Advanced Programming Practice

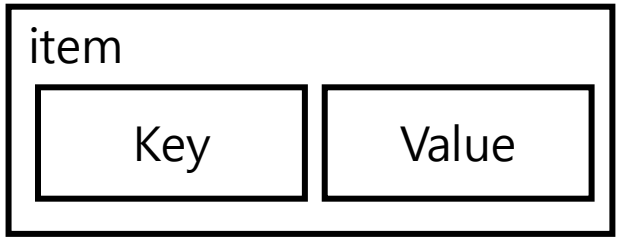
Hashing

2022 Fall

Sogang University

Dictionary Problem

- Dictionary is an abstract object that maintains set of items.
  - Each item is with a key.
- Three operations able to execute:
  - insert (item): add item to set
  - Delete (key): get rid of item from set
  - Search (key): return item with key if it exists
- Items are (key, value) pairs



Dict = {('Sogang', 1), ('nice', 64)}

Dict.items() → **{('Sogang', 1), ('nice', 64)}**
Dict['nice'] → 64
Dict[64] → **KeyError**
'nice' in Dict → **Dict**
64 in Dict → **False**

Where we use dictionaries?

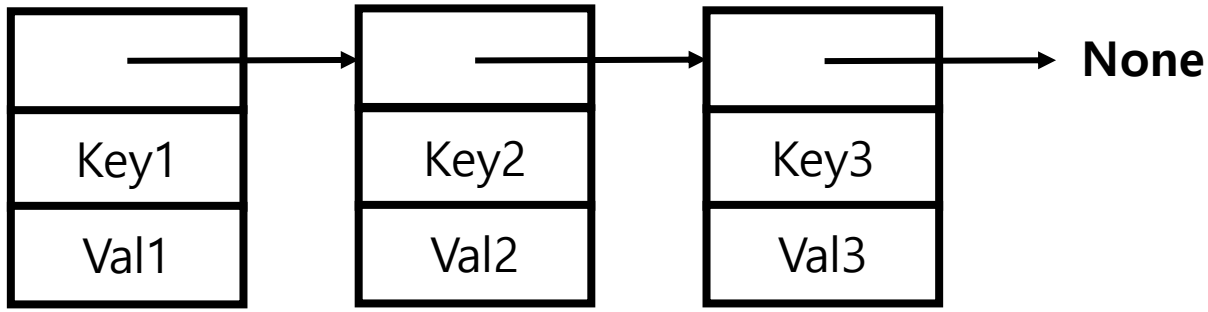
- Document distance (Docdist): word counting & inner product.
- Implementing databases
  - English Dictionary: English word → definition
  - English words: for spelling correction: wrong word → correct word
  - Search engine: word → all webpages containing that word
  - Log-in server: username → account object
- Compilers & interpreters: names → variables
- Network routers: IP address → wire
- Network server: Port number → socket/app
- Virtual memory: Virtual address → physical address

THE MOST POPULAR DATA STRUCTURE IN CS

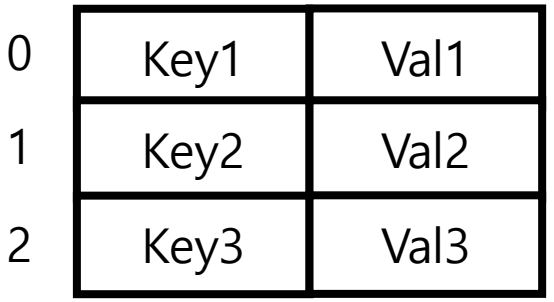
How do we solve the dictionary problem?

- How to make dictionary operations as efficient as possible?

- Approach #1: List
  - Insertion time: O(1)
  - Deletion time: O(n)
  - Searching time: **O(n)**



- Approach #2: 1D array
  - Insertion time: O(1)
  - Deletion time: O(n) ← why?
  - Searching time: **O(n)**



How do we solve the dictionary problem?

- How about with efficient data structure like Binary Search Tree (BST)?

