Name: Nikita Bhrugumaharshi Emberi

STA 220 Assignment 2

Due Februrary 9, 2024 by 11:59pm. Submit your work by uploading it to Gradescope through Canvas.

Instructions:

1. **Provide your solutions in new cells following each exercise description. Create as many new cells as necessary. Use code cells for your Python scripts and Markdown cells for explanatory text or answers to non-coding questions. Answer all textual questions in complete sentences.**
2. **The use of assistive tools is permitted, but must be indicated. You will be graded on you proficiency in coding. Produce high quality code by adhering to proper programming principles.**
3. **Export the .jpynb as .pdf and submit it on Gradescope in time. To facilitate grading, indicate the area of the solution on the submission. Submissions without indication will be marked down. No late submissions accepted.**
4. **If test cases are given, your solution must be in the same format.**
5. **The total number of points is 10.**

Exercise 1

We will compute the PageRank of the articles of the Sinhala wikipedia, which is available at si.wikipedia.org. Additional information of the Sinhala wiki can be found here.

Hints: If you don't speak Sinhalese, you might want to learn the wiki logic from the english wikipedia, and translate your findings. Also, caching is highly recommended.

Importing Required Libraries throughout the assignment

In [37]:

```
import requests
import lxml.html as lx
import time
import pandas as pd
import requests_cache
import re
import concurrent.futures, threading
import numpy as np
from scipy.sparse import csr_matrix
import tqdm
from tqdm import tqdm

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.common.exceptions import TimeoutException
```

In [3]:

```
requests_cache.install_cache("assignment02")
```

(a) Use the special AllPages page and understand its logic to retrieve the url of all articles in the sinhalese wikipedia. Make sure to skip redirections.

How many articles are there?

URL of sinhala wikipedia's AllPages page:

In [4]:

```
url = "https://si.wikipedia.org/wiki/%E0%B7%80%E0%B7%92%E0%B7%81%E0%B7%9A%E0%B7%82:%E0%B7%83%E0%B7%92%E0%B6%BA%E0%B7%85%E0%B7%94_%E0%B6%B4%E0%B7%92%E0%B6%A7%E0%B7%94"
```

In [5]:

```
# Sending a GET request to the specified URL
result = requests.get(url)

# Raising an exception if the request isn't successful
result.raise_for_status()
```

In [6]:

```
# Initializing a Chrome webdriver instance
driver = webdriver.Chrome()

# Setting a page load timeout of 20 seconds
driver.set_page_load_timeout(20) # twenty seconds should be enough

try:
    # Attempt to load the URL in the browser
    driver.get(url)
except TimeoutException:
    # If the page load times out, stop loading the page
    driver.execute_script("window.stop();")
```

Final Answer:

In [7]:

```
# Initializing an empty list to store page URLs
pages = []

# loop until we reach the last page of AllPages site
while True:
    try:
        time.sleep(0.2)

        # Parsing the current page's HTML source
        html = lx.fromstring(driver.page_source)

        # Extracting the <ul> element containing page links
        ul_element = html.xpath('//div[@class="mw-allpages-body"]/ul[@class="mw-allpages-chunk"]')[0]

        # Extracting links from <li> items excluding redirections
        li_items = ul_element.xpath('./li[not(@class="allpagesredirect")]/a/@href')

        # Extending the list of pages with the extracted links
        pages.extend(li_items)

        div = driver.find_element("xpath", '//*[@id="mw-content-text"]/div[4]')

        # Finding the <a> tag to click on it to go to the next page
        a_tags = div.find_elements("xpath", './/a[contains(text(), "මීළඟ පිටුව")]')[0]
        a_tags.click()

    except:
        break
```

```
print(f"Number of articles after skipping Redirections are: {len(pages)}")
```

```
Number of articles after skipping Redirections are: 24233
```

In [7]:

```
# Closing the Chrome webdriver instance after storing the required URLs
driver.quit()
```

Saving the results got so far in the file, because when kernel dies while running multi-threading, I can retrieve the results from pickle file

In [9]:

```
import pickle

# Saving list using pickle serialization
def save_list_with_pickle(lst, filename):
    with open(filename, 'wb') as file:
        pickle.dump(lst, file)

# Example usage
save_list_with_pickle(pages, 'pages_3.pickle')
```

