CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 8: Searching, Hashing

Searching Problems

Searching Problem

- Problem: Given an array A[1:n] prepare a data structure so that:
  - Given a number x, we can quickly check if x is in A or not

Searching Problem

- Problem: Given an array A[1:n] prepare a data structure so that:
  - Given a number x, we can quickly check if x is in A or not
- Comparison-based approach:
  - Sort the array A[1:n] and store it as the data structure
  - Use binary search to find x in A
  - Preparing the data structure takes $O(n \log n)$ time
  - Searching the element requires $O(\log n)$ time

Searching Problem

- Can we do better if numbers are in $\{1, \ldots, M\}$ for small M?

Searching Problem

- Can we do better if numbers are in $\{1,\ldots,M\}$ for small M?
  - Better attempt:
    - Store the array C of counting sort first in O(n+M) time
      - C[j]: number of times j appears in A
    - Just check if C[x] > 0 or not in O(1) time

Hashing: A More Clever Way of Searching

Hashing

- Problem: Given an array A[1:n] of numbers prepare a data structure so that:
  - Given a number x, we can quickly check if x is in A or not
- Data structure here is a hash table:
  - An array T of size m (size of m depends on storage capacity)

Hashing

- Problem: Given an array A[1:n] of numbers prepare a data structure so that:
  - Given a number x, we can quickly check if x is in A or not
- Data structure here is a hash table:
  - An array T of size m (size of m depends on storage capacity)
- We also have a hash function $h : \mathbb{N} \to \{1, \ldots, m\}$
- For every i, we compute b(i) = h(A[i]) and place A[i] in T[b(i)]
- Given x, we check if T[h(x)] = x or not

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

				20			

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

				20		150	

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16				20		150	

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16				20		150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:	20	150	16	71	29	51	25	34

T:	16				20	29	150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16			51	20	29	150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25		51	20	29	150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Does x=51 belong to A?

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Does x=51 belong to A? h(51) = 4 so we check T[4]

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Does x=51 belong to A? h(51) = 4 so we check T[4]

Does x=17 belong to A?

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

Does x=51 belong to A? h(51) = 4 so we check T[4]

Does x=17 belong to A? h(17) = 2 so we check T[2]

Hashing: Example

What would happen if in the input we changed 71 with 72?

n=m=8

20	150	16	71	29	51	25	34

T:

16	25	34	51	20	29	150	71

- h(
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

Hashing: Example

- $h(x) = (x \mod 8)$+1
- h(20) = 5
- h(150) = 7
- h(16) = 1
- h(71) = 8
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	72	29	51	25	34

T:

16	25	34	51	20	29	150	

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- ~~h(71) = 8~~ h(72)=1
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	72	29	51	25	34

T:

16	25	34	51	20	29	150	

Hashing: Example

- $h(x) = (x \mod 8)+1$
- h(20) = 5
- h(150) = 7
- h(16) = 1
- ~~h(71) = 8~~ h(72)=1
- h(29) = 6
- h(51) = 4
- h(25)=2
- h(34) = 3

n=m=8

A:

20	150	16	72	29	51	25	34

T:

16	25	34	51	20	29	150	

This is called a **collision**

Collisions

- Main questions:
  - How to avoid collisions?
  - How to handle collisions?

Collisions

- Main questions:
  - ~~How to avoid collisions?~~ How to limit collisions?
  - How to handle collisions?

Collisions

- Main questions:
  - How to limit collisions? **Random hash functions**, cryptographic hash functions, secure hash functions, …
  - How to handle collisions? **Chaining**, open addressing, cuckoo hashing, …

Hash Functions: Uniform, Universal, and Near-Universal

Hash Functions: Uniform, Universal, and Near-Universal

How to limit the number of collisions?

Hash Function

- A hash function: $h : \mathbb{N} \to \{1,\ldots,m\}$
- If we fix a hash function, there is ALWAYS an input that makes EVERY entry collide

Hash Function

- A hash function: $h : \mathbb{N} \to \{1,\ldots,m\}$

- If we fix a hash function, there is ALWAYS an input that makes EVERY entry collide

- **Example:**

  - Suppose $h(x) = (x \mod 8) + 1$

Hash Function

- A hash function: $h : \mathbb{N} \to \{1,\ldots,m\}$
- If we fix a hash function, there is ALWAYS an input that makes EVERY entry collide
- **Example:**
  - Suppose $h(x) = (x \mod 8) + 1$
  - We can set $A = [8,16,24,\ldots,8n]$
  - All these numbers will be hashed to position 1

Random Hash Functions

- We can pick h randomly from a known family $\mathscr{H}$ of hash functions
  - Note that h itself is not at all random — it is a mathematical function like any other
  - It is the choice of h from $\mathscr{H}$ that is random

Random Hash Functions

- Example: The family $\mathscr{H}$ can have these four hash functions:
  - $h_1(x) = (2x + 3) \mod 8$
  - $h_2(x) = (4x + 1) \mod 8$
  - $h_3(x) = (6x + 2) \mod 8$
  - $h_4(x) = (x + 7) \mod 8$

Desired Properties of Random Hash Functions

- **Uniform:**
  - $\Pr_{h\in\mathscr{H}}(h(x) = i) = \dfrac{1}{m}$ for all $x \in \mathbb{N}$ and index $i \in \{1,\ldots,m\}$
  - In words, over the random choice of h from the hash family, each number x is mapped to a uniformly random number

Desired Properties of Random Hash Functions

- **Universal:**
  - $\Pr_{h \in \mathscr{H}}(h(x) = h(y)) \leq \frac{1}{m}$ for all $x \neq y \in \mathbb{N}$
  - In words, over the random choice of h from the hash family, the probability that two different fixed numbers map to the same position is at most $\frac{1}{m}$

Desired Properties of Random Hash Functions

- **Universal:**
  - $\text{Pr}_{h\in\mathscr{H}}(h(x) = h(y)) \leq \dfrac{1}{m}$ for all $x \neq y \in \mathbb{N}$
  - In words, over the random choice of h from the hash family, the probability that two different fixed numbers map to the same position is at most $\dfrac{1}{m}$
- Universality is very helpful for limiting number of collisions as we will see soon

Desired Properties of Random Hash Functions

- **Universal:**
  - $\Pr_{h \in \mathscr{H}}(h(x) = h(y)) \leq \dfrac{1}{m}$ for all $x \neq y \in \mathbb{N}$
  - In words, over the random choice of h from the hash family, the probability that two different fixed numbers map to the same position is at most $\dfrac{1}{m}$
- Universality is very helpful for limiting number of collisions as we will see soon
- But universality can sometimes be hard to achieve

Desired Properties of Random Hash Functions

- **Near-Universal:**
  - $\Pr_{h \in \mathscr{H}}(h(x) = h(y)) \leq \dfrac{2}{m}$ for all $x \neq y \in \mathbb{N}$
  - In words, over the random choice of h from the hash family, the probability that two different fixed numbers map to the same position is at most $\dfrac{2}{m}$

Near-Universal Hash Functions

- There are standard ways of creating a universal hash function
- **Example:**
- Suppose we know all numbers in A are between 1 and m
- Pick a prime number p > m
- $\mathcal{H} := \{h_a(x) = ((a \cdot x \mod p) \mod m) + 1 \mid 1 \leq a \leq p - 1\}$
- Pick $h_a \in \mathcal{H}$ uniformly at random to get a random hash function
- This family is near-universal (see textbook for proof)

Hash Tables:
Chaining

Hash Tables: Chaining

How to handle the collisions?

