بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۲۲

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

Hash Functions and Hash Tables

- A hash function h maps keys of a given type to integers in a fixed interval $[0, N - 1]$
- Example:

  $$h(x) = x \bmod N$$

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

Keys: $\{200, 205, 210, 215, 220, \ldots, 600\}$

- ***N=?***
- ***Other keys?***

Hash Functions and Hash Tables

- A hash function h maps keys of a given type to integers in a fixed interval $[0, N - 1]$
- Example:

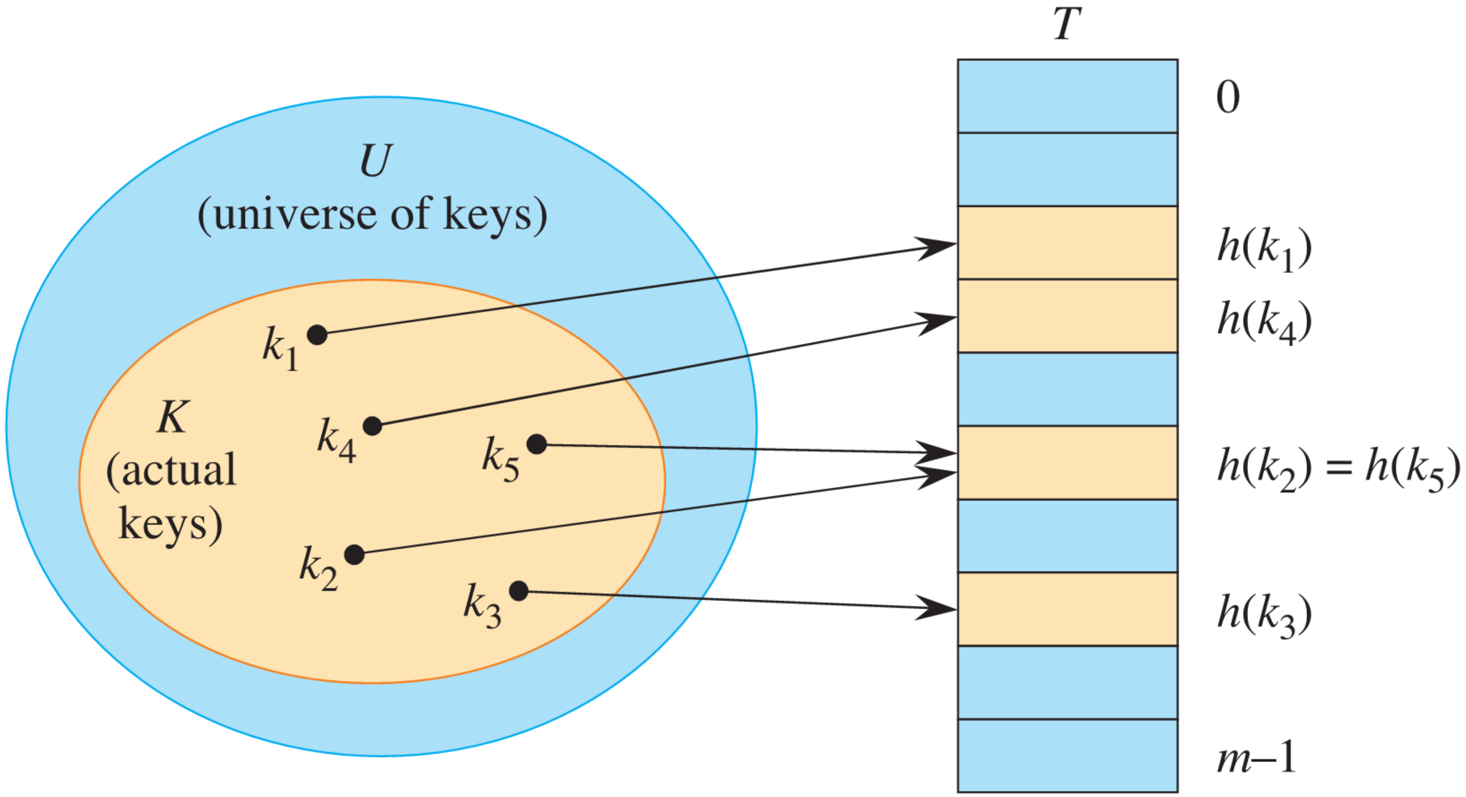
  $$h(x) = x \bmod N$$

  is a hash function for integer keys
- 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 or bucket array) of size N
- When implementing a map with a hash table, the goal is to store item (k, o) at index $i = h(k)$

