Search algorithms:Hash Tables

Bijendra Nath Jain

bnjain@iiitd.ac.in

Partly based on slides prepared by Shweta Agarwal (IITD, now at IITM) & Amit Kumar (IITD), and Linda Shapiro (UWash), Douglas W. Harder (Uwaterloo)

Outline of next 9 lectures

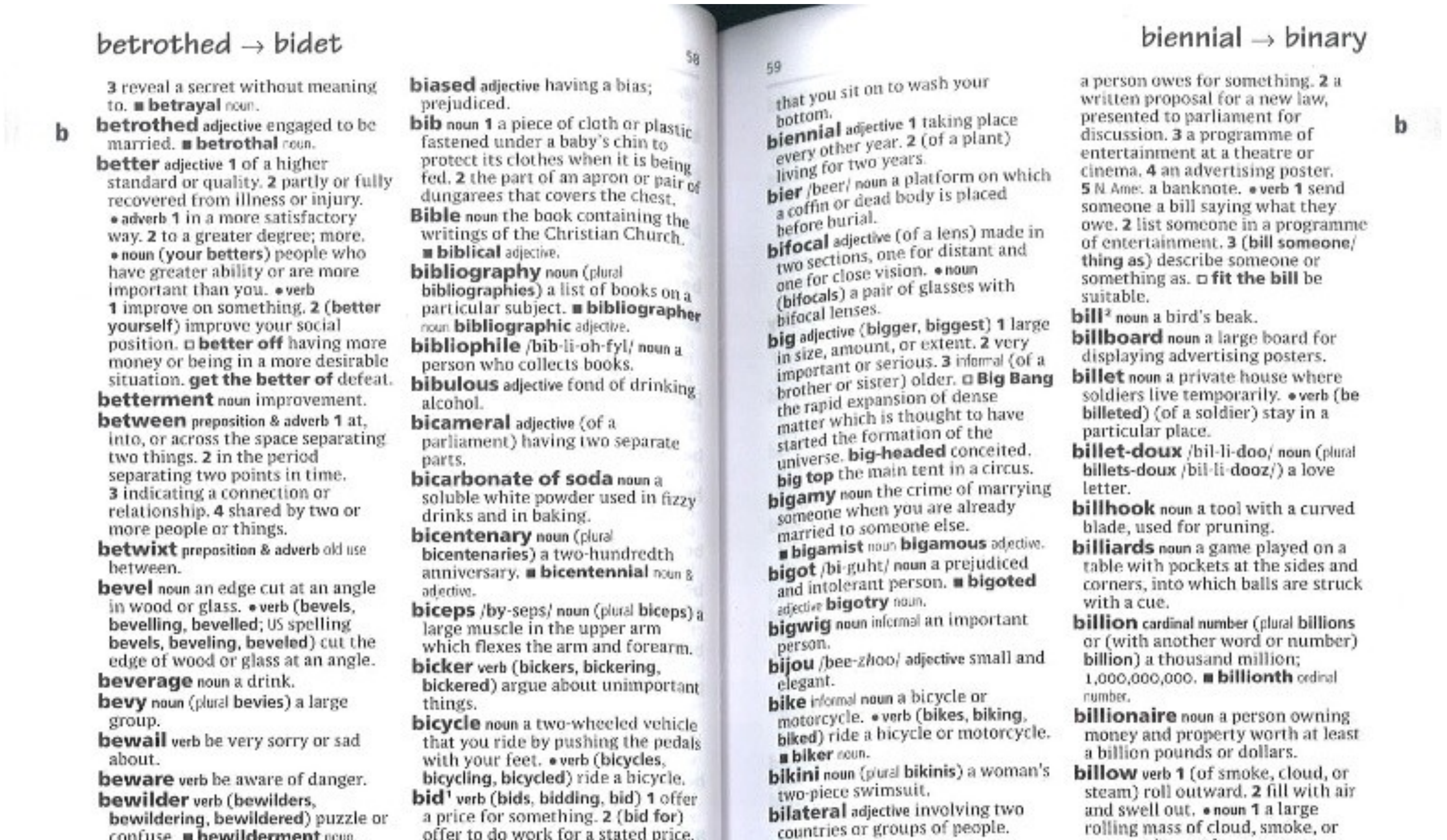
- Graphs:
  - Undirected graphs
  - Directed graphs
  - (Directed) acyclic graphs (or DAGs)
  - Sparse graphs
  - Weighted graphs
- Graph applications
- Representation of graphs:
  - Adjacency matrix
  - Linked lists
- Algorithms:
  - Traversal algorithms:
    - BFS
    - DFS
  - Topological sort
  - Minimum spanning trees
  - Dijkstra's Shortest path
    - One-to-one
    - One-to-many
    - Many-to-many
- Search algorithms
  - Hash Tables

Some applications of search algorithms

- Applicable to any/every kind of data storage and retrieval
  - Dictionaries
  - Academic records of students
  - Pharmacy
  - Super market: goods
  - Compilers
  - Etc.

Some applications of search algorithms

- Applicable to any/every kind of data storage and retrieval
  - Academic records of students
  - Dictionaries
  - Super market: goods
  - Etc.



betrothed → bidet

58

b

3 reveal a secret without meaning to. ■ **betrayal** noun.
betrothed adjective engaged to be married. ■ **betrothal** noun.
better adjective 1 of a higher standard or quality. 2 partly or fully recovered from illness or injury. • adverb 1 in a more satisfactory way. 2 to a greater degree; more. • noun (**your betters**) people who have greater ability or are more important than you. • verb 1 improve on something. 2 (**better yourself**) improve your social position. □ **better off** having more money or being in a more desirable situation. **get the better of** defeat.
betterment noun improvement.
between preposition & adverb 1 at, into, or across the space separating two things. 2 in the period separating two points in time. 3 indicating a connection or relationship. 4 shared by two or more people or things.
betwixt preposition & adverb old use between.
bevel noun an edge cut at an angle in wood or glass. • verb (**bevels, bevelling, bevelled**; US spelling **bevels, beveling, beveled**) cut the edge of wood or glass at an angle.
beverage noun a drink.
bevy noun (plural **bevies**) a large group.
bewail verb be very sorry or sad about.
beware verb be aware of danger.
bewilder verb (**bewilders, bewildering, bewildered**) puzzle or confuse. ■ **bewilderment** noun

biased adjective having a bias; prejudiced.
bib noun 1 a piece of cloth or plastic fastened under a baby's chin to protect its clothes when it is being fed. 2 the part of an apron or pair of dungarees that covers the chest.
Bible noun the book containing the writings of the Christian Church. ■ **biblical** adjective.
bibliography noun (plural **bibliographies**) a list of books on a particular subject. ■ **bibliographer** noun **bibliographic** adjective.
bibliophile /bib-li-oh-fyl/ noun a person who collects books.
bibulous adjective fond of drinking alcohol.
bicameral adjective (of a parliament) having two separate parts.
bicarbonate of soda noun a soluble white powder used in fizzy drinks and in baking.
bicentenary noun (plural **bicentenaries**) a two-hundredth anniversary. ■ **bicentennial** noun & adjective.
biceps /by-seps/ noun (plural **biceps**) a large muscle in the upper arm which flexes the arm and forearm.
bicker verb (**bickers, bickering, bickered**) argue about unimportant things.
bicycle noun a two-wheeled vehicle that you ride by pushing the pedals with your feet. • verb (**bicycles, bicycling, bicycled**) ride a bicycle.
bid¹ verb (**bids, bidding, bid**) 1 offer a price for something. 2 (**bid for**) offer to do work for a stated price.

