2525. Categorize Box According to Criteria

Easy Math

Leetcode Link

Problem Description

The problem presents a scenario where we have a box with given dimensions (`length`, `width`, and `height`) and `mass`. We are required to categorize the box based on its size and weight into one of four possible categories: "Bulky", "Heavy", "Both", or "Neither". A box is considered "Bulky" if any of its dimensions are greater than or equal to 10,000 units or if its volume is greater than or equal to a billion cubic units. It is considered "Heavy" if its mass is greater than or equal to 100 units. The box can also fall into the "Both" category if it meets the criteria for both "Bulky" and "Heavy", or "Neither" if it meets neither criteria. The main task is to assess the box's dimensions and mass and return the corresponding category as a string.

Intuition

The straightforward approach to solving this problem is to simulate the conditions given in the problem statement. The first step is to calculate the volume of the box, which can be done by multiplying `length`, `width`, and `height`. Once we have the volume, we can check if any of the dimensions reach the "Bulky" criteria or if the volume itself makes the box "Bulky".

For the "Heavy" criteria, we simply compare the `mass` against the threshold of `100` units. Combining these two checks, we can determine which of the four categories the box falls into.

The given solution utilizes bitwise operations to efficiently encode the state of the box based on the "Bulky" and "Heavy" criteria. A binary representation is formed by shifting and combining the Boolean results of the "Heavy" check (`heavy`) and the "Bulky" check (`bulky`). This binary number is then used as an index to access the correct category from a predefined list `d`. Hence, the solution is both intuitive and efficient by avoiding multiple `if-else` conditions and leveraging binary state representation.

Solution Approach

The implementation of the solution follows these key steps:

1. Calculate the volume (`v`) of the box by multiplying `length`, `width`, and `height`.
2. Determine if the box is "Bulky" by checking two conditions:
   - If any one of the dimensions (`length`, `width`, or `height`) is greater than or equal to `10000`.
   - Or, if the volume (`v`) is greater than or equal to `10**9`. If any of these conditions are true, set the `bulky` variable to `1`, otherwise to `0`.

   This is achieved using the expression:

   ```
   bulky = int(any(x >= 10000 for x in (length, width, height)) or v >= 10**9)
   ```

   The `any()` function is used here to check if at least one of the dimensions is "Bulky". It iterates through the tuple of dimensions and the volume and returns `True` if any element meets the "Bulky" condition.
3. Determine if the box is "Heavy" by comparing the `mass` to `100`. If `mass` is greater than or equal to `100`, set `heavy` to `1`, otherwise set it to `0`.

   This is achieved using the expression:

   ```
   heavy = int(mass >= 100)
   ```

4. Combine the `heavy` and `bulky` indicators to form a binary representation.
   - Shift the `heavy` indicator to the left by one bit, resulting in `Heavy` occupying the 2's place in a binary number (effectively corresponding to the value of `2` if true).
   - Combine (`|`) `heavy` with `bulky` to form a 2-bit binary number (indices ranging from `0` to `3`).

   This is done using:

   ```
   i = heavy << 1 | bulky
   ```

5. Use the index `i` to return the corresponding category from the predefined list `d` which contains the strings 'Neither', 'Bulky', 'Heavy', and 'Both'.

   The final step in the code is:

   ```
   return d[i]
   ```

The use of a bitwise shift (`<<`) and bitwise OR (`|`) allows for an elegant handling of the different combinations of `heavy` and `bulky`. Instead of using a series of `if-else` statements to check each combination, this solution uses the two binary digits to index directly into the list that contains the appropriate category strings. This is both a space-efficient and time-efficient solution, as there are no complex data structures involved and the runtime is constant, being independent of the size of the input.

Example Walkthrough

Let's consider a box with dimensions `length = 11000`, `width = 8000`, `height = 6000`, and `mass = 150`. We will walk through the solution approach to determine the category of this box.

1. Calculate the volume (v) of the box: (volume = length \times width \times height = 11000 \times 8000 \times 6000 = 528,000,000,000) This volume exceeds the billion (10^9) cubic units threshold.
2. Determine if the box is "Bulky":
   - The `length` is greater than `10000`, so without further checks, we know at least one dimension makes the box "Bulky."
   - Our volume calculation already exceeds `10**9`, confirming the box is indeed "Bulky."

   Since we know the box is "Bulky" based on just the `length` or volume, we would set `bulky` to `1`.
3. Determine if the box is "Heavy":
   - The `mass` of the box is `150`, which is greater than `100`.

   Since the mass exceeds `100`, we set `heavy` to `1`.
4. Combine the `heavy` and `bulky` indicators:
   - `heavy` is set to `1`, and when shifted left by one bit, it becomes `10` in binary, which is `2` in decimal.
   - `bulky` is `1`, and combining (with bitwise OR) it with the shifted `heavy` value, we get `11` in binary, or `3` in decimal.

   So, `i = 2 << 1 | 1 = 3`.
5. Use the index `i` to return the category:
   - With `i` equal to `3`, we select the fourth element (0-indexed) from the list `d` which contains `['Neither', 'Bulky', 'Heavy', 'Both']`.
   - The selected category is `'Both'`, because the box is both "Bulky" and "Heavy".

Thus, for a box with the given dimensions and mass, the category returned by the solution would be `'Both'`. The box is both bulky due to its size and heavy due to its mass.