Hash Functions and Hash Tables

$$h : U \to \{0, 1, \ldots, m-1\} ,$$



Hash Functions

IUT-ECE

- A hash function is usually specified as the composition of two functions:

  Hash code:

  h_1: keys $\rightarrow$ integers

  Compression function:

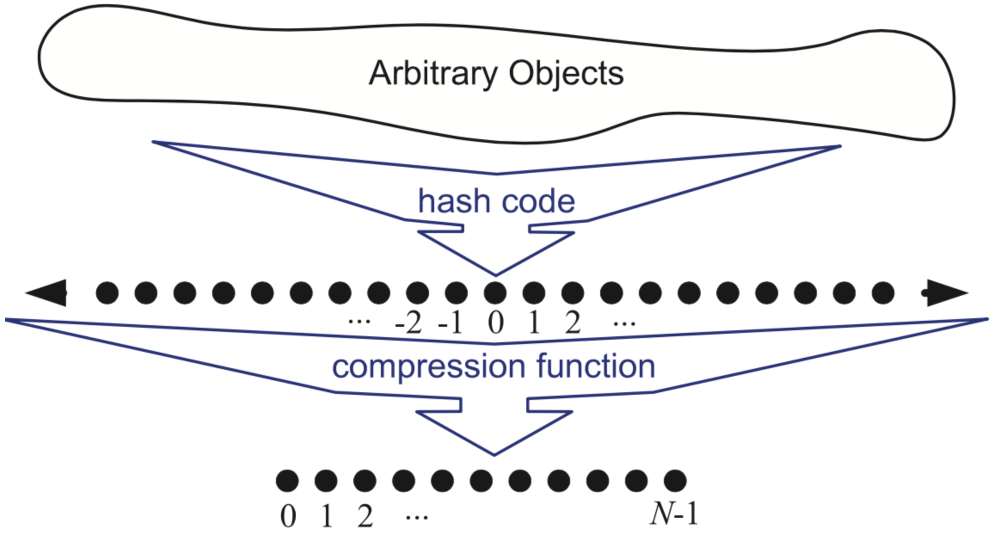
  h_2: integers $\rightarrow$ $[0, N - 1]$

- The hash code is applied first, and the compression function is applied next on the result, i.e.,

  $$h(x) = h_2(h_1(x))$$

- The goal of the hash function is to "disperse" the keys in an apparently random way

(Note) Keys can be arbitrary objects, e.g., string "goodrich"



Hash Code and Compress Function

- There are extensive theoretical and experiment research about "good" hash code and compress functions

- In the next 3 slides,
  - We will discuss some basic hash codes and compress functions.
  - Looking at their more details is not the beyond of our scope.

(1) Hash Codes

IUT-ECE

- Integer cast:
  - We reinterpret the bits of the key as an integer
  - Suitable for keys of length less than or equal to the number of bits of the integer type (e.g., byte, short, int and float in C++)

- Component sum:
  - We partition the bits of the key into components of fixed length (e.g., 16 or 32 bits) and we sum the components (ignoring overflows)
  - Suitable for numeric keys of fixed length greater than or equal to the number of bits of the integer type (e.g., long and double in C++)
  - But, not good for strings
    - “temp01” and “temp10”
    - “stop”, “tops”, “pots”, “spot”

Polynomial Hash Code

- Polynomial accumulation:
  - We partition the bits of the key into a sequence of components of fixed length (e.g., 8, 16 or 32 bits)

    $$a_0\, a_1\, \ldots\, a_{n-1}$$

  - We evaluate the polynomial

    $$p(z) = a_0 + a_1 z + a_2 z^2 + \ldots$$
    $$\ldots + a_{n-1} z^{n-1}$$

    at a fixed value z, ignoring overflows
  - Especially suitable for strings (e.g., the choice $z = 33$ gives at most 6 collisions on a set of 50,000 English words)

