411. Minimum Unique Word Abbreviation

Problem Explanation

Given a target string and a set of strings in a dictionary, you are to find an abbreviation of this target string with the smallest possible length such that it doesn't conflict with abbreviations of the strings in the dictionary.

It is important to note that each number or letter in the abbreviation is considered length = 1.

For example, given `target = "apple"` and `dictionary = ["blade"]`, the result would be "a4" because "5" or "4e" might conflict with "blade".

Approach

The approach is to generate all possible abbreviations of a given target string and then check these against the dictionary to find the abbreviation that does not conflict with any of the words in the dictionary. If there are multiple possible abbreviations with the same shortest length, any of them can be returned.

The way to generate all possible abbreviations is by iterating through all the bitmasks of the length of the string. Each bit in the bitmask represents whether the corresponding character in the string is replaced by a number (1) or remained as is (0). When we find a candidate abbreviation, it means that this abbreviation does not conflict with any of the words in the dictionary. We then store it in a list of possible abbreviations.

After generating all possible abbreviations, we go through this list, calculate their length and return the one with the smallest possible length.

Let's see through examples how it works.

Consider `target = "apple"` and `dictionary = ["blade"]`. The target word "apple" can be abbreviated as ["5", "4e", "a4"....]. And the dictionary word "blade" can be abbreviated as ["5", "4e", "b4"....]. It can be seen that "5" or "4e" is conflict with "blade" because they're both valid abbreviations for "apple" and "blade". We then need to find a valid abbreviation for "apple" that does not conflict with any of the abbreviations for "blade", return it with the smallest possible length which is "a4".

Solution

C++

```
cpp
class Solution {
public:
    string minAbbreviation(string target, vector<string>& dictionary) {
        const int m = target.length();
        vector<int> masks;
        for (const string& word : dictionary) {
            if (word.length() != m)
                continue;
            // Compute each valid word's mask
            masks.push_back(getMask(target, word));
        }
        // If there's no word with the same length, simply return the length
        if (masks.empty())
            return to_string(m);
        // Compute all the abbreviations
        vector<string> abbrs;
        const int maxCand = pow(2, m);
        for (int cand = 0; cand < maxCand; ++cand)
            if (all_of(begin(masks), end(masks),  [cand](int mask) { return cand & mask; }))
                abbrs.push_back(getAbbr(target, cand));
        string ans = target;
        // Among all the abbreviations find the smallest one
        for (const string& abbr : abbrs)
            if (getAbbrLen(abbr) < getAbbrLen(ans))
                ans = abbr;
        return ans;
    }

private:
    int getMask(const string& target, const string& word) {
        const int m = target.length();
        // If two characters at the same position are the same, the mask bit is 0
        // Otherwise, the mask bit is 1
        int mask = 0;
        for (int i = 0; i < m; ++i)
            if (word[i] != target[i])
                mask |= 1 << m - 1 - i;
        return mask;
    }

    // Convert the candidate to the abbreviation
    string getAbbr(const string& target, int cand) {
        const int m = target.length();
        string abbr;
        int replacedCount = 0;
        for (int i = 0; i < m; ++i)
            if (cand >> m - 1 - i & 1) {
                if (replacedCount > 0)
                    abbr += to_string(replacedCount);
                abbr += target[i];
                replacedCount = 0;
            } else {
                ++replacedCount;
            }
        if (replacedCount > 0)
            abbr += to_string(replacedCount);
        return abbr;
    }

    int getAbbrLen(const string& abbr) {
        int abbrLen = 0;
        int i = 0, j = 0;
        while (i < abbr.length()) {
            if (isalpha(abbr[j]))
                ++j;
            else
                while (i < abbr.length() && isdigit(abbr[j]))
                    ++j;
            ++abbrLen;
            i = j;
        }
        return abbrLen;
    }
};
```

Unfortunately the solution code provided for this problem does not include solutions in Python, JavaScript, Java and C#. The algorithms used in the solution are quite advanced and could take a considerable amount of time to converted to each programming language.## Python

Here's how you might implement this algorithm in Python:

```
python
class Solution:
    def minAbbreviation(self, target, dictionary):
        def mask(word1, word2):
            """Calculate a bitmask for two words of same length"""
            word_len = len(word1)
            msk = 0
            for i in range(word_len):
                if word1[i] != word2[i]:
                    msk |= 1 << word_len - 1 - i
            return msk

        def to_abbr(target, cand):
            """Convert a candidate bitmask to an abbreviation"""
            word_len = len(target)
            abbr = ''
            count = 0
            for i in range(word_len):
                if cand >> word_len - 1 - i & 1:
                    if count > 0:
                        abbr += str(count)
                    abbr += target[i]
                    count = 0
                else:
                    count += 1
            if count > 0:
                abbr += str(count)
            return abbr

        def abbr_length(abbr):
            """Calculate the length of an abbreviation"""
            abbr_len = 0
            i = 0
            while i < len(abbr):
                if abbr[i].isalpha():
                    i += 1
                else:
                    while i < len(abbr) and abbr[i].isdigit():
                        i += 1
                abbr_len += 1
            return abbr_len

        m = len(target)
        masks = [mask(target, word) for word in dictionary if len(word) == m]

        if not masks:
            return str(m)

        abbrs = []
        max_cand = 2 ** m
        for cand in range(max_cand):
            if all(cand & msk for msk in masks):
                abbrs.append(to_abbr(target, cand))

        shortest = target
        for abbr in abbrs:
            if abbr_length(abbr) < abbr_length(shortest):
                shortest = abbr

        return shortest
```