Sentence Similarity

Given two sentences `words1, words2` (each represented as an array of strings), and a list of similar word pairs `pairs`, determine if two sentences are similar.

For example, "great acting skills" and "fine drama talent" are similar, if the similar word pairs are `pairs = [["great", "fine"], ["acting","drama"], ["skills","talent"]]`.

Note that the similarity relation is not transitive. For example, if "great" and "fine" are similar, and "fine" and "good" are similar, "great" and "good" are **not** necessarily similar.

However, similarity is symmetric. For example, "great" and "fine" being similar is the same as "fine" and "great" being similar.

Also, a word is always similar with itself. For example, the sentences `words1 = ["great"], words2 = ["great"], pairs = []` are similar, even though there are no specified similar word pairs.

Finally, sentences can only be similar if they have the same number of words. So a sentence like `words1 = ["great"]` can never be similar to `words2 = ["doubleplus","good"]`.

Note:

- The length of `words1` and `words2` will not exceed `1000`.
- The length of `pairs` will not exceed `2000`.
- The length of each `pairs[i]` will be `2`.
- The length of each `words[i]` and `pairs[i][j]` will be in the range `[1, 20]`.

Solution 1

java

```java
class Solution {
    public boolean areSentencesSimilar(String[] a, String[] b, String[][] pairs) {
        if (a.length != b.length) return false;
        Map<String, Set<String>> map = new HashMap<>();
        for (String[] p : pairs) {
            if (!map.containsKey(p[0])) map.put(p[0], new HashSet<>());
            map.get(p[0]).add(p[1]);
        }

        for (int i = 0; i < a.length; i++)
            if (!a[i].equals(b[i]) && !map.getOrDefault(a[i], new HashSet<>()).conta
ins(b[i]) && !map.getOrDefault(b[i], new HashSet<>()).contains(a[i]))
                return false;
        return true;
    }
}
```

C++

```cpp
class Solution {
public:
    bool areSentencesSimilar(vector<string>& words1, vector<string>& words2, vector
<pair<string, string>> pairs) {
        if (words1.size() != words2.size()) return false;
        map<string, set<string>> map;
        for (pair<string, string> p : pairs)
            map[p.first].insert(p.second);

        for (int i = 0; i < words1.size(); i++)
            if (words1[i] != words2[i] && !map[words1[i]].count(words2[i]) && !map[w
ords2[i]].count(words1[i]))
                return false;
        return true;
    }
};
```

written by alexander original link here

Solution 2

Sentence Similarity I (Transitive is not allowed.)

```
class Solution {
    public boolean areSentencesSimilar(String[] words1, String[] words2, String[][]
pairs) {
        if (words1.length != words2.length) {
            return false;
        }

        Map<String, Set<String>> pairInfo = new HashMap<>();

        for (String[] pair : pairs) {
            if (!pairInfo.containsKey(pair[0])) {
                pairInfo.put(pair[0], new HashSet<>());
            }
            if (!pairInfo.containsKey(pair[1])) {
                pairInfo.put(pair[1], new HashSet<>());
            }

            pairInfo.get(pair[0]).add(pair[1]);
            pairInfo.get(pair[1]).add(pair[0]);
        }

        for (int i = 0; i < words1.length; i++) {
            if (words1[i].equals(words2[i])) continue;

            if (!pairInfo.containsKey(words1[i])) {
                return false;
            }
            if (!pairInfo.get(words1[i]).contains(words2[i])) {
                return false;
            }
        }

        return true;
    }
}
```

Sentence Similarity II (Transitive is allowed.)
The idea is simple:

1. Build the graph according to the similar word pairs. Each word is a graph node.
2. For each word in words1, we do DFS search to see if the corresponding word is existing in words2.

See the clean code below. Happy coding!

```
class Solution {
    public boolean areSentencesSimilarTwo(String[] words1, String[] words2, String[
][] pairs) {
        if (words1.length != words2.length) {
            return false;
        }

        //Build the graph;
        Map<String, Set<String>> pairInfo = new HashMap<>();
        for (String[] pair : pairs) {
            if (!pairInfo.containsKey(pair[0])) {
                pairInfo.put(pair[0], new HashSet<>());
            }
            if (!pairInfo.containsKey(pair[1])) {
                pairInfo.put(pair[1], new HashSet<>());
            }
            pairInfo.get(pair[0]).add(pair[1]);
            pairInfo.get(pair[1]).add(pair[0]);
        }

        for (int i = 0; i < words1.length; i++) {
            if (words1[i].equals(words2[i])) continue;
            if (!pairInfo.containsKey(words1[i])) return false;
            if (!dfs(words1[i], words2[i], pairInfo, new HashSet<>())) return false
;    //Search the graph.
        }

        return true;
    }

    public boolean dfs(String source, String target, Map<String, Set<String>> pairI
nfo, Set<String> visited) {
        if (pairInfo.get(source).contains(target)) return true;

        visited.add(source);
        for (String next : pairInfo.get(source)) {
            if (!visited.contains(next) && dfs(next, target, pairInfo, visited)) {
                return true;
            }
        }
        return false;
    }
}
```

written by FLAGbigoffer original link here

Solution 3

Time complexity: O(nk), space complexity O(nk). n is number of words, k is the average length of words.

```
class Solution {
    public boolean areSentencesSimilar(String[] words1, String[] words2, String[][]
pairs) {
        if (words1.length != words2.length) return false;
        Map<String, Set<String>> map = new HashMap<>();

        for (String[] p : pairs) {
            Set<String> set0 = map.getOrDefault(p[0], new HashSet<>());
            set0.add(p[1]);
            map.put(p[0], set0);

            Set<String> set1 = map.getOrDefault(p[1], new HashSet<>());
            set1.add(p[0]);
            map.put(p[1], set1);
        }

        for (int i = 0; i < words1.length; i++) {
            if (words1[i].equals(words2[i])) continue;
            if (map.containsKey(words1[i]) && map.get(words1[i]).contains(words2[i]
)) continue;
            return false;
        }

        return true;
    }
}
```

written by shawngao original link here