Hash Tables

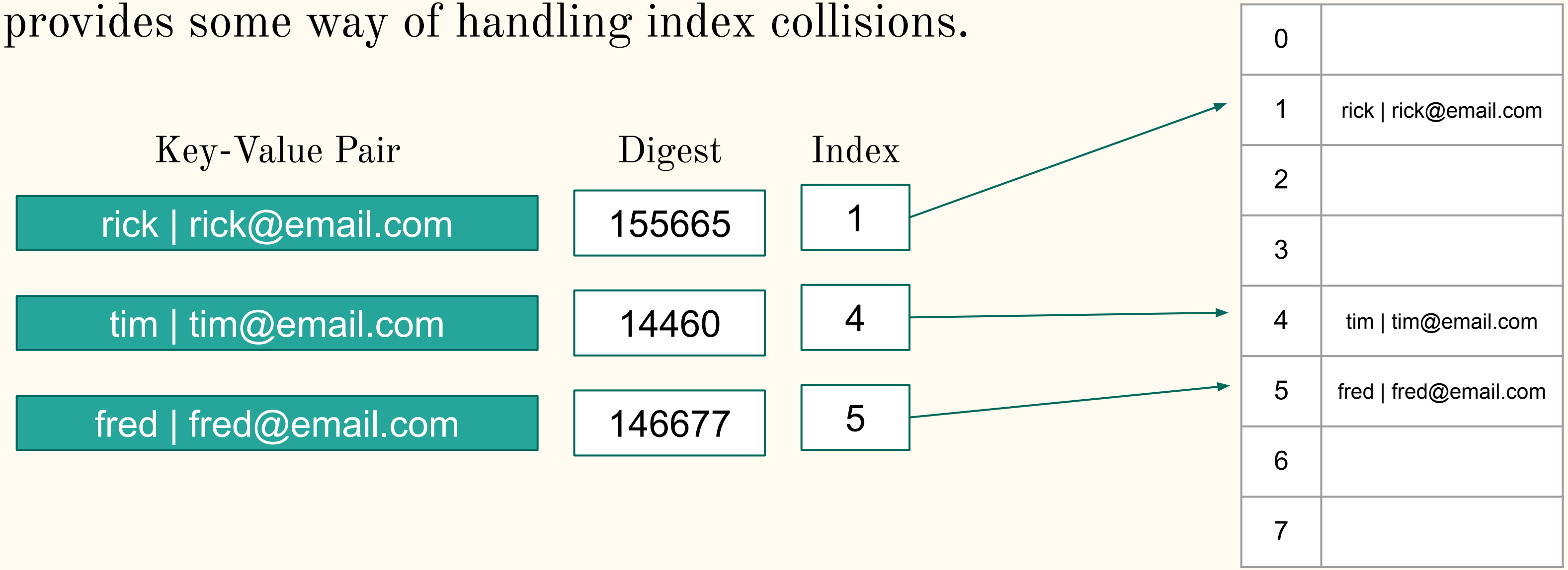
Denis Espino

Introduction

- Hash tables are a collection of key-value pairs.
- They are an implementation of the *associative arrays / dictionaries* ADT.
- *Abstract data types* are objects whose behavior is defined by a set of values and a set of operations.
- Associative arrays are an ADT that stores a collection of key-value pairs such that each possible key appears at most once in the collection.
- It supports the insert, lookup, and remove operations.

Hash Table

- A **hash table** is an implementation of the *associative array* abstract data type that uses an array of *buckets* to store key-value pairs and a *hash function* to compute an index which the value can be inserted/found. A hash table also provides some way of handling index collisions.



Key-Value Pair	Digest	Index
rick | rick@email.com	155665	1
tim | tim@email.com	14460	4
fred | fred@email.com	146677	5

0	
1	rick | rick@email.com
2	
3	
4	tim | tim@email.com
5	fred | fred@email.com
6	
7	