59

biennial → binary

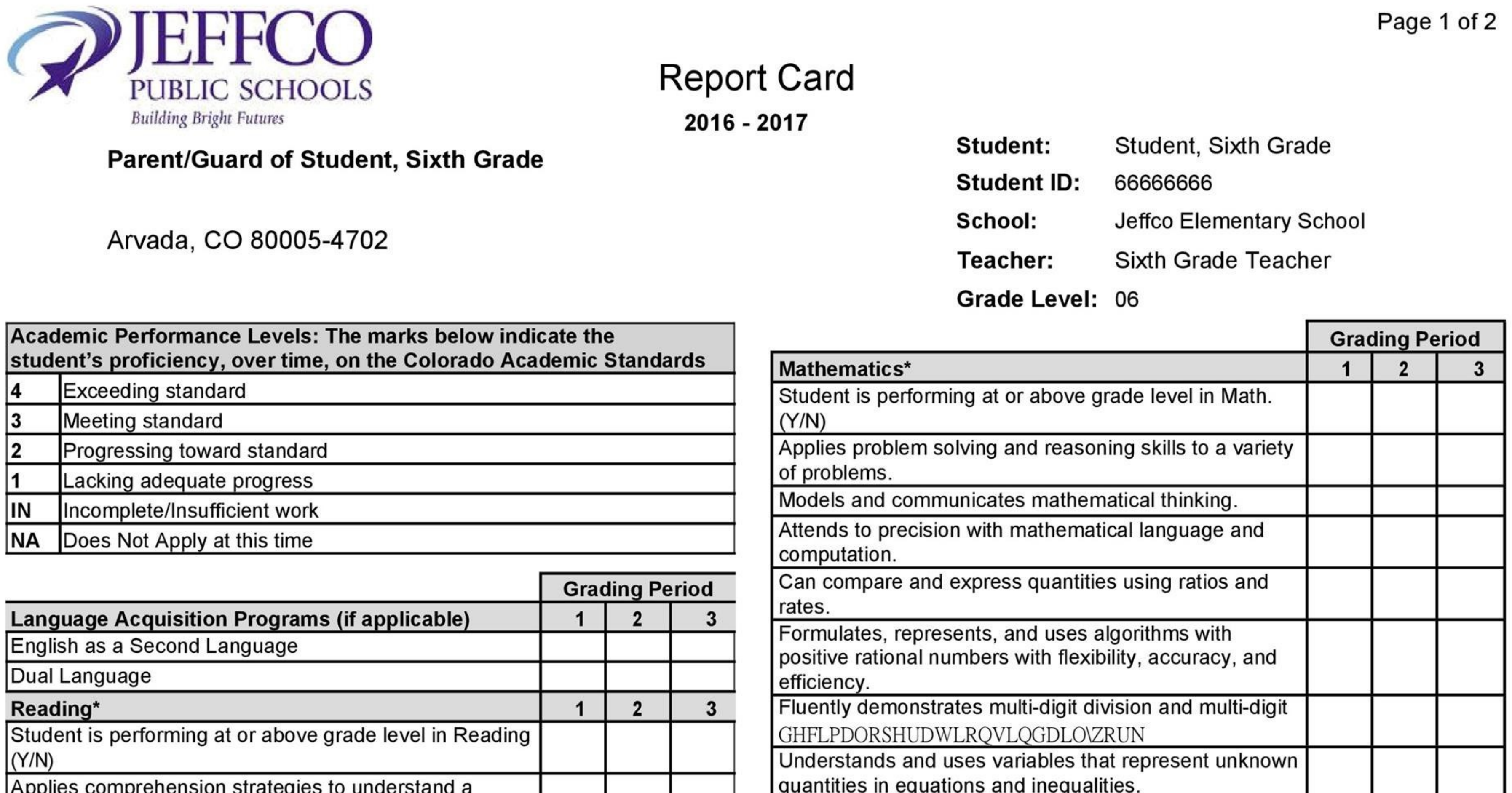
b

that you sit on to wash your bottom.
biennial adjective 1 taking place every other year. 2 (of a plant) living for two years.
bier /beer/ noun a platform on which a coffin or dead body is placed before burial.
bifocal adjective (of a lens) made in two sections, one for distant and one for close vision. • noun (**bifocals**) a pair of glasses with bifocal lenses.
big adjective (**bigger, biggest**) 1 large in size, amount, or extent. 2 very important or serious. 3 informal (of a brother or sister) older. □ **Big Bang** the rapid expansion of dense matter which is thought to have started the formation of the universe. **big-headed** conceited. **big top** the main tent in a circus.
bigamy noun the crime of marrying someone when you are already married to someone else. ■ **bigamist** noun **bigamous** adjective.
bigot /bi-guht/ noun a prejudiced and intolerant person. ■ **bigoted** adjective **bigotry** noun.
bigwig noun informal an important person.
bijou /bee-zhoo/ adjective small and elegant.
bike informal noun a bicycle or motorcycle. • verb (**bikes, biking, biked**) ride a bicycle or motorcycle. ■ **biker** noun.
bikini noun (plural **bikinis**) a woman's two-piece swimsuit.
bilateral adjective involving two countries or groups of people.

a person owes for something. 2 a written proposal for a new law, presented to parliament for discussion. 3 a programme of entertainment at a theatre or cinema. 4 an advertising poster. 5 N. Amer. a banknote. • verb 1 send someone a bill saying what they owe. 2 list someone in a programme of entertainment. 3 (**bill someone/thing as**) describe someone or something as. □ **fit the bill** be suitable.
bill² noun a bird's beak.
billboard noun a large board for displaying advertising posters.
billet noun a private house where soldiers live temporarily. • verb (**be billeted**) (of a soldier) stay in a particular place.
billet-doux /bil-li-doo/ noun (plural **billets-doux** /bil-li-dooz/) a love letter.
billhook noun a tool with a curved blade, used for pruning.
billiards noun a game played on a table with pockets at the sides and corners, into which balls are struck with a cue.
billion cardinal number (plural **billions** or (with another word or number) **billion**) a thousand million; 1,000,000,000. ■ **billionth** ordinal number.
billionaire noun a person owning money and property worth at least a billion pounds or dollars.
billow verb 1 (of smoke, cloud, or steam) roll outward. 2 fill with air and swell out. • noun 1 a large rolling mass of cloud, smoke, or

Some applications of search algorithms

- Applicable to any/every kind of data storage and retrieval
  - Academic records of students
  - Dictionaries
  - Pharmacy
  - Super market: goods



JEFFCO PUBLIC SCHOOLS
Building Bright Futures

Page 1 of 2

Report Card

2016 - 2017

Parent/Guard of Student, Sixth Grade

Arvada, CO 80005-4702

Student:	Student, Sixth Grade
Student ID:	66666666
School:	Jeffco Elementary School
Teacher:	Sixth Grade Teacher
Grade Level:	06

Academic Performance Levels: The marks below indicate the student's proficiency, over time, on the Colorado Academic Standards	
4	Exceeding standard
3	Meeting standard
2	Progressing toward standard
1	Lacking adequate progress
IN	Incomplete/Insufficient work
NA	Does Not Apply at this time

	Grading Period		
Language Acquisition Programs (if applicable)	**1**	**2**	**3**
English as a Second Language			
Dual Language			
Reading*	**1**	**2**	**3**
Student is performing at or above grade level in Reading (Y/N)			
Applies comprehension strategies to understand a			

	Grading Period		
Mathematics*	**1**	**2**	**3**
Student is performing at or above grade level in Math. (Y/N)			
Applies problem solving and reasoning skills to a variety of problems.			
Models and communicates mathematical thinking.			
Attends to precision with mathematical language and computation.			
Can compare and express quantities using ratios and rates.			
Formulates, represents, and uses algorithms with positive rational numbers with flexibility, accuracy, and efficiency.			
Fluently demonstrates multi-digit division and multi-digit GHFLPDORSHUDWLRQVLQGDLO\ZRUN			
Understands and uses variables that represent unknown quantities in equations and inequalities.			

Some applications of search algorithms

- Applicable to any/every kind of data storage and retrieval
  - Academic records of students
  - Dictionaries
  - Super market: goods
  - Pharmacy
  - Etc.

Extended-Release Capsules 50 mg
Metolar XR-50
Each hard gelatin extended-release capsule contains:
47.5 mg of Metoprolol Succinate IP equivalent
to 50 mg of Metoprolol Tartrate
Approved colours used in empty capsule shell.
Important: Do not chew or crush the capsules.
They should be swallowed whole with water.
Dosage: As directed by the Physician
Store below 25°C
SCHEDULE H PRESCRIPTION DRUG - CAUTION
Not to be sold by retail without the prescription of a Registered Medical Practitioner
M.L. MNB/05/109
Mfd. by CIPLA LTD.
Malpur, Dist. Solan 173 205 INDIA
Cipla
Rx Metoprolol Succinate
Extended-Release Capsules 50 mg
Metolar XR-50
B.NO.BA20731 MFD.MAR.22 EXP.FEB.24
M.R.P RS.107.53 FOR 15 CAPS.
(INCL.OF ALL TAXES)

What is wrong with other data structures for insert, delete, search?

- Binary search trees:
  - Insert, search, delete are O(n log n), where n is the actual no. of (key, value) elements
  - Exceptionally good if (a) insert, search, delete are equally frequent, and (b) space is limited
- Sorted list of (key, value) elements:
  - Sorting requires O(n log n) time
  - Insert and delete operations require O(n) time
  - Exceptionally great if prepare once, search forever

Direct-address tables

- Consider searching through a dynamic set that permits operations: inset, search and delete
- Each element has associated key, drawn from universe U = {0, 1, …, |U|-1}, |U| is small
  - Element x = (key, value)
  - Keys are natural numbers, or character strings (e.g. ‘CSE 102’)
  - No two elements have the same key
  - Total no. of elements is n <= m, but invariably n << m
- Solution: use an array, or “direct address table”, T[0, 1, …, m-1], where position/slot/index K corresponds to key k in the universe

Direct-address tables

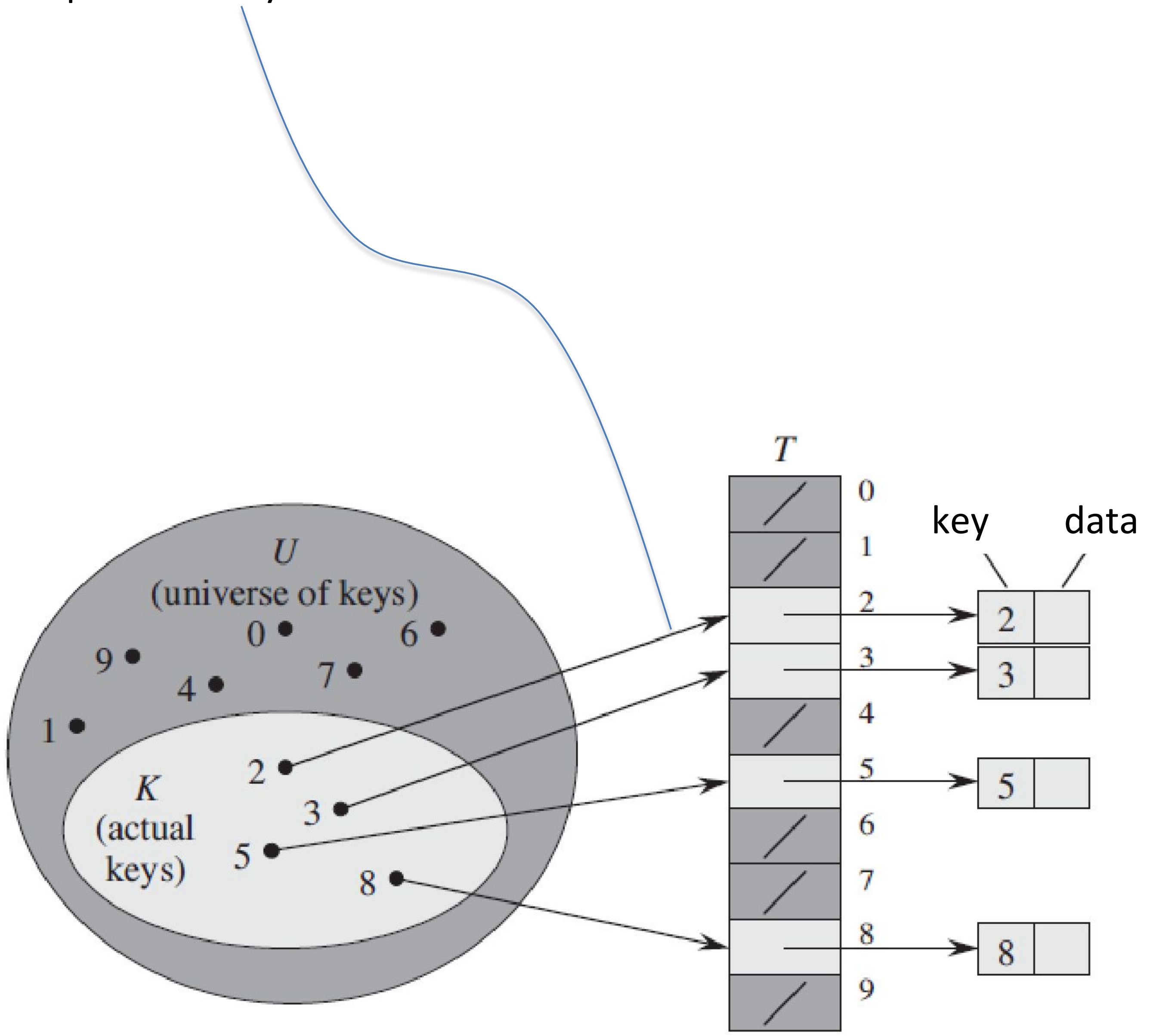
- Consider grade card of students at IIITD
- Student roll numbers act as keys (year of admission, roll number)
  - UG students: 2000- 0000 through 2099- 0999 → 100,000 students
  - PG students: 2000- 1000 through 2099- 1999 → 100,000 students
  - PhD students: 2000- 2000 through 2099- 2999 → 100,000 students
  - Other students: 2000- 3000 through 2099- 3999 → 100,000 students

