HomeWork-3 Report

1. **Crawling [Note down steps implemented for each of the below]**

    a. **Any changes in the seed URLs provided?**
        i. **Canonicalizing the seed urls before adding it to the frontier upon initialization of the WebCrawler.**
        ii. **Experimented with some other individual seed url options (espn.com, baseball-reference.com), however, in final version made no changes to the original seed url list.**

    b. **URL Canonicalization –**
        i. **Parsing the url**
        ii. **Extracting components (host, path, params)**
        iii. **Lower the host**
        iv. **Remove ports 80 or 443 if present**
        v. **Remove double slashes**
        vi. **Remove fragment**
        vii. **Standardize all urls to http (reduces duplication, redirects to https when necessary)**
        viii. **Used the urllib python library**
        ix. **Experimented with filtering query and params, and lowering entire url, however in final version to ensure I aligned with the group's canonicalization I removed these.**

    c. **Frontier Management**
        i. **A python priority queue, storing entries with wave number, priority score and next crawl time. This way, since the queue retrieves lower values first, the frontier crawