

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | **B.Tech** | Semester | : | **Win Sem 21-22** |
| Course | : | **Web Mining Lab** | Code | : | **CSE3024** |
| Faculty | : | **Dr.Bhuvaneswari A** | Slot | : | **L7+L8** |
| Date | : | **10-01-2022** | Marks | : | **10 Points** |

**Name: Anisha Kaushik**

**Reg Number: 19BCE1409**

**Exercise 2: Custom Search Engines / Crawlers**

1. **Implement crawlers which take as input a url, a search word and maximum number of pages to be searched and returns as output all the web pages it searched till it found the search word on a web page or return failure. (5 Marks)**
   1. **Breadth-First-Search**

**Code:**

from bs4 import BeautifulSoup

import requests

import re

root\_URL = input("Enter the website you want to crawl: ")

search\_word = input("Search Word: ")

n = int(input("Number of max webpages to search: "))

class linkQuence:

    def \_\_init\_\_(self):

        #  Url collection visited

        self.visted = []

        #  URL collection to be visited

        self.unVisited = []

    #  Get visited URL queue

    def getVisitedUrl(self):

        return self.visted

    #  Get unvisited URL queue

    def getUnvisitedUrl(self):

        return self.unVisited

    #  Add to the visited URL queue

    def addVisitedUrl(self, url):

        self.visted.append(url)

    #  Remove visited URL

    def removeVisitedUrl(self, url):

        self.visted.remove(url)

    #  Unvisited url out of the queue

    def unVisitedUrlDeQuence(self):

        try:

            return self.unVisited.pop()

        except:

            return None

    #  Ensure that each url is only accessed once

    def addUnvisitedUrl(self, url):

        if url != "" and url not in self.visted and url not in self.unVisited:

            self.unVisited.insert(0, url)

    #  Get the number of URLs visited

    def getVisitedUrlCount(self):

        return len(self.visted)

    #  Get the number of unvisited URLs

    def getUnvistedUrlCount(self):

        return len(self.unVisited)

    #  Determine whether the unvisited URL queue is empty

    def unVisitedUrlsEnmpy(self):

        return len(self.unVisited) == 0

class MyCrawler:

    def \_\_init\_\_(self, seeds):

        #  Initialize the current crawl depth

        self.current\_deepth = 1

        #  Use seed to initialize URL queue

        self.linkQuence = linkQuence()

        if isinstance(seeds, str):

            self.linkQuence.addUnvisitedUrl(seeds)

        if isinstance(seeds, list):

            for i in seeds:

                self.linkQuence.addUnvisitedUrl(i)

        print("Add the seeds url %s to the unvisited url list" % str(self.linkQuence.unVisited))

        #  The main function of the crawling process

    def crawling(self, seeds, crawl\_deepth):

            #  Cycle condition: crawl depth does not exceed crawl\_deepth

            while self.current\_deepth <= crawl\_deepth:

                #  Cycle condition: the link to be crawled is not empty

                while not self.linkQuence.unVisitedUrlsEnmpy():

                    #  Head url out of the queue

                    visitUrl = self.linkQuence.unVisitedUrlDeQuence()

                    print("\"%s\" sunvisited url list" % visitUrl)

                    if visitUrl is None or visitUrl == "":

                        continue

                    #  Get hyperlink

                    links = self.getHyperLinks(visitUrl)   #Get all hyperlinks in visiturl

                    print("Get %d new links" % len(links))

                    #  Put visitUrl in the visited url

                    self.linkQuence.addVisitedUrl(visitUrl)

                    print("Visited url count: " + str(self.linkQuence.getVisitedUrlCount()))

                    print("Visited depth: " + str(self.current\_deepth))

