2755. Deep Merge of Two Objects

objects or arrays that have been parsed from JSON strings.

Medium

Problem Description This problem involves creating a function that can deeply merge two objects or arrays based on specific rules. A deep merge means

that any nested structures within the objects or arrays will also be merged according to certain rules. Here are the rules for deep merging provided in the problem: If both values are objects, the merged result should include all keys from both objects. If a key is present in both, their

Leetcode Link

- associated values should be deep merged. If a key is only present in one object, its value should simply be included. If both values are arrays, the merged array should be as long as the longer of the two original arrays. Just like with objects, apply the same deep merge logic to each position in the arrays, treating the indices as if they were keys.
- In any other case (e.g., values are of different types or not objects/arrays), the result should be the second value obj2. The problem statement clarifies that the inputs obj1 and obj2 are the results of JSON.parse(), meaning they are valid JavaScript

The solution is based on recursion, a common technique for dealing with nested data structures like objects and arrays.

Intuition

not an object, or they are not the same type (one is an array and the other is not), we immediately return obj2. If both are objects or arrays, we iterate through the keys of obj2 and apply the deepMerge function to each key. This means we attempt to merge obj1[key] and obj2[key], recursively ensuring that any nested objects or arrays are also merged correctly.

 We then return obj1 as it now contains merged values from obj2. This recursive strategy neatly handles arbitrarily complex and deep object structures and directly maps onto the merging rules provided in the problem statement. The base case of the recursion is effectively whenever we reach a point where the values to be

We first check if the inputs are both objects or arrays since the merging logic applies only in those cases. If either obj1 or obj2 is

merged are not both objects or arrays.

The solution uses a recursive function deepMerge to merge two potentially complex data structures, which could be objects or arrays. Here's how this is carried out: First, we define helper functions is0bj and isArr to check if a value is an object or array, respectively. These are used to ensure

1 const isObj = (obj: any) => obj && typeof obj === 'object'; 2 const isArr = (obj: any) => Array.isArray(obj);

Solution Approach

 We then examine the input values obj1 and obj2. If either is not an object (or if one is an array and the other isn't), we return obj2. This serves as the base case for non-object/array values or mismatched type pairs.

- 1 if (!isObj(obj1) || !isObj(obj2)) { return obj2; if (isArr(obj1) !== isArr(obj2)) {
- return obj2;

that we only apply merging logic to appropriate data types.

 Next, the function iterates through the keys of obj2 using a for...in loop. For each key, we call deepMerge on the value at that key from both obj1 and obj2, essentially performing a merge operation on any nested structures:

```
1 for (const key in obj2) {
        obj1[key] = deepMerge(obj1[key], obj2[key]);

    The operation obj1[key] = deepMerge(obj1[key], obj2[key]) ensures that if the same key is present in both obj1 and obj2,

 their values will be merged. Otherwise, if the key exists only in obj2, it gets added to obj1.
• Finally, obj1 is returned, now containing merged data from both obj1 and obj2.
```

problem statement.

a: 1,

f: 10,

11 };

e: [5, 6],

Example Walkthrough

b: { c: 3, d: 4 },

To illustrate the solution approach using a small example, let's consider two objects that we wish to merge using the deepMerge function described in the solution approach: 1 let obj1 = {

Recursion is the key algorithmic pattern that makes the implementation succinct and able to cope with objects and arrays of any

depth. The deepMerge function merges obj2 into obj1 at every level of the data structure, complying with the rules laid out in the

7 let obj2 = { b: { c: 8, e: 9 }, e: [7],

Here, we want to merge obj2 into obj1. Here's a step-by-step walkthrough of how the deep merge would function: 1. Since both obj1 and obj2 are objects, we proceed with merging. We do not immediately return obj2 because the base case does not apply.

- Both have a key c, so we merge the values. The value from obj2 (8) takes precedence. obj1[b] has a key d that is not present in obj2[b]; it remains as is. obj2[b] has a key e that is not present in obj1[b]; it gets added to the result. The merged result for key b is: { c: 8, d: 4, e: 9 }.
- We extend obj1[e] to match the length of the longer array, but since obj2[e] is the shorter one, no change is made to the length. The deep merge logic doesn't introduce new items from the longer array unless there are corresponding items in obj2. So

5. For key f, which is only present in obj2, we simply add it to obj1:

2. We iterate over the keys in obj2. The keys are b, e, and f.

3. For key b, obj1[b] and obj2[b] are both objects:

So, we call deepMerge on these two objects:

4. For key e, obj1[e] is an array and so is obj2[e]:

the merged result for key e is [7, 6].