- Approach 3: AVL tree with keys (self-balancing binary search tree)
  - Insertion time: O(log n)
  - Deletion time: O(log n)
  - Searching time: **O(log n)**

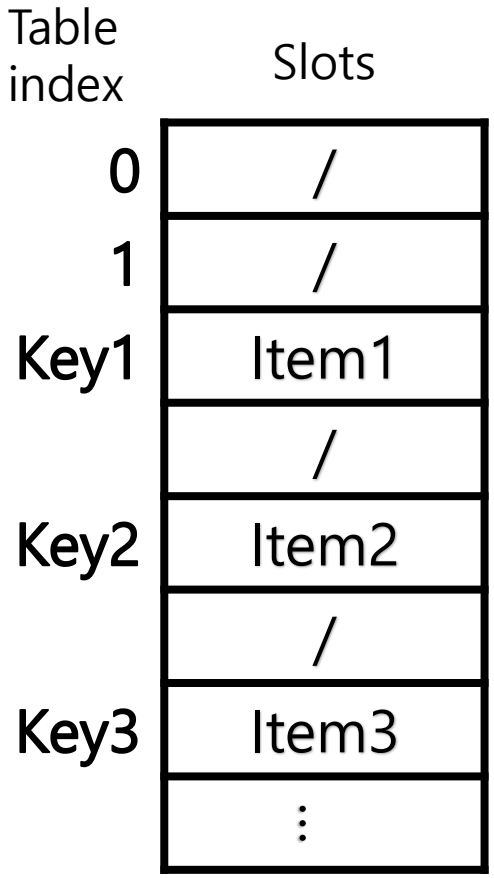
- Those method is based on comparison only.

AVL tree from Wikipedia

How do we solve the dictionary problem?

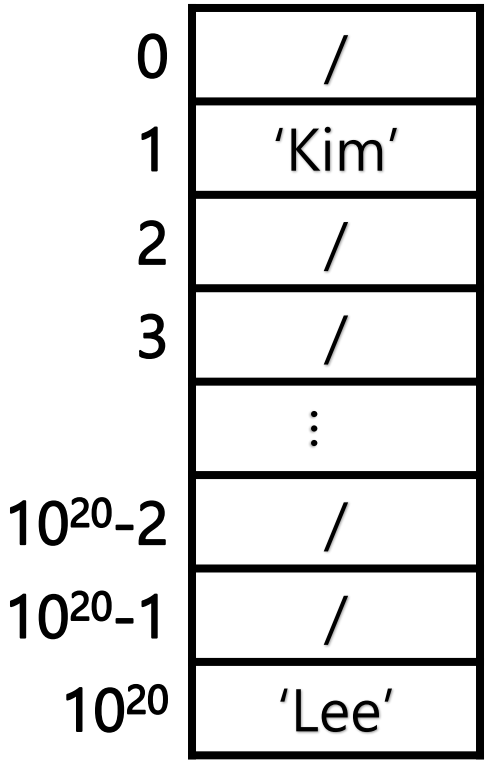
- Approach 4. Direct-access table
  - If a key is an positive integer, store items in an array indexed by key
  - **Random access**
    - Access in constant time O(1)
    - Contrast to sequential access O(n)
  - Insertion time: **O(1)**
  - Deletion time: **O(1)**
  - Searching time: **O(1)**

- What is a problem with it?



Problem of Direct Access Table

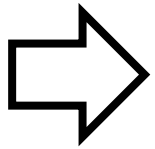
1. Key must be a non-negative integer!
2. If we have items (1, ‘Kim’) and (10^{20}, ‘Lee’), the storage needs 10^{20} slots though only two slots are used in the array!
   - Spatial Complexity O(K) where K is the limit of the key.



Remedy for Direct Access Table

- Solution for 1: map keys to integers
  - Theoretically, keys are finite → set of keys is countable!
  - Since all data types (real number, integer, string, and objects) in computer are represented by the finite number of bits, possible to map values of those types to integers.

