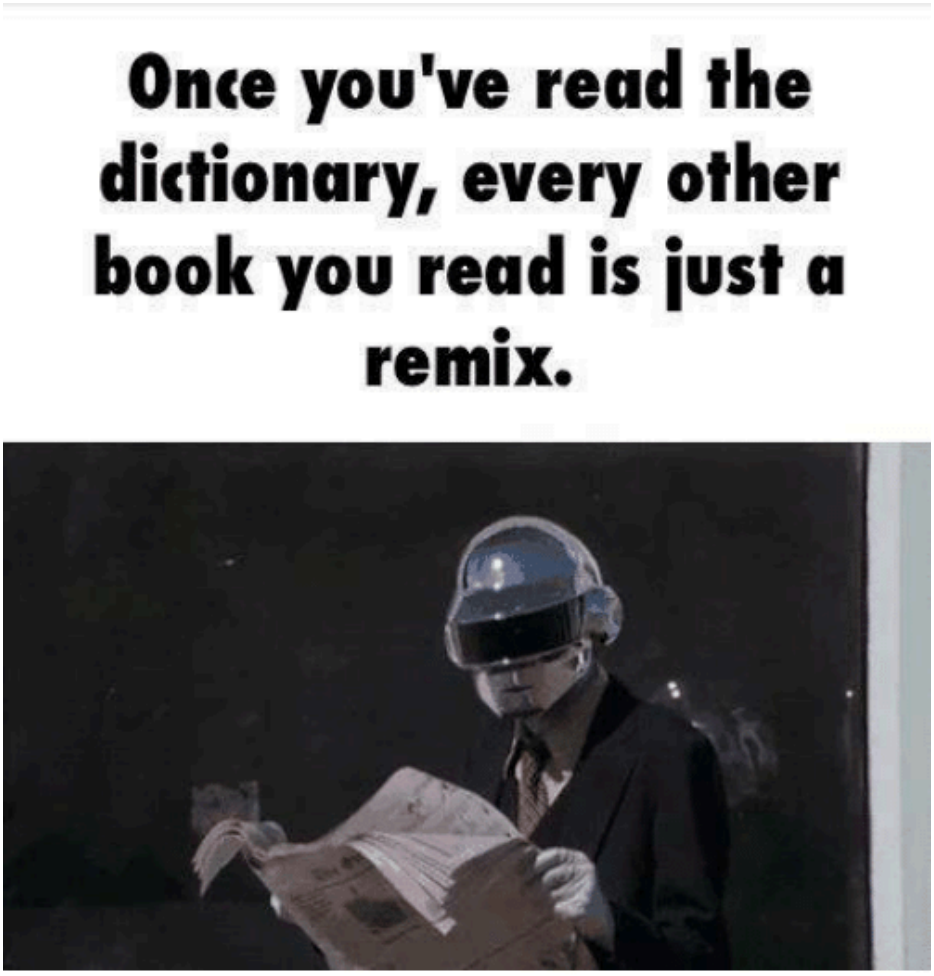
# 10.0 - Week 10 - Workshop (MA)

## Learning Objectives

- Dictionary ADT and Hash Tables
- Collision Resolution

Week 10 Padlet Discussion Board link: https://monashmalaysia.padlet.org/fermi/2022week10