The final merged object obj1 now looks like this:

1 obj1[b]: { c: 3, d: 4 } 2 obj2[b]: { c: 8, e: 9 }

1 obj1[e]: [5, 6]

2 obj2[e]: [7]

1 obj1[f]: 10

a: 1, b: { c: 8, d: 4, e: 9 }, e: [7, 6], f: 10,

This example makes it clear how each step in the solution approach actively contributes to the desired final result according to the

If the types of container for both objects differ (one is a list and one is a dictionary), return second_object.

rules of deep merging. Recursion ensures that objects and arrays nested at any level are merged correctly and the rules are followed.

Python Solution 1 def deep_merge(first_object, second_object):

:param first_object: The first dictionary or object to merge.

:param second_object: The second dictionary or object to merge.

Helper function to check if a value is a dictionary (object).

if not is_object(first_object) or not is_object(second_object):

Recursively merge properties from both objects.

if is_array(first_object) != is_array(second_object):

Iterate over each property of the second_object.

Map<String, Object> map1 = new HashMap<>();

Map<String, Object> map2 = new HashMap<>();

// Should display: {a=2, b=2, c=3}

System.out.println(mergedMap);

Map<String, Object> mergedMap = deepMerge(map1, map2);

map1.put("a", 1);

map1.put("c", 3);

map2.put("a", 2);

map2.put("b", 2);

Function to deeply merge two objects, combining their properties and

:return: A dictionary with properties from both first_object and second_object merged.

If either argument is not a dictionary (object), return the second_object.

If the property key is in the dictionary's own properties (not inherited).

Return the merged first_object, which now contains properties from both objects.

first_object[key] = deep_merge(first_object.get(key), second_object[key])

```
14
       # Helper function to check if a value is a list (array).
15
       def is_array(item):
16
            return isinstance(item, list)
17
18
```

return second_object

return second_object

if key in second_object:

for key in second_object:

return first_object

return isinstance(item, dict)

sub-properties.

def is_object(item):

6

9

10

11

12

13

19

20

21

23

24

25

26

27

29

30

31

32

33

34

36

37

38

39

40

41

42

43

44

45

46

47

48

50

49 }

```
36
37 # Example usage:
38 # obj1 = {"a": 1, "c": 3}
39 # obj2 = {"a": 2, "b": 2}
40 # merged_object = deep_merge(obj1, obj2)
41 # Should print: {"a": 2, "c": 3, "b": 2}
42
Java Solution
   import java.util.HashMap;
 2 import java.util.Map;
   import java.util.Set;
   public class DeepMergeExample {
 6
       // Method to check if an object is a Map (used to simulate an 'object' in Java).
       private static boolean isMap(Object item) {
           return item instanceof Map;
10
11
12
       // Deeply merge two Maps and return the result.
       @SuppressWarnings("unchecked")
13
       public static Map<String, Object> deepMerge(Map<String, Object> firstMap, Map<String, Object> secondMap) {
14
            for (String key : secondMap.keySet()) {
15
               Object firstValue = firstMap.get(key);
16
               Object secondValue = secondMap.get(key);
17
18
19
               // If the current key is present in both maps and both values are also maps, then merge them recursively.
               if (isMap(firstValue) && isMap(secondValue)) {
20
                   Map<String, Object> firstMapValue = (Map<String, Object>) firstValue;
21
                   Map<String, Object> secondMapValue = (Map<String, Object>) secondValue;
22
23
                   firstMap.put(key, deepMerge(firstMapValue, secondMapValue));
               } else {
24
25
                   // If the second map has a key that's not in the first or the values are not maps, put it in the first map.
26
                    firstMap.put(key, secondValue);
27
28
29
30
           // Return the merged Map object.
           return firstMap;
31
32
33
34
       // Example usage:
       public static void main(String[] args) {
35
```

