Array Subset

Learn how to tell if two arrays contain the same values. We'll discuss how to deal with both primitive values and objects and we'll go over how a JavaScript Map helps us with this problem.

We'll cover the following

- Introducing Maps

Array Subset#

Instructions#

Write a function that accepts two parameters, both arrays. The arrays can have both strings and numbers. Return `true` if the second array is a subset of the first.

In other words, determine if every item in the 2nd array is also present somewhere in the 1st array.

Input: Array of Numbers & Strings, Array of Numbers & Strings

Output: Boolean

Examples#

```
arraySubset([2, 1, 3], [1, 2, 3]); // -> true
arraySubset([2, 1, 1, 3], [1, 2, 3]); // -> true
arraySubset([1, 2], [1, 2, 3]); // -> false
arraySubset([1, 2, 3], [1, 2, 2, 3]); // -> false
```

Hints#

- This problem has multiple solutions with different time complexities.
- We'll need to consider how to deal with repeats, such as when an item is present twice.

```
1  function arraySubset(arr, sub) {
2      // Your code here
3  }
```

Solution 1#

```
function arraySubset(arr, sub) {
    if(sub.length > arr.length) {
        return false;
    }

    for(let i = 0; i < sub.length; i++) {
        const arrIndex = arr.indexOf(sub[i]);
        if(arrIndex === -1) {
            return false;
        }

        delete arr[arrIndex];
    }

    return true;
}
```

How it Works#

This function starts with a simple check. The subset can't be larger than the superset, by definition.

We continue by looping through every item in the second array. Our function then finds the index of this item in the first array. If that index is `-1`, indicating that it is not present, we can return `false` immediately.

Otherwise, we delete it from the first array. This ensures that if it is encountered again in the second array, it can't use the same value in the first array.

Once we get through the entire second array we can return `true`.

Time#

We have two inputs, so we'll refer to their lengths as `m` and `n`.

We loop over `n`, indicating `O(n)` already. Inside the loop, there is a call to `indexOf()`. This itself is a loop, as the engine has to go through the array to find the index.

So we're at:

O(m * n),#

which is our final time complexity.

Space#

We use the same amount of space no matter how large the inputs, so it's:

O(1).#

Caveats#

- Our function is not a pure function. We delete items out of the first array. To get around this, we could first make a copy of the first array and work with that instead. It would not change time complexity but would increase space complexity to `O(m)`. Solution 2 works around this problem.

Solution 2#

```
function arraySubset(arr, sub) {
    if(sub.length > arr.length) {
        return false;
    }

    const arrCount = {};

    for(let i = 0; i < arr.length; i++) {
        const item = arr[i];
        if(arrCount[item] !== undefined) {
            arrCount[item]++;
        } else {
            arrCount[item] = 1;
        }
    }

    for(let i = 0; i < sub.length; i++) {
        const currentItem = sub[i];
        if(arrCount[currentItem] === undefined) {
            return false;
        }

        arrCount[currentItem]--;
        if(arrCount[currentItem] === 0) {
            delete arrCount[currentItem];
        }
    }

    return true;
}
```

How it Works#

Time#

Every object manipulation performed here has `O(1)` efficiency so they won't factor into our calculations.

We start by looping over the first array, giving us `O(m)`. Everything in the loop is `O(1)`.

Looping over the second array gives us `O(n)`. Since it's not inside the first loop, we add the two variables, giving us:

O(m + n).#

This is much better than `O(m * n)`.

Space#

In this solution, we end up making a copy of the items in `m`, so our space complexity is `O(m)`.

Extending the Problem#

Our solution works for arrays that contain only numbers and strings. What if we wanted to make this function work for arrays that contain any type of data?

Objects#

Using an object wouldn't work. Object keys can only be strings. Any key we insert into the object would automatically be coerced to a string.

This means that different objects can end up being coerced to the same key string even if they are distinct references. This is due to the idea of objects being stored as references (https://www.educative.io/courses/introduction-to-javascript-first-steps/xV8p1GA6K0r). This would make our counts incorrect.

Introducing Maps#

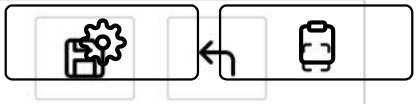
To solve this problem, ES2015 introduced the Map (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map) into JavaScript. A map has much of the functionality of both an object and a set.

It stores key-value pairs like an object. However, unlike an object, it does not stringify its keys. This solves our problem of objects being coerced into strings.

Like a set, a map has methods such as `has` and `set`. It's essentially a hybrid of the two data structures.

When we replace our object with a map, the code looks like this.

```
function arraySubset(arr, sub) {
    if(sub.length > arr.length) {
        return false;
    }

    const arrMap = new Map();

    for(let i = 0; i < arr.length; i++) {
        const item = arr[i];
        if(arrMap.has(item)) {
            arrMap.set(item, arrMap.get(item) + 1);
        } else {
            arrMap.set(item, 1);
        }
    }

    for(let i = 0; i < sub.length; i++) {
        const currentItem = sub[i];
        if(!arrMap.has(currentItem)) {
            return false;
        }

        arrMap.set(currentItem, arrMap.get(currentItem) - 1);
        if(arrMap.get(currentItem) === 0) {
            arrMap.delete(currentItem);
        }
    }

    return true;
}
```



Notice that the last two tests here return `false`. This is correct behavior. While

```
[{}, [], {}]
```

and

```
[{}, [], {}]
```

clearly look similar, each item in these arrays has a distinct reference. The 2nd is not the same as the 1st.

For a better understanding of this idea, try reading this article (https://www.educative.io/courses/step-up-your-js-a-comprehensive-guide-to-intermediate-javascript/7nAZrnYW9rG) of mine from another JavaScript course I've created.

← Back

String Rotation

Next →

Maximum Profits

Mark as Completed

Report an Issue