CSE 373
Data Structures & Algorithms

Lecture 10

Midterm Postmortem and Hashing

Midterm Postmortem

- Will discuss the solutions in class…

Dictionary Implementations So Far

	Unsorted linked list	Sorted Array	BST	AVL	B-trees
Insert					
Find					
Delete					

Dictionary Implementations So Far

	Unsorted linked list	Sorted Array	BST	AVL	B-trees
Insert	**O(1)**				
Find	**O(N)**				
Delete	**O(N)**				

Dictionary Implementations So Far

	Unsorted linked list	Sorted Array	BST	AVL	B-trees
Insert	**O(1)**	**O(N)**			
Find	**O(N)**	**O(log N)**			
Delete	**O(N)**	**O(N)**			

Dictionary Implementations So Far

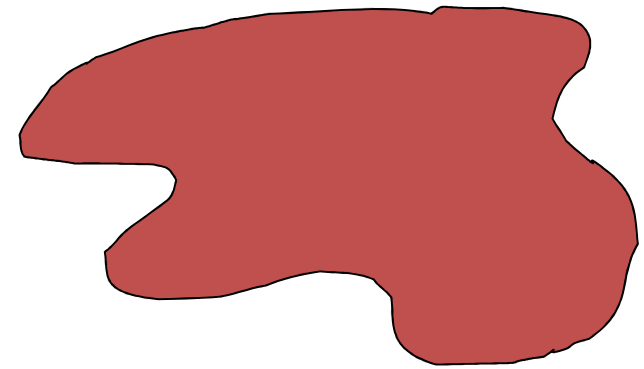
	Unsorted linked list	Sorted Array	BST	AVL	B-trees
Insert	**O(1)**	**O(N)**	**O(N)**		
Find	**O(N)**	**O(log N)**	**O(N)**		
Delete	**O(N)**	**O(N)**	**O(N)**		

Dictionary Implementations So Far

	Unsorted linked list	Sorted Array	BST	AVL	B-trees
Insert	**O(1)**	**O(N)**	**O(N)**	**O(log N)**	**O(log N)**
Find	**O(N)**	**O(log N)**	**O(N)**	**O(log N)**	**O(log N)**
Delete	**O(N)**	**O(N)**	**O(N)**	**O(log N)**	**O(log N)**

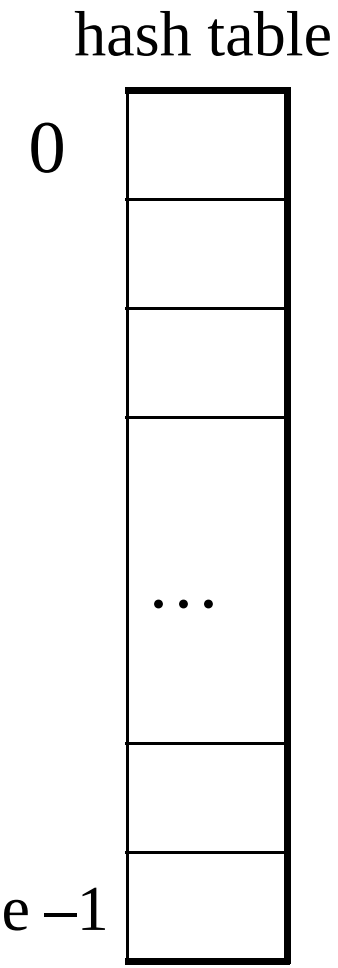
Hash Tables

- Find, insert, delete:
  constant time on average!
- A **hash table** is an array
  of some fixed size.
- General idea:



key space (e.g., integers, strings)

hash function:
index = h(K)



Hash Tables

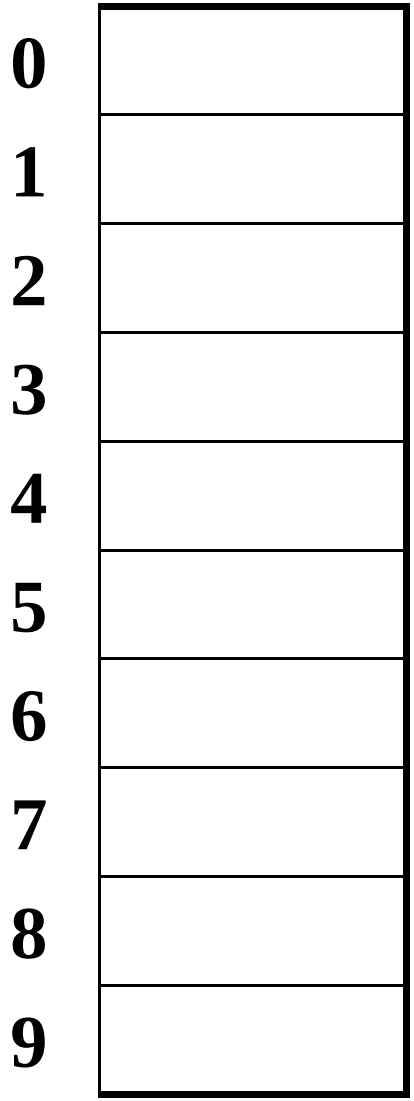
Key space of size M, but we only want to store subset of size N, where N<<M.