- Polynomial $p(z)$ can be evaluated in $O(n)$ time using Horner's rule:
  - The following polynomials are successively computed, each from the previous one in $O(1)$ time

    $$p_0(z) = a_{n-1}$$
    $$p_i(z) = a_{n-i-1} + z p_{i-1}(z)$$
    $$(i = 1, 2, \ldots, n-1)$$

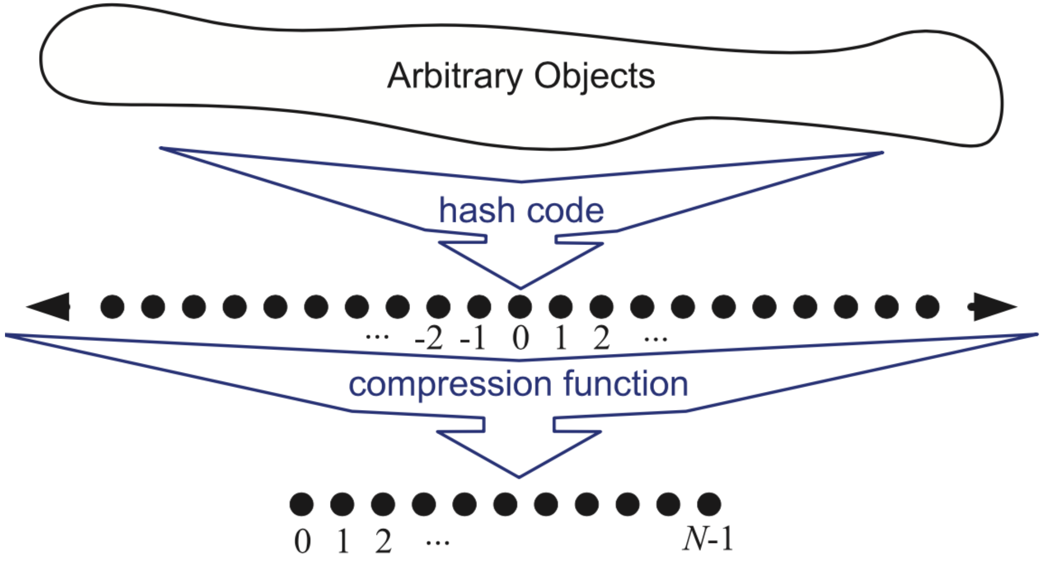
- We have $p(z) = p_{n-1}(z)$
- Lots of research about "good hash code"

(2) Compression Functions

IUT-ECE

Division:

- $h_2(y) = |y| \bmod N$
- 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



Keys: $\{200, 205, 210, 215, 220, \ldots, 600\}$

- N=100 or N=101?

(2) Compression Functions

IUT-ECE

- **Division:**
  - $h_2(y) = |y| \bmod N$
  - 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