  |U| = 400,000
- Or
  - All students: 2000-0000 through 2099-9999 → 1,000,000 students

  |U| = 1,000,000

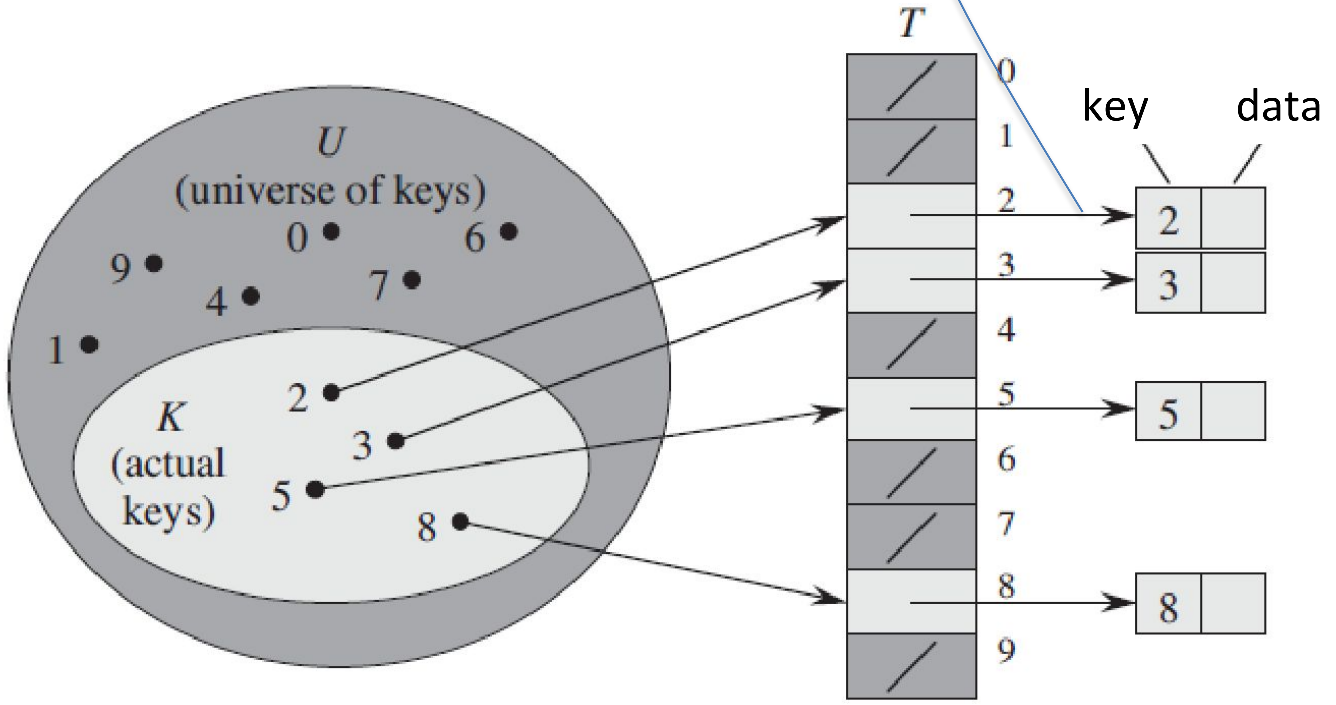
Direct-address tables

- Use an array, or "direct-address table", T[0, 1, …, m-1], where position/slot/index k corresponds to key k in the universe



Direct-address tables

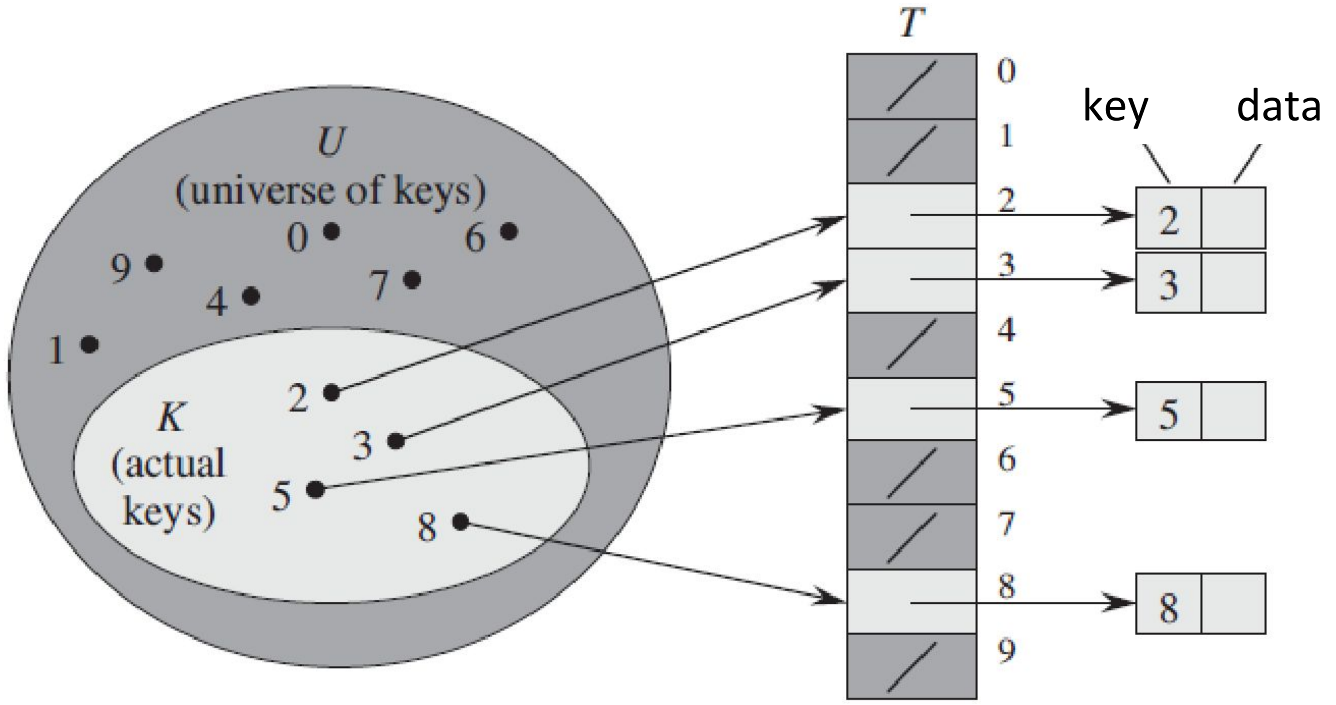
- Two ways to look at the direct table, T:
  - The data is stored in the table itself
    - where nothing is stored in T[k] if there is no data corresponding to key = k
  - Data is stored in a separate location, pointed to by T[k]
    - where T[k] = Nil if there is no data corresponding to key = k



Direct-address tables

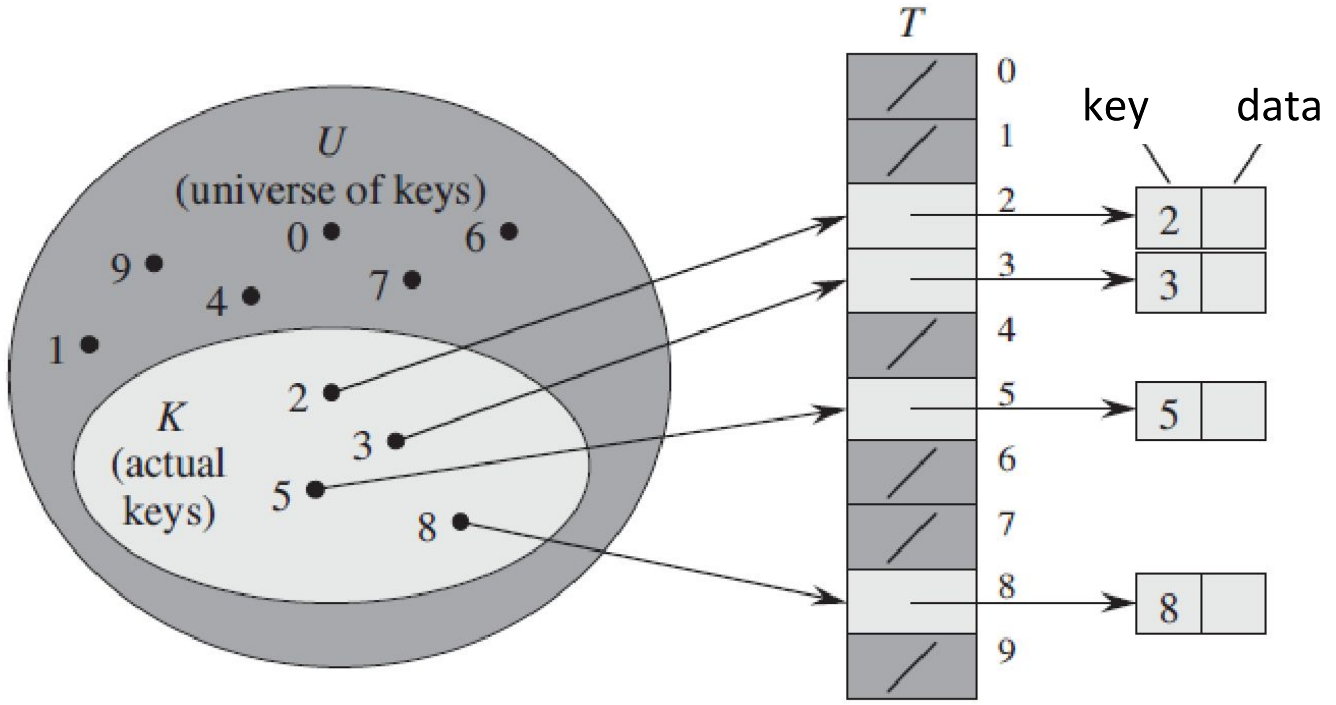
- Operations: Insert(T, x); Search(T, k), Delete(T, x)

```
DIRECT-ADDRESS-SEARCH(T, k)          ← O(1)
1  return T[k]

DIRECT-ADDRESS-INSERT(T, x)          ← O(1)
1  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)          ← O(1)
1  T[x.key] = NIL
```