Chaining

- The hash table is an array with each cell being a linked-list
- Given the array A[1:n]:
  - For every i, we compute b(i) = h(A[i]) and add A[i] to the tail of the linked-list at T[b(i)]
- Given x to be searched:
  - We iterate over elements of the linked-list T[h(x)] to find x or output it does not exist

Chaining : Example

- $h(x) = (x \mod 4)+1$

n=8 **m=4**

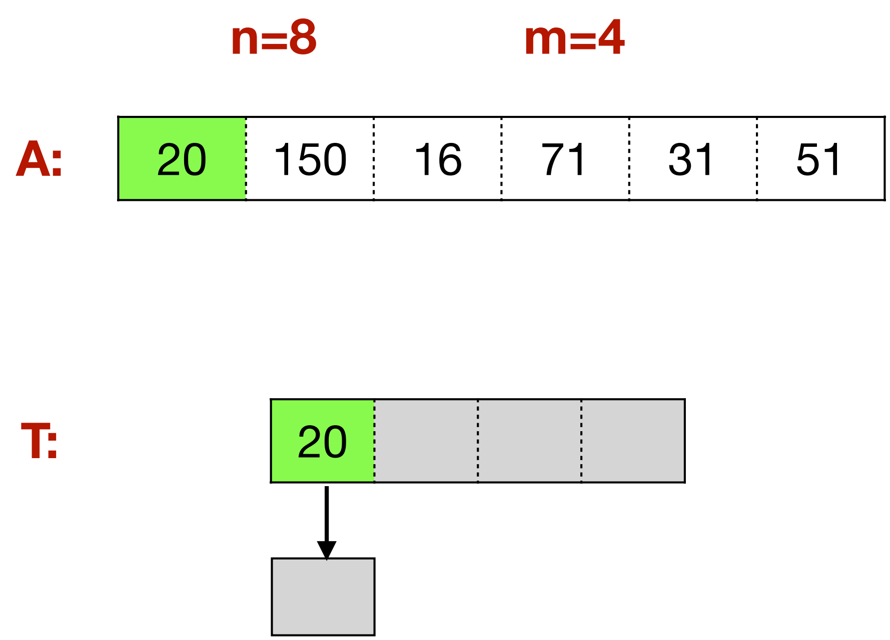
A:

20	150	16	71	31	51

T:

Chaining : Example

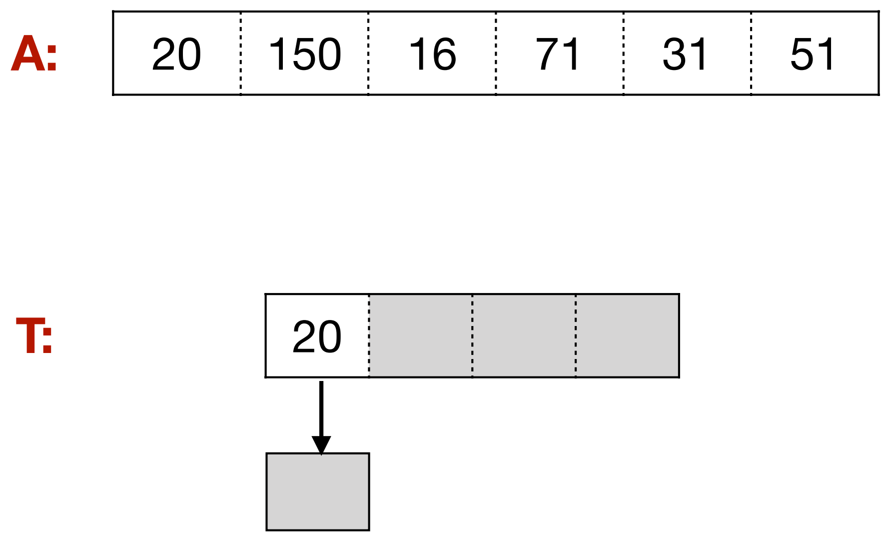
- $h(x) = (x \mod 4)+1$
- h(20) = 1



Chaining : Example

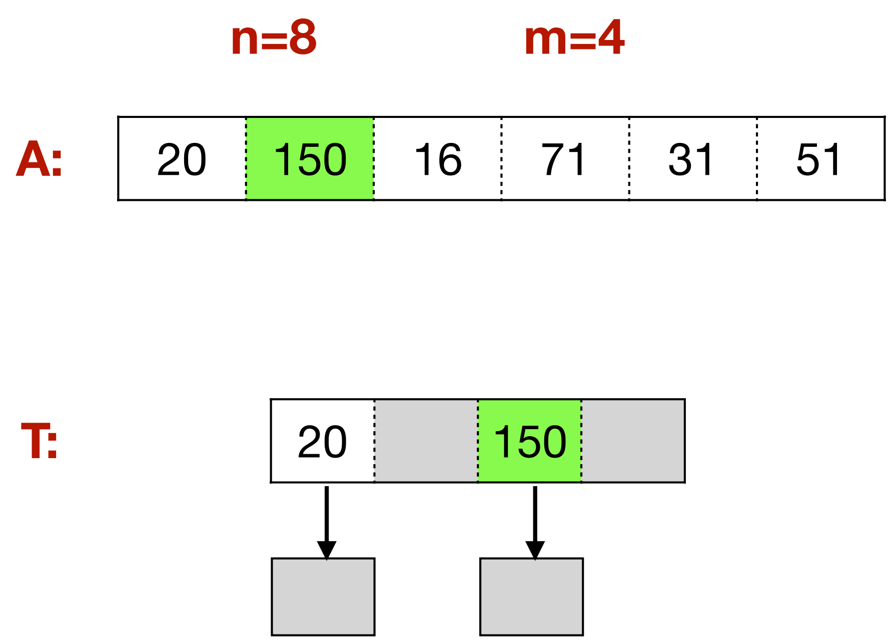
- $h(x) = (x \mod 4)$+1
- h(20) = 1

n=8 **m=4**



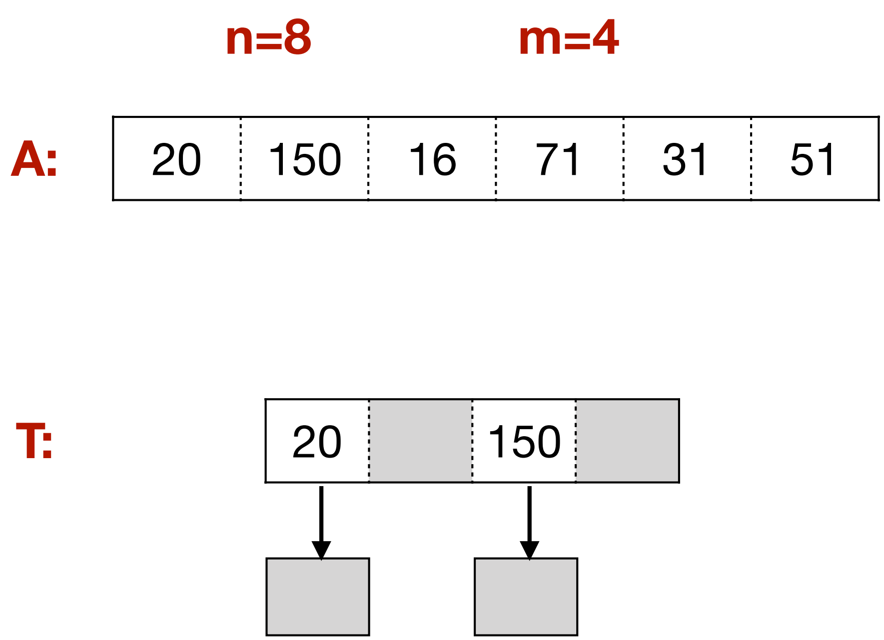
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3



