ARYAMAN MISHRA

19BCE1027

EXERCISE 1

Problem Statement

Given a root URL, e.g., "https://en.wikipedia.org/wiki/Web_mining", Design a simple crawler to return all pages that contains a string “crawler" from this site.

Proposed Algorithm

1.Import libraries.
2.Use urlopen to access the web mining Wikipedia page.
3.Call BeautifulSoup function passing url as parameter.
4.Assign string “crawler” to string variable.
5.Use for loop to find all <a> tags(hyperlinks).
6.Check for href and if links contain substring.
7.If true,print the URL.

Data Structure Proposed:None

Implementation

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('https://en.wikipedia.org/wiki/Web_minin
g')
bs = BeautifulSoup(html)
substring = "Crawler"
for link in bs.find_all('a'):
    if 'href' in link.attrs :
      if substring in link.attrs['href'] :
        print(link.attrs['href'])
```

Results

```
/wiki/WebCrawler
/wiki/MetaCrawler
```

EXERCISE 2

Problem Statement

Write a web crawler program which takes 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.

Proposed Algorithm

1. First we import the installed libraries urllib, urlopen, BeautifulSoup, re, requests, numpy.
2. Take user input for URL of the webpage and word to be searched along with number of web pages.
3. Initialize variables reqs and soups for retrieving an arbitrary page and hence produce a list of links on that page.
4. Initialize urls as array to store the links.
5. Use a for loop to find all the valid links.
6. Next we use another for loop to find whether the keyword is present in those links or not else print “failure”.

Data Structure Proposed

Arrays

Implementation

```
import urllib
import urllib.request
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

import requests
import numpy as np
url = input("Enter URL of webpage.")
word=input("Enter word to be searched.")
n=int(input("Enter number of webpages to be searched.")
)
reqs = requests.get(url)
soup = BeautifulSoup(reqs.text, 'html.parser')
i=0
##urls = ["1"] * 100
```

```
urls = ["" for x in range(n)]
for link in soup.find_all('a'):
    if i==n :
      break
    ##print(link.get('href'))
    urls[i]=link.get('href')
    i=i+1

del urls[0]
print(urls)

for j in urls:
  page = urllib.request.urlopen(j).read().decode('utf-
8')
  ##re.findall(word, page)
  page.find(word)
  if page.find(word):
    print(j)
  else:
    print("Failure")
```

Results

```
Enter URL of webpage.https://www.geeksforgeeks.org/numpy-ndarray-fill-in-python/
Enter word to be searched.python
Enter number of webpages to be searched.20
['https://www.geeksforgeeks.org/', 'https://practice.geeksforgeeks.org/topic-tags/', 'https://practice.geeksforgeeks.org/
https://www.geeksforgeeks.org/
https://practice.geeksforgeeks.org/topic-tags/
https://practice.geeksforgeeks.org/company-tags
https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/?ref=ghm
https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/?ref=ghm
https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/?ref=ghm
https://www.geeksforgeeks.org/analysis-of-algorithems-little-o-and-little-omega-notations/?ref=ghm
https://www.geeksforgeeks.org/lower-and-upper-bound-theory/?ref=ghm
https://www.geeksforgeeks.org/analysis-of-algorithms-set-4-analysis-of-loops/?ref=ghm
https://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/?ref=ghm
https://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/?ref=ghm
https://www.geeksforgeeks.org/g-fact-86/?ref=ghm
https://www.geeksforgeeks.org/pseudo-polynomial-in-algorithms/?ref=ghm
https://www.geeksforgeeks.org/polynomial-time-approximation-scheme/?ref=ghm
https://www.geeksforgeeks.org/a-time-complexity-question/?ref=ghm
https://www.geeksforgeeks.org/searching-algorithms/?ref=ghm
https://www.geeksforgeeks.org/sorting-algorithms/?ref=ghm
https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/?ref=ghm
https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/?ref=ghm
```

EXERCISE 3

Problem Statement

Implement breadth-first-search and depth-first-search crawlers for Ex. 2.

Proposed Algorithm(1-6:BFS;1-3:DFS)

1.First we import the installed libraries.

2.Then, we create two empty sets called internal_links and external_links which will store internal and external links separately and ensure that they do not contain duplicates.

3.We then create a method called level_crawler which takes an input URL and crawls it and displays all the internal and external links using the following steps –

- Define a set called url to temporarily store the URLs.
- Extract the domain name of the url using urlparse library.
- Create a beautifulsoup object using HTML parser.
- Extract all the anchor tags from the beautifulsoup object.
- Get the href tags from the anchor tags and if they are empty, don't include them.
- Using urljoin method, create the absolute URL.
- Check for the validity of the URL.
- If the url is valid and the domain of the url is not in the href tag and is not in external links set, include it into external links set.
- Else, add it into internal links set if it is not there and print and put it in temporary url set.
- Return the temporary url set which includes the visited internal links. This set will be used later on.

4.If the depth is 0, we print the url as it is. If the depth is 1, we call the level_crawler method defined above.

5.Else, we perform a breadth first search (BFS) traversal considered the formation of a URL page as tree structure. At the first level we have the input URL. At the next level, we have all the URLs inside the input URL and so on.

6.We create a queue and append the input url into it. We then pop anurl and insert all the urls inside it into the queue. We do this until all the urls at a particular level is not parsed. We repeat the process for the number of times same as the input depth.

1. For each link on the current page, recursively explore it before visiting the remaining links on the page.
2. Use a visited set to keep track of which pages have already been crawled to avoid getting caught in cycles.
3. A "depth" cap allows us to explore all of the links max_depth pages away from the current one. Hence we use an 'if' loop to compare the depth and max_depth and hence execute the following try and except statements.

Data Structure Proposed

Implementation

BREADTH FIRST SEARCH

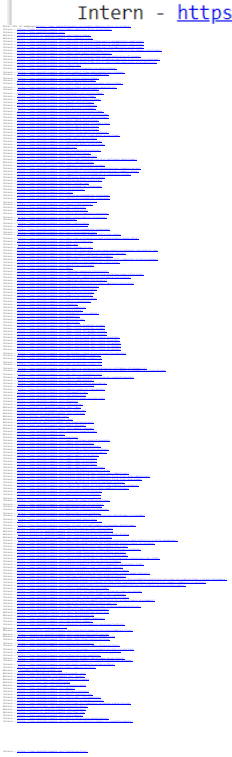
```
from urllib.request import urljoin
from bs4 import BeautifulSoup
import requests
from urllib.request import urlparse