Hash Function

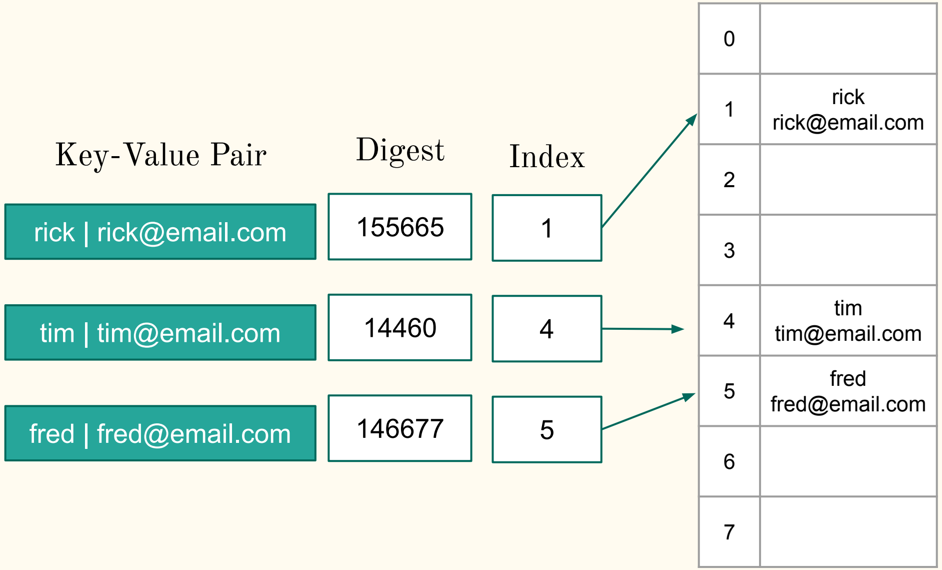
- A hash function maps the set of keys to array indices within the table.
- A hash function should be uniformly distributed to decrease the number of collisions

- The most common hashing scheme is “Hashing by division”
  $h(x) = M \bmod m$

Hashing Strings

```
function hashString(string, size) {
  let result = 0;
  for (let i = 0; i < string.length; i++)
    result +=
      string.charCodeAt(i) *
      Math.pow(31, i);
  return result % size;
}
```

$$\begin{aligned}\operatorname{hash}(s) &= s[0] + s[1] \cdot p + s[2] \cdot p^2 + \ldots + s[n-1] \cdot p^{n-1} \mod m \\ &= \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m,\end{aligned}$$



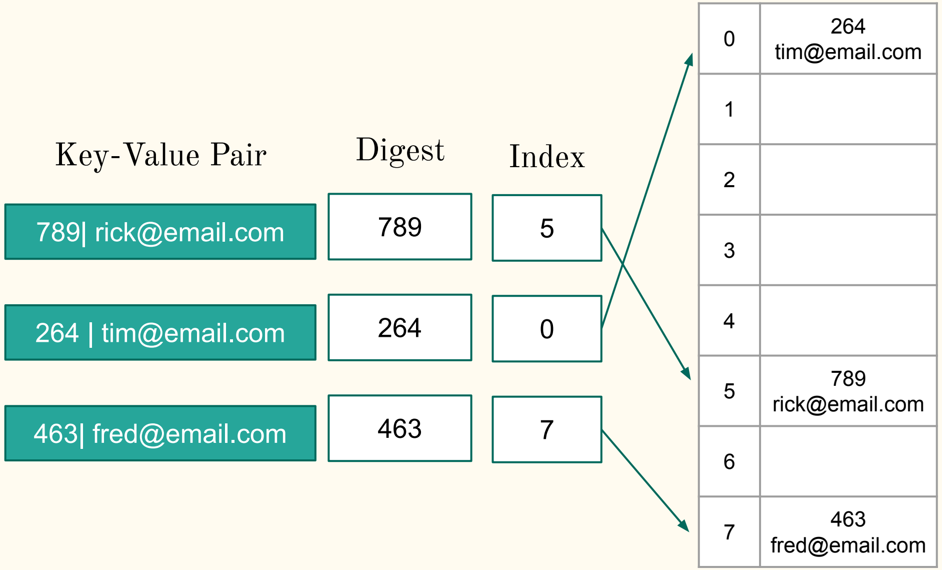
Key-Value Pair	Digest	Index
rick | rick@email.com	155665	1
tim | tim@email.com	14460	4
fred | fred@email.com	146677	5

