Word Ladder (Length of shortest chain to reach a target word)

Difficulty Level : Medium ● Last Updated : 24 Feb, 2022

Given a dictionary, and two words 'start' and 'target' (both of same length). Find length of the smallest chain from 'start' to 'target' if it exists, such that adjacent words in the chain only differ by one character and each word in the chain is a valid word i.e., it exists in the dictionary. It may be assumed that the 'target' word exists in dictionary and length of all dictionary words is same.

Example:

Input: *Dictionary = {POON, PLEE, SAME, POIE, PLEA, PLIE, POIN}, start = TOON, target = PLEA*
Output: *7*
Explanation: *TOON – POON – POIN – POIE – PLIE – PLEE – PLEA*

Input: *Dictionary = {ABCD, EBAD, EBCD, XYZA}, start = ABCV, target = EBAD*
Output: *4*

Approach: The idea to solve the problem is to use BFS. To find the shortest path through BFS, start from the **start** word and push it in a queue. And once the **target** is found for the first time, then return that level of BFS traversal. In each step of BFS one can get all the words that can be formed using that many steps. So whenever the **target** word is found for the first time that will be the length of the shortest chain of words.

1. Start from the given **start** word.
2. Push the word in the queue
3. Run a loop until the queue is empty
4. Traverse all words that adjacent (differ by one character) to it and push the word in a queue (for BFS)
5. Keep doing so until we find the **target** word or we have traversed all words.

Below are the implementations of the above idea.

C++

```
// C++ program to find length
// of the shortest chain
// transformation from source
// to target
#include <bits/stdc++.h>
using namespace std;

// Returns length of shortest chain
// to reach 'target' from 'start'
// using minimum number of adjacent
// moves.  D is dictionary
int shortestChainLen(
string start, string target,
set<string>& D)
{

    if(start == target)
      return 0;

    // If the target string is not
    // present in the dictionary
```

```
// To store the current chain length
// and the length of the words
int level = 0, wordlength = start.size();

// Push the starting word into the queue
queue<string> Q;
Q.push(start);

// While the queue is non-empty
while (!Q.empty()) {

    // Increment the chain length
    ++level;

    // Current size of the queue
    int sizeofQ = Q.size();

    // Since the queue is being updated while
    // it is being traversed so only the
    // elements which were already present
    // in the queue before the start of this
    // loop will be traversed for now
    for (int i = 0; i < sizeofQ; ++i) {

        // Remove the first word from the queue
        string word = Q.front();
        Q.pop();

        // For every character of the word
        for (int pos = 0; pos < wordlength; ++pos) {

            // Retain the original character
            // at the current position
            char orig_char = word[pos];

            // Replace the current character with
            // every possible lowercase alphabet
            for (char c = 'a'; c <= 'z'; ++c) {
                word[pos] = c;

                // If the new word is equal
                // to the target word
                if (word == target)
                    return level + 1;

                // Remove the word from the set
                // if it is found in it
```

```
                    // And push the newly generated word
                    // which will be a part of the chain
                    Q.push(word);
                }

                // Restore the original character
                // at the current position
                word[pos] = orig_char;
            }
        }
    }

    return 0;
}

// Driver program
int main()
{
    // make dictionary
    set<string> D;
    D.insert("poon");
    D.insert("plee");
    D.insert("same");
    D.insert("poie");
    D.insert("plie");
    D.insert("poin");
    D.insert("plea");
    string start = "toon";
    string target = "plea";
    cout << "Length of shortest chain is: "
         << shortestChainLen(start, target, D);
    return 0;
}
```

