# Hashing

# Motivation

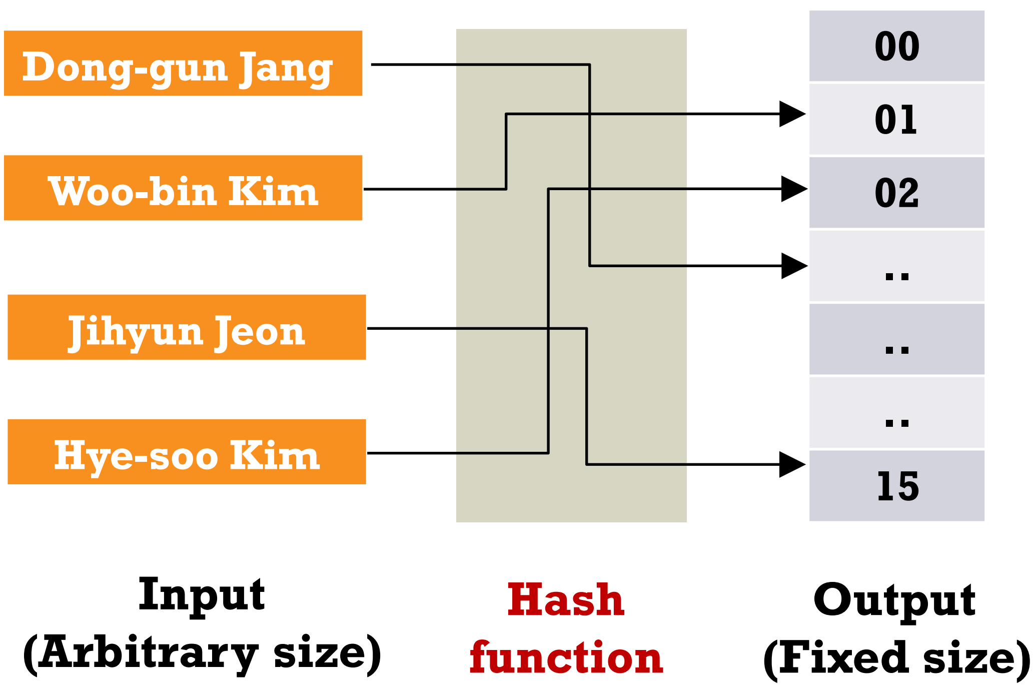
- Is there any good structure for searching an element?
  - Sorted list
  - Binary search tree
  - AVL tree
  - …

- Is it possible to find an element with O(1)?
  - O(1) means that we know the position of what we want to find
    - Contradiction!!
  - Or, if we can predict, with very high probability, where the element will be, it may be possible.
  - But … how?

# What is Hashing?

- Definition
  - A **hash function** is used to **map** a value of **arbitrary** size to a hash value of **fixed** size.

- Example: **Dictionary**
  - It consists of a pair of **keys** and **values**.




# What is Hashing?

- Example: 112, 23, 64, 205, 99, 47, 8, 31
  - Hash function: $h(x) = x \ mod \ 10$
  - why 10 buckets?
    - $h(x)$ produces 10 numbers from 0 to 9 **uniformly**.
  - Put a number into the bucket whose index is $h(x)$
    - 112: bucket[2]
    - 23: bucket[3]
    - 64: bucket[4]
    - 205: bucket[5]
    - …
    - 31: bucket[1]

**Hash Table**

| Key | Values |
|---|---|
| 0 | |
| 1 | 31 |
| 2 | 112 |
| 3 | 23 |
| 4 | 64 |
| 5 | 205 |
| 6 | |
| 7 | 47 |
| 8 | 8 |
| 9 | 99 |

# What is Hashing?

- Example: 112, 23, 64, 205, 99, 47, 8, 31
  - Hash function: $h(x) = x \ mod \ 10$
  - Is there 981?
    - No!
    - $h(981) = 1$, so 981 is expected to be at bucket[1], but it does not exist.
  - Is there 99?
    - Yes!
    - $h(99) = 9$, so 99 is expected to be at bucket[9], and it is there!
  - Is there $x$?
    - **Evaluate $\boldsymbol{h(x)}$:** $\boldsymbol{O}(1)$
    - **Look at bucket[$\boldsymbol{h(x)}$]:** $\boldsymbol{O}(1)$

**Hash Table**

| Key | Values |
|---|---|
| 0 | |
| 1 | 31 |
| 2 | 112 |
| 3 | 23 |
| 4 | 64 |
| 5 | 205 |
| 6 | |
| 7 | 47 |
| 8 | 8 |
| 9 | 99 |

# Good Hash Functions

- Which one is better?

- The hash function should scatter the elements well enough to be uniformly distributed.
  - The hash function should be **unbiased**.
  - Should use **ALL** bits in an element.

