fit@hcmus

Hash Table

Contents

- Hashing
- Hash Functions
- Resolving Collisions

Hashing

- Binary search tree retrieval have order $O(\log_2 n)$
- Need a different strategy to locate an item
- Consider a “magic box” as an address calculator
  - Place/retrieve item from that address in an array
  - Ideally to a unique number for each key

Hashing

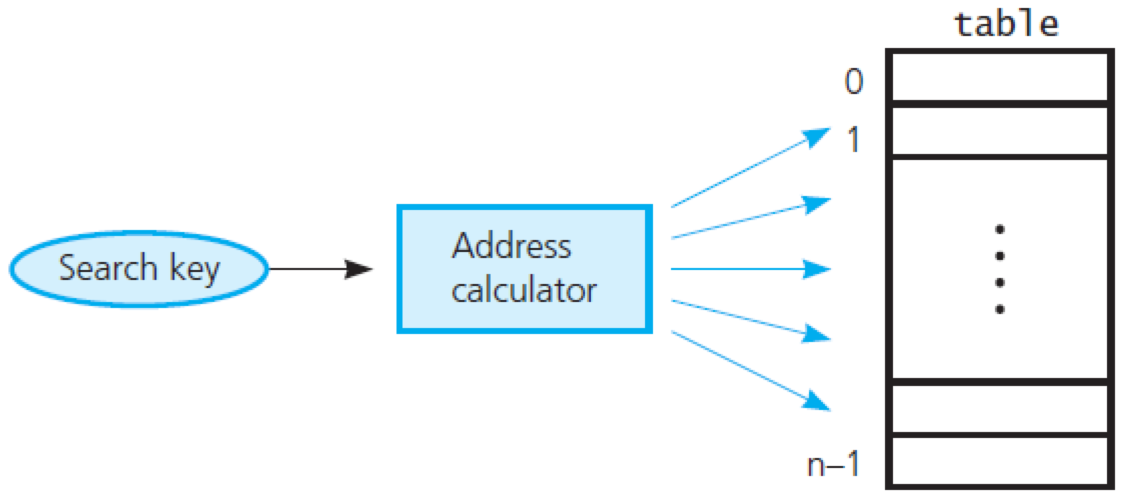
- Hashing is a technique to convert a range of key values into a range of indexes of an array.

- *Large* keys are converted into *small* keys by using **hash functions**.
- The values are then stored in a data structure called **hash table**.

Hashing

- Idea:
  - Distribute entries (key/value pairs) uniformly across an array.
  - Each element is assigned a key (converted key).
  - Using that key to access the element in O(1) time. (The hash function computes an index suggesting where an entry can be found or inserted.)