23241.234234
"SOGANG IS NOT GANG"
-31413
string("It is an object")

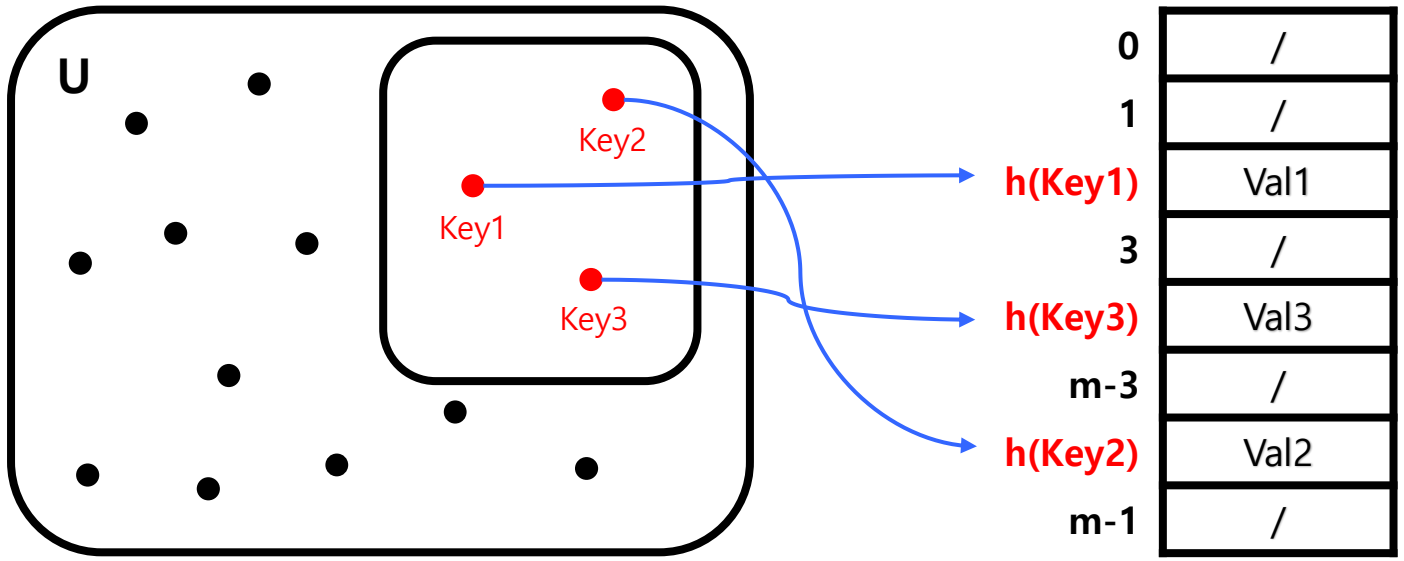


01101010111010101010101010
11010101011110000010010 1
10010011011110100000010 1
00100101001010001001000 1
01101011010101110101010 0
10100101010101011010110 1
01011010010010…

Remind Cantor's theorem

Remedy for Direct Access Table

- Solution to 2: Hashing
  - Need to map universe U of keys to a set of integers with small size m for table.
  - Idea: m ≈ n = # keys stored in dictionary
  - Hash function h: U → { 0,1, 2, 3, … , m-1 }

