leetcode-cn.com /problems/word-ladder/solution/yan-du-you-xian-bian-li-shuang-xiang-yan-du-y…

精选题解：广度优先遍历、双向广度优先遍历（Java） - 力扣（LeetCode）

10-12 minutes

一句话题解

- 无向图中两个顶点之间的最短路径的长度，可以通过广度优先遍历得到；
- 为什么 BFS 得到的路径最短？可以把起点和终点所在的路径拉直来看，两点之间线段最短；
- 已知目标顶点的情况下，可以分别从起点和目标顶点（终点）执行广度优先遍历，直到遍历的部分有交集，这是双向广度优先遍历的思想。

（参考代码应评论区用户要求进行了修改，2020 年 9 月 4 日。）

视频解题

视频时间线：建议倍速观看

- 读题、讲解注意事项：00:00 开始
- 分析示例 1 并讲解如何建图、BFS、双向 BFS 思路：02:58 开始
- 讲解 BFS 代码：11:41 开始
- 讲解双向 BFS 代码：18:54 开始

code：

vid:

uuid:

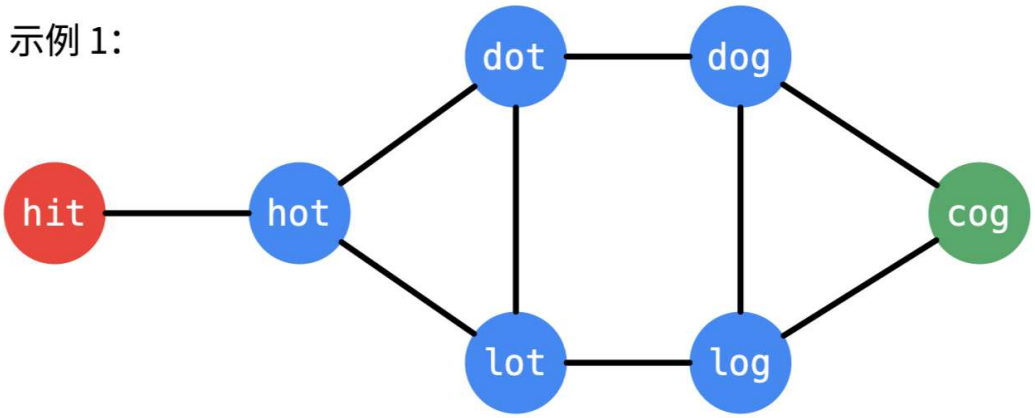
requestId:

播放时间：

提示信息

分析题意：

- 「转换」意即：两个单词对应位置只有一个字符不同，例如 "hit" 与 "hot"，这种转换是可以逆向的，因此，根据题目给出的单词列表，可以构建出一个无向（无权）图；



- 如果一开始就构建图，每一个单词都需要和除它以外的另外的单词进行比较，复杂度是 $O(N\text{wordLen})$，这里 N 是单词列表的长度；
- 为此，我们在遍历一开始，把所有的单词列表放进一个哈希表中，然后在遍历的时候构建图，每一次得到在单词列表里可以转换的单词，复杂度是 $O(26 \times \text{wordLen})$，借助哈希表，找到邻居与 N 无关；
- 使用 BFS 进行遍历，需要的辅助数据结构是：
  - 队列；
  - `visited` 集合。说明：可以直接在 `wordSet` (由 `wordList` 放进集合中得到) 里做删除。但更好的做法是新开一个哈希表，遍历过的字符串放进哈希表里。这种做法具有普遍意义。绝大多数在线测评系统和应用场景都不会在意空间开销。

方法一：广度优先遍历

参考代码 **1**：

- Java
- Python3

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Set;

public class Solution {

    public int ladderLength(String beginWord, String endWord,
List<String> wordList) {