- Keys are identifiers in programs. Compiler keeps track of them in a symbol table.
- Keys are student names. We want to look up student records quickly by name.
- Keys are chess configurations in a chess playing program.
- Keys are URLs in a database of web pages.

Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = ?

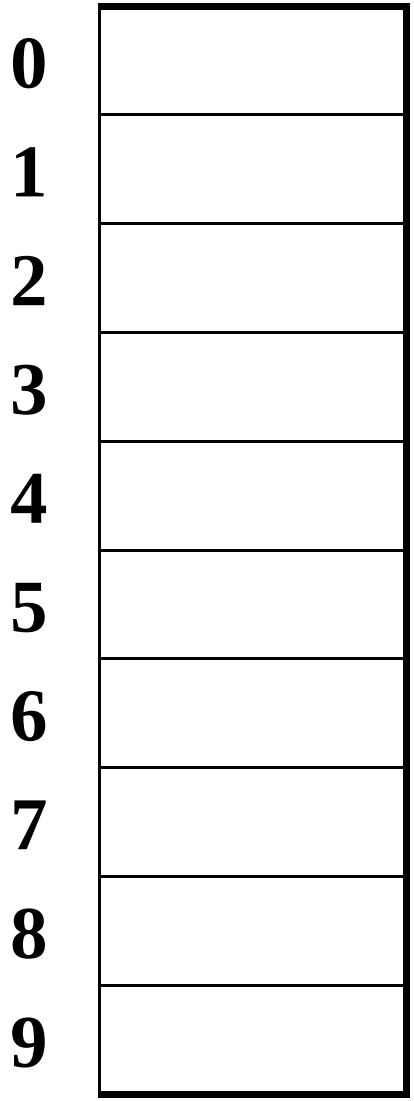


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K

- **Insert**: 7, 18, 41, 34

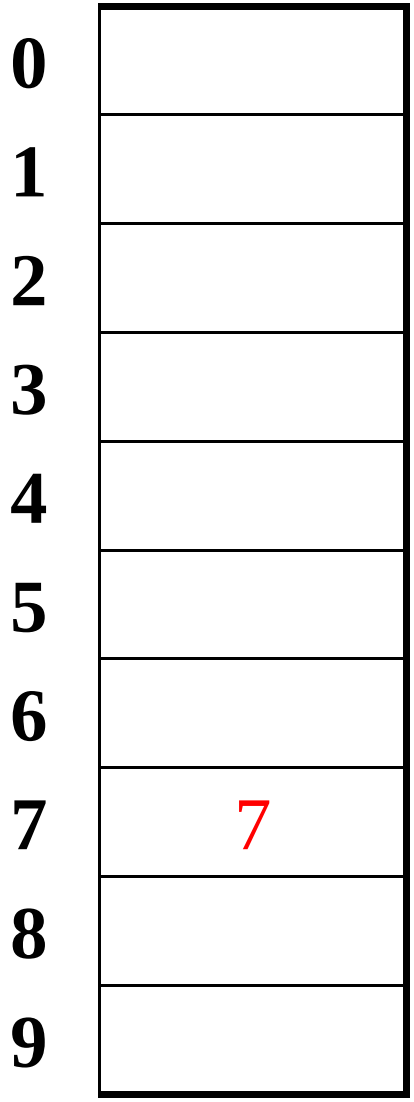


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K

- **Insert**: 7, 18, 41, 34

