October 21, 2022 ▪ Graph

Word Ladder – I : G-29

Given are the two distinct words **startWord** and **targetWord**, and a list denoting **wordList** of unique words of equal lengths. Find the length of the shortest transformation sequence from startWord to targetWord.

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

Note: If there's no possible way to transform the sequence from startWord to targetWord

Search

Search

Recent Posts

2023 – Striver's SDE Sheet Challenge

Top Array Interview Questions – Structured Path with Video Solutions

Longest Subarray with sum K | [Postives and Negatives]

return **0**.

Example 1:

```
Input:
wordList =
{"des","der","dfr","dgt","dfs"}
startWord = "der", targetWord =
"dfs"
Output:
3
Explanation:
The length of the smallest
transformation sequence from
"der" to
"dfs" is 3 i.e. "der" ->
(replace ‘e’ by ‘f’) -> "dfr" ->
(replace ‘r’ by ‘s’)  -> "dfs".
So, it takes 3 different strings
for us to reach the targetWord.
Each of these strings are
present in the wordList.
```

Example 2:

```
Input:
wordList = {"geek", "gefk"}
startWord = "gedk", targetWord=
"geek"
Output:
2
Explanation:
The length of the smallest
transformation sequence
from "gedk" to "geek" is 2 i.e.
"gedk" -> (replace ‘d’ by ‘e’) -
> "geek"
So, it takes 2 different strings
for us to reach the targetWord.
```

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

```
Each of these strings are
present in the wordList.
```


Solution

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

Problem Link

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

Approach:

Let's first understand the Brute force approach to this problem. In Brute force, we just simply replace the startingWord **character by character** and then check whether the transformed word is present in the wordList. If a word is present in the wordList, we try replacing another character in that word by again following similar steps as above, in order to attain the targetWord. We do this for all the characters in the startWord and then eventually return the minimum length of transforming the startWord to targetWord.

Now, to make this algorithm a little less time-consuming and easier, we implement this using a BFS traversal technique.

Initial configuration:

- **Queue:** Define a queue data structure to store the BFS traversal.
- **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.

The Algorithm involves the following steps:

- Firstly, we start by creating a queue data structure in order to store the word and the length of the sequence to reach that word as a pair. We store them in form of {word, steps}.
- Then, we push the startWord into the queue with length as ‘1’ indicating that

this is the word from which the sequence needs to start from.

- We now create a hash set wherein, we put all the elements in wordList to keep a check on if we've visited that word before or not. In order to mark a word as visited here, we simply delete the word from the hash set. There is no point in visiting someone multiple times during the algorithm.
- Now, we pop the first element out of the queue and carry out the BFS traversal where, for each word popped out of the queue, we try to replace every character with 'a' – 'z', and we get a transformed word. We check if the transformed word is present in the wordList or not.
- If the word is present, we push it in the queue and increase the count of the sequence by 1 and if not, we simply move on to replacing the original character with the next character.
- Remember, we also need to delete the word from the wordList if it matches with the transformed word to ensure that we do not reach the same point again in the transformation which would only increase our sequence length.
- Now, we pop the next element out of the queue ds and if at any point in time, the transformed word becomes the same as the targetWord, we return the count of the steps taken to reach that word. Here,

we're only concerned about the first occurrence of the targetWord because after that it would only lead to an increase in the sequence length which is for sure not **minimum.**

- If a transformation sequence is not possible, return 0.

Intuition:

The intuition behind using the BFS traversal technique for this particular problem 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.

In level-order traversal, when we reach the destination, we stop the traversal. Similar to that, when we reach our targetWord, we terminate the algorithm and return the counted steps.

We no longer continue the algorithm after that because that would only **increase** the step count to reach the targetWord.

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:
    int wordLadderLength(string star
                         vector<stri
    {
    // Creating a queue ds of type {
        queue<pair<string, int>> q;

        // BFS traversal with pushin
        // when after a transformati
        q.push({startWord, 1});

        // Push all values of wordLi
        // to make deletion from it
        unordered_set<string> st(wor
        st.erase(startWord);
        while (!q.empty())
        {
            string word = q.front().
            int steps = q.front().se
            q.pop();

            // we return the steps a
            // the first occurence o
            if (word == targetWord)
                return steps;

            for (int i = 0; i < word
```

```
                {
                    // Now, replace each
                    // from a-z then che
                    char original = word
                    for (char ch = 'a';
                    {
                        word[i] = ch;
                        // check if it e
                        if (st.find(word
                        {
                            st.erase(wor
                            q.push({word
                        }
                    }
                    word[i] = original;
                }
            }
            // If there is no transforma
            return 0;
        }
    };

    int main()
    {

        vector<string> wordList = {"des"
        string startWord = "der", target

        Solution obj;

        int ans = obj.wordLadderLength(s

        cout << ans;
        cout << endl;
        return 0;
    }
```

Output:

3

Time Complexity: O(N * M * 26)

Where N = size of wordList Array and M = word length of words present in the wordList..

Note that, hashing operations in an **unordered set** takes O(1) time, but if you want to use **set** here, then the time complexity would increase by a factor of log(N) as hashing operations in a set take O(log(N)) time.

Space Complexity: O(N) { for creating an unordered set and copying all values from wordList into it }

Where N = size of wordList Array.

Java Code

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

class Main {

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

        Solution obj = new Solution(
        int ans = obj.wordLadderLeng

        System.out.print(ans);

        System.out.println();
```

```
        }
    }

    class Pair {
        String first;
        int second;
        Pair(String _first, int _second)
            this.first = _first;
            this.second = _second;
        }
    }
    class Solution {
        public int wordLadderLength(Stri
            // Creating a queue ds of ty
            Queue < Pair > q = new Linke

            // BFS traversal with pushin
            // when after a transformati
            q.add(new Pair(startWord, 1)

            // 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]);
            }
            st.remove(startWord);
            while (!q.isEmpty()) {
                String word = q.peek().f
                int steps = q.peek().sec
                q.remove();

                // we return the steps a
                // the first occurence o
                if (word.equals(targetWo

                // Now, replace each cha
                // from a-z then check i
                for (int i = 0; i < word
                    for (char ch = 'a';
                        char replacedCha
                        replacedCharArra
                        String replacedW

                        // check if it e
```

```
                    if (st.contains(
                        st.remove(re
                        q.add(new Pa
                    }
                }

            }
        }
        // If there is no transforma
        return 0;
    }
}
```

Output:

3

Time Complexity: O(N * M * 26)

Where N = size of wordList Array and M = word length of words present in the wordList..

Note that, hashing operations in an **unordered set** takes O(1) time, but if you want to use **set** here, then the time complexity would increase by a factor of log(N) as hashing operations in a set take O(log(N)) time.

Space Complexity: O(N) { for creating an unordered set and copying all values from wordList into it }

Where N = size of wordList Array.

> 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
Detect a Cycle in Directed Graph | Topological Sort | Kahn's Algorithm | G-23

Next Post »
Minimum cost to cut the stick| (DP-50)

Load Comments