CSCI 241

Scott Wehrwein

Hash Tables: Rehashing

Goals

Know how and why to grow and shrink the capacity of a hash table by resizing the array and rehashing its contents.

Be prepared to implement rehashing so it runs in worst-case O(C + n).

Load Factor: Performance Implications

$$\text{Load factor } \lambda = \frac{\text{\# entries in table}}{\text{size of the array}}$$

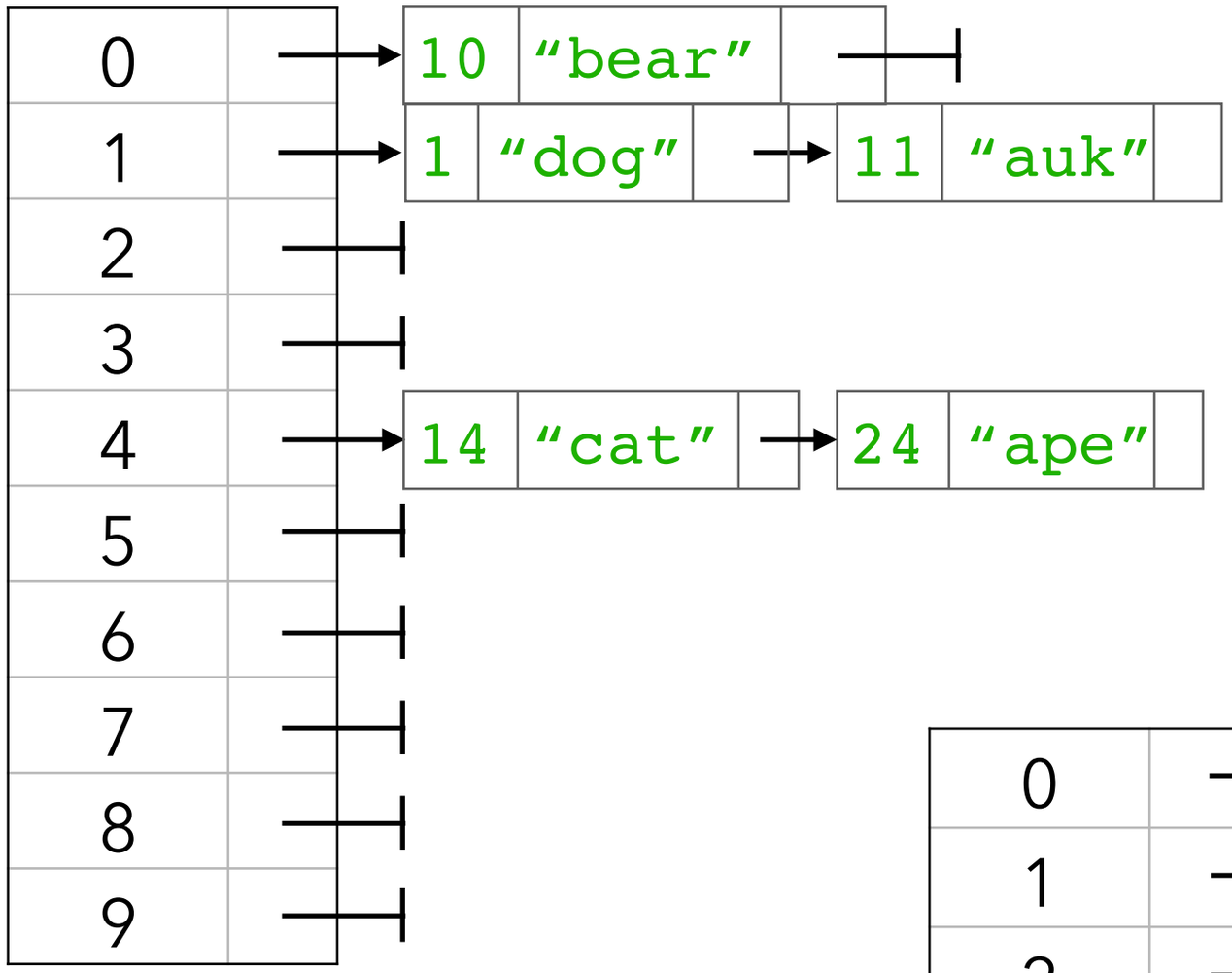
If λ is large, runtime is slow.

If λ is small, memory is wasted.

Strategy: grow or shrink array when λ gets too large *or* small.

Shrinking the array

Need to **rehash**: put each element where it belongs in the new array.



