BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design

Febin.A.Vahab

SESSION 5 -PLAN

Sessions(#)	List of Topic Title	Text/Ref Book/external resource
5	Unordered Dictionary :ADT, Applications Hash Tables: Notion of Hashing and Collision (with a simple vector based hash table)Hash Functions: Properties, Simple hash functions Methods for Collision Handling: Separate Chaining, Notion of Load Factor, Rehashing, Open Addressing [Linear; Quadratic Probing, Double Hash]	T1: 2.5

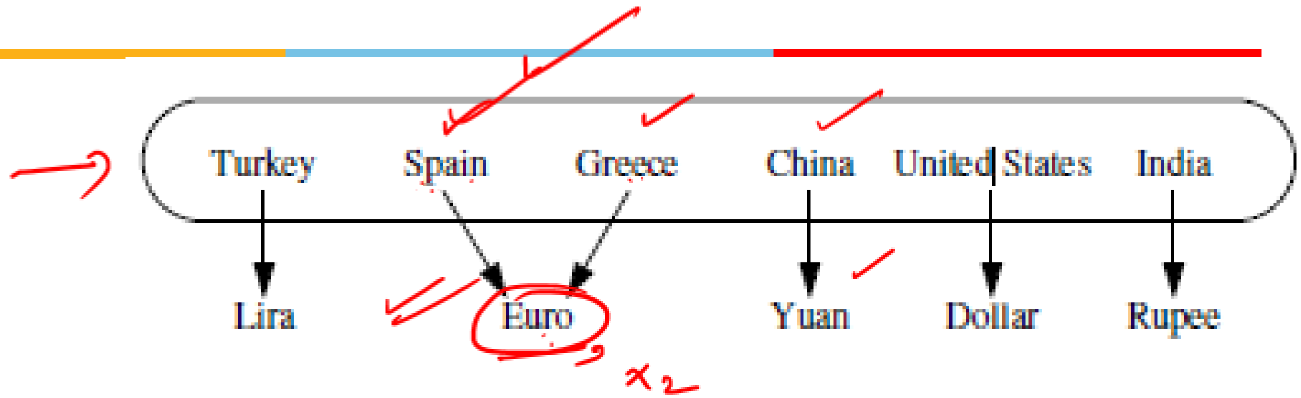
Dictionary ADT

- The dictionary ADT models a searchable collection of key-element items.
- A dictionary stores **key-element pairs (k , e),** which we call **items**, where k is the key and e is the element
- The main operations of a dictionary are searching, inserting, and deleting items
- A key is an identifier that is assigned by an application or user to an associated element.
- *Multiple items with the same key are allowed*
- In cases when keys are unique, the key associated with an object can be viewed as an "address" for that object in memory.

The Dictionary ADT

- For example, in a dictionary storing student records (such as the student's name, address, and course grades), the key might be the **student's ID number**. **(**we would probably want to disallow two students having the same ID).

The Unordered Dictionary ADT



From countries (the keys) to their units of currency (the values).

- Dictionaries use an array-like syntax for indexing such as **currency[Greece]** to access a value associated with a given key **currency[Greece] = New value** to remap it to a new value.
- Unlike a standard array, indices for a dictionary need not be consecutive nor even numeric.

Applications

- The domain-name system (DNS)maps a host name, such as www.bits-pilani.ac.in,to an Internet-Protocol (IP) address.
- A social media site typically relies on a (nonnumeric) username as a key that can be efficiently mapped to a particular user's associated information.
- A computer graphics system may map a color name, such as turquoise , to the triple of numbers that describes the color's RGB (red-green-blue) representation,such as (64,224,208).
- Python uses a dictionary to represent each namespace, mapping an identifying string, such as pi , to an associated object, such as 3.14159.

Applications-HW

- Counting Word Frequencies
  - Consider the problem of counting the number of occurrences of words in a document.
  - A dictionary is an ideal data structure to use here, for we can use words as keys and word counts as values.

Try implementing this using Python Dictionary class.!!!