Chaining : Example

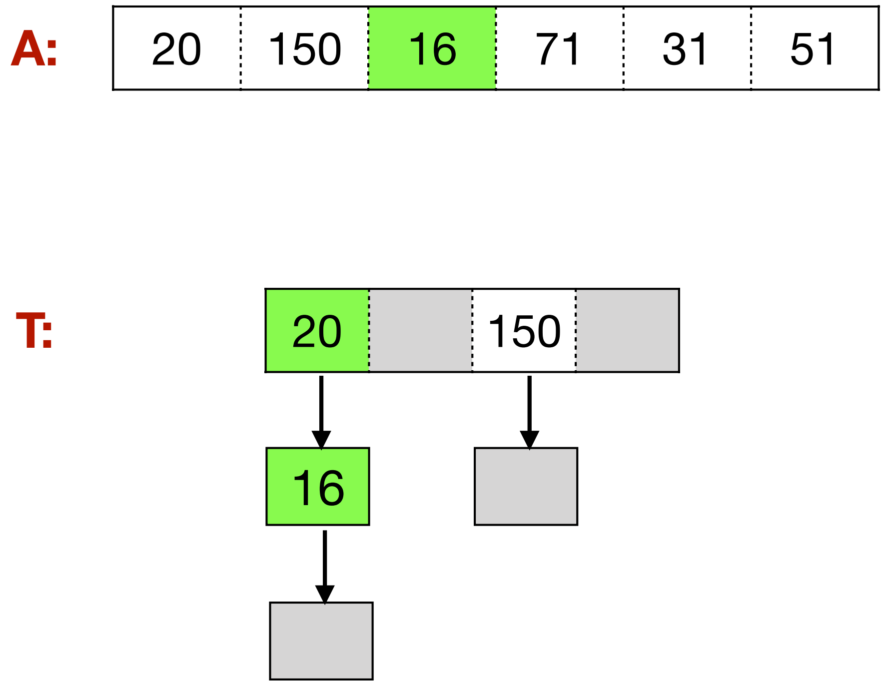
- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3



Chaining : Example

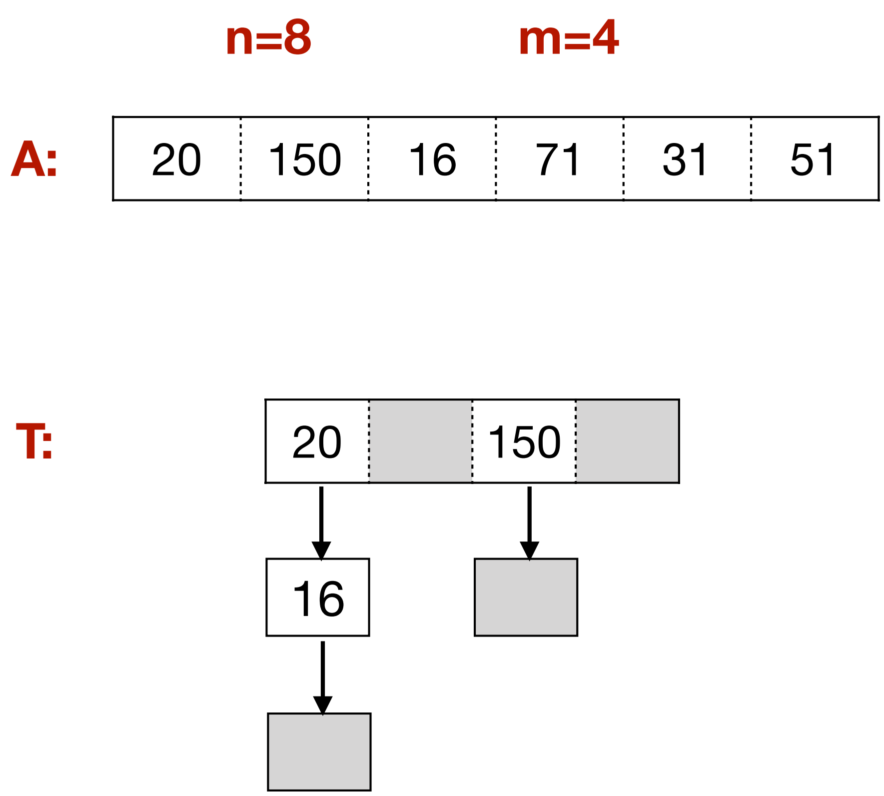
- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1

n=8 **m=4**



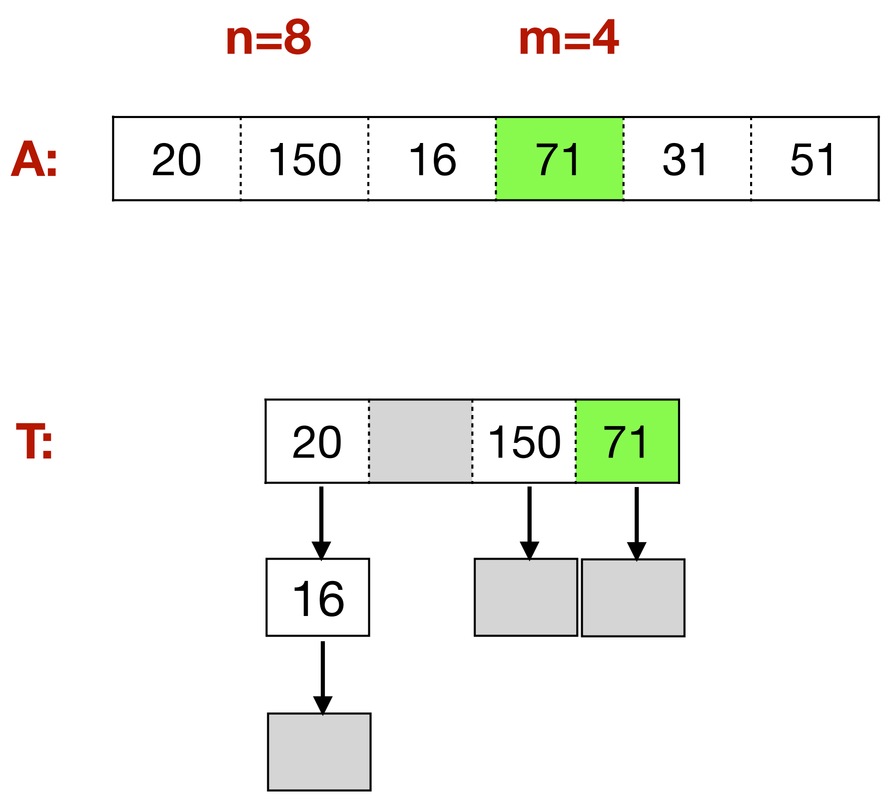
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1



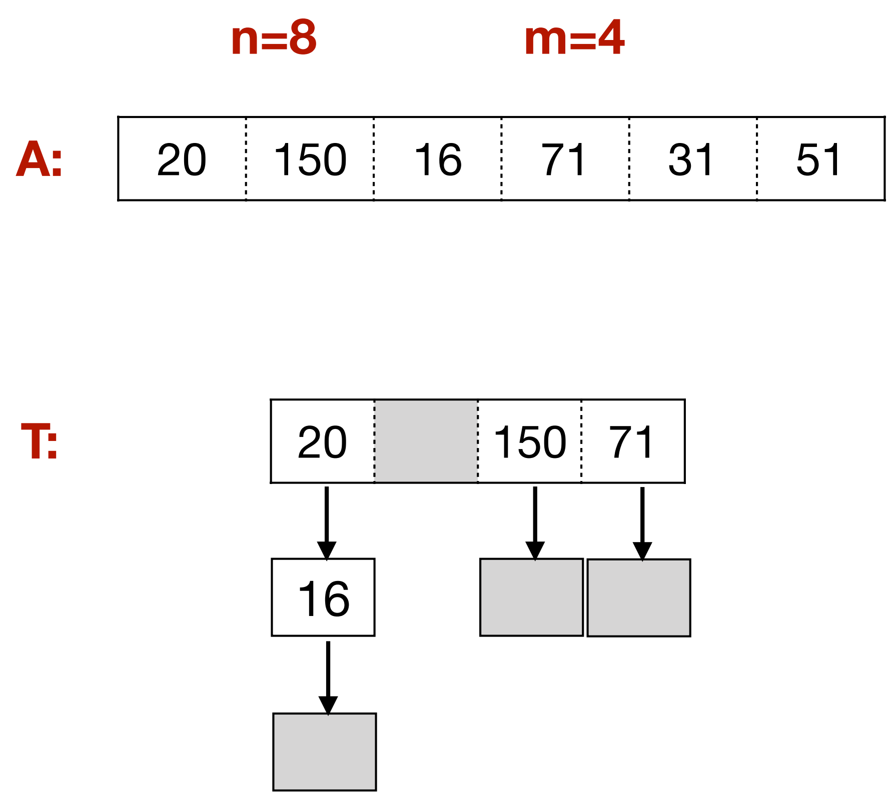
Chaining : Example

- $h(x) = (x \mod 4)$+1
- h(20) = 1
- h(150) = 3
- h(16) =1
- h(71) = 4



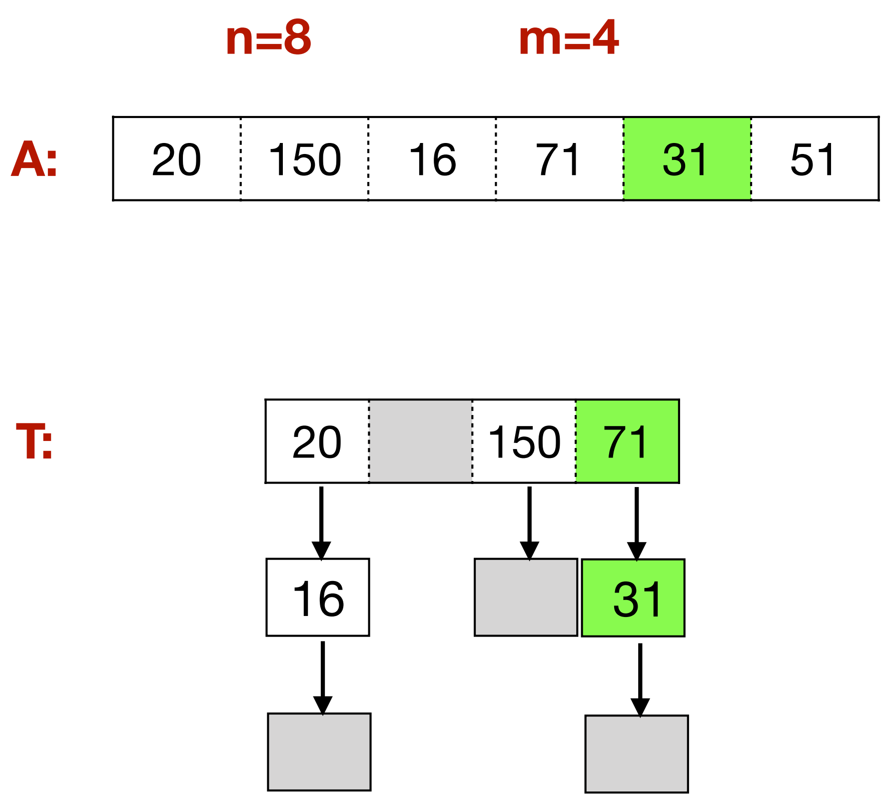
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1
- h(71) = 4



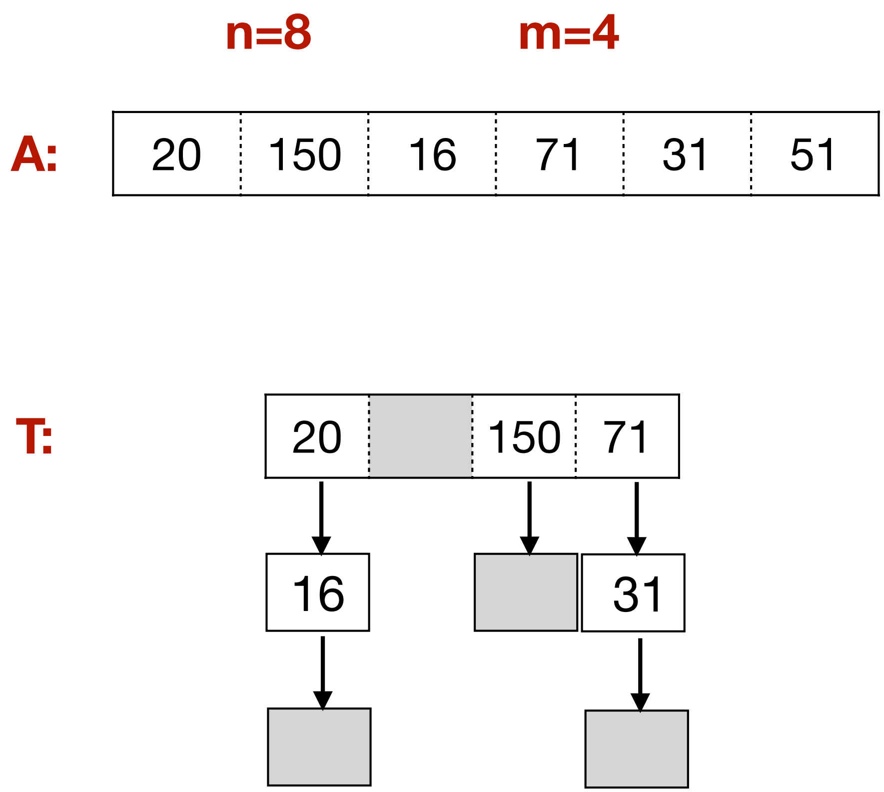
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1
- h(71) = 4
- h(31) = 4