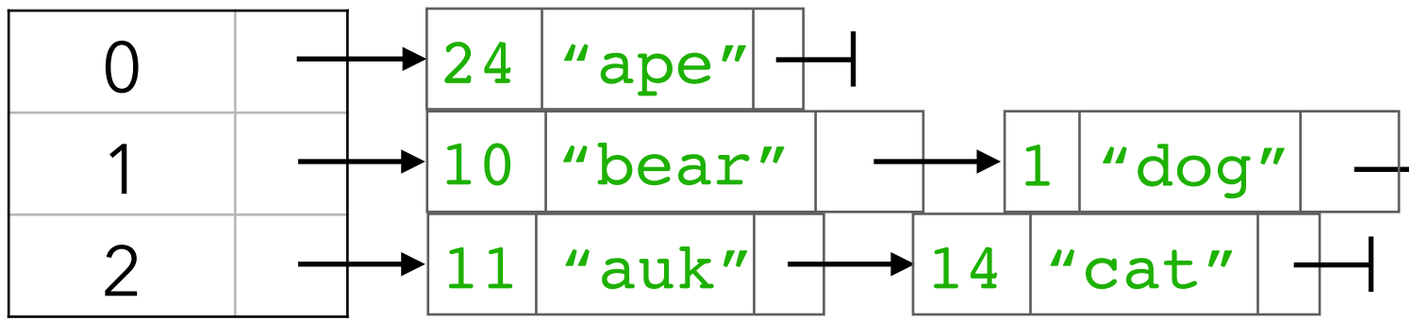
(10 % 3) -> 1

(1 % 3) -> 1

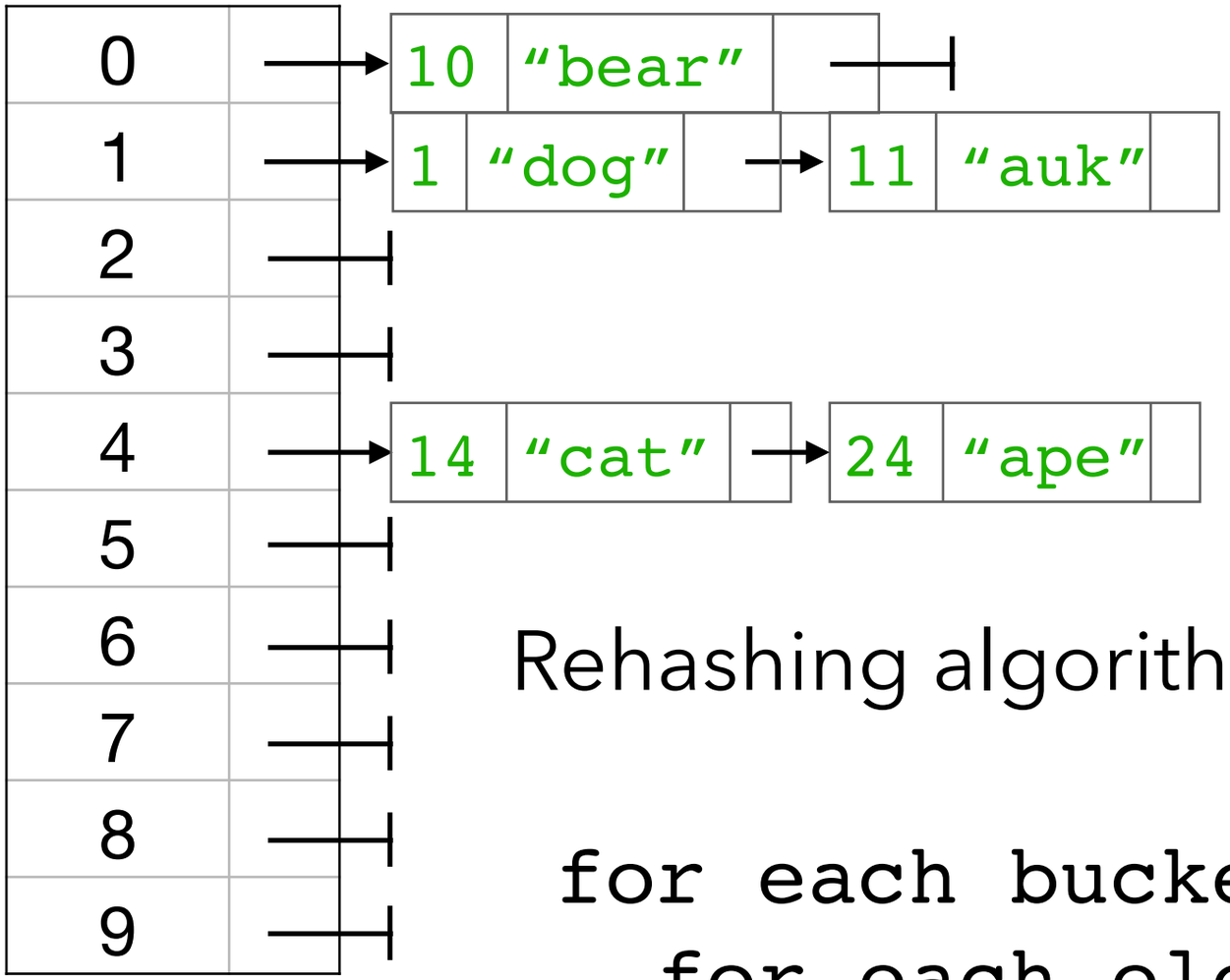
(11 % 3) -> 2

(14 % 3) -> 2

(24 % 3) -> 0



Rehashing: Runtime, take 1



Let C = array size

Let n = number of entries

Overall runtime is:

