Hash Maps: Introduction

Let's understand the Hash Maps pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following

- Overview
- Examples
- Does my problem match this pattern?
- Real-world problems
- Strategy time!

Overview

The **hash map** pattern is a method to store data. It aims to reduce the time taken to find and access values.

Hash maps store data in the form of key-value pairs. They are similar to arrays because array values are stored against numeric keys. These keys are known as indexes. We don't get to pick the value of these keys as they are always sequential integers starting from 0. Therefore, if we want to find an element within an array and don't know it's index, we'll have to search the entire array which, in the worst case, will take $O(N)$ time.

On the contrary, hash maps allow us to have flexible keys. Each key is unique and is mapped against a value. Therefore, we can look up its value in $O(1)$ time.

A practical application of using hash maps over arrays would be to look up the marks of students. Had we used arrays, we would have to create the

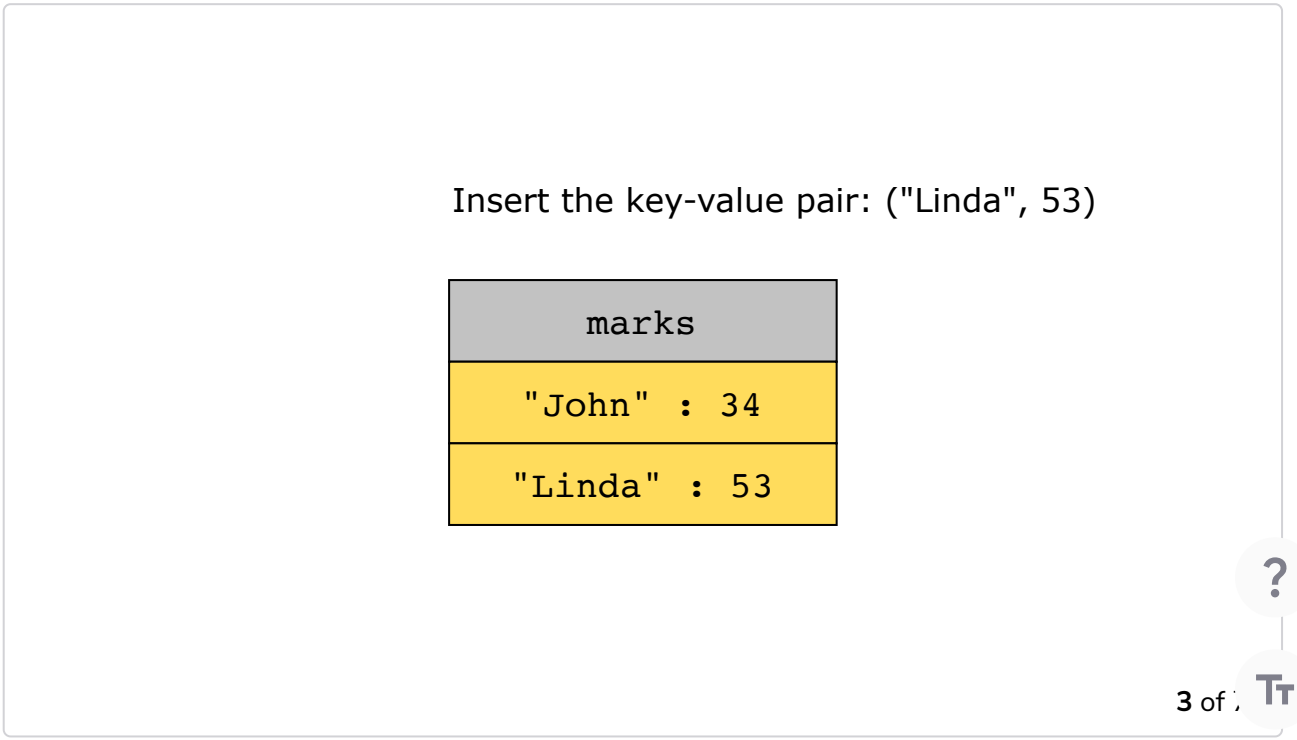
following two arrays:

- `names`: This stores the names of the students.
- `marks`: This stores the marks of the students.

We would have to make sure that the `names` and `marks` arrays store their values in the same order. This means if "John" is at index `4` in the `names` array, his marks should also be present at the same index in the `marks` array. This process is very tedious since we first have to make a lookup in the `names` array in $O(N)$ time and then, using the corresponding index, perform another lookup in the `marks` array in $O(1)$ time.

Using a hash map makes our life much easier. We can use the names of the students as keys, and their marks are the corresponding values. Now we only have to perform one lookup for the marks of a student in $O(1)$ time.

The following illustration shows how values are inserted and accessed from a hash map in the above example:

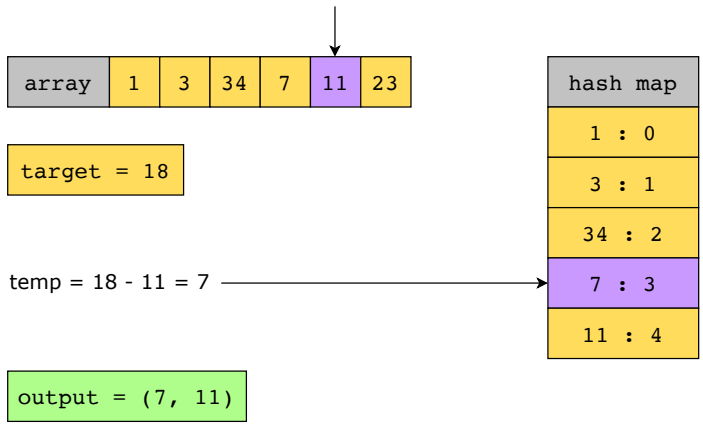


Examples

The following examples illustrate some problems that can be solved with this approach:

Check for a pair in an array with a target sum (Two Sum)

Traverse the array and at each element, calculate the difference between the target value and the current value, and assign it to a variable called temp. For example, if we're at value 11 in the array, and our target is 18, temp will contain 18 - 11 = 7. Check whether temp is in the hash map. If it is, we return temp and the element we're currently on as we've gotten our answer. If it is not, we add the element to the hash map, with the value of the element as the key and its index in the array as the value.