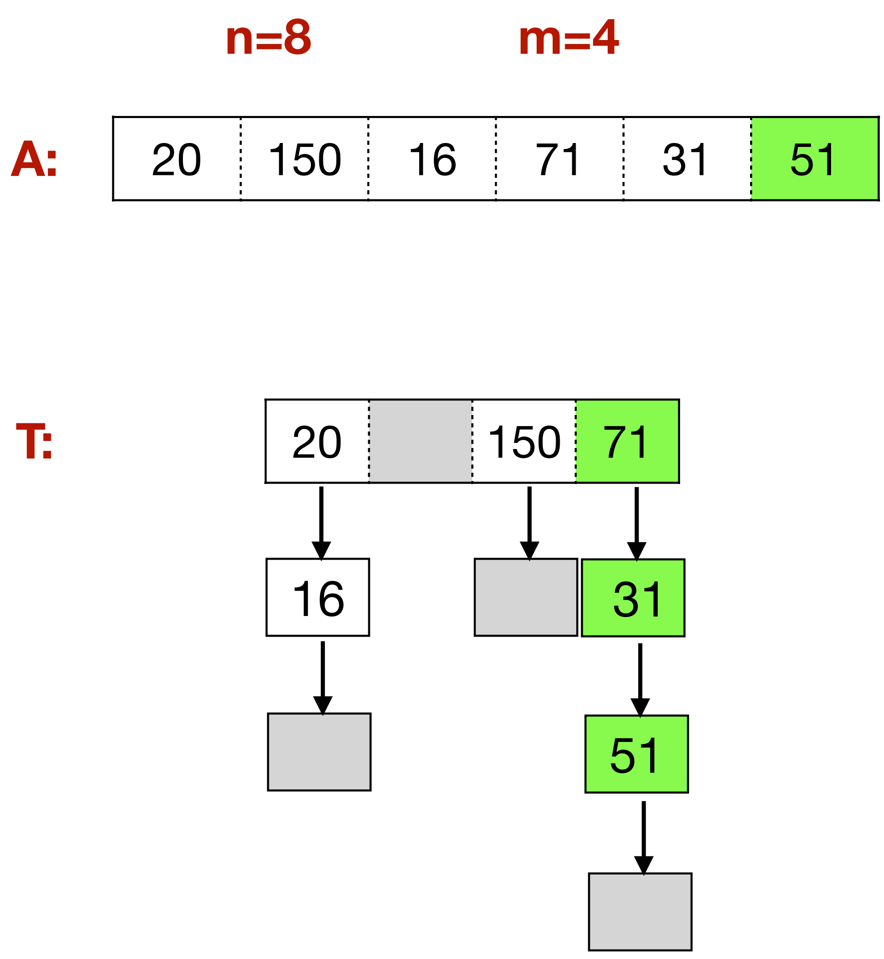
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1
- h(71) = 4
- h(31) = 4



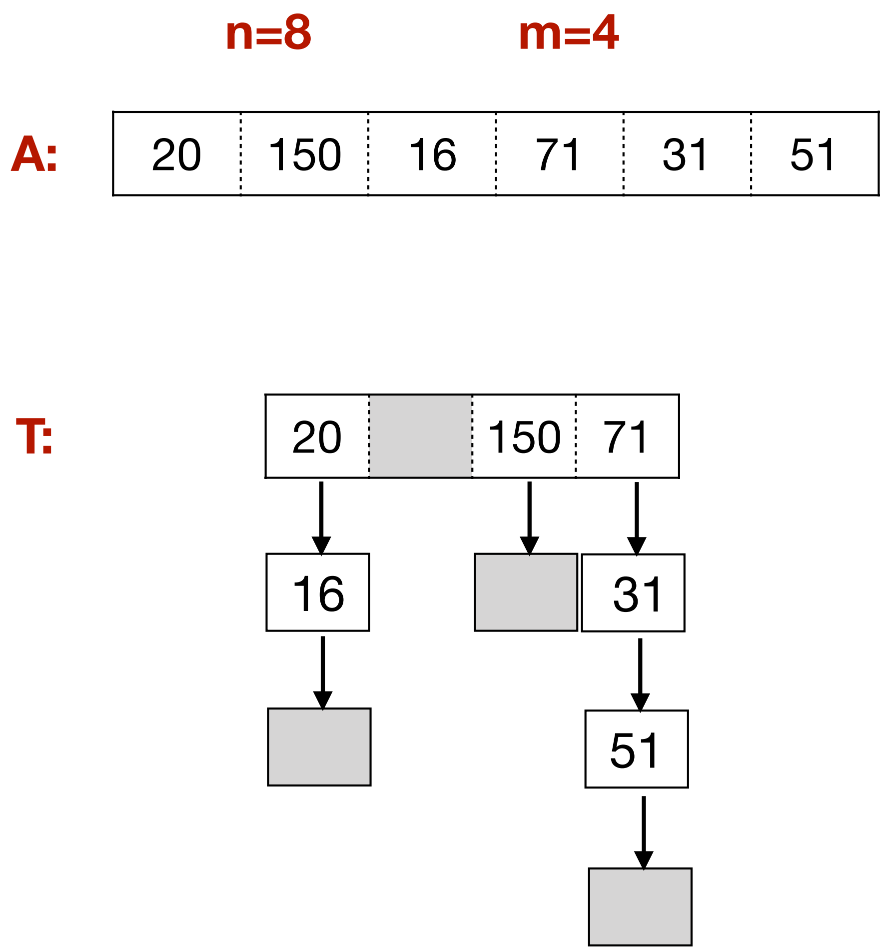
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1
- h(71) = 4
- h(31) = 4
- h(51) = 4



