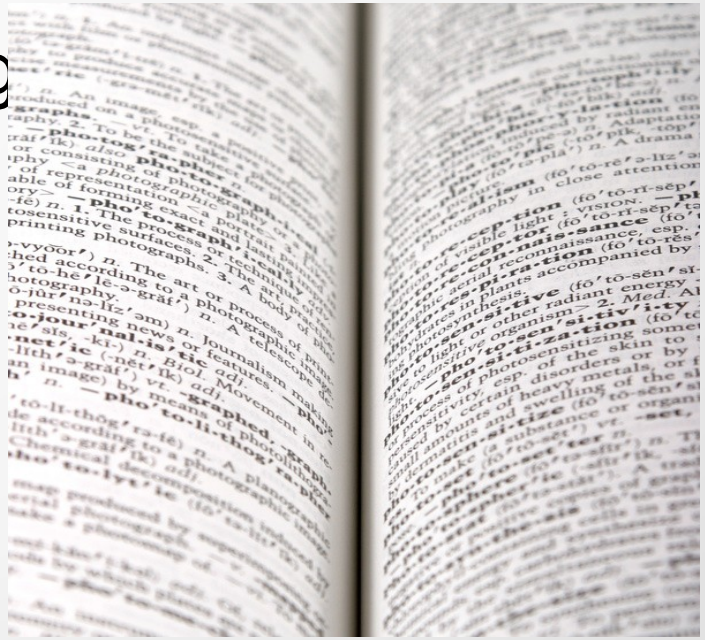
# 19CSE 111:Foundations of Data Structures

## Lecture 6: Dictionaries and Hash Tables

Dr. Vidhya Balasubramanian

# Dictionaries

- Models a searchable collection of key-element items
  - Multiple items with same key allowed
- Main operations include
  - insertion, searching, and deleting
- Applications
  - Telephone directory
  - Mapping student info to roll nos



# Dictionary ADT

- find(k): if the dictionary has an item with key k, returns the position of this item,

  else, returns a null position.
- insertItem(k, o): inserts item o with key k into the dictionary
- removeElement(k): removes the item with key k from the dictionary. Exception of no such element.
- Other functions
  - size(), isEmpty()
  - keys(), Elements()