Direct-address tables

- Downside of direct addressing:
  - If |U| is large, say 10^9, one may never set aside a 10^9 size table
  - If m << |U|, then space in T wasted since large portion of table, T, will be unused

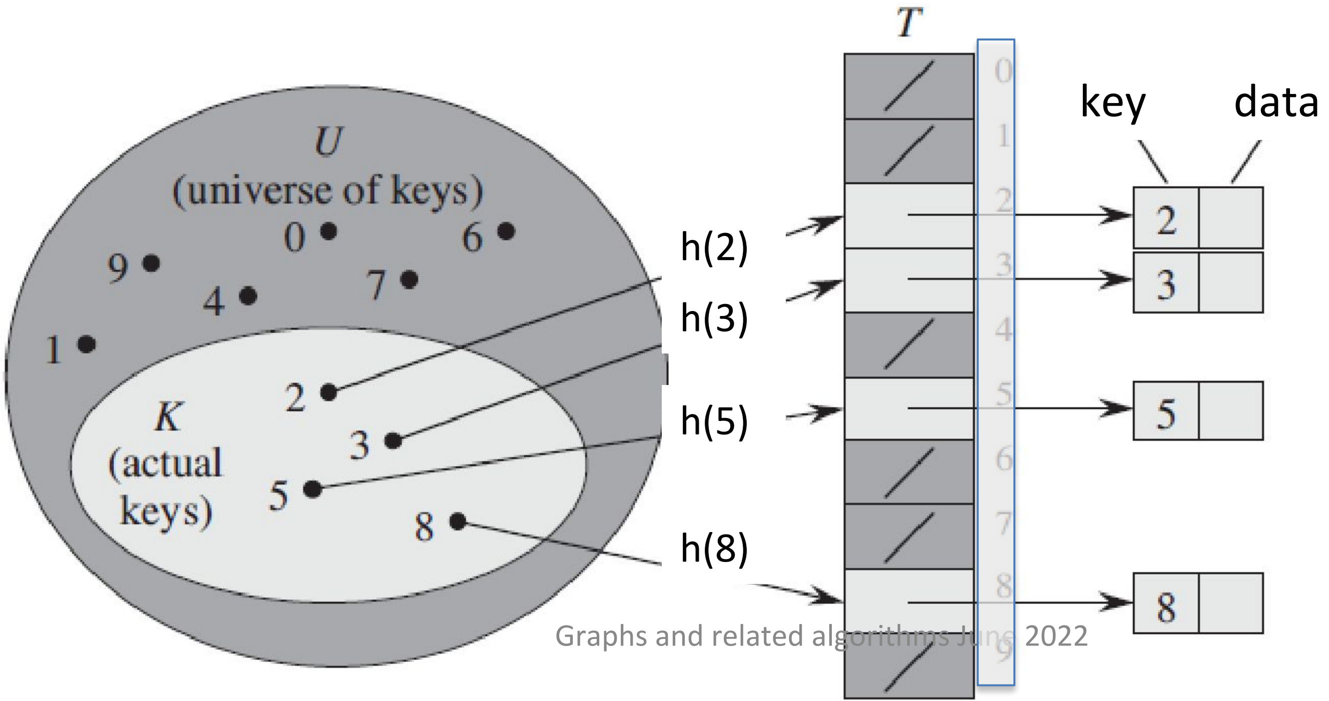


Hash tables

- Hash table requires much less storage than a direct address table
  - reduce the storage requirement to θ(m), where m is actual no. of (key, value) pairs
  - searching an element requires O(1) – this is the average case
    - the worst-case time can be very large
    - Compare with search time of direct-address tables

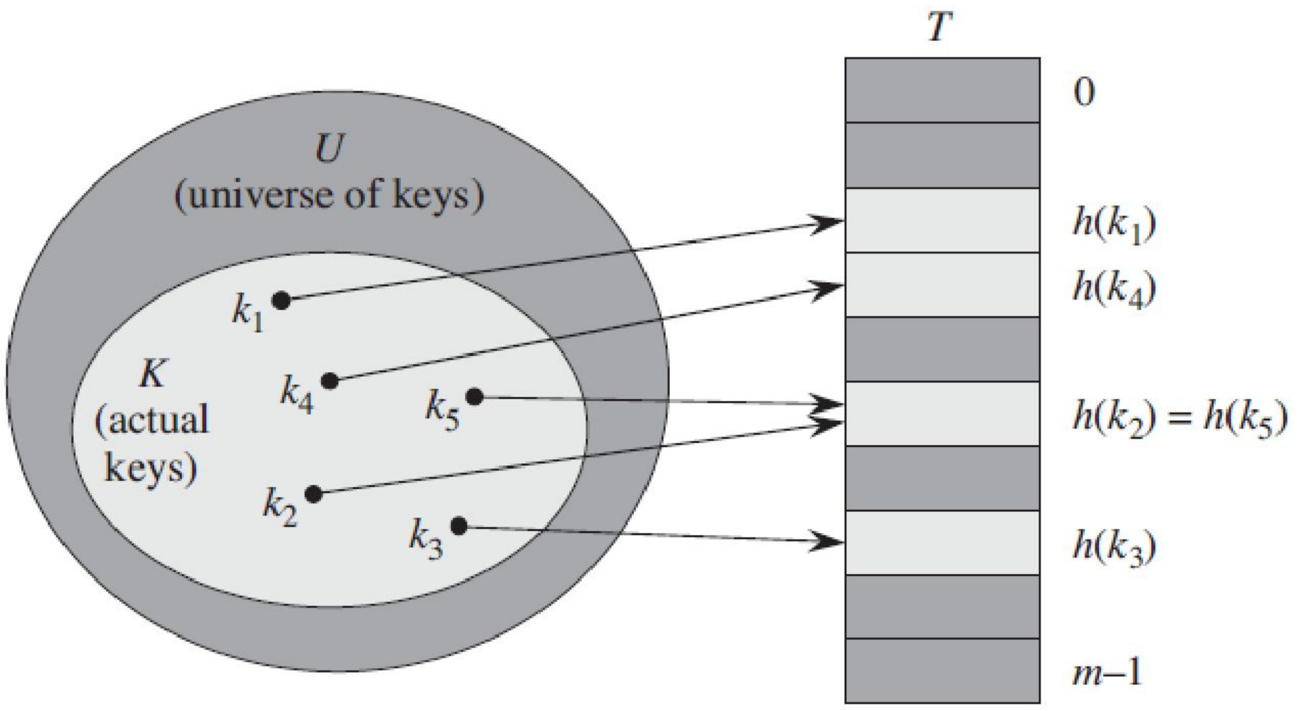
Hash tables

- In direct addressing, element with key = k is stored in slot T[k]. With hashing, this element is stored in slot h(k), viz. T[h(k)]
  - Or, h(.) maps universe U of keys into the slots of a *hash table* T[0, 1, …, m-1], or
    h: U T[0, 1, .., m]
  - Element (k1) is mapped onto T[h(k1)], element (k2) is mapped onto T[h(k2)]
  - etc.



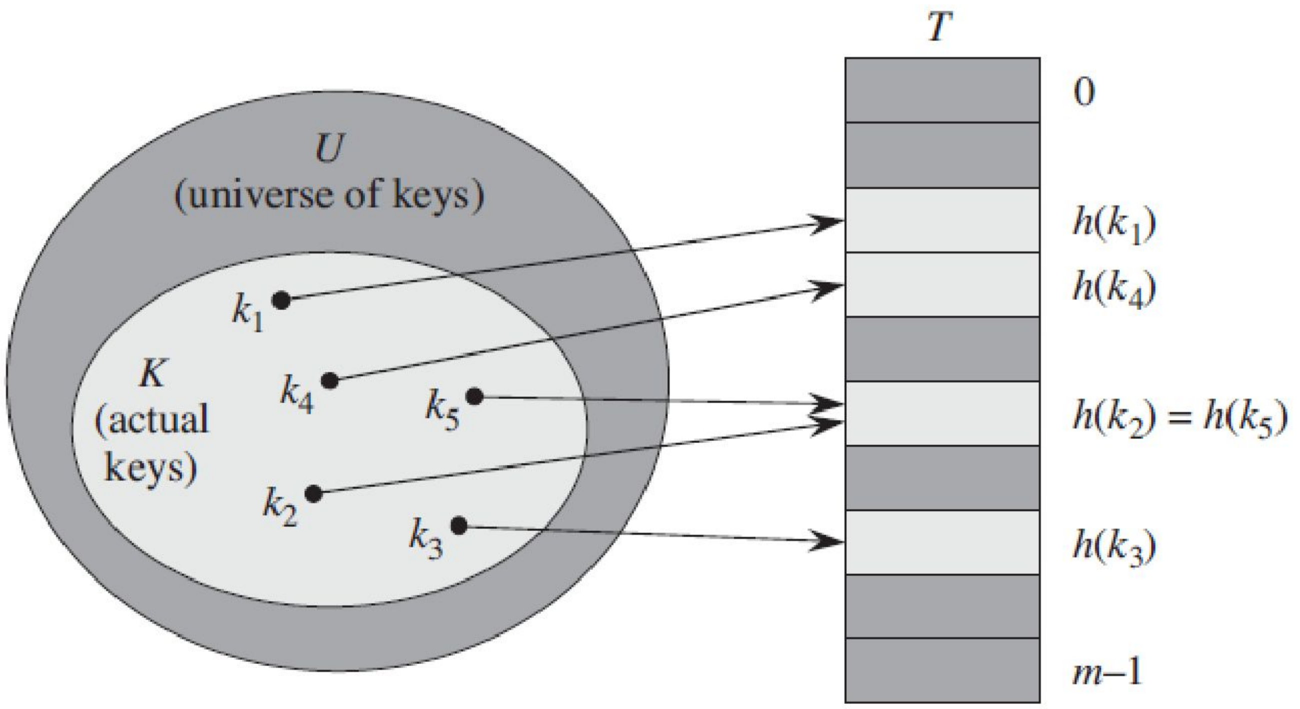
Hash tables

- In direct addressing, an element with key = k is stored in slot T[k]. With hashing, this element is stored in slot h(k), viz. T[h(k)]
  - Or, h(.) maps universe U of keys into the slots of a *hash table* T[0, 1, …, m-1]
  - Element (k1) is mapped into T[h(k1)], element (k2) is mapped into T[h(k2)]
  - Etc.



