2822. Inversion of Object

## Easy

# **Problem Description**

The task is to create a function that accepts either an object (often called a map or dictionary in other languages) or an array obj, and returns a new object, inverted0bj. This inverted object should swap the keys and values from the original obj: each original key becomes a value, and each original value becomes a key.

For example, given an object { 'a': '1', 'b': '2' }, the inverted0bj would be { '1': 'a', '2': 'b' }. Handling arrays means considering their indices as keys. For instance, if obj is an array like ['a', 'b'], the resulting inverted0bj

would be { 'a': '0', 'b': '1' } (indices '0' and '1' become values in the inverted object).

A twist in this problem is how to handle duplicate values in the obj. If a value appears multiple times, then in invertedObj, this value becomes a key mapped to an array containing all the original keys that had the value. For instance, if obj is an object like { 'a': '1', 'b': '1' }, then invertedObj would be { '1': ['a', 'b'] }.

The function guarantees that obj will only have strings as values, which simplifies the possible types of values we have to handle

for keys in invertedObj. Intuition

## to invert the key-value pairs, we'll iterate through all the key-value pairs of the input obj.

For each pair, we'll check if the value we're looking at is already a key in the ans (the accumulator or result object). If so, we need to handle it differently depending on whether it's already associated with multiple original keys (which would mean it's already an array) or not.

The intuition behind the solution is fairly straightforward given the constraints and objectives of the problem: since we are looking

1. If the value is not yet a key in the ans, we simply set the value as a key in ans and assign it the current key as its value. 2. If the value already exists as a key and it's associated with an array, we append the current key to this array. 3. If the value already exists as a key but not as an array (meaning this is the second occurrence of this value), we transform the value into an array and add the current key to it.

- **Solution Approach**
- The solution uses a simple iteration approach with a conditional structure to handle the creation of the inverted object. The primary data structure used is a JavaScript object (ans), which is essentially acting like a hash table, allowing us to store keyvalue pairs efficiently.

## store our inverted key-value pairs.

using ans.hasOwnProperty(obj[key]).

We iterate over the obj parameter using a for...in loop. In JavaScript, a for...in loop iterates over the enumerable properties (keys) of an object. Inside the loop, for every key-value pair in obj, we check if the value (which will become a key in ans) already exists in ans

We define a function named invert0bject that takes an object as a parameter and initializes an empty object ans which will

- If it does not exist already, we simply assign the value from obj as the key in ans, and set the original key as the value, like so: ans [obj [key]] = key.
- If it does exist, we need to differentiate between two cases: 0 ■ When the existing value is an array, which means the value has appeared before and we've already converted it into an array. Here we
- push the current key to that array: ans [obj [key]].push(key). ■ When it is not an array, which means this is the second occurrence of that value and it is currently stored as a single string. We transform it into an array containing the previously mapped key and the current key: ans [obj [key]] = [ans [obj [key]], key].

After we have iterated over all key-value pairs in obj, the ans object is fully constructed and now contains all the inverted key-

Finally, the function returns the ans object as the output, giving us the required inverted0bj. The elegance of this solution lies in its simplicity and efficiency. We use a single pass over the input obj, leveraging hash table

operations for quick access and update, and we handle duplicate values seamlessly by converting them to an array only when

required. This approach ensures optimal use of space since we don't prematurely create arrays for values that don't have multiple

value pairs, with single values being kept as strings and duplicated values being stored as arrays.

**Example Walkthrough** Let's walkthrough the solution approach using a small example. Suppose we have the following object:

1. We initiaize our empty object ans that will store the inverted key-value pairs: **let** ans = {};

According to our problem description, we want to invert this object's keys and values but also handle the case where a single

#### Since '1' is not already a key in ans, we set ans [1] = 'a'. Moving to the next key-value pair ('b': '2'):

keys.