                #  Unvisited urls are listed, that is, all hyperlinks in the visiturl webpage

                for link in links:

                    self.linkQuence.addUnvisitedUrl(link)

                print("%d unvisited links:" % len(self.linkQuence.getUnvisitedUrl()))

                self.current\_deepth += 1

    #  Get the hyperlink in the source code

    def getHyperLinks(self, url):

        links = []

        data = self.getPageSource(url)  #Get url webpage source code

        soup = BeautifulSoup(data,'html.parser')

        a = soup.findAll("a", {"href": re.compile('^http|^/')})

        for i in a:

            if re.search(search\_word, i["href"], re.IGNORECASE) and i["href"].find("http://") != -1:

                links.append(i["href"])

        return links

    #  Get webpage source code

    def getPageSource(self, url):

        try:

            r = requests.get(url)

            r.raise\_for\_status()

            r.encoding = 'utf-8'

            return r.text

        except:

            return ''

def main(seeds, crawl\_deepth):

    craw = MyCrawler(seeds)

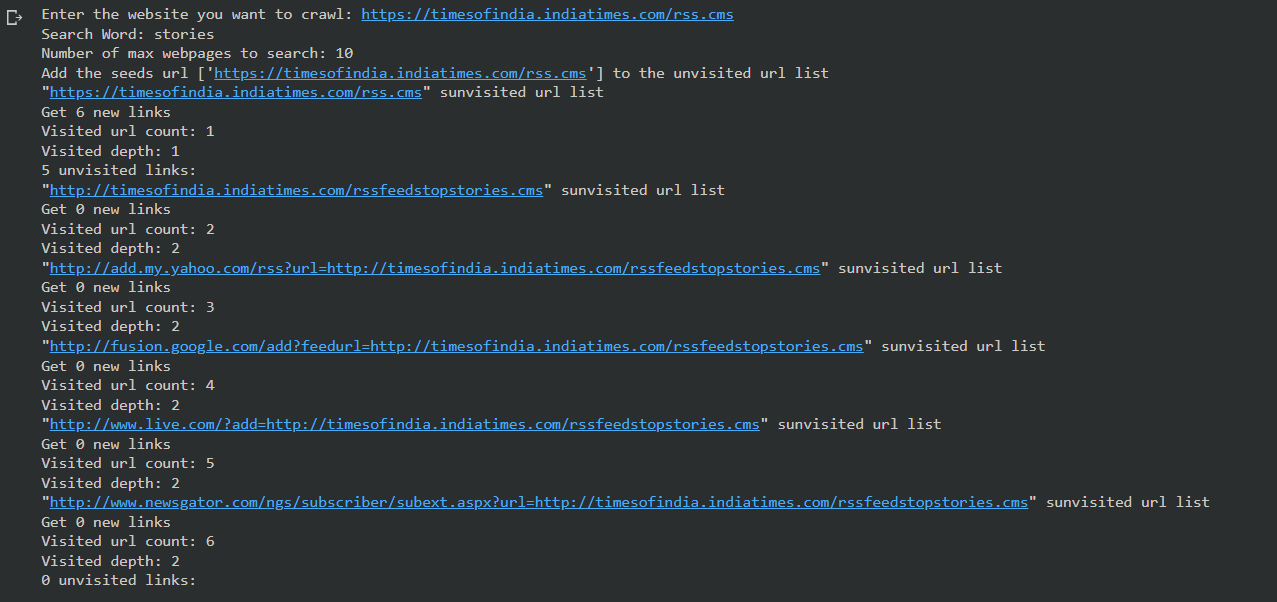
    craw.crawling(seeds, crawl\_deepth)

#Crawl Sina Weibo hyperlinks, depth is 3

#if \_\_name\_\_ == '\_\_main\_\_':

main(root\_URL, 2)

**Output:**



* 1. **Depth-First-Search**

**Code:**

import requests

from bs4 import BeautifulSoup

def search\_dfs(base, path, visited, max\_depth=3, depth=0):

  if depth < max\_depth:

    try:

      soup = BeautifulSoup(requests.get(base + path).text, "html.parser")

      for link in soup.find\_all("a"):

        href = link.get("href")

        if href not in visited:

          visited.add(href)

          print(f"at depth {depth}: {href}")

          if href.startswith("http"):

            search\_dfs(href, "", visited, max\_depth-1, depth + 1)

          else:

            search\_dfs(base, href, visited, max\_depth-1, depth + 1)

