Sentence Similarity II

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, `words1 = ["great", "acting", "skills"]` and `words2 = ["fine", "drama", "talent"]` are similar, if the similar word pairs are `pairs = [["great", "good"], ["fine", "good"], ["acting","drama"], ["skills","talent"]]`.

Note that the similarity relation **is** transitive. For example, if "great" and "good" are similar, and "fine" and "good" are similar, then "great" and "fine" **are similar**.

Similarity is also 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

```
public boolean areSentencesSimilarTwo(String[] words1, String[] words2, String[][]
pairs) {

        if(words1.length!=words2.length)
            return false;

        Map<String,String> map = new HashMap<String,String>();

        for(String[] pair : pairs){

            String word1 = pair[0];
            String word2 = pair[1];

            if(!map.containsKey(word1))
                map.put(word1,word1);
            if(!map.containsKey(word2))
                map.put(word2,word2);

            setParent(map,word1,word2);

        }

        for(int i=0;i<words1.length;i++){

            String word1 = words1[i];
            String word2 = words2[i];

            String parent1 = getParent(word1,map);
            String parent2 = getParent(word2,map);

            if(!parent1.equals(parent2))
                return false;

        }
        return true;


    }

    public String getParent(String word,Map<String,String> map){

        if(!map.containsKey(word))
            return word;

        while(word!=map.get(word))
            word = map.get(word);
        return word;

    }

    public void setParent(Map<String,String> map,String word1,String word2){

        String p1 = getParent(word1,map);
        String p2 = getParent(word2,map);
```

```
        map.put(p1,p2);

    }
```

written by godusopp original link here

Solution 2

Notice there's no java DFS solution posted by others. I love DFS, what about you? 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;
        }

        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

Whenever we see a list of pairs as input, one probable approach will be to treat that as a list of edges and model the question as a graph. In this question, the idea here is to connect words to their similar words, and all connected words are similar. In each connected component of a graph, select any word to be the root word and then generate a mapping of word to root word. If two words are similar, they have the same root word.

First build a graph from `pairs`. An input of `[["great","good"],["fine","good"],["drama","acting"],["skills","talent"]]` will have a graph that looks like:

```
# words
{
  "great": set(["good"]),
  "good": set(["great", "fine"]),
  "talent": set(["skills"]),
  "skills": set(["talent"]),
  "drama": set(["acting"]),
  "acting": set(["drama"]),
  "fine": set(["good"]),
}
```

Next, we do a DFS on each word to try to group the connected words together by assigning each word to a root word. The `similar_words` dict maps every word to a root word so that we can immediately know whether two words are similar just by looking up this dict and seeing if they have the same root word:

```
# similar_words
{
  "great": "great",
  "good": "great",
  "talent": "talent",
  "skills": "talent",
  "drama": "drama",
  "acting": "drama",
  "fine": "great",
}
```

- Yangshun

```
class Solution(object):
    def areSentencesSimilarTwo(self, words1, words2, pairs):
        from collections import defaultdict
        if len(words1) != len(words2): return False
        words, similar_words = defaultdict(set), {}
        [(words[w1].add(w2), words[w2].add(w1)) for w1, w2 in pairs]
        def dfs(word, root_word):
            if word in similar_words: return
            similar_words[word] = root_word
            [dfs(synonym, root_word) for synonym in words[word]]
        [dfs(word, word) for word in words]
        return all(similar_words.get(w1, w1) == similar_words.get(w2, w2) for w1, w2
in zip(words1, words2))
```

A longer version with inline comments can be found below:

```
class Solution(object):
    def areSentencesSimilarTwo(self, words1, words2, pairs):
        from collections import defaultdict
        if len(words1) != len(words2):
            return False
        words = defaultdict(set)
        # Build the graph from pairs.
        for w1, w2 in pairs:
            words[w1].add(w2)
            words[w2].add(w1)

        similar_words = {}
        def dfs(word, root_word):
            if word in similar_words:
                return
            similar_words[word] = root_word
            [dfs(synonym, root_word) for synonym in words[word]]

        # Assign root words.
        [dfs(word, word) for word in words]

        # Compare words.
        return all(similar_words.get(w1, w1) == similar_words.get(w2, w2) for w1, w2
in zip(words1, words2))
```

written by yangshun original link here