Dictionary ADT methods:

innovate achieve lead

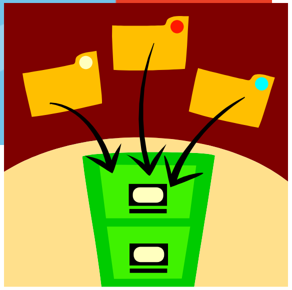
- Dictionary ADT methods:
  - **findElement(k)**: if the dictionary has an item with key k, returns its element, else, returns the special element NO_SUCH_KEY
  - **insertItem(k, e):** Insert an item with element e and key k into D
  - **removeElement(k)**: if the dictionary has an item with key k, removes it from the dictionary and returns its element, else returns the special element NO_SUCH_KEY
  - **size(), isEmpty()**
  - **keys(), elements()-Iterators**

Dictionary ADT methods:

$k \rightarrow x_1, x_2, x_3$

- Special element (NO_SUCH_KEY) is known as a sentinel.
- If we wish to store an item 'e' in a dictionary so that the item is itself its own key, then we would insert e with the method call insertltem(e, e) .
- *findAIIElements(k) - which returns an iterator of all elements with key equal to k*
- *removeAIIElements (k) , which removes from D all the items with key equal to k*

Log Files/Unordered Sequence Implementation



- A log file is a dictionary implemented by means of an **unsorted sequence**
- Often called **audit trail**
- We store the items of the dictionary in a sequence (based on an array or list to store the key-element pairs), in arbitrary order
- The space required for a log file is O(n), since the array data structure can maintain its memory usage to be proportional to its size.

10
110
230
4

5
1987

8
10
0
3

Log Files

- Performance:
  - **Insertion?** insert Item (k, e)
  - **insertItem takes O(1) time** since we can insert the new item at the end of the sequence
  - **Search?? Removal??**
  - **[findElement(k), removeElement(k)]**
  - findElement and removeElement **take O(n) time** since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key
- The log file is effective only for dictionaries of small size or for dictionaries on which insertions are the most common operations, while searches and removals are rarely performed
- (e.g., historical record of logins to a workstation)

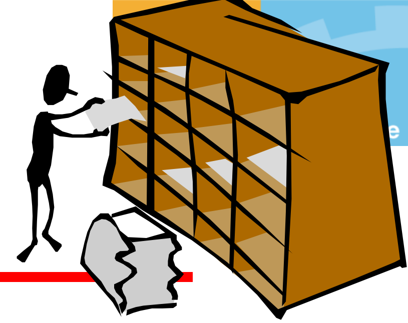
Dictionary Implementation using Hash Tables

- A hash table for a given key type consists of
  - Array (called table) of size N-(Bucket Array)
  - Hash function h

- **When implementing a dictionary with a hash table, the goal is to store item (k, e) at index i = h(k)**

