HARD

Word Ladder-I

Intuition

Given : WordList , StartWord

Find length of shortest transformation sequence from StartWord to TargetWord

Note :

- Word only consist of lowercase characters
- Only 1 letter can be changed in each transformation
- Each word must exist in WordList including target word
- StartWord may or may not be part of the WordList

Eg.

beginWord = “hit”
endWord = “cog”

WordList = [hot, dot, dog, lot, log, cog]

What we can do is we can change any one characters of hit

Hit -> hot : hot is in wordlist hence can be changed
Dot -> dog -> cog : end word hence we would require 4 transformations required and the transformation sequence length is 5

If not possible to perform then return 0

Hit ?

HIT -> HOT

H can be change
I can be changed
T can be changed

H -> AIT
H -> BIT
H -> CIT
H -> …….IT

H -> A...Z
I -> A….Z
T -> A…Z

Start with particular word and change it character by character and check if its present in the word list, then count the minimum length of the transformation sequence

Eg.

HIT : level 1

AIT , BIT , CIT ……. ZIT or HAT, HBT……..HOT (USED) or HIA, HIB, HIC ….. HIZ:Level2

.
.
.
.
COG…… : Level 5 (return)

First time whenever we encounter the first word we will return level

Queue = [HIT,1]

POP(HIT)
H : A
Is AIT in WordListSet
Is BIT in WordListSet
.
.

HOT in WorldListSet : q.push(HOT, 2) | remove HOT from word_list set
.
.
HIZ

Iter-2

POP(HOT)

AOT
BOT
COT
DOT - push(DOT,2) | Remove DOT from word list set
.

.
LOT - push(LOT,3) } Remove LOT from word list set
.
.
ZOT

Iter-3

AOT
BOT
.
.
.
ZOT
DAT
.
.
.
DZT
.
.
DOG - push(DOG,4)
.
.
DAZ

Continue
.
.
COG - word == COG return level 5

Approach

- Declare an empty queue storing the pair of the word and level
- Insert the starting word with level as level 1 in the queue
- Create an unordered set having the words given in the word list
- Erase the starting word from the set
- Traverse until the queue becomes empty :
    - Extract the first element of the queue
    - Pop the first element of the queue
    - Check if the word is same as the target :
        - Return the level of the word

- Transform the word by replacing every character of the english alphabet at every place of the word alphabets :
    - If the transformed word is in the set :
        - Erase the word from the set
        - Push the transformed word into the queue with level+1
- Return 0 because in this case the word cannot be formed

Function Code

```
int wordLadderLength(string startWord, string targetWord, vector<string>&
wordList) {
        // creating an empty queue
        queue<pair<string,int>>q;
        // inserting the start word with level count as 1 in the queue
        q.push({startWord,1});
        // creating a set to store the strings of the word_list
        unordered_set<string> st(wordList.begin(),wordList.end());
        // erasing the start word from the set word list
        st.erase(startWord);
        // traversing until the queue becomes empty
        while(!q.empty())
        {
            // extracting the first word and its level from the queue
            string word = q.front().first;
            int steps = q.front().second;
            // popping the first element of the queue
            q.pop();
            // if the word is same as target then return the level/steps
            if(word==targetWord)return steps;
            // traverse for all the word characters
            for(int i=0;i<word.size();i++)
            {
                // storing the original word character
                char original = word[i];
                // traversing for all the element of english alphabets and
try changing the word with the them
                for(char c = 'a'; c<='z';c++)
                {
                    word[i] = c;
                    // checking if the word is present in the word list set
                    if(st.find(word) !=st.end())
                    {
                        // erasing the word from word list set
```

```
                        st.erase(word);
                        // pushing the word with increased level into the
queue
                        q.push({word,steps+1});
                    }

                }
                // changing word back to its original form
                word[i] = original;
            }
        }
        // return 0 because in this case the word was not found
        return 0;
    }
```

Time Complexity

N * word.length * 26 * log(N)