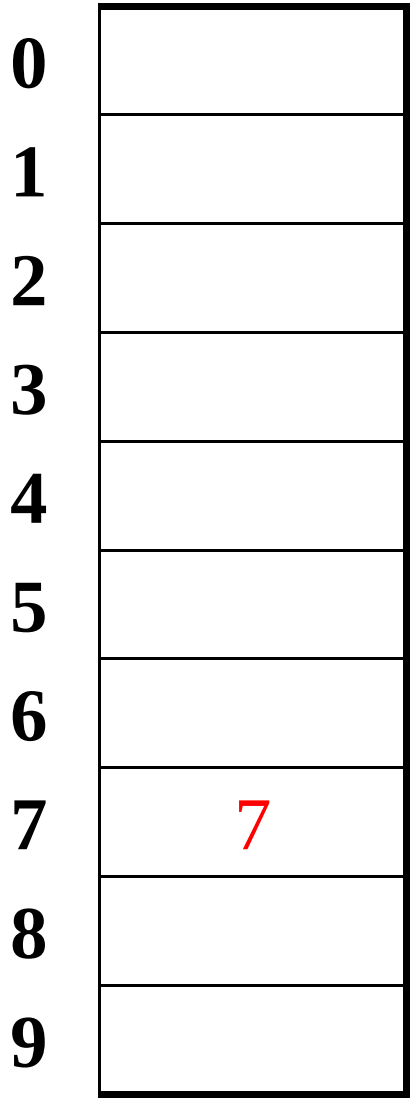


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K

- **Insert**: 7, 18, 41, 34

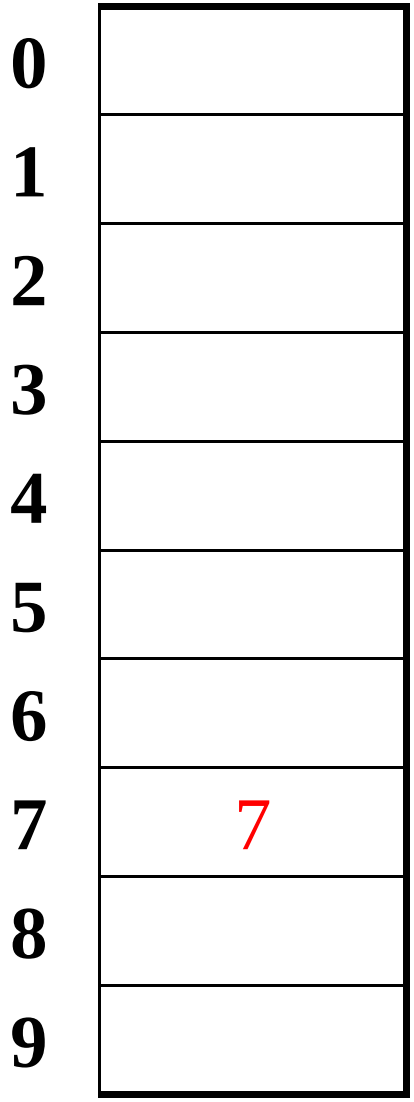


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K % 10

- **Insert**: 7, 18, 41, 34

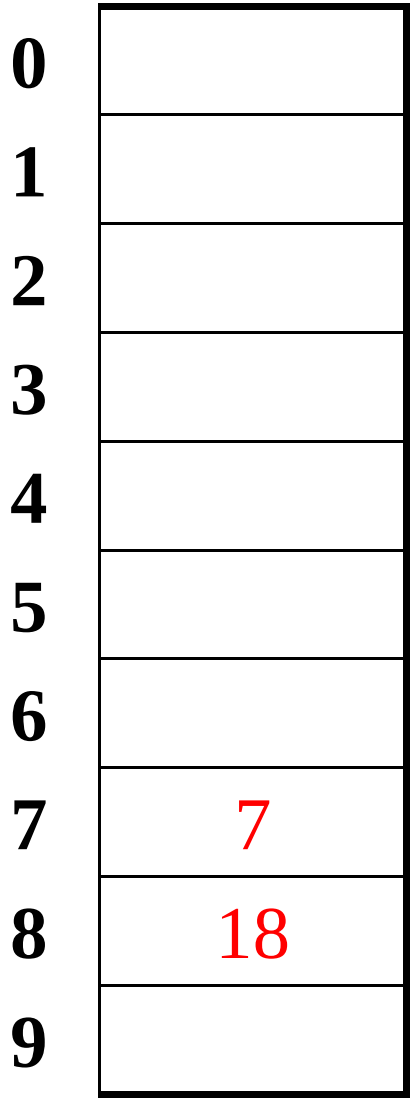


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K % 10

- **Insert**: 7, 18, 41, 34

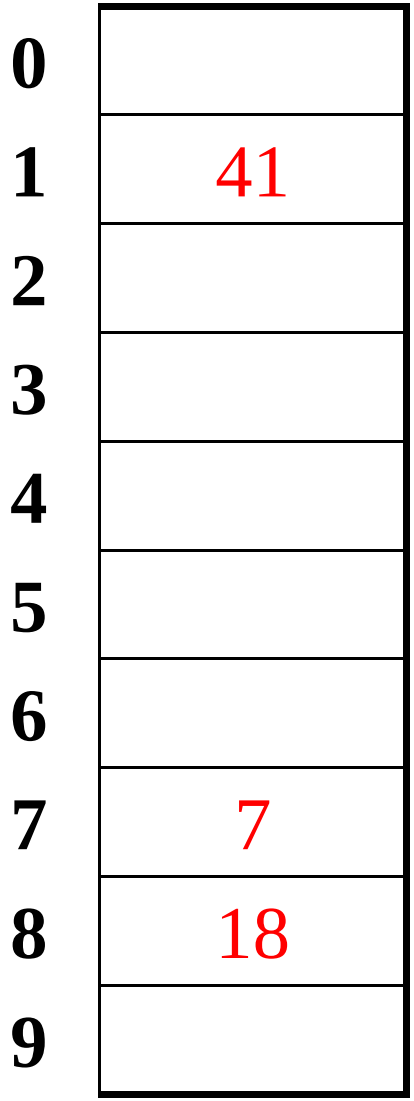


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K % 10

- **Insert**: 7, 18, 41, 34

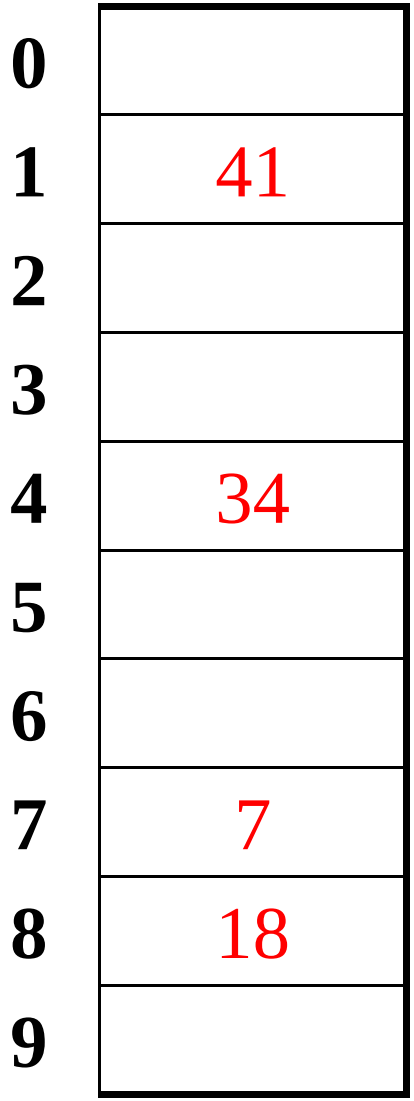