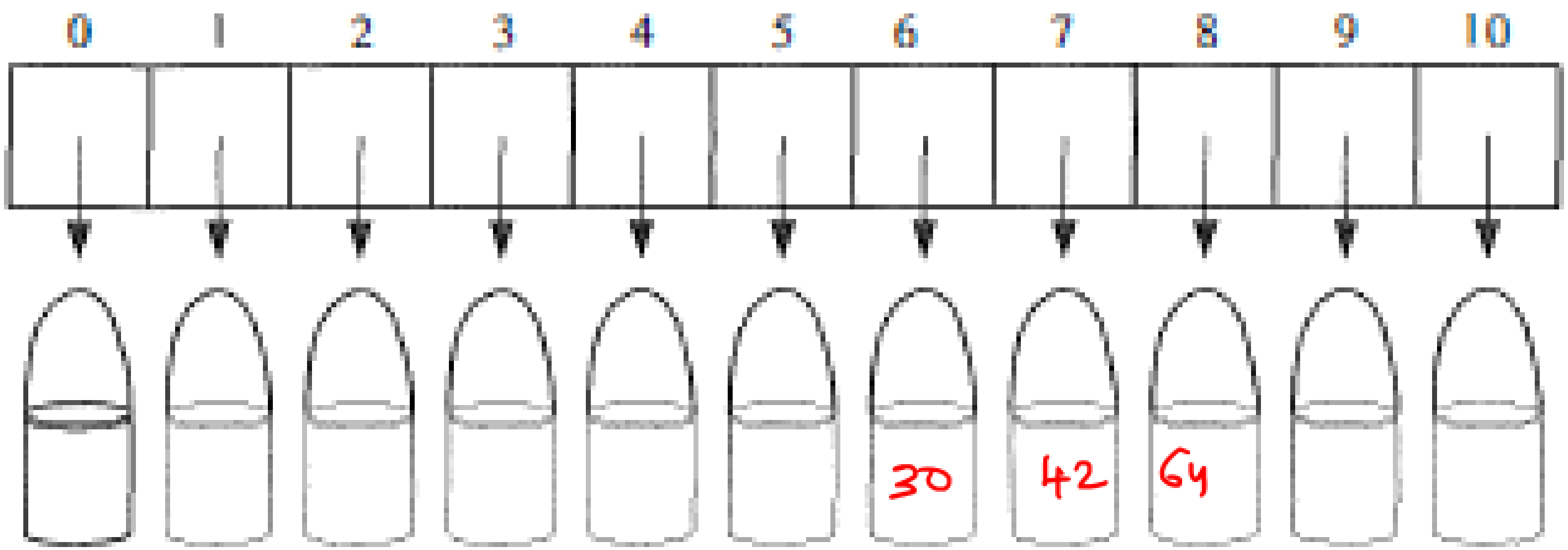
$A[k]$ $A[h(k)] \leftarrow e$

Bucket Arrays



- A bucket array for a hash table is an array A of size N, where each cell of A is thought of as a "bucket" (that is, a container of key-element pairs)
- Integer N defines the capacity of the array.
- **An element e with key k is simply inserted into the bucket A [k].** A [h(k)]
- Any bucket cells associated with keys not present in the dictionary are assumed to hold the special NO_SUCH _KEY object.

Bucket Arrays



The bucket for items with key = 6

h(k)

0
1
2
3
4

A
B
C
D

Bucket Arrays

- **If keys are not unique**, then two different elements may be mapped to the same bucket in A .
- A ***collision*** has occurred.
- If each bucket of A can store only a single element, then we cannot associate more than one element with a single bucket
- problem in the case of collisions.
- There are ways of dealing with collisions
- The best strategy is to try to avoid them in the first place

Bucket Arrays-Analysis

- **If keys are unique**, then collisions are not a concern, and searches, insertions, and removals in the hash table take
- **worst-case time O(1) .**
- **Uses O(N) space**

$A[h(k)] \rightarrow e$

Dictionary Implementation using Hash Tables-Motivation

- The bucket array requires keys be **unique integers** in the **range [0 ,N - 1]** , which is often not the case

N

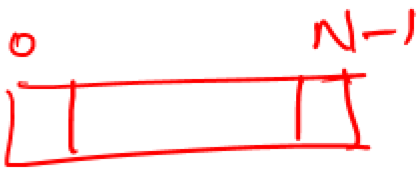
- There are two challenges in extending this framework to the more general setting
- **What can we do if we have at most 100 entries with integer keys but the keys are in range 0 to 1,000,000,000 ?**
- **What can we do if keys are not integers ?**

Bucket Arrays-Analysis

- What we can do???
- Define the hash table data structure to consist of a bucket array together with a "good" mapping from our keys to **1. integers,**

  2.in the range [0, N - 1]