Java

```
// Java program to find length
// of the shortest chain
// transformation from source
// to target
import java.util.*;

class GFG
{
```

```
// using minimum number of adjacent moves.
// D is dictionary
static int shortestChainLen(String start,
                            String target,
                            Set<String> D)
{

     if(start == target)
      return 0;
    // If the target String is not
    // present in the dictionary
    if (!D.contains(target))
        return 0;

    // To store the current chain length
    // and the length of the words
    int level = 0, wordlength = start.length();

    // Push the starting word into the queue
    Queue<String> Q = new LinkedList<>();
    Q.add(start);

    // While the queue is non-empty
    while (!Q.isEmpty())
    {

        // Increment the chain length
        ++level;

        // Current size of the queue
        int sizeofQ = Q.size();

        // Since the queue is being updated while
        // it is being traversed so only the
        // elements which were already present
        // in the queue before the start of this
        // loop will be traversed for now
        for (int i = 0; i < sizeofQ; ++i)
        {

            // Remove the first word from the queue
            char []word = Q.peek().toCharArray();
            Q.remove();

            // For every character of the word
            for (int pos = 0; pos < wordlength; ++pos)
            {
```

```
                char orig_char = word[pos];

                // Replace the current character with
                // every possible lowercase alphabet
                for (char c = 'a'; c <= 'z'; ++c)
                {
                    word[pos] = c;

                    // If the new word is equal
                    // to the target word
                    if (String.valueOf(word).equals(target))
                        return level + 1;

                    // Remove the word from the set
                    // if it is found in it
                    if (!D.contains(String.valueOf(word)))
                        continue;
                    D.remove(String.valueOf(word));

                    // And push the newly generated word
                    // which will be a part of the chain
                    Q.add(String.valueOf(word));
                }

                // Restore the original character
                // at the current position
                word[pos] = orig_char;
            }
        }
    }

    return 0;
}

// Driver code
public static void main(String[] args)
{
    // make dictionary
    Set<String> D = new HashSet<String>();
    D.add("poon");
    D.add("plee");
    D.add("same");
    D.add("poie");
    D.add("plie");
    D.add("poin");
    D.add("plea");
    String start = "toon";
```

```
}
}

// This code is contributed by PrinciRaj1992
```

Python3

```
# Python3 program to find length of the
# shortest chain transformation from source
# to target
from collections import deque

# Returns length of shortest chain
# to reach 'target' from 'start'
# using minimum number of adjacent
# moves. D is dictionary
def shortestChainLen(start, target, D):

    if start == target:
      return 0
    # If the target is not
    # present in the dictionary
    if target not in D:
        return 0

    # To store the current chain length
    # and the length of the words
    level, wordlength = 0, len(start)

    # Push the starting word into the queue
    Q =  deque()
    Q.append(start)

    # While the queue is non-empty
    while (len(Q) > 0):

        # Increment the chain length
        level += 1

        # Current size of the queue
        sizeofQ = len(Q)

        # Since the queue is being updated while
        # it is being traversed so only the
        # elements which were already present
```

```
            # Remove the first word from the queue
            word = [j for j in Q.popleft()]
            #Q.pop()

            # For every character of the word
            for pos in range(wordlength):

                # Retain the original character
                # at the current position
                orig_char = word[pos]

                # Replace the current character with
                # every possible lowercase alphabet
                for c in range(ord('a'), ord('z')+1):
                    word[pos] = chr(c)

                    # If the new word is equal
                    # to the target word
                    if ("".join(word) == target):
                        return level + 1

                    # Remove the word from the set
                    # if it is found in it
                    if ("".join(word) not in D):
                        continue

                    del D["".join(word)]

                    # And push the newly generated word
                    # which will be a part of the chain
                    Q.append("".join(word))

                # Restore the original character
                # at the current position
                word[pos] = orig_char

    return 0

# Driver code
if __name__ == '__main__':

    # Make dictionary
    D = {}
    D["poon"] = 1
    D["plee"] = 1
    D["same"] = 1
```

```
    D["plea"] = 1
    start = "toon"
    target = "plea"

    print("Length of shortest chain is: ",
    shortestChainLen(start, target, D))

# This code is contributed by mohit kumar 29
```

C#

```
// C# program to find length of the shortest chain
// transformation from source to target
using System;
using System.Collections.Generic;

class GFG
{

// Returns length of shortest chain
// to reach 'target' from 'start'
// using minimum number of adjacent moves.
// D is dictionary
static int shortestChainLen(String start,
                            String target,
                            HashSet<String> D)
{

     if(start == target)
       return 0;
    // If the target String is not
    // present in the dictionary
    if (!D.Contains(target))
        return 0;

    // To store the current chain length
    // and the length of the words
    int level = 0, wordlength = start.Length;

    // Push the starting word into the queue
    List<String> Q = new List<String>();
    Q.Add(start);

    // While the queue is non-empty
    while (Q.Count != 0)
```

```
        ++level;

        // Current size of the queue
        int sizeofQ = Q.Count;

        // Since the queue is being updated while
        // it is being traversed so only the
        // elements which were already present
        // in the queue before the start of this
        // loop will be traversed for now
        for (int i = 0; i < sizeofQ; ++i)
        {

            // Remove the first word from the queue
            char []word = Q[0].ToCharArray();
            Q.RemoveAt(0);

            // For every character of the word
            for (int pos = 0; pos < wordlength; ++pos)
            {

                // Retain the original character
                // at the current position
                char orig_char = word[pos];

                // Replace the current character with
                // every possible lowercase alphabet
                for (char c = 'a'; c <= 'z'; ++c)
                {
                    word[pos] = c;

                    // If the new word is equal
                    // to the target word
                    if (String.Join("", word).Equals(target))
                        return level + 1;

                    // Remove the word from the set
                    // if it is found in it
                    if (!D.Contains(String.Join("", word)))
                        continue;
                    D.Remove(String.Join("", word));

                    // And push the newly generated word
                    // which will be a part of the chain
                    Q.Add(String.Join("", word));
                }
```

```
            }
        }
    }
    return 0;
}

// Driver code
public static void Main(String[] args)
{
    // make dictionary
    HashSet<String> D = new HashSet<String>();
    D.Add("poon");
    D.Add("plee");
    D.Add("same");
    D.Add("poie");
    D.Add("plie");
    D.Add("poin");
    D.Add("plea");
    String start = "toon";
    String target = "plea";
    Console.Write("Length of shortest chain is: "
        + shortestChainLen(start, target, D));
}
}

// This code is contributed by PrinciRaj1992
```

