天津大学

离散数学实验报告

用图论中的算法解决词梯问题

学　　院　国际工程师学院

专　　业　计算机

学　　号　**2016229082**

姓　　名　王雨朦

年　　级　**2016 级**

2016 年 1 月 14 日

目 录

I. 问题描述

在英文单词表中，有些单词非常相似，它们可以通过只变换一个字符而得到另一个单词。比如：hive-->five，wine-->line，line-->nine，nine-->mine……

给定所有的牛津词典中的单词，大约有 8002 个，我们需要找出通过初始单词通过最少几次变化可以变为目标单词。

比如：(zero-->five)：（zero-->hero-->here-->hire-->five）

II. 问题转化和定义

如果把单词看作结点，假如两个单词之间只有一个字母不同，它们就是相邻结点，结点之间有边相连。就可以根据跟定的牛津单词表构造一个图，而原问题也抽象转化为结点之间的最短路径问题。

由于最短路径问题中，求解源点到终点的最短路径与求解源点到图中所有顶点的最短路径复杂度差不多，故求解两个单词之间的最短路径相当于求解源点单词到所有单词之间的最短路径。

原来的问题定义为：给定两个单词，一个作为初始结点，另一个作为目标结点。需要找出从初始结点开始，经过最少次单个字母的替换，变成目标结点，这个过程中经过了哪些结点？

III. 算法分析

假设所有的单词存储在一个.txt 文件中，每行一个单词。

现在的问题主要有两个：①从文件中读取单词，并构造一个图；②图的广度优先遍历（BFS）和最短路径算法--Dijkstra 算法实现。

3.1 图的构造

由于单词 A 替换一个字符变成单词 B，那么反过来单词 B 替换一个字符也可以变成单词 A（自反性）【wine-->fine； fine-->wine】。故图是一个无向图。

假设单词已经读取到一个 List<String>中，图采用邻接表形式存储，构造图就是：如何根据 List<String> 构造一个 Map<String, List<String>>。

其中，Map 中的 Key 是某个单词，Value 则是该单词的“邻接单词”列表，邻接单词即：该单词经过一个字符的替换变成另一个单词。

如：wine 的邻接单词有：fine，line，nine.....

由于单词都在 List<String>中存储，那么从第 1 个单词开始，依次扫描第 2 个至第 N 个单词，判断第 1 个单词是否与第 2,3,.....N 个单词只差一个字符。这样一遍扫描，找出了 List<String>中第 1 个单词的邻接表。

对于第 2 个单词，依次扫描到第 3,4,....N 个单词，找出 List<String>中第 2 个单词的邻接表。

……

以次类推，能够找出全部 N 个单词的邻接表。

找到所有结点的邻接表，就完成了单词表的无向图的构建，并以邻接表的数据结构存储这个无向图。

3.2 图的 BFS 和 Dijkstra 算法

无向图的 Dijkstra 实现需要一个队列，采用广度优先遍历（BFS）的思想从源点开始向外扩散求解图中其他顶点到源点的距离，之所以这样，是因为无向图一旦访问到某个顶点，更新它的前驱顶点后，它的前驱顶点以后都不会再变。

IV. 伪代码

```
//读单词入单词表
List<String> read(filepath) {
    List<String> theWords;
    read(filepath);
    while(readline()!=NULL)
        theWords.add(words);
}
//构造图
Map<String, List<String>> AdjMap(theWords){
    Map<String, List<String>> adjWords;
    for(word: theWords)
        //将单词按照长度分类，Key 表示单词长度，Value 表示长度相同的单词集合
        update(wordsByLength, word.length(), word);
    //分组处理单词
    for ( groupWords : wordsByLength.values()) {
```

```
            //只在一个组内所有的单词之间进行比较,构造邻接表
            for (int i = 0; i < words.length; i++)
                    for (int j = i + 1; j < words.length; j++)                    if
                        if(oneCharOff(words[i], words[j])) {
                            update(adjWords, words[i], words[j]);
                            update(adjWords, words[j], words[i]);
                        }
            }
            return adjWords;
        }
}// AdjMap
/**使用 Dijkstra 算法求解无向图 从 start 到 end 的最短路径
 * adjWords 保存单词 Map, key:表示某个单词, Value:与该单词只差一个字符的单词
 * start 起始单，end 结束单词
 * return 从 start 转换成 end 经过的中间单词*/
List<String> findChain(adjWords, start, end){
    // Key:某个单词,Value:该单词的前驱单词
    Map<String, String> previousWord=  new HashMap<String, String>();
    Queue<String> queue;
    queue.offer(start);
    while(!queue.isEmpty()){
        String preWord = queue.poll();
        List<String> adj = adjcentWords.get(preWord);
            for (word : adj)
                //这个 word 的距离(前驱单词)没有被更新过.(第一次遍历到该 word)
                if(previousWord.get(word) == null){
                    previousWord.put(word, preWord);
                    queue.offer(word);
                }
            }
        }
        previousWord.put(start, null);//把源点的前驱顶点添加进去
        return getChainFromPreviousMap(previousWord, start, end);
  }
List<String> geChainFromPreviousMap(previousWord, start, end){
        LinkedList<String> result = null;
        if(previousWord.get(end) != null){
            result = new LinkedList<>();
            for(String pre = end; pre != null; pre = previousWord.get(pre))
                result.addFirst(pre);
        }
        return result;
    }
}
```

```
void main(String[] args){
        List<String> thewords = read("resource/vacabulary.txt");
        Map<String, List<String>> adjWords =computeAdjacentWords(thewords);
        //输出图
        /*for(Map.Entry<String, List<String>> entry : adjWords.entrySet()){
                System.out.print("Key = "+entry.getKey()+",value="+entry.getValue());
        }*/
        //输出从起始结点到目标结点经过的最短变换
        System.out.println(findChain(adjWords,"zero","five"));
        System.out.println(findChain(adjWords,"accident","occident"));
        System.out.println(findChain(adjWords,"map","bat"));
    }
}
```

V. 输入和输出

【输入】：初始单词和目标单词。

例：

```
System.out.println(findChain(adjWords,"zero","five"));
System.out.println(findChain(adjWords,"accident","occident"));
System.out.println(findChain(adjWords,"map","bat"));
```

【输出】：从初始单词到目标单词经过的最少转化单词结点。

例：

```
<terminated> WordLadder [Java Application]
[zero, hero, here, hire, fire, five]
[accident, occident]
[map, mat, bat]
```