```
8# -*- coding: utf-8 -*-
"""
Created on Tue 26 Feb 16:05:01 2019

@author: heyu1
"""

import numpy as np, pandas as pd
from collections import deque
import re, requests
from bs4 import BeautifulSoup
isCSVinput = False
WIKI_LEN = len("/wiki/")
while True:
    choice = input("Import search terms from a .CSV (Y/N)?: ")
    if re.match(re.compile("^([YN]|Yes|No)$", re.I), choice) != None:
        if re.match(re.compile("^Y$", re.I), choice[0]) != None:
            isCSVinput = True
        break

enWikiRoot="https://en.wikipedia.org/wiki/"
def hasArticle(searchTerm):
    if searchTerm.strip() == "":
        return False
    reqAttempt = requests.get(enWikiRoot + searchTerm)
    return(reqAttempt.status_code != 404)

def findActualTitleHelper(soupFromPage):
    linkTitles=soupFromPage.find_all('a', title=True)
    titleText=[elem['title'] for elem in linkTitles]
    hasPerm=[re.search(re.compile("Permanent link"),x) != None for
             x in titleText]
    idxPerm=np.where(hasPerm)
#    assert(len(idxPerm) == 1)
    permLinkSuffix = linkTitles[idxPerm[0][0]]['href']
    charIdxTitle = re.search(re.compile("title="), permLinkSuffix).end()
    charIdxOldid = re.search(re.compile("&oldid="), permLinkSuffix).start()
    return permLinkSuffix[charIdxTitle:charIdxOldid]

def findActualTitle(s):
    page = requests.get(enWikiRoot + s)
    soupFromPage = BeautifulSoup(page.text, 'html.parser')
    return findActualTitleHelper(soupFromPage)

def linksOnPage(searchTerm):
    page = requests.get(enWikiRoot + searchTerm)
    soupFromPage = BeautifulSoup(page.text, 'html.parser')
    aTags=soupFromPage.find_all('a',href=True)
    hrefs=[elem['href'] for elem in aTags]
    wikiRe = re.compile("^/wiki/")
    # exclude non-mainspace article pages on Wiki, i.e. those with the
    # following prefixes:
    prefixes = ['User','Wikipedia','File','MediaWiki','Template','Help',
                'Category','Portal','Book','Draft','Education Program',
                'TimedText','Module','Gadget','Gadget definition']
    prefixTalk = [s + ' talk' for s in prefixes]
    prefixesComplete = prefixes + prefixTalk
```

```
    prefixesComplete += ['Special','Talk','MOS']
    pipe = "|"
    regexPrefixes = pipe.join(prefixesComplete)
    # also exclude the Main Page, which changes daily and will lead to
    # unstable results
    nonArticleRe = re.compile("^/wiki/((" + regexPrefixes +
                                        ":)|Main_Page)", re.I)

    articleLinks = list(filter(lambda x: re.search(wikiRe, x) != None and
                               re.search(nonArticleRe, x) == None, hrefs))
    # "/wiki/" is 6 characters
    articles = [lk[WIKI_LEN:] for lk in articleLinks]
    uniqueArticles = set(articles)

    thisSearchTermActualTitle = findActualTitleHelper(soupFromPage)
    if thisSearchTermActualTitle in uniqueArticles:
        uniqueArticles.remove(thisSearchTermActualTitle)
    res = dict(zip(list(uniqueArticles), ["Linked from " + searchTerm
                                    for s in uniqueArticles]))
    return res

# Reasoning adapted from Wikipedia's article on Breadth First Search
def bfs(originTerm, targetArticle, verbose=False):
    futureVertices = deque()
    visited = set()
    pathDict = dict()
    distDict = dict()
    root = originTerm
    # key stores the parent node
    pathDict[root] = None
    distDict[root] = 0
    maxDist = 0
    futureVertices.appendleft(root)
    while len(futureVertices) > 0:
        subtreeRoot = futureVertices.pop()
        currentDist = distDict[subtreeRoot]
        if verbose and currentDist > maxDist:
            maxDist = currentDist
            print("Reached depth ", currentDist, " with term ", subtreeRoot,
                  ", having visited ", str(len(visited)), " articles",
                  sep="")
            # too many lines (684 for depth 1 for Coffee)
#         if verbose and currentDist < 2:
#             print("Examining", subtreeRoot, "at depth =", currentDist)
        if subtreeRoot == targetArticle:
            if verbose:
                print("Found path")
            return constructPath(subtreeRoot, pathDict)
        links = linksOnPage(subtreeRoot)
        if targetArticle in set(links.keys()):
            pathDict[targetArticle] = subtreeRoot
            if verbose:
                print("1st degree connection:", targetArticle, "-->",
                      links[targetArticle])
                print("Visited", str(len(visited)), "articles;", "Added",
                      str(len(pathDict)), "entries to path dictionary")
            return constructPath(targetArticle, pathDict)
```

```
        linkChildren = set(links.keys())
        # remove articles that have been previously visited
        linkChildren.difference_update(visited)
        # remove articles that have already been marked for visitation
        linkChildren.difference_update(set(futureVertices))
        for child in linkChildren:
            pathDict[child] = subtreeRoot
            futureVertices.appendleft(child)
            distDict[child] = currentDist+1
        visited.add(subtreeRoot)

def constructPath(destination, pathDict):
    articleList = list()
    node = destination
    while pathDict[node] != None:
        newNode = pathDict[node]
        node = newNode
        articleList.append(newNode)
    articleList.reverse()
    articleList.append(destination)
    return articleList

def process(s1, s2):
    valid1 = hasArticle(s1); valid2 = hasArticle(s2)
    if not(valid1):
        print("No such path exists. English Wikipedia does not have the" +
              "following term: ", s1)
    if not(valid2):
        print("No such path exists. English Wikipedia does not have the" +
              "following term: ", s2)
    if valid1 and valid2:
        article1 = findActualTitle(s1)
        article2 = findActualTitle(s2)
        if article1 == article2:
            print("\'" + s1 + "\'" + " and " + "\'" + s2 + "\'" +
                  " redirect to " + article1)
        else:
            print("PROCESSING", s1, "and", s2)
            resultantPath = bfs(s1, article2, True)
            print(resultantPath)

if isCSVinput:
    dfInput = pd.read_csv("Sample Wikipedia inputs.csv")
    dfInput['concat'] = dfInput['Article1'] + ' ' + dfInput['Article2']
    dfInput['Answer No.'] = list(dfInput.index)
    distinctPairs = dfInput.groupby('concat').agg({'Answer No.': np.min})
    allArticle1 = list(dfInput['Article1'])
    allArticle2 = list(dfInput['Article2'])
    idxs=list(distinctPairs['Answer No.'])
    idxs.sort()
    for k in idxs:
        process(allArticle1[k], allArticle2[k])
else:
    term1 = input("Enter search term 1: ")
    term2 = input("Enter search term 2: ")
    process(term1, term2)
```