

COL106: Data Structures and Algorithms

Lecture 10

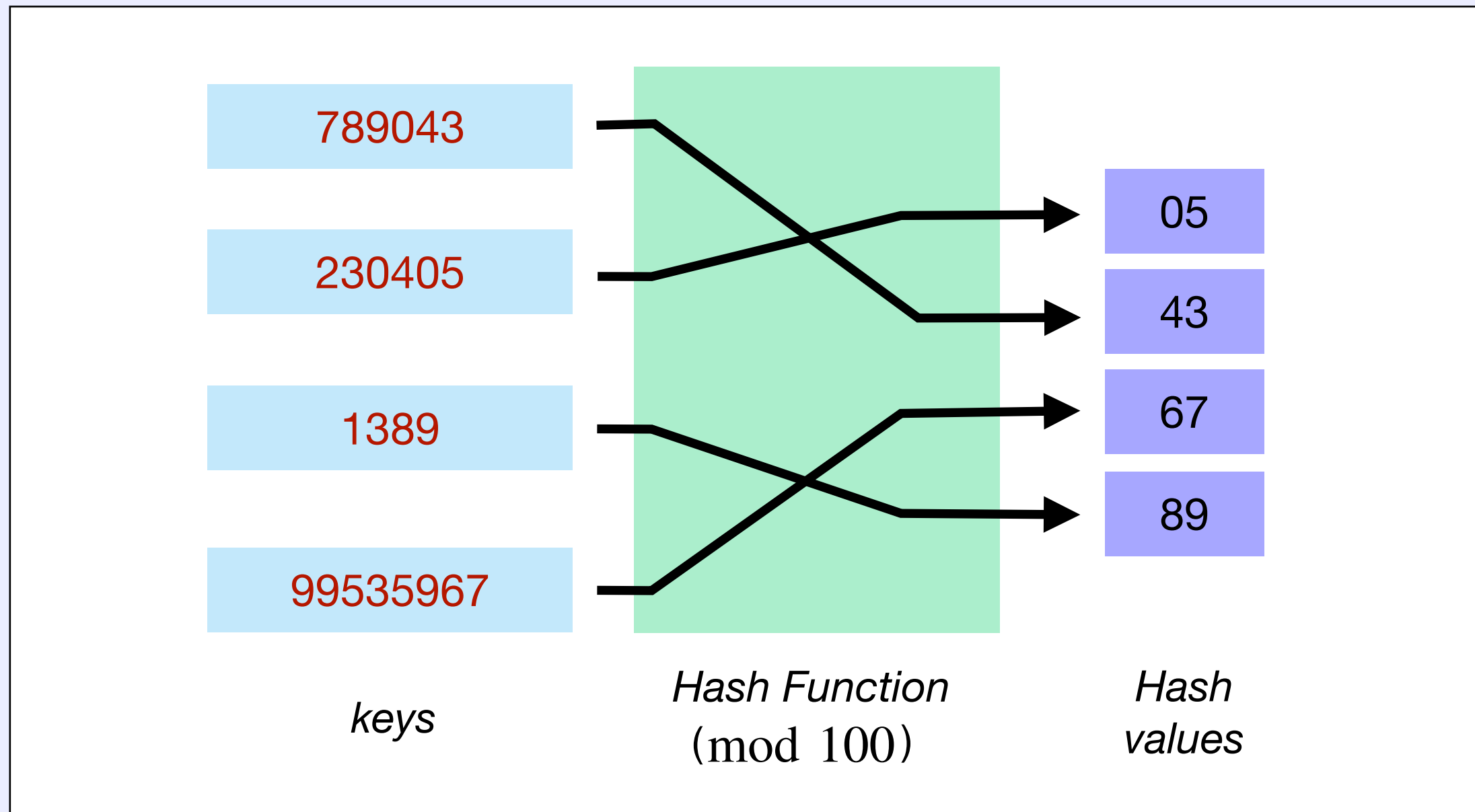
Keerti Choudhary
Department of Computer Science, IIT Delhi

Announcements

- Assignment submissions:
 - **No late submissions are allowed under any circumstances.**
 - We are already considering best $k - 1$ scores out of k assignments.
- Assignment 2 is uploaded on Piazza.