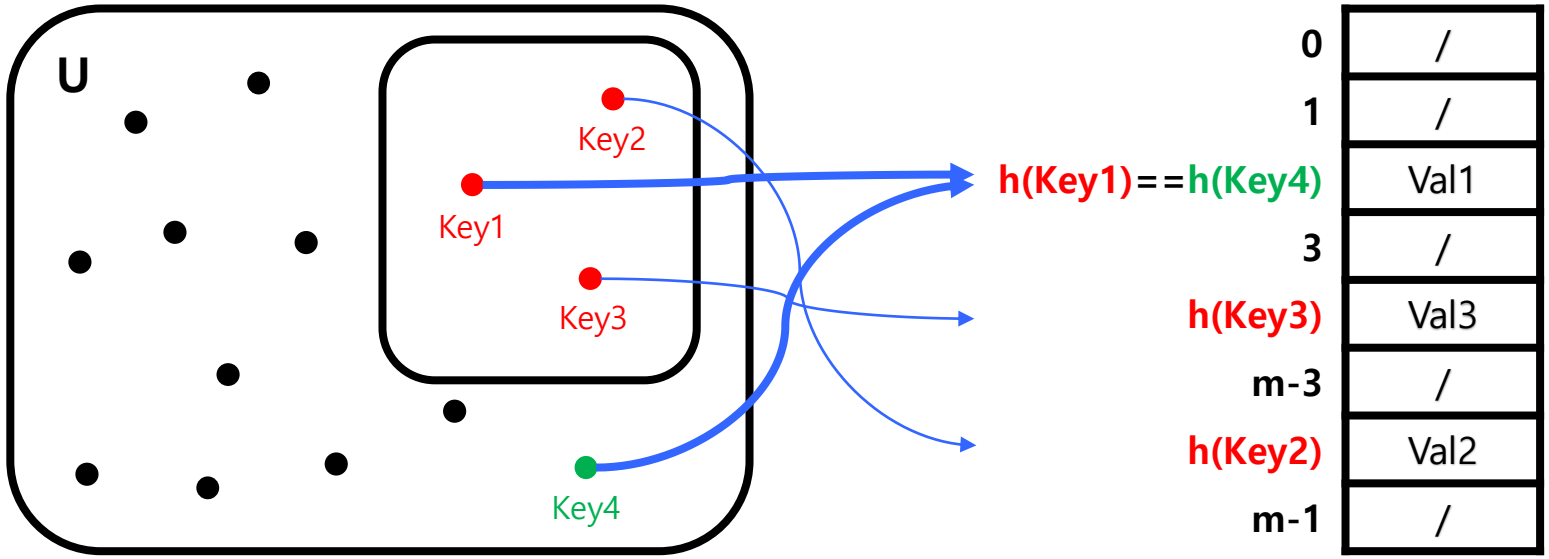


Hash (verb)
To chop into small pieces, to make into a hash.
Hash (noun)
1. Food, especially meat and potatoes, chopped and mixed together.
2. A confused mess

