Hash Sets and Hash Maps

Presented by Alexander Mak

Hash Set Analogy

Let’s say we have a bag of marbles. Our goal is to keep every marble in the bag a unique color.

For example, if we are given a red marble to put in the bag, we would first check to see whether there is a red marble already in the bag. If there is no red marble yet, we can place our new red marble into the bag. However, if there is already a red marble, we cannot add another red marble unless we remove the original red marble first (otherwise we can just do nothing).

Hash Sets Usage

- Hash set (or just “set”) - a set of unique values
  - You can think of this like an array that cannot have duplicates or indices
- Common useful methods
  - Create a new set
    - new_set = set()
  - Check the size of the set
    - len(new_set)
  - Add to the set
    - new_set.add(value)
  - Remove of the set
    - new_set.remove(value) - will error if value does not exist
    - new_set.discard(value) - will not error out if value does not exist
  - Check if value is in set
    - value in new_set => boolean

Hash Map Analogy

Let's say we have a menu from a restaurant.

You can imagine the menu itself as the hash map. Each dish on the menu is a value associated with a unique key on the menu (item number or name). When you order a dish, you provide the item number or name (which is the key), and the restaurant will retrieve the corresponding dish (the value).

SPECIALTY SUSHI

[illegible]

Party Roll [illegible]
[illegible]

Eel Roll [illegible]
[illegible]

Dragon Roll [illegible]
[illegible]

Fantasy Roll (Spicy) [illegible]
[illegible]

Rainbow Roll 11
[illegible]

New York Roll [illegible]
[illegible]

Boston Roll 11
[illegible]

Tuna Lover's Roll [illegible]
[illegible]

Soft Shell Crab Roll 11
[illegible]

Midori Roll [illegible]
[illegible]

Crunchy Munchy Roll 11
[illegible]

Surf and Turf Roll [illegible]
[illegible]

Kentucky Roll [illegible]
[illegible]

Crab Rangoon Roll [illegible]
[illegible]

Fried Scallop Roll [illegible]
[illegible]

Tempura Rainbow Roll [illegible]
[illegible]

Volcano Roll [illegible]
[illegible]

Chicago Roll 11
[illegible]

Crunchy Texas Roll [illegible]
[illegible]

Yum Yum Roll [illegible]
[illegible]

Tuna Stack [illegible]
[illegible]

Poke Bowl [illegible]
[illegible]

Teriyaki Chicken Bowl [illegible]
[illegible]

Hash Sets Summary

- Hash sets are a collection of unique values
- Do not have indices
- Useful for when we want to check for uniqueness or check if we have “seen” something before that should be unique

Operations	Big-O Time
Search for element	O(1)
Insert element	O(1)
Remove element	O(1)

Hash Maps Usage

- Hash maps are probably the most common data structure you'll encounter after arrays, and it's one of if not the most useful data structure (both for interviews and in practical use).
  - It is so useful, that you can ask yourself whether a hash map would be helpful for every problem.
- In Python, we use "dictionaries" as hash maps.
- Hash map- a set of key-value pairs
- Common useful methods
  - Create new map
    - new_map = { }
  - Set new value to a key
    - new_map[key] = value
  - Check value of a key
    - new_map[key] => returns value
  - Check if key exists
    - key in new_map => boolean
  - Return list of keys
    - new_map.keys()
  - Return list of values
    - new_map.values()
  - Return list of tuples that contain the key and value
    - new_map.items()
- Python Default Dictionary
  - from collections import defaultdict
  - defaultdict_demo = defaultdict(int)
  - Other values for parameter
    - set
    - list

Hash Maps Summary

- Hash maps are a collection of key-value pairs
- Useful for when we need to keep track of counts or frequencies.
- Extremely useful, but they take significantly more memory than an array does.

Operations	Big-O Time
Search for key or value	O(1)
Insert key or value	O(1)
Remove key or value	O(1)

Hash Map Implementation

- It is very unlikely that you will be asked to implement a hash map from scratch. However, understanding how a hash map works can be useful for you.
- Hash maps are most commonly implemented using arrays under the hood.
- Suppose that we want to fill up our array with the follow key-value pairs.
  - hashmap["Alice"] = "NYC"
  - hashmap["Brad"] = "Chicago"
  - hashmap["Collin"] = "Seattle"
- Using a hashing function, convert key into an integer, then use that integer as the index in our array. (For simplicity's sake, let sum the ASCII codes of the letters in the strings)
- The sum of ASCII codes for "Alice" is 25. We take this value and mod it by the array size
  - So if our array size is 2, we get 25 % 2 = 1

Hash Map Implementation - Collisions

- It is possible that another key would return the same result from the hashing function.
  - For example, "Brad" would give 21 % 2 = 1 with our hashing function.
- When two different keys attempt to place a value in the same place in the array, that is called a collision.
- Resize the array to minimize collisions
  - Resize the array whenever we fill half the array to minimize collision
    - Once we resize, we have to re-hash and possibly re-assign values
  - Choose a prime number sized hash map.
    - Mathematically, it makes sense to choose the hashmap to be of a prime size.
    - This is because the prime number is only divisible by 1 and itself!
    - This post from CS StackExchange provides a mathematical proof if you are interested.
- Collisions, however, are inevitable. Here is how we can handle them:
  - Chaining
    - We can "chain" linked list nodes so that multiple values can be written at the same index
    - This raises searching, inserting, and deleting to O(n)
  - Open Addressing
    - Simply write to the next available slot so that we do not store more than one value per index.
    - This is more efficient with a small number of collisions, but it is limited to the size of the array.

Questions?

Let’s practice!

https://github.com/Dijkstra-LLC/dsa_live_pro/tree/main/W01D02/classwork