# What is Dictionary?

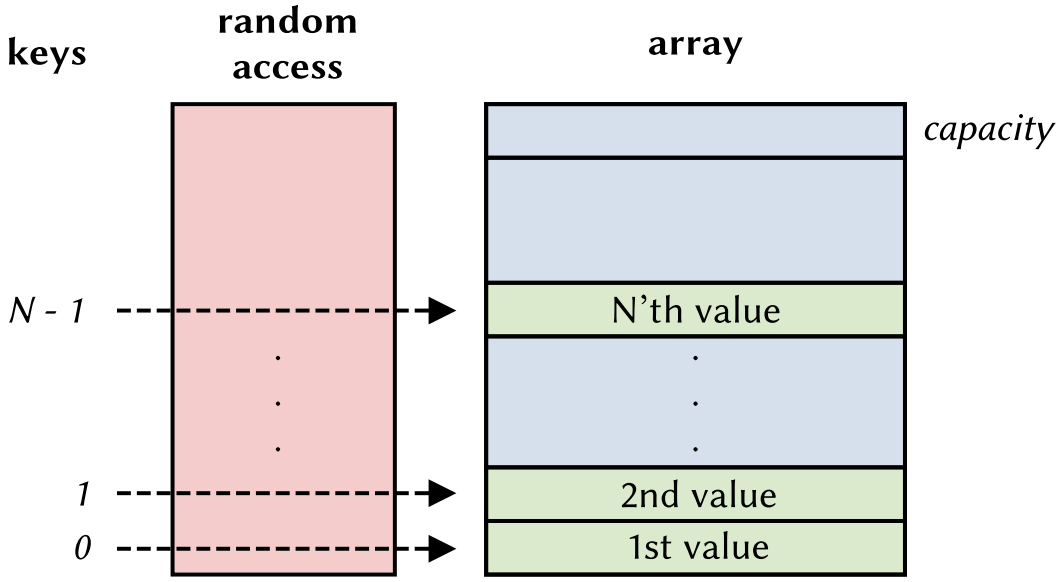
**Question** *Submitted Oct 3rd 2022 at 10:24:06 am*

What is referred to as ***Dictionary ADT*** in computer science?

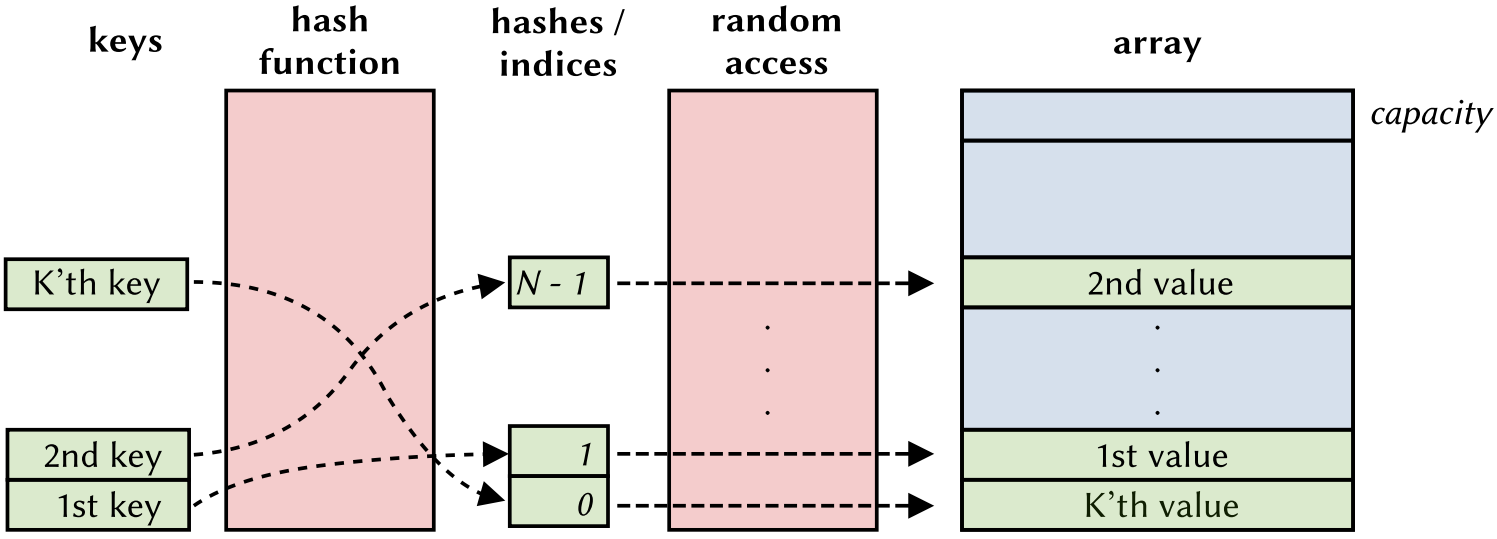
- ( ) Not sure but it must be *that thing* from the previous lesson...
- ( ) It is a listing of words in one or more specific languages, often arranged *alphabetically*.
- (•) It is a *collection* of `(key, value)` pairs, such that each possible key appears at most once in the collection, with efficient *insertion*, *deletion*, *modification* and *lookup* operations for the keys.

