note @120 25 views

HW3 - Hints and Pseudocode

Following are few hints that could help you understand the workflow of the crawler and how the different components work together. You may choose to use these hints or follow your own methods.

HTML Parser (Python):

Libraries used: BeautifulSoup, requests (urllib2 can also be used instead of requests)

1) Fetch the content of the url using a requests session object.
2) Create a beautiful soup object for the content fetched.
3) Check for redirection. If it exists, you may choose to either skip it or choose the most recent redirected version; latter recommended.
4) Check if the url is html; use content-type for this.
5) Fetch the outgoing links by scraping for <a> tags.
6) Canonicalize these outgoing links using the rules mentioned in the assignment.
6) If necessary, fetch other stuff like title, header information and raw html.
7) Fetch relevant content of the url (not menu text, table information, image caption, etc); try finding which tags contribute to content in most urls.
8) Maybe preprocess the content which could include removing punctuation, newline characters, trimming for extra whitespaces.

URL Canonicalization:

1) These are just rules applied to the urls to normalize them.
2) Make sure to have test cases to check if your canonicalization is done right.
3) Few edge cases: (i) 'https://www.abc.com' and 'http://www.abc.com' are the same.
(ii) Many out links in Wiki pages have links as '/wiki/SomeText' which is the same as 'www.wikipedia.org/wiki/SomeText'
(iii) There maybe more edge cases that could be unique to the urls you crawl, so make sure you identify them and include it in your canonicalizer.

Robots.txt parser:

1) Use a robots.txt parser library to fetch: (i) whether the url can be crawled (ii) how long to sleep to crawl the same url again?
2) Every domain has a robots.txt so it would be useful to cache this information.

Data Structures for the crawler (you are free to use other structures as well):

1) Frontier: built-in/custom queue or max/min priority queue or bucket structure
2) Link Graph: dictionary/hash map or custom class; would contain inlinks, outlinks, relevance score, wave number for each url

Basic workflow of crawler (Python):

initialize frontier with seed urls # frontier is a queue of link_graph objects
visited_urls = []

while len(visited_urls) != 40000:

```
current_url = frontier[0].url # fetching the url of the link_graph object at index 0 of the frontier

if current_url has already been crawled:
    continue
else:
    if robot_parser.can_crawl(current_url):
        sleep_time = robot_parser.get_sleep_time(current_url) # sometimes this may return None in which case you can set it to 1 by default
        sleep(sleep_time)
        content = htmlparser.get_content(current_url)
        outlinks = htmlparser.get_outlinks(current_url) # canonicalization is done as a part of get_outlinks()
        for each outlink in outlinks:
            # create a link_graph object for the outlink
            frontier.push(link_graph object)
        visited_url.append(current_url)
        frontier.pop(0)
if some_condition:
    frontier.partial_sort()
    # Basic idea behind the partial sort (can be implemented in different ways):
    # 1) Select and remove those urls from the frontier belonging to the highest wave (lowest wave number) as we are following BFS to traverse the frontier. Assuming seed urls form
    # wave 1, seed urls' outgoing links form wave 2 and so on. Wave here represents a level in the graph.
    # 2) Sort these urls based on their score in decreasing order.
    # 3) Reinsert the sorted urls into the beginning of the frontier.
```

Scoring:

1) Every time you create a link graph object for a url, you give score it a score as well.
2) Score can be calculated based on the parameters mentioned in the homework under frontier management.
3) Popular parameters used: wave number, inlink count, keywords matching the url
4) You can come up with your own formula to calculate the score using these parameters.

Partial Sorting:

1) When to sort your frontier? # the 'if some_condition' part in the pseudo-code above
(i) When you have crawled a particular number of urls.
(ii) When you have added a particular number of outlinks to the frontier.
Note: These values usually scale exponentially as your crawler progresses, so you may want to update this threshold accordingly.
2) It is imperative you follow BFS to traverse the frontier so, make sure you're sorting the right urls while partial sorting.
3) If you're finding it hard to implement the above strategy mentioned for partial sorting in the pseudo-code on queues, you can use a bucket structure strategy. Do visit us during TA hours to know more.

Writing Documents:

1) This is similar to the AP dataset in the first 2 HWs. Based on the information you scrap from the url, you write them into a document in the format mentioned in the homework.
2) Try not to write to a file every time you finish crawling a url as I/O operations are time-consuming operations. Do it in batches.

Tips for better and faster crawls:

1) Run multiple test crawls and have a look at the urls you are crawling and also the content of these pages.
2) Create blacklists that you can use to ignore bad urls and urls with bad content.

3) Sometimes beautiful soup may give you empty content, make sure to avoid those urls.

4) Constantly save the state of your variables because in case your internet or computer crashes decides to crash, you don't have to start from scratch. You can just restart using the saved variables. Pickling is a common way to save your variables in Python. You could also do this when your program crashes for unknown errors you encounter during the crawl.

5) It's only necessary to sleep when the previous domain you crawled is the same domain as current domain you're crawling. In case you aren't able to keep track of the previous domain, you would have to sleep on every crawl like shown in the pseudo-code.

6) Some robots.txt may ask you to sleep for a 'long' time. This may halt your crawl a lot. In this case, you can have a check to see if the sleep time is 'long' and ignore the url or start another multi-threaded process to continue your crawl.

hw3

~ An instructor (Raaghavv Devgon) thinks this is a good note ~

Updated 1 day ago by Amrutha Vempati and Ghita

followup discussions *for lingering questions and comments*