# 喜刷刷

**Thursday, November 27, 2014**

## [LeetCode] Word Ladder I, II

### Word Ladder I

Given two words (*start* and *end*), and a dictionary, find the length of shortest transformation sequence from *start* to *end*, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the dictionary

For example,

Given:

*start* = `"hit"`

*end* = `"cog"`

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

As one shortest transformation is `"hit" -> "hot" -> "dot" -> "dog" -> "cog"`,

return its length `5`.

**Note:**

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

### Word Ladder II

Given two words (*start* and *end*), and a dictionary, find all shortest transformation sequence(s) from *start* to *end*, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the dictionary

For example,

Given:

*start* = `"hit"`

*end* = `"cog"`

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

Return

```
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
```

**Note:**

- All words have the same length.

- All words contain only lowercase alphabetic characters.

**思路:**

LeetCode中为数不多的考图的难题。尽管题目看上去像字符串匹配题，但从“shortest transformation sequence from start to end”还是能透露出一点图论中题的味道。如何转化?

1. 将每个单词看成图的一个节点。
2. 当单词s1改变一个字符可以变成存在于字典的单词s2时，则s1与s2之间有连接。
3. 给定s1和s2，问题I转化成了求在图中从s1->s2的最短路径长度。而问题II转化为了求所有s1->s2的最短路径。

无论是求最短路径长度还是求所有最短路径，都是用BFS。在BFS中有三个关键步骤需要实现:

1. 如何找到与当前节点相邻的所有节点。
这里可以有两个策略:
(1) 遍历整个字典，将其中每个单词与当前单词比较，判断是否只差一个字符。复杂度为：n*w，n为字典中的单词数量，w为单词长度。
(2) 遍历当前单词的每个字符x，将其改变成a~z中除x外的任意一个，形成一个新的单词，在字典中判断是否存在。复杂度为：26*w，w为单词长度。
这里可以和面试官讨论两种策略的取舍。对于通常的英语单词来说，长度大多小于100，而字典中的单词数则往往是成千上万，所以策略2相对较优。

2. 如何标记一个节点已经被访问过，以避免重复访问。
可以将访问过的单词从字典中删除。

3. 一旦BFS找到目标单词，如何backtracking找回路径?

**Word Ladder I**

```
class Solution {
public:
    int ladderLength(string start, string end, unordered_set<string> &dict) {
        dict.insert(end);
        queue<pair<string,int>> q;
        q.push(make_pair(start,1));
        while(!q.empty()) {
            string s = q.front().first;
            int len = q.front().second;
            if(s==end) return len;
            q.pop();
            vector<string> neighbors = findNeighbors(s, dict);
            for(int i=0; i<neighbors.size(); i++)
                q.push(make_pair(neighbors[i],len+1));
        }
        return 0;
    }

    vector<string> findNeighbors(string s, unordered_set<string> &dict) {
        vector<string> ret;
        for(int i=0; i<s.size(); i++) {
            char c = s[i];
            for(int j=0; j<26; j++) {
                if(c=='a'+j) continue;
                s[i] = 'a'+j;
                if(dict.count(s)) {
                    ret.push_back(s);
                    dict.erase(s);
                }
            }
            s[i] = c;
        }
        return ret;
    }
};
```

**Word Ladder II**

Posted by Yanbing Shi at 8:09 PM G+1 +2 Recommend this on Google

Labels: algorithm, backtracking, breath first search, graph, hash table, Leetcode, queue, string

## 2 comments:

王丽 May 31, 2016 at 12:53 AM

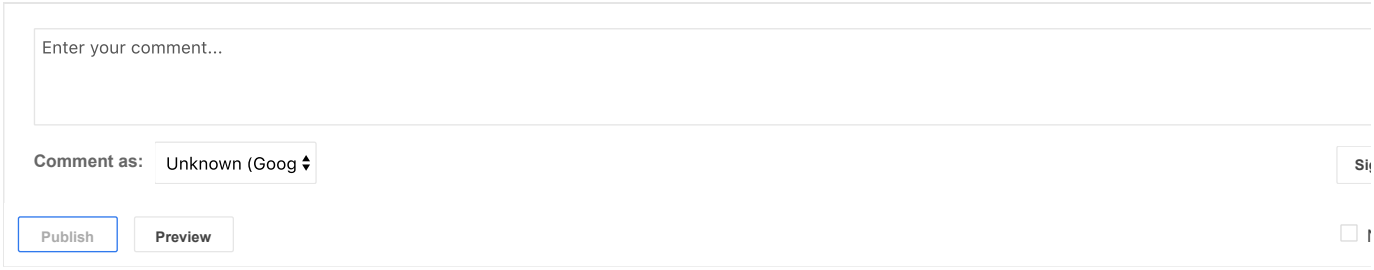
good job

Reply

**Wei Lin** October 13, 2016 at 7:25 PM

where is the source code for Word Ladder II?

Reply




Newer Post Home

Subscribe to: Post Comments (Atom)