CSC 225 - Summer 2019

Hashing II

Bill Bird

Department of Computer Science
University of Victoria

July 5, 2019

Load Factor

Some degree of clustering is inevitable with any hashing scheme, especially when the table size is not much larger than the number of entries.

The **load factor** of a hash table of size M containing n keys is

$$\alpha = \frac{n}{M}.$$

Rule of thumb: Choose the table size to keep $\alpha \leq 0.6$.

Uniform Hashing

An ideal hash function should distribute keys **uniformly** and **independently** among table indices. There should be **no discernable correlation** between a key k and its hash value $h(k)$. In other words, for any set of keys $k_1, k_2, \ldots, k_n$, the set of hash values $h(k_1), h(k_2), \ldots, h(k_n)$ should be nearly indistinguishable from a set of random values.

As a result, an ideal hash function is identical to a **pseudorandom number generator**. Pseudorandom number generators and ideal hash functions are covered in CSC 429 (Cryptography).

Uniform Hashing

The techniques to prove that a hash function distributes keys uniformly and independently are beyond the level of this course.

Instead, we will prove that hash tables have $\Theta(1)$ FIND and INSERT operations under the assumption that hash values are uniform and independent.

Expected Time: Chaining

Theorem: Under the assumption of uniformity, the expected size of the linked list in each index of a hash table of size M storing n keys is

$$n/M$$

which is equal to the load factor α.

Advice (1)

Choosing a high quality hash function for a particular application usually requires some knowledge of the expected input data. Simulations using sample data can be helpful in choosing a good function for particular inputs.

For a hash table with n **integer keys**, the following rules of thumb often produce good results:

- Choose the table size to be a prime number p such that $1.5n < p$.
- Choose values $a, b > 0$ and use

$$h(k) = ak + b \bmod p$$

as the hash function.

Advice (2)

When the input values are not integers, the easiest approach is often to find a mapping from the input data to integers, then follow the advice for integer hash tables.

If the input data is structured (e.g. strings or a record type), it may be advantageous to choose a hash function which incorporates positional information, to mitigate collisions caused by anagrams.

Advice (3)

For a compound data type $k = (x_1, x_2, \ldots, x_q)$ (where each x_i is some data element, such as a character in the string), functions like

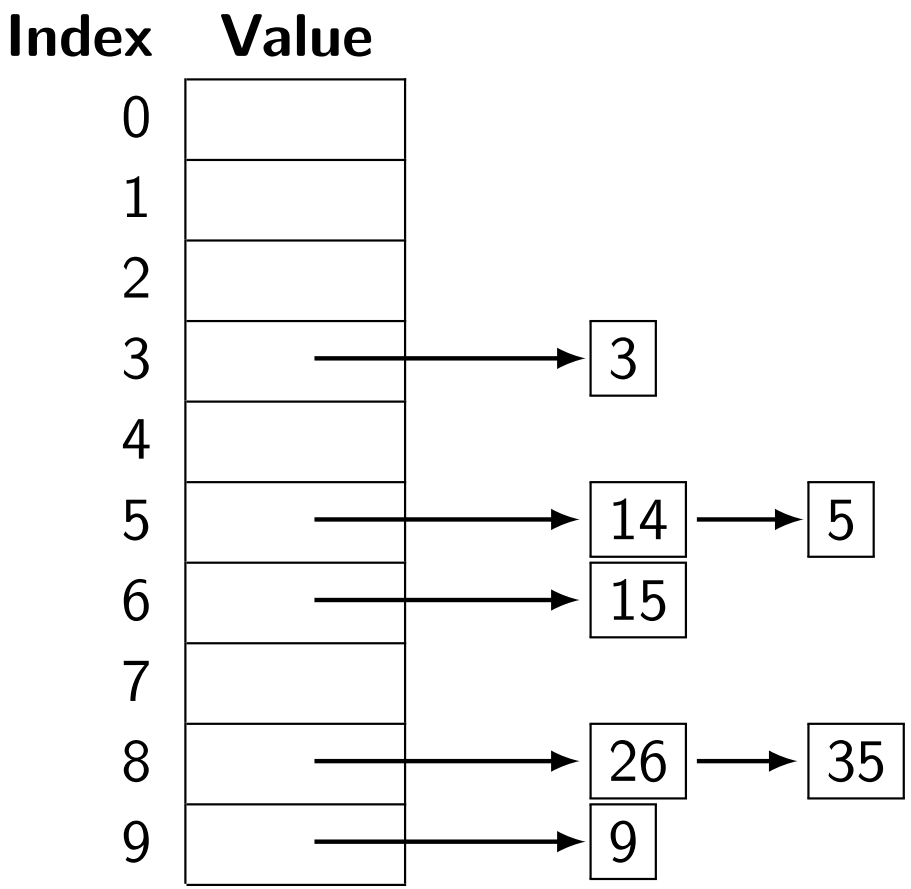
$$h(k) = (x_1^1 + x_2^2 + \ldots x_k^k) \bmod M$$

