

Lecture 25: More Dictionaries & Hashing

CS 62

Fall 2018

Alexandra Papoutsaki & William Devanny

Naïve Version

- Warning: this code is simplified!

```
public class Map<K, V> {  
    protected V[] entries;  
  
    public V get(K key) {  
        int index = key.hashCode() % entries.length;  
        return entries[index];  
    }  
    public void put(K key, V value) {  
        int index = key.hashCode() % entries.length;  
        entries[index] = value;  
    }  
}
```

Hash Collisions

- `k1.hashCode() == k2.hashCode()` but `k1 != k2`
 - May also be caused by the modulus operation
- This is inevitable (e.g., the birthday paradox)
- A “good” hash function rarely collides

Two main strategies to avoid collisions

1) Open addressing (closed hashing):

Each bucket can store at most one entry

If hash falls in occupied bucket then search procedure for next empty bucket based on:

- Linear probing
- Quadratic probing
- Double probing

2) Closed Addressing (open or external hashing/bucketing):

Each bucket can store multiple entries

- Separate chaining

Linear Probing

- If we collide, check next entry until one is empty. Wrap around when at the end of table
- Deletion is complicated
- Can only hold `entries.length` items
- Resizing the table requires rehashing everything
- Suffers from primary clustering

Linear Probing Example: $h(k) = k \% 13$

Keys to insert: 17, 33, 18, 20, 44, 11, 19, 7 (ignore values)

0	1	2	3	4	5	6	7	8	9	10	11	12		
				17										
				17			33							
				17	18		33							
				17	18		33	20						Collision!
				17	18	44	33	20						Collision!
				17	18	44	33	20			11			
				17	18	44	33	20	19		11			Collision!
				17	18	44	33	20	19	7	11			Collision!

Linear Probing

- Keys with same hash will be clustered together
- The same thing can happen with unrelated keys forming primary clusters
- The more elements we add, the more collisions

Linear Probing Lookup

- Start at location returned by hashing function
 - If key was found → value
 - If key was not found search linearly until:
 - You find the key → value
 - You find an empty slot before you have found key → null
 - You wrapped around and ended up where you started → null
- Example: `get(7)` returns the value for 7

0	1	2	3	4	5	6	7	8	9	10	11	12
				17	18	44	33	20	19	7	11	

- Example: `get(6)` returns null

0	1	2	3	4	5	6	7	8	9	10	11	12
				17	18	44	33	20	19	7	11	

Quadratic Probing

- $h(k, i) = (h(k) + c_1i + c_2i^2) \pmod n, c_2 \neq 0$
- If $c_2 = 0$ then degrades to linear probing
- E.g., $h(k, i) = (h(k) + i^2) \pmod n$, then for every probing
 $h(k), h(k) + 1, h(k) + 4, \dots$
- Can result in cases where we don't try all slots
 - E.g., $n = 5$, and start with $h(k) = 1$.
 - Rehashings give 2, 0, 0, 2, 1, 2, 0, 0, ...
 - The slots 3 and 4 will never be examined to see if they have room
- Secondary Clustering

Quadratic Probing: $h(k, i) = (k \% 13) + i^2$

Keys to insert: 17, 33, 18, 20, 44, 11, 19, 7 (ignore values)

0	1	2	3	4	5	6	7	8	9	10	11	12		
				17										
				17			33							
				17	18		33							
				17	18		33	20						Collision!
				17	18	44	33	20						Collision!
				17	18	44	33	20			11			
				17	18	44	33	20		19	11			Collision!
			7	17	18	44	33	20		19	11			Collision!

Double Hashing

- Use second hash function on key to determine delta (interval) for next try
- $h(k, i) = (h_1(k) + i \cdot h_2(k)) \pmod n$,
- E.g., $h_2(k) = (k \pmod {n - 2}) + 1$
- Helps with primary and secondary clustering

- Example:
 - Suppose $h_1(n) = n \% 5$
 - Then $h_1(1) = h_1(6) = h_1(11)$
 - However, $h_2(1) = 2, h_2(6) = 1, h_2(11) = 3$

Separate Chaining

- Turn each bucket into a linked list (or array, etc.)
- On collision add to the bucket
- Searching list is fast if lists are small
- Deletion is simple
- Can hold more than `entries.length` items easily

Load Factor

- Performance depends on *load factor*
- Load factor is $\alpha = \frac{n}{N}$ where n = items in table and N = size of table
- Higher load factor \rightarrow more collisions \rightarrow slow
- Can be > 1 for external chaining
- For open addressing usually want to ensure $\alpha < 0.75$
 - Generally $\alpha > 0.75$ means resize the table (& rehash everything)