Simple Integer Hash Functions

- key space = integers
- TableSize = 10

- h(K) = K % 10

- **Insert**: 7, 18, 41, 34

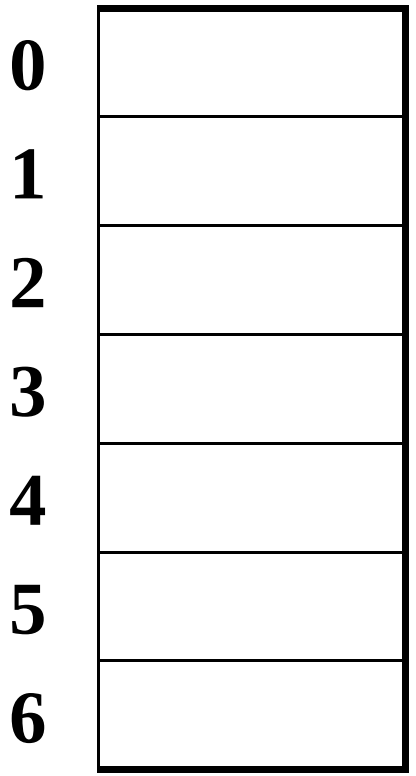


Simple Integer Hash Functions

- key space = integers
- TableSize = 7

- h(K) = K % 7

- **Insert**: 7, 18, 41, 34

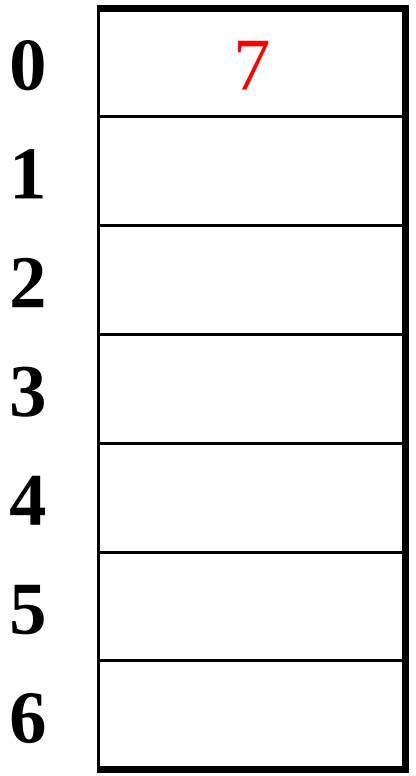


Simple Integer Hash Functions

- key space = integers
- TableSize = 7

- h(K) = K % 7

- **Insert**: 7, 18, 41, 34

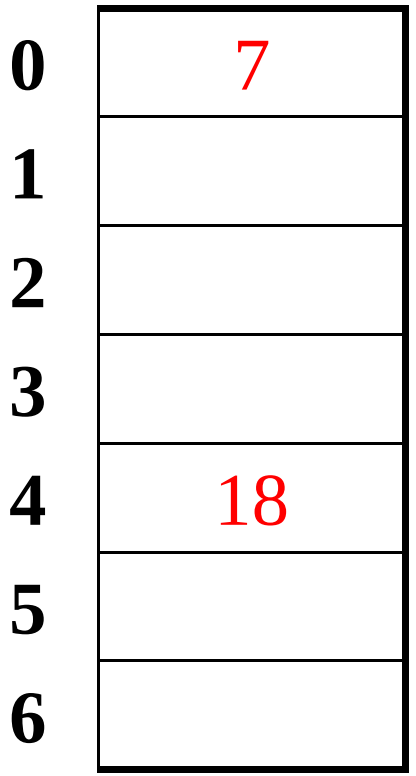


Simple Integer Hash Functions

- key space = integers
- TableSize = 7

- h(K) = K % 7