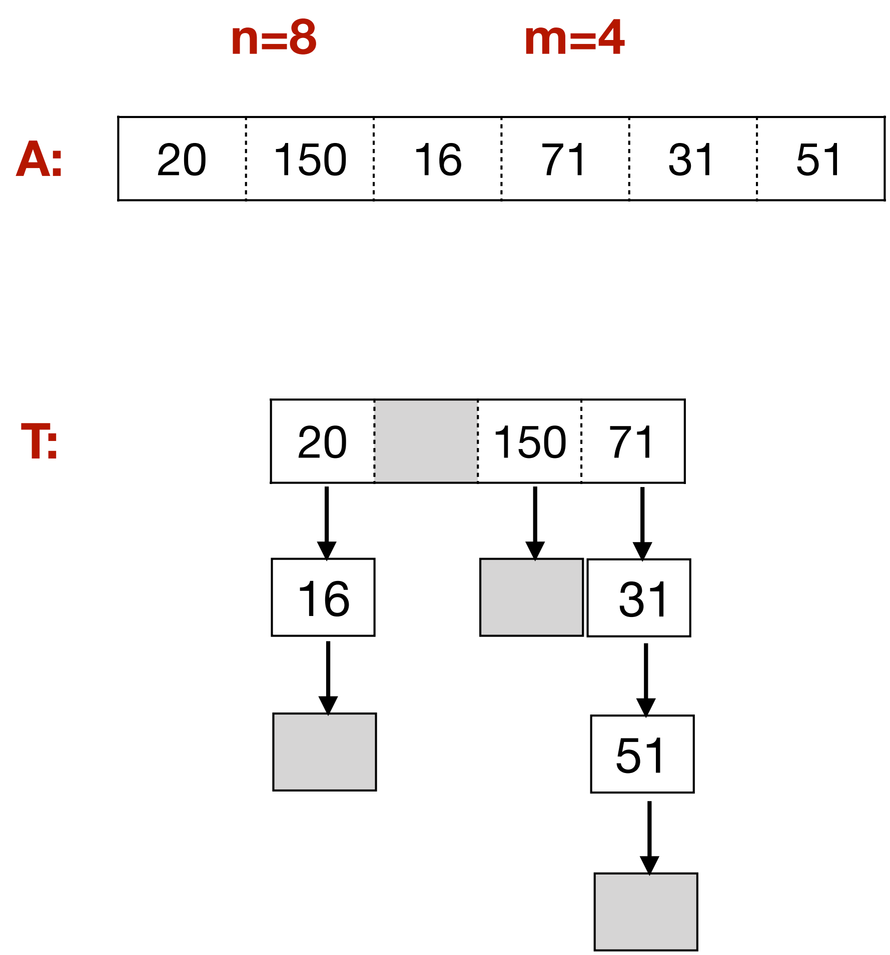
Chaining : Example

- $h(x) = (x \mod 4)+1$
- h(20) = 1
- h(150) = 3
- h(16) = 1
- h(71) = 4
- h(31) = 4
- h(51) = 4