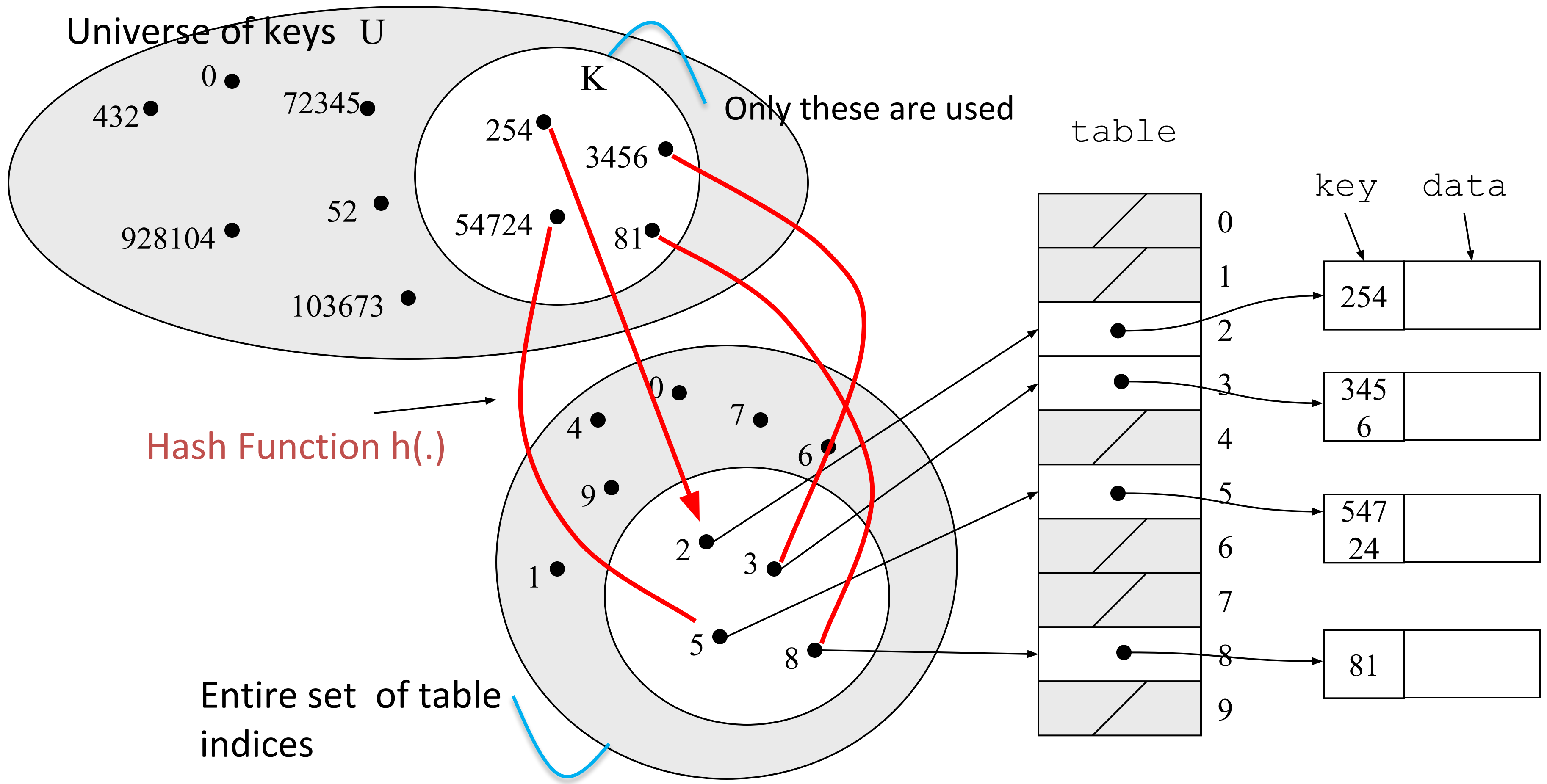
Hash tables

- In direct addressing, an element with key = k is stored in slot T[k]. With hashing, this element is stored in slot h(k), viz. T[h(k)]
  - Or, h(.) maps universe U of keys into the slots of a *hash table* T[0, 1, …, m-1]
  - Element (k1) is mapped into T[h(k1)], element (k2) is mapped into T[h(k2)]
  - Etc.



Hash tables

- h(.) maps universe U of keys into the slots of a *hash table* T[0, 1, …, m-1]



Hash tables

- Advantage: instead of working with array T of size |U|, now working with T of size m << |U| -- m is determined by hash function h(.)

   ⇒ significantly reducing the storage requirement
- For example if U = {32-bit natural numbers}, or |U| = 2^{32} = 4 billion:
  - IF h(k) is a 16 bit number, then m = 2^{16} = 64K, or a reduction by factor of 2^{16}
    - Implying that 2^{16} keys will map onto one index in T, or h(k)
  - IF h(k) is a 24 bit number, then m = 2^{24} = 16 million, or a reduction by factor of 2^{8}
    - Implying that 256 keys map onto one index in T, or h(k)
- Necessarily, the above results in "collision" – we will deal this a bit later
- The way out is choose an h(.)
  - A large enough m, and
  - An appropriate hash function h(.)
    - An h(.) that maps keys only 0, 1, …, m-1
    - An h(.) that evenly spreads the h(k) across the entire range 0, 1, …, m-1
    - And such that h(.) is seemingly random
    - An h(.) that depends upon all bits and bytes of the original key

Hash tables

- Plus point is: instead of working with array T of size |U|, one is now working with T of size m, as determined by hash function h(.)

  ⍰ significantly reducing the storage requirement
- For example if U = {32-bit numbers}, or |U| = 2^{32} = 4 billion:
  - and h(k) is a 16 bit number, then m = 2^{16} = 64K, or a reduction by factor of 2^{16}
    - Implying that 2^{16} keys will map onto one index in T, or h(k)
  - and h(k) is a 24 bit number, then m = 2^{24} = 16 million, or a reduction by factor of 2^{8}
    - Implying that 2^{8} keys will map onto one index in T, or h(k)
- Necessarily, the above WILL results in “collision” – we will deal with this a bit later
- The way out is choose an h(.)
  - A large enough m, and
  - An appropriate hash function h(.)
    - An h(.) that maps keys only 0, 1, …, m-1
    - An h(.) that evenly spreads the h(k) across the entire range 0, 1, …, m-1
    - And such that h(.) is seemingly random
    - An h(.) that depends upon all bits and bytes of the original key

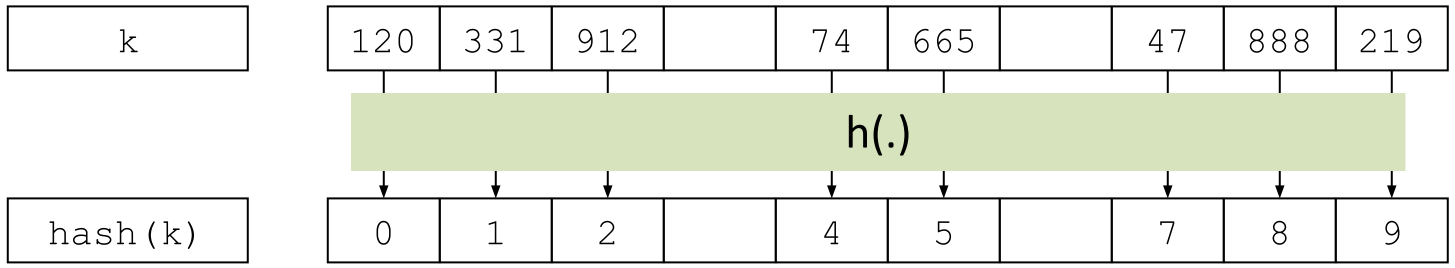
Hash tables

- Plus point is: instead of working with array T of size |U|, one is now working with T of size m, as determined by hash function h(.)

  ⇒ significantly reducing the storage requirement
- For example if U = {32-bit numbers}, or |U| = 2^{32} = 4 billion:
  - and h(k) is a 16 bit number, then m = 2^{16} = 64K, or a reduction by factor of 2^{16}
    - Implying that 2^{16} keys will map onto one index in T, or h(k)
  - and h(k) is a 24 bit number, then m = 2^{24} = 16 million, or a reduction by factor of 2^{8}
    - Implying that 2^{8} keys will map onto one index in T, or h(k)
- Necessarily, the above WILL results in "collision" – we will deal with this a bit later
- The way out is to choose an h(.)
  - That results in large enough m, and
  - An appropriate hash function h(.)
    - An h(.) that maps keys onto {0, 1, …, m-1}
    - An h(.) that evenly spreads the h(k) across the entire range 0, 1, …, m-1
    - And such that h(.) is seemingly random
    - An h(.) that depends upon all bits and bytes of the original key

Hash functions

- Choose an h(.)
  - A large enough m, and
  - An appropriate hash function h(.)
- Some bad choices for h(.)



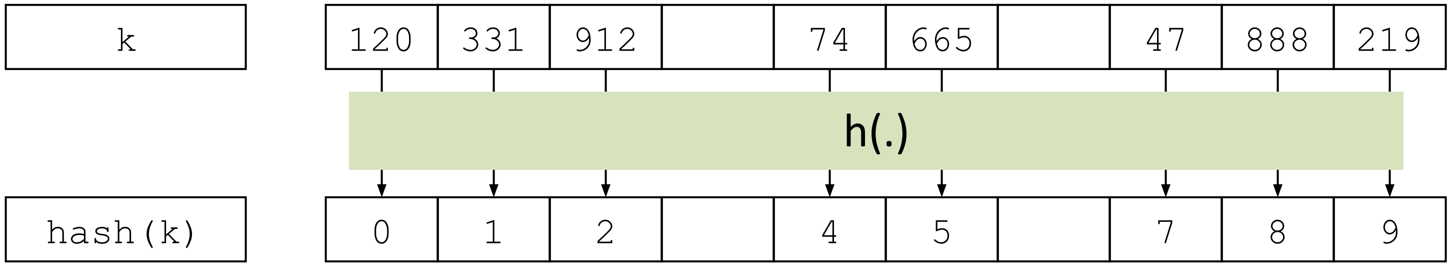
- $h(k) = k \bmod m$
- $h(c_0 c_1 c_2 \ldots c_{n-1}) = (c_0 + c_1 + c_2 + \ldots + c_{n-1}) \bmod m$
- Etc.

what happens if $m = 2^p$ for some p?