- **Insert**: 7, 18, 41, 34

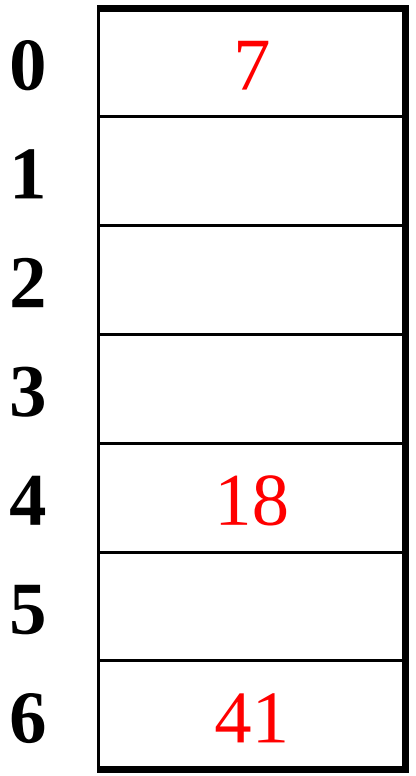


Simple Integer Hash Functions

- key space = integers
- TableSize = 7

- h(K) = K % 7

- **Insert**: 7, 18, 41, 34

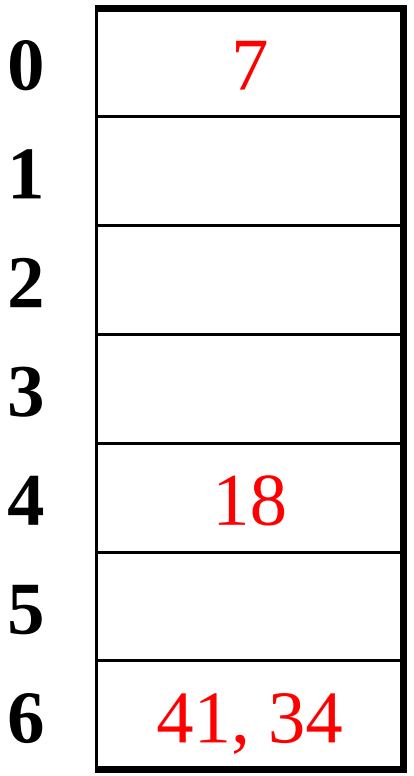


Simple Integer Hash Functions

- key space = integers
- TableSize = 7

- h(K) = K % 7

- **Insert**: 7, 18, 41, 34



- Multiple objects in a cell called a collision

String Hash Functions?

key space = strings

$K = s_0\ s_1\ s_2\ \ldots\ s_{m-1}$ (where s_i are chars: $s_i \in [0, 128]$)

What are some desirable properties for a hash function?