17 18 19 20 }

C++ Solution

2 #include <vector>

#include <any>

#include <string>

#include <unordered_map>

```
// Function to check if a std::any value is an object (i.e., std::unordered_map).
   bool isObject(const std::any &item) {
        if (item.type() == typeid(std::unordered_map<std::string, std::any>)) {
           return true;
10
       return false;
13
   // Function to check if a std::any value is an array (i.e., std::vector).
15 bool isArray(const std::any &item) {
       if (item.type() == typeid(std::vector<std::any>)) {
16
           return true;
       return false;
21
   // Function to deeply merge two objects, combining their properties and sub-properties.
   std::any deepMerge(std::any &firstObject, std::any &secondObject) {
       if (!isObject(firstObject) || !isObject(secondObject)) {
24
25
            return secondObject;
26
27
28
       if (isArray(firstObject) != isArray(secondObject)) {
29
           return secondObject;
30
31
32
       std::unordered_map<std::string, std::any>& firstMap =
33
            std::any_cast<std::unordered_map<std::string, std::any>&>(firstObject);
34
       std::unordered_map<std::string, std::any>& secondMap =
           std::any_cast<std::unordered_map<std::string, std::any>&>(secondObject);
35
36
37
       for (auto &pair : secondMap) {
38
           const std::string &key = pair.first;
39
           if (firstMap.find(key) == firstMap.end()) {
               firstMap[key] = pair.second;
40
           } else {
41
42
                firstMap[key] = deepMerge(firstMap[key], pair.second);
43
44
45
       return firstObject;
46 }
47
   // Example usage:
   // std::unordered_map<std::string, std::any> obj1 = {{"a", 1}, {"c", 3}};
50 // std::unordered_map<std::string, std::any> obj2 = {{"a", 2}, {"b", 2}};
51 // auto result = deepMerge(obj1, obj2);
52 // The resulting 'obj1' will be deep-merged with 'obj2'.
53
Typescript Solution
   // Function to deeply merge two objects, combining their properties and sub-properties.
   function deepMerge(firstObject: any, secondObject: any): any {
       // Helper function to check if a value is an object.
       const isObject = (item: any): boolean => item && typeof item === 'object' && !Array.isArray(item);
       // Helper function to check if a value is an array.
       const isArray = (item: any): boolean => Array.isArray(item);
 8
```

// If either argument is not an object or array, return the secondObject.

// Recursively merge properties from both objects.

// If the property key is from the object's own properties (not inherited).

firstObject[key] = deepMerge(firstObject[key], secondObject[key]);

// If the types of container for both objects differ (one is array and one is object), return secondObject.

if (!isObject(firstObject) || !isObject(secondObject)) {

if (isArray(firstObject) !== isArray(secondObject)) {

// Iterate over each property of the secondObject.

if (secondObject.hasOwnProperty(key)) {

// let obj1 = {"a": 1, "c": 3}, obj2 = {"a": 2, "b": 2};

34 // deepMerge(obj1, obj2); // Should log: {"a": 2, "c": 3, "b": 2}

return secondObject;

return secondObject;

for (const key in secondObject) {

Time and Space Complexity

28 // Return the merged firstObject, which now contains properties from both objects. 29 return firstObject; 30 } 31 32 // Example usage:

9

10

12

13

14

15

16

17

18

19

20

21

24

25

26

27

size of the two input objects, obj1 and obj2. For each key in obj2, it recursively merges the corresponding values. If n is the total number of keys in both obj1 and obj2 at all levels, and assuming the worst case where each value is a nested object requiring a recursive merge, the time complexity would be O(n). It's important to note, though, that each merge itself takes time proportional to

during the merge process.

Time Complexity

the number of keys in the object being merged, which could imply more than just a simple O(n) complexity, but for each merge operation, we consider a constant time assuming that objects/arrays being merged at the same depth are relatively small. Note: The recursion depth also depends on the depth of the object structure, which might impact the time complexity if we account for the cost of function calls; however, this is often disregarded in favor of analyzing the operation's complexity on the data set rather than the call stack.

The time complexity of the deepMerge function is a bit tricky to calculate due to its recursive nature. It depends on the structure and

Space Complexity The space complexity is affected by two factors: the depth of the recursion (call stack) and the creation of new objects/arrays

1. Recursion Depth (Call Stack): If we consider d to be the maximum depth of recursion (which is the deepest level of nested objects/arrays), the space complexity in terms of the call stack would be 0(d).

- 2. Object Creation: Since the function modifies obj1 directly without creating new objects for each merge operation, the space complexity due to object creation would be 0(1) – constant space.
- Taking both points into account, the overall space complexity of deepMerge would be O(d), assuming that we only consider the additional space required by the recursion stack and not the space taken by the input objects, which would remain the same throughout the execution of the function.