Javascript

```
<script>
// Javascript program to find length
// of the shortest chain
// transformation from source
// to target

// Returns length of shortest chain
// to reach 'target' from 'start'
// using minimum number of adjacent moves.
// D is dictionary
    function shortestChainLen(start,target,D)
    {
        if(start == target)
      return 0;

    // If the target String is not
```

```
// To store the current chain length
// and the length of the words
let level = 0, wordlength = start.length;

// Push the starting word into the queue
let Q = [];
Q.push(start);

// While the queue is non-empty
while (Q.length != 0)
{

    // Increment the chain length
    ++level;

    // Current size of the queue
    let sizeofQ = Q.length;

    // Since the queue is being updated while
    // it is being traversed so only the
    // elements which were already present
    // in the queue before the start of this
    // loop will be traversed for now
    for (let i = 0; i < sizeofQ; ++i)
    {

        // Remove the first word from the queue
        let word = Q[0].split("");
        Q.shift();

        // For every character of the word
        for (let pos = 0; pos < wordlength; ++pos)
        {

            // Retain the original character
            // at the current position
            let orig_char = word[pos];

            // Replace the current character with
            // every possible lowercase alphabet
            for (let c = 'a'.charCodeAt(0); c <= 'z'.charCodeAt(0); ++c)
            {
                word[pos] = String.fromCharCode(c);

                // If the new word is equal
                // to the target word
```

```
                    // Remove the word from the set
                    // if it is found in it
                    if (!D.has(word.join("")))
                        continue;
                    D.delete(word.join(""));

                    // And push the newly generated word
                    // which will be a part of the chain
                    Q.push(word.join(""));
                }

                // Restore the original character
                // at the current position
                word[pos] = orig_char;
            }
        }
    }

    return 0;
    }

    // Driver code
    // make dictionary
    let D = new Set();
    D.add("poon");
    D.add("plee");
    D.add("same");
    D.add("poie");
    D.add("plie");
    D.add("poin");
    D.add("plea");
    let start = "toon";
    let target = "plea";
    document.write("Length of shortest chain is: "
        + shortestChainLen(start, target, D));

    // This code is contributed by unknown2108
</script>
```

Output

```
ength of shortest chain is: 7
```

Auxiliary Space: O(M * N)

Alternate Implementation: (Maintaining the mapping of the intermediate words and the original word):

Below is an alternative implementation to the above approach.

Here, in this approach, we find out all the intermediate words of the start word and the words in the given list of dictionary and maintain a map of the intermediate word and a vector of the original word (map<string, vector<string>>). For instance, for the word "POON", the intermediate words are "*OON" , "P*ON", "PO*N", "POO*". Then, we perform BFS traversal starting with the start word and push a pair of start word and the distance (pair(word, distance)) to the queue until we reach the target word. Then, the distance is our answer.