Corn beef hash

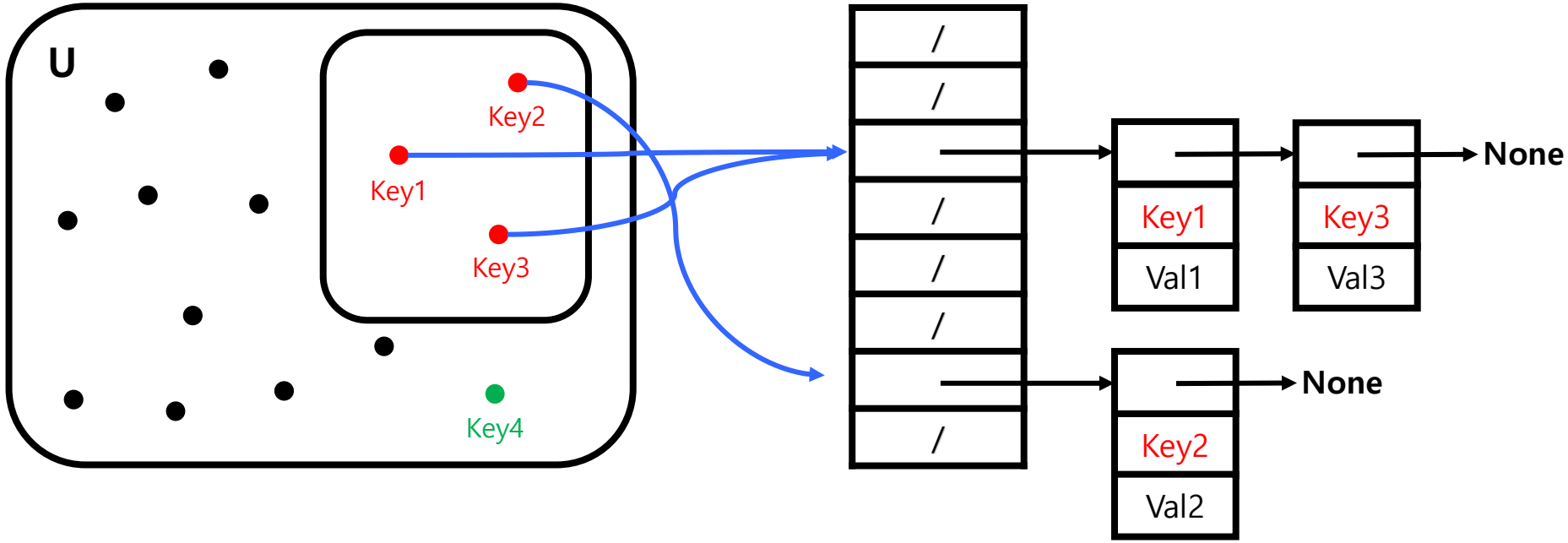
Collide in hashing

- Collide: two keys are collide if $h(k_i)=h(k_j)$.
- Collide is inevitable!
  - Due to the pigeonhole principle



Chaining

- Storing colliding elements in an linked list of each slot of table
  - Search must go through all elements in the linked list T[h(key)]
  - Worst case: all keys have the same hash key ➔ Θ(n) per operation



Simple Uniform Hashing

- Not true but easy to derive hashing properties.
- Every key is uniformly hashed to a slot and independent.
- Load factor = # keys stored in table / # slots in table
  - Expected number of keys per slot is the expected length of a chain.
  - Search time = Θ(random access time + search time) = Θ(1+load factor)

