Collision Resolution

_15-121_

Data Structures

Ananda Gunawardena

Hashing

Big Idea in Hashing

- Let $S=\{a_1,a_{2,\ldots}a_m\}$ be a set of objects that we need to map into a table of size N.
    - Find a function such that $H:S \Rightarrow [1\ldots n]$
    - Ideally we'd like to have a 1-1 map
    - But it is not easy to find one
    - Also function must be easy to compute
    - Also picking a prime as the table size can help to have a better distribution of values

Collisions

- Two keys mapping to the same location in the hash table is called "Collision"
- Collisions can be reduced with a selection of a good hash function
- But it is not possible to avoid collisions altogether
  - Unless we can find a perfect hash function
  - Which is hard to do

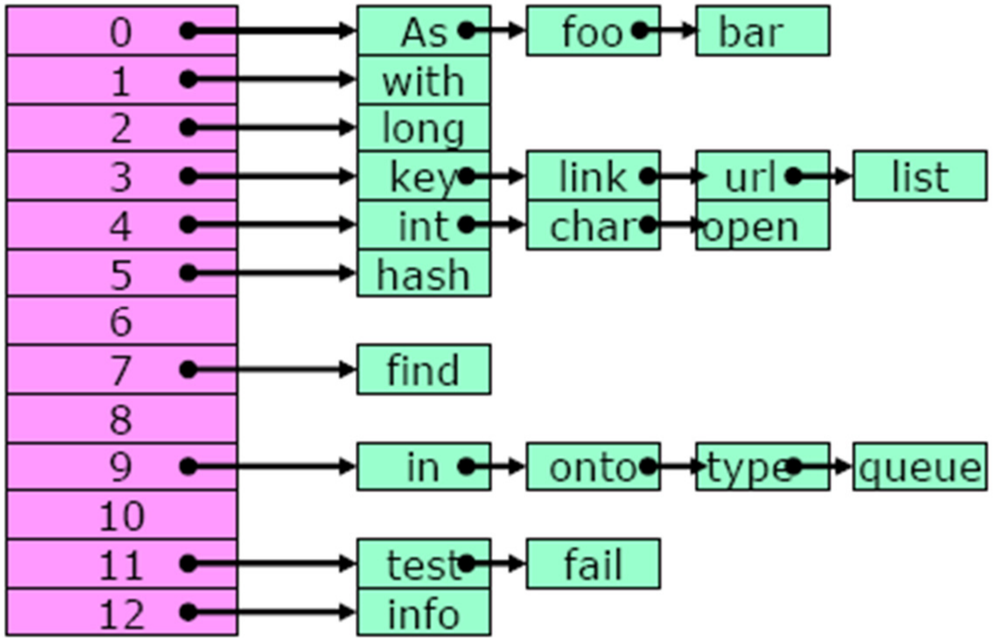
Collision Resolving strategies

- Few Collision Resolution ideas
  - Separate chaining
  - Some Open addressing techniques
    - Linear Probing
    - Quadratic Probing

Separate Chaining

Separate Chaining

- Collisions can be resolved by creating a list of keys that map to the same value



Separate Chaining

- Use an array of linked lists
  - LinkedList[ ]   Table;
  - Table = new LinkedList(N), where N is the table size
- Define Load Factor of Table as
  - λ = number of keys/size of the table

    (λ can be more than 1)
- Still need a good hash function to distribute keys evenly
  - For search and updates