and

$$h(k) = (3^1 x_1 + 3^2 x_2 + 3^3 x_3 + \ldots 3^k x_k) \bmod M$$

incorporate the position of each element into the result. Both approaches come at the cost of extra computation time.

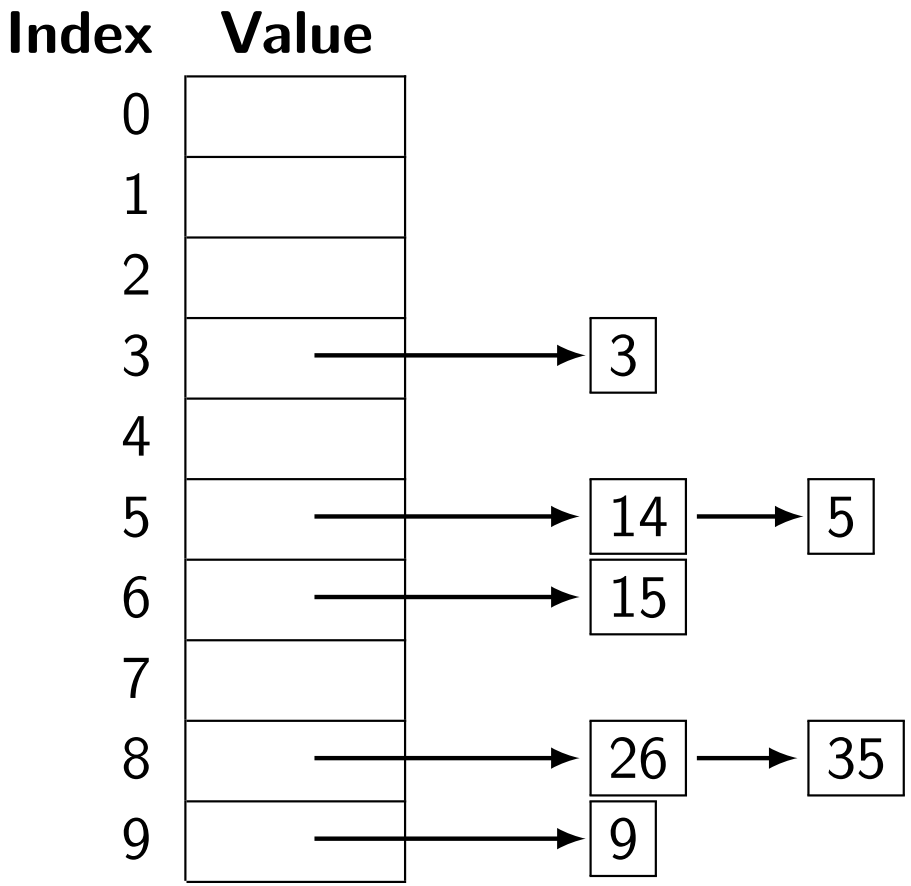
Chaining (1)



$$h(k) = (k + \lfloor k/10 \rfloor) \bmod 10$$

- Chaining is a simple and effective collision resolution strategy.

Chaining (2)



$$h(k) = (k + \lfloor k/10 \rfloor) \bmod 10$$

- However, the linked lists increase overhead, particularly for cache performance.

Open Addressing (1)

Open Addressing collision resolution schemes store every key in a table index, using a **probing scheme** to find an available index when a collision occurs.