1 of 2

Does my problem match this pattern?

- Yes, if both these conditions are fulfilled:
  - When we require repeated fast access to data during the execution of the algorithm.
  - We need to store the relationship between two sets of data in order to compute the required result. This is achieved through the mechanism of a key-value pair, where we stored the hashed version of the key to enable fast look-ups.
- No, if no useful relation can be established between two sets of data.

Real-world problems

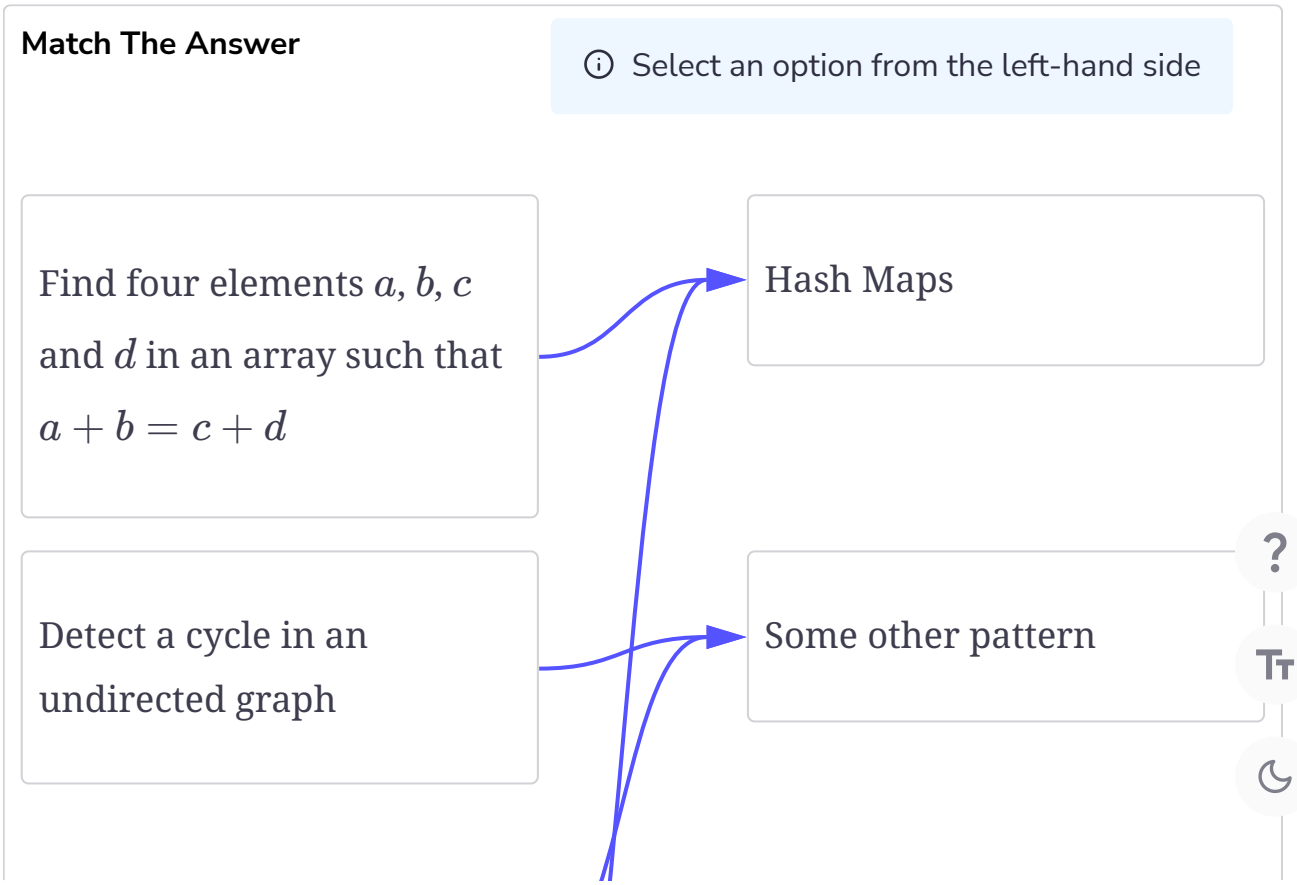
Many problems in the real world share the hash maps pattern. Let's look at some examples.

- **Telecommunications:** Implement a phone book with the name of the person as the key, and their number as the corresponding value.
- **E-commerce:** Search for details of a product using its product ID as the key.
- **File system:** When a user interacts with a file system, they see the file name and the path. The system uses a hash map to store the correspondence between the file name and its path.

Strategy time!

Match the problems that can be solved using the hash map pattern

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.



Rotate a linked list

Divide an array into pairs whose sum is divisible by 2

Reset | Show Solution | Submit

Back
Lexicographical Numb...

Next
Design HashMap

Mark as Completed