# Good Hash Functions

- Universal hash function
  - If we have $b$ buckets, the probability that $h(x) = i$ is $1/b$ for all buckets.
  - An element $x$ can be put into any bucket $i$ with $1/b$ probability.

$$\boxed{\Pr_{i,j \in D, i \neq j}(h(i) = h(j)) \leq \frac{1}{b}}$$

- Use Mid-square, Division, and Folding.

# Building Hash Functions

- Mid-square (middle of square)
  - $h(x)$: take $\boldsymbol{r}$ bits from the middle of $x * x$
    - $2^r$ buckets are generated.

- Example: 8 612 13 15 235
  - $h(x)$ will take 3 bits from 7th bits of $x * x$

| $x$ | $x * x$ | **Binary number of $x * x$** | **7, 8, 9 bits** | $h(x)$ |
|---|---|---|---|---|
| **8** | 64 | 0000000000**001**000000 | 001 | **1** |
| **612** | 374544 | 1011011011**100**010000 | 100 | **4** |
| **13** | 169 | 0000000000**010**101001 | 010 | **2** |
| **15** | 225 | 0000000000**011**100001 | 011 | **3** |
| **235** | 55225 | 0001101011**110**111001 | 110 | **6** |

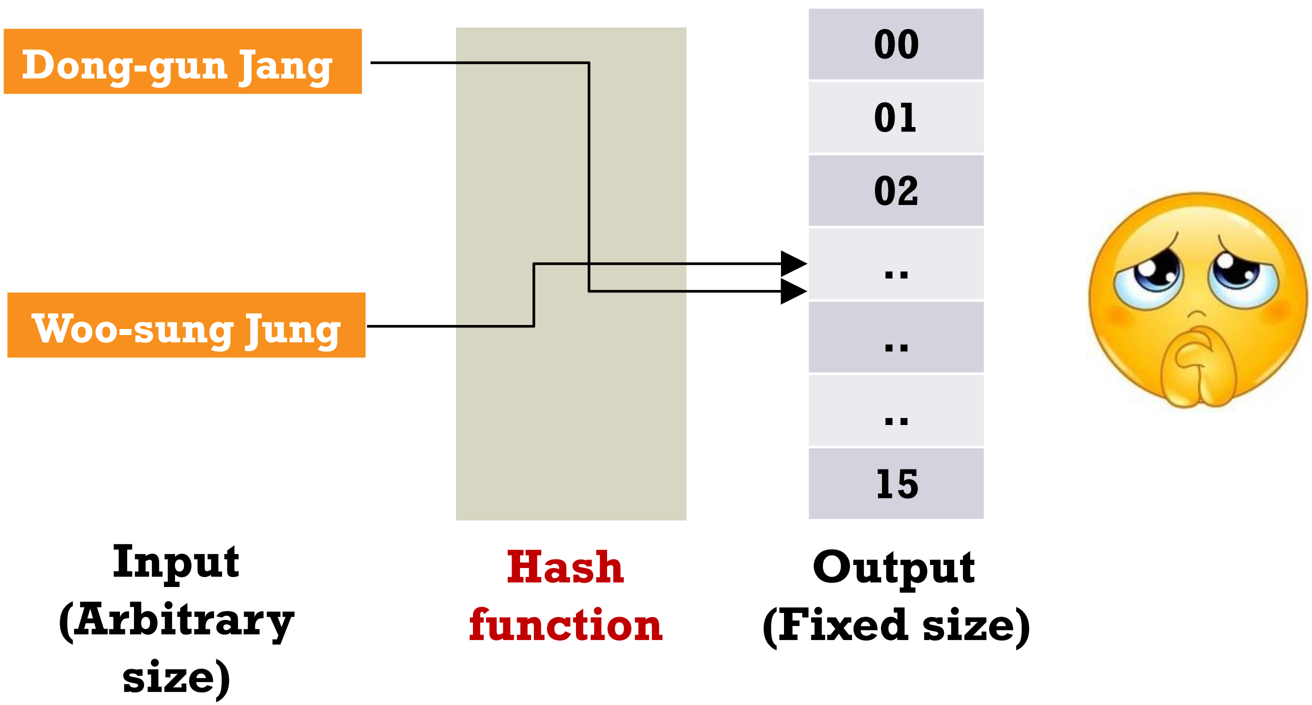
# Building Hash Functions

- Division (modular)
  - $h(x) = x \ mod \ M$, where $M$ is table size
    - Range of bucket address: $0 \sim M - 1$
  - The choice of $M$ is critical
    - Choose $M$ as a **prime number** such that it is not too close to exact powers of 2.

- Folding: shift folding
  - The given key is partitioned into subparts $k_1, k_2, \ldots, k_n$ each of which has the same length as the required address.
  - Example: $h(94034) = h(94 + 03 + 4) = h(101)$

# Collision in Hash Functions

- Collision
  - Given a hash function $h(x)$, two different keys $i$ and $j$ may have the same value $h(i) = h(j)$.