Hashing

A process of mapping keys or strings of arbitrary size to shorter, fixed-length outputs.



Hash Table

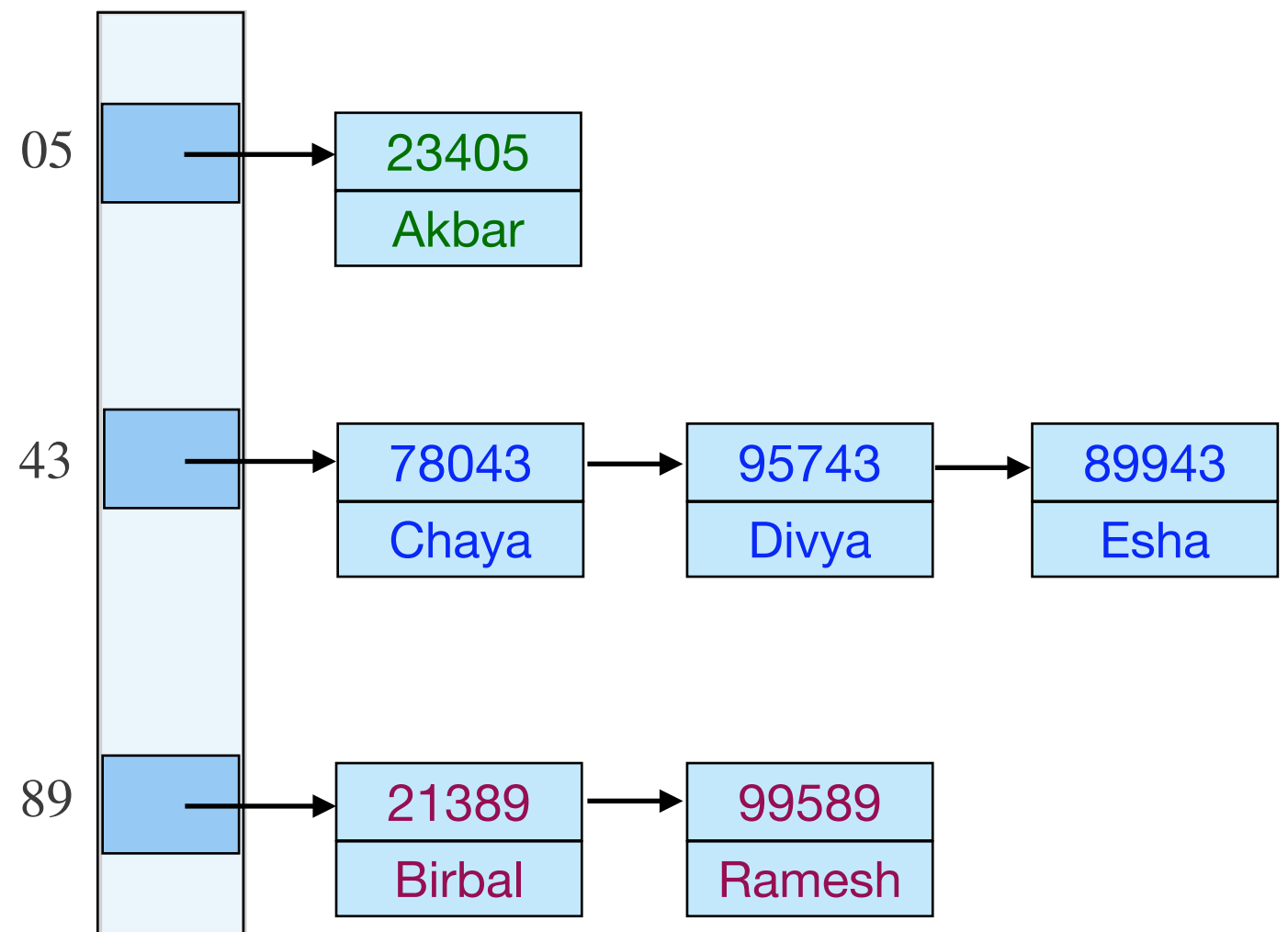
- A data-structure that uses a hash-function H to store keys and their associated elements.
- **Techniques for collision resolution:** chaining, secondary hash tables, etc.