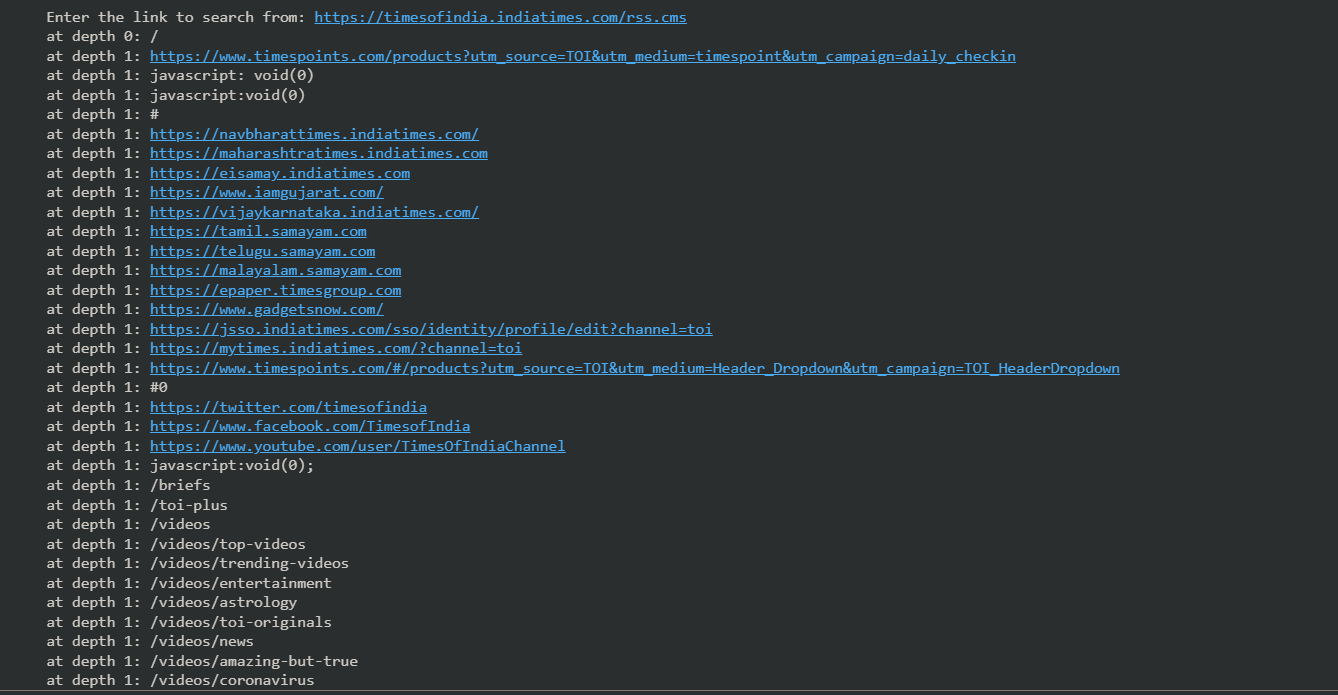
    except:

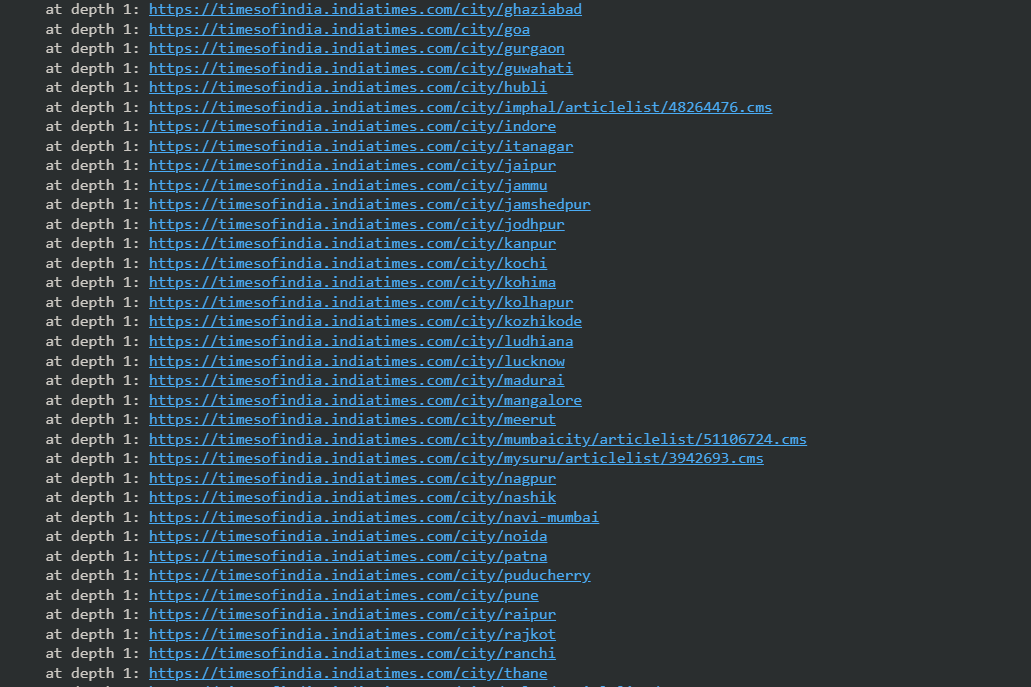
      pass

link=input("Enter the link to search from: ")

search\_dfs(link, "", set([link]))

**Output:**

****

****

1. **Write a Python program to read the given website and extract the phone numbers and emails and contact addresses from Chennai, Amaravathi, Bhopal vit website. (5 Marks) (Challenging Task)**

**Code:**

!pip install selenium

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import pandas as pd

DRIVER\_PATH = 'C:/Users/91942/Downloads/chromedriver'

#driver = webdriver.Chrome(executable\_path='C:/Users/91942/Downloads/chromedriver.exe')

driver = webdriver.Chrome(DRIVER\_PATH)

class color:

   BOLD = '\033[1m'

   END = '\033[0m'

print(color.BOLD + "\n\n Contact Details of Chennai Campus \n\n" + color.END)

driver.get('https://chennai.vit.ac.in/about/contactus/')

chennai = driver.find\_elements\_by\_xpath('/html/body/div[1]/div/div[2]/div/div/div/article/div[2]/div[1]')

chennai\_list = []

for p in range(len(chennai)):

    chennai\_list.append(chennai[p].text)

    print(chennai\_list[p])

print(color.BOLD +  "\n\n Contact Details of Vellore Campus \n\n" + color.END)

driver.get('https://vit.ac.in/contactus') #ignore this cell

vellore = driver.find\_elements\_by\_xpath('/html/body/div[1]/div[2]/div[2]/div[1]/div/div[2]/div/div/section/div/div/div[1]/div/div/div/table[1]/tbody')

vellore\_list = []

for p in range(len(vellore)):

    vellore\_list.append(vellore[p].text)

    print(vellore\_list[p])

print(color.BOLD + "\n\n Contact Details of Andhra Pradesh Campus \n\n" + color.END)

driver.get('https://vitap.ac.in/contact-us/') #ignore this cell

Andhra = driver.find\_elements\_by\_xpath('/html/body/div/div/div/section[2]/div/div/div/div/div/div[1]/div/div[1]')

Andhra\_list = []

for p in range(len(Andhra)):

    Andhra\_list.append(Andhra[p].text)

    print(Andhra\_list[p])

print(color.BOLD + "\n\n Contact Details of Bhopal Campus \n\n" + color.END)

driver.get('https://vitbhopal.ac.in/contacts/')

Bhopal = driver.find\_elements\_by\_xpath('/html/body/div[3]/div/div/div/div')

Bhopal\_list = []