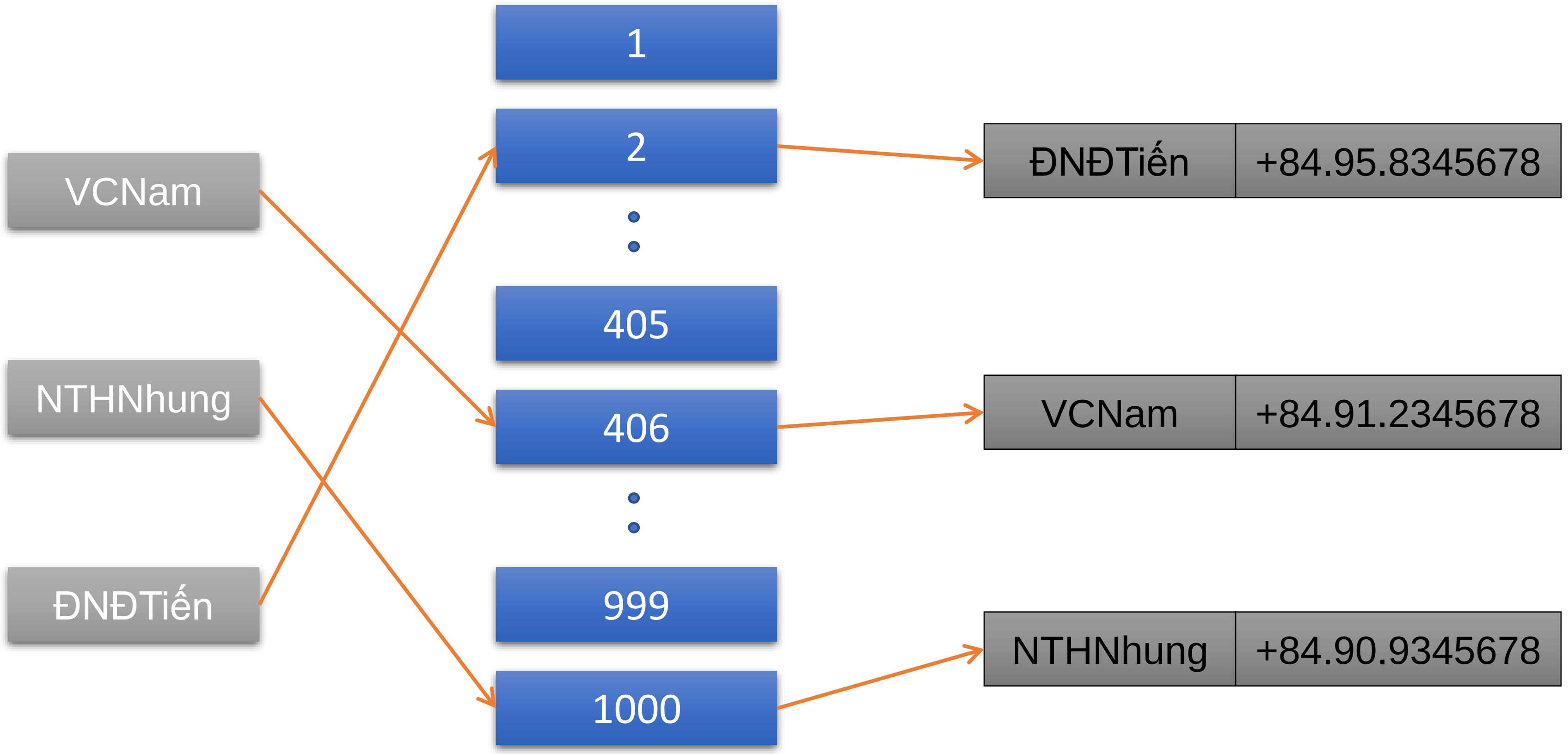
Hashing



Hash Table

- A hash table is a data structure that is used to store keys/value pairs.
- It uses a hash function to compute an index into an array in which an element will be inserted or searched.

Example



VCNam
NTHNhung
ĐNĐTiến
1
2
405
406
999
1000
ĐNĐTiến
+84.95.8345678
VCNam
+84.91.2345678
NTHNhung
+84.90.9345678

Hash Functions

- Hash function is any function that can be used to map/converts a key to an array index (integer value).

- The values returned by a hash function
  - hash values
  - hash codes
  - hash sums
  - hashes.

Some Hash Functions

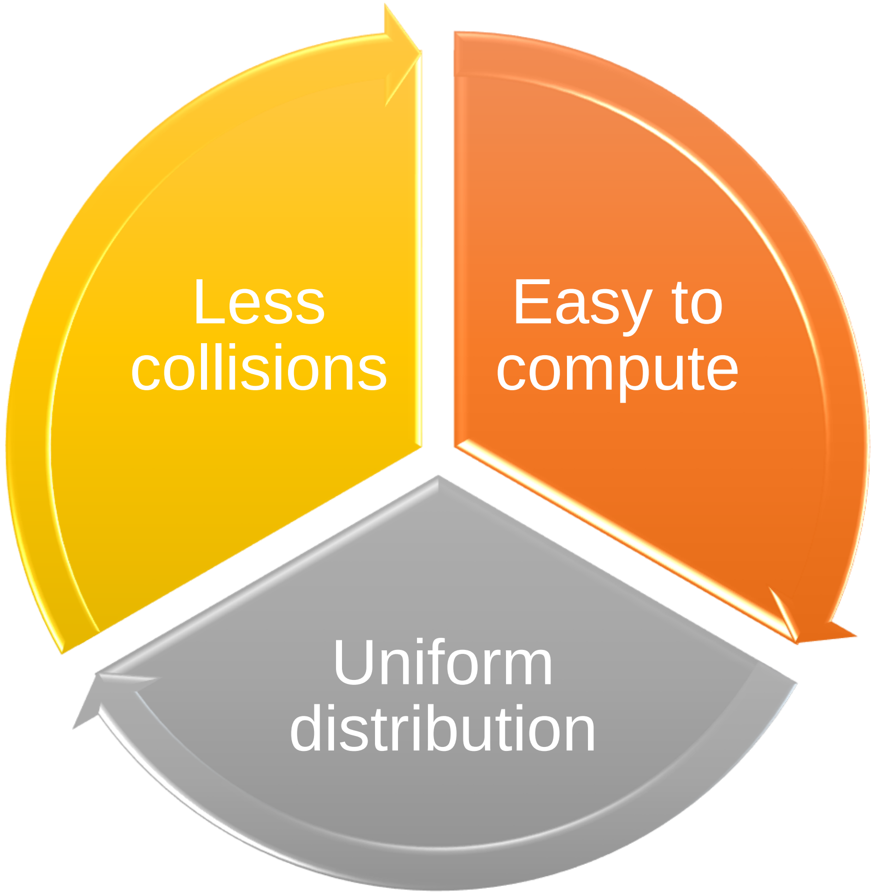
- Possible algorithms
  - Selecting digits
  - Folding
  - Modulo arithmetic
  - Converting a character string to an integer
    - Use ASCII values
    - Factor the results, Horner’s rule

