Documentation for compareVersion Method

Overview

The compareVersion method is designed to compare two version strings, each consisting of numeric revisions separated by dots (.). The comparison is done by evaluating each revision from left to right, and the method returns an integer indicating whether the first version is less than, greater than, or equal to the second version.

Parameters

- **version1: str**
  - A version string representing the first version to be compared.
  - The version string consists of one or more numeric revisions separated by dots (.).
  - *Constraints:*
    - ✓ The length of version1 is between 1 and 500 characters.
    - ✓ The version string contains only digits and dots.
    - ✓ Each revision in the version string can be stored within a 32-bit integer.

- **version2: str**
  - A version string representing the second version to be compared.
  - Similar to version1, the version string consists of one or more numeric revisions separated by dots (.).
  - *Constraints:*
    - ✓ The length of version2 is between 1 and 500 characters.
    - ✓ The version string contains only digits and dots.
    - ✓ Each revision in the version string can be stored within a 32-bit integer.

Return Value

- **int**
  - *The method returns an integer to indicate the result of the comparison:*
    - ✓ -1: Indicates that version1 is less than version2.
    - ✓ 1: Indicates that version1 is greater than version2.
    - ✓ 0: Indicates that both versions are equal.

Detailed Description

1. **Splitting the Version Strings:**
   - The method begins by splitting the version strings version1 and version2 into lists of individual revisions using the dot (.) as a delimiter. Each list contains strings representing the revisions of the version.

2. **Normalizing the Length of Revision Lists:**
   - Since the number of revisions may differ between the two version strings, the method ensures that both revision lists have the same length. This is done by padding the shorter list with 0 values for any missing revisions. The length of the lists is determined by the version string with the most revisions.

3. **Comparing Revisions:**
   - *The method compares the corresponding revisions from both version strings in a left-to-right order:*
     - Each revision is converted to an integer to handle cases where revisions might have leading zeros.
     - If a revision from version1 is less than the corresponding revision from version2, the method returns -1, indicating that version1 is less than version2.
     - If a revision from version1 is greater than the corresponding revision from version2, the method returns 1, indicating that version1 is greater than version2.

4. **Handling Equal Revisions:**
   - If all corresponding revisions are equal after comparing each pair, the method returns 0, indicating that the two version strings are equivalent.

Example 1:

- **Input:** version1 = "1.2", version2 = "1.10"
- **Output:** -1
- **Explanation:** The second revision in version1 is 2, and in version2, it is 10. Since 2 is less than 10, version1 is considered less than version2.

Example 2:

- **Input:** version1 = "1.01", version2 = "1.001"
- **Output:** 0
- **Explanation:** Both revisions represent the same integer 1 when leading zeros are ignored. Therefore, the versions are equal.

Example 3:

- **Input:** version1 = "1.0", version2 = "1.0.0.0"
- **Output:** 0
- **Explanation:** Missing revisions in version1 are treated as 0, making the two versions equivalent.

Constraints

- The length of both version strings is between 1 and 500 characters.
- Both version strings consist of digits and dots only.
- Each revision in the version strings can be safely converted to and stored as a 32-bit integer.

Edge Cases

- **Different Number of Revisions:** The method handles cases where one version string has fewer revisions than the other by treating the missing revisions as 0.

- **Leading Zeros:** Revisions with leading zeros are correctly interpreted as their integer value, ensuring accurate comparisons.

- **Empty Revisions:** The method assumes that version strings are valid and non-empty. Empty revisions are treated as 0.

This method is efficient and adheres to the given constraints, ensuring accurate comparison of version numbers in a variety of scenarios.