Hash Function

$$H : k \rightarrow k \bmod 100$$

23405 Akbar	21389 Birbal
78043 Chaya	95743 Divya
89943 Esha	99589 Ramesh

$n = 6$ records



Hash Table

- A data-structure that uses a hash-function H to store keys and their associated elements.
- **Techniques for collision resolution:** chaining, secondary hash tables, etc.

Search(Key k):

1. Compute $i = H(k)$
2. Scan linked-list at $T[i]$ to verify if $k \in T[i]$

Notations:

- m = size of Hash table
 n = Number of keys
 $[1, L]$ = Universe when keys are integers.

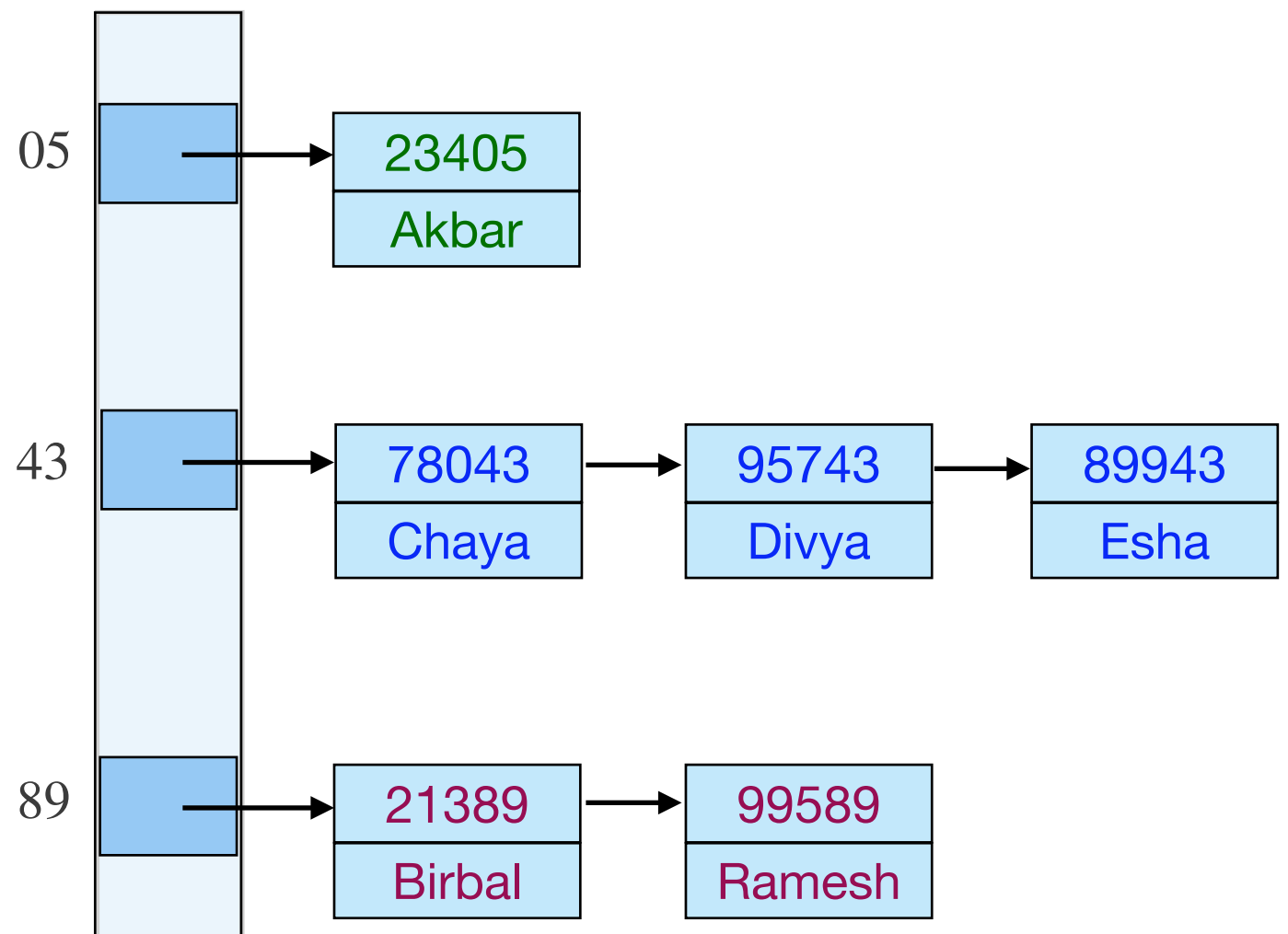


Table T

What should be a Good Hash Function?