links_intern = set()
input_url = input("Enter URL of webpage.")
depth = 1

links_extern = set()

def level_crawler(input_url):
    temp_urls = set()
```

```
    current_url_domain = urlparse(input_url).
netloc


    beautiful_soup_object = BeautifulSoup(
        requests.get(input_url).content, "lxm
l")

    for anchor in beautiful_soup_object.findA
ll("a"):
        href = anchor.attrs.get("href")
        if(href != "" or href != None):
            href = urljoin(input_url, href)
            href_parsed = urlparse(href)
            href = href_parsed.scheme
            href += "://"
            href += href_parsed.netloc
            href += href_parsed.path
            final_parsed_href = urlparse(href
)
            is_valid = bool(final_parsed_href
.scheme) and bool(
                final_parsed_href.netloc)
            if is_valid:
                if current_url_domain not in
href and href not in links_extern:
                    print("Extern -
 {}".format(href))
```

```
                    links_extern.add(href)
                if current_url_domain in href
 and href not in links_intern:
                    print("Intern -
 {}".format(href))
                    links_intern.add(href)
                    temp_urls.add(href)
    return temp_urls


if(depth == 0):
    print("Intern - {}".format(input_url))

elif(depth == 1):
    level_crawler(input_url)

else:

    queue = []
    queue.append(input_url)
    for j in range(depth):
        for count in range(len(queue)):
            url = queue.pop(0)
            urls = level_crawler(url)
            for i in urls:
                queue.append(i)
```

Results

```
Enter URL of webpage.https://www.geeksforgeeks.org/numpy-ndarray-fill-in-python/
Intern - https://www.geeksforgeeks.org/numpy-ndarray-fill-in-python/
Intern - https://www.geeksforgeeks.org/
Extern - https://practice.geeksforgeeks.org/topic-tags/
Extern - https://practice.geeksforgeeks.org/company-tags
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithems-little-o-and-little-omega-notations/
Intern - https://www.geeksforgeeks.org/lower-and-upper-bound-theory/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-4-analysis-of-loops/
Intern - https://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/
Intern - https://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/
Intern - https://www.geeksforgeeks.org/g-fact-86/
Intern - https://www.geeksforgeeks.org/pseudo-polynomial-in-algorithms/
Intern - https://www.geeksforgeeks.org/polynomial-time-approximation-scheme/
Intern - https://www.geeksforgeeks.org/a-time-complexity-question/
Intern - https://www.geeksforgeeks.org/searching-algorithms/
Intern - https://www.geeksforgeeks.org/sorting-algorithms/
Intern - https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
Intern - https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/
Intern - https://www.geeksforgeeks.org/geometric-algorithms/
Intern - https://www.geeksforgeeks.org/mathematical-algorithms/
Intern - https://www.geeksforgeeks.org/bitwise-algorithms/
Intern - https://www.geeksforgeeks.org/randomized-algorithms/
Intern - https://www.geeksforgeeks.org/greedy-algorithms/
Intern - https://www.geeksforgeeks.org/dynamic-programming/
Intern - https://www.geeksforgeeks.org/divide-and-conquer/
Intern - https://www.geeksforgeeks.org/backtracking-algorithms/
Intern - https://www.geeksforgeeks.org/branch-and-bound-algorithm/
Intern - https://www.geeksforgeeks.org/fundamentals-of-algorithms/
```

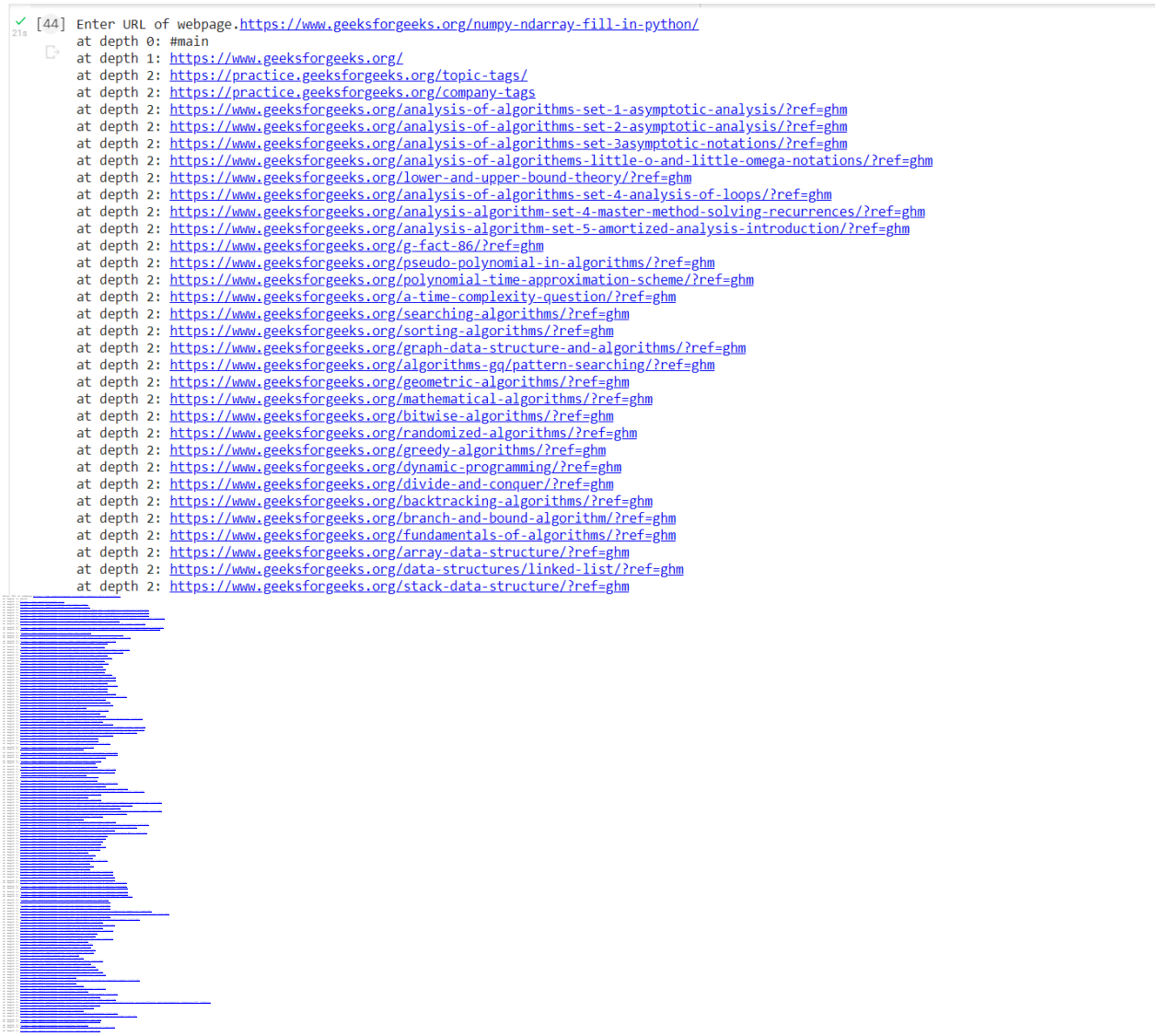


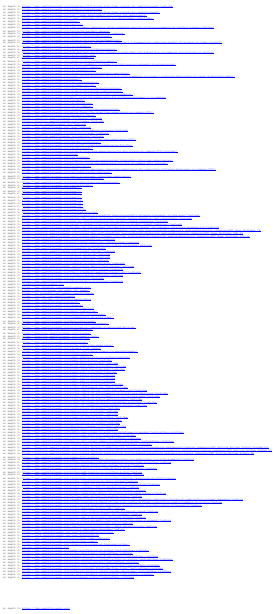
DEPTH FIRST SEARCH

```
import requests
from bs4 import BeautifulSoup

def get_links_recursive(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"
):
                        get_links_recursive(h
ref, "", visited, max_depth, depth + 1)
                    else:
                        get_links_recursive(b
ase, href, visited, max_depth, depth + 1)
        except:
            pass
```

```
##input_url = "https://www.geeksforgeeks.org/machine-learning/"
input_url = input("Enter URL of webpage.")
get_links_recursive(input_url, "", set([input_url]))
```

Results





Conclusion:

All 3 exercises have been succesfully executed and full output has been added with font size 1.