Linear Probing

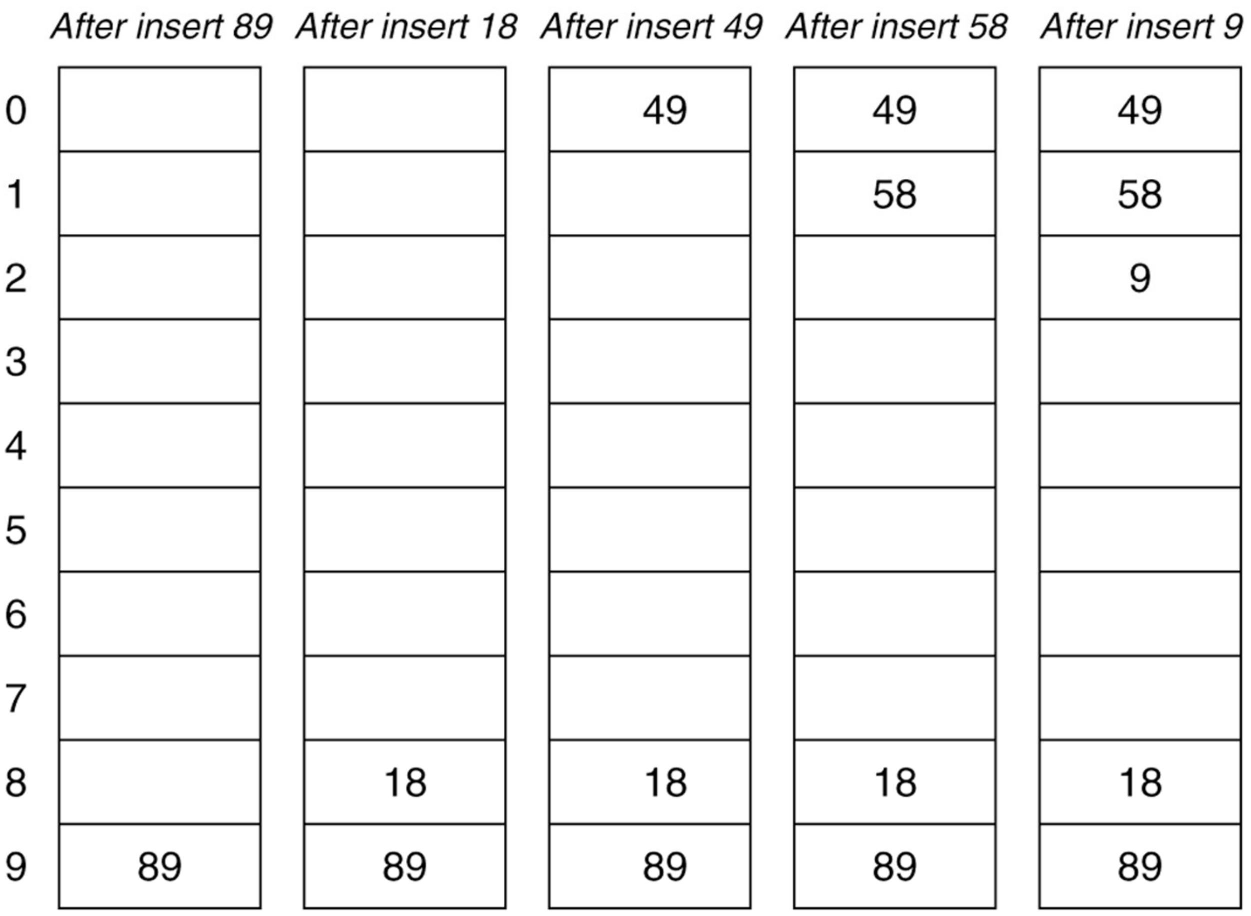
Linear Probing

- The idea:
  - Table remains a simple array of size N
  - On **insert(x)**, compute f(x) mod N, if the cell is full, find another by sequentially searching for the next available slot
    - Go to f(x)+1, f(x)+2 etc..
  - On **find(x)**, compute f(x) mod N, if the cell doesn't match, look elsewhere.
  - Linear probing function can be given by
    - $h(x, i) = (f(x) + i) \bmod N$ $(i=1,2,\ldots.)$