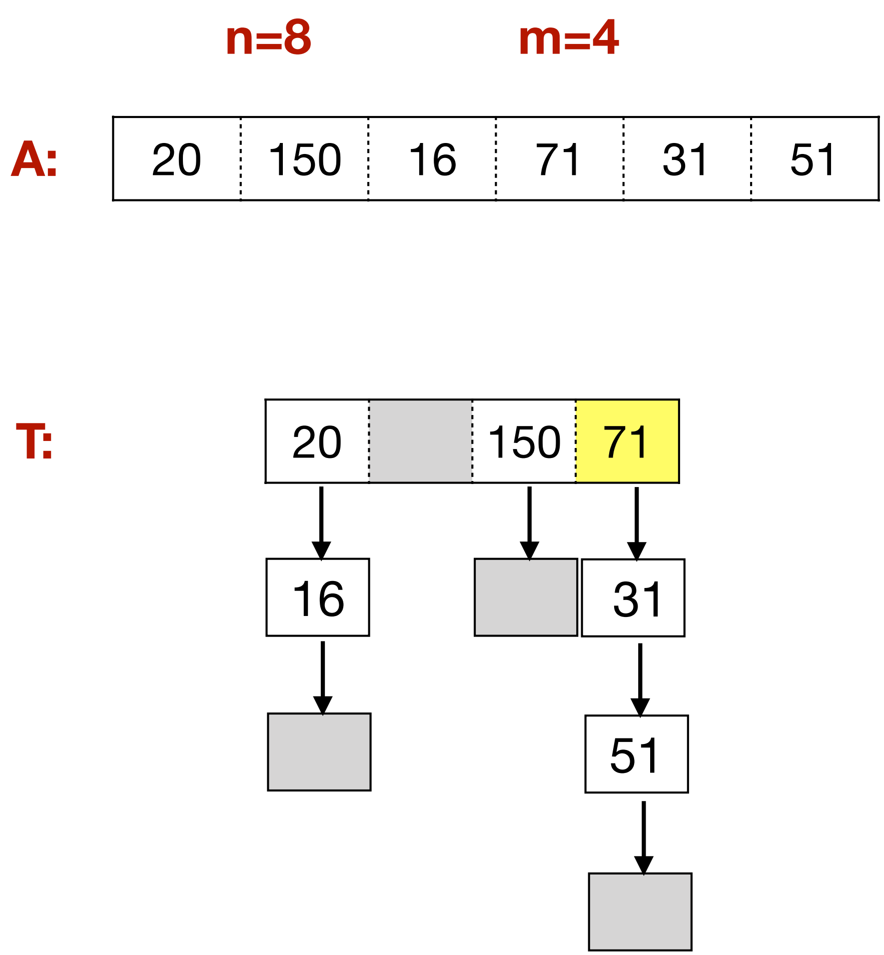
Chaining : Example

- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4



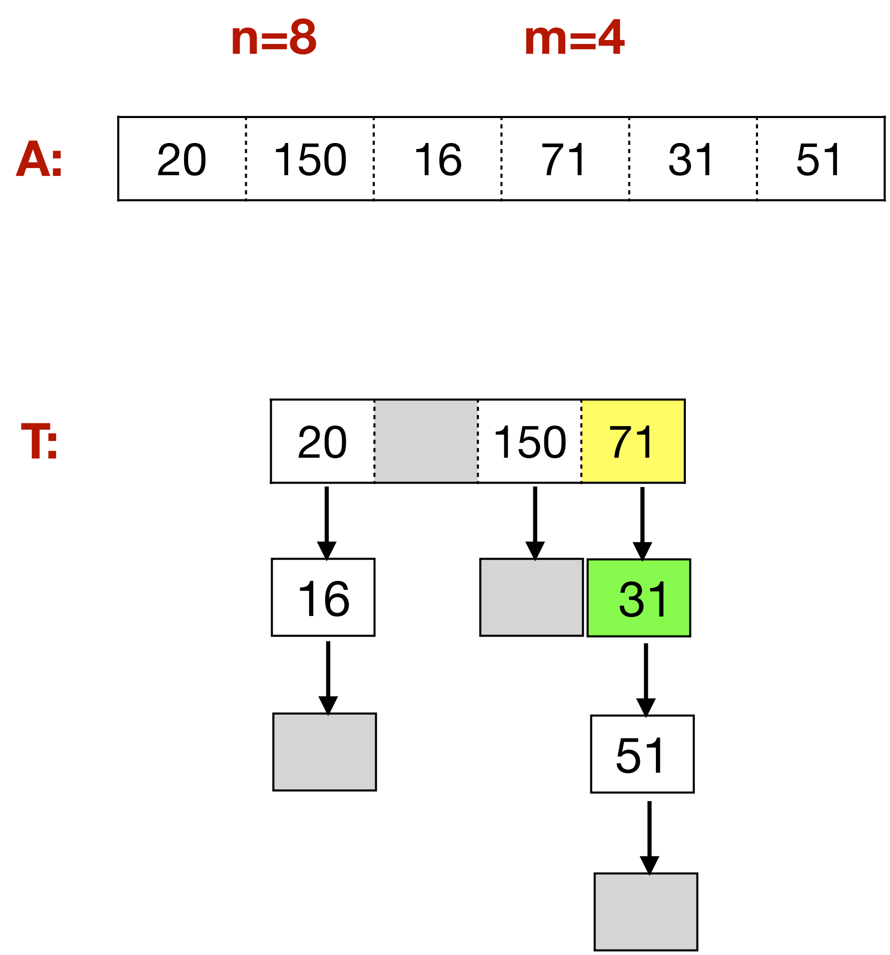
Chaining : Example

- $h(x) = (x \mod 4)$+1
- Search for x=31
  - h(31) = 4



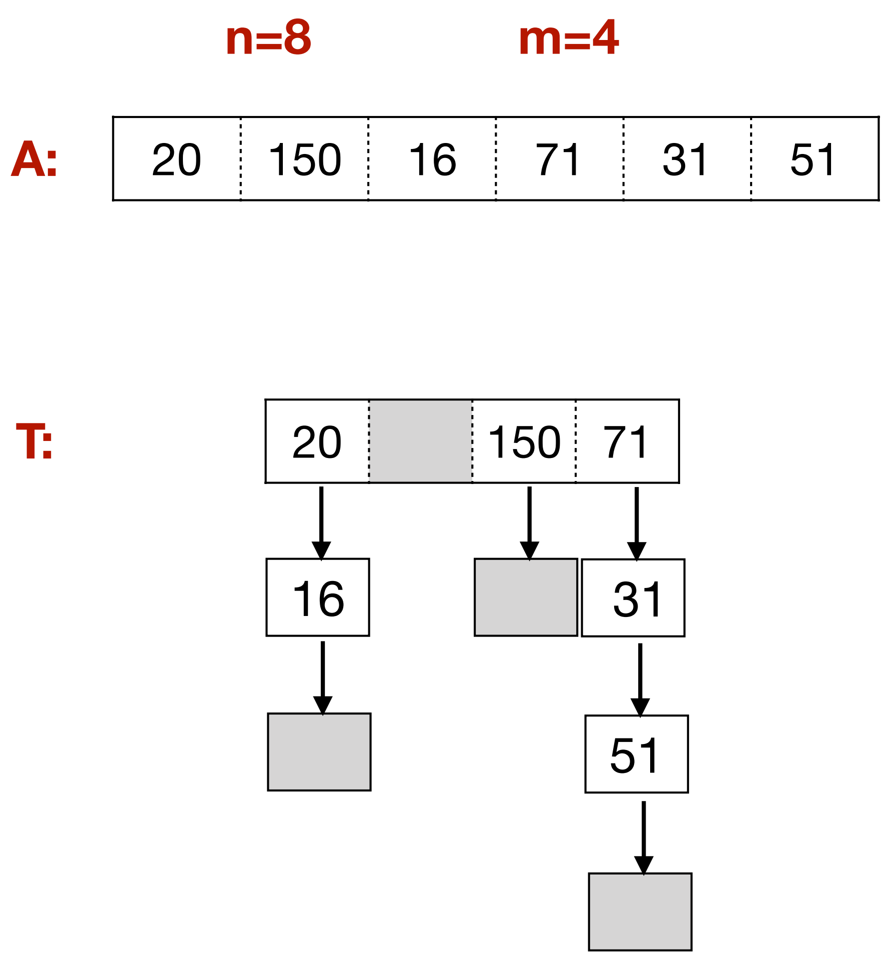
Chaining : Example

- $h(x) = (x \mod 4)$+1
- Search for x=31
  - h(31) = 4



Chaining : Example

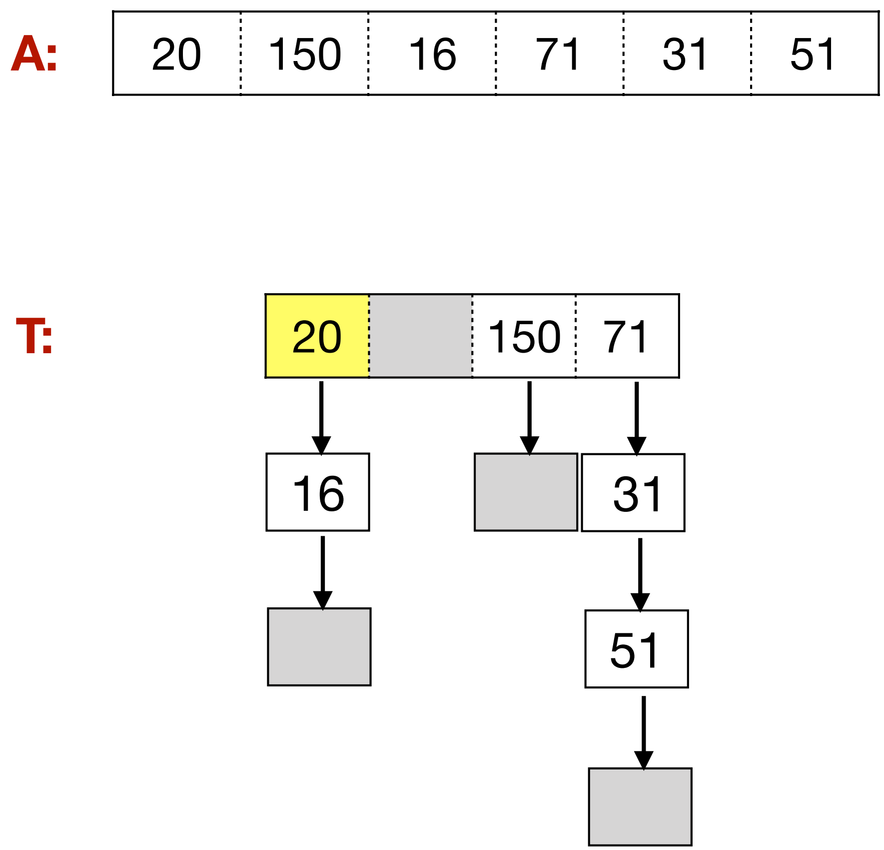
- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4
- Search for x=64
  - h(64)=1



Chaining : Example

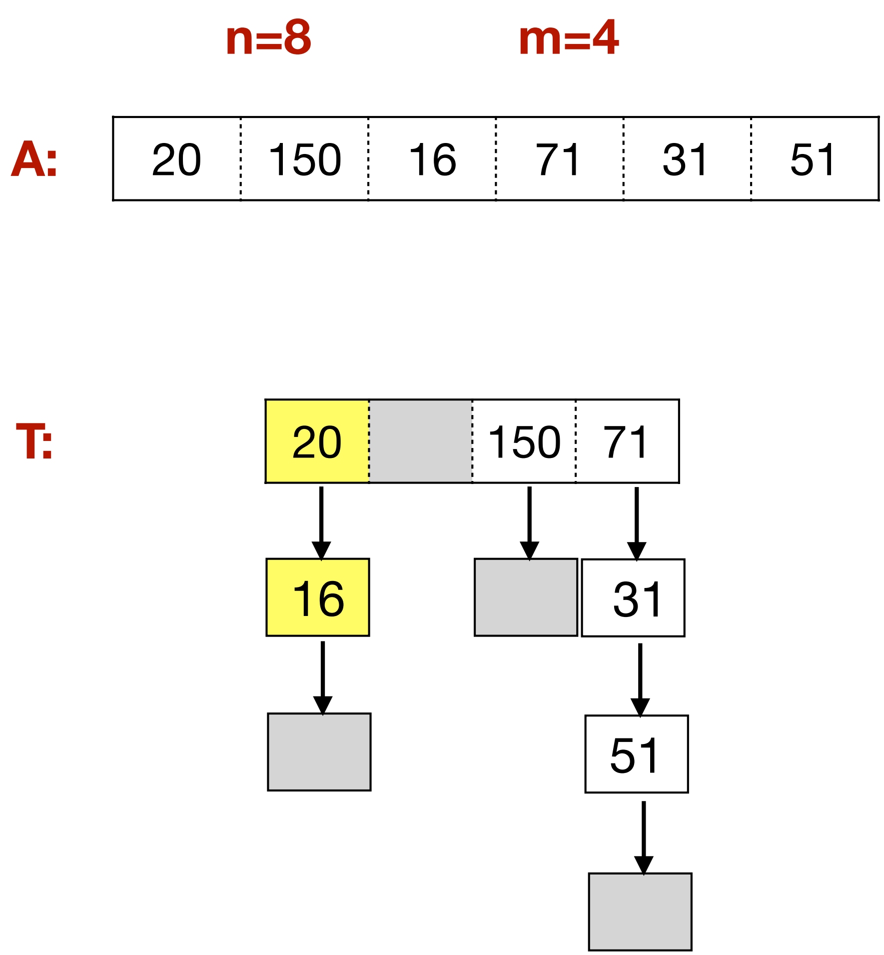
- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4
- Search for x=64
  - h(64)=1