Consider keys ‘CSE 102’, ‘CSE 201’

Hash functions

- Choose an h(.)
  - A large enough m, and
  - An appropriate hash function h(.)
- Some bad choices for h(.)



- h(k) = k mod m
- $h(c_0c_1c_2 \dots c_{n-1}) = (c_0+c_1+c_2+\dots+c_{n-1}) \bmod m$
- Etc.

what happens if $m = 2^p$ for some p?

Consider keys ‘CSE 102’, ‘CSE 201’

Hash functions

- Note: If keys are strings, we can convert the string into an integer using their ASCII values and then form a *key*
- *Example:*

character →	C	S	E		3	7	3	<0>
ASCII value →	67	83	69	32	51	55	51	0

Hash functions

- Choose an h(.)
  - A large enough m, and
  - An appropriate hash function h(.)
- Some good choices for h(.)

1. h(k) = k mod m, where m is a prime number, and one that does not divide $r^k + a$ or $r^k - a$ where k and a are small, and r is the radix (such as 64 or 128)

3. h(k) = floor(m (k A mod 1)

   where 0 < A < 1, such as A = (sqrt(5) -1)/2,

Collisions

- A collision occurs when two different keys hash onto the same value

  Example:

  Let h(k) = k mod m, m = 17

  18 mod 17 = 1

  35 mod 17 = 1

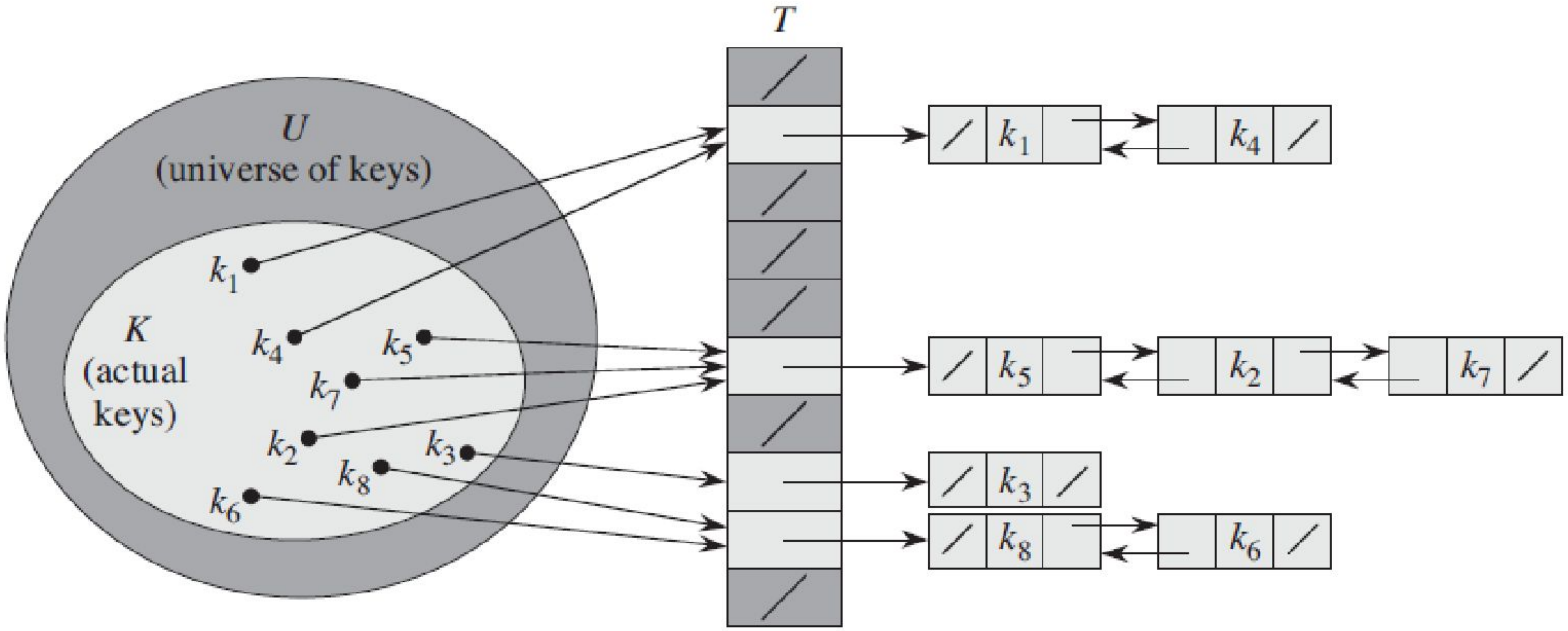
  ⍰ 18 and 35 hash into the same value
- Clearly, two records cannot be stored in the same slot in array, T

Collisions

- Collision Resolution:
- Two strategies:
  - Chaining: Use a linked list, for example, to store multiple items that hash to same slot
  - Probing (linear or quadratic): search for an empty slot using a second function and store item in first empty slot