Dictionary ADT

## Integer keys - arrays suffice!




## What if keys aren't integers? Hash tables:



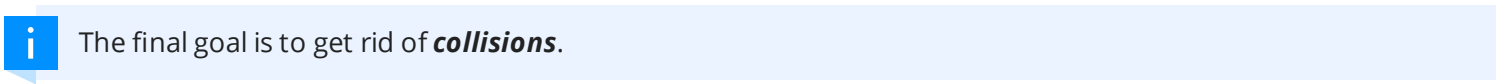
## Hash Functions

Let $K$ be the set of *possible keys* and $\mathcal{T}$ be the *table*. A hash function $h$ is a *mapping*:

$$h: \quad K \to \{0, \ldots, |\mathcal{T}| - 1\}$$

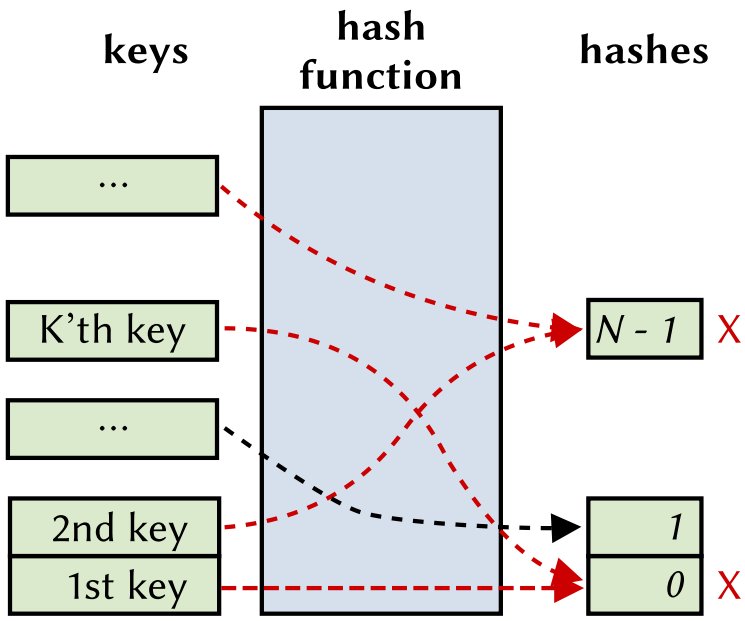
**Some properties:**

- efficiency
- type-dependency
- uniformity



The final goal is to get rid of ***collisions***.

## But the world is not perfect:



## Trying out Hash Functions

Now, we're going to try implementing two hash functions for strings, and see how they distribute the hashes of different keys.

- `simple_hash()`: Takes the numeric value of the first character of the string.
- `better_hash()`: A hash function taking into account all characters of the string.

When we're done, we'll consider a table of size `17` and will compute hash values for `1700` *randomly selected* words.
Ideally, a hash function should *map* $\approx 100 = \frac{1700}{17}$ words into *every* hash value.

## Collisions happen...


⚠ Even if we are careful with the choice of our hash function and the (**large enough**) size of the table, ***collisions may happen***!

## How do we deal with them?

**Question** *Submitted Oct 3rd 2022 at 10:24:26 am*

What should we do when a collision happens?