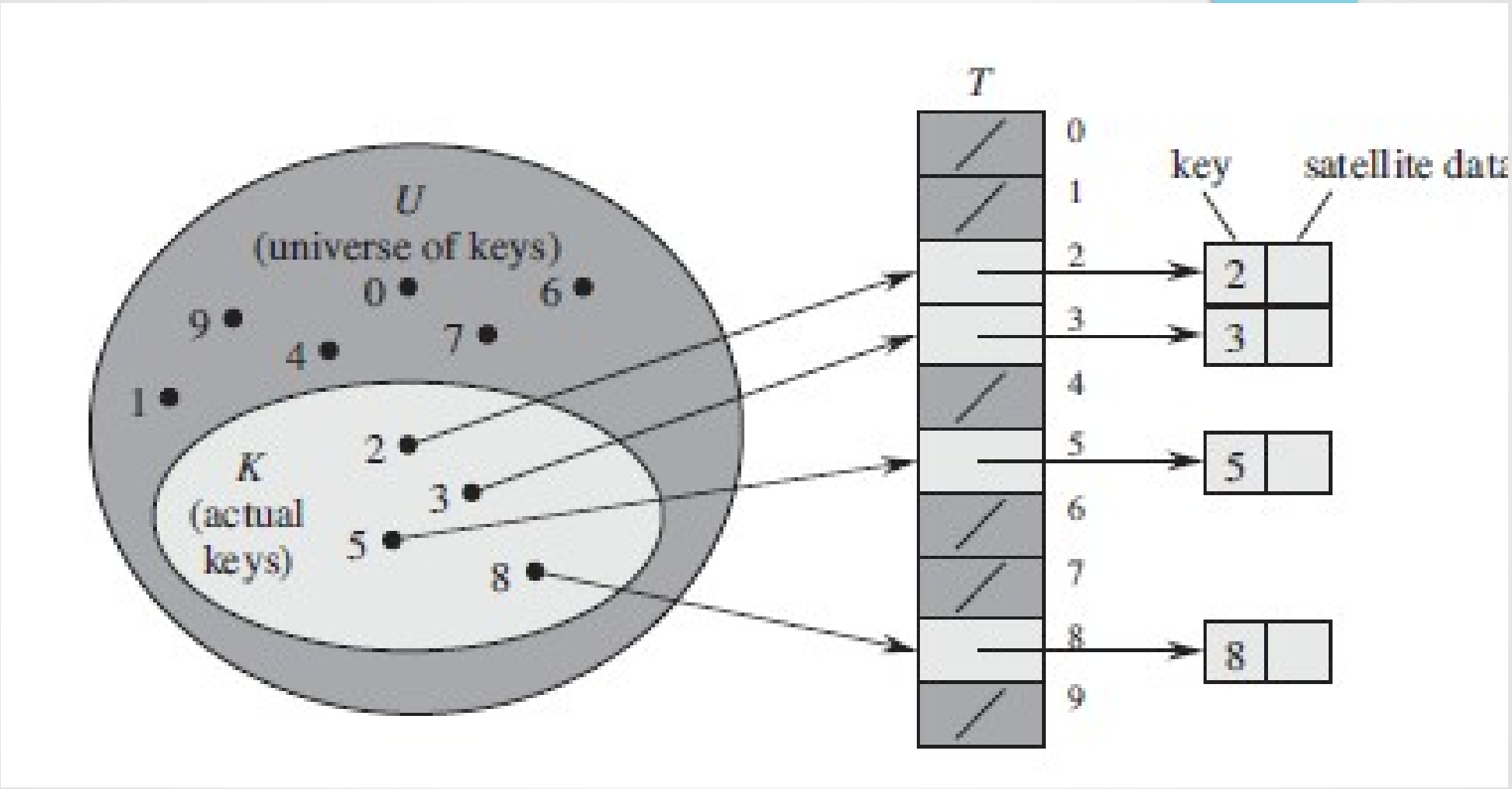
# Dictionaries

- Types
  - Ordered Dictionaries
    - A total order relation is defined on the keys
  - Unordered Dictionaries
    - No order relation is assumed on the keys
    - Only equality testing between keys is used
- Associative Stores
  - When keys are unique, keys are like addresses to the location where the element is stored

# Direct Addressing

- Applied when the number of keys are small and are unique
- Use an array, or direct-address table, denoted by T[0..m-1], in which each position, or slot, corresponds to a key in the universe U
  - Key k is stored in slot k
  - If the set contains no elements, then the slot is empty
- When the universe is large this is impractical
  - Use hashing