C++

```
// C++ program to find length
// of the shortest chain
// transformation from source
// to target
#include <bits/stdc++.h>
using namespace std;

// Returns length of shortest chain
// to reach 'target' from 'start'
// using minimum number of adjacent
// moves.  D is dictionary
int shortestChainLen(
string start, string target,
set<string>& D)
{

   if(start == target)
      return 0;

  // Map of intermediate words and
  // the list of original words
  map<string, vector<string>> umap;

  // Find all the intermediate
  // words for the start word
```

```
                        start.substr(i+1);
    umap[str].push_back(start);
  }

  // Find all the intermediate words for
  // the words in the given Set
  for(auto it = D.begin(); it != D.end(); it++)
  {
    string word = *it;
    for(int j = 0; j < word.size(); j++)
    {
      string str = word.substr(0,j) + "*" +
                          word.substr(j+1);
      umap[str].push_back(word);
    }
  }

  // Perform BFS and push (word, distance)
  queue<pair<string, int>> q;

  map<string, int> visited;

  q.push(make_pair(start,1));
  visited[start] = 1;

  // Traverse until queue is empty
  while(!q.empty())
  {
    pair<string, int> p = q.front();
    q.pop();

    string word = p.first;
    int dist = p.second;

    // If target word is found
    if(word == target)
    {
      return dist;
    }

    // Finding intermediate words for
    // the word in front of queue
    for(int i = 0; i < word.size(); i++)
    {
      string str = word.substr(0,i) + "*" +
                          word.substr(i+1);
```

```
        // If the word is not visited
        if(visited[vect[j]] == 0)
        {
          visited[vect[j]] = 1;
          q.push(make_pair(vect[j], dist + 1));
        }
      }
    }

  }

    return 0;
}

// Driver code
int main()
{
    // Make dictionary
    set<string> D;
    D.insert("poon");
    D.insert("plee");
    D.insert("same");
    D.insert("poie");
    D.insert("plie");
    D.insert("poin");
    D.insert("plea");
    string start = "toon";
    string target = "plea";
    cout << "Length of shortest chain is: "
         << shortestChainLen(start, target, D);
    return 0;
}
```

Java

```
// Java program to find length
// of the shortest chain
// transformation from source
// to target
import java.util.*;

lass GFG{
  static class pair
  {
    String first;
```

```
    this.first = first;
    this.second = second;
  }
}

// Returns length of shortest chain
// to reach 'target' from 'start'
// using minimum number of adjacent
// moves.  D is dictionary
static int shortestChainLen(
  String start, String target,
  HashSet<String> D)
{

  if(start == target)
    return 0;

  // Map of intermediate words and
  // the list of original words
  Map<String, Vector<String>> umap = new HashMap<>();

  // Find all the intermediate
  // words for the start word
  for(int i = 0; i < start.length(); i++)
  {
    String str = start.substring(0,i) + "*" +
      start.substring(i+1);
    Vector<String> s = umap.get(str);
    if(s==null)
      s = new Vector<String>();
    s.add(start);
    umap.put(str, s);
  }

  // Find all the intermediate words for
  // the words in the given Set
  for(String it : D)
  {
    String word = it;
    for(int j = 0; j < word.length(); j++)
    {
      String str = word.substring(0, j) + "*" +
        word.substring(j + 1);
      Vector<String> s = umap.get(str);
      if(s == null)
        s = new Vector<String>();
      s.add(word);
```

```
    // Perform BFS and push (word, distance)
    Queue<pair> q = new LinkedList<>();

    Map<String, Integer> visited = new HashMap<String, Integer>();

    q.add(new pair(start, 1));
    visited.put(start, 1);

    // Traverse until queue is empty
    while(!q.isEmpty())
    {
      pair p = q.peek();
      q.remove();

      String word = p.first;
      int dist = p.second;

      // If target word is found
      if(word == target)
      {
        return dist;
      }

      // Finding intermediate words for
      // the word in front of queue
      for(int i = 0; i < word.length(); i++)
      {
        String str = word.substring(0, i) + "*" +
          word.substring(i + 1);

        Vector<String> vect = umap.get(str);
        for(int j = 0; j < vect.size(); j++)
        {
          // If the word is not visited
          if(!visited.containsKey(vect.get(j)) )
          {
            visited.put(vect.get(j), 1);
            q.add(new pair(vect.get(j), dist + 1));
          }
        }
      }

    }

    return 0;
  }
```

```
  {

    // Make dictionary
    HashSet<String> D = new HashSet<String>();
    D.add("poon");
    D.add("plee");
    D.add("same");
    D.add("poie");
    D.add("plie");
    D.add("poin");
    D.add("plea");
    String start = "toon";
    String target = "plea";
    System.out.print("Length of shortest chain is: "
                     + shortestChainLen(start, target, D));
  }
}

// This code is contributed by 29AjayKumar
```

Output

```
Length of shortest chain is: 7
```

Time Complexity: O(N^2 * M), where M is the number of entries originally in the dictionary and N is the size of the string.

Auxiliary Space: O(M * N)

RECOMMENDED ARTICLES

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : princiraj1992, andrew1234, ryadav2, mohit kumar 29, Codilis, UtkarshPandey6, unknown2108, 29AjayKumar, nehakumariintern

Article Tags : BFS, Graph

Practice Tags : Graph, BFS

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

feedback@geeksforgeeks.org