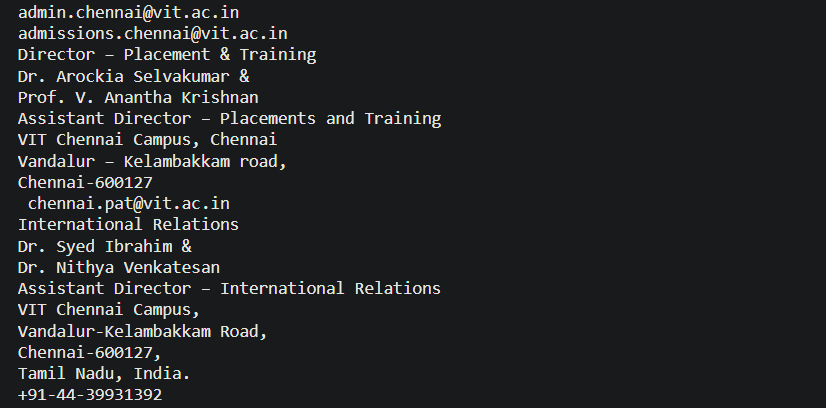
for p in range(len(Bhopal)):

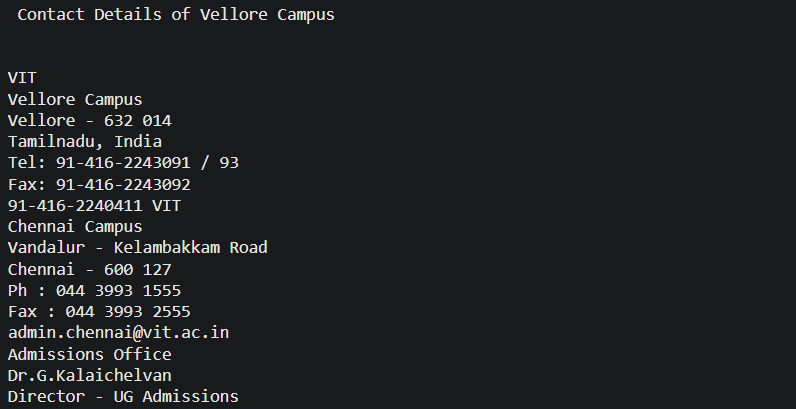
    Bhopal\_list.append(Bhopal[p].text)

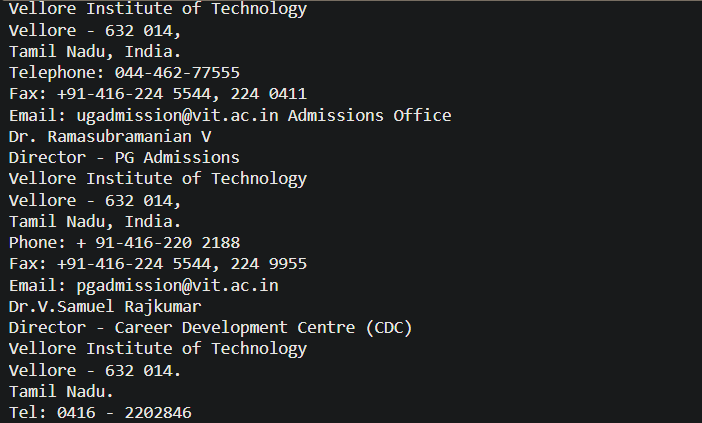
    print(Bhopal\_list[p])

