126. Word Ladder II

Jul 24, 2018 | leetcode | Hits

Problem description:

Given two words (beginWord and endWord), and a dictionary's word list, find all shortest transformation sequence(s) from beginWord to endWord, such that:

Only one letter can be changed at a time
Each transformed word must exist in the word list. Note that beginWord is not a transformed word.
Note:

Return an empty list if there is no such transformation sequence.
All words have the same length.
All words contain only lowercase alphabetic characters.
You may assume no duplicates in the word list.
You may assume beginWord and endWord are non-empty and are not the same.
Example 1:

Input:
beginWord = “hit”,
endWord = “cog”,
wordList = [“hot”,”dot”,”dog”,”lot”,”log”,”cog”]

Output:
[
[“hit”,”hot”,”dot”,”dog”,”cog”],
[“hit”,”hot”,”lot”,”log”,”cog”]
]
Example 2:

Input:
beginWord = “hit”
endWord = “cog”
wordList = [“hot”,”dot”,”dog”,”lot”,”log”]

Output: []

Explanation: The endWord “cog” is not in wordList, therefore no possible transformation.

Solution:

This is a follow up for 127. Word Ladder. If you have this question on interview, maybe go with a naive BFS solution, then improve it with bi-direction BFS.

The solution came with one direction BFS, the step is as follows.

1. Use BFS to solve question, so we need a queue. `queue<vector<string>> paths` , to store the paths in current level.
2. A dictionary that help to search if a mutate string is in wordlist. The mutate string is created by the same method in 127. Word Ladder. For each character in a word, try to change it to another character, and check whether if it's in the dictionary.
3. Once we know the words in previous level(can think of as the BFS level), we can not use those words again, so we need a set(`unordered_set<string> words`) to record the used string on this level.
4. Maintain a `minlevel` to speed up the process. Because if we can find a path to find `endWord` in 5 hops, then we don't need to find any path that is greater than 5.

```
class Solution {
public:
    vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wo
        vector<vector<string>> res;
        unordered_set<string> dict(wordList.begin(), wordList.end()); //for lookup if the c
        vector<string> p{beginWord};
        queue<vector<string>> paths;

        paths.push(p);
        int level = 1, minLevel = INT_MAX;
        unordered_set<string> words;

        while (!paths.empty()) {
            auto t = paths.front(); //start with beginWord
            paths.pop();

            if (t.size() > level) {
                for (string w : words) dict.erase(w); //to remove the word that already use
                words.clear();
                level = t.size();
                if (level > minLevel) break;
            }
            string last = t.back(); //last element of curent path, try to find next word to
            for (int i = 0; i < last.size(); ++i) {
                string newLast = last;
                for (char ch = 'a'; ch <= 'z'; ++ch) {
                    newLast[i] = ch;
                    if (!dict.count(newLast)) continue;

                    //can find another word after changing one character in dictionary
                    words.insert(newLast); //put it into set, so we won't use it again

                    /*
                    example:
                    t: abc->abd->acd
                    newPath= abc->abd->acd + acc
                    */
                    vector<string> nextPath = t;
                    nextPath.push_back(newLast);

                    if (newLast == endWord) {
                        res.push_back(nextPath);
                        minLevel = level;
                    }
                    else paths.push(nextPath);
                }
            }
        }
        return res;
    }
};
```

reference:
https://www.youtube.com/watch?v=lmypbtgdpuQ
http://www.cnblogs.com/grandyang/p/4548184.html
https://goo.gl/4rcWS6
https://goo.gl/t1vxXg

string backtracking facebook microsoft google bfs amazon array

57. Insert Interval
154. Find Minimum in Rotated Sorted Array II

Search

categories
Reading
STL
algorithm
behavioral question
amazon
google
leetcode
amazon
hash table
design
linked list
note
microsoft
note
programming language
study
project
study
programming language
tutorial

tags
greddy leetcode easy hard string dynamic programming backtracking facebook microsoft google apple bfs medium stack amazon dfs array two pointers greedy dp design graph hash table linked list pure storage uber union find citrix easy medium hard sort merge sort linkedin binary search bloomberg yelp two pointer heap topological sort trie adobe binary search tree airbnb divide and conquer lyft math recursion Floyd's Tortoise and Hare Algorithm epic system binary indexed tree permutation brainteaser bit manipulation queue bucket sort has table peak adobe facebook oracle tree BFS sliding window String two sigma paypal DFS partition subset ebay twitter east indeed map snapchat interview STL C++ 11 MSDN double linked list BST java spring space complexity cycle detection algorithm economics blockchain c++ lower_bound upper_bound Tree tree traversal preorder inorder postorder binary tree refer job web hexo github

recent
1428. Leftmost Column with at Least a One
1396. Design Underground System
953. Verifying an Alien Dictionary
117. Populating Next Right Pointers in Each Node II
116. Populating Next Right Pointers in Each Node
122. Best Time to Buy and Sell Stock II
863. All Nodes Distance K in Binary Tree
1448. Count Good Nodes in Binary Tree
394. Decode String
695. Max Area of Island

blogroll
linlaw's Linkedin
linlaw's Github