Some String Hash Functions

key space = strings

K = $s_0\ s_1\ s_2 \ldots s_{m-1}$ (where s_i are chars: $s_i \in [0, 128]$)

1. h(K) = s_0 % TableSize
2. h(K) = $\left(\sum_{i=0}^{m-1} s_i \right)$ % TableSize
3. h(K) = $\left(\sum_{i=0}^{k-1} s_i \cdot 37 \right)$ % TableSize
4. h(K) = $\left(\sum_{i=0}^{m-1} s_i \cdot 128^i \right)$ % TableSize

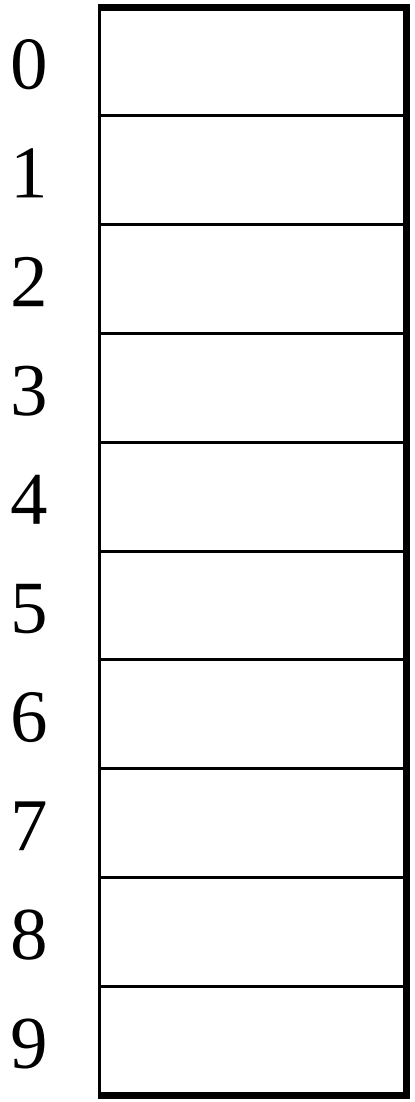
Collision Resolution

Collision:

when two keys map to the same location in the hash table.

How can we cope with collisions?

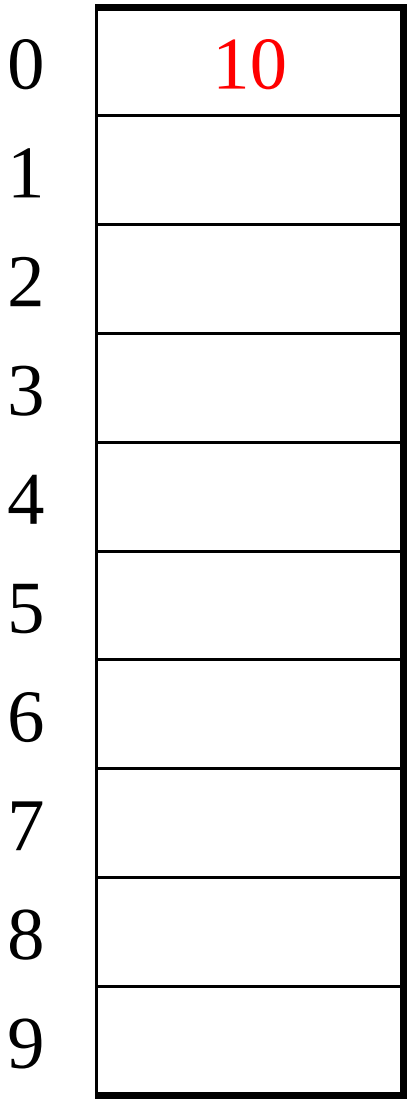
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