Some Hash Functions

- Digit-selection:
  - Select some digits in the keys to create the hash value.
  - h(001**3**6482**5**) = 35
- Folding
  - h(001364825) = 0 + 0 + 1 + 3 + 6 + 4 + 8 + 2 + 5 = 29
  - h(**001**364**825**) = 001 + 364 + 825 = 1190
- Modulo arithmetic
  - h(Key) = Key mod 101
  - h(001364825) = 12

Good Hash Functions

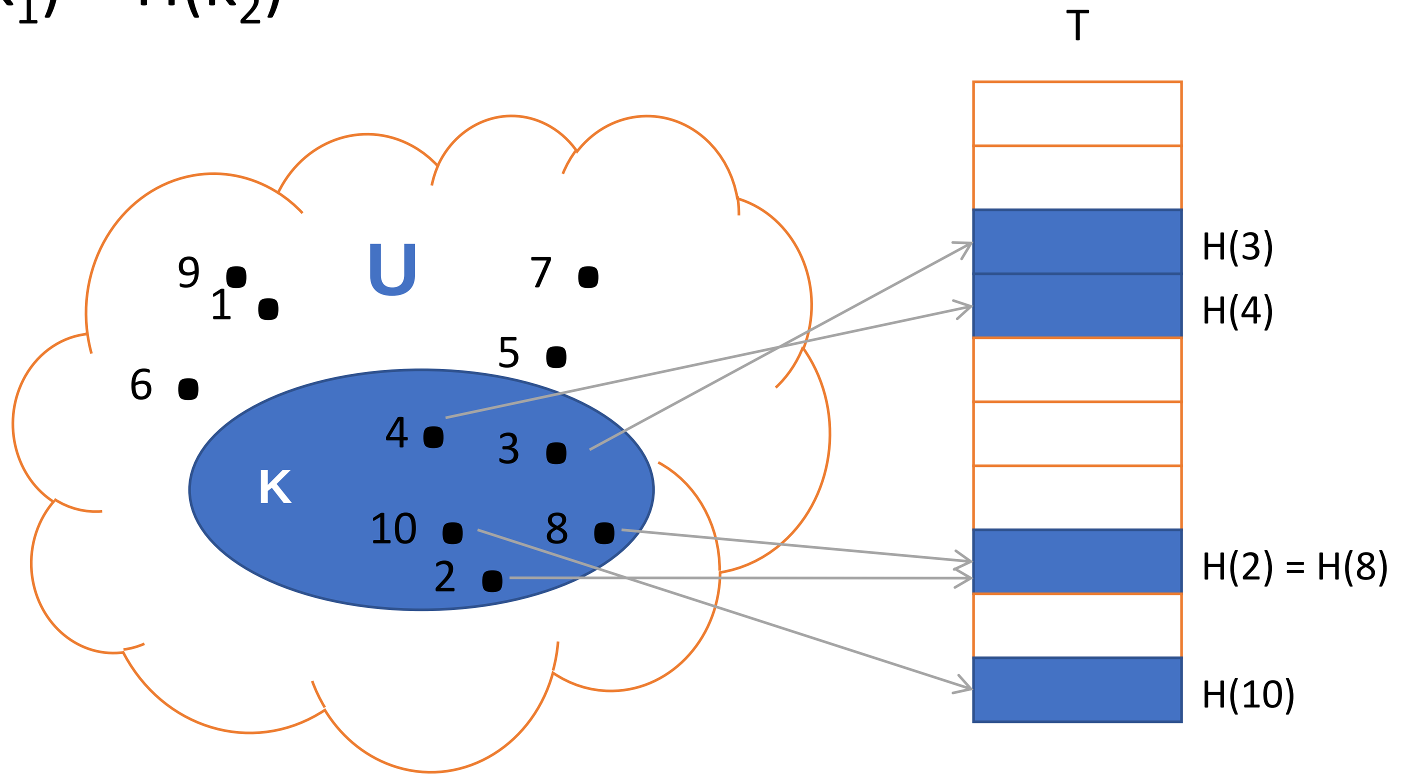
- Properties of good hash functions



Collisions

- $\exists k_1, k_2 \in K$:

