import re  
import urllib  
import urllib.request  
from urllib.parse import urljoin  
import networkx as nx  
import matplotlib.pyplot as plt  
import time  
import subprocess  
from bs4 import BeautifulSoup  
  
absolute\_links = [] #List of all the complete links to crawl  
visited\_sublinks = set() #List of all the visited weblinks  
visited\_edges = set() #List of web links mapping pair  
file\_save\_location = 'C:\\Users\\KCP\\Desktop\\Web\_Crawler\_Download\\' #File Directory Path to save the web page  
downloaded\_files = set() #Complete File Path of the downloaded files  
G = nx.Graph() #Intialize Graph  
  
colors\_mapping={(0,'red'),(1,'blue'),(2,'green'),(3,'gray'),(4,'black'),(5,'yellow')}  
edge\_colors=['red','green','blue','black','yellow']  
colors=[]  
label\_map=dict()  
  
def extract\_weblinks(web\_content):  
 weblink\_regex = re.compile("""<a[^>]+href=["'](.\*?)["']""", re.IGNORECASE) #Regular Expression to extract all the web links on the web page  
 web\_sublinks = re.findall(weblink\_regex, web\_content) #Find all the web links in the web page  
 return web\_sublinks  
  
def get\_save\_webcontent(web\_link):  
 response = urllib.request.urlopen(web\_link)  
 webpage\_source = response.read()  
 web\_content = webpage\_source.decode()  
 mod\_link = web\_link.replace('http://', '') #To generate the name of the webpage file to be downloaded  
 mod\_link = mod\_link.replace('https://', '')  
 mod\_link = mod\_link.replace('/', '.')  
 if mod\_link.endswith('.'):  
 file\_path = file\_save\_location + mod\_link + 'html' #Generate the whole directory location path for the web page to be saved  
 else:  
 file\_path = file\_save\_location + mod\_link + '.html'  
 try:  
 file = open(file\_path, 'r')  
 file\_content = file.read()  
 if file\_content != web\_content: #Check if the downloaded file content is different than the web page content  
 urllib.request.urlretrieve(web\_link, file\_path)  
 downloaded\_files.add(file\_path)  
 return web\_content  
 else:  
 return web\_content  
 except:  
 urllib.request.urlretrieve(web\_link, file\_path) #Download the web page if it is not downloaded  
 return web\_content  
  
def add\_nodes(webpage):  
 G.add\_node(webpage) #Adding the webpage link as the node to the Network Graph G  
 return  
  
def add\_edges(node1, node2):  
 G.add\_edge(node1, node2) #Adding the webpage link as the node to the Network Graph G  
 return  
  
#for key,value in colors\_mapping:  
# if color == key:  
# colors.append(value)  
# print (node1,node2,value)  
  
  
def ping\_latency(web\_link):  
 p = subprocess.Popen('ping ' + web\_link, stdout=subprocess.PIPE)  
 while(True):  
 print(p.communicate()[0])  
 time.sleep(600) #Wait for 10 minutes before pinging again  
 p = subprocess.Popen('ping ' + web\_link, stdout=subprocess.PIPE)  
 if p.poll() == 0: #If Web site is not up  
 break  
 return  
  
def word\_search(input\_word,total\_count\_of\_word):  
 for file\_path in file\_list:  
 file = open(file\_path, 'r')  
 file\_data = file.read()  
 soup = BeautifulSoup(file\_data, "html.parser")  
 paragraph\_list = [p.get\_text() for p in soup.find\_all("p", text=True)]  
 for paragraph in paragraph\_list:  
 words = paragraph.split()  
 count = words.count(input\_word)  
 total\_count\_of\_word = total\_count\_of\_word + count  
 return total\_count\_of\_word  
  
def web\_crawl(web\_link, count):  
 print ("count is %s" %count)  
 try:  
 visited\_sublinks.add(web\_link) # adding the visited weblinks to the Visited Set  
 web\_content = get\_webcontent(web\_link) # Returns the web content of the url link  
 web\_sublinks = (extract\_weblinks(web\_content))  
 except:  
 print ("Some error encountered in %s " %(web\_link))  
 return  
  
 for link in web\_sublinks:  
 if link.startswith('/'): # to check the start of the sub links  
 complete\_link = urljoin(initial\_weblink, link)  
 absolute\_links.append(complete\_link)  
  
 for sub\_link in absolute\_links:  
 if sub\_link not in visited\_sublinks:  
 visited\_edges.append((web\_link,sub\_link,count))  
 print (web\_link,sub\_link,count)  
  
 for sub\_link in absolute\_links:  
 if sub\_link not in visited\_sublinks:  
 if count>=2:  
 break  
 else:  
 count+=1  
 web\_crawl(sub\_link, count)  
  
  
initial\_weblink=input("Enter the website link to crawl")  
web\_crawl(initial\_weblink, 0)  
node\_number=1  
for weblink in visited\_sublinks:  
 add\_nodes(weblink,node\_number)  
 node\_number+=1  
 print(weblink)  
  
for entry in visited\_edges:  
 add\_edges(entry[0], entry[1], entry[2])  
  
#pos=nx.spring\_layout(G)  
  
labels=dict()  
for i,node in enumerate(G.nodes()):  
 labels[node]=i  
#nx.draw\_networkx\_labels(G,pos,labels,font\_size=10)  
nx.draw(G,labels=labels, with\_labels=True,node\_color='yellow',arrows=True)  
  
plt.savefig("web\_crawler\_graph.png") # save as png  
plt.show(G) # display  
word=input("Enter the word to search in the web pages")  
number=word\_search(word,0)  
print (number)  
ping\_latency(initial\_weblink)