        Set<String> wordSet = new HashSet<>(wordList);
        if (wordSet.size() == 0 || !wordSet.contains(endWord)) {
```

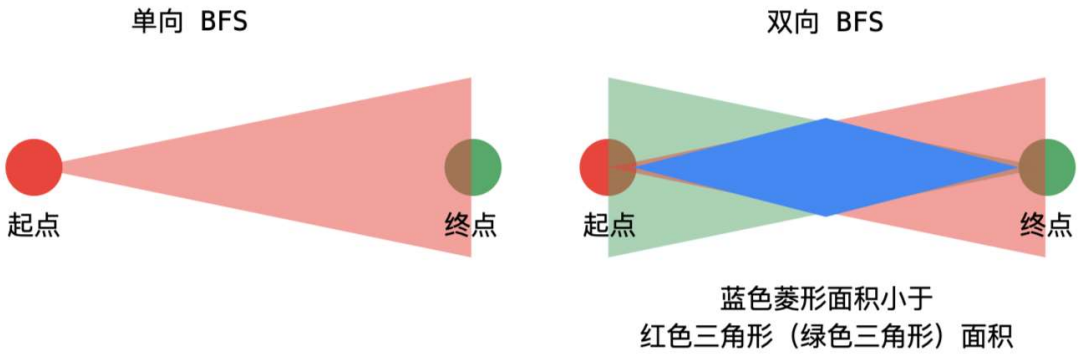
```
            return 0;
        }
        wordSet.remove(beginWord);


        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        Set<String> visited = new HashSet<>();
        visited.add(beginWord);


        int step = 1;
        while (!queue.isEmpty()) {
            int currentSize = queue.size();
            for (int i = 0; i < currentSize; i++) {

                String currentWord = queue.poll();

                if (changeWordEveryOneLetter(currentWord, endWord,
queue, visited, wordSet)) {
                    return step + 1;
                }
            }
            step++;
        }
        return 0;
    }


    private boolean changeWordEveryOneLetter(String currentWord, String
endWord,
                                             Queue<String> queue,
Set<String> visited, Set<String> wordSet) {
        char[] charArray = currentWord.toCharArray();
        for (int i = 0; i < endWord.length(); i++) {

            char originChar = charArray[i];
            for (char k = 'a'; k <= 'z'; k++) {
                if (k == originChar) {
                    continue;
                }
                charArray[i] = k;
                String nextWord = String.valueOf(charArray);
                if (wordSet.contains(nextWord)) {
                    if (nextWord.equals(endWord)) {
                        return true;
                    }
                    if (!visited.contains(nextWord)) {
                        queue.add(nextWord);

                        visited.add(nextWord);
                    }
                }
            }
```

```
                charArray[i] = originChar;
            }
            return false;
        }
    }

    from typing import List
    from collections import deque


    class Solution:
        def ladderLength(self, beginWord: str, endWord: str, wordList:
    List[str]) -> int:
            word_set = set(wordList)
            if len(word_set) == 0 or endWord not in word_set:
                return 0

            if beginWord in word_set:
                word_set.remove(beginWord)

            queue = deque()
            queue.append(beginWord)

            visited = set(beginWord)

            word_len = len(beginWord)
            step = 1
            while queue:
                current_size = len(queue)
                for i in range(current_size):
                    word = queue.popleft()

                    word_list = list(word)
                    for j in range(word_len):
                        origin_char = word_list[j]

                        for k in range(26):
                            word_list[j] = chr(ord('a') + k)
                            next_word = ''.join(word_list)
                            if next_word in word_set:
                                if next_word == endWord:
                                    return step + 1
                                if next_word not in visited:
                                    queue.append(next_word)
                                    visited.add(next_word)
                        word_list[j] = origin_char
                step += 1
            return 0


    if __name__ == '__main__':
        beginWord = "hit"
        endWord = "cog"
        wordList = ["hot", "dot", "dog", "lot", "log", "cog"]
```

```
solution = Solution()
res = solution.ladderLength(beginWord, endWord, wordList)
print(res)
```

方法二：双向广度优先遍历

- 已知目标顶点的情况下，可以分别从起点和目标顶点（终点）执行广度优先遍历，直到遍历的部分有交集。这种方式搜索的单词数量会更小一些；
- 更合理的做法是，每次从单词数量小的集合开始扩散；
- 这里 beginVisited 和 endVisited 交替使用，等价于单向 BFS 里使用队列，每次扩散都要加到总的 visited 里。



参考代码 **2**：

- Java
- Python3

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Solution {

    public int ladderLength(String beginWord, String endWord,
List<String> wordList) {

        Set<String> wordSet = new HashSet<>(wordList);
        if (wordSet.size() == 0 || !wordSet.contains(endWord)) {
            return 0;
        }


        Set<String> visited = new HashSet<>();

        Set<String> beginVisited = new HashSet<>();
        beginVisited.add(beginWord);
        Set<String> endVisited = new HashSet<>();
        endVisited.add(endWord);
```

```
        int step = 1;
        while (!beginVisited.isEmpty() && !endVisited.isEmpty()) {

            if (beginVisited.size() > endVisited.size()) {
                Set<String> temp = beginVisited;
                beginVisited = endVisited;
                endVisited = temp;
            }


            Set<String> nextLevelVisited = new HashSet<>();
            for (String word : beginVisited) {
                if (changeWordEveryOneLetter(word, endVisited, visited,
wordSet, nextLevelVisited)) {
                    return step + 1;
                }
            }


            beginVisited = nextLevelVisited;
            step++;
        }
        return 0;
    }



    private boolean changeWordEveryOneLetter(String word, Set<String>
endVisited,
                                             Set<String> visited,
                                             Set<String> wordSet,
                                             Set<String>
nextLevelVisited) {
        char[] charArray = word.toCharArray();
        for (int i = 0; i < word.length(); i++) {
            char originChar = charArray[i];
            for (char c = 'a'; c <= 'z'; c++) {
                if (originChar == c) {
                    continue;
                }
                charArray[i] = c;
                String nextWord = String.valueOf(charArray);
                if (wordSet.contains(nextWord)) {
                    if (endVisited.contains(nextWord)) {
                        return true;
                    }
                    if (!visited.contains(nextWord)) {
                        nextLevelVisited.add(nextWord);
                        visited.add(nextWord);
                    }
                }
            }

            charArray[i] = originChar;
```

```
        }
        return false;
    }
}
```

```
from typing import List
from collections import deque


class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList:
List[str]) -> int:
        word_set = set(wordList)
        if len(word_set) == 0 or endWord not in word_set:
            return 0

        if beginWord in word_set:
            word_set.remove(beginWord)

        visited = set()
        visited.add(beginWord)
        visited.add(endWord)

        begin_visited = set()
        begin_visited.add(beginWord)

        end_visited = set()
        end_visited.add(endWord)

        word_len = len(beginWord)
        step = 1

        while begin_visited and end_visited:




            if len(begin_visited) > len(end_visited):
                begin_visited, end_visited = end_visited, begin_visited

            next_level_visited = set()
            for word in begin_visited:
                word_list = list(word)

                for j in range(word_len):
                    origin_char = word_list[j]
                    for k in range(26):
                        word_list[j] = chr(ord('a') + k)
                        next_word = ''.join(word_list)
                        if next_word in word_set:
                            if next_word in end_visited:
                                return step + 1
                            if next_word not in visited:
                                next_level_visited.add(next_word)
```

```
                                visited.add(next_word)
                        word_list[j] = origin_char
                begin_visited = next_level_visited
                step += 1
        return 0


if __name__ == '__main__':
    beginWord = "hit"
    endWord = "cog"
    wordList = ["hot", "dot", "dog", "lot", "log", "cog"]

    solution = Solution()
    res = solution.ladderLength(beginWord, endWord, wordList)
    print(res)
```

下一篇：算法实现和优化（Java 双向 BFS，23ms）

学完这一题真的受益匪浅，不仅仅是复习了BFS，其实也涉及到不少跟算法小技巧之类的只是

weiwei哥有男朋友么，我可以男上加男，强人锁男

打开题解，看见大佬的头像，眼前一亮，这题有救了~

自己写了一个简单版的BFS，用时501ms，来学习学习双向BFS。

谢谢大佬的讲解，思路很清晰，也是我第一个看明白的答案，不过提个小建议哈，方法一的python，一开始定义visited的时候，大佬写的是visited = set(beginWord)，但其实这个visited是记录所有转换过的单词集合，一开始初始化visited = set(beginWord)会把beginWord里面所有的字母加进去，第一遍的时候会有误会。建议改成set()空集，这样也可以AC~

P.S 看到下面评论说之前你还会写一句visited.add(beginWord), 其实如果初始化空集，再加visited.add(beginWord)会比较make sense，虽然加不加都不影响结果哈哈哈

大哥，题解变简洁了，或许是我还是很菜，看不懂。不过还是先点赞

跟着威威哥学了不少，尤其是是动态规划的系列题中。 这道题虽然麻烦点，但是经过威威哥讲解后，自己照着思路写了一遍代码已经可以理解了。 特此感谢一下威威哥

为什么方法1的visited后面就不add元素了？我从hit找到了hot，不需要把hot加入visited吗

方法一的Python3解答中，以下两句话功能重复了

```
        visited = set(beginWord)
        visited.add(beginWord)
```

应该删掉visited.add(beginWord)这句

大佬，方法二里面

```
if (charArray[i] == c) {
        continue;
}
```

感觉应该是 charArray[i] == currentChar 吧