HARD

Word Ladder - II

Intuition

Begin = “bat”
End = “coz”

wordList = [pat, bot, pot, poz, coz]

Possible shortest transformations :

- Bat -> pat -> pot -> poz -> coz
- Bat -> bot -> pot -> poz -> coz

We have to return both transformation in any order

We took a queue data structure with sequence
[bat]

pop(bat)
Change b-> pat | pot |

Now queue will look like

[bat, bot]
[bat, pot]

Now delete pat and pot from set

[bat ,pat]

Pat -> pot

[bat, pat, pot]
[bat, bot]
[bat, pat]

[bat, bot, pot]
[bat, pat, pot]
[bat, bot]
[bat, pat]

Continue this across all levels …….

Finally it will look like

// we don't go further
[bat, bot, pot, poz, coz]
[bat, pat, pot, poz, coz]
[bat, bot, pot,poz]
[bat, pat, pot,poz]
[bat, bot, pot]
[bat, pat, pot]
[bat, bot]
[bat, pat]

Return [[bat, bot, pot, poz, coz],[bat, pat, pot, poz, coz]]

Approach

Function Code

```
vector<vector<string>> findSequences(string beginWord, string endWord,
vector<string>& wordList) {
// creating a word list set
    unordered_set<string> s(wordList.begin(), wordList.end());
// creating a queue having the vectors of string transformation sequence
    queue<vector<string>> q;
// pushing first element into the queue
    q.push({beginWord});
// unordered set defined to store the words being currently used on a
// level during BFS
    unordered_set<string> usedOnLevel;
// pushing first word into the UsedOnLevel
    usedOnLevel.insert(beginWord);
// defining a level variable having status 0
    int level = 0;
// creating a result/ans vector containing the conversion sequence
    vector<vector<string>> ans;

// traversing until the queue becomes empty
    while (!q.empty()) {
// extracting the first vector and then popping queue
        vector<string> vec = q.front();
        q.pop();
        // if vector’s size is greater than current level then
        if (vec.size() > level) {
// increasing the level as we require more transformations
            level += 1;
// traversing for words in used on level and erasing them from set
            for (const string& word : usedOnLevel) {
                s.erase(word);
            }
// clearing used on level set
            usedOnLevel.clear();
        }

// last word extraction as it is the matcher
        string word = vec.back();
// checking if word is same as target
        if (word == endWord) {
// check if it's the first vector then insert it directly
            if (ans.empty()) {
                ans.push_back(vec);
            }

// else check if the vector size is same then we insert it into the result
vector
else if (ans[0].size() == vec.size()) {
                ans.push_back(vec);
            }
        }
        // traversing for the conversion of last word
        for (int i = 0; i < word.size(); i++) {
            char original = word[i];
            for (char c = 'a'; c <= 'z'; c++) {
                word[i] = c;
                // checking if the word is in the set
                if (s.count(word) > 0) {
                   // pushing the word into the vector
                    vector<string> newVec = vec;
                    newVec.push_back(word);
// pushing the vector into the queue
                    q.push(newVec);
// insert the word into used on level set
                    usedOnLevel.insert(word);
                }
            }
            word[i] = original;
        }
    }
    // return the ans vector
    return ans;
    }
```