Cel zadania
Celem zadania było napisanie, w języku python, aplikacji budującej graf połączeń pomiędzy stronami internetowymi. Począwszy od zadanego adresu url, rekurencyjnie podążającej za znalezionymi na stronie linkami aż do zadanej głębokości i wyświetlającej graf połączeń w postaci graficznej.

Opis programu

Program składa się z dwóch skryptów, scraping.py i create_graph.py.

Skrypt scraping.py z linków o domenie z ustalonego adresu URL wyszukuje zewnętrzne linki, z których także wyszukuje zewnętrzne linki, aż do zadanej głębokości. Program w międzyczasie tworzy zbiór węzłów, czyli zbiór zewnętrznych linków ustalonego adresu URL i tworzy listę połączeń między linkiem z którego wyszukiwano, a tym który znaleziono. Następnie węzły i połączenia zapisywane są do plików csv.

Skrypt create_graph.py z plików csv skryptu scraping.py tworzy graficzną reprezentację grafu.

Wykorzystane biblioteki:

Request - Requests jest biblioteką HTTP, jej celem jest uczynienie żądań HTTP prostszymi i bardziej przyjaznymi dla człowieka.

Urllib - jest modułem Pythona, który może być używany do otwierania adresów URL. Definiuje funkcje i klasy, które pomagają w działaniach związanych z adresami URL.

Beautiful Soup - Beautiful Soup to pakiet Pythona do parsowania dokumentów HTML i XML. Tworzy drzewo parsowania dla parsowanych stron, które może być użyte do ekstrakcji danych z HTML, co jest przydatne w web scrapingu.

Networkx - jest biblioteką Pythona służącą do badania grafów i sieci.

Colorama - generuje kody znaków ANSI do drukowania kolorów na terminalach

Ssl - zapewnia dostęp do funkcji szyfrowania warstwy transportowej (często znanej jako "Secure Sockets Layer") i uwierzytelniania peer dla gniazd sieciowych, zarówno po stronie klienta, jak i serwera.

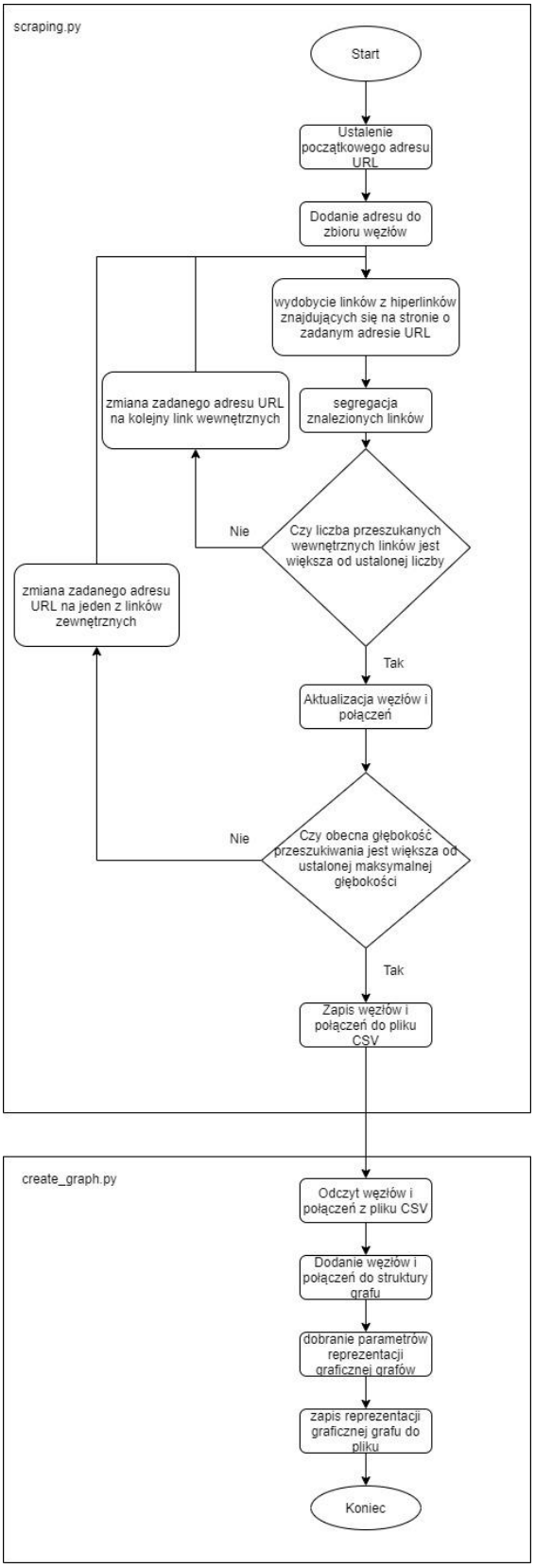
Opis funkcji:

get_all_website_links() – funkcja odpowiadająca za znajdowanie hiperlinków na stronie internetowej o zadanym adresie url (parametr funkcji). Znalezione linki są przypisywane do odpowiednich kontenerów. Zwraca wewnętrzne linki, które nie powtórzyły się w poprzednich wywołaniach funkcji.