In [2]:

```
# Load list using pickle
def load_list_with_pickle(filename):
    with open(filename, 'rb') as file:
        return pickle.load(file)

# Example usage
pages = load_list_with_pickle('pages_3.pickle')
print(pages[0])
```

```
/wiki/%22%E0%B6%91%E0%B6%B8%E0%B7%8A%E0%B6%B4%E0%B6%BA%E0%B6%BB%E0%B7%8A_%E0%B7%83%E0%B7%8A%E0%B6%A7%E0%B7%9A%E0%B6%A7%E0%B7%8A%22_%E0%B6%BA%E0%B6%B1_%E0%B6%B1%E0%B6%B8%E0%B7%99%E0%B7%84%E0%B7%92_%E0%B7%83%E0%B6%B8%E0%B7%8A%E0%B6%B7%E0%B7%80%E0%B6%BA
```

(b, i) Scan all articles in the sinhalese wikipedia and retrieve all links to other articles. Avoid links to special pages, images or the ones that point to another website. Only count the proper article for links that point to a specific section. Use regular expressions to manage these cases. (ii) Make sure to match redirections to their correct destiation article. To this end, find how wikipedia treats redirections and retrieve the true article. ***(Help: Try searching for 'uc davis' on en.wikipedia.org')*** **(iii) Use threading to request all articles and obtain all links to other articles.** ***(Attention: This takes about thirty minutes!)***

How many links to other articles are there?

Assigning an ID to each page

In [3]:

```
# Removing duplicate pages, if there are any
pages_set = list(set(pages))

pages_with_id = {}

# Iterating through unique pages and assigning IDs
for id, page in enumerate(pages_set):
    pages_with_id[page] = id
```

In [4]:

```
len(pages_set)
```

Out[4]:

```
24233
```

An example of how links are assigned an Id and stored in dictionary

In [5]:

```
count = 0
for page, id in pages_with_id.items():
    if count < 5:
        print(f"ID: {id} ----> {page}")
        count += 1
    else:
        break
```

```
ID: 0 ----> /wiki/%E0%B6%85%E0%B7%81%E0%B7%8A%E0%B7%80_%E0%B6%BD%E0%B7%8F%E0%B6%A9%E0%B6%
B8
ID: 1 ----> /wiki/%E0%B7%84%E0%B7%99%E0%B6%A2%E0%B6%B8%E0%B6%B1%E0%B7%92%E0%B6%BA
ID: 2 ----> /wiki/%E0%B6%B6%E0%B7%94%E0%B6%AF%E0%B7%8A%E0%B6%B0%E0%B7%92%E0%B6%B8%E0%B6%A
D%E0%B7%8A_%E0%B6%B1%E0%B7%92%E0%B6%BA%E0%B7%9D%E0%B6%A2%E0%B7%92%E0%B6%AD%E0%B6%BA%E0%B7
%8F
ID: 3 ----> /wiki/%E0%B6%9A%E0%B7%90%E0%B6%AD%E0%B6%BB%E0%B7%92%E0%B6%B1%E0%B7%8A,_%E0%B7
%80%E0%B7%92%E0%B6%BD%E0%B7%92%E0%B6%BA%E0%B7%9A%E0%B6%B1%E0%B7%84%E0%B7%93_%E0%B6%86%E0%
B6%AF%E0%B7%92%E0%B6%B4%E0%B7%8F%E0%B6%AF%E0%B7%80%E0%B6%BB%E0%B7%92%E0%B6%BA
ID: 4 ----> /wiki/%E0%B6%AD%E0%B7%99%E0%B6%BB%E0%B7%9A%E0%B7%83%E0%B7%8F_%E0%B6%B8%E0%B7%
80%E0%B7%94%E0%B6%AD%E0%B7%94%E0%B6%B8%E0%B7%92%E0%B6%BA
```

(b, i) Scan all articles in the sinhalese wikipedia and retrieve all links to other articles. Avoid links to special pages, images or the ones that point to another website. Only count the proper article for links that point to a specific section. Use regular expressions to manage these cases.

