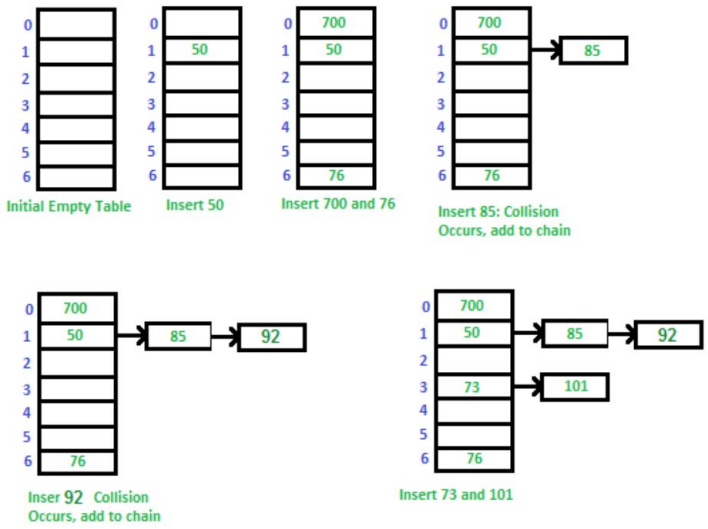
- If the input keys are integers then simply *Key* mod *TableSize* is a general strategy.
  - Unless key happens to have some undesirable properties. (e.g. all keys end in 0 and we use mod 10)
- If the keys are strings, hash function needs more care.
  - First convert it into a numeric value.

Collision Resolution

- If, when an element is inserted, it hashes to the same value as an already
  inserted element, then we have a collision and need to resolve it.
- There are several methods for dealing with this:
  - **Separate chaining**
  - **Open addressing**
    - Linear Probing (h(k) +i) mod Size_table
    - Quadratic Probing (h(k) +$c_1 i + c_2 i^2$) mod Size_table
    - Double Hashing

Separate chaining

Example: Let us consider a simple hash function as "**key mod 7**" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101



Open Addressing

1. Linear Probing:

In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

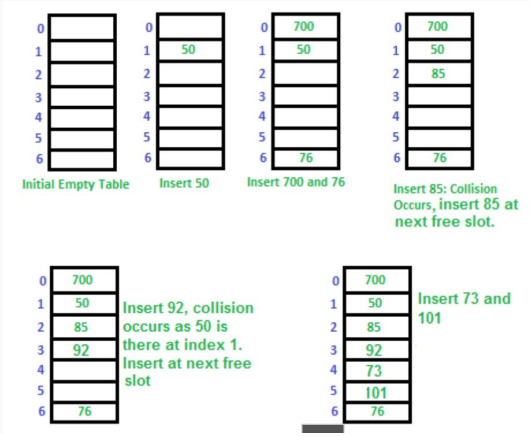
The function used for rehashing is as follows: rehash(key) = (n+1)%table-size.

*Let **hash(x)** be the slot index computed using a hash function and **S** be the table size*

If slot hash(x) % S is full, then we try (hash(x) + 1) % S
If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S
If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S
.....
.....

Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101,

which means hash(key)= key% S, here S=size of the table =7,indexed from 0 to 6.We can define the hash function as per our choice if we want to create a hash table,although it is fixed internally with a pre-defined formula.



2. Quadratic Probing

If you observe carefully, then you will understand that the interval between probes will increas proportionally to the hash value. Quadratic probing is a method with the help of which we can solve the problem of clustering that was discussed above. This method is also known as the **mid-square** method. In this method, we look for the **i^2'th** slot in the **i^{th}** iteration. We always start from the original hash location. If only the location is occupied then we check the other slots.

let hash(x) be the slot index computed using hash function.

*If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S*
*If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S*
*If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S*
.....
.....

Exercise

Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be $f(i) = i^2$. Insert = 22, 30, and 50.

Double Hashing

Double hashing is a collision resolution technique used in hash tables. It works by using two hash functions to compute two different hash values for a given key. The first hash function is used to compute the initial hash value, and the second hash function is used to compute the step size for the probing sequence.

Double hashing has the ability to have a low collision rate, as it uses two hash functions to compute the hash value and the step size. This means that the probability of a collision occurring is lower than in other collision resolution techniques such as linear probing or quadratic probing.

However, double hashing has a few drawbacks. First, it requires the use of two hash functions, which can increase the computational complexity of the insertion and search operations. Second, it requires a good choice of hash functions to achieve good performance. If the hash functions are not well-designed, the collision rate may still be high.

Double hashing can be done using :

(hash1(key) + i * hash2(key)) % TABLE_SIZE

Here hash1() and hash2() are hash functions and TABLE_SIZE is size of hash table.

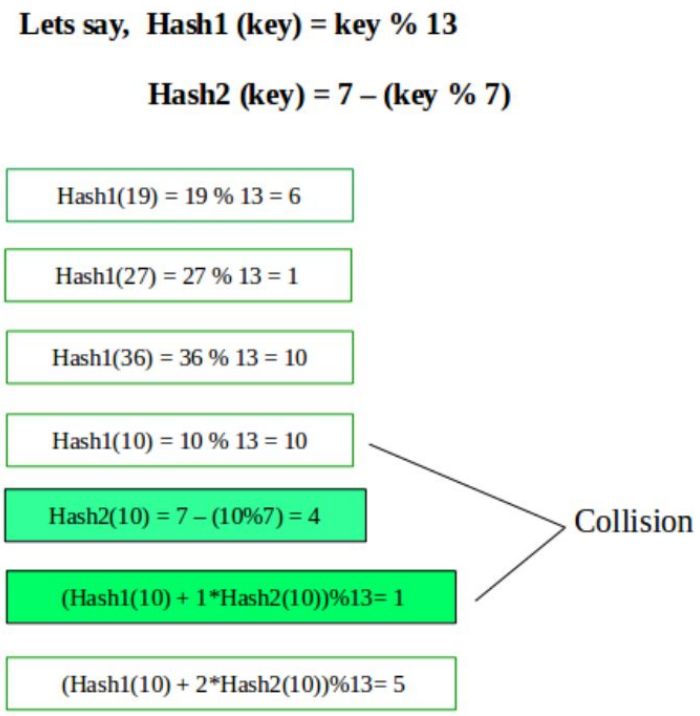
(We repeat by increasing i when collision occurs)

Method 1: First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.

A good second Hash function is:

- It must never evaluate to zero
- Just make sure that all cells can be probed



H(key)=key%7
Data: 984,376,32,776,49,453,231
Table size :10
Collision resolution technique: separate chaining, linear probing, quadratic probing

Collision Resolution

- Discussion

Summary

- In this lecture
  - The basic concept of hashing is covered along with it's collision resolution techniques.
  - It is concluded that choice of hash function is important to get the constant average time for search operation
  - This data structure is not good for functions like finding max, min, sorting etc.