- Solutions: open addressing, closed addressing

# Open Addressing: Linear Probing

- Basic idea
  - If the bucket is **full**, move to the **next** bucket until finding an **empty** bucket.

- Example: 12 32
  - Hash function: $h(x) = x \ mod \ 10$
  - https://visualgo.net/en/hashtable

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | 12 | | | | | | | |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | 12 | 32 | | | | | | |

$$h(x) \quad \rightarrow \quad h(x)+1 \quad \rightarrow \quad h(x)+2 \quad \rightarrow \quad h(x)+3 \quad \rightarrow \quad \dots$$

- It may incur clustering and coalescing.
  - **Clustering**: elements are grouped.
  - **Coalescing**: two adjacent groups are merged.

# Open Addressing: Linear Probing

- Example: 25 3 1 31 11 62 51
  - Hash function: $h(x) = x \ mod\ 10$

- How to insert each element
  - 25: $h(25) = 5$, it is inserted into bucket[5].
  - 3: $h(3) = 3$, it is inserted into bucket[3].
  - 1: $h(1) = 1$, it is inserted into bucket[1].
  - 31: $h(31) = 1$, bucket[1] has been already occupied.
    - Find the next available bucket.
    - It's bucket[2].

| Key | | | | |
|---|---|---|---|---|
| 1 | | | 1 | 1 |
| 2 | | | | 31 |
| 3 | | 3 | 3 | 3 |
| 4 | | | | |
| 5 | 25 | 25 | 25 | 25 |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |

# Open Addressing: Linear Probing

- Searching: How many buckets are searched?
  - Is there 22?   6 buckets
  - Is there 51?   7 buckets
  - Hmm??
    - Buckets are **linearly** searched.
    - Elements may **not be scattered**.
    - Elements are **easily merged**.

| Key | |
|---|---|
| 1 | 1 |
| 2 | 31 |
| 3 | 3 |
| 4 | 11 |
| 5 | 25 |
| 6 | 62 |
| 7 | 51 |
| 8 | |
| 9 | |

- How to resolve this problem?
  - Quadratic probing
  - Random probing
  - Rehashing

# Open Addressing: Quadratic Probing

- Quadratic probing
  - Searching the buckets in the order of $h(x) + inc * inc \ mod \ M$

- Example: 12 32 62 42
  - Hash function: $h(x) = x \ mod \ 10$

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | 12 | 32 | | | 62 | | | |

➡

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 42 | 12 | 32 | | | 62 | | | |

$$h(x) \rightarrow h(x)+1 \rightarrow h(x)+4 \rightarrow h(x)+9 \rightarrow h(x)+16 \rightarrow \ldots$$

- Better to avoid the clustering and coalescing problem.

# Open Addressing: Double Hashing

- Double hashing
  - Use another hashing function in cases when collisions occur.
  - It is also called **rehashing**.

- How double hashing differs from quadratic hashing?
  - For collisions, use another function.
    - Step: $inc * inc \rightarrow inc * h_2(x)$.
  - Searching the buckets in the order of $h_1(x) + inc * h_2(x)\ mod\ M$

$$h(x) \rightarrow h(x)+1 \rightarrow h(x)+4 \rightarrow h(x)+9 \rightarrow h(x)+16 \rightarrow \ldots$$

$$\Downarrow$$

$$h_1(x) \rightarrow h_1(x) + \boldsymbol{h_2(x)} \rightarrow h_1(x) + \boldsymbol{2h_2(x)} \rightarrow h_1(x) + \boldsymbol{3h_3(x)} \rightarrow h(x) + \boldsymbol{4h_2(x)} \rightarrow \ldots$$

# Open Addressing: Double Hashing

- Example: 12  32  62  42
  - Hash function: $h_1(x) = x \ mod \ 10$, $h_2(x) = 7 - (x \ mod \ 7)$

- How to insert each element
  - 32: $h_1(32) = 2$, bucket[2] has been already occupied.
    - $h_1(32) + h_2(32) \ mod \ 10 = 5.$
  - 62: $h(62) = 2$, bucket[2] has been already occupied.
    - $h_1(62) + h_2(62) \ mod \ 10 = 3.$
  - 42: $h(42) = 2$, bucket[2] has been already occupied.
    - $h_1(42) + h_2(42) \ mod \ 10 = 9.$

| Key | | | | |
|---|---|---|---|---|
| 1 | | | | |
| 2 | 12 | 12 | 12 | 12 |
| 3 | | | 62 | 62 |
| 4 | | | | |
| 5 | | 32 | 32 | 32 |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | 42 |

# Closed Addressing: Chaining

- Chaining
  - Use a **linked list** of colliding elements for each bucket.

- Example: 11 22 8 3 31 51
  - Hash function: $h(x) = x\ mod\ 10$