对于使用递归完成WordLadder的同学，如果在编程中遇到问题可以阅读以下指南：

首先，下面的伪代码是一个朴素的递归实现：

bool dfs(string source\_word, string target\_word) {

如果source\_word == target\_word

成功找到

更新最短转换路径

return true

bool found = false;

对所有与source\_word相邻的word

bool success = dfs(word, target\_word)

如果success是成功的

找到了

found = true

return found

}

我们现在以cat->dog的转换为一个例子，继续解释上面的伪代码。

在第一层递归中，”cat”的第一个相邻word，“aat”会被检查，由于它不在dictionary中，所以跳过。 第二个相邻word是“bat”，它存在于dictionary中，因而对“bat”递归调用。

进入第二层递归调用，这次的目标是“bat”，再依次对与它相邻的word进行遍历和递归调用，以此类推直到某一次递归调用找到了target\_word，我们得到了一个可行的转换路径。

但是，这个路径不一定是最短的可行路径，我们还要继续遍历所有的可能路径。这也是为什么在上面伪代码的遍历中，success表明找到成功了时，仅仅将found置为true而非直接返回。

当第一层对于“cat”的递归调用结束时，我们的整个转换完成。

Hint：

1. 以上伪代码只是一个框架，而不是一个正确的实现。
2. 在WordLadder编程中，极力推荐使用斯坦福库的Lexicon类作为你的Dictionary。
3. 请考虑如何解决cat->aat->bat->cat->aat->bat->cat->…的无限循环？
4. 由于本次测试的数据集相对较大，需要在递归时作一些优化，否则运行时间会过长，包括但不限于：
   1. 在通过递归函数匹配成功后，记录最短的转换路径的长度min\_path，在以后的递归查询中，如果递归深度大于该长度min\_path，直接返回, 取消这一递归分支。
   2. 记录source\_word到每一个可达到的word的最短深度min\_from\_source，当递归查找到这个word时，如果深度已经大于等于min\_from\_source，直接返回，取消这一递归分支。