Hashing, Day 2

Concept that we use to implement Maps/Sets.

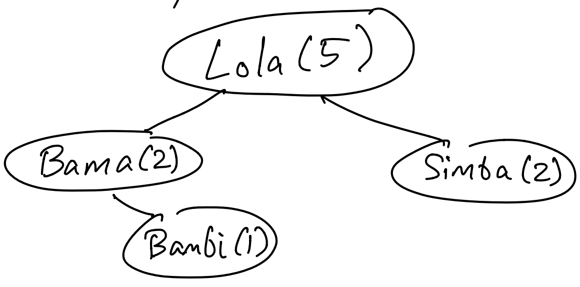
In a perfect world, hash tables → lead to maps/sets that can run in O(1) time.

"Hash Function" → takes an item that you're trying to store in the map/set, gives you back an integer.

↳ In Java, this is called hashCode().

(Key) Name	(Value) Age	Hashcode for Name
Lola	5	41
Simba	2	73
Bama	2	50
Bambi	1	26
Schatzi	1	92
Karbon	1	34
Biscuit	9	18
Tucker	10	65
Karma	11	77

BST →



Hashtable

Arraylist (of size 5)

[0]	Bama (2)
[1]	Lola (5)
[2]	
[3]	Simba (2)
[4]	

put(Lola, 5)
(Key, Value)

hashCode("Lola") ⟶ 41

Take remainder of hashcode & size of the table.

41 % 5 = 1 index in the hashtable

get ("Lola") → hashcode ("Lola") → 41 % 5 → 1

Hash function

Goal: to make up an integer that combines as many pieces of the data as possible.

– Idea – Add up all the unicode #s for each ~~item~~ letter in a string.

$\frac{\text{"L"}}{\#} + \frac{\text{"o"}}{\#} + \frac{\text{"l"}}{\#} + \frac{\text{"a"}}{97} = \boxed{?}$

Unicode

"A" → 65

"a" → 97

$\left.\begin{array}{l}\text{hashcode("abc")}\\ \text{hashcode("cba")}\end{array}\right\}$ collision

- Idea 2.0 → Take just the unicode code for position [0] in the string.

hashCode("abc") → 97
hashCode("cba") → 99

- Idea 3.0 →

code for [0] + (code for index [1])

Collision?

↳ Either when 2 pieces of data have the same hashCode.
↳ or, when (after you % size of table) → get the same index.

Open Addressing

We are going to allow items in the hashtable to be stored at a different index than what the hashCode function tells us.

→ Linear Probing

When there is a collision, store/search the item at the following index.

↳ if you want to store something at index i & it's full, try ~~[illegible]~~ i+1, i+2, i+3 . . . .

0	Bama (2)
1	Lola (5)
2	Bambi(1)
3	Simba(2)
4	null

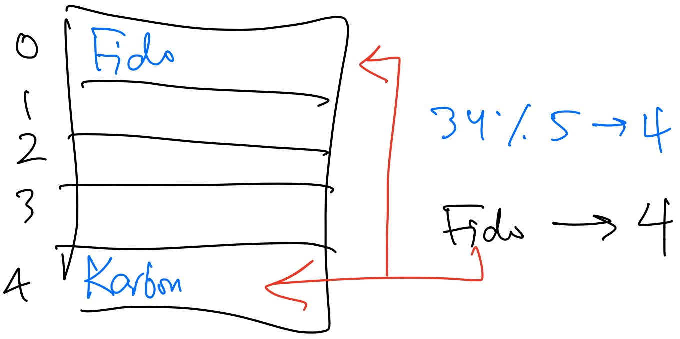
(Key) Name	(Value) Age	Hashcode for Name
Lola	5	41
Simba	2	73
Bama	2	50
Bambi	1	26
Schatzi	1	92
Karbon	1	34
Biscuit	9	18
Tucker	10	65
Karma	11	77

hashCode("Bambi") → 26 % 5
→ 1

get("Bambi") → 26 % 5 → 1

Alg for linear probing to add an item into a hashtable

```
put(key, value)
    compute index by taking hashcode(key) % size of table
    if table[index] == null
        store (key, value) at table[index]
    else if table[index] is full
        check if table[index] == key
            replace the old value w/ the new one
        if table[index] != key
            increment index by 1.
            Try again.
            Loop to keep incrementing until we
            find a null slot, with wrap-around.
```