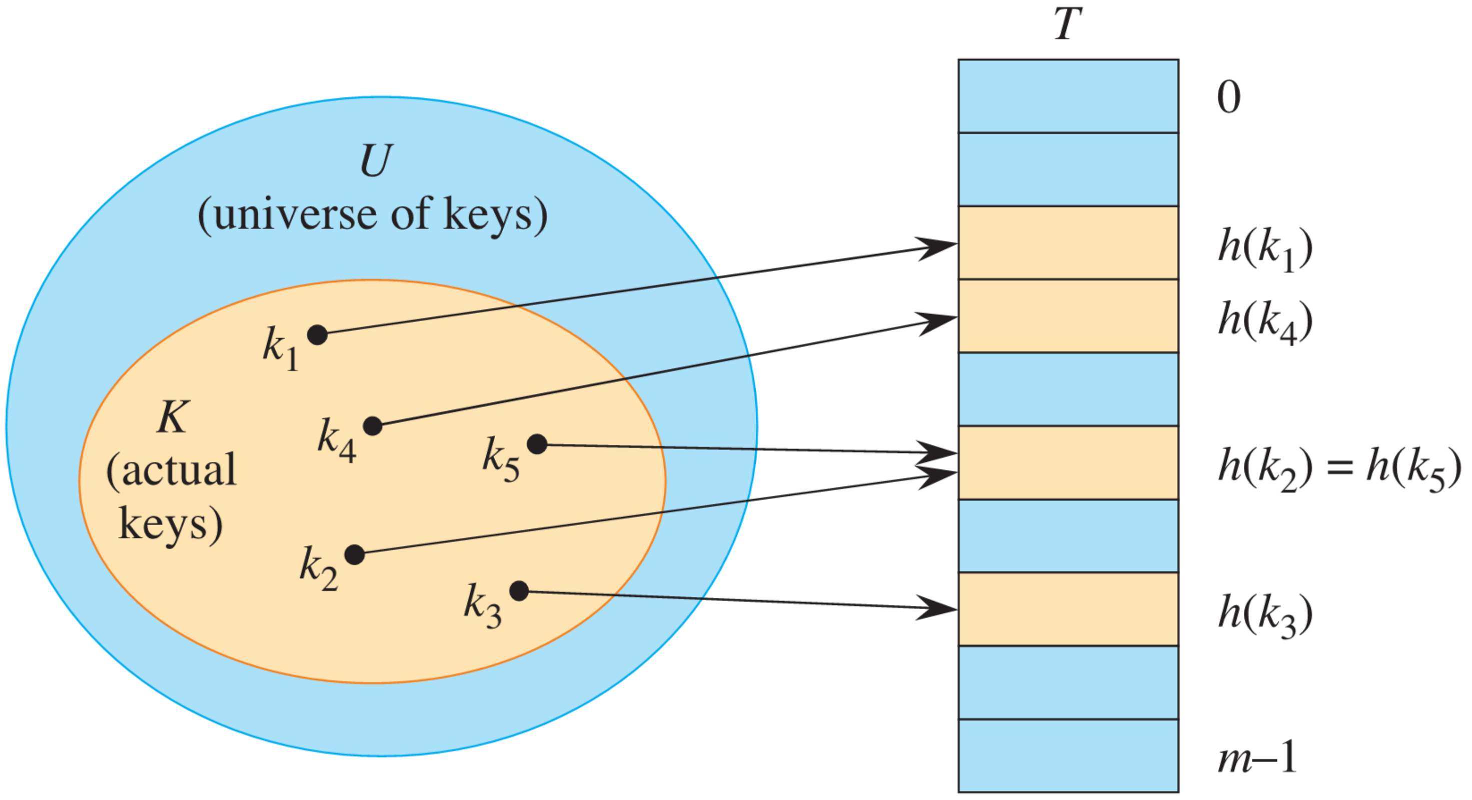
- **Multiply, Add and Divide (MAD):**
  - $h_2(y) = |ay + b| \bmod N$
  - a and b are nonnegative integers such that

    $$a \bmod N \neq 0$$

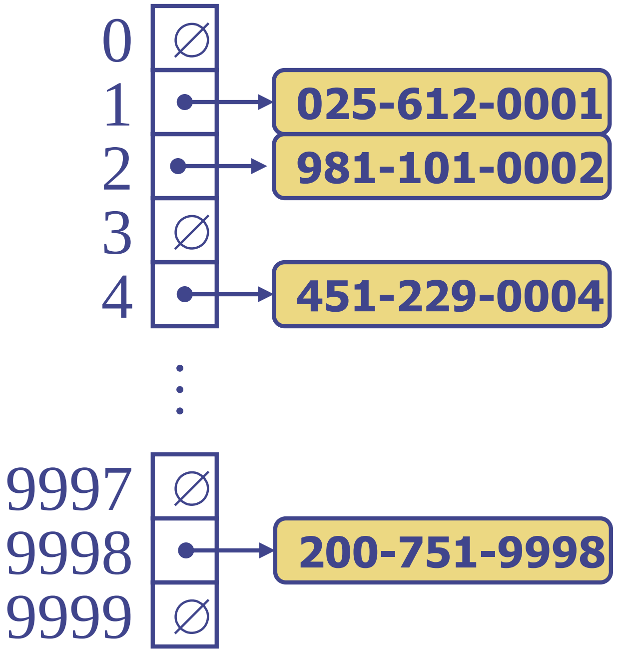
    - Otherwise, every integer would map to the same value b