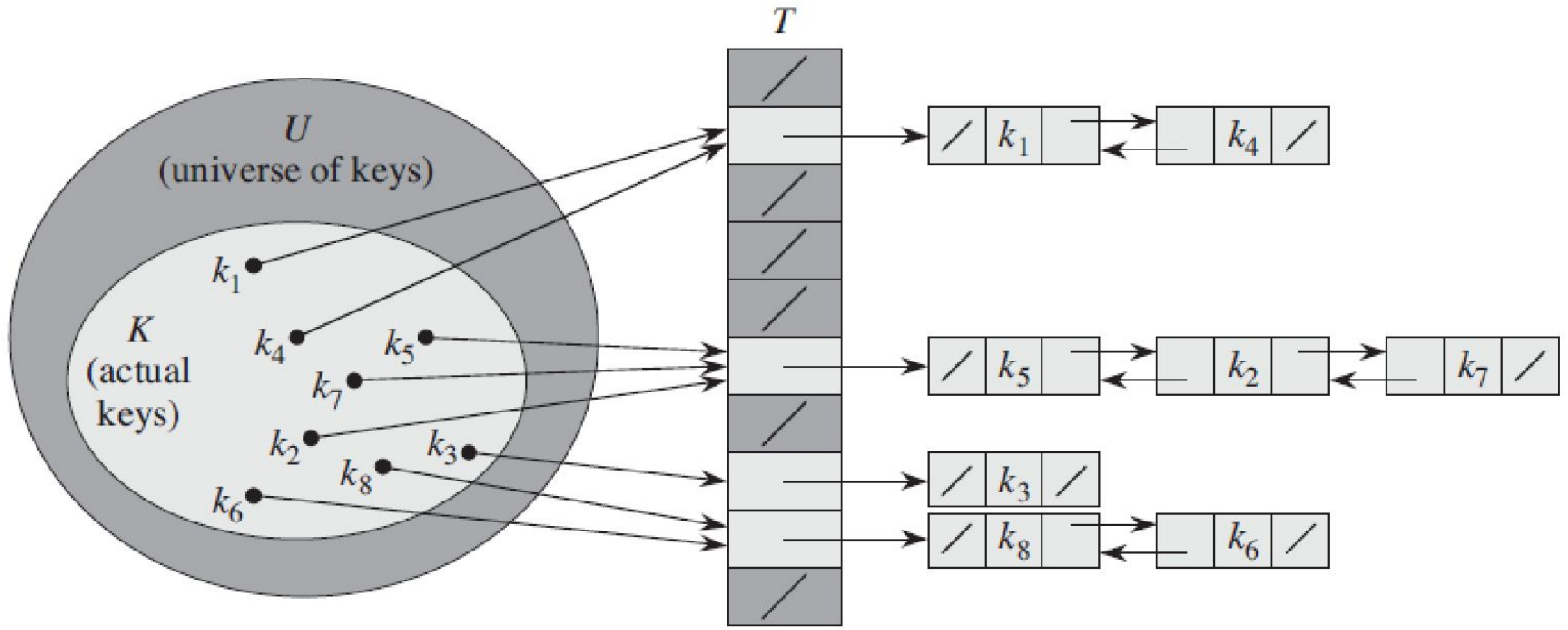
Collisions

- Chaining: Use a linked list to store multiple items that hash to same slot



Collisions

- Chaining: Use a linked list to store multiple items that hash to same slot

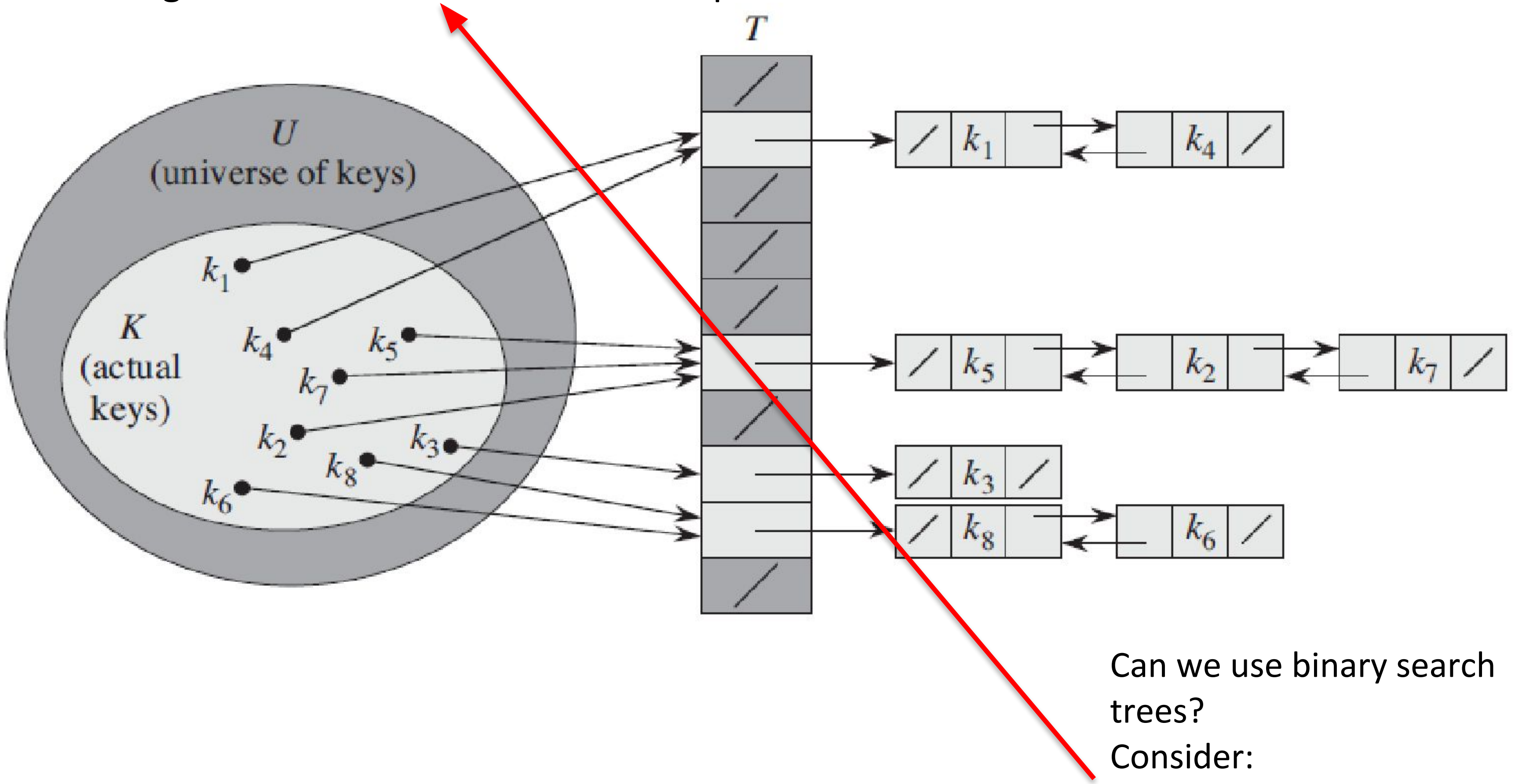


```
CHAINED-HASH-INSERT(T, x)
1  insert x at the head of list T[h(x.key)]

CHAINED-HASH-SEARCH(T, k)
1  search for an element with key k in list T[h(k)]

CHAINED-HASH-DELETE(T, x)
1  delete x from the list T[h(x.key)]
```

O(B) runtime where B is the number of elements in the particular chain

Collisions

- Chaining: Use a linked list to store multiple items that hash to same slot



Can we use binary search trees?

Consider:

a. Time complexity

b. no. of elements in each slot

c. Programming effort

Load factor

- Load factor $\lambda = n/m$, where n = number of elements stored, m = size of Table
  - $m = 101$ and $n = 303$, then $\lambda = 3$
  - $m = 101$ and $n = 20$, then $\lambda = 0.2$
- Average length of chained list = λ

  ⍰ average time to access an item = $\theta(1+\lambda)$
  - λ will be < 1 and close to 1 if we use a good hashing function, and appropriate m
  - With chaining, hashing continues to work for $\lambda > 1$

Collisions

- Probing (linear or quadratic): search for an empty slot using a second function and store item in first empty slot
- Consider the sequence:

  Check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for purpose of inserting element with key k

  $h_i(k) = (h(k) + F(i)) \bmod m$

    where m = size of table

    $F(0) = 0$

    $F(i) = i$ -- for linear probe

    $F(i) = i^2$ -- for quadratic probe

Collisions

- Probing (linear or quadratic): search for an empty slot using a second function and store item in first empty slot
- Consider the sequence:

Check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for purpose of inserting element with key k

$h_i(k) = (h(k) + F(i)) \bmod m$

where m = size of table

$F(0) = 0$

$F(i) = i$ -- for linear probe

$F(i) = i^2$ -- for quadratic probe

For linear probe: check out:
h(k)
h(k) + 1
h(k) + 2
h(k) + 3…
till one finds an unused slot in T[.]

For quadratic probe: check out:
h(k)
h(k) + 1
h(k) + 4
h(k) +9
…
till one finds an unused slot in T[.]

Collisions

- Probing: check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for inserting element with key k

  $h_i(k) = (h(k) + F(i)) \bmod m$

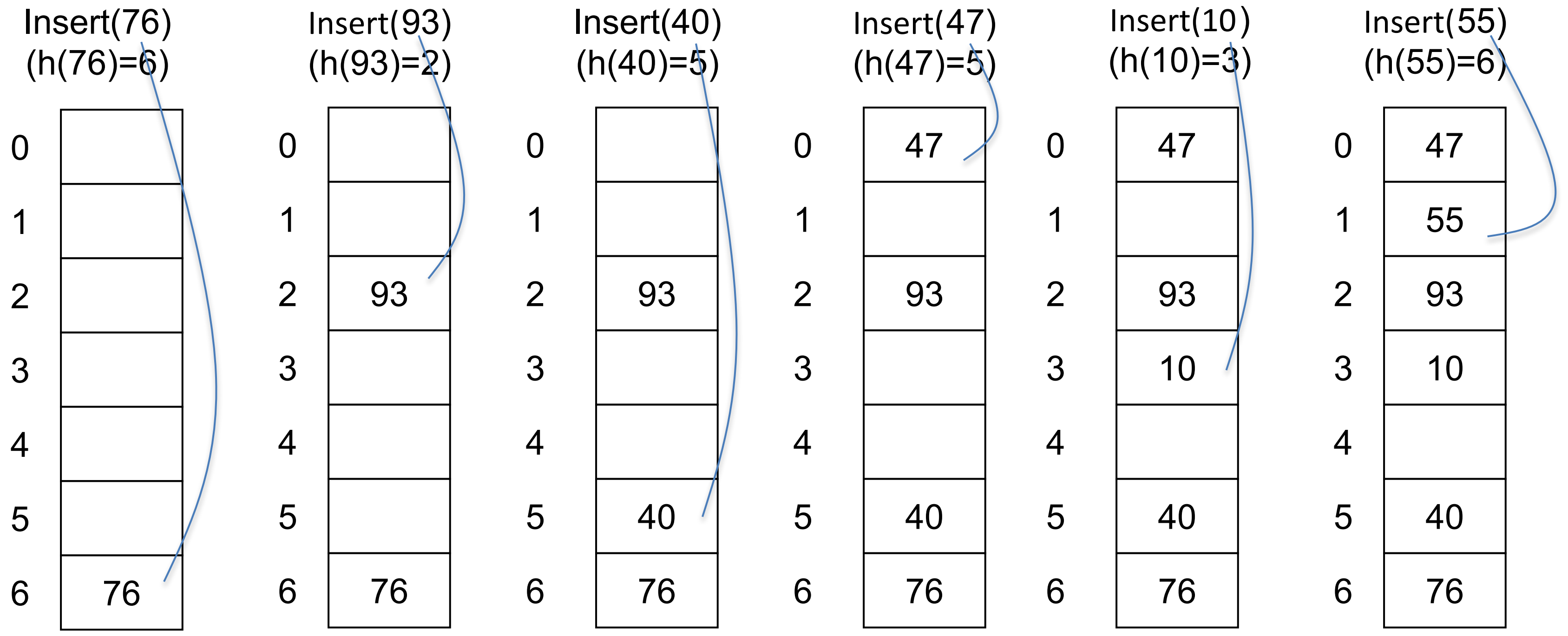
  where m = size of table

  $F(0) = 0$

  $F(i) = i$ -- for linear probe

  $F(i) = i^2$ -- for quadratic probe

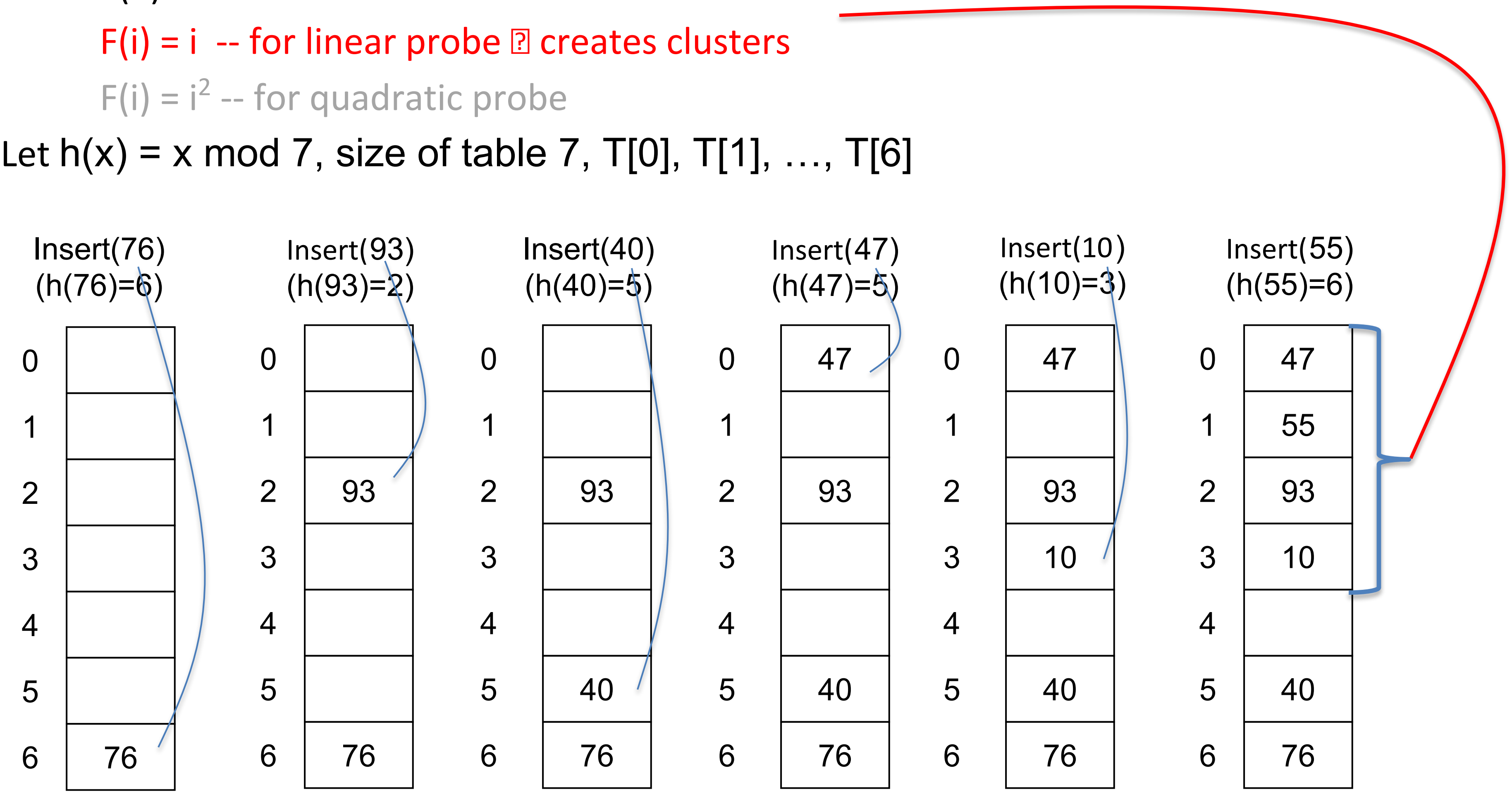
- Let h(x) = x mod 7, size of table 7, T[0], T[1], …, T[6]