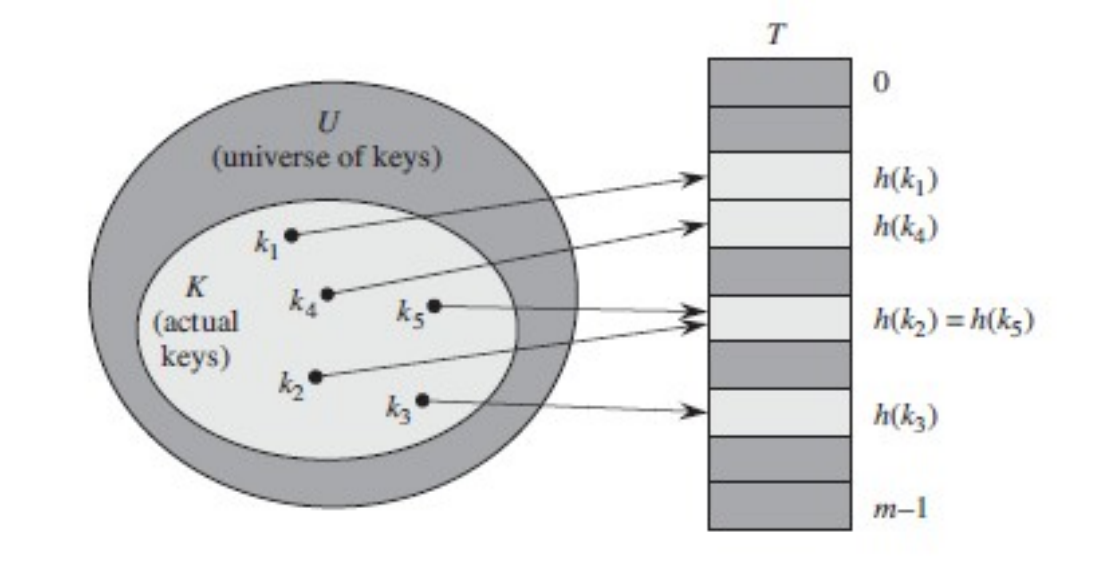
# Direct Addressing




Src: CLRS Textbook

# Hashing

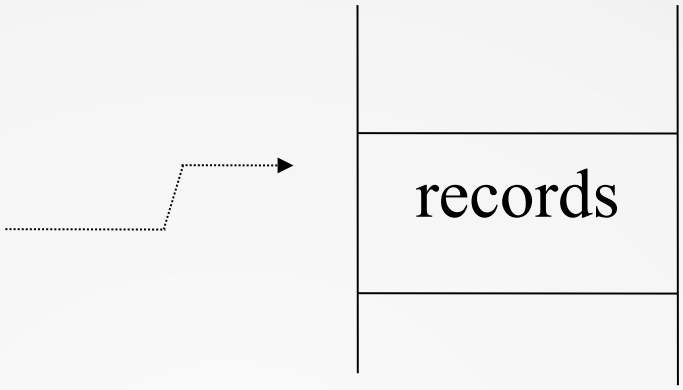
- Element k is stored in slot h(k)
  - Use a hash function to compute slot from the key
  - h maps the universe U of keys into the slots of a hash table T[0..m-1]
  - Size m is much lesser than the |U|



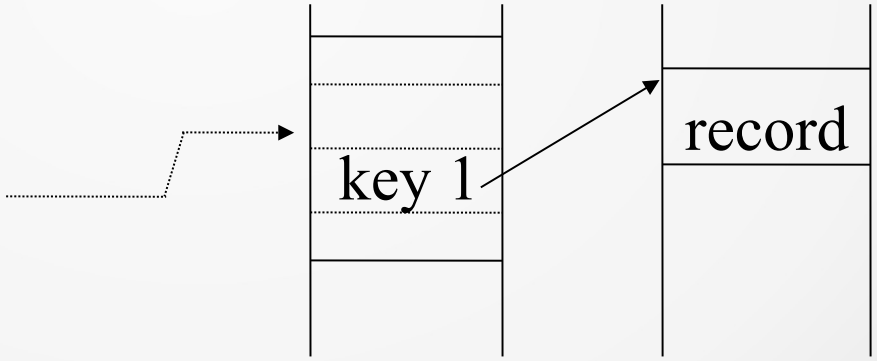

Src: CLRS Text

# Alternatives

(1) key -> h(key)




(2) key -> h(key)