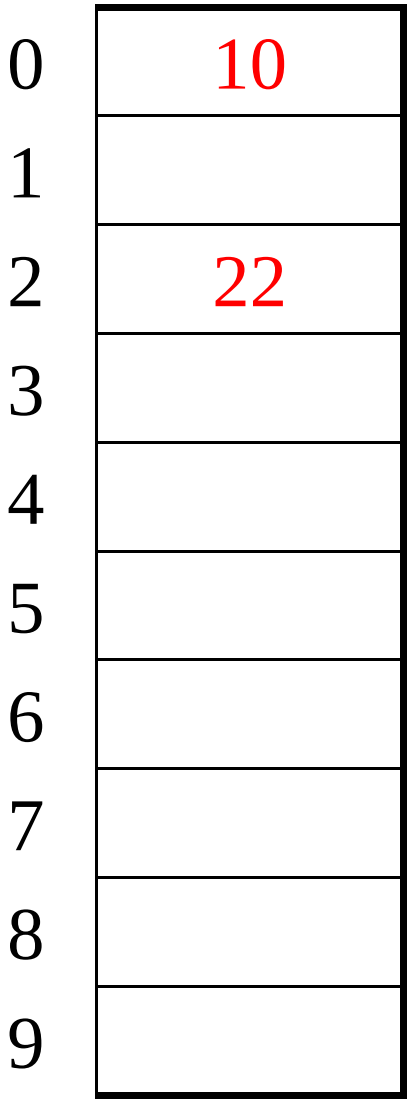
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

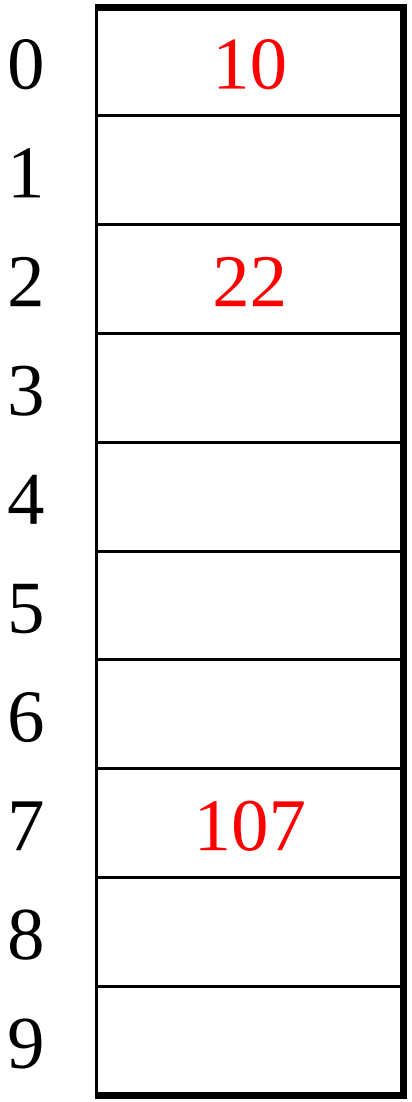
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

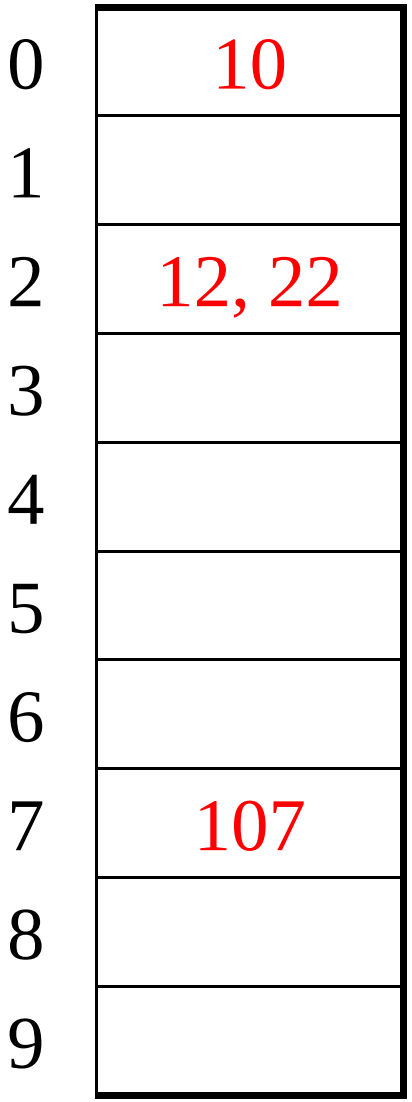
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

