

HOMework ASSIGNMENT #1 – XPATH, CRAWLING AND RANKING

MANAGEMENT OF BIG WEB DATA (FALL 2019-20)

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> in the case of 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. Try to 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 Do not publish your solution or save it in any public place – in cases of cheating, all involved students will be equally held liable.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 professional tennis players, e.g., https://en.wikipedia.org/wiki/Andy_Ram. 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 tennis player, e.g., https://en.wikipedia.org/wiki/Andy_Ram,
 - **Output:** URLs of *Wikipedia pages of related tennis players*. We define a related tennis player as one who played a match with the current player or was the coach of the current player.

The XPath expressions must be *legal* (check yourselves using python code), but they do not need to 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. **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.

For example:

The XPath `“//a/@href”` is too general. It finds 1898 links only in https://en.wikipedia.org/wiki/Andy_Ram

The XPath `“//a[contains(text(),'Jonathan')]/@href”` is better, and for https://en.wikipedia.org/wiki/Andy_Ram returns the URL of Andy's partner `/wiki/Jonathan_Erich` (relative URL, actually pointing to https://en.wikipedia.org/wiki/Jonathan_Erich)
However, it also returns pages of unrelated players, e.g., `/wiki/Jonathan_Stark_(tennis)`

Pages outside Wikipedia http://www.protennisfan.com/2006/03/andy_ram_and_jo.html

And of course misses most of the related players (the XPath has to work for *every* tennis player!).

So, in the answers file you could write:

```
//a[contains(text(),'Jonathan')]/@href
```

Example: given https://en.wikipedia.org/wiki/Andy_Ram the XPath returns
/wiki/Jonathan_Erich

Tips:

- Look at the structure of the page to extract only links from certain parts of the page. You can use source viewers in your browsers (e.g., “inspect” option in Google Chrome)
- Look at the URL structure to extract **only links inside en.wikipedia**
- Look at textual context to find only names of players
- Look at a few example pages, to make sure your expressions are general enough

2. Write a function `def crawl(url, xpaths)` whose purpose is to crawl pages of tennis players in Wikipedia

- The input `url` is a string containing the URL of the start page (e.g., https://en.wikipedia.org/wiki/Andy_Ram)
- The input `xpaths` is a list of strings representing legal XPath expressions
- The function will use the `xpaths` to extract a set of URLs from the web page.
- These URLs will also be crawled in order of priority:
 - Keep counts for the number of times each URL was found. URLs seen the highest number of times have the highest priority.
 - Break ties as you choose.
- Mind crawling ethics as learned in class and particularly, wait at least 3 seconds between page reads.
- In total, at most 100 URLs will be crawled in this manner, and only URLs of `en.wikipedia.org`. avoid crawling the same URL twice!
- The function will return **a list of lists**. Each inner list will contain two strings: the first will be the *full* source URL, and the second will be the *full* URL of a page detected in the source URL by the crawler. For example, if `/wiki/Jonathan_Erich` was extracted from the page https://en.wikipedia.org/wiki/Andy_Ram, the output will contain the list

```
[ 'https://en.wikipedia.org/wiki/Andy_Ram', https://en.wikipedia.org/wiki/Jonathan_Erich' ]
```

The output should not contain repeated pairs, even if some link appears in the source page more than once.

- Submit the code in the file `Q2.crawl.py`

Tip:

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

3. We will compute a PageRank score for tennis players.

- Write a function `def playerPageRank(listOfPairs)`
- `listOfPairs` is a list of lists in the format of the output of `crawl` from question 2.

- c. The function will treat each inner list [X, Y] as a link from X to Y.
- d. We will use the random surfer model:
 - i. The graph nodes are all the URLs in the input
 - ii. Start from a random node
 - iii. At each step, the surfer decides with probability 0.85 to follow a link or with probability 0.15 to jump to a random tennis player, chosen uniformly at random from all the URLs in the input. If there are no outgoing links, always jump to a random member.
- e. Repeat this process 200,000 step. Record the number of times you have visited each page in the first 100,000 steps, and in the last 100,000 steps.
- f. The PageRank of each member is the number of times it was visited divided by the number of steps.
- g. The function returns a dict where the keys are URLs and the values are the scores computed from the first and last 100,000 steps , e.g., {'https://en.wikipedia.org/wiki/Andy_Ram': [0.1, 0.09], 'https://en.wikipedia.org/wiki/Jonathan_Erlich': [0.05,0.06] ...}
- h. Submit the code in the file *Q3.playerPageRank.py*
- i. In the answers file, write: what was the page with highest PageRank you got when the function was executed on the output of `crawl('https://en.wikipedia.org/wiki/Andy_Ram',xpaths_from_question1)?`
Was the page with the highest score for the first 100,000 steps different from the best page for the last 100,000 steps? Explain briefly.

Tips:

- Use the random package.