November 9, 2022 ▪ Graph

G-30 : Word Ladder-II

Given two distinct words **startWord** and **targetWord**, and a list denoting wordList of unique words of equal lengths. Find all shortest transformation sequence(s) from startWord to targetWord. You can return them in any order possible.

In this problem statement, we need to keep the following conditions in mind:

- A word can only consist of lowercase characters.
- Only one letter can be changed in each transformation.
- Each transformed word must exist in the wordList including the targetWord.
- startWord may or may not be part of the wordList.
- Return an empty list if there is no such transformation sequence.

Note: Please watch the previous video of this series before moving on to this particular problem as this is just an extension of the problem Word Ladder-I that is being discussed previously. The approach used for this problem would be similar to the approach used in that question.

Examples:

```
Example 1:
Input:
startWord = "der", targetWord =
"dfs",
wordList =
{"des","der","dfr","dgt","dfs"}
Output:
[ [ “der”, “dfr”, “dfs” ], [
“der”, “des”, “dfs”] ]
Explanation:
The length of the smallest
transformation sequence here is
3.
Following are the only two
shortest ways to get to the
targetWord from the startWord :
"der" -> ( replace ‘r’ by ‘s’ )
-> "des" -> ( replace ‘e’ by ‘f’
) -> "dfs".
"der" -> ( replace ‘e’ by ‘f’ )
-> "dfr" -> ( replace ‘r’ by ‘s’
) -> "dfs".

Example 2:
Input:
startWord = "gedk", targetWord=
"geek"
wordList = {"geek", "gefk"}
Output:
[ [ “gedk”, “geek” ] ]
```

Count Subarray sum Equals K

Binary Tree Representation in Java

Accolite Digital Amazon Arcesium arrays Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing infosys inorder Interview Experience Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCS TCS CODEVITA TCS Ninja

```
Explanation:
The length of the smallest
transformation sequence here is
2.
Following is the only shortest
way to get to the targetWord
from the startWord :
"gedk" -> ( replace ‘d’ by ‘e’ )
-> "geek".
```

Disclaimer: *Don’t jump directly to the solution, try it out yourself first.*

Note: In case any image/dry run is not clear please refer to the video attached at the bottom.

Intuition:

The intuition behind using the BFS traversal technique for these kinds of problems is that if we notice carefully, we go on replacing the characters one by one which seems just like we’re moving level-wise in order to reach the destination i.e. the targetWord. Here, in the example below we can notice there are two possible paths in order to reach the targetWord.

Contrary to the previous problem, here we do not stop the traversal on the first occurrence of the targetWord, but rather continue it for as many occurrences of the word as possible as we need **all** the shortest possible sequences in order to reach the destination word. The only trick here is that we **do not** have to delete a particular word immediately from the wordList even if during the replacement of characters it matches with the transformed word. Instead, we delete it after the traversal for a particular level when completed which allows us to explore all possible paths. This allows us to discover multiple sequences in order to reach the targetWord involving similar words.

From the above figure, we can configure that there can be 2 shortest possible sequences in order to reach the word ‘cog’.

Approach:

This problem uses the BFS traversal technique for finding out all the shortest possible transformation sequences by exploring all possible ways in which we can reach the targetWord.

Initial configuration:

- **Queue:** Define a queue data structure to store the level-wise formed sequences. The queue will be storing a List of strings, which will be representing the path till now. The last word in the list will be the last converted word.
- **Hash set:** Create a hash set to store the elements present in the word list to carry out the search and delete operations in O(1) time.
- **Vector:** Define a 1D vector 'usedOnLevel' to store the words which are currently being used for transformation on a particular level and a 2D vector 'ans' for storing all the shortest sequences of transformation.

The Algorithm for this problem involves the following steps:

- Firstly, we start by creating a hash set to store all the elements present in the wordList which would make the search

and delete operations faster for us to implement.

- Next, we create a Queue data structure for storing the successive sequences/ path in the form of a vector which on transformation would lead us to the target word.
- Now, we add the startWord to the queue as a List and also push it into the usedOnLevel vector to denote that this word is currently being used for transformation in this particular level.
- Pop the first element out of the queue and carry out the BFS traversal, where for each word that popped out from the back of the sequence present at the top of the queue, we check for all of its characters by replacing them with 'a' – 'z' if they are present in the wordList or not. In case a word is present in the wordList, we simply first push it onto the usedOnLevel vector and do not delete it from the wordList immediately.
- Now, push that word into the vector containing the previous sequence and add it to the queue. So we will get a new path, but we need to explore other paths as well, so pop the word out of the list to explore other paths.
- After completion of traversal on a particular level, we can now delete all the words that were currently being used on that level from the usedOnLevel vector

which ensures that these words won't be used again in the future, as using them in the later stages will mean that it won't be the shortest path anymore.

- If at any point in time we find out that the last word in the sequence present at the top of the queue is equal to the target word, we simply push the sequence into the resultant vector if the resultant vector 'ans' is empty.
- If the vector is not empty, we check if the current sequence length is equal to the first element added in the ans vector or not. This has to be checked because we need the shortest possible transformation sequences.
- In case, there is no transformation sequence possible, we return an empty 2D vector.

Note: If you wish to see the dry run of the above approach, you can watch the video attached to this article.

Code:

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution
{
public:
    vector<vector<string>> findSeque

    {
```

```
// Push all values of wordLi
// to make deletion from it
unordered_set<string> st(wor

// Creating a queue ds which
// required to reach the tar
queue<vector<string>> q;

// BFS traversal with pushin
// when after a transformati

q.push({beginWord});

// A vector defined to store
// on a level during BFS.
vector<string> usedOnLevel;
usedOnLevel.push_back(beginW
int level = 0;

// A vector to store the res
vector<vector<string>> ans;
while (!q.empty())
{
    vector<string> vec = q.f
    q.pop();

    // Now, erase all words
    // used in the previous
    if (vec.size() > level)
    {
        level++;
        for (auto it : usedO
        {
            st.erase(it);
        }
    }

    string word = vec.back()

    // store the answers if
    if (word == endWord)
    {
        // the first sequenc
        if (ans.size() == 0)
        {
            ans.push_back(ve
```

```
                    }
                    else if (ans[0].size
                    {
                        ans.push_back(ve
                    }
                }
                for (int i = 0; i < word
                {
                    // Now, replace each
                    // from a-z then che
                    char original = word
                    for (char c = 'a'; c
                    {
                        word[i] = c;
                        if (st.count(wor
                        {
                            // Check if
                            // push the
                            vec.push_bac
                            q.push(vec);
                            // mark as v
                            usedOnLevel.
                            vec.pop_back
                        }
                    }
                    word[i] = original;
                }
            }
            return ans;
        }
    };

    // A comparator function to sort the
    bool comp(vector<string> a, vector<s
    {
        string x = "", y = "";
        for (string i : a)
            x += i;
        for (string i : b)
            y += i;

        return x < y;
    }

    int main()
    {
```

```
    vector<string> wordList = {"des"
    string startWord = "der", target
    Solution obj;
    vector<vector<string>> ans = obj

    // If no transformation sequence
    if (ans.size() == 0)
        cout << -1 << endl;
    else
    {
        sort(ans.begin(), ans.end(),
        for (int i = 0; i < ans.size
        {
            for (int j = 0; j < ans[
            {
                cout << ans[i][j] <<
            }
            cout << endl;
        }
    }

    return 0;
}
```

Output:

der des dfs

der dfr dfs

Time Complexity and Space Complexity: It cannot be predicted for this particular algorithm because there can be multiple sequences of transformation from startWord to targetWord depending upon the example, so we cannot define a fixed range of time or space in which this program would run for all the test cases.

Note: This approach/code will give TLE when solved on the Leetcode platform due to the strict time constraints being put up there. So, you need to optimize it to a greater extent in order to pass all the test cases for LeetCode.For the optimized approach to this question please check out the next video.

Java Code

```
import java.util.*;
import java.lang.*;
import java.io.*;

// A comparator function to sort the
class comp implements Comparator < A

    public int compare(ArrayList < S
        String x = "";
        String y = "";
        for (int i = 0; i < a.size()
            x += a.get(i);
        for (int i = 0; i < b.size()
            y += b.get(i);
        return x.compareTo(y);
    }
}

public class Main {

    public static void main(String[]
        String startWord = "der", ta
        String[] wordList = {
            "des",
            "der",
            "dfr",
            "dgt",
            "dfs"
        };

        Solution obj = new Solution(
        ArrayList < ArrayList < Stri
```

```
            // If no transformation sequ
            if (ans.size() == 0)
                System.out.println(-1);
            else {

                Collections.sort(ans, ne
                for (int i = 0; i < ans.
                    for (int j = 0; j <
                        System.out.print
                    }
                    System.out.println()
                }
            }
        }
    }

    class Solution {
        public ArrayList < ArrayList < S
            String[] wordList) {

            // Push all values of wordLi
            // to make deletion from it
            Set < String > st = new Hash
            int len = wordList.length;
            for (int i = 0; i < len; i++
                st.add(wordList[i]);
            }

            // Creating a queue ds which
            // required to reach the tar
            Queue < ArrayList < String >
            ArrayList < String > ls = ne
            ls.add(startWord);
            q.add(ls);
            ArrayList < String > usedOnL
            usedOnLevel.add(startWord);
            int level = 0;

            // A vector to store the res
            ArrayList < ArrayList < Stri
            int cnt = 0;

            // BFS traversal with pushin
            // when after a transformati
            while (!q.isEmpty()) {
                cnt++;
```

```
                ArrayList < String > vec
                q.remove();

                 // Now, erase all words
                 // used in the previous
                 if (vec.size() > level)
                    level++;
                    for (String it: used
                        st.remove(it);
                    }
                }

                String word = vec.get(ve

                // store the answers if
                if (word.equals(targetWo
                    // the first sequenc
                    if (ans.size() == 0)
                        ans.add(vec);
                    } else if (ans.get(0
                        ans.add(vec);
                    }
                }
                for (int i = 0; i < word

                    // Now, replace each
                    // from a-z then che
                    for (char c = 'a'; c
                        char replacedCha
                        replacedCharArra
                        String replacedW
                        if (st.contains(
                            vec.add(repl
                            // Java work
                            // otherwise
                            // remove fr
                            ArrayList <
                            q.add(temp);
                            // mark as v
                            usedOnLevel.
                            vec.remove(v
                        }
                    }

                }
            }
```

```
                return ans;
            }
        }
```

Output:

der des dfs

der dfr dfs

Time Complexity and Space Complexity: It cannot be predicted for this particular algorithm because there can be multiple sequences of transformation from startWord to targetWord depending upon the example, so we cannot define a fixed range of time or space in which this program would run for all the test cases.

Note: This approach/code will give TLE when solved on the Leetcode platform due to the strict time constraints being put up there. So, you need to optimize it to a greater extent in order to pass all the test cases for LeetCode.For the optimized approach to this question please check out the next video.

> Special thanks to **Priyanshi Goel** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com

« Previous Post
Alien Dictionary – Topological Sort: G-26

Next Post »
Word Ladder-II (Optimised Approach) G-31

Load Comments