- ( ) Nothing! In the end, it will still be fine, right?..
- ( ) My hash functions are **perfect**!
- (•) We need to apply one of the *collision resolution* mechanisms.
- ( ) Simple! Just increase the table size and rehash!

## Linear Probing: Adding Things

**The idea is simple:** if a slot for a given hash value is *occupied*, find the ***next free*** slot.

Assume we have the following collisions:

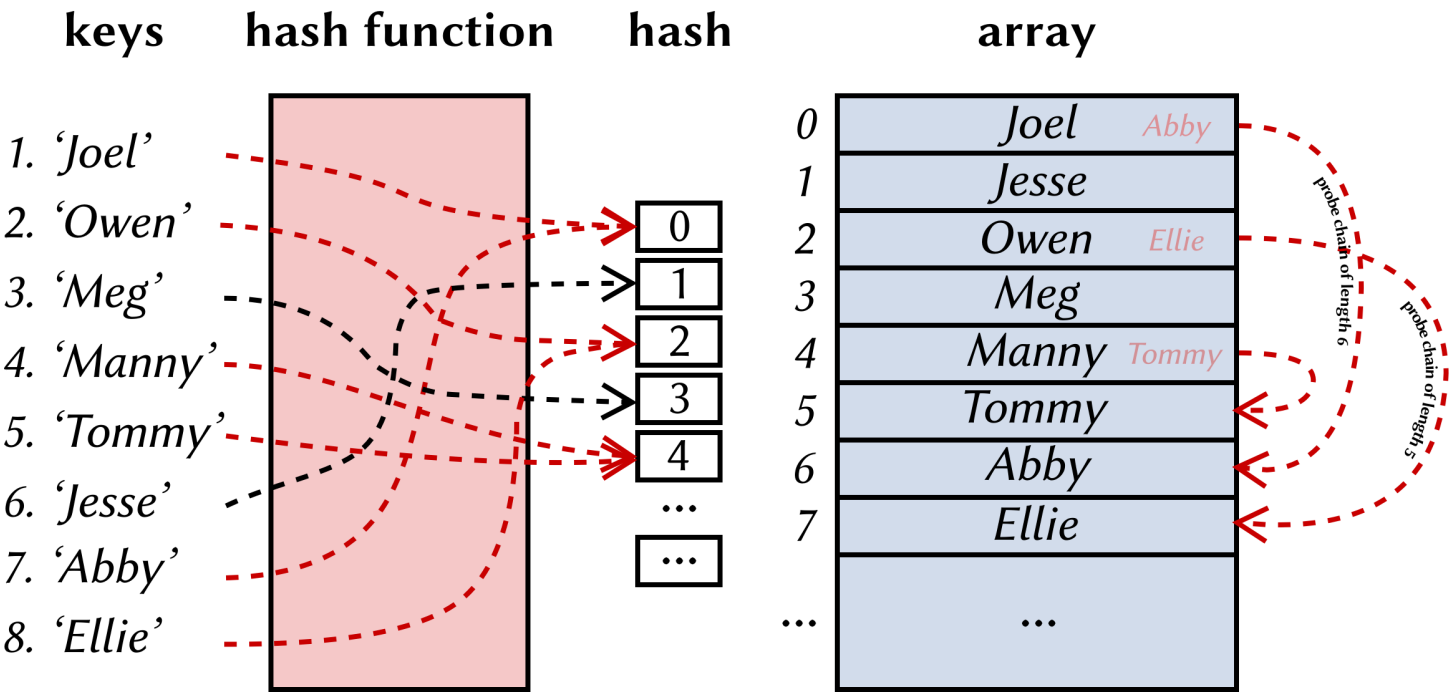
$$h(\text{'Joel'}) = h(\text{'Abby'}) = 0$$

$$h(\text{'Jesse'}) = 1$$

$$h(\text{'Owen'}) = h(\text{'Ellie'}) = 2$$

$$h(\text{'Meg'}) = 3$$

$$h(\text{'Manny'}) = h(\text{'Tommy'}) = 4$$




Let's implement `__setitem__()` ***for inserting*** keys in the hash table. Also, try to analyze its complexity.

