Shortest Word Distance II

This is a **follow up** of Shortest Word Distance. The only difference is now you are given the list of words and your method will be called *repeatedly* many times with different parameters. How would you optimize it?

Design a class which receives a list of words in the constructor, and implements a method that takes two words *word1* and *word2* and return the shortest distance between these two words in the list.

For example,
Assume that words = `["practice", "makes", "perfect", "coding", "makes"]`.

Given *word1* = `“coding”`, *word2* = `“practice”`, return 3.
Given *word1* = `"makes"`, *word2* = `"coding"`, return 1.

Note:
You may assume that *word1* **does not equal to** *word2*, and *word1* and *word2* are both in the list.

Solution 1

```java
public class WordDistance {

private Map<String, List<Integer>> map;

public WordDistance(String[] words) {
    map = new HashMap<String, List<Integer>>();
    for(int i = 0; i < words.length; i++) {
        String w = words[i];
        if(map.containsKey(w)) {
            map.get(w).add(i);
        } else {
            List<Integer> list = new ArrayList<Integer>();
            list.add(i);
            map.put(w, list);
        }
    }
}

public int shortest(String word1, String word2) {
    List<Integer> list1 = map.get(word1);
    List<Integer> list2 = map.get(word2);
    int ret = Integer.MAX_VALUE;
    for(int i = 0, j = 0; i < list1.size() && j < list2.size(); ) {
        int index1 = list1.get(i), index2 = list2.get(j);
        if(index1 < index2) {
            ret = Math.min(ret, index2 - index1);
            i++;
        } else {
            ret = Math.min(ret, index1 - index2);
            j++;
        }
    }
    return ret;
}
}
```

written by qianzhige original link here

Solution 2

```
class WordDistance {
public:
    WordDistance(vector<string>& words) {
        for(int i=0;i<words.size();i++)
            wordMap[words[i]].push_back(i);
    }
    int shortest(string word1, string word2) {
        int  i=0, j=0, dist = INT_MAX;
        while(i < wordMap[word1].size() && j <wordMap[word2].size()) {
            dist = min(dist, abs(wordMap[word1][i] - wordMap[word2][j]));
            wordMap[word1][i]<wordMap[word2][j]?i++:j++;
        }
        return dist;
    }
private:
    unordered_map<string, vector<int>> wordMap;
};
```

written by luming89 original link here

Solution 3

Special thanks to @qianzhige and @jeantimex

The basic idea is using HashMap to store the position of each word in the array in constructor function. Then in 'shortest' function, use the method similar as merge sort to get the minimum difference between indices of word1 and word2.

Time complexity of 'shortest' function = O(NumberOfWord1 + NumberOfWord2)

Extra space = O(NumberOfWords)

```java
public class WordDistance {
    private Map<String, List<Integer>>map;
    public WordDistance(String[] words) {
        map = new HashMap<String, List<Integer>>();
        for (int i = 0; i < words.length; i++) {
            if (!map.containsKey(words[i]))
                map.put(words[i], new LinkedList<Integer>());
            map.get(words[i]).add(i);
        }
    }
    public int shortest(String word1, String word2) {
        List<Integer> list1 = map.get(word1);
        List<Integer> list2 = map.get(word2);
        int i = 0, j = 0, res = Integer.MAX_VALUE;
        while (i < list1.size() && j < list2.size()) {
            int index1 = list1.get(i);
            int index2 = list2.get(j);
            if (index1 < index2) {
                i++;
                res = Math.min(res, index2 - index1);
            }
            else {
                j++;
                res = Math.min(res, index1 - index2);
            }
        }
        return res;
    }
}
```

written by Pixel_ original link here