- A good hash function will be one which
 - Distributes keys uniformly
 - Easy to compute
 - Takes less space
- An ideal hash function:
 - Should yield list length of size $\alpha = n/m$.
 - Here,
 - n = Number of keys
 - m = size of Hash table
- **Remark:** There is NO hash function that works for all key spaces.
(Homework: why?)

Simple Hash Function: Division

- $H(k) = k \bmod m$
- Bad if all keys are from set $\{m, 2m, 3m, \dots\}$.
- Good if the keys are chosen **randomly**.

Lemma (Lec 9): If keys $k_1, \dots, k_n$ are chosen uniformly randomly from universe $U = [1, L]$, then for any $j \in [0, m - 1]$ we have $E(|T[j]|) = n/m$.

Expected Search time: $O(1 + n/m)$ Space: $O(n + m)$

Expected Deletion time: $O(1 + n/m)$

Expected Insertion time: $O(1)$

Parameter choice: So, m can be chosen to be $\Theta(n)$.

Hash Function: MAD

- More sophisticated hash function: Multiply-Add-and-Divide (or “MAD”) method.
- Hash of a key $k \in [1, L]$ is defined as

$$H(k) = ((ak + b) \bmod p) \bmod m$$

Choosing a random Hash function is beneficial when keys are not random.

where, p is any prime larger than L , and $a \neq 0$, b are **random** integers in range $[0, p - 1]$.

Proof of this claim is not part of syllabus.

- **Advantages:**
 - **Claim:** For any key k , and integer $i \in [0, m - 1]$, probability that $H(k) = i$ is $1/m$.
 - If there are n keys then, we have $E(|T[j]|) = n/m$.
 - For $m = \Theta(n)$, we get that time for Search, Insertion, Deletion of a single key is $O(1)$.
- If n changes with time due to Insertions, Deletions then we must do periodic rebuilding.

Hash Function for Strings

- A hash function for strings comprises of two parts:
 - Hash-Code: $string \rightarrow integer$
 - Compression Function: $integer \rightarrow hash-value$.
- Advantage of separating into two components:
 - Hash-code is independent of hash table size m .
 - This is particularly important when we need to dynamically change hash table size.

Polynomial Hash Code

- For an r -length string $x = (x_0, x_1, \dots, x_{r-1})$ the polynomial hash code is defined as

$$A(x) = (x_0 p^{r-1} + x_1 p^{r-2} + \dots + x_{r-2} p + x_{r-1})$$

where, p is a prime, and

x_i is integer corresponding to i^{th} character of string x .

- Character-mapping:
 - If we ONLY deal with letters $[a - z]$ then a possible mapping is $a \rightarrow 1, \dots, z \rightarrow 26$
 - If we deal with ALL characters, then we can choose ASCII map:
 $A \rightarrow 65, \dots, Z \rightarrow 90, \dots, a \rightarrow 97, \dots, z \rightarrow 122$
- Choice of prime:
 - If p is larger than integers corresponding to character-map then hash code is *injective*.

Polynomial Hash Code

- For an r -length string $x = (x_0, x_1, \dots, x_{r-1})$ the polynomial hash code is defined as

$$A(x) = (x_0 p^{r-1} + x_1 p^{r-2} + \dots + x_{r-2} p + x_{r-1})$$

where, p is a prime, and

x_i is integer corresponding to i^{th} character of string x .

- Horner's method for polynomial evaluation:

$$x_{r-1} + p(x_{r-2} + p(x_{r-3} + \dots + p(x_2 + p(x_1 + px_0))\dots))$$

- After converting string to integer we can use any of the earlier described hash function for integers.
- A commonly used function is : $\text{mod } m$, for a relatively large prime m .
Example: $m = 1,000,000,007$