## Linear Probing: Finding Things

To **find** something, we just have to start at the computed hash, and look forward until we find the key, or reach an empty slot.

Assume we have the following collisions:

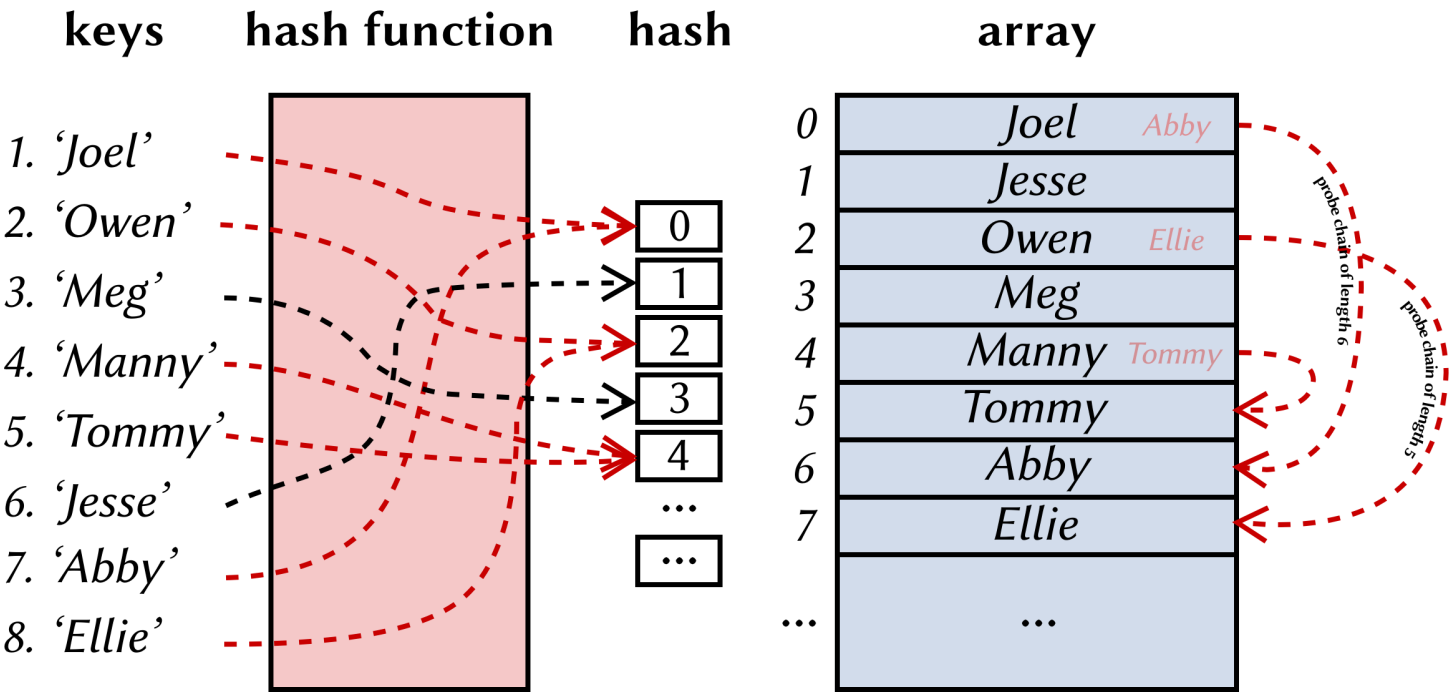
$$h(\text{'Joel'}) = h(\text{'Abby'}) = 0$$