Figure 20.4

Linear probing hash table after each insertion

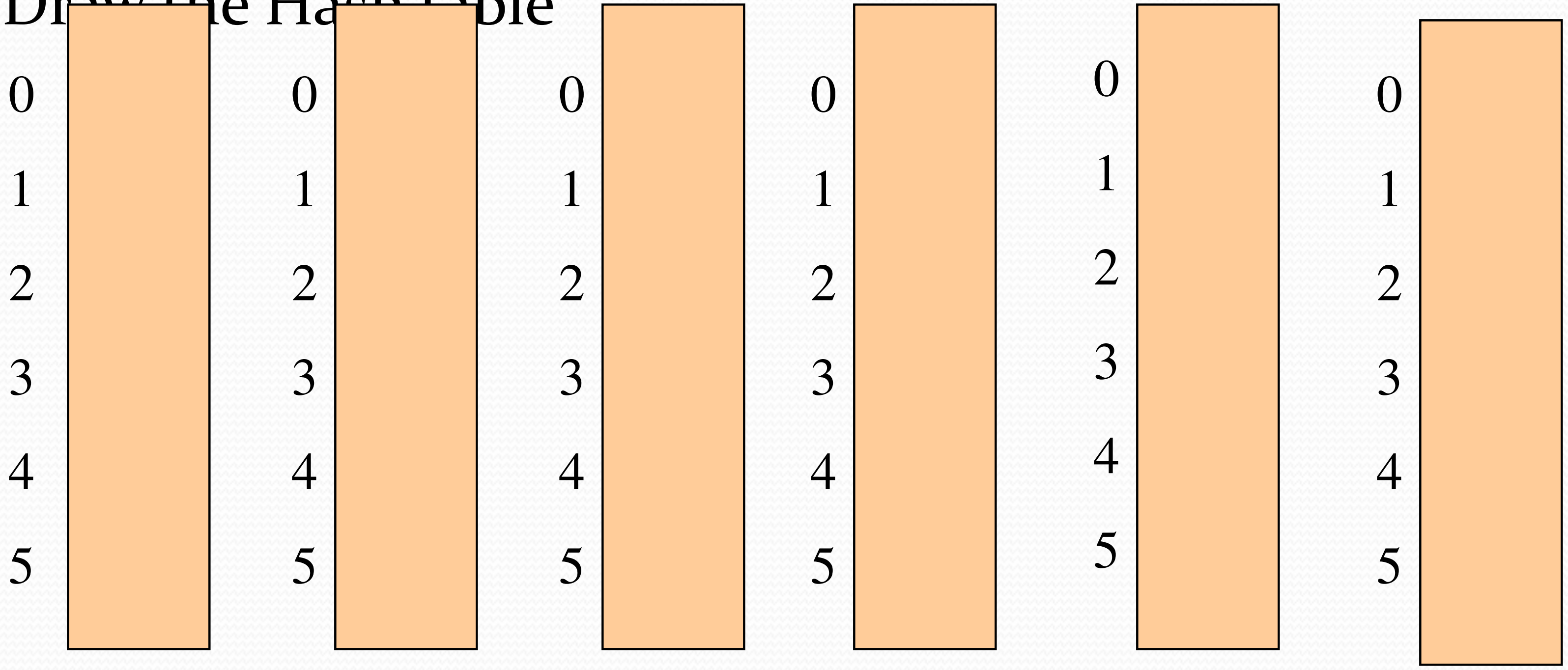
```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```



	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1				58	58
2					9
3					
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

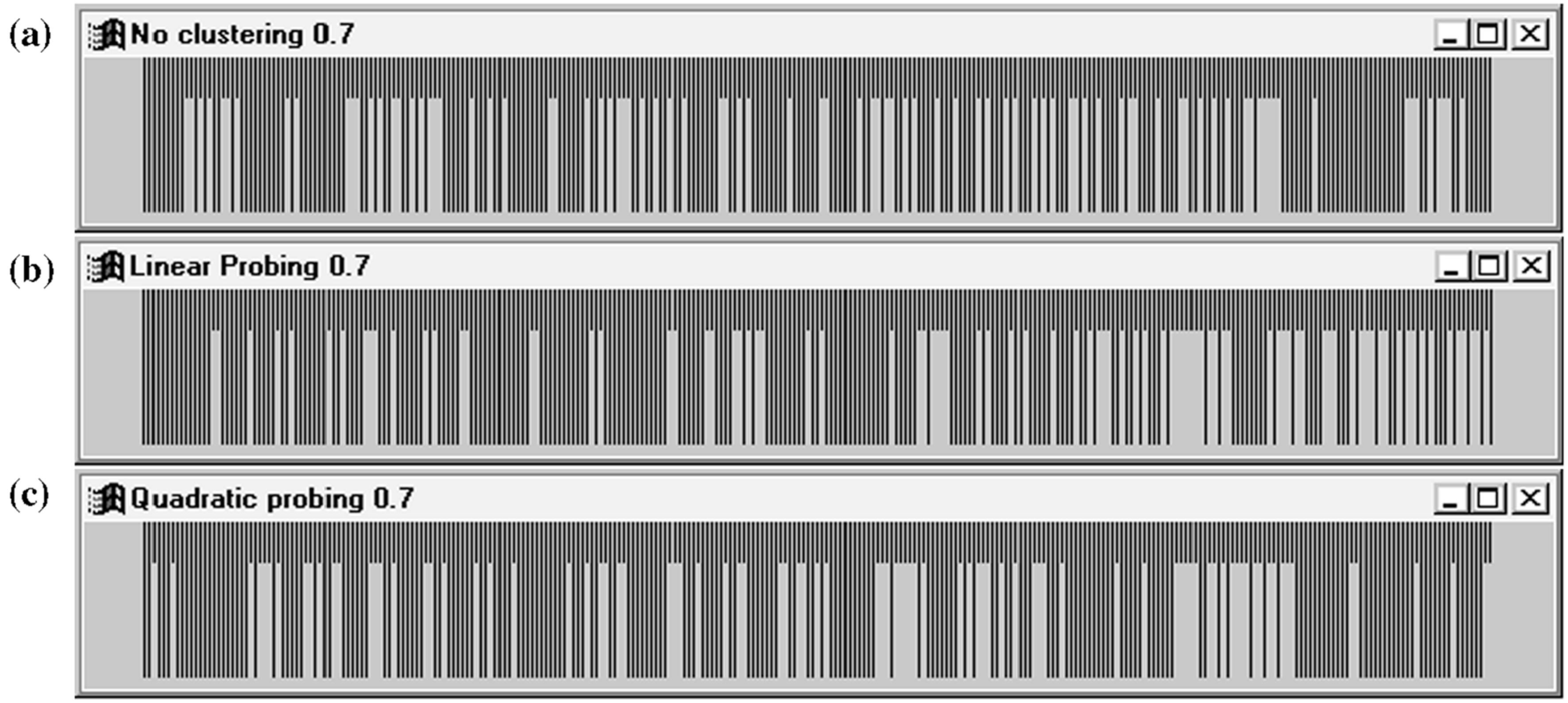
Linear Probing Example

- Consider H(key) = key Mod 6 (assume N=6)
- H(11)=5, H(10)=4, H(17)=5, H(16)=4,H(23)=5
- Draw the Hash table



Clustering Problem

- Clustering is a significant problem in linear probing. Why?