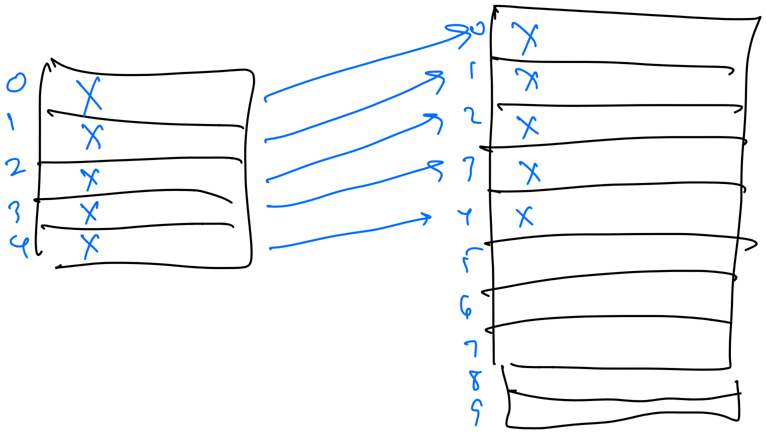
"Wrap-around"

What happens when the array fills up?

Rehashing

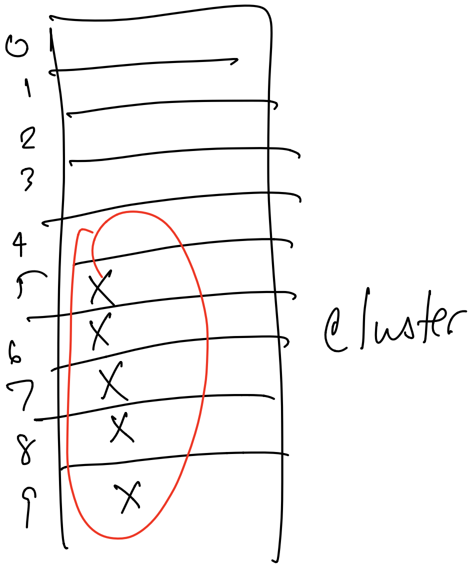
Make a new table w/ a bigger size.

Rehash all items into the new table.



Linear probing often causes "clusters"

5, 6, 5, 6, 7



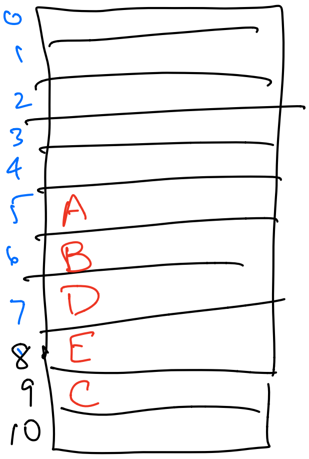
Fido → 5

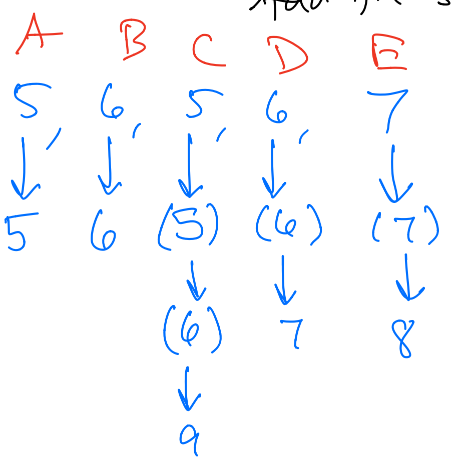
Quadratic probing → designed to break up clusters.

Linear probing → $\underbrace{i, i+\boxed{1}, i+\boxed{2}, i+\boxed{3}, i+\boxed{4}}_{\text{all consecutive} \to \text{clusters}}\ldots$

Quadratic → $i, i+\boxed{1}, i+\boxed{4}, i+\boxed{9}, i+\boxed{16}\ldots$

Add the square of each number


0
1
2
3
4
5
6
7
8
9
10
A
B
D
E
C


A
B
C
D
E
5,
6,
5,
6,
7
5
6
(5)
(6)
(7)
(6)
7
8
9