  $k_1 \neq k_2, H(k_1) = H(k_2)$



Resolving Collisions

Resolving Collisions

- Separate Chaining (open hashing)
- Open Addressing (closed hashing)

Separate Chaining

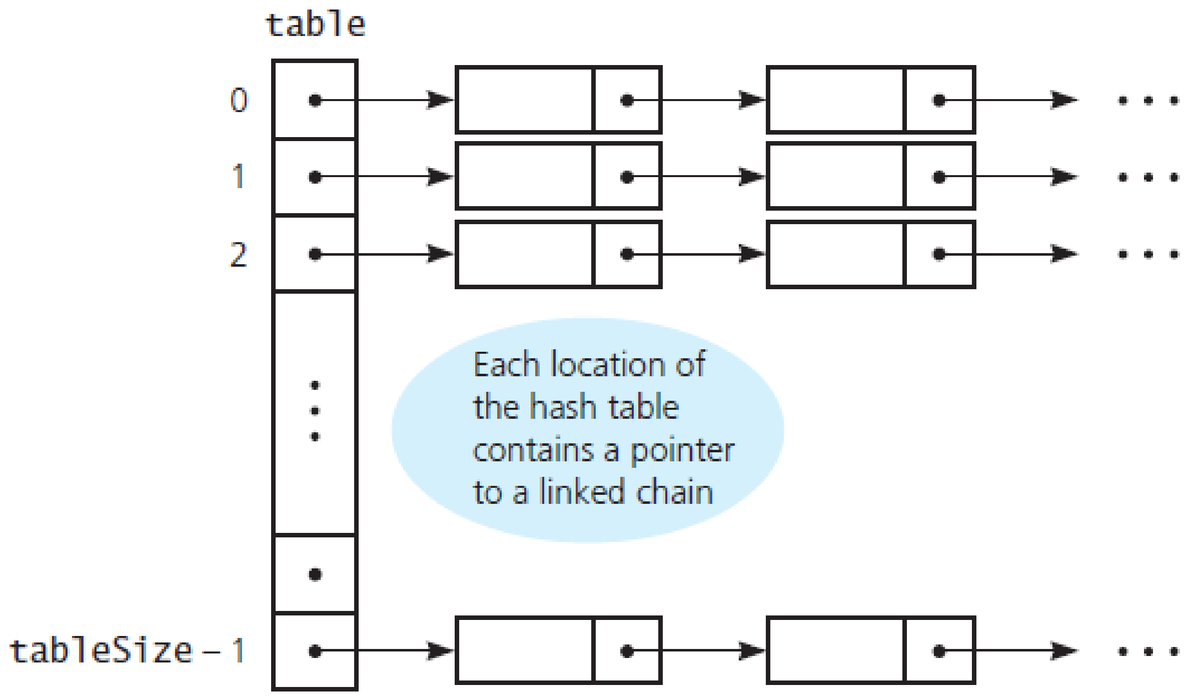
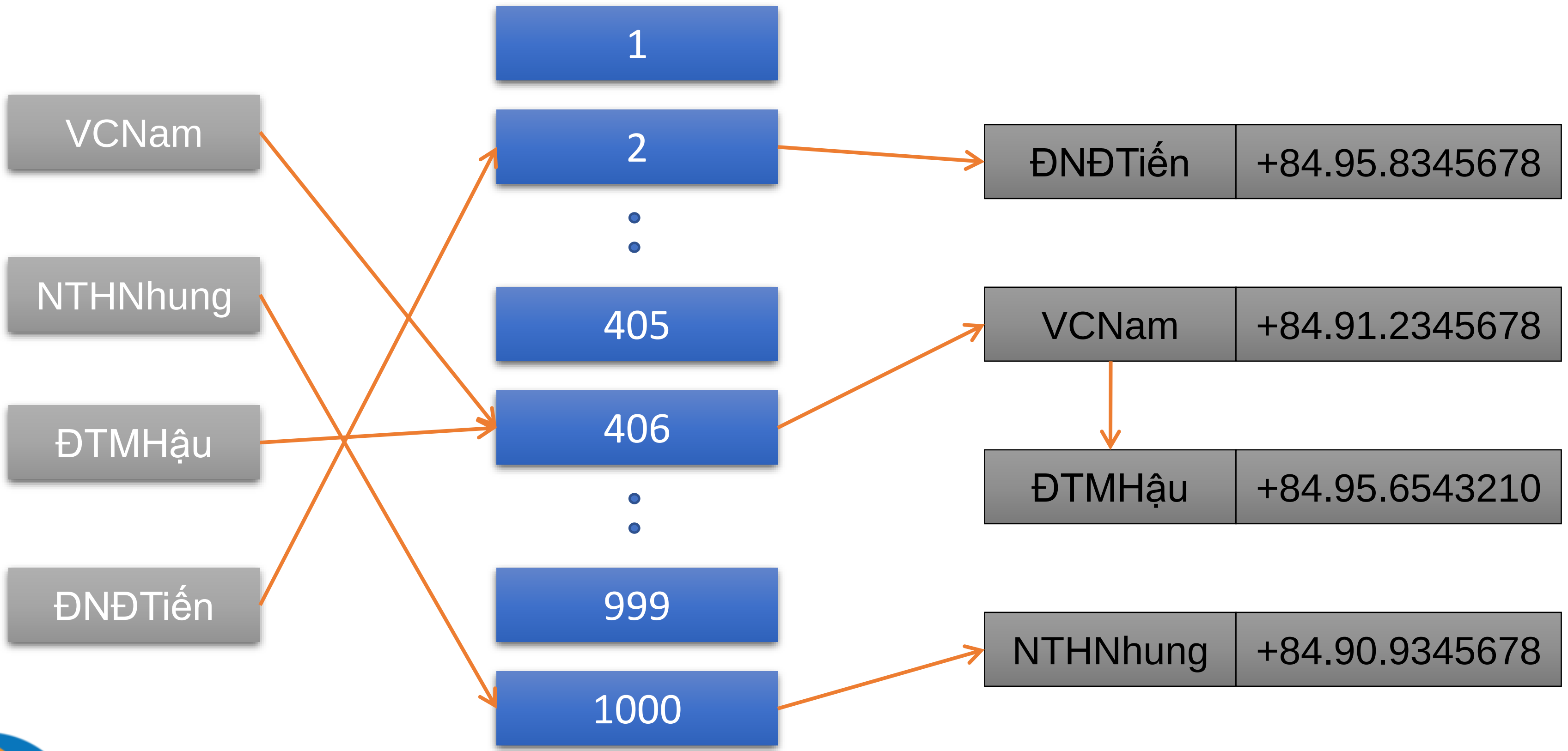