- Illustration of primary clustering in linear probing (b) versus no clustering (a) and the less significant secondary clustering in quadratic probing (c). Long lines represent occupied cells, and the load factor is 0.7.



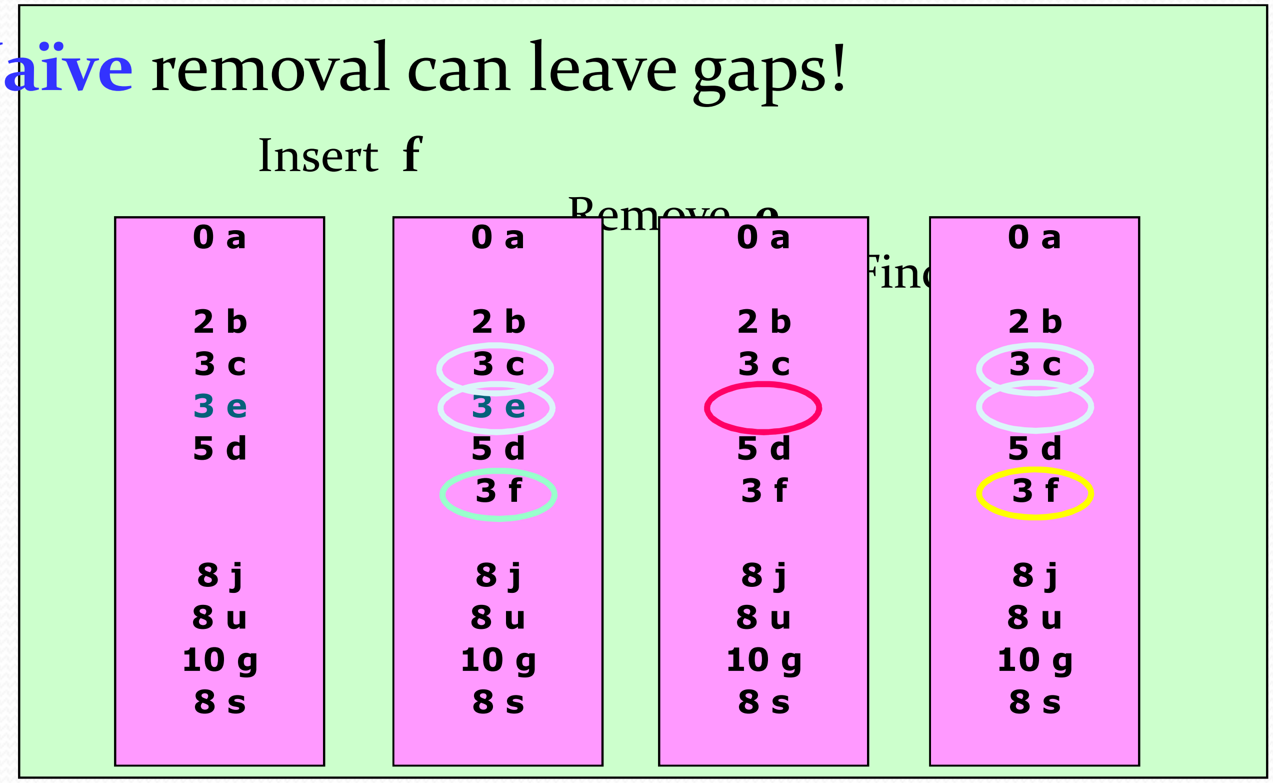
Deleting items from a hash table

Deleting items

- How about deleting items from Hash table?
  - Item in a hash table connects to others in the table(eg: BST).
  - Deleting items will affect finding the others
  - "Lazy Delete" – Just mark the items as inactive rather than removing it.

Lazy Delete

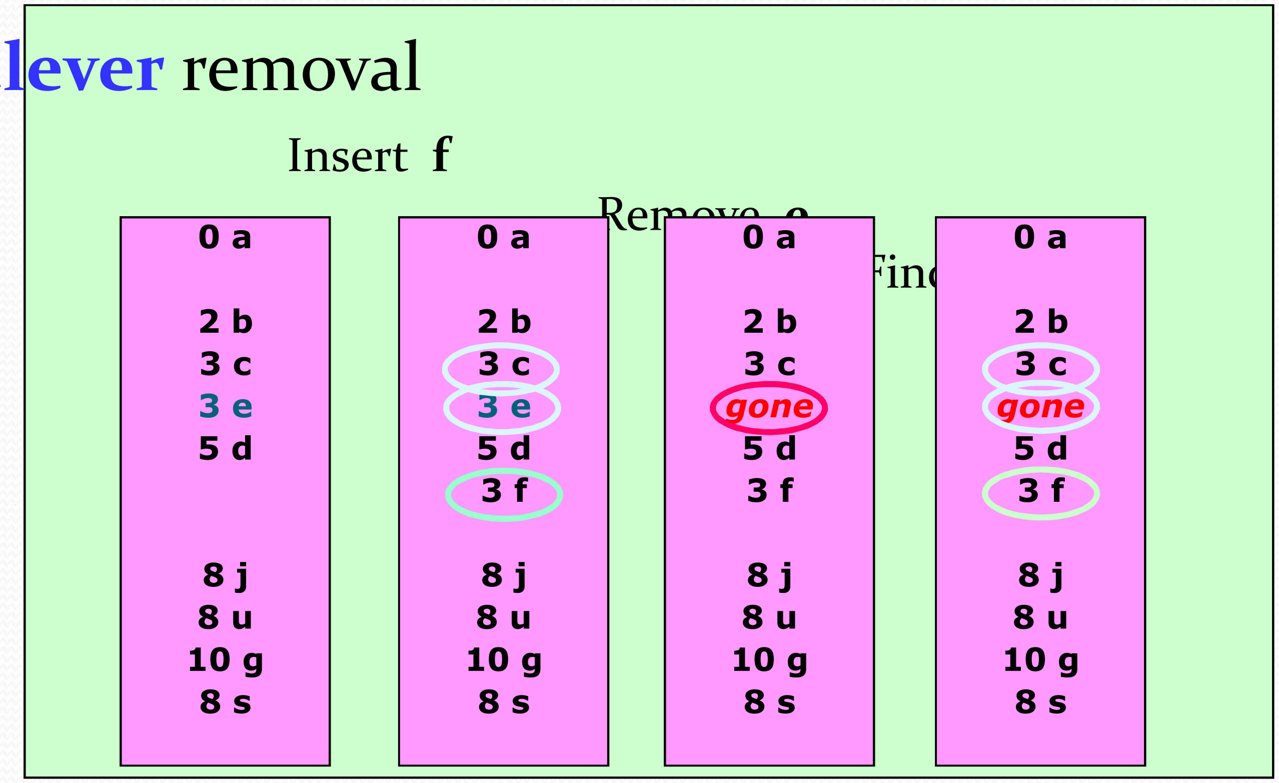
- **Naïve** removal can leave gaps!



"3 f" means *search key* f and *hash key* 3