Hash Functions and Hash Tables

$$h : U \rightarrow \{0, 1, \ldots, m-1\} \ ,$$



Collision Handling



Insert the entry: 032-637-0004

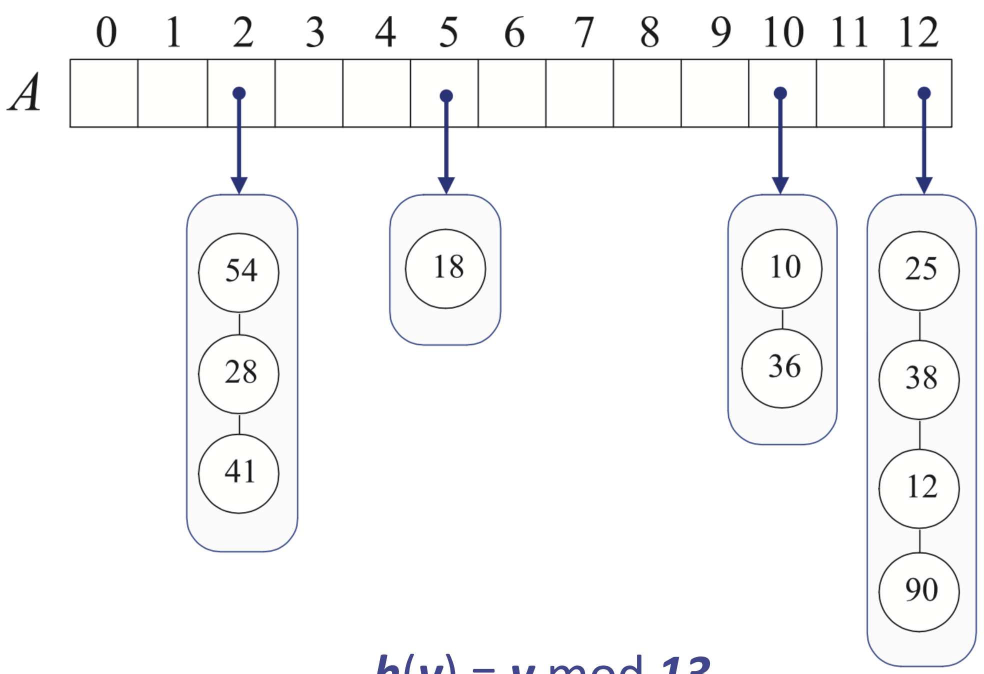
Collisions occur when different elements are mapped to the same cell

Collision Handling

IUT-ECE

- Separate Chaining: let each cell in the table point to a linked list of entries that map there

- Separate chaining is simple, but requires additional memory outside the table



$h(y) = y \bmod 13$

Collision Handling

Separate Chaining (functions):

Algorithm $\mathsf{find}(k)$:

Output: The position of the matching entry of the map, or end if there is no key k in the map

return $A[h(k)].\mathsf{find}(k)$ {delegate the $\mathsf{find}(k)$ to the list-based map at $A[h(k)]$}

Algorithm $\mathsf{put}(k, v)$:

$p \leftarrow A[h(k)].\mathsf{put}(k, v)$ {delegate the put to the list-based map at $A[h(k)]$}

$n \leftarrow n + 1$

return p

Algorithm $\mathsf{erase}(k)$:

Output: None

$A[h(k)].\mathsf{erase}(k)$ {delegate the erase to the list-based map at $A[h(k)]$}

$n \leftarrow n - 1$

Collision Handling

❖ Separate Chaining (analysis):