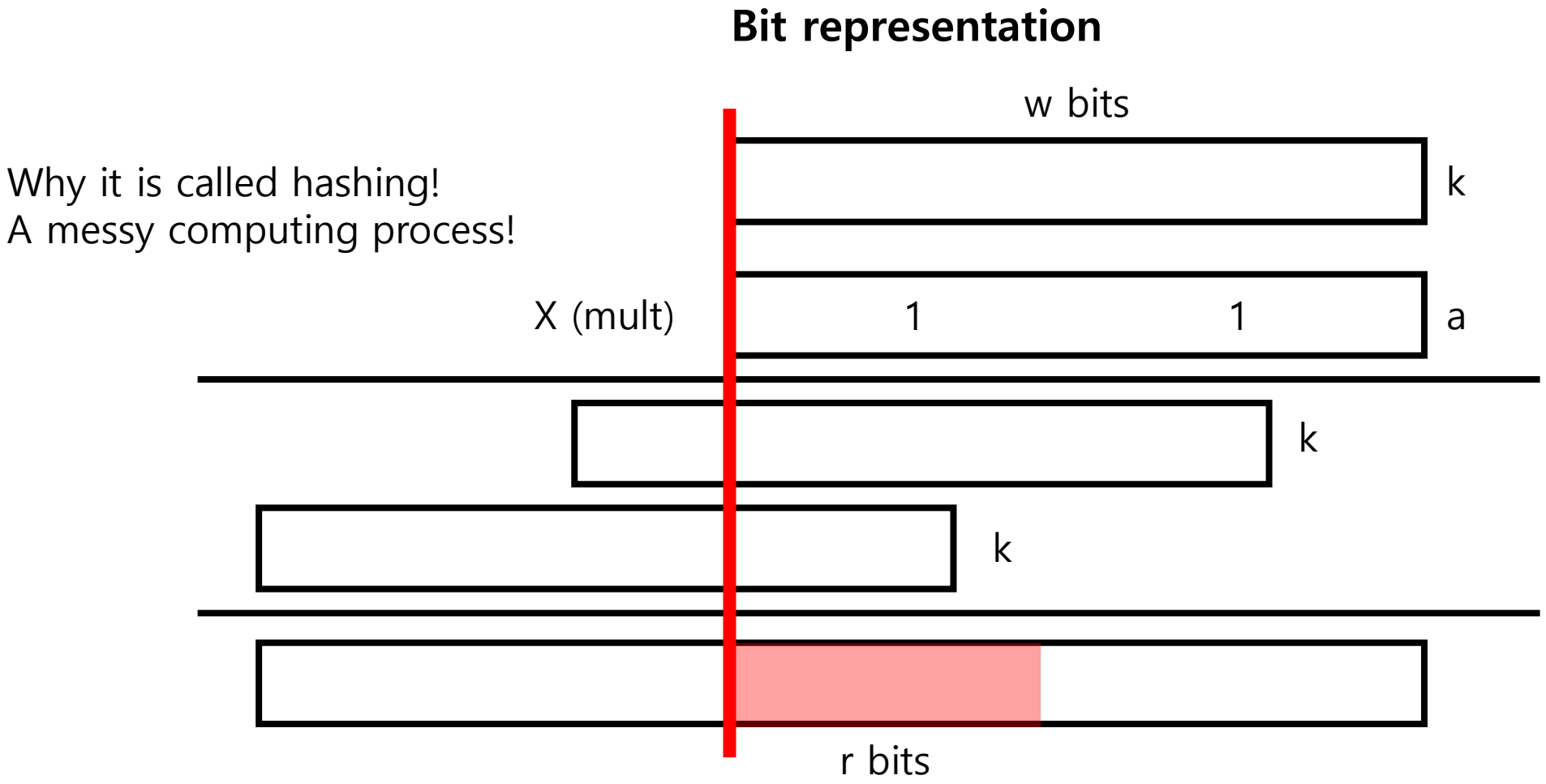
    = O(1) if load factor = O(1) that is m= Ω(1)

Making a Hash Function

- Division method: $h(k) = k \bmod m$
  - Practical when m is prime
  - But not to close to power of 2 or 10 (depending on only low bits or digits)
- Multiplication method: $h(k) = [\ (ak) \bmod 2^w\ ] >> (w-r)$
  - a is random (sampled at first)
  - Practical when a is odd and $2^{w-1} < a < 2^w$, not to close power of 2
  - Fast
- Universal hashing: $h(k) = [(ak+b) \bmod p] \bmod m$
  - a and b is random in {1,…,p-1}, p is large prime so bigger than the number of keys.
  - Satisfy that probability of (a, b) such that h(k1)=k(k2) is 1/m.
  - Collide expectation of a slot over a and b is load factor = n/m.

Multiplication Method

$$h(k) = [\,(ak) \bmod 2^w\,] >> (w-r)$$



Assignments

Exercise 1. Implement a hash function

- There are three types of a hashing function.
  - Division method: h(k) = k mod m
  - Multiplication method: h(k) = [(ak) mod 2^w]>>(w-r)
  - Universal hashing: h(k)= [(ak+b) mod p] mod m

- You task is to implement each hashing function and analyze hashing functions with sequential or random access.

- When the division method works badly? What hashing functions promise uniformity in general cases?

Exercise 2. Implement your own dictionary

- Time to comment in exercise2() in main function.
- Implement dictionary (key: string, item: integer) with a hash function
- We assume that strings are composed of only lower-case alphabets.
- Your task is to implement following three member functions.
  - insert(string key, int item): store item with key
  - erase(string key): delete the item with key in the set
  - find(string key): return the item with key
- Each slot has a linked list
- Two steps to compute a key value from string
  - Hashing from string to integer
  - Hashing from integer to m-integers
- For hashing from string to integer, refer to https://cp-algorithms.com/string/string-hashing.html#calculation-of-the-hash-of-a-string