In [52]:

```
def get_all_links(page):
    article_links = []

    # Constructing the full valid URL
    url = "https://si.wikipedia.org/" + page

    # Sending a GET request to the URL
    response = requests.get(url)

    try:
        # Raising an exception if the request was not successful
        response.raise_for_status()
    except Exception as e:
        return None

    # Parsing the HTML content
    html = lx.fromstring(response.text)

    # Finding the div element with the specific ID
    body_div = html.xpath('//div[@id="bodyContent"]')

    # Checking if the div element was found
    if body_div:
        # Selecting all <a> tags inside the div element
        links = body_div[0].xpath('.//a/@href')
        for link in links:
            # Filtering out links that seem to fall under the Category of special pages,
images, Discussions, edits, etc.
            # (?<!org): Removes links containing "org" (for external URLs).
```

```
            # \/wiki\/: Checks "/wiki/" substring.
            # (?!.*:): Removes links with a colon (for non-article URLs).
            if re.search(r'(?<!org)\/wiki\/(?!.*:)', link):
                # [^#]: Substitutes any character after "#" with "".
                link = re.sub(r'#.*$', '', link)
                article_links.append(link)

        return list(set(article_links))

    else:
        print("Div element with id 'bodyContent' not found.")
        return None
```

(b ii) Make sure to match redirections to their correct destiation article. To this end, find how wikipedia treats redirections and retrieve the true article. ***(Help: Try searching for 'uc davis' on en.wikipedia.org')***

In [53]:

```
def handle_redirections(link, session):
    # Spliting the link to extract the string after '/wiki/'
    parts = link.split('/wiki/')
    string_after_wiki = parts[1]

    # Constructing the URL to check for redirection
    url = f"https://si.wikipedia.org/w/index.php?title={string_after_wiki}&redirect=no"

    # Sending a GET request to the URL
    response = session.get(url)

    try:
        response.raise_for_status()
    except Exception as e:
        return None

    # Parsing the HTML content
    html = lx.fromstring(response.text)
    try:
        # Finding the original page link in case of redirection
        original_page = html.xpath('//ul[@class="redirectText"]|//span[@class="mw-redirec
tedfrom"]')
        original_page_link = original_page[0].xpath('.//a/@href')
        link = original_page_link[0]
    except:
        # Returning the original link if no redirection is detected
        return link

    # Checking if the link is valid based on the requirements specified in (b,i)
    if re.search(r'(?<!org)\/wiki\/(?!.*:)', link):
        # [^#]: Substitutes any character after "#" with "".
        link = re.sub(r'#.*$', '', link)
        return link

    # Returning None if the link doesn't match the pattern
    return None
```

(b iii) Use threading to request all articles and obtain all links to other articles. ***(Attention: This takes about thirty minutes!)***

In [54]:

```
thread_local = threading.local()

def get_session():
    '''Create a new requests.Session if there is none in thread_local'''
    if not hasattr(thread_local, "session"):
```

```
        thread_local.session = requests.Session()
    return thread_local.session
```

Creating Connection pairs (list of tuples) between an article to other articles

In [55]:

```
def create_pairs(page):
    session = get_session()

    pairs = []

    # Geting the ID of the specified page from the pages_with_id dictionary
    page_id = pages_with_id.get(page)
    if page_id is None:
        return pairs  # No need to proceed if page ID is not available

    # Retrieve all links from the specified page
    articles = get_all_links(page)
    if articles:
        for article in articles:
             # Geting the ID of the linked article from the pages_with_id dictionary
            article_id = pages_with_id.get(article)

            # If the ID of the linked article is not available, then it's a redirection o
r the article link is not valid
            if article_id is None:
                # Handling redirections to get the correct article ID
                article = handle_redirections(article, session)
                article_id = pages_with_id.get(article)

            # If the ID of the linked article is available and it's not the same as the c
urrent page ID
            if article_id is not None and page_id != article_id:
                pairs.append((page_id, article_id))

    return pairs
```

Function to perform concurrent threading

In [56]:

```
def perform_concurrent_threading(pages):
    # Defining the total number of tasks
    total_tasks = len(pages)

    # Creating a progress bar using tqdm
    with tqdm(total=total_tasks, desc="Processing pages") as progress_bar:
        # Using ThreadPoolExecutor with tqdm to create a progress bar
        with concurrent.futures.ThreadPoolExecutor(max_workers = 12) as executor:
            # Using executor.map() to apply the function to each page
            answers = list(tqdm(executor.map(create_pairs, pages), total=total_tasks, de
sc="Processing pages", position=0))
            progress_bar.update(total_tasks)

    return answers
```

In [57]:

```
answers = perform_concurrent_threading(pages_set)
```

```
Processing pages: 100%|██████████████████| 24233/24233 [21:15<00:00, 18.99it/s]
Processing pages: 100%|██████████████████| 24233/24233 [21:16<00:00, 18.99it/s]
```

Final Answer:

In [58]:

```
sum_of_lengths = sum(len(lst) for lst in answers)
print(f"The number of links to other articles: {sum_of_lengths}")
```

```
The number of links to other articles: 340684
```

(c) Compute the transition matrix (see here and here for step-by-step instructions). Make sure to tread dangling nodes. You may want to use:

```
from scipy.sparse import csr_matrix
```

In [59]:

```
# List to store dangling nodes
dangling_nodes = []

# Initializing row counter
row = 0

# I iterate through list of tuples which give position (row and column values) to mark "1
" in matrix
size = len(pages_set)
transition_matrix = np.zeros((size, size))
for lst in answers:
    # This means this particular article has connections to other articles
    if len(lst) > 0:
        for pairs in lst:
            # Calculating the probability for each pair and updating the transition matr
ix accordingly
            transition_matrix[pairs[0]][pairs[1]] = 1/len(lst)
    # This means this article had no outgoing links, therefore should be stored in dangli
ng nodes list
    else:
        dangling_nodes.append(row)
    row = row + 1

# converting the matrix to csr_matrix
transition_matrix = csr_matrix(transition_matrix)
```

Transition Matrix Before Handling Dangling Nodes

In [60]:

```
print(transition_matrix.toarray())
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Transition Matrix After Handling Dangling Nodes

In [61]:

```
print(f"Number of Dangling nodes: {len(dangling_nodes)}")
```

```
Number of Dangling nodes: 5465
```

substituting Entire Row of Dangling Node with mean of the number of articles (1/(number_of_articles))

In [62]:

```
# substituting Entire Row of Dangling Node with mean of the number of articles (1/(number
_of_articles))
transition_matrix[dangling_nodes, : ] = 1/size
```

Final Answer:

In [64]:

```
print(transition_matrix.toarray())
```

```
[[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 ...
 [4.12660422e-05 4.12660422e-05 4.12660422e-05 ... 4.12660422e-05
  4.12660422e-05 4.12660422e-05]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
```

(d, i) Set the damping factor to `0.85` and compute the PageRank for each article, using fourty iterations and starting with a vector with equal entries. (ii) Obtain the top ten articles in terms of PageRank, and, retrieving the articles again, find the correponding English article, if available.

Return the corresponding English article titles of the top ten articles from the Sinhalese wikipedia.

(d, i) Set the damping factor to `0.85` and compute the PageRank for each article, using fourty iterations and starting with a vector with equal entries

Step 1: Calculating Google Matrix

In [66]:

```
damping_factor = 0.85
additional_matrix = csr_matrix(np.full((size, size), 1/size))

google_matrix = damping_factor * transition_matrix + (1 - damping_factor) * additional_m
atrix
google_matrix = csr_matrix(google_matrix)
```

In [67]:

```
google_matrix.shape
```

Out[67]:

```
(24233, 24233)
```

In [68]:

```
print(f"Google Matrix:")
print(google_matrix.toarray())
```

```
Google Matrix:
```

```
[[6.18990633e-06 6.18990633e-06 6.18990633e-06 ... 6.18990633e-06
  6.18990633e-06 6.18990633e-06]
 [6.18990633e-06 6.18990633e-06 6.18990633e-06 ... 6.18990633e-06
  6.18990633e-06 6.18990633e-06]
 [6.18990633e-06 6.18990633e-06 6.18990633e-06 ... 6.18990633e-06
  6.18990633e-06 6.18990633e-06]
 ...
 [4.12660422e-05 4.12660422e-05 4.12660422e-05 ... 4.12660422e-05
  4.12660422e-05 4.12660422e-05]
 [6.18990633e-06 6.18990633e-06 6.18990633e-06 ... 6.18990633e-06
  6.18990633e-06 6.18990633e-06]
 [6.18990633e-06 6.18990633e-06 6.18990633e-06 ... 6.18990633e-06
  6.18990633e-06 6.18990633e-06]]
```

Step 2: Calculating Initial Vector with equal entries resembling that initially all pages are considered to have equal rank

In [69]:

```
initial_vector = np.array([1/size] * size)
initial_vector.shape
```

Out[69]:

```
(24233,)
```

In [70]:

```
print(f"Initial Vector:")
print(initial_vector)
```

```
Initial Vector:
[4.12660422e-05 4.12660422e-05 4.12660422e-05 ... 4.12660422e-05
 4.12660422e-05 4.12660422e-05]
```

Step 3: Calculation Page Ranks over 40 iterations

In [71]:

```
def pagerank(google_matrix, max_iterations=40):
    # Number of Proper Articles in wikipedia
    n = transition_matrix.shape[0]

    # Starting vector with equal entries
    initial_vector = np.array([1/n] * n)


    for _ in tqdm(range(max_iterations)):
        '''V(K + 1) = V(K) * G'''
        pagerank_vector = initial_vector * google_matrix

        '''V(K + 1) = V(K)'''
        initial_vector = pagerank_vector

    # Calculating the PageRank vector
    return pagerank_vector

# Final Ranking of the pages after doing 40 iterations
pagerank_vector = pagerank(google_matrix)
```

```
100%|██████████████████████████████████████████| 40/40 [00:13<00:00,  3.07it/s]
```

Final PageRank Vector

In [81]:

```
print(pagerank_vector)
```

```
[5.23284799e-06 4.08998207e-06 4.08998207e-06 ... 6.45449396e-06
 2.42784814e-05 1.33552957e-05]
```

Assigning Ranks to articles

In [85]:

```
# Sort the indices in descending order based on pagerank values
sorted_indices = np.argsort(pagerank_vector)[::-1]

# Assign ranks to pagerank vector as integers
ranked_pagerank_vector = np.empty_like(pagerank_vector, dtype=int)
ranked_pagerank_vector[sorted_indices] = np.arange(1, len(pagerank_vector) + 1)

print(ranked_pagerank_vector)
```

```
[13017 18932 18933 ... 10168  2906  5081]
```

In [78]:

```
final_pageRank_vector = [pagerank_vector[index] for index in np.argsort(pagerank_vector)
[-10:][::-1]]
print(final_pageRank_vector)
```

```
[0.00533969560192397, 0.0037821329233922382, 0.003135997399567751, 0.002754614849192115,
0.002447542517285014, 0.0023573941358048384, 0.0022774825103400657, 0.00226968122701788,
0.0021472083866992436, 0.0017623414164594667]
```

(d ii) Obtain the top ten articles in terms of PageRank, and, retrieving the articles again, find the correponding English article, if available.

In [72]:

```
# Sorting the pageRanks in descending order and getting Top 10 indices so that I can late
r retrieve article links using
# these Indices from pages_with_id Dictionary
top_indices = np.argsort(pagerank_vector.flatten())[::-1][:10]
top_indices
```

Out[72]:

```
array([13584, 13155,  9768, 10950,  6549, 10328,  3278, 18458, 18301,
       12592])
```

In [73]:

```
# mapping top 10 indices from above to their respective article links
top_10_articles_dict = {}

def get_article_url(dictionary, value):
    for key, val in dictionary.items():
        if val == value:
            return key
    return None  # If the value is not found in the dictionary

for index in top_indices:
    key = get_article_url(pages_with_id, index)
    top_10_articles_dict[index] = key

top_10_articles_dict
```

Out[73]:

```
{13584: '/wiki/%E0%B7%81%E0%B7%8A%E2%80%8D%E0%B6%BB%E0%B7%93_%E0%B6%BD%E0%B6%82%E0%B6%9A%
```

```
E0%B7%8F%E0%B7%80',
 13155: '/wiki/%E0%B7%80%E0%B7%9A%E0%B6%B6%E0%B7%90%E0%B6%9A%E0%B7%8A_%E0%B6%B8%E0%B7%90%
E0%B7%81%E0%B7%92%E0%B6%B1%E0%B7%8A',
 9768: '/wiki/%E0%B6%A2%E0%B7%8F%E0%B6%AD%E0%B7%8A%E2%80%8D%E0%B6%BA%E0%B6%B1%E0%B7%8A%E0
%B6%AD%E0%B6%BB_%E0%B7%83%E0%B6%B8%E0%B7%8A%E0%B6%B8%E0%B6%AD_%E0%B6%B4%E0%B7%9C%E0%B6%AD
%E0%B7%8A_%E0%B6%85%E0%B6%82%E0%B6%9A%E0%B6%BA',
 10950: '/wiki/%E0%B7%80%E0%B7%9A%E0%B6%BD%E0%B7%8F_%E0%B6%9A%E0%B6%BD%E0%B7%8F%E0%B6%B4'
,
 6549: '/wiki/%E0%B7%81%E0%B7%8A%E2%80%8D%E0%B6%BB%E0%B7%93_%E0%B6%BD%E0%B6%82%E0%B6%9A%E
0%B7%8F%E0%B7%80%E0%B7%9A_%E0%B7%83%E0%B6%B8%E0%B7%8A%E0%B6%B8%E0%B6%AD_%E0%B7%80%E0%B7%9
A%E0%B6%BD%E0%B7%8F%E0%B7%80',
 10328: '/wiki/%E0%B6%91%E0%B6%9A%E0%B7%8A%E0%B7%83%E0%B6%AD%E0%B7%8A_%E0%B6%BB%E0%B7%8F%
E0%B6%A2%E0%B6%B0%E0%B7%8F%E0%B6%B1%E0%B7%92%E0%B6%BA',
 3278: '/wiki/%E0%B6%AF%E0%B7%92%E0%B6%BA%E0%B6%AB%E0%B7%92%E0%B6%BA',
 18458: '/wiki/%E0%B6%91%E0%B6%9A%E0%B7%8A%E0%B7%83%E0%B6%AD%E0%B7%8A_%E0%B6%A2%E0%B6%B1%
E0%B6%B4%E0%B6%AF%E0%B6%BA',
 18301: '/wiki/%E0%B6%AF%E0%B7%92%E0%B6%BA%E0%B6%AB%E0%B7%92%E0%B6%BA_(%E0%B6%B6%E0%B7%84
%E0%B7%94%E0%B6%BB%E0%B7%94%E0%B6%AD%E0%B7%8A%E0%B7%84%E0%B6%BB%E0%B6%AB%E0%B6%BA)',
 12592: '/wiki/%E0%B7%83%E0%B6%B8%E0%B7%8F%E0%B6%BA%E0%B7%9D%E0%B6%AD_%E0%B7%83%E0%B7%8F%
E0%B6%BB%E0%B7%8A%E0%B7%80%E0%B6%AD%E0%B7%8A%E2%80%8D%E0%B6%BB_%E0%B7%80%E0%B7%9A%E0%B6%B
D%E0%B7%8F%E0%B7%80'}
```

In [74]:

```python
# Function to get English Article Names
def english_title(link):
    response = requests.get(link)
    try:
        response.raise_for_status()
    except Exception as e:
        print("Error:", e)
        return None

    # Parse the HTML content
    html = lx.fromstring(response.text)

    # Find the div element with the specific ID
    title = html.xpath('//*[@id="firstHeading"]/span')
    return title[0].text
```

In [75]:

```python
# Function to map articles links present in Sinhala Wikipedia to English Wikipedia
def get_article_name(link):
    url = "https://si.wikipedia.org/" + link
    response = requests.get(url)
    try:
        response.raise_for_status()
    except Exception as e:
        print("Error:", e)
        return None

    # Parsing the HTML content
    html = lx.fromstring(response.text)

    # Finding the div element with the specific ID
    title = html.xpath('//*[@id="firstHeading"]/span')

    english_title_link = html.xpath('//*[@id="p-lang-btn"]/div/div/ul/li[(@class="interla
nguage-link interwiki-en mw-list-item")]/a/@href')

    # Getting the title of the English Wikipedia article if the link is found
    if len(english_title_link) > 0:

        title = english_title(english_title_link[0])
        return title, 1
```

```
    return title[0].text, 0
```

Final Answer:

In [76]:

```
# English Names of the top 10 Articles:
rank = 0
for id, link in top_10_articles_dict.items():
    rank = rank + 1
    english_titles, found = get_article_name(link)
    if found != 0:
        print(f"top #{rank} ----> {english_titles}")
    else:
        print(f"top #{rank} ----> English title not present, Title Name in Sinhala Wikip
edia is {english_titles}")
```

```
top #1 ----> Sri Lanka
top #2 ----> Wayback Machine
top #3 ----> ISBN
top #4 ----> Time zone
top #5 ----> Sri Lanka Standard Time
top #6 ----> United Kingdom
top #7 ----> Daughter
top #8 ----> United States
top #9 ----> English title not present, Title Name in Sinhala Wikipedia is දියණිය (බහුරුත් හරණය)
top #10 ----> Coordinated Universal Time
```

Acknowledgment

I received assistance from `ChatGPT` while working on certain questions in this notebook. I want to clarify that I independently completed the majority of the tasks, seeking help only in instances where I encountered challenges or felt lost. The collaboration with ChatGPT was instrumental in providing guidance and insights during those moments. ChatGpt: https://chat.openai.com/

--- Nikita Bhrugumaharshi Emberi

References:

1) https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Pagerank5.pdf

In [ ]:

```

```