Python Solution

```python
class Solution:
    def categorize_box(self, length: int, width: int, height: int, mass: int) -> str:
        # Calculate the volume of the box
        volume = length * width * height

        # Check for the bulky condition
        # A box is bulky if any of its dimensions are 10,000 or more, or if its volume is 1 billion or more
        is_bulky = int(any(dimension >= 10000 for dimension in [length, width, height]) or volume >= 10**9)

        # Check for the heavy condition
        # A box is heavy if its mass is 100 or more
        is_heavy = int(mass >= 100)

        # Encode the condition using binary representation
        # This uses bit shifting to represent two binary digits, where
        # the left digit represents the "heavy" condition and
        # the right digit represents the "bulky" condition
        condition_code = (is_heavy << 1) | is_bulky

        # Define the dictionary to map the condition code to the corresponding string
        condition_dict = ['Neither', 'Bulky', 'Heavy', 'Both']

        # Return the string corresponding to the condition of the box
        return condition_dict[condition_code]

```

Java Solution

```java
class Solution {
    /**
     * Categorizes a box based on its dimensions and mass.
     *
     * @param length the length of the box
     * @param width  the width of the box
     * @param height the height of the box
     * @param mass   the mass of the box
     * @return a string that categorizes the box as "Neither", "Bulky", "Heavy", or "Both"
     */
    public String categorizeBox(int length, int width, int height, int mass) {
        // Calculate the volume of the box and store it as a long to prevent overflow.
        long volume = (long) length * width * height;

        // Determine if the box is bulky using the provided conditions.
        boolean isBulky = length >= 10000 || width >= 10000 || height >= 10000 || volume >= 1000000000;

        // Determine if the box is heavy using the provided condition.
        boolean isHeavy = mass >= 100;

        // Create an array of possible descriptions.
        String[] descriptions = {"Neither", "Bulky", "Heavy", "Both"};

        // Generate the index for the descriptions array based on the bulky and heavy flags.
        // isHeavy contributes to the higher order bit, so it's shifted left. isBulky contributes to the lower order bit.
        int index = (isHeavy ? 1 : 0) << 1 | (isBulky ? 1 : 0);

        // Return the corresponding description from the array.
        return descriptions[index];
    }
}
```

C++ Solution

```cpp
#include <string>

class Solution {
public:
    // This method categorizes a box based on its dimensions and mass.
    // The categories are 'Neither', 'Bulky', 'Heavy', or 'Both'.
    // A box is considered 'Bulky' if any of its dimensions is greater than or equal to 10000,
    // or its volume is greater than or equal to 1 billion.
    // A box is considered 'Heavy' if its mass is greater than or equal to 100.
    std::string categorizeBox(int length, int width, int height, int mass) {
        // Calculate the volume of the box as a long integer to prevent overflow
        long volume = static_cast<long>(length) * width * height;

        // Determine whether the box is bulky
        bool isBulky = (length >= 10000 || width >= 10000 || height >= 10000 || volume >= 1000000000);

        // Determine whether the box is heavy
        bool isHeavy = (mass >= 100);

        // Define an array of strings to hold the potential categories
        std::string descriptions[4] = {"Neither", "Bulky", "Heavy", "Both"};

        // Use bitwise logic to index the correct description:
        // - Shift 'isHeavy' left by 1 bit and 'or' it with 'isBulky' to form a 2-bit index
        int index = (isHeavy << 1) | isBulky;

        // Return the description based on the index
        return descriptions[index];
    }
};
```

Typescript Solution

```typescript
/**
 * Categorizes a box based on its dimensions and mass.
 *
 * @param {number} length - The length of the box in millimeters.
 * @param {number} width - The width of the box in millimeters.
 * @param {number} height - The height of the box in millimeters.
 * @param {number} mass - The mass of the box in kilograms.
 * @returns {string} A string categorization of the box: 'Neither', 'Bulky', 'Heavy', or 'Both'.
 */
function categorizeBox(length: number, width: number, height: number, mass: number): string {
    // Calculate the volume of the box
    const volume = length * width * height;

    // Initialize the category index
    let categoryIndex = 0;

    // Check for 'Bulky' category criteria: any dimension or volume above the threshold
    const maxDimensionSize = 10000; // millimeters
    const maxVolume = 1000000000;   // cubic millimeters
    if (length >= maxDimensionSize || width >= maxDimensionSize || height >= maxDimensionSize || volume >= maxVolume) {
        categoryIndex |= 1; // Set the first bit if 'Bulky'
    }

    // Check for 'Heavy' category criteria: mass above the threshold
    const maxMass = 100; // kilograms
    if (mass >= maxMass) {
        categoryIndex |= 2; // Set the second bit if 'Heavy'
    }

    // Determine the category based on the category index
    const categories = ['Neither', 'Bulky', 'Heavy', 'Both'];
    return categories[categoryIndex];
}
```

Time and Space Complexity

The time complexity of the function `categorizeBox` is `O(1)` because the operations performed within the function do not depend on the size of the input; they consist of basic arithmetic operations, comparisons, and bitwise operations which all take constant time.

The space complexity of the function is also `O(1)` as it only uses a fixed amount of additional memory for variables `v`, `bulky`, `heavy`, `i`, and the constant size list `d`, regardless of the input size.