 Now we find '2' already as a key in ans. Since it is not associated with an array yet, we convert it into an array including the current and prior keys, resulting in ans[2] = ['b', 'c']. Finally, we have the key-value pair ('d': '3'):

value may correspond to multiple keys. Here is how we would apply the solution steps outlined above:

After iterating through all key-value pairs, our ans object looks like this:

Now, we iterate over the object. Starting with the first key-value pair ('a': '1'):

'2': ['b', 'c'],

Since '2' is not in ans yet, we set ans [2] = 'b'.

Again, '3' is not a key in ans, so we set ans [3] = 'd'.

6. This ans object is our inverted object that we return from the function:

Next, we have the key-value pair ('c': '2'):

let obj = { 'a': '1', 'b': '2', 'c': '2', 'd': '3' };

- return ans;
- The value '3' corresponds to the key 'd'. The solution effectively handles duplicate values by storing all keys that correspond to a single value in an array, thus maintaining

The value '2' corresponds to both keys 'b' and 'c'.

The value '1' corresponds to the key 'a' in the original object.

the integrity of the original object in the inverted output.

The output indicates that:

Solution Implementation

def invert\_object(source\_object):

inverted\_object = {}

**Python** 

# Iterate over each key-value pair in the source object

# the existing and new keys

# which is the original object's key

inverted\_object[value] = key

for key, value in source\_object.items():

if value in inverted\_object:

# Initialize a dictionary to store the inverted key-value pairs

# Check if the value already exists as a key in the inverted object

inverted\_object[value] = [inverted\_object[value], key]

# If the inverted key does not exist, add it with its new value

# Return the inverted dictionary after all key-value pairs have been processed

# If the inverted key (which is the original object's value) already has a list,

# append the new key (original object's key) to that list if isinstance(inverted\_object[value], list): inverted\_object[value].append(key) else: # If there is a single value, convert it into a list containing

### Java import java.util.HashMap; import java.util.Map;

import java.util.ArrayList;

} else {

C++

#include <unordered\_map>

#include <vector>

#include <string>

#include <typeinfo>

invertedMap.put(value, key);

// This function takes a map and inverts its keys and values.

// Iterate over each key-value pair in the source map.

for (const auto& kvp : sourceMap) {

const std::string& key = kvp.first;

auto it = invertedMap.find(value);

it->second.push\_back(key);

if (it != invertedMap.end()) {

const std::string& value = kvp.second;

// Initialize a map to store the inverted key-value pairs.

std::unordered\_map<std::string, std::vector<std::string>> invertedMap;

// Check if the value already exists as a key in the inverted map.

// If the inverted key (which is the original map's value) is found,

// we add the new key (original map's key) to the existing vector.

// Return the resulting map after all keys have been inverted.

import java.util.List;

return inverted\_object

else:

```
public class ObjectInverter {
    /**
    * Inverts the keys and values of the given map. If the same value is encountered more than once,
    * the corresponding keys are grouped in a list.
    * @param sourceMap The map to invert.
    * @return A map with inverted keys and values.
    public static Map<Object, Object> invertObject(Map<Object, Object> sourceMap) {
       // Initialize a map to store the inverted key-value pairs.
       Map<Object, Object> invertedMap = new HashMap<>();
       // Iterate over each entry in the source map.
        for (Map.Entry<Object, Object> entry : sourceMap.entrySet()) {
            Object key = entry.getKey();
            Object value = entry.getValue();
           // Check if the value already exists as a key in the inverted map.
           if (invertedMap.containsKey(value)) {
               // Retrieve the existing entry for the current value.
                Object existingEntry = invertedMap.get(value);
               // If the corresponding inverted value is already a List,
                // we add the new key into that List.
                if (existingEntry instanceof List) {
                    ((List) existingEntry).add(key);
                } else {
                   // Otherwise, we create a List to combine the existing and new keys,
                   // then put it as the new value for the current inverted key.
                    List<Object> keysList = new ArrayList<>();
                    keysList.add(existingEntry);
                    keysList.add(key);
                    invertedMap.put(value, keysList);
```

