2288. Apply Discount to Prices

Medium **String** Leetcode Link

Problem Description

The problem presents us with a string called a `sentence`, which is comprised of several words separated by single spaces. Among these words, some are considered `price` words. A word is identified as a price if it starts with a dollar sign `$` followed by a sequence of digits, such as `$100` or `$23`. The task is to identify these price words and apply a given discount percentage to them. After applying the discount, the new price should be rounded to two decimal places, and the original price within the sentence should be updated to the new discounted price. The goal is to then return the modified sentence with the updated prices. The discount provided is an integer value, and prices in the sentence can have up to 10 digits.

Intuition

To solve this problem, we first need to identify the words that represent prices, which can be done by checking if a word starts with the `$` sign and the following characters are all digits. For this, we iterate through each word in the sentence and apply this check.

Once we confirm a word is indeed a price, we proceed to calculate the discounted price. To do this, we strip the dollar sign and convert the remaining part of the string to an integer to get the original price. We then apply the discount by multiplying the original price by `(1 - discount / 100)` to obtain the discounted price.

To ensure that the discounted price is correctly formatted with two decimal places, we use Python's string formatting feature: `"{:.2f}".format(discounted_price)` or `f'{discounted_price:.2f}'`. The string formatting ensures the resulting price string will always have two decimal places.

After all the processing, we combine the words back into a single string making sure to add back the space separator between words, and return the modified sentence.

Solution Approach

To implement the solution to this problem, the solution makes use of some built-in Python features such as string methods and list comprehensions. Here's a step-by-step explanation of the approach:

1. We start by splitting the given sentence into individual words. This is done with the `.split()` method, which, when used without any arguments, splits a string by white spaces.
2. We iterate over each word in the list of words to determine if a word represents a price. This is identified by checking two conditions: the first character must be the dollar sign '`$`', and all the subsequent characters must be digits. This is checked using the `str.isdigit()` method on the slice of the word excluding the first character.
3. For each word that represents a price, we remove the dollar sign and calculate the discounted price. This is achieved by converting the remaining string to an integer (`int(w[1:])`), then calculating the discount, which is the original price subtracted by the original price multiplied by the discount rate (`int(w[1:]) * (1 - discount / 100)`). The discount rate is created by dividing the `discount` by `100` since `discount` is given as a percentage.
4. The discounted price is then formatted to have exactly two decimal places using Python's formatted string syntax: `f'${discounted_price:.2f}'`. This ensures that even if a discounted price is a whole number or has fewer than two decimal places, it will still be displayed with exactly two decimal places.
5. The word list is then transformed by replacing the original price words with the new discounted price words where necessary.
6. Finally, the words are joined back together into a single string sentence with white space separators using the `.join()` method. This reconstructed sentence, containing the updated prices, is returned as the final solution.

The code uses simple data structures like lists and strings and combines them with string formatting and list comprehension, which makes the approach efficient, as it requires only a single pass through the list of words. No complex data structures or algorithms are necessary due to the nature of the problem.

Example Walkthrough

Let's assume our input sentence is "`I need $500 for groceries and $30 for the bus.`", and the discount we are supposed to apply is `10%`.

1. We start by splitting the given sentence into individual words using the `.split()` method.

   Original sentence: "I need $500 forgroceriesand$30 for the bus." After splitting: `["I", "need", "$500", "for", "groceries", "and", "$30", "for", "the", "bus."]`

2. We iterate over each word in the list:
   - "I", "need", "for", "groceries", "and", "for", "the", "bus." do not start with `$`, so we keep them as they are.
   - "$500” and”$30" start with `$` and are followed by digits, so they are identified as price words.
3. For each price word:
   - We remove the dollar sign and convert the remaining part of the word to an integer: `500` and `30`.
   - Apply the discount by multiplying the price by `(1 - 0.10)`:
     - $500 * (1 - 0.10)$ =500 * 0.90 = \$450.00
     - $30 * (1 - 0.10)$ =30 * 0.90 = \$27.00
4. The discounted prices are then formatted to have exactly two decimal places:
   - For $500 with a 10${450.00:.2f}'`results in`"\$450.00"`.
   - For $30 with a 10${27.00:.2f}'`results in`"\$27.00"`.
5. We replace the original price words in the word list with the new discounted price words:

   Before: `["I", "need", "$500", "for", "groceries", "and", "$30", "for", "the", "bus."]` After: `["I", "need", "$450.00", "for", "groceries", "and", "$27.00", "for", "the", "bus."]`

6. Finally, we join the words back together using the `.join()` method to form the final sentence:

   `["I", "need", "$450.00", "for", "groceries", "and", "$27.00", "for", "the", "bus."]` becomes "I need $450.00 forgroceriesand$27.00 for the bus."

The method returns this modified sentence with updated prices, reflecting the applied discount.