Linear probing starts at the hash value $h(k)$, then checks successive indices until an empty space is found. The **probe sequence** is

$$h(k), \quad h(k)+1, \quad h(k)+2, \quad h(k)+3, \ldots$$

where all values are taken modulo the table size M. The index checked at step i is

$$(h(k)+i) \bmod M$$

Open Addressing (2)

Quadratic probing uses the probe sequence

$$h(k) + 0^2, \quad h(k) + 1^2, \quad h(k) + 2^2, \quad h(k) + 3^2, \ldots$$

modulo M. The index checked at step i is

$$(h(k) + i^2) \bmod M$$

Double hashing uses two hash functions $h_1(k)$ and $h_2(k)$, where $h_2(k) \neq 0$ for any k. The probe sequence is

$$h_1(k) + h_2(k), \quad h_1(k) + 2h_2(k), \quad h_1(k) + 3h_2(k), \ldots$$

and the index checked at step i is

$$(h_1(k) + ih_2(k)) \bmod M$$

Linear Probing (1)

Index	Value
0	
1	
2	
3	
4	
5	
6	

$$h(k) = 2k + 4 \bmod 7$$

- **Exercise**: Insert the sequence

$$1, 3, 4, 10, 18$$

into the hash table above using linear probing to resolve collisions.

Linear Probing (2)

Index	Value
0	
1	
2	
3	
4	
5	
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(1) = 6$
- Since index 6 is empty, the key is inserted.

Linear Probing (3)

Index	Value
0	
1	
2	
3	3
4	
5	
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(3) = 3$
- Since index 3 is empty, the key is inserted.

Linear Probing (4)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(4) = 5$
- Since index 5 is empty, the key is inserted.

Linear Probing (5)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(10) = 3$
- Index 3 is occupied, so the following index is probed.

Linear Probing (6)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(10) = 3$
- Since index 4 is empty, the key is inserted.

Linear Probing (7)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) = 5$
- Index 5 is occupied.

Linear Probing (8)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) = 5$
- Index 6 is also occupied.

Linear Probing (9)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) = 5$
- Index 0 is empty, so the key is inserted.

Linear Probing (10)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- **Exercise**: Search for the key 17 in the hash table above.
- To search for a key, probe successive indices starting at the initial hash value until the key is found (a successful search) or an empty space is reached (an unsuccessful search).

Linear Probing (11)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Linear Probing (12)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Linear Probing (13)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Linear Probing (14)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Linear Probing (15)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Linear Probing (16)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$
- Since an empty space was reached before the key was found, it is not in the table and the search was unsuccessful.

Linear Probing (17)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- Due to the clustering of the keys in the table, it was necessary to search through every key in the table (even though the hash function did not exhibit much clustering) during the unsuccessful search.

Linear Probing (18)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- One disadvantage of linear probing is that if a region of the table becomes clustered, every index in that region will suffer from long probing sequences.
- One way to alleviate this problem is to make the 'step size' of a probing sequence vary in some way based on the starting point.

Linear Probing (19)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- **Exercise**: Delete the key 1 from the hash table above.

Linear Probing (20)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- First, find the key in the table.
- $h(1) = 6$, and the key is found without any additional probes.

Linear Probing (21)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	

$$h(k) = 2k + 4 \bmod 7$$

- It is tempting to simply clear index 6, but creating an empty space at index 6 will break the FIND algorithm.
- Consider the result of FIND(18) (given that $h(18) = 5$) if index 6 is cleared.

Linear Probing (22)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	$\times$

$$h(k) = 2k + 4 \bmod 7$$

- The usual protocol for deletion from a hash table using an open addressing scheme is to replace the key with a sentinel 'invalid element' marker, which is treated as an empty space during INSERT operations but treated as an element during FIND operations.

Linear Probing (23)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- **Question**: What happens if we want to insert more than M elements into a hash table of size M?

Linear Probing (24)

Index	Value
0	18
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- Using chaining, there is no limit on the number of elements stored in the table, since the lists at each index can be any size.
- With open addressing, it is necessary to resize the table (which requires rehashing and reinserting every element) if more than M elements must be inserted.