// If the inverted key does not exist, simply add it with its value (original map's key).

```
return invertedMap;
// Optional: main method for testing the invertObject function.
public static void main(String[] args) {
   Map<Object, Object> sourceMap = new HashMap<>();
   sourceMap.put("a", 1);
    sourceMap.put("b", 2);
   sourceMap.put("c", 1);
   Map<Object, Object> invertedMap = invertObject(sourceMap);
   // This should print "{1=[a, c], 2=b}"
    System.out.println(invertedMap);
```

// If the same value is encountered more than once, the corresponding keys are grouped in a vector.

std::unordered\_map<std::string, std::vector<std::string>> InvertObject(std::unordered\_map<std::string, std::string>& sourceMap) +

```
} else {
           // If the inverted key does not exist, we create a new vector
           // with the original map's key and add it to the inverted map.
            invertedMap[value] = std::vector<std::string>{key};
    // Return the result after all keys and values have been inverted.
    return invertedMap;
// Note: The use of string as the type for keys and values is an assumption.
// If, in practice, keys or values have different types, the appropriate data
// structures and type handling would need to be used.
TypeScript
// This function takes an object and inverts its keys and values.
// If the same value is encountered more than once, the corresponding keys are grouped in an array.
function invertObject(sourceObject: Record<any, any>): Record<any, any> {
```

```
// Otherwise, we convert it into an array containing the existing and new keys.
        invertedObject[value] = [invertedObject[value], key];
} else {
   // If the inverted key does not exist, we simply add it with its value
   // (original object's key).
    invertedObject[value] = key;
```

# Check if the value already exists as a key in the inverted object

# append the new key (original object's key) to that list

// Check if the value already exists as a key in the inverted object.

// we add the new key (original object's key) into that array.

// If the inverted key (which is the original object's value) already has an array,

// Initialize an object to store the inverted key-value pairs.

const invertedObject: Record<any, any> = {};

const value = sourceObject[key];

for (const key in sourceObject) {

} else {

return invertedObject;

def invert\_object(source\_object):

inverted\_object = {}

else:

// Iterate over each key in the source object.

if (invertedObject.hasOwnProperty(value)) {

if (Array.isArray(invertedObject[value])) {

invertedObject[value].push(key);

// Return the result after all keys have been inverted.

# Initialize a dictionary to store the inverted key-value pairs

# Iterate over each key-value pair in the source object

if isinstance(inverted\_object[value], list):

inverted\_object[value].append(key)

for key, value in source\_object.items():

if value in inverted\_object:

Time and Space Complexity

```
# the existing and new keys
            inverted_object[value] = [inverted_object[value], key]
    else:
        # If the inverted key does not exist, add it with its new value
        # which is the original object's key
        inverted_object[value] = key
# Return the inverted dictionary after all key-value pairs have been processed
return inverted_object
```

# If there is a single value, convert it into a list containing

# If the inverted key (which is the original object's value) already has a list,

**Time Complexity** The time complexity of invert0bject is 0(n), where n is the number of properties in the input object. This is because the function

array resizing is infrequent compared to the number of push operations.

iterates through all the properties of the object exactly once.

an array, or creates a new property. The hasownProperty check, access, and assignment of a property in an object are all 0(1) operations on average, assuming the properties are sufficiently distributed in the underlying hash table. However, when multiple keys map to the same value and an array is created, the push operation on the array is also 0(1) on average, assuming dynamic

During iteration, the function checks if the ans object has a property with the current value as its name, adds the current key to

The given TypeScript function inverts a key-value mapping in an object by making the values as keys and the original keys as

values. If the function comes across duplicate values in the input, it stores the keys corresponding to that value in an array.

Hence, the loop which constitutes the main workload of the function performs a constant amount of work for each property, ensuring an overall linear time complexity. **Space Complexity** 

object, but with flipped keys and values. In the worst case, where no values are duplicated, each property from the input will be represented in ans. When values are duplicated and arrays are created, these arrays are stored within the same ans object, not

increasing the order of space complexity but only the constants involved. Furthermore, the space needed for the arrays to accommodate the duplicate keys is included in the O(n) complexity because the

size of the arrays is contingent on the number of keys, which at maximum can be n (when all keys have the same value).

The space complexity of invert0bject is 0(n) because it creates a new object ans that stores all the properties from the original