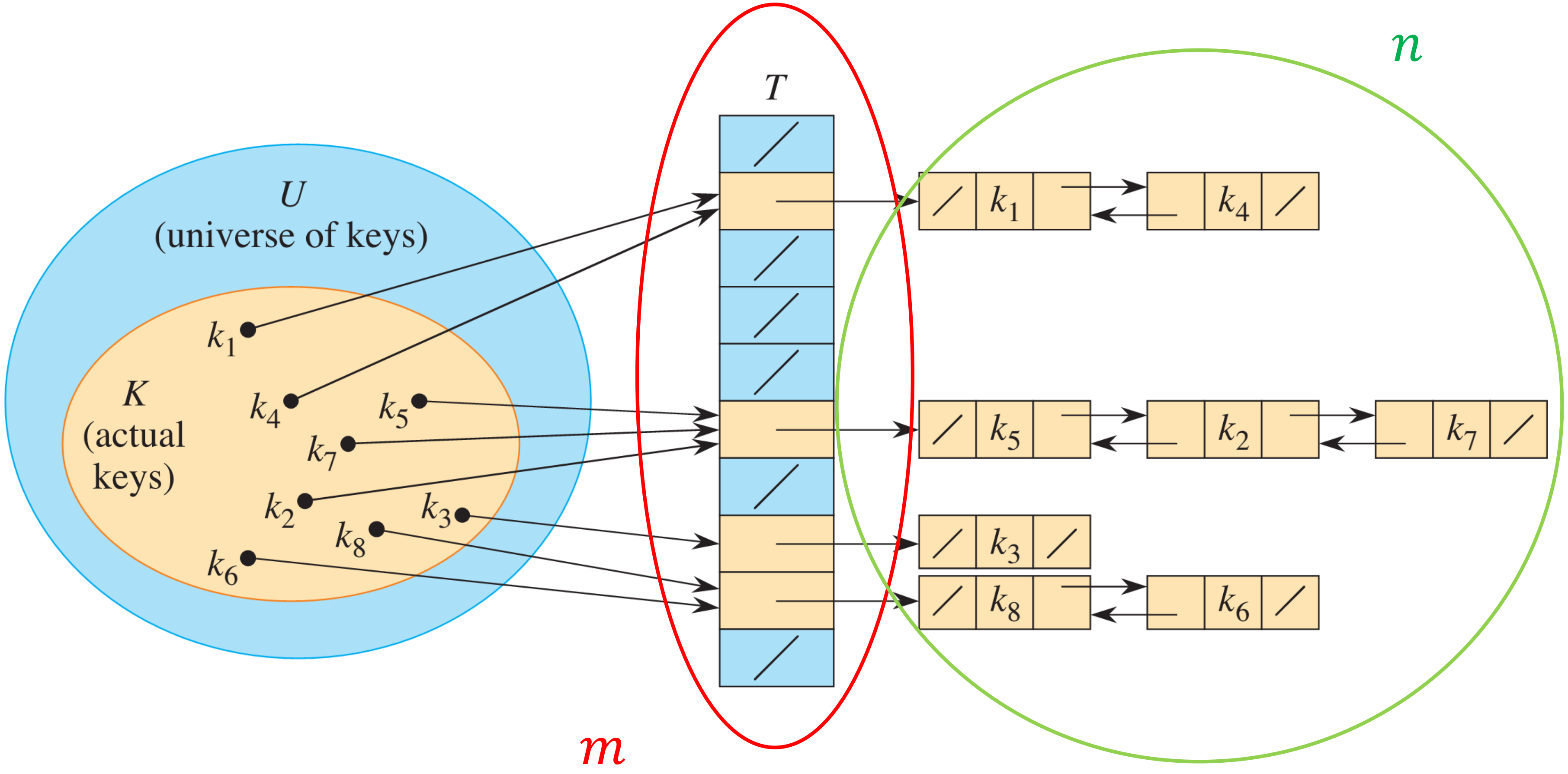
Given a hash table T with m slots that stores n elements, we define the ***load factor*** α for T as n/m, that is, the average number of elements stored in a chain. Our analysis will be in terms of α, which can be less than, equal to, or greater than 1.

$$\alpha = \frac{n}{m}$$

Collision Handling

◆ Separate Chaining (analysis):

denote the length of the list $T[j]$ by n_j



$$\alpha = \frac{n}{m}$$

Collision Handling

Separate Chaining (analysis):

independent uniform hashing:

the chance that any two distinct keys k_1 and k_2 collide is at most $1/m$.

Collision Handling

❖ Separate Chaining (analysis):

Theorem 11.1
In a hash table in which collisions are resolved by chaining, an unsuccessful search takes $\Theta(1 + \alpha)$ time on average, under the assumption of independent uniform hashing.

Proof Under the assumption of independent uniform hashing, any key k not already stored in the table is equally likely to hash to any of the m slots. The expected time to search unsuccessfully for a key k is the expected time to search to the end of list $T[h(k)]$, which has expected length $\mathrm{E}[n_{h(k)}] = \alpha$. Thus, the expected number of elements examined in an unsuccessful search is α, and the total time required (including the time for computing $h(k)$) is $\Theta(1 + \alpha)$. ■

Collision Handling

◆ Separate Chaining (analysis):

Theorem 11.2

In a hash table in which collisions are resolved by chaining, a successful search takes $\Theta(1 + \alpha)$ time on average, under the assumption of independent uniform hashing.