0	
1	rick rick@email.com
2	
3	
4	tim tim@email.com
5	fred fred@email.com
6	
7	

Hashing Numbers

```
function hashInteger(integer, size) {
  return integer % size;
}
```

$$h(x) = M \bmod m$$

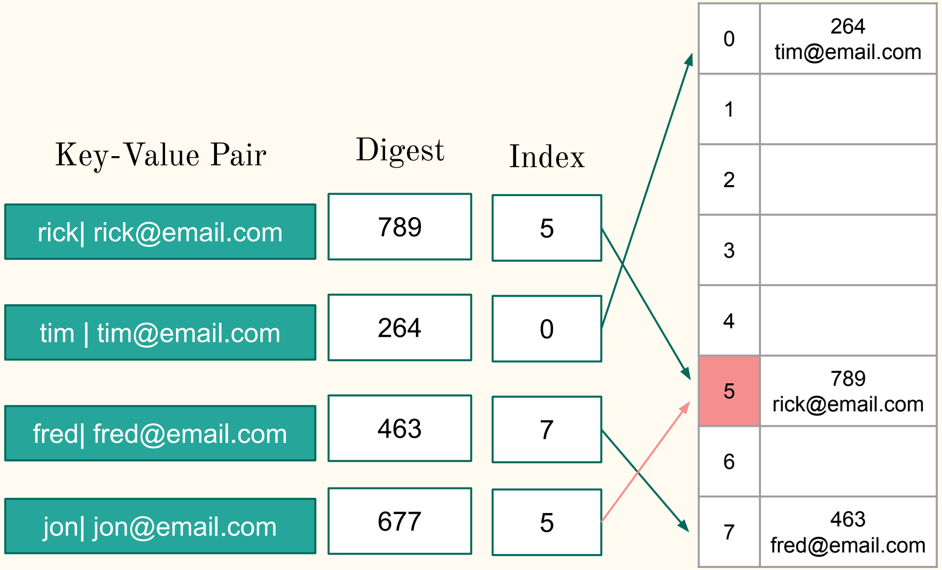


Key-Value Pair	Digest	Index
789| rick@email.com	789	5
264 | tim@email.com	264	0
463| fred@email.com	463	7

0	264 tim@email.com
1	
2	
3	
4	
5	789 rick@email.com
6	
7	463 fred@email.com

Collision resolution

- Because we are mapping a large set of keys into a relative smaller set of array indexes it is possible that two keys could hash to the same index. This is called a collision.

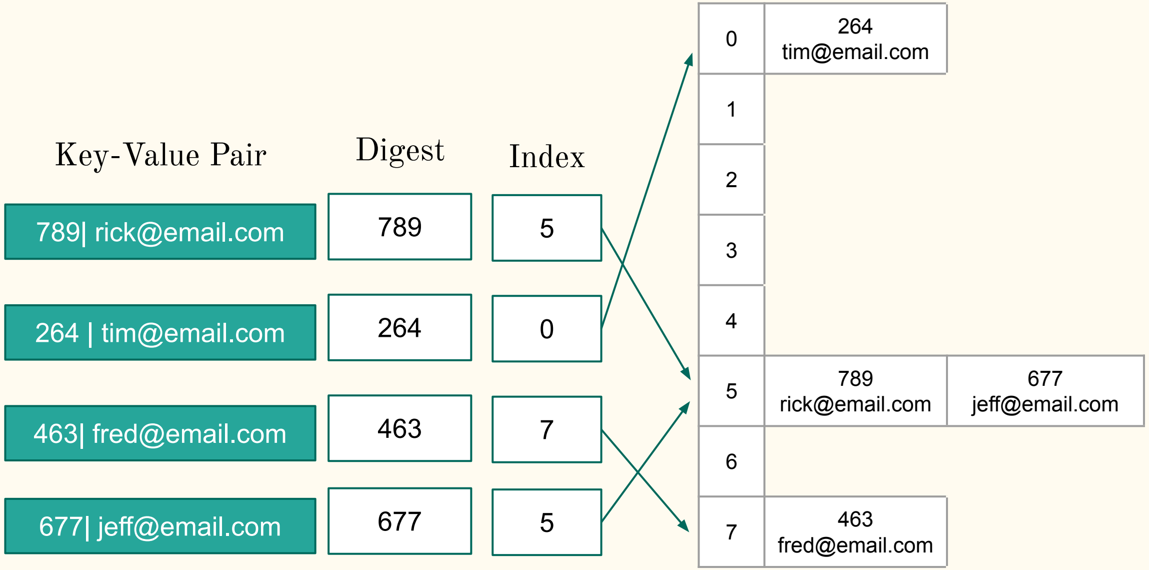


Key-Value Pair	Digest	Index
rick| rick@email.com	789	5
tim | tim@email.com	264	0
fred| fred@email.com	463	7
jon| jon@email.com	677	5

0	264 tim@email.com
1	
2	
3	
4	
5	789 rick@email.com
6	
7	463 fred@email.com

Chaining

This involves building a linked list of key-value pairs for each search array index. The collided items are chained together through a single linked list.