- worst-case $O(C + n^2)$
- average-case $O(C + n)$

Rehashing algorithm:

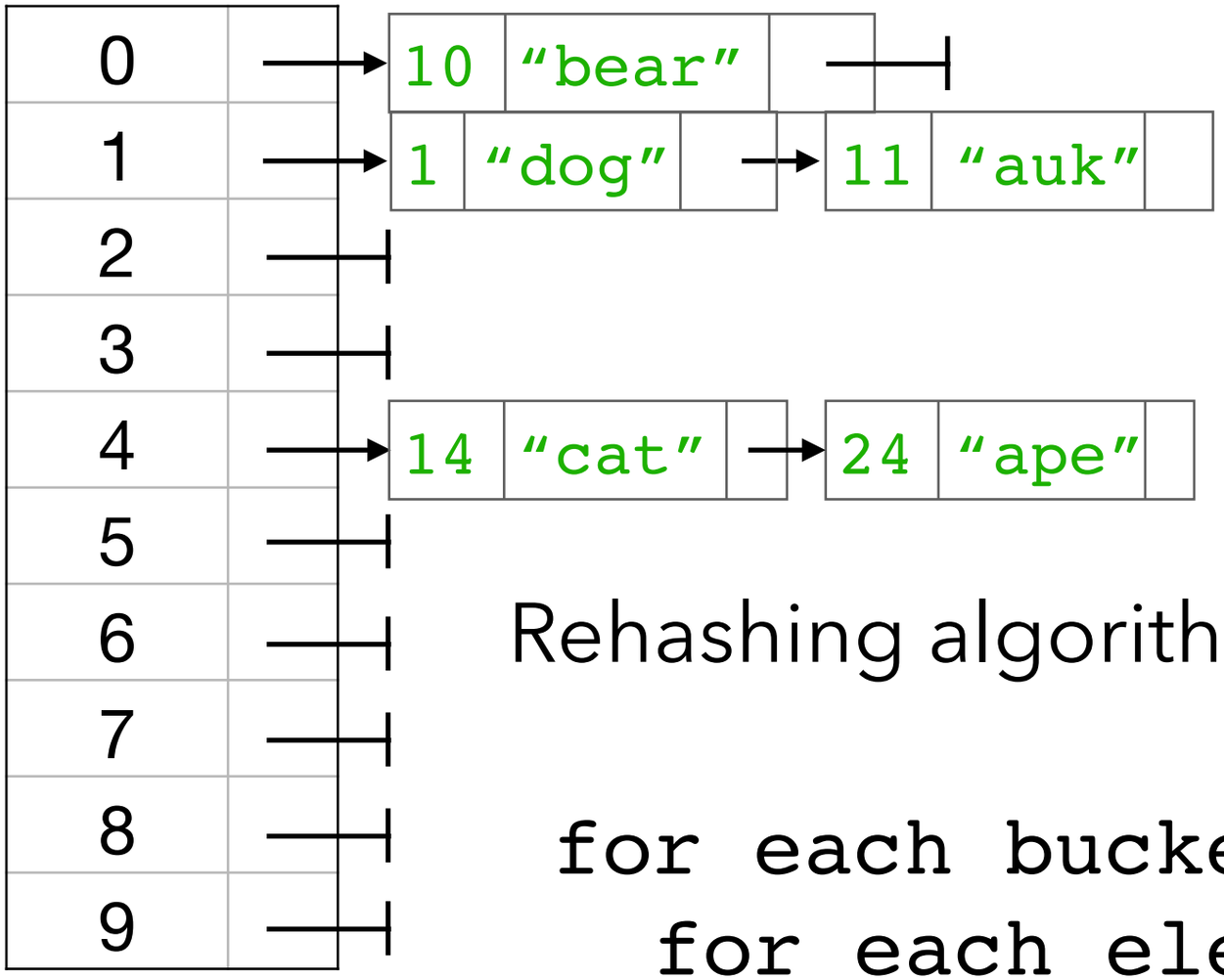
visits C buckets

visits n entries (total)

could be O(n) =(

```
for each bucket b:
  for each element e in b:
    put e into the new array
```

Rehashing: Runtime, take 2



Let C = array size

Let n = number of entries

Overall runtime is:

- worst-case O(C + n)

Rehashing algorithm:

visits C buckets

visits n entries (total)

could it be O(n)?

```
for each bucket b:
  for each element e in b:
    put e into the new array
```

`put` is O(n) because it has to search for existing keys.

Here, we **can't** have duplicate keys: all entries were already in the map!

Consequence: we don't need to search the bucket when rehashing