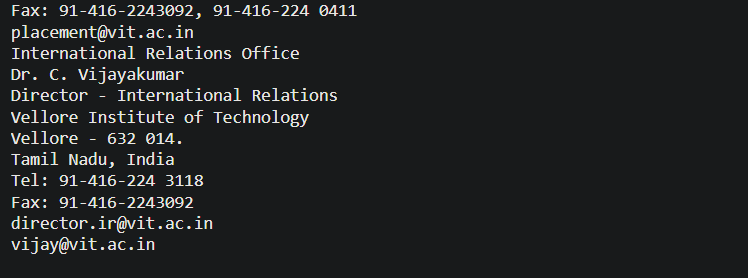
**Output:**

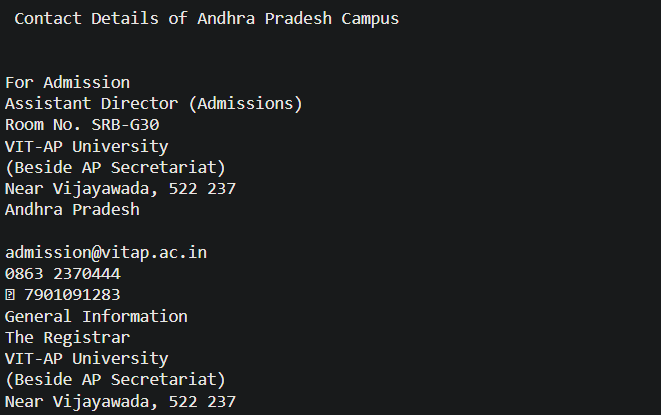
****

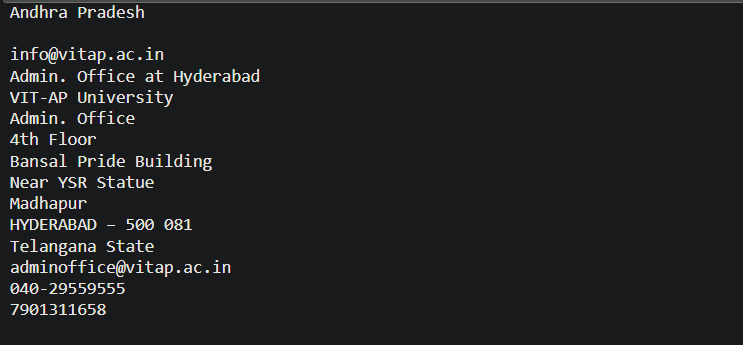
****

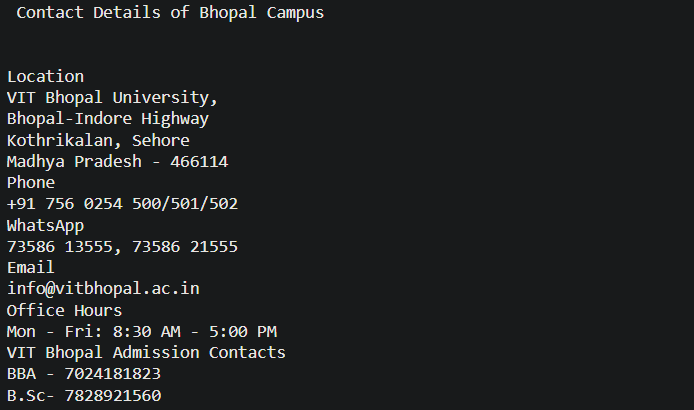
****

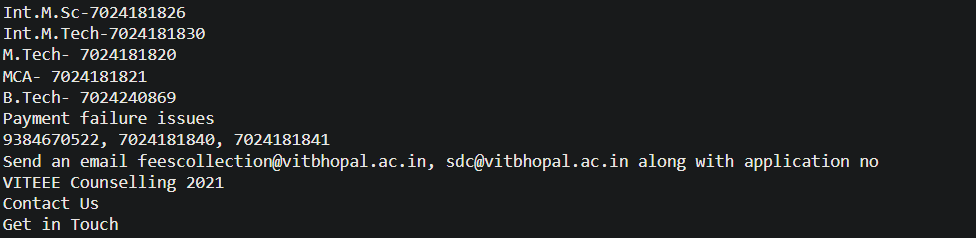
****

****

****

****

****

****

**Link –**

1. <https://colab.research.google.com/drive/1JSSRksHN38AbhgkYZtrWQIj_vXKhuM8x?usp=sharing>
2. <https://drive.google.com/file/d/1RU_YDwT2S86m536Gxcg__-HlUK_KMpZn/view?usp=sharing>

**(CHALLENGING TASK)**