HOMEWORK ASSIGNMENT #1 – XPATH AND CRAWLING

MANAGEMENT OF BIG WEB DATA (FALL 2022-23)

GENERAL INSTRUCTIONS

- The work can be done in pairs or in singles.
- For the purpose of submission, your **username** will be <first_name>.<last_name>, or <first_name_of_member1>.<last_name_of _member1>.<first_name_of_member2>.<last_name_of _member2> for a pair. E.g., alice.smith or alice.smith.bob.walker
- Your solution should be submitted in a single zip file named **<username>-hw1.zip**, through Moodle. Only one member of the pair should submit.
- The zip will include an answers document in pdf format, named **<username>-hw1.pdf** make sure to write your name(s) on the top. The answers can be written in English or Hebrew.
- For every question that requires writing code, code should be written in python version 3.XX and submitted in a **.py** file. Keep the code readable, as it will be graded.
- You are expected to submit an original work. Copying from references (e.g. existing crawlers) or looking at others' solutions is not acceptable. Do not publish your solution or save it in any public place (specifically, public GitHub projects) – in cases of cheating, all involved students will be equally held liable.

TASKS

1. Consider the Wikipedia pages (in the English version, en.wikipedia.org) of members of the British royal family, e.g., https://en.wikipedia.org/wiki/Charles_III. Look at the HTML sources of the pages and try to write **XPath expressions** (at most 10) with
    - **Input:** HTML contents of a Wikipedia page (in the English version, en.wikipedia.org) of any member of the British royal family, e.g., https://en.wikipedia.org/wiki/Charles_III
    - **Output:** strings representing URLs of *Wikipedia pages of other members of the royal family*. We define a member as any British king/queen and all of their descendants.

    The XPath expressions must be *legal* (check yourselves using python code), but they may not be *perfect*: you may miss some links (false negatives) or extract some wrong links (false positives). Try to write the best expressions you can find, to minimize false positives and negatives and account for different page formats. **Provide the XPath expressions in the answers file** and for each XPath expression give **one example** of a source URL and the output URL that was found. Mention next to each expression if it captures **descendants** of the current person (children, grandchildren, etc.), **ancestors** of the current person (parents, grandparents, etc.) or **other** (both/neither/unknown). Try to have at least one expression that detects descendants and one that detects ancestors.

For example:
The XPath "//a/@href" is too general. It finds 4119 links only in
https://en.wikipedia.org/wiki/Charles_III

The XPath "//a[contains(text(),'mother')]/@href" is better, and for
https://en.wikipedia.org/wiki/Charles_III returns the URL of
/wiki/Queen_Elizabeth_The_Queen_Mother
(relative URL, actually pointing to
https://en.wikipedia.org/wiki/Queen_Elizabeth_The_Queen_Mother)
However, it also returns pages that are not of royal members, e.g.,
/wiki/Coronation_of_Elizabeth_II and of course misses most of the royal member links.

So, in the answers file you could write:

//a[contains(text(),'mother')]/@href (**ancestor**) Example: given https://en.wikipedia.org/wiki/Charles_III the XPath returns /wiki/Queen_Elizabeth_The_Queen_Mother

Tips:

- Check the structure of the page to extract links from specific parts of the page. You can use source viewers in your browsers (e.g., "inspect" option in Google Chrome). Use only the minimum constraints necessary, e.g., //v/a/@href is better than /html/body/x/y/z/w/u/v/a/@href
- Check the URL structure to extract **only links inside en.wikipedia**
- Check textual context to find only people
- Check different example pages, to make sure your expressions are general enough

2. Write an Xpath expression that, given a Wikipedia page (in the English version, en.wikipedia.org), verifies that this page represents a member of the royal family. The Xpath will return a non-empty result if the page represents a member of the royal family, and an empty result if the page does not represent one.