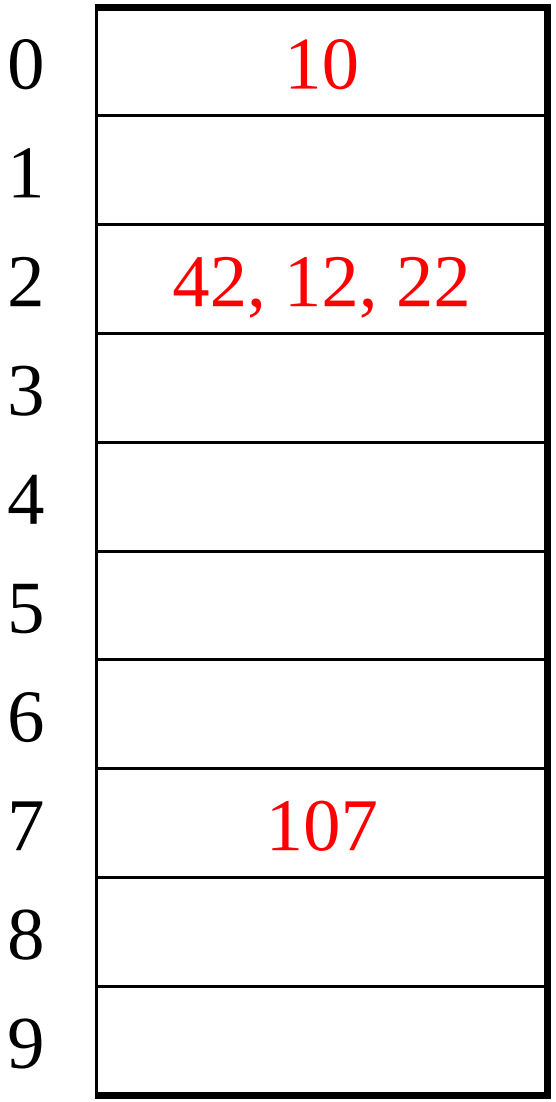
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

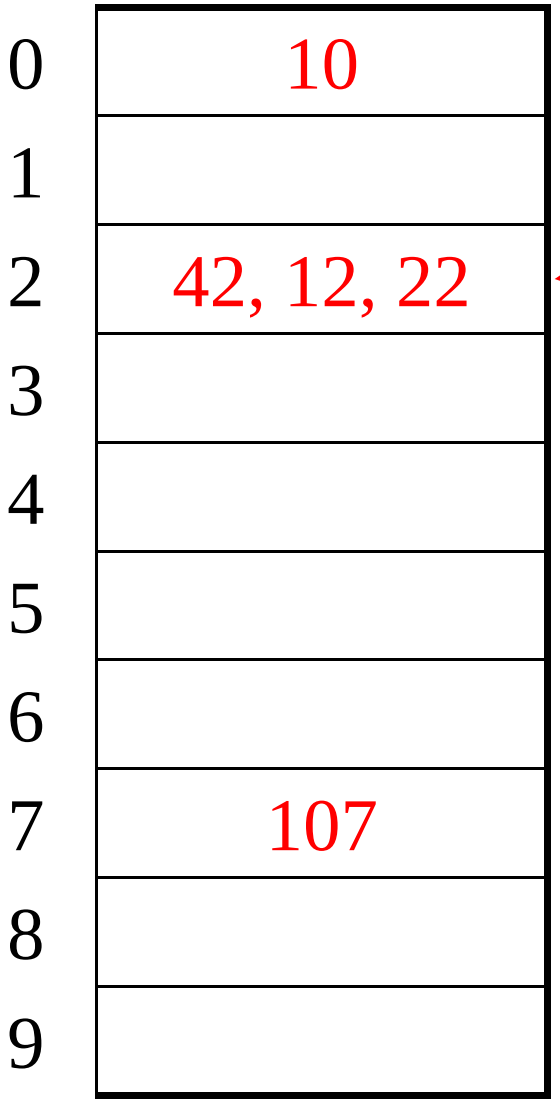
Separate Chaining



Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

Separate Chaining



Thoughts about this?

Our goal is to keep it such that a simple list is good enough

Insert:
10
22
107
12
42

Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket").

Analysis of Separate Chaining

The load factor, λ, of a hash table is

$$\lambda = \frac{\text{N} \quad \leftarrow \text{no. of elements}}{\text{TableSize}}$$

Separate chaining: λ = average # of elems per bucket

Average cost of:

- Unsuccessful find?
- Successful find?
- Insert?

Analysis of Separate Chaining

The load factor, λ, of a hash table is

$$\lambda = \frac{\text{N} \quad \leftarrow \text{no. of elements}}{\text{TableSize}}$$

Separate chaining: λ = average # of elems per bucket

Average cost of:

- Unsuccessful find? λ
- Successful find? $\lambda / 2$
- Insert? λ (assuming you check the item is not already there)