**Word Ladder**

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that only one letter can be changed at a time and each intermediate word must exist in the dictionary. For example, given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

#include <iostream>

#include <string>

#include <unordered\_set>

#include <queue>

using namespace std;

class WordNode {

public:

string word;

int numSteps;

WordNode(string word, int numSteps) {

this->word = word;

this->numSteps = numSteps;

}

};

int ladderLength(string beginWord, string endWord, unordered\_set<string> dictionary) {

queue<WordNode> q;

q.push(WordNode(beginWord, 1));

while (!q.empty()) {

WordNode top = q.front();

q.pop();

if (top.word == endWord) {

return top.numSteps;

}

for (string::size\_type i = 0; i < top.word.size(); ++i) {

for (char c = 'a'; c <= 'z'; ++c) {

char temp = top.word[i];

if (c != top.word[i]) {

top.word[i] = c;

}

if (dictionary.find(top.word) != dictionary.end()) {

q.push(WordNode(top.word, top.numSteps + 1));

// To remove cycles, remove word

dictionary.erase(top.word);

}

top.word[i] = temp;

}

}

}

return 0;

}

int main() {

string start = "hit";

string end = "cog";

unordered\_set<string> dictionary = { "hot", "dot", "dog", "lot", "log", "cog" };

cout << "Word Ladder Length: " << ladderLength(start, end, dictionary) << endl;

system("PAUSE");

}