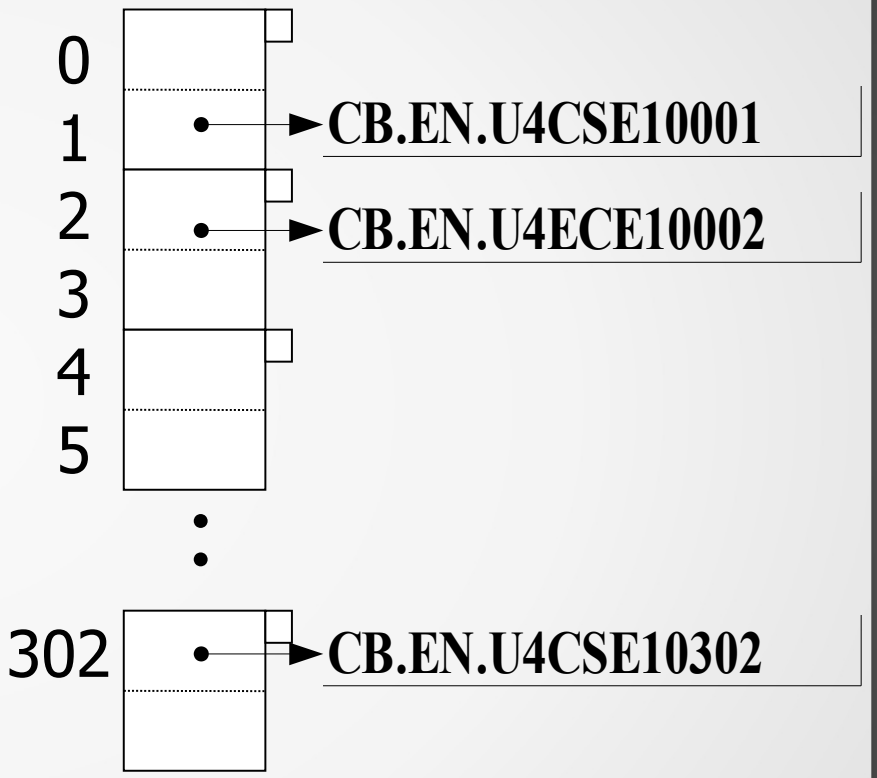

Index

# Hash Function and Hash Tables

- Hash function *h*
  - Maps keys of a given type to integers in a fixed interval [0 to N-1]
  - e.g h(x) = x mod N
    - The integer h(x) is called the hash value of key x
- A hash table for a given key type consists of
  - Hash function h
  - Array (called table) of size N
  - goal is to store item (k, o) at index i = h(k)

# Example

- Design a hash table for a dictionary storing items (Roll No, Name)
- Proposed hash table uses array of size N = 10,000 and the hash function h(x) = last three digits of x



# Hash Functions

- A hash function specified as composition of two functions:
  - Hash code map
    - h1: keys → integers
  - Compression map:
    - h2: integers → [0, N − 1]
  - h(x) = h2(h1(x))
    - The values returned by a hash function are called hash codes or hash values

# Hash Functions: Compression Maps

- A function to map the integer to some fixed range of values
- Division Method
  - h2 (y) = y mod N
  - The size N of the hash table is usually chosen to be a prime
- Multiply, Add and Divide (MAD)
  - h2 (y) = (ay + b) mod N
  - a and b are nonnegative integers such that a mod N ≠ 0
  - Otherwise, every integer would map to the same value b

# Hash Tables

- Insert(k,o): Insert object o with key k
  - Apply hash function h(k) e.g h(k) = k mod n
  - Insert O in bucket pointed by h(k)
- FindElement(k)
  - Let i be the bucket as a result of applying the hashing h(k)
  - Goto bucket i and search for k
- DeleteElement(k)
  - Find the bucket containing element using hash function and remove it

# Collision Handling

- The goal of hashing is to map the keys randomly so that
  - Keys are distributed in the buckets evenly
  - Skews increase the worst case search complexity
- Collisions occur when different elements are mapped to the same cell
  - And if the bucket is full

# Chaining

- Chaining:
  - Add a linked list to end of bucket and add additional elements to the linked list
  - Elements that hash to the same bucket are stored in a linked list
- Performance
  - Simple
  - Requires extra space, and search time increases

# Chaining

INSERT:

h(1) = 1

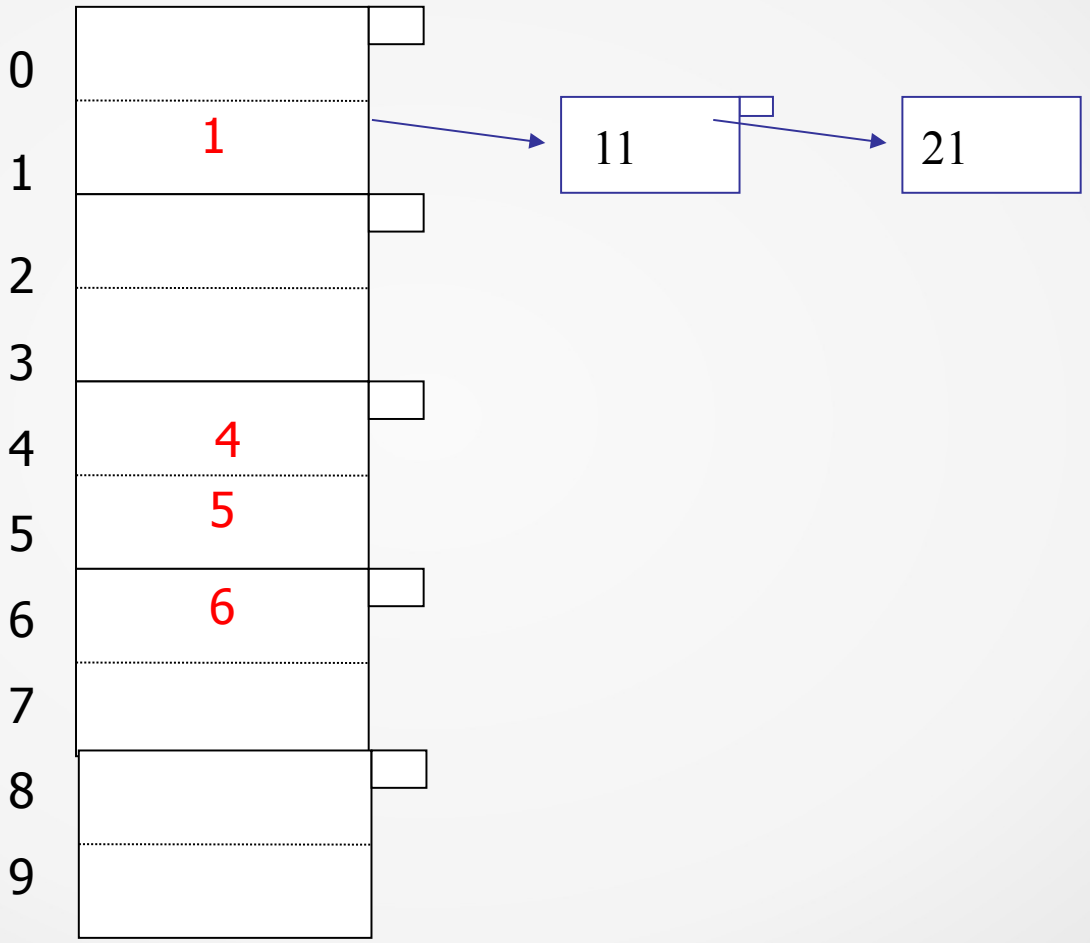
h(6) = 6

h(5) = 5

h(4) = 4

h(11) = 1

h(21) = 1



# Open Addressing

- Chaining requires auxilliary data structures
- Open addressing
  - The colliding item is placed in a different cell of the table
- Linear probing
  - handles collisions by placing the colliding item in the next (circularly) available table cell
  - Each table cell inspected is referred to as a “probe”
  - Try to insert item into bucket A[i], where i = h(k).If collision occurs we try
    - A[(*i*+1) mod *N*], else A[(*i*+2) mod *N*] and so on until an empty bucket is found

# Linear Probing

h=kmod 10

INSERT:

h(1) = 1

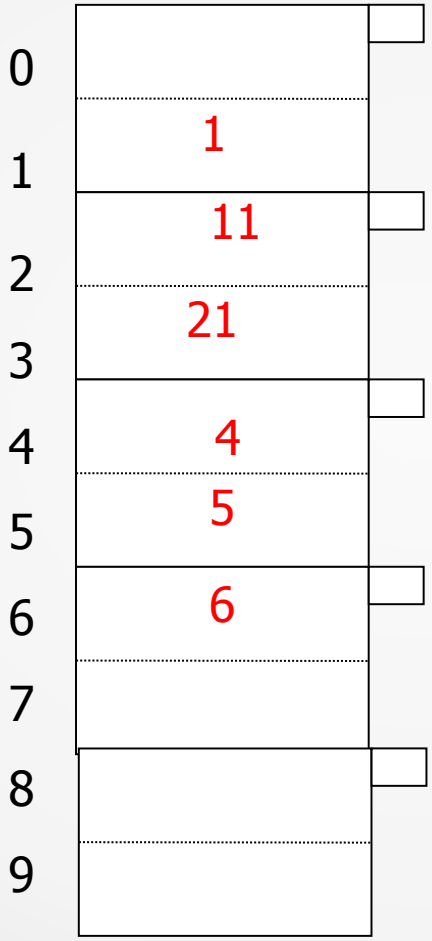
h(6) = 6

h(5) = 5

h(4) = 4

h(11) = 1

h(21) = 1




- **Search**
  - Start at cell h(k)
  - Probe consecutive locations until
    - An item with key k is found, or
    - An empty cell is found, or
    - N cells have been unsuccessfully probed

# Update Operations

- When deleting an item, search becomes complex
  - Elements that have been placed using probing may have to be shifted after each delete so that search is not affected
- To handle insertions and deletions, we introduce a special object, called AVAILABLE, which replaces deleted elements removeElement(k)
- We search for an item with key k
  - If such an item (k, o) is found, we replace it with the special item AVAILABLE and we return the position of this item
  - Else, we return a null position