Lazy Delete

- Clever removal



0 a	0 a	0 a	0 a
2 b	2 b	2 b	2 b
3 c	3 c	3 c	3 c
3 e	3 e	*gone*	*gone*
5 d	5 d	5 d	5 d
	3 f	3 f	3 f
8 j	8 j	8 j	8 j
8 u	8 u	8 u	8 u
10 g	10 g	10 g	10 g
8 s	8 s	8 s	8 s

"3 f" means *search key* f and *hash key* 3

Load Factor (open addressing)

- **definition:** The *load factor* λ of a probing hash table is the fraction of the table
that is full. The load factor ranges from 0 (empty) to 1 (completely full).
- It is better to keep the load factor under 0.7
- Double the table size and rehash if load factor gets high
- Cost of Hash function f(x) must be minimized
- When collisions occur, linear probing can always find an empty cell
    - But clustering can be a problem

Quadratic Probing

Quadratic probing

- Another open addressing method
- Resolve collisions by examining certain cells (1,4,9,...) away from the original probe point
- Collision policy:
  - Define $h_0(k), h_1(k), h_2(k), h_3(k), \ldots$
    where $h_i(k) = (\text{hash}(k) + i^2) \bmod \text{size}$
- Caveat:
  - May not find a vacant cell!
    - Table must be less than half full ($\lambda < ½$)
  - (Linear probing always finds a cell.)

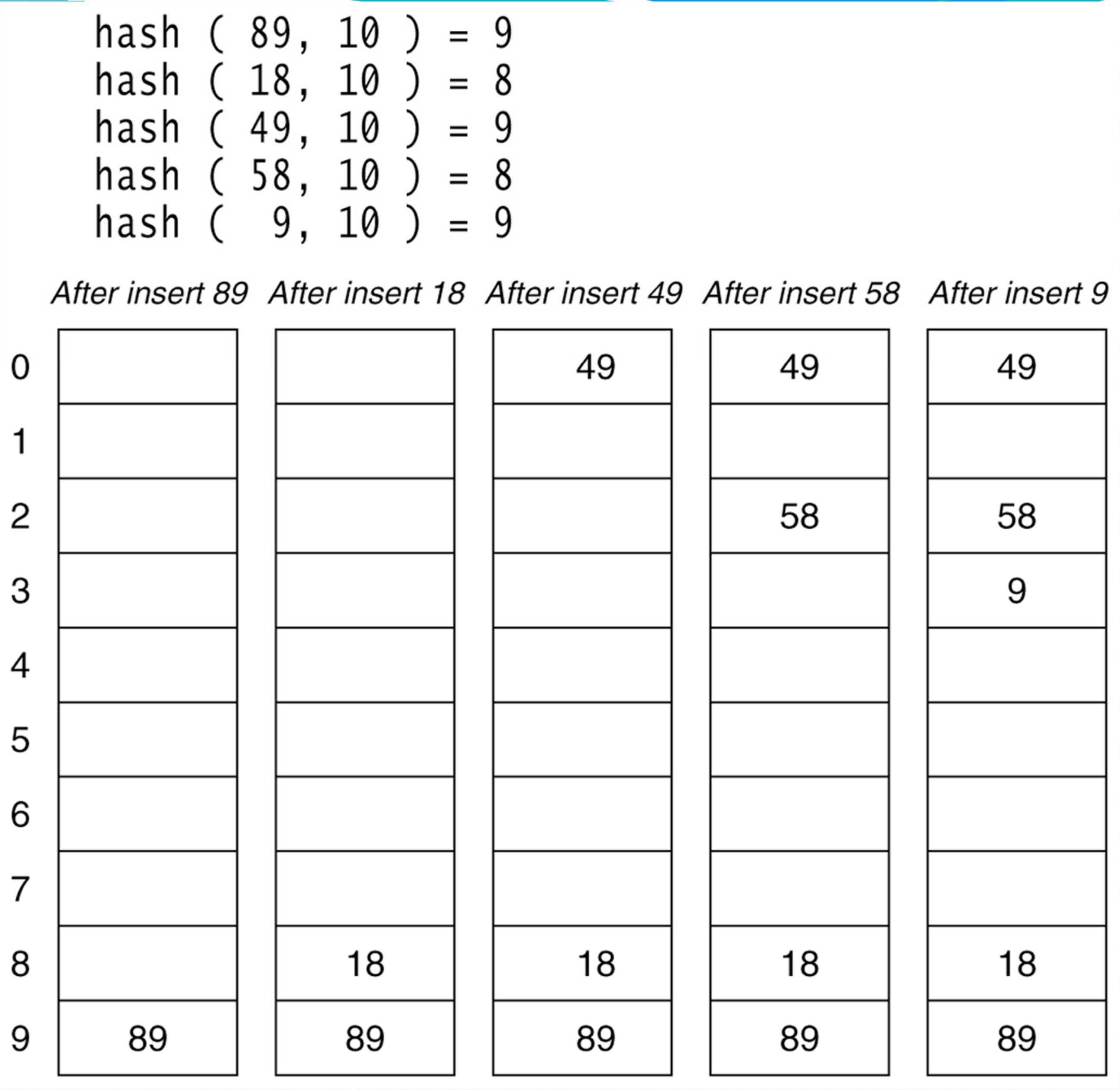
Quadratic probing

- Another issue
  - Suppose the table size is 16.
  - Probe offsets that will be tried:

$$\left.\begin{array}{l} 1 \bmod 16 = 1 \\ 4 \bmod 16 = 4 \\ 9 \bmod 16 = 9 \\ 16 \bmod 16 = 0 \\ 25 \bmod 16 = 9 \\ 36 \bmod 16 = 4 \\ 49 \bmod 16 = 1 \\ 64 \bmod 16 = 0 \end{array}\right\} \text{only four different values!}$$

$$81 \bmod 16 = 1$$

Figure 20.6
A quadratic probing hash table after each insertion (note that the table size was poorly chosen because it is not a prime number).



```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1					
2				58	58
3					9
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89