n=8 **m=4**



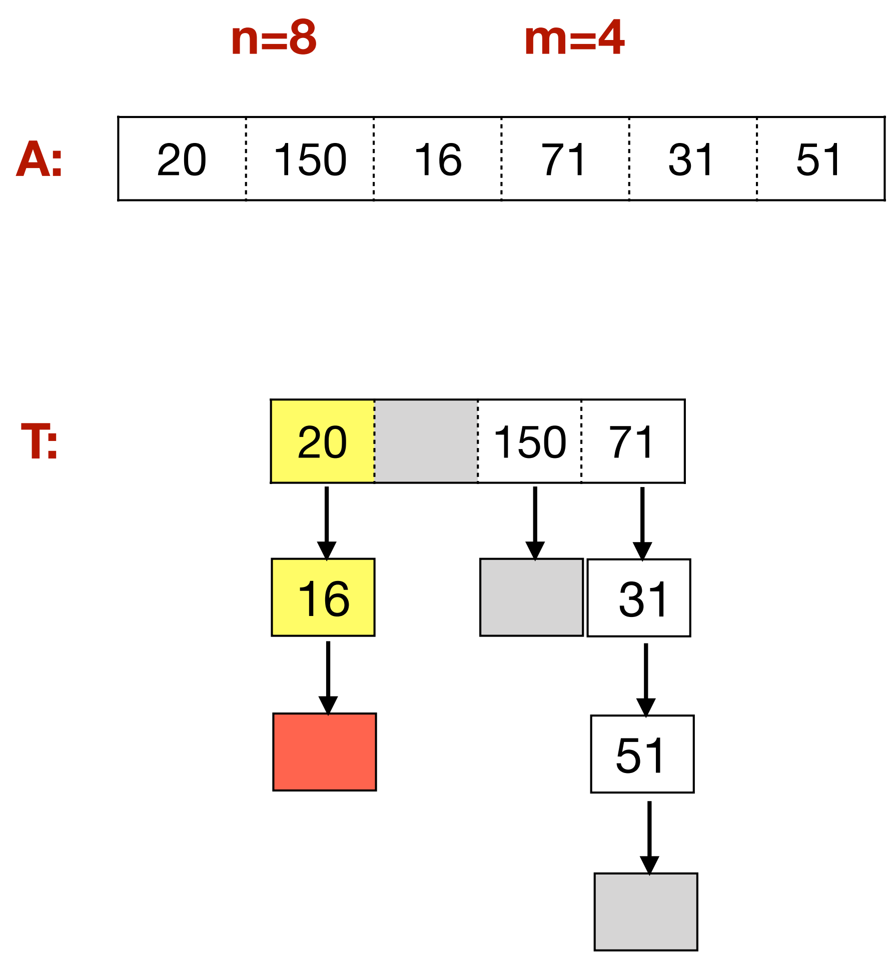
Chaining : Example

- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4
- Search for x=64
  - h(64)=1



Chaining : Example

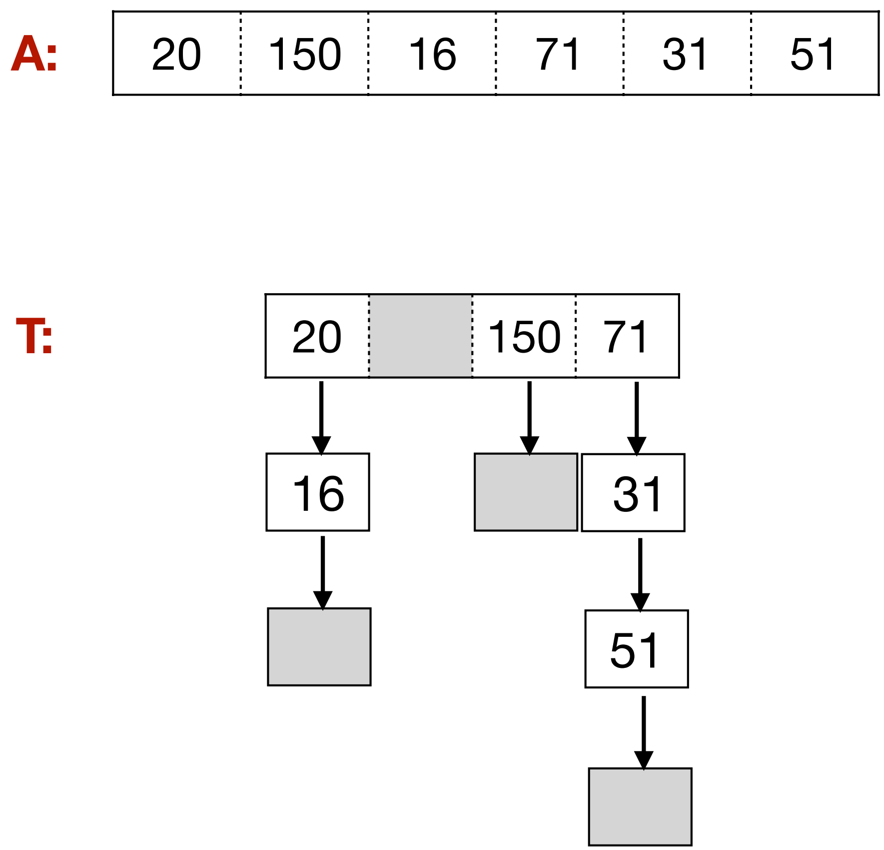
- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4
- Search for x=64
  - h(64)=1



Chaining : Example

- $h(x) = (x \mod 4)+1$
- Search for x=31
  - h(31) = 4
- Search for x=64
  - h(64)=1

n=8 **m=4**



Chaining: Proof of Correctness

- Every element A[i] of is added to the linked-list of T[h(A[i])] and no other number appears in the linked-list
- For any number x, it can only be in the linked-list T[h(x)] and we search the entire list for it

Chaining: Runtime Analysis

- What is worst-case runtime of **search**(x)?

Chaining: Runtime Analysis

- What is worst-case expected runtime of **search**(x)?

Chaining: Runtime Analysis

- Worst-case expected runtime of **search**(x) using near-universal hash functions:
  - Define $\ell(x)$ as the number of elements in A[1:n] mapped to x by the hash function h
  - Runtime of **search**(x) is $O(1 + \ell(x))$
- Worst-case expected runtime of **search**(x) is $O(1 + \mathbf{E}_{h \in \mathscr{H}}[\ell(x)])$

Chaining: Runtime Analysis

- So worst-case expected runtime is:

$$O(1 + \mathbf{E}_{h \in \mathscr{H}}[\ell(x)]) = O(1 + \frac{n}{m})$$

Called **Load**
ratio of number of elements to hash to the size of hash table