Python Solution

```python
class Solution:
    def discountPrices(self, sentence: str, discount: int) -> str:
        # Initialize a list to hold the resulting words after processing
        processed_words = []

        # Split the sentence into words
        for word in sentence.split():
            # Check if the word is a price (starts with '$' and followed by digits)
            if word[0] == '$' and word[1:].isdigit():
                # If it is a price, calculate the new price after the discount
                original_price = int(word[1:])  # Remove the '$' sign and convert the rest to integer
                discounted_price = original_price * (1 - discount / 100)  # Apply discount
                word = f'${discounted_price:.2f}'  # Format the discounted price

            # Append the processed word to the list
            processed_words.append(word)

        # Join the processed words back into a sentence and return it
        return ' '.join(processed_words)

```

Java Solution

```java
class Solution {
    /**
     * Applies a discount to all prices within the sentence.
     *
     * @param sentence The sentence containing potential prices.
     * @param discount The discount rate to be applied.
     * @return A sentence with discounts applied to valid prices.
     */
    public String discountPrices(String sentence, int discount) {
        // Split the sentence into individual words
        String[] words = sentence.split(" ");

        // Iterate over each word to check if it's a price
        for (int i = 0; i < words.length; ++i) {
            // If a word is a valid price, apply discount
            if (isValidPrice(words[i])) {
                // Parse the numeric value from the price, apply discount and convert it back to a string
                double originalPrice = Long.parseLong(words[i].substring(1));
                double discountedPrice = originalPrice * (1 - discount / 100.0);
                words[i] = String.format("$%.2f", discountedPrice);
            }
        }

        // Join the words back into a single string with spaces between them
        return String.join(" ", words);
    }

    /**
     * Checks if a given string is a valid price format.
     *
     * @param word The word to be checked.
     * @return True if the word is a valid price; otherwise, False.
     */
    private boolean isValidPrice(String word) {
        // A valid price must start with a dollar sign and have more than one character
        if (word.charAt(0) != '$' || word.length() == 1) {
            return false;
        }

        // Check each character after the dollar sign to ensure they are digits
        for (int i = 1; i < word.length(); ++i) {
            if (!Character.isDigit(word.charAt(i))) {
                return false;
            }
        }

        // If all checks pass, this is a valid price
        return true;
    }
}

```

C++ Solution

```cpp
#include <sstream> // For istringstream
#include <string>
#include <cctype> // For isdigit

class Solution {
public:
    // Function to apply a discount to price tags in a given sentence
    string discountPrices(string sentence, int discount) {
        istringstream inputStream(sentence);
        string word;
        string answer;

        // Lambda function to check if a word is a valid price
        auto isValidPrice = [](string s) {
            // Price should start with '$' and be longer than 1 character
            if (s[0] != '$' || s.size() == 1) {
                return false;
            }
            // Check if all characters except the first are digits
            for (int i = 1; i < s.size(); ++i) {
                if (!isdigit(s[i])) {
                    return false;
                }
            }
            return true;
        };

        // Iterate over words in the input sentence
        while (inputStream >> word) {
            // If the word is a valid price, calculate the discounted price
            if (isValidPrice(word)) {
                // Extract numeric value , apply discount and convert back to price format
                long long originalPrice = stoll(word.substr(1));
                long long discountedPrice = originalPrice * (100 - discount);

                char formattedPrice[20];
                // Format the discounted price to include dollars and cents
                sprintf(formattedPrice, "$%lld.%02lld", discountedPrice / 100, discountedPrice % 100);
                answer += formattedPrice;
            } else {
                // Keep the word unchanged if it's not a valid price
                answer += word;
            }
            // Add a space after each word
            answer += ' ';
        }

        // Remove the trailing space from the last word
        answer.pop_back();
        return answer;
    }
};

```

Typescript Solution

```typescript
function discountPrices(sentence: string, discount: number): string {
    // Calculate the multiplier for the discount (e.g., if discount is 20%, multiplier will be 0.8)
    const discountMultiplier = (100 - discount) / 100;

    // Regular expression to match prices in the format $[number] which may or may not have decimal places
    // It ensures that the number doesn't start with 0 unless it's followed by a decimal point
    let priceRegex = new RegExp(/^(\$)(([1-9]\d*\.?\d*)|(0\.\d*))$/g);

    // Split the sentence into words, and map over each word to check for price patterns
    let words = sentence.split(' ').map(word => {
        // If word does not match the price format, return it unchanged
        if (!priceRegex.test(word)) return word;

        // If word is a price, apply the discount and format the result to two decimal places
        return word.replace(priceRegex, (match, dollarSign, numericPart) => {
            // Parse the numeric part as a float, apply the discount, and convert it back to a string
            let discountedPrice = (parseFloat(numericPart) * discountMultiplier).toFixed(2);
            return `${dollarSign}${discountedPrice}`; // Re-attach the dollar sign and return the discounted price
        });
    });

    // Join the processed words back into a single string and return it
    return words.join(' ');
}

```

Time and Space Complexity

Time Complexity

The given code has a primary loop that iterates over each word in the input sentence. This operation takes `O(n)` time where `n` is the number of characters in the input string since the `split()` function runs in O(n) time and iterations depend on the number of words which is proportional to the number of characters.

Inside the loop, the code checks whether a word starts with a dollar sign and then whether the rest of the word is a valid digit string. The check `w[1:].isdigit()` also takes at worst `O(m)` time where `m` is the number of characters in the word.

If a word is a valid price, it calculates the discounted price. The calculations inside the loop (`int(w[1:]) * (1 - discount / 100)`) run in constant time, `O(1)`, however, formatting this to a string with two decimal places has a cost, but it is generally considered `O(1)`.

Therefore, the total time complexity is `O(n * m)`, however, if we consider that the length of the words can be bounded by a constant due to the nature of the prices (which doesn't usually exceed a certain number of digits), the complexity can also be represented as `O(n)` where `n` is the total number of characters in the sentence.

Space Complexity

The space complexity of the algorithm is due to the storage required for the list `ans`. This list stores each word after potentially modifying it. Since every word has to be stored regardless of whether it is modified, this means that it requires `O(n)` additional space, where `n` is the number of characters in the input string. The input string itself also takes `O(n)` space.

Thus, the space complexity of the code is `O(n)`, where `n` represents the number of characters in the input string.