$$h(\text{'Jesse'}) = 1$$

$$h(\text{'Owen'}) = h(\text{'Ellie'}) = 2$$

$$h(\text{'Meg'}) = 3$$

$$h(\text{'Manny'}) = h(\text{'Tommy'}) = 4$$



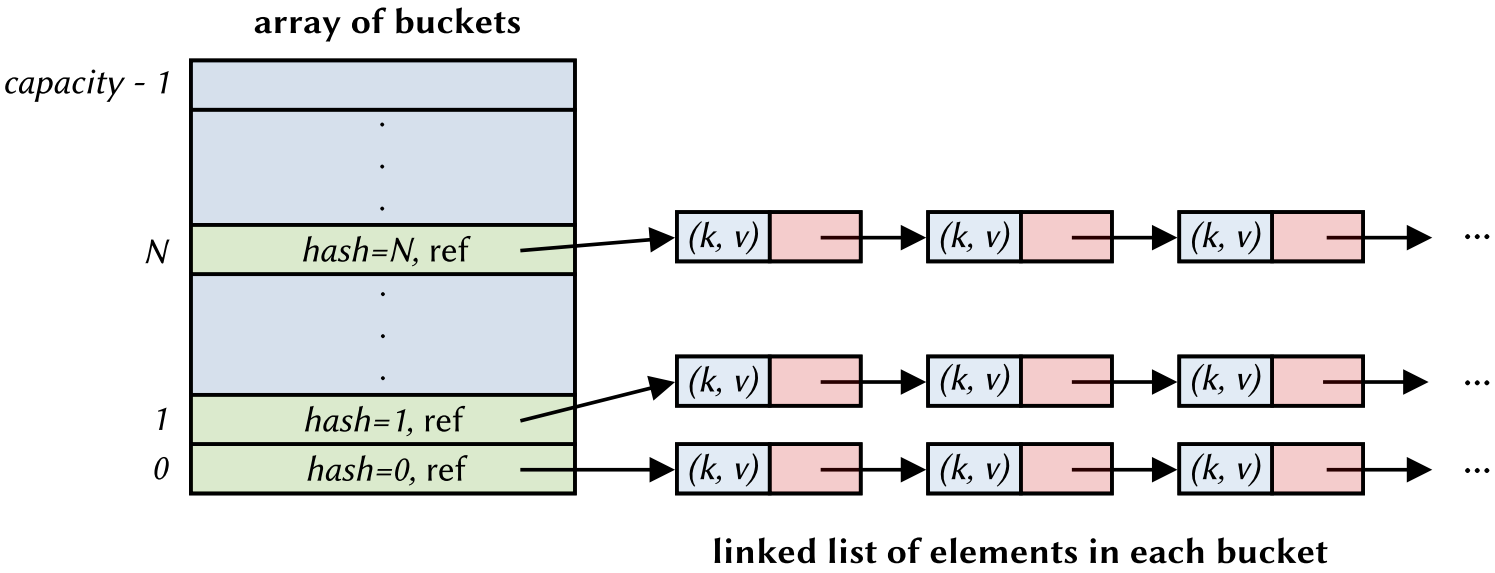

Now let's implement `__getitem__()` for **searching for** keys in the hash table. Also, try to analyze its complexity.

# Dealing with Clustering

Linear Probing is susceptible to ***clustering*** - see the previous slides.

**Solutions are:**

1. *increase* table size + `rehash()` + hope for the best!
2. consider alternative forms of hashing, e.g. *separate chaining:*




***Separate chaining*** is out of scope of this unit but you can read about it if interested.

Feedback Form

# Weekly Workshop Feedback Form

**Question 1**

I am enrolled in:

- FIT1008
- FIT2085
- FIT1054

**Question 2**

What needs improvement?

*No response*

**Question 3**

What worked best?

*No response*

**Question 4**

How engaged were you by the workshop?

- □□□ Very engaged
- □□□ Engaged
- □😐□ Not impressed
- 😕😴□ Lost