Performance

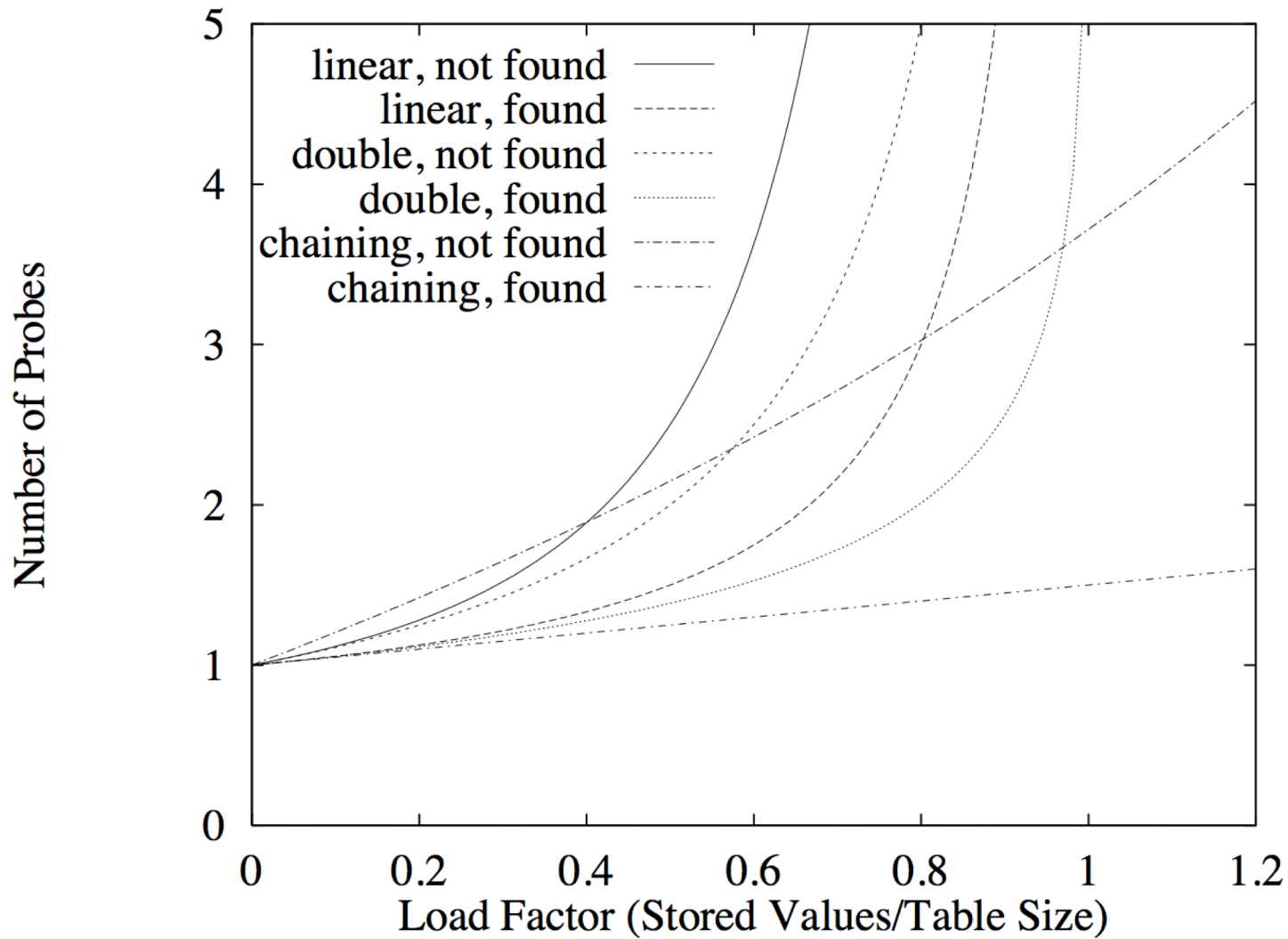
Strategy	Unsuccessful	Successful
Linear Probing	$1/2 (1 + 1/(1 - a)^2)$	$1/2 (1 + 1/(1 - a))$
Double Probing	$1/(1 - a)$	$-(1/a)/\log(1 - a)$
External Chaining	$a + e^{-a}$	$1 + 1/2a$

Entries represent number of comparisons needed to find a specific element or demonstrate that it is not in the hash table

Performance for $a = .9$

Strategy	Unsuccessful	Successful
Linear Probing	55	5.5
Double Probing	10	~4
External Chaining	3	1.45

Entries represent number of comparisons needed to find a specific element or demonstrate that it is not in the hash table



Space requirements

- Open addressing: $\text{TableSize} + n * \text{objectsize}$
- External chaining: $\text{TableSize} + (n * \text{objectsize} + 1)$
- Rule of thumb:
 - Small elements, small load factor: open addressing
 - Large elements, large load factor: external chaining