Quadratic Probing (1)

Index	Value
0	
1	
2	
3	
4	
5	
6	

$$h(k) = 2k + 4 \bmod 7$$

- **Exercise**: Insert the sequence

$$1, 3, 4, 10, 18$$

into the hash table above using quadratic probing to resolve collisions.

Quadratic Probing (2)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- No collisions occur for the first three insertions.
- $h(1) = 6$
- $h(3) = 3$
- $h(4) = 5$

Quadratic Probing (3)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(10) = 3$
- Index 3 is occupied, so index $h(10) + 1^2 = 4$ is probed.

Quadratic Probing (4)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(10) + 1^2 = 4$
- Since index 4 is empty, the key is inserted.

Quadratic Probing (5)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) = 5$
- Index 5 is occupied, so index $h(18) + 1^2 = 6$ is probed.

Quadratic Probing (6)

Index	Value
0	
1	
2	
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) + 1^2 = 6$
- Index 6 is occupied, so index $h(18) + 2^2 = 2$ is probed.

Quadratic Probing (7)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(18) + 2^2 = 2$
- Since index 2 is empty, the key is inserted.

Quadratic Probing (8)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- The table elements are all clustered in a group of consecutive indices, similar to the way they appeared with linear probing.
- In this case, the clustering is inevitable, and is mainly caused by the table size being too small.
- Quadratic probing does result in shorter probe sequences for the FIND operation, though.

Quadratic Probing (9)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- **Exercise**: Search for the key 17 in the hash table above.

Quadratic Probing (10)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) = 3$

Quadratic Probing (11)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) + 1^2 = 4$

Quadratic Probing (12)

Index	Value
0	
1	
2	18
3	3
4	10
5	4
6	1

$$h(k) = 2k + 4 \bmod 7$$

- $h(17) + 2^2 = 0$
- Since index 0 is empty, the search terminates unsuccessfully.

Double Hashing (1)

Index	Value
0	
1	
2	
3	
4	
5	
6	

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- **Exercise**: Insert the sequence

$$1, 3, 4, 10, 18$$

into the hash table above using double hashing to resolve collisions.

Double Hashing (2)

Index	Value
0	
1	
2	
3	
4	
5	
6	

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- Note: To use double hashing, we must guarantee that $h_2(k)$ is always non-zero.
- Otherwise, the probing sequence could enter an infinite loop.
- In general, a hashing scheme **fails** if it is not possible to insert M elements into a table of size M.

Double Hashing (3)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- No collisions occur for the first three insertions.
- $h_1(1) = 6$
- $h_1(3) = 3$
- $h_1(4) = 5$

Double Hashing (4)

Index	Value
0	
1	
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- $h_1(10) = 3$
- $h_2(10) = 5$
- Index 3 is occupied, so index $(h_1(10) + h_2(10)) \bmod 7 = 1$ is probed.

Double Hashing (5)

Index	Value
0	
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- Since index 1 is empty, the key is inserted.

Double Hashing (6)

Index	Value
0	
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- $h_1(18) = 5$
- $h_2(18) = 1$
- Index 5 is occupied, so index $(h_1(18) + h_2(18)) \bmod 7 = 6$ is probed.

Double Hashing (7)

Index	Value
0	
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- $h_1(18) = 5$
- $h_2(18) = 1$
- Index 6 is occupied, so index $(h_1(18) + 2h_2(18)) \bmod 7 = 0$ is probed.

Double Hashing (8)

Index	Value
0	18
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- Since index 0 is empty, the key is inserted.

Double Hashing (9)

Index	Value
0	18
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- **Exercise**: Search for the key 17 in the hash table above.

Double Hashing (10)

Index	Value
0	18
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- $h_1(17) = 3$
- $h_2(17) = 6$
- Since 17 is not found at index 3, index $(h_1(17) + h_2(17)) \bmod 7 = 2$ is probed.

Double Hashing (11)

Index	Value
0	18
1	10
2	
3	3
4	
5	4
6	1

$$h_1(k) = 2k + 4 \bmod 7$$

$$h_2(k) = 1 + (k \bmod 6)$$

- $h_1(17) = 3$
- $h_2(17) = 6$
- Since index 2 is empty, the search terminates unsuccessfully.