	Insert(76)	Insert(93)	Insert(40)	Insert(47)	Insert(10)	Insert(55)
0				47	47	47
1						55
2		93	93	93	93	93
3					10	10
4						
5			40	40	40	40
6	76	76	76	76	76	76

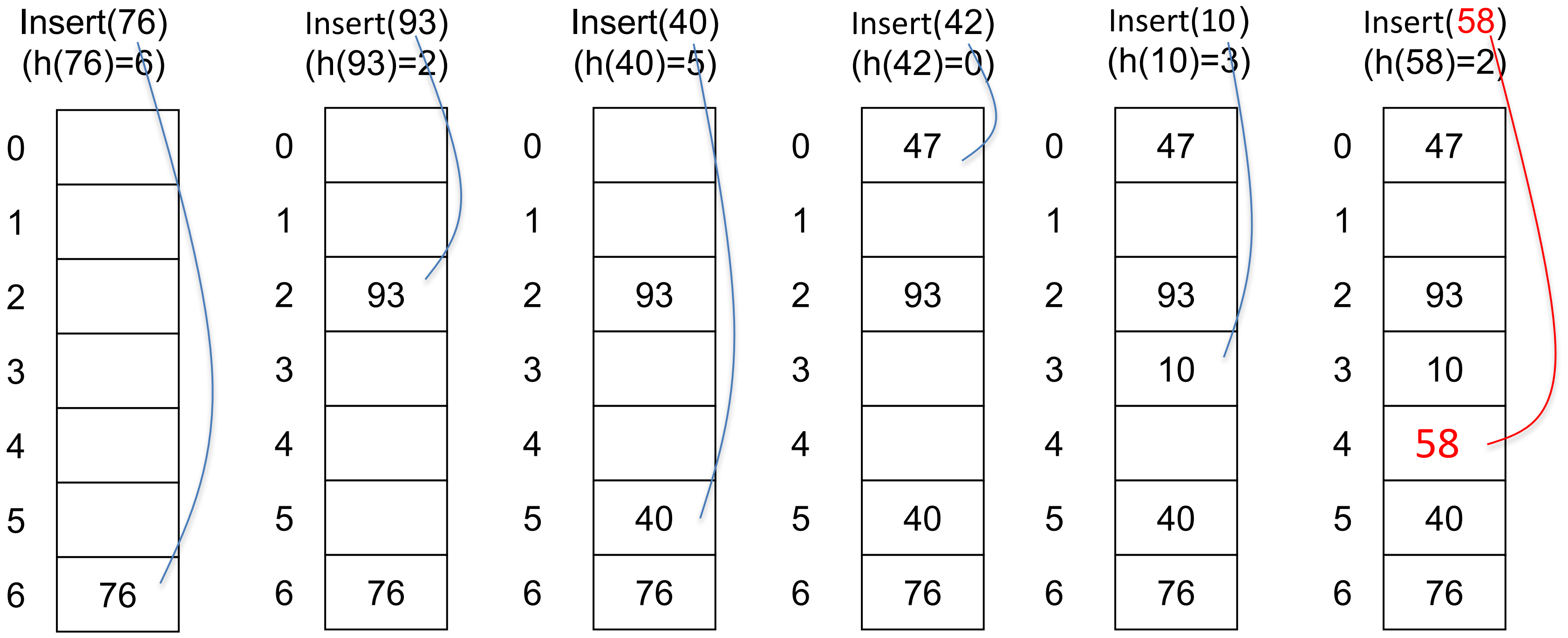
Collisions

- Probing: check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for inserting element with key k
  $h_i(k) = (h(k) + F(i)) \bmod m$
    where m = size of table
    $F(0) = 0$
    $F(i) = i$ -- for linear probe creates clusters
    $F(i) = i^2$ -- for quadratic probe
- Let h(x) = x mod 7, size of table 7, T[0], T[1], …, T[6]



Collisions

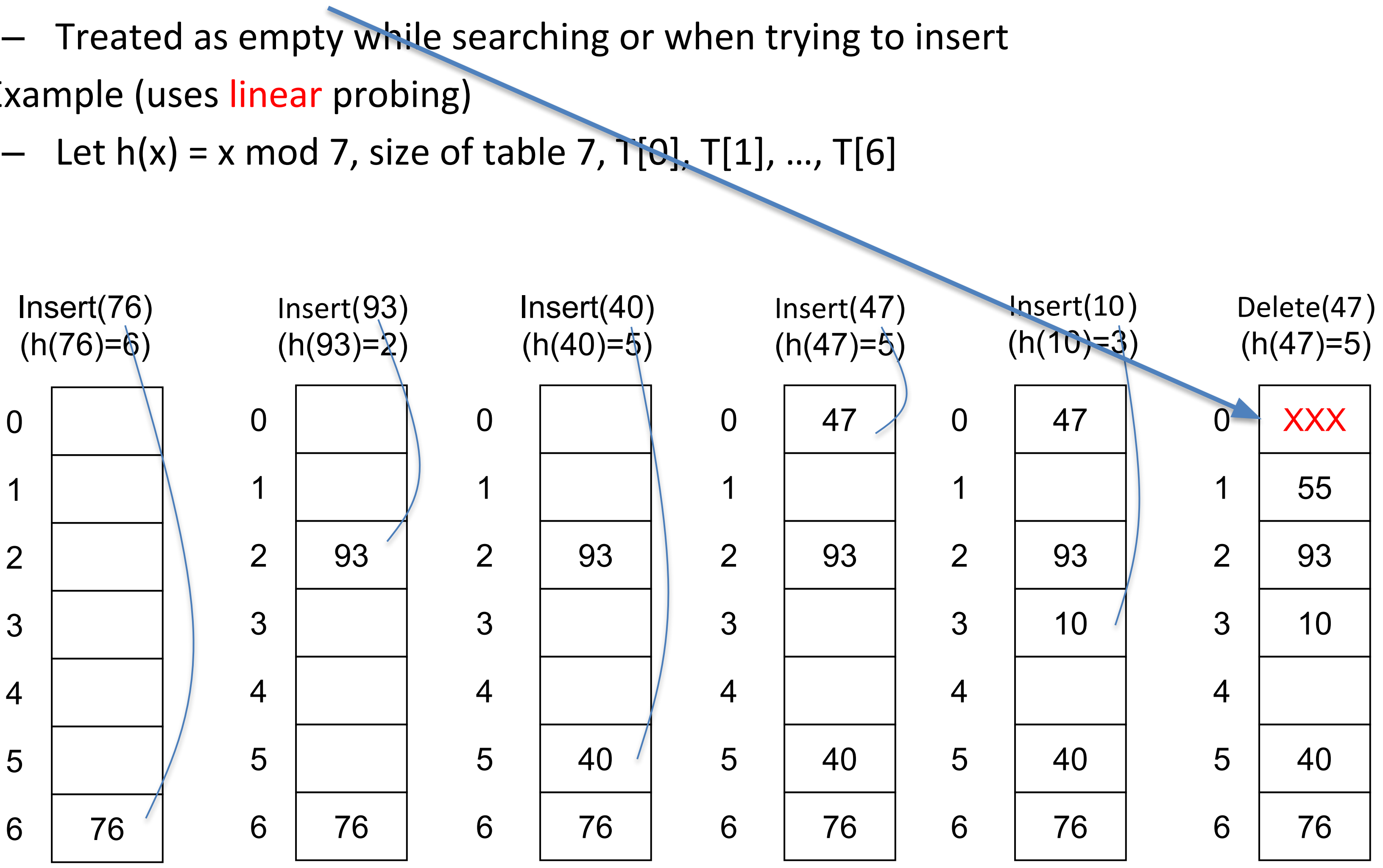
- Probing: check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for inserting element with key k
  $h_i(k) = (h(k) + F(i)) \bmod m$
  where m = size of table
  $F(0) = 0$
  $F(i) = i$ -- for linear probe
  $F(i) = i^2$ -- for quadratic probe
- Let h(x) = x mod 7, size of table 7, T[0], T[1], …, T[6]



Index	Insert(76)	Insert(93)	Insert(40)	Insert(42)	Insert(10)	Insert(58)
0				47	47	47
1						
2		93	93	93	93	93
3					10	10
4						58
5			40	40	40	40
6	76	76	76	76	76	76

Collisions

- Probing: check out $h_0(k)$, $h_1(k)$, $h_2(k)$, $h_3(k)$, … for inserting element with key k
- Search: normally, till an entry with key is encountered, or an empty cell if found
- Delete: do a lazy deletion, viz. deleted keys are marked "deleted"
  - Treated as empty while searching or when trying to insert
- Example (uses linear probing)
  - Let h(x) = x mod 7, size of table 7, T[0], T[1], …, T[6]



	Insert(76) (h(76)=6)	Insert(93) (h(93)=2)	Insert(40) (h(40)=5)	Insert(47) (h(47)=5)	Insert(10) (h(10)=3)	Delete(47) (h(47)=5)
0				47	47	XXX
1						55
2		93	93	93	93	93
3					10	10
4						
5			40	40	40	40
6	76	76	76	76	76	76

Q&A