table
0
1
2
tableSize – 1
Each location of the hash table contains a pointer to a linked chain

Separate Chaining

- Each hash location can accommodate more than one item
- Each location is a “bucket” or an array itself
- Alternatively, design the hash table as an array of linked chains (“*separate chaining*”).

Separate Chaining



Open Addressing

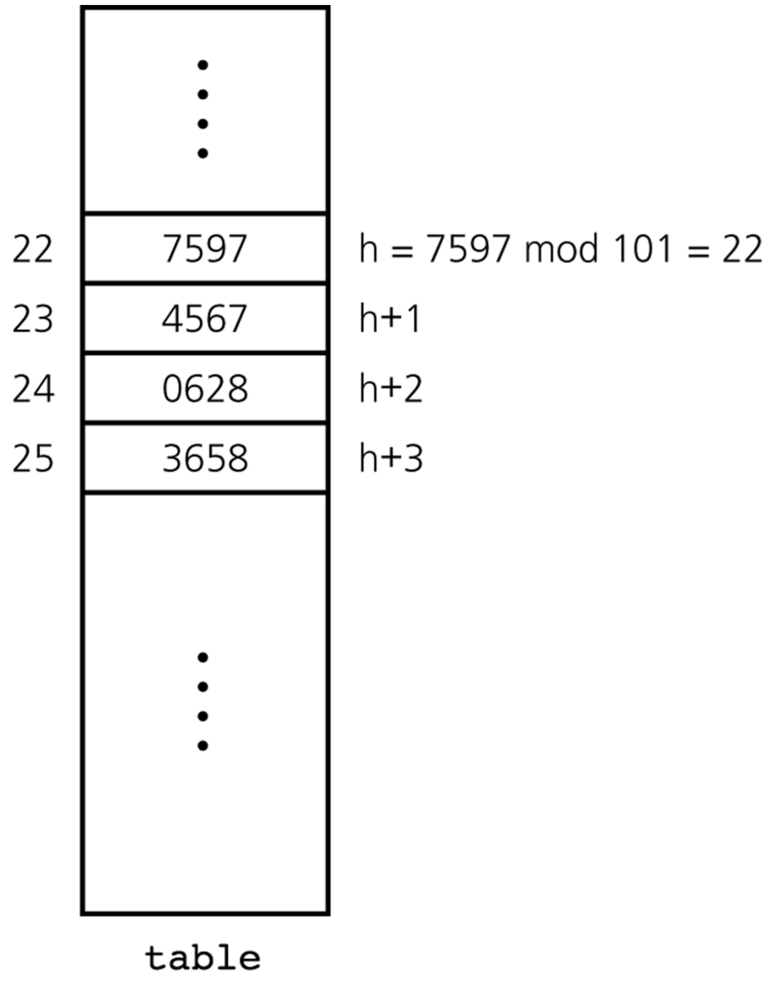
- Probe for another available location
- Some techniques:
    - Linear probing
    - Quadratic probing
    - Double hashing