3. Write a function `def britishCrawler(url, verifyXpath, descendantXpaths, ancestorXpaths, royaltyXpaths)` whose purpose is to crawl pages of the royal family in Wikipedia
    a. The input `url` is a string containing the URL a start pages (e.g., https://en.wikipedia.org/wiki/Charles_III)
    b. The input `verifyXpath` is a string representing a single legal Xpath expression
    c. The inputs `descendantXpaths, ancestorXpaths, royaltyXpaths` are (possibly empty) lists of strings representing legal XPath expressions
    d. The function will use `verifyXpath` to filter out pages that are not Wikipedia pages of members of the British royal family. If `verifyXpath` is null, no page will be filtered out.

For the remaining pages, it will use `descendantXpaths` (expressions detecting pages of children, grandchildren etc.), `ancestorXpaths` (expressions detecting pages of parents, grandparents etc.) and `royaltyXpaths` (expressions detecting other pages of royalty) to extract a set of URLs that match at least one of the expressions (make sure to support null/empty lists.

e. These URLs will be crawled in order of priority:
    i. The priority of a URL – crawl pages of people with **less descendants first**. For that, keep for each URL a set of its descendants detected so far (based on `descendantXpaths` and `ancestorXpaths`), and keep updating it. For example, assume the crawler started from the page of Charles III and found a link to William using `descendantXpaths`, a link to Elizabeth II using `ancestorXpaths` and a link to Anne using `royaltyXpaths`. It knows at this point of two descendants of Elizabeth, one of Charles and none of Anne and William, so it may crawl either Anne or William.
    ii. Note that two links to the same URL from the same page count as one.
    iii. Use a deterministic (=non-random) rule of your choice to **break ties**, for example, prefer URLs with more links or that were discovered first. The rule is deterministic so that every execution of your crawler on the same input should return the same output.
f. Mind **crawling ethics** and particularly, wait at least 3 seconds between page reads.
g. In total, crawl (download and analyze) at most 30 URLs, and only URLs of en.wikipedia.org. Even if a downloaded page does not match `verifyXpath`, it still counts as a crawled page.
   Avoid crawling the same URL twice!
h. The function will return **a list of lists**. Each inner list will contain two strings and a number: the first is a source URL whose contents match `verifyXpath.` The second is each URL of a page detected in the source URL by the crawler. Convert relative URLs to full ones. The last number is 0 if the destination page was not crawled yet and 1 if it was crawled and matches `verifyXpath.` Discard links to URLs that were crawled and that do not match `verifyXpath`. For example,

```
['https://en.wikipedia.org/wiki/Charles_III', 'https://en.wikipedia.org/wiki/King_George_VI', 0]
```

   URLs may repeat both in the source and the destination URL, but there should not be two identical pairs (same source and destination).
i. Submit the code in the file `britishCrawler.py`

Tip:

- You can start from the simple crawler example from class, and change as needed.

4. Evaluate the quality of your crawler using the precision, recall and F1 metrics we learned in class.
    a. Write a function `def crawlerQuality(listOfPairs)`
    b. `listOfPairs` is a list of lists in the format of the output of `britishCrawler` from question 3.
    c. The function will return a dictionary where the keys are names of quality metrics (out of the ones we learned in class) with their values, e.g., `{'precision': 0.9, 'recall': 0.7 …}`
    d. Submit the code in the file `crawlerQuality.py`
    e. In the answers file, write which metrics you have implemented and how; which metrics you could not compute, if there were any, and ideas on what could be done to compute or estimate them in our specific scenario.
    f. Execute your crawler from question 3 over the Xpaths in questions 1-2, and use the crawlerQuality function to compute the output quality. Write the inputs and outputs of `britishCrawler` and `crawlerQuality` in the answers file.
    g. Execute the same functions again, but this time use a null value for `verifyXpath.` Write the inputs and outputs of `britishCrawler` and `crawlerQuality` in the answers file, and explain what metrics changed, if at all, and why.