# **INDEX** :

1. Introduction
2. Applications of Web-Crawling
3. Objective
4. Implementation
   - 4.1 Libraries used
   - 4.2 Explanation
   - 4.3 Code
   - 4.4 Graph
5. Conclusion

## <u>**INTRODUCTION**</u> :

The World Wide Web is an interlinked collection of billions of documents formatted using HTML. Due to the large size of this collection, it becomes hard for the users to retrieve data. For these purposes, “Web crawlers” are used. Web crawlers - also known as robots, spiders, worms, walkers, and wanderers- are almost as old as the web itself. The first crawler was built by David Eichmann of NASA using the languages Oracle, C, and wais. At the time when this crawler was built, the size of the web was just about 100,000 web pages.

Web crawlers continuously keep on crawling the web and find any new web pages that have been added to the web and the  pages that have been removed from the web. Due to growing and dynamic nature of the web, it has become a challenge to traverse all URLs in the web documents and to handle these URLs. A focused crawler is an agent that targets a particular topic and visits and gathers only relevant web pages.

## <u>**APPLICATIONS OF WEB-CRAWLING**</u> :

- Many sites, in particular search engines, use spidering as a means of providing up-to-date data.
- Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, that will index the downloaded pages to provide fast searches.
- Crawlers can also be used for checking links or validating HTML codes.
- Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting spam email addresses.

## OBJECTIVE :

The objective of this mini-project is to develop a basic version of a web-crawler. This crawler travels through all the links present in the sub-domain of IIT Ropar (with the seed URL being *http://www.iitrpr.ac.in/*) and keeps a track of the visited URLs. As an output, it displays a graph which shows the interlinking between the various aforesaid URLs.

## IMPLEMENTATION :

- ### LIBRARIES USED:

BeautifulSoup: Beautiful Soup is a Python package for parsing HTML and XML documents. It creates parse trees that is helpful in extracting data easily.

networkx: NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

requests: The requests module allows us to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

To request a response from the server, there are mainly two methods:

1. GET: to request data from the server.
2. POST: to submit data to be processed to the server.

- **EXPLANATION** :

Steps involved in web scraping:

1. Send an HTTP request to the URL of the webpage you want to access (here, *“http://www.iitrpr.ac.in/”).* The server responds to the request by returning the HTML content of the webpage. For this task, we will use a third-party HTTP library (here, “requests”) for python requests.

2. Once we have accessed the HTML content, we are left with the task of parsing the data. Since most of the HTML data is nested, we cannot extract data simply through string processing. One needs a parser which can create a nested/tree structure of the HTML data. There are many HTML parser libraries available. We have used “lxml” for this purpose.
3. Now, all we need to do is navigating and searching the parse tree that we have created, i.e. tree traversal. For this task, we have used another third-party python library, Beautiful Soup. It is a Python library for extracting data out of HTML and XML files.

The crawler starts from *“http://www.iitrpr.ac.in/”* and extracts the URLs in the sub-domain of *“http://www.iitrpr.ac.in/”*. The URLs are added to a “Queue" (or the crawl frontier ) for further processing. Queue is a data structure which works on “FIFO” concept (“First in, First out”).

The crawler, then picks up URLs successively from the front-end of the queue (or the crawl frontier) and crawls the non-visited URLs (in the sub-domain of “*http://www.iitrpr.ac.in/*”) linked to the picked URL. These new entries are added to the rear end of the queue.

The program, ideally, continues till all the URLs in the queue are exhausted. However, for the sake of simplicity, we have considered the URLs upto the third - neighbours from the seed URL.

For visualisation of the network, we have formed a Graph using the library “networkx”. The graph has been edited using **"Gephi"**. The graph shows interlinking pattern of all the URLs, upto the third neighbours, starting from the seed URL.

- **CODE** :

```
from bs4 import BeautifulSoup

import requests

import networkx as nx

g=nx.Graph()

def my_neighbours(queue,visited):

    central_node=queue[0]

    n_queue=[]

    while(len(queue)>0):

        try:

            res=requests.get(central_node)

            soup = BeautifulSoup(res.text,"lxml")

            nlink=[]


            try:
                for link in soup.find_all('a'):

                    temp=link.get('href')

                    if(temp!=""):

                        if(temp.startswith("http://www.iitrpr.ac.in")):

                            nlink.append(temp)

                        elif(temp.startswith("/")):

                            nlink.append("http://www.iitrpr.ac.in"+temp)

                        elif(not(temp.startswith("http")) and
not(temp.startswith("/"))):

                            nlink.append("http://www.iitrpr.ac.in/"+temp)

                for link in nlink:

                    if(link not in visited):

                        visited.append(link)

                        n_queue.append(link)

                    g.add_edge(central_node, link)

            except AttributeError:

                central_node=queue[0]

                del queue[0]

            else:

                central_node=queue[0]

                del queue[0]

        except(requests.exceptions.MissingSchema,
requests.exceptions.ConnectionError,requests.exceptions.InvalidURL,
requests.exceptions.InvalidSchema):
```

```
            continue

    return n_queue,visited

central_node="http://www.iitrpr.ac.in"

visited=[central_node]

res=requests.get(central_node)

soup = BeautifulSoup(res.text,"lxml")

nlink=[]

for link in soup.find_all('a'):

    temp=link.get('href')

    if(temp!=""):

        if(temp.startswith("http://www.iitrpr.ac.in")):

            nlink.append(temp)

        elif(temp.startswith("/")):

           nlink.append("http://www.iitrpr.ac.in"+temp)

        elif(not(temp.startswith("http")) and not(temp.startswith("/"))):

           nlink.append("http://www.iitrpr.ac.in/"+temp)

#first neighbours

queue1=nlink[:]

for link in nlink:

    visited.append(link)

    g.add_edge(central_node, link)

#second neighbours
```

```
queue2,visited=my_neighbours(queue1,visited)

#third neighbours

queue3,visited=my_neighbours(queue2,visited)

nx.draw(g)

nx.write_gexf(g, “an.gexf”)
```

---

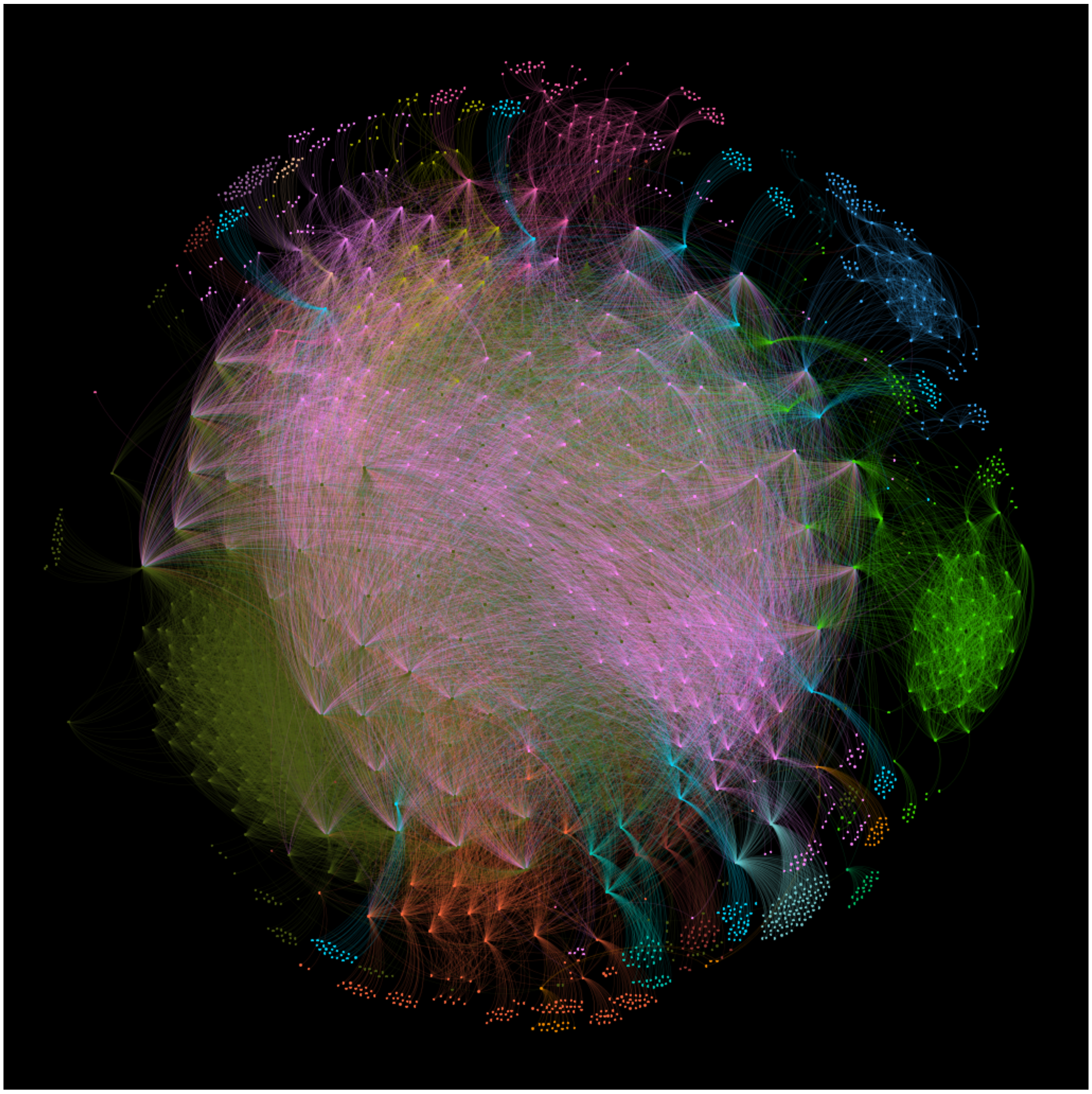
For obtaining the network of all URLs in the sub-domain of “http://www.iitrpr.ac.in”, instead of only the URLs upto the third neighbours, the following modifications can be made to the previous code :

```
central_node="http://www.iitrpr.ac.in"

visited=[central_node]

queue=[central_node]

while(len(queue)>0):

        try:

            res=requests.get(central_node)

            soup = BeautifulSoup(res.text,"lxml")

            nlink=[]

            try:

                for link in soup.find_all('a'):

                    temp=link.get('href')

                    if(temp!=""):
```

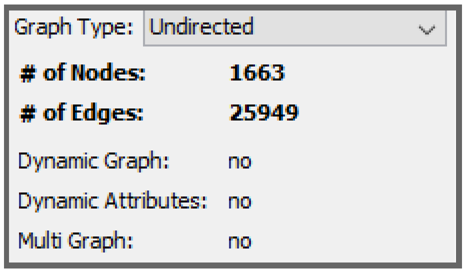
```
if(temp.startswith("http://www.iitrpr.ac.in")):

                            nlink.append(temp)

                        elif(temp.startswith("/")):

                            nlink.append("http://www.iitrpr.ac.in"+temp)

                        elif(not(temp.startswith("http")) and
not(temp.startswith("/"))):

                            nlink.append("http://www.iitrpr.ac.in/"+temp)

                for link in nlink:

                    if(link not in visited):

                        visited.append(link)

                        queue.append(link)

                    g.add_edge(central_node, link)

            except AttributeError:

                central_node=queue[0]

                del queue[0]

            else:

                central_node=queue[0]

                del queue[0]

        except(requests.exceptions.MissingSchema,
requests.exceptions.ConnectionError,requests.exceptions.InvalidURL,
requests.exceptions.InvalidSchema):

            continue
```

- **GRAPH :**

  Graph representing the network of URLs (upto 3rd neighbours of the seed link) :

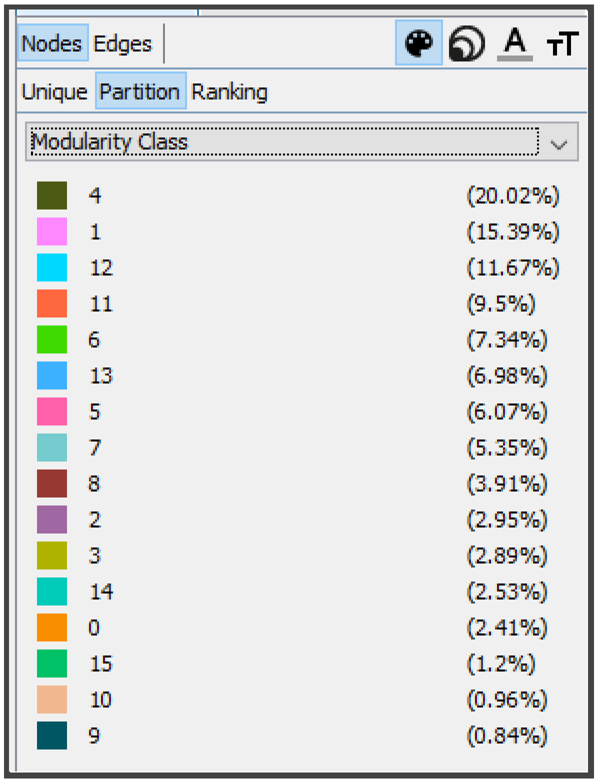


- **GRAPH EXPLANATION**



Graph Type: Undirected

| | |
|---|---|
| **# of Nodes:** | **1663** |
| **# of Edges:** | **25949** |
| Dynamic Graph: | no |
| Dynamic Attributes: | no |
| Multi Graph: | no |



Modularity Class

| | | |
|---|---|---|
| 🟩 | 4 | (20.02%) |
| 🟪 | 1 | (15.39%) |
| 🟦 | 12 | (11.67%) |
| 🟧 | 11 | (9.5%) |
| 🟩 | 6 | (7.34%) |
| 🟦 | 13 | (6.98%) |
| 🟥 | 5 | (6.07%) |
| 🟦 | 7 | (5.35%) |
| 🟥 | 8 | (3.91%) |
| 🟪 | 2 | (2.95%) |
| 🟨 | 3 | (2.89%) |
| 🟦 | 14 | (2.53%) |
| 🟧 | 0 | (2.41%) |
| 🟩 | 15 | (1.2%) |
| 🟧 | 10 | (0.96%) |
| 🟦 | 9 | (0.84%) |

The graph generated by networkx is edited using "**Gephi**", which is an open-source network analysis and visualization software package written in Java on the NetBeans platform.
The graph consists of 1663 nodes and 25949 edges. The palette of colours used for the nodes is shown alongside. The colour of edges depends on the target node.

# BIBLIOGRAPHY

https://ieeexplore.ieee.org/document/5395052

https://www.edureka.co/blog/web-scraping-with-python/

https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://gephi.org/