crwal() – funkcja wywołująca funkcję get_all_website_links(). Pierwszym parametrem jest adres url. Dla liczby linków ograniczonej przez drugi parametr wywołuje się rekurencyjnie.

Deep_crawl() – funkcja umożliwiająca wyszukiwanie linków do zadanej głębokości. Aktualizuje kontenery z węzłami i połączeniami. Wywołuje funkcję crawl i samą siebie. Jej parametrami wejściowymi jest zadana głębokość i maksymalna liczba wyszukiwania linków z linków o tej samej domenie.

Schemat blokowy



Rysunek 1

Przykład działania

Przykładem działania jest zbudowania grafu ze strony mchtr.pw.edu.pl. Pozostałe parametry początkowe to :

maksymalna liczba wyszukiwania linków z linków o tej samej domenie – 5

maksymalna głębokość – 2 (liczenie głębokości zaczyna się od 0)

Fragment połączeń dla mchtr.pw.edu.pl

```
source,target,value
www.mchtr.pw.edu.pl,sspw.pl,1
www.mchtr.pw.edu.pl,www.klub-mechanik.pl,1
www.mchtr.pw.edu.pl,www.bss.ca.pw.edu.pl,1
www.mchtr.pw.edu.pl,writefull.com,1
www.mchtr.pw.edu.pl,absolwenci.mchtr.pw.edu.pl,1
www.mchtr.pw.edu.pl,www.pw.edu.pl,1
www.mchtr.pw.edu.pl,www.facebook.com,1
www.mchtr.pw.edu.pl,www.facebook.com,1
www.mchtr.pw.edu.pl,info.mchtr.pw.edu.pl,1
www.mchtr.pw.edu.pl,pl-pl.facebook.com,1
www.mchtr.pw.edu.pl,www.bss.ca.pw.edu.pl,1
sspw.pl,sspw.pl,1
sspw.pl,www.klub-mechanik.pl,1
sspw.pl,www.bss.ca.pw.edu.pl,1
sspw.pl,writefull.com,1
sspw.pl,absolwenci.mchtr.pw.edu.pl,1
sspw.pl,www.pw.edu.pl,1
sspw.pl,www.facebook.com,1
sspw.pl,www.facebook.com,1
sspw.pl,info.mchtr.pw.edu.pl,1
sspw.pl,pl-pl.facebook.com,1
sspw.pl,www.bss.ca.pw.edu.pl,1
www.klub-mechanik.pl,panel.home.pl,1
www.klub-mechanik.pl,poczta.home.pl,1
www.klub-mechanik.pl,home.pl,1
www.klub-mechanik.pl,home.pl,1
www.klub-mechanik.pl,home.pl,1
www.klub-mechanik.pl,home.pl,1
www.klub-mechanik.pl,home.pl,1
www.klub-mechanik.pl,home.pl,1
```

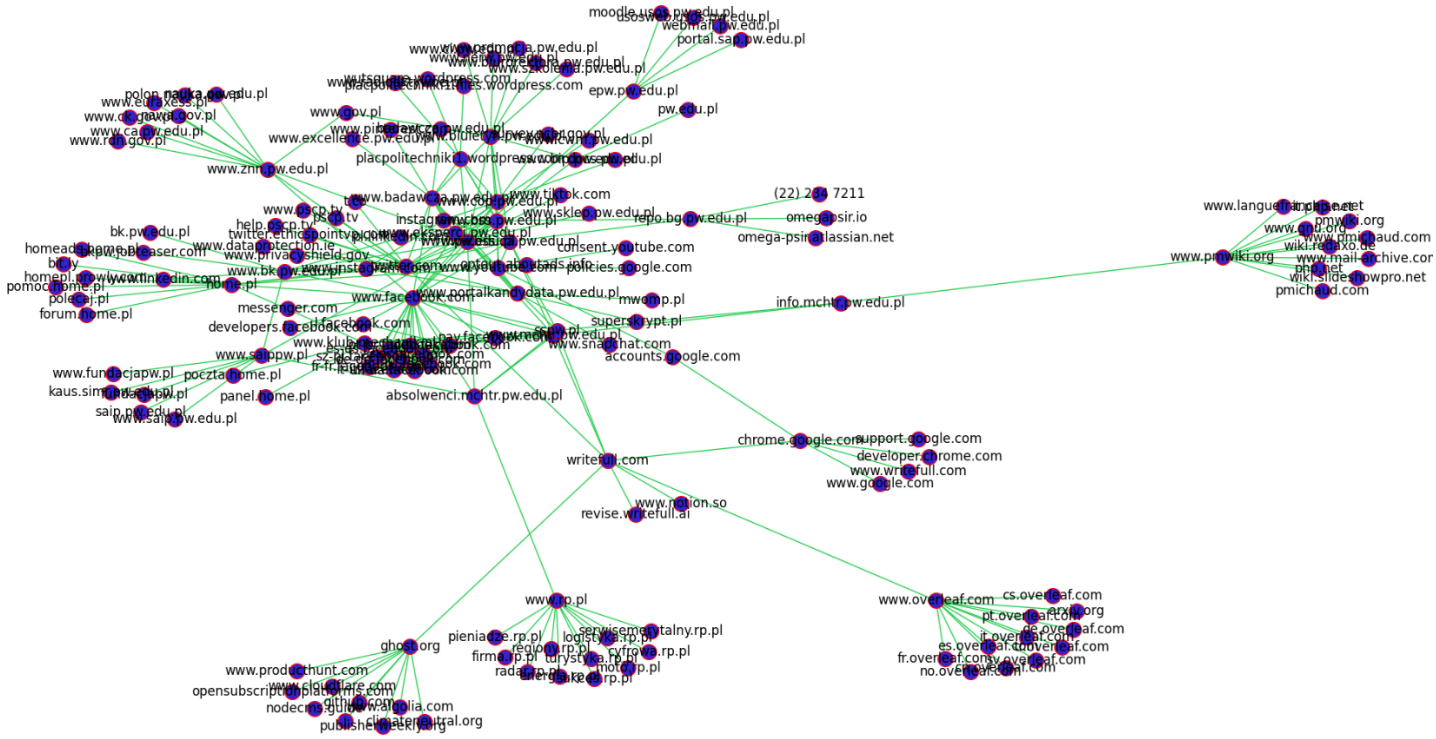
Rysunek 2

Fragment węzłów dla mchtr.pw.edu.pl

name
bit.ly
www.nerw.pw.edu.pl
www.bip.pw.edu.pl
www.privacyshield.gov
wutsquare.wordpress.com
www.youtube.com
de.overleaf.com
policies.google.com
www.rdn.gov.pl
placpolitechniki1.wordpress.com
polon.nauka.gov.pl
support.google.com
writefull.com
revise.writefull.ai
bkpw.jobteaser.com
repo.bg.pw.edu.pl
superskrypt.pl
www.fundacjapw.pl
webmail.pw.edu.pl
www.klub-mechanik.pl
de-de.facebook.com
home.pl
www.szkolenia.pw.edu.pl
www.pscp.tv
turystyka.rp.pl
climateneutral.org
www.bss.ca.pw.edu.pl
www.algolia.com
www.cwm.pw.edu.pl
sukces.rp.pl
it.overleaf.com
www.pw.edu.pl
moodle.usos.pw.edu.pl

Rysunek 3

Uzyskany graf połączeń



Rysunek 4

Podsumowanie

Pierwotny cel został zrealizowany. Dzięki wielu bibliotekom możliwych do wykorzystanie w web scraping'u python jest odpowiednim narzędziem do realizowania tego typu aplikacji. Z pozoru najtrudniejsze zadaniem mogło się wydawać znalezienie hiperlinków na stronie internetowej, jednak zostało to zrealizowanie przy pomocy metody findAll() obiektu typu BeautifulSoup. Przy wywołaniu programu należy zwrócić uwagę na ilość wykonywanych obliczeń, ponieważ ich liczba wzrasta wraz z ilością linków zewnętrznych na danej stronie internetowej. Dlatego też ograniczono liczbę linków zewnętrznych szukanych na danej stronie do 10. Z uwagi na niewielką interakcję programu z użytkownikiem zdecydowano się nie implementować GUI.

Biblografia

https://pythonspot.com/extract-links-from-webpage-beautifulsoup/

https://www.kaggle.com/jncharon/python-network-graph?select=stack_network_links.csv

Kod programu:

scraping.py

```
import requests
from urllib.parse import urlparse, urljoin
from bs4 import BeautifulSoup
import colorama
import ssl


colorama.init()
GRAY = colorama.Fore.LIGHTBLACK_EX
RESET = colorama.Fore.RESET

internal_urls = set() #internal links form one domain
external_urls = set() #external links form one domain
all_external_urls = set() #external link from one deepth
nodes = set()
edges = list()
total_urls_visited = 0
deepth = 0

def is_valid(url):
    #check if url is valid
    parsed = urlparse(url)
    return bool(parsed.netloc) and bool(parsed.scheme)
def get_all_website_links(url):

    # urls set for scraping one link
    urls = set()

    # domain of URL
    domain_name = urlparse(url).netloc

    #get links from URL
    try:
        soup = BeautifulSoup(requests.get(url).content, "html.parser")
    except ssl.SSLCertVerificationError:
        pass

    #search for a tags, egs: <a href="https://www.onet.pl">Visit Onet.pl!</a>
    for a_tag in soup.findAll("a"):

        #get href
        href = a_tag.attrs.get("href")
        if href == "" or href is None:
            continue
        # join the URL if it's relative (not absolute link)
        href = urljoin(url, href)
        parsed_href = urlparse(href)
        # remove URL GET parameters, URL fragments, etc.
        href = parsed_href.scheme + "://" + parsed_href.netloc + parsed_href.path
        if "download" in href:
            #should exclude .pdf, .docx, etc.
```

```
            continue
        if ".pdf" in href:
            #should exclude .pdf, .docx, etc.
            continue
        if parsed_href.scheme == "mailto":
            #exclude mails
            continue
        if not is_valid(href):
            # not a valid URL
            continue
        if href in internal_urls:
            # already in the set
            continue
        if domain_name not in href:
            # external link
            if href not in external_urls:
                if len(external_urls) > 10: #number of external links form input link
                    continue
                print(f"{GRAY}External link: {href}{RESET}")
                external_urls.add(href)
                all_external_urls.add(href)
            continue
        urls.add(href)
        internal_urls.add(href)
    return urls


def crawl(url, max_urls):

    global total_urls_visited
    total_urls_visited += 1

    #get links from URL
    links = get_all_website_links(url)

    # get links form multiple links with domain of URL
    for link in links:
        if total_urls_visited > max_urls:
            break
        crawl(link, max_urls=max_urls)

def deep_crawl(dp,max_urls):

    # all links from deepth
    ext_temp = all_external_urls.copy()
    all_external_urls.clear()

    global deepth
    deepth += 1

    if deepth > dp:
            return

    # get links form every link from deepth
```

```
    for link in ext_temp:

        global total_urls_visited
        total_urls_visited = 0

        internal_urls.clear()

        crawl(link,max_urls)

        #add domains to edges and nodes
        for link2 in external_urls:
            edges.append((urlparse(link).netloc,urlparse(link2).netloc))
            nodes.add(urlparse(link2).netloc)

        external_urls.clear()

    deep_crawl(dp, max_urls)



def main():
    max_urls = 5 #number of internal links of url from which we scrap external links
    how_deep = 2
    url = "http://www.mchtr.pw.edu.pl/"
    nodes.add(urlparse(url).netloc)

    #first iteration - deepth 0
    crawl(url, max_urls)
    for link in external_urls:
        edges.append((urlparse(url).netloc,urlparse(link).netloc))
    deep_crawl(how_deep, max_urls)

    #saving nodes and edges to csv file
    with open(f"nodes.csv", "w") as f:
        print("name", file=f)
        for node in nodes:
            print(node.strip(), file=f)

    with open(f"edges.csv", "w") as f:
        print("source,target,value", file=f)
        for link in edges:
            print( str(link[0]) + ',' + str(link[1]) + ",1", file=f)

main()
```

create_graph.py

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd

G = nx.Graph(day="external_links_scraping")
df_nodes = pd.read_csv('nodes.csv', error_bad_lines=False)
df_edges = pd.read_csv('edges.csv', error_bad_lines=False)

for index, row in df_nodes.iterrows():
    G.add_node(row['name'])

for index, row in df_edges.iterrows():
    G.add_weighted_edges_from([(row['source'], row['target'], row['value'])])

plt.figure(figsize=(20,20))
options = {
    'edge_color': '#23cc4d',
    'width': 1,
    'with_labels': True,
    'font_weight': 'regular',
}
nx.draw(G, node_color='#3723cc', node_size=200, pos=nx.spring_layout(G, k=0.25, iterations
=150), **options)
ax = plt.gca()
ax.collections[0].set_edgecolor("#ff0505")
plt.savefig('graph.png')
plt.show()
```