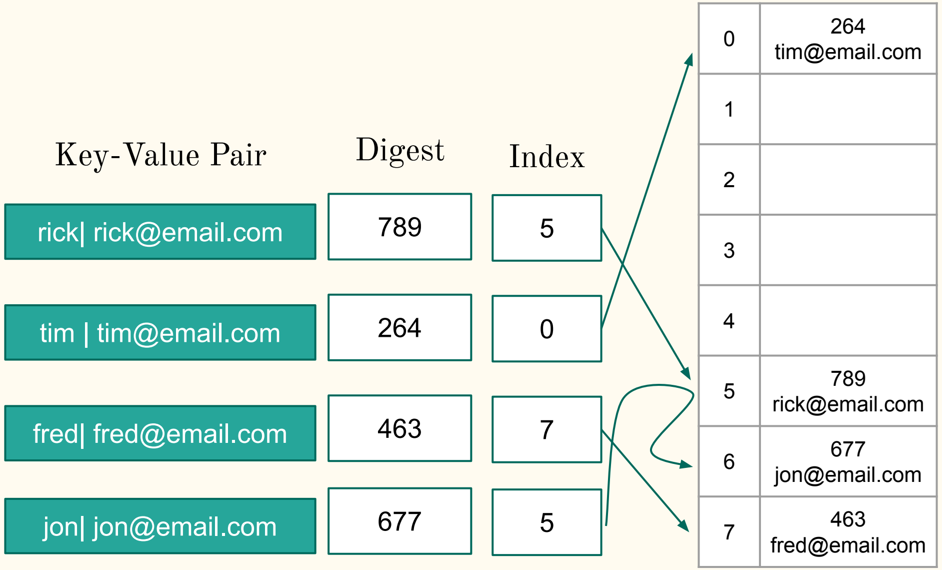


Key-Value Pair	Digest	Index
789| rick@email.com	789	5
264 | tim@email.com	264	0
463| fred@email.com	463	7
677| jeff@email.com	677	5

0	264 tim@email.com	
1		
2		
3		
4		
5	789 rick@email.com	677 jeff@email.com
6		
7	463 fred@email.com	

Open Addressing

Every entry is stored in the bucket array itself and the hash resolution is done through probing each slot for an unused space.



Key-Value Pair	Digest	Index
rick| rick@email.com	789	5
tim | tim@email.com	264	0
fred| fred@email.com	463	7
jon| jon@email.com	677	5

0	264 tim@email.com
1	
2	
3	
4	
5	789 rick@email.com
6	677 jon@email.com
7	463 fred@email.com

Probe Sequences

- Linear probing: in which the interval between probes is fixed
  - h(x), h(x)+1, h(x)+2, h(x)+3 ...
- Quadratic probing: in which the interval between probes is increased by adding the successive outputs of a quadratic polynomial
  - h(x), h(x)+1, h(x)+4, h(x)+9 ...
- Double hashing: in which the interval between probes is computed by a secondary hash function
  - h(x), h(x)+h_2(x), h(x)+2*h_2(x), h(x)+3*h_2(x), ...

Growing / Shrinking

- Repeated insertions cause the number of entries in the hash table to grow which increases the load factor.
- Load factor = entries in the hash table / hash table size
- To maintain performance a hash table is dynamically resized and the items are rehashed into the buckets of the new hash table.
- Rehashing should occur when the load factor reaches 0.6 to 0.75

Runtime

	Average	Worst Case
Space	$\Theta(n)$	$O(n)$
Search	$\Theta(1)$	$O(n)$
Insert	$\Theta(1)$	$O(n)$
Delete	$\Theta(1)$	$O(n)$

Resources

- Articles
  - https://en.wikipedia.org/wiki/Hash_table
  - https://en.wikipedia.org/wiki/Associative_array
  - https://cp-algorithms.com/string/string-hashing.html
- Videos
  - https://www.youtube.com/watch?v=knV86FlSXJ8&ab_channel=MichaelSambol
  - https://www.youtube.com/watch?v=FsfRsGFHuv4&ab_channel=BroCode
  - https://www.youtube.com/watch?v=7eLDTtbzX4M&ab_channel=WilliamFiset
- Visualizer
  - https://visualgo.net/en/hashtable