Linear Probing

H(k, *step*) = (h(k) + *step*) mod M

step = 0, 1,…

M: size of hash table

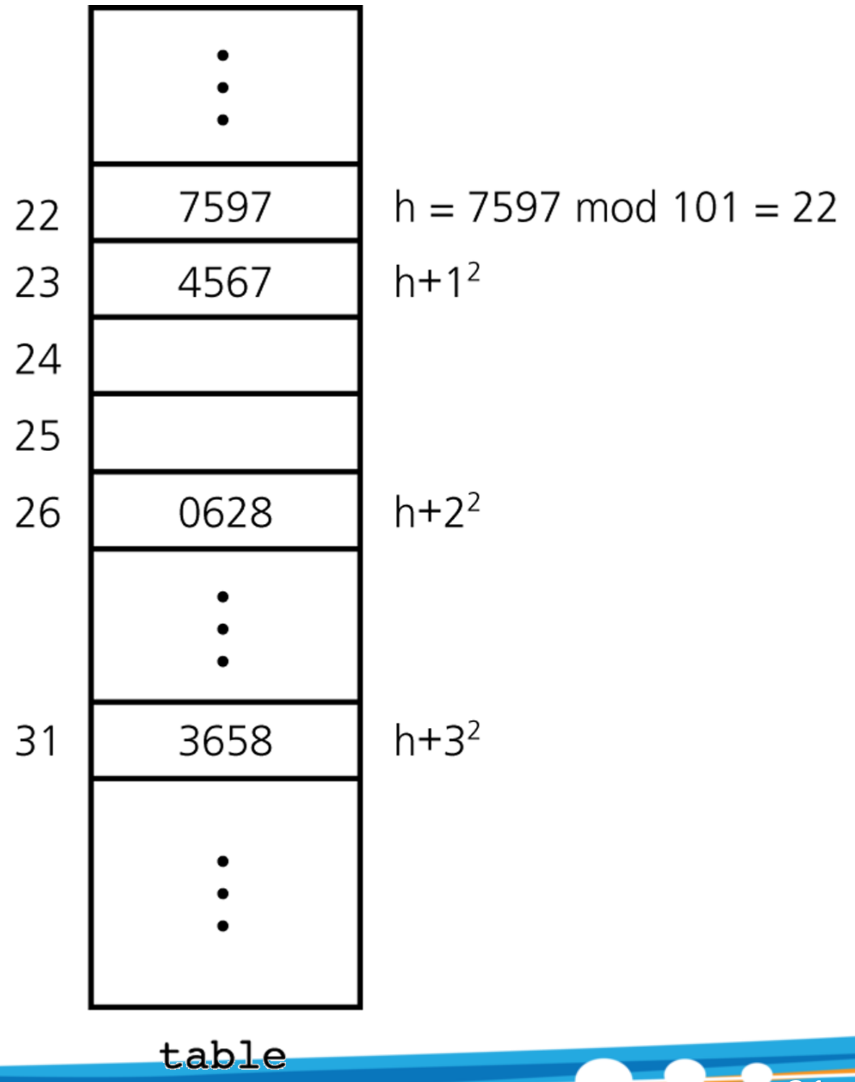


Quadratic Probing

$H(k, \textit{step}) = (h(k) + \textit{step}^2) \bmod M$

step = 0, 1,…

M: size of table

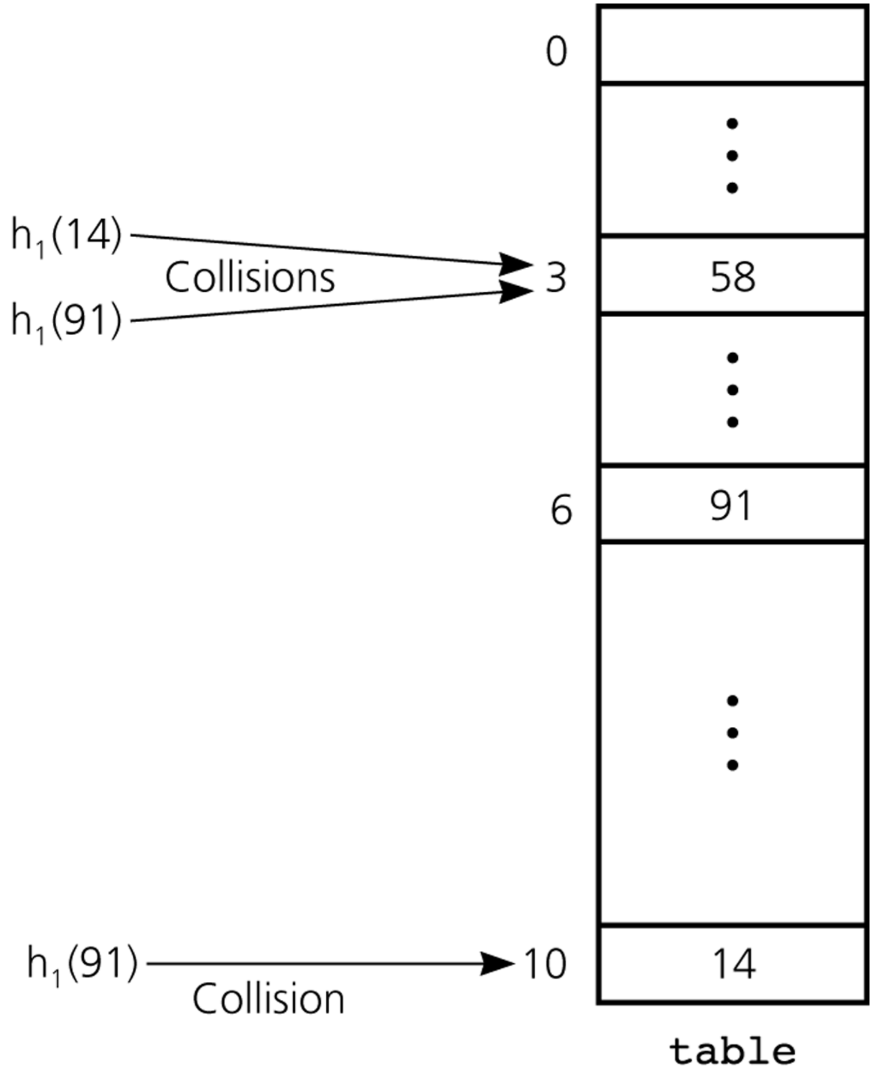


Double Hashing

$H(k, \textit{step}) = (h(k) + \textit{step} * h_2(k)) \bmod M$

- *step* = 0, 1,…
- M: size of hash table
- $h_2(k)$: second hash function

$h(key) = key \bmod 11$
$h_2(key) = 7 - (key \bmod 7)$



Separate Chaining

- Advantages:
  - Simple to implement.
  - Hash table never fills up, we can always add more elements to the chain.
  - Less sensitive to the hash function or load factors.
  - It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Separate Chaining

- Disadvantages:
  - Cache performance of chaining is not good as keys are stored using a linked list. Wastage of space (Some parts of hash table are never used)
  - If the chain becomes long, then search time can become O(n) in the worst case.
  - Uses extra space for links.

Open Addressing

- Removal requires specify state of an item
  - Occupied, emptied, removed
- Clustering is a problem
- Double hashing can reduce clustering

Open Addressing

- Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute.
- Quadratic probing lies between the two in terms of cache performance and clustering.
- Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

The Efficiency of Hashing

The Efficiency of Hashing

- Efficiency of hashing involves the load factor alpha (α)

$$\alpha = \frac{\textit{Current number of table items}}{\textit{tableSize}}$$

The Efficiency of Hashing

- Linear probing – average value for α

$$\frac{1}{2}\left[1 + \frac{1}{1 - \alpha}\right]$$ for a successful search, and

$$\frac{1}{2}\left[1 + \frac{1}{(1 - \alpha)^2}\right]$$ for an unsuccessful search

The Efficiency of Hashing

- Quadratic probing and double hashing – efficiency for given α

$$\frac{-\log_e(1-\alpha)}{\alpha}$$ for a successful search, and

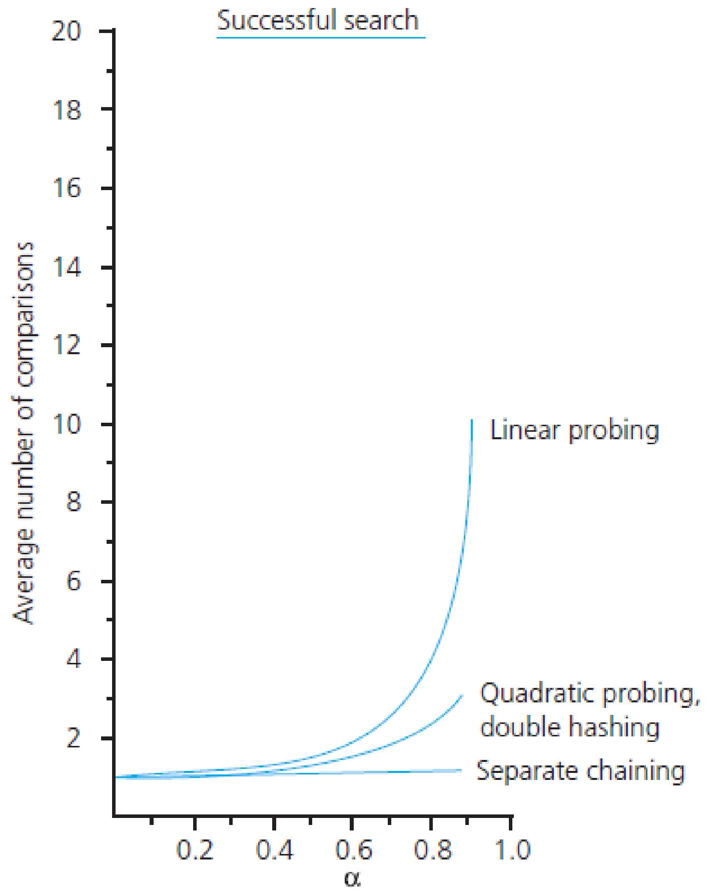
$$\frac{1}{1-\alpha}$$ for an unsuccessful search

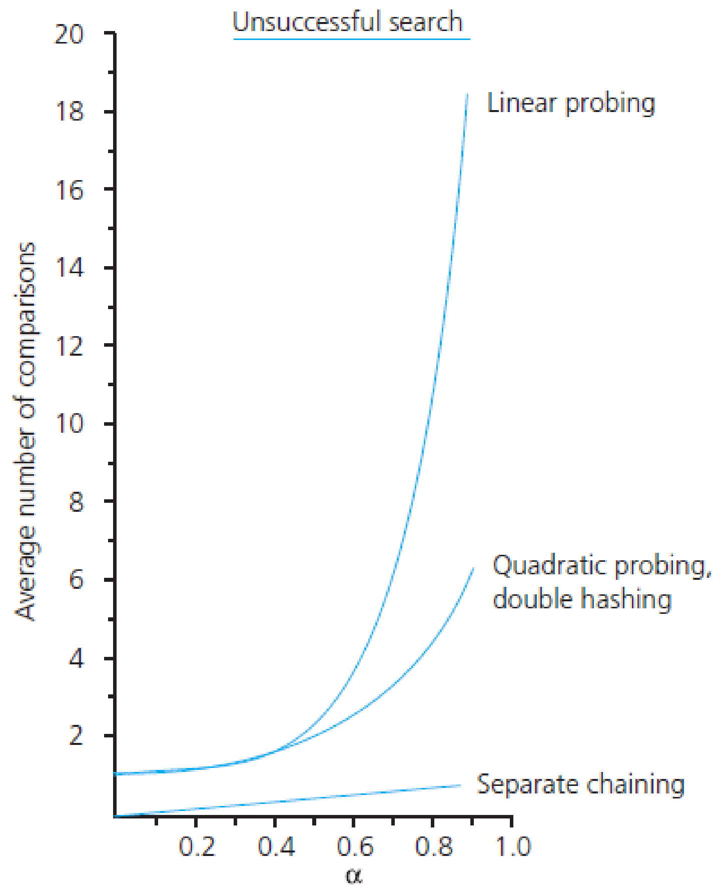
The Efficiency of Hashing

- Separate chaining – efficiency for given α

$1 + \frac{\alpha}{2}$	for a successful search, and
α	for an unsuccessful search

The Efficiency of Hashing





Maintaining Hashing Performance

- Collisions and their resolution typically cause the load factor α to increase
- To maintain efficiency, restrict the size of α
  - $\alpha \leq 0.5$ for open addressing
  - $\alpha \leq 1.0$ for separate chaining
- If load factor exceeds these limits
  - Increase size of hash table
  - Rehash with new hashing function

Exercise

Given a hash table with m = 13 entries and the hash function

`h(key) = key mod m`

Insert the keys **`{10, 22, 31, 4, 15, 28, 17, 88, 59}`** in the given order (from left to right) to the hash table. If there is a collision, use each of the following open addressing resolving methods:

A. Linear probing

B. Quadratic probing

C. Double hashing with `h2(key) = (key mod 7) + 1`

Questions and Answers