N



Hash Functions

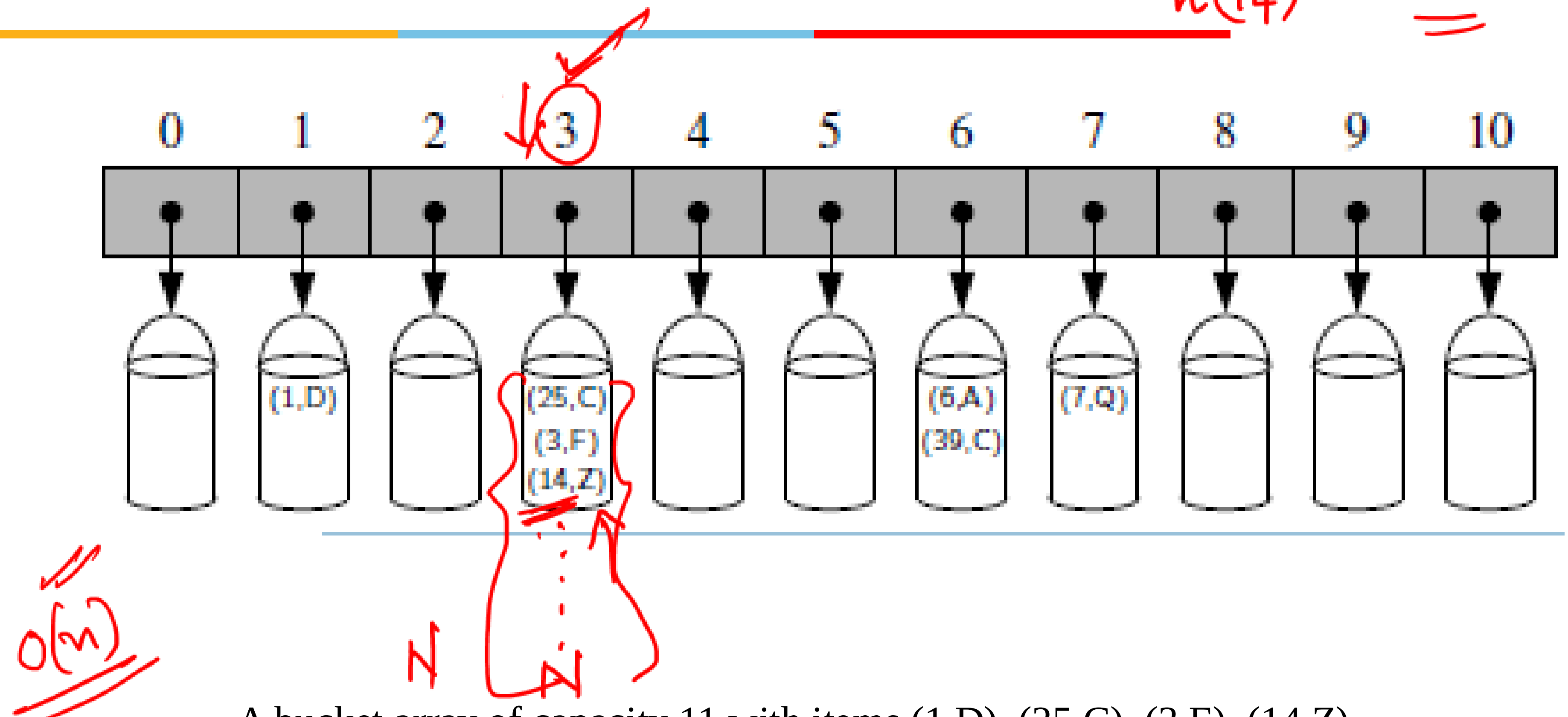
- A hash function h maps keys of a given type to integers in a fixed interval [0, N - 1] N
- Now, bucket array method can be applied to arbitrary keys
- Example:

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x} \bmod \boldsymbol{N}$$

  is a hash function for integer keys
- The integer h(x) is called **the hash value of key x.**

Bucket Arrays with a hash function

$h(14) = 3$



A bucket array of capacity 11 with items (1,D), (25,C), (3,F), (14,Z), (6,A), (39,C), and (7,Q), using a simple hash function

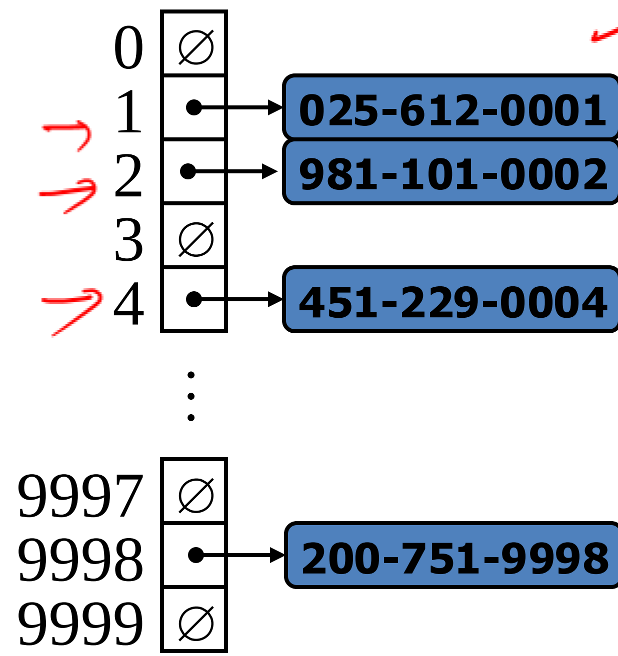
Hash Functions

- **Main Idea**
- Use the hash function value, h(k) , as an index to bucket array, A, instead of the key k (which is most likely inappropriate for use as a bucket array index).
- That is, store the item(k, e) in the bucket **A [h(k)]** .
- A hash function is "good" if it maps the keys in our dictionary so as to **minimize collisions** as much as possible.

Hash Tables-Example

- We design a hash table for a dictionary storing items (SSN, Name), where SSN (social security number) is a nine-digit positive integer
- Hash table uses an array of size

$\boldsymbol{N} = 10{,}000$ and the hash function $\boldsymbol{h}(\boldsymbol{x}) =$ last four digits of $\boldsymbol{x}$



Evaluation of a hash function, h(k)

- A hash function ,h(k) , is usually specified as the composition of two functions:

SPOT

Hash code map:

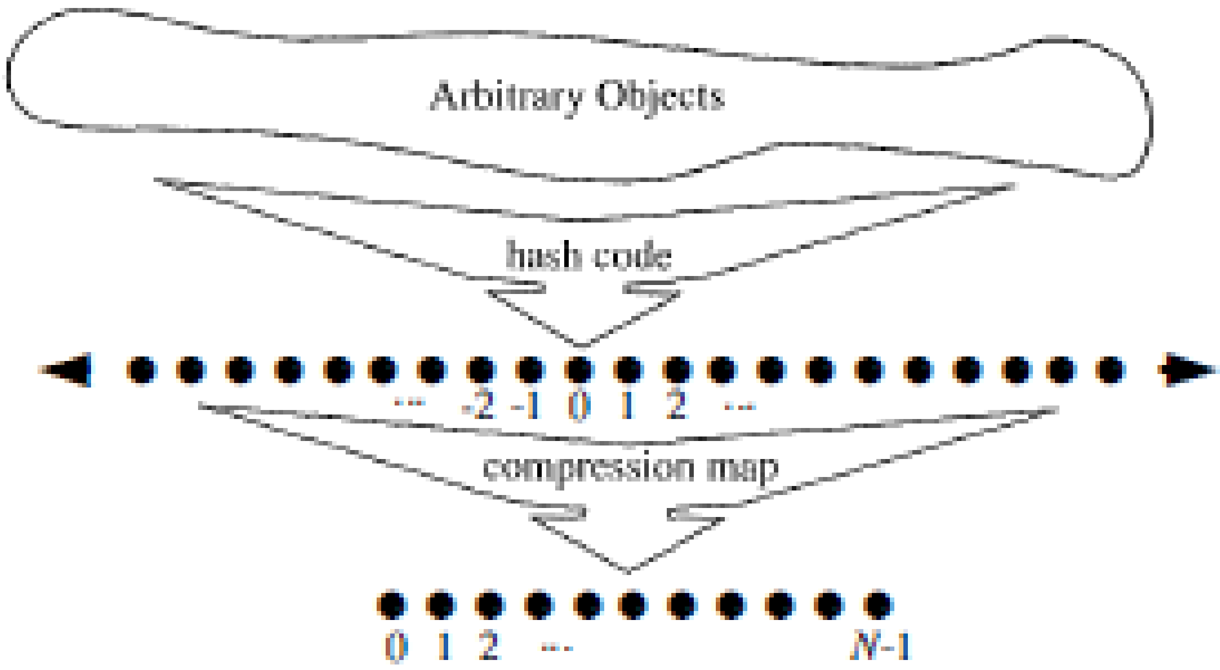
h1: keys → integers[mapping the key k to an integer]

Compression map:

h2: integers → [0, N - 1]

[mapping the hash code to an integer within the range of indices of a bucket array]

Evaluation of a hash function



Evaluation of a hash function, h(k)

- The hash code map is applied first, and the compression map is applied next on the result, i.e.,
  $\boxed{h(x)} = h_2(h_1(x))$
- The goal of the hash function is to "disperse" the keys in an apparently random way.

Hash code in python

In [12]:
```
print('Hash for 220 is:', hash(220))

# hash for decimal
print('Hash for 220.34 is:',hash(220.34))

# hash for string
print('Hash for Data is:', hash('Data'))
```

```
Hash for 220 is: 220
Hash for 220.34 is: 783986623132664028
Hash for Data is: 2539907043859605924
```

In [13]:
```
id(220.34)
```

Out[13]:
```
3207284319792
```

In [14]:
```
id('Data')
```

Out[14]:
```
3207240693944
```

Compression Maps

- **Division:**
  - **Let y=h_1(x)//integer hash code for a key object k**
  - h_2 (y) = $\boxed{\text{y mod N}}$
  - The size N of the hash table is usually chosen to be a prime
  - **The reason has to do with number theory and is beyond the scope of this course!!!**

k = [200, 205, 210, 215 . . . . . . 600]

N = 100

N = 101

h(k) = k mod N

h(200) = 200 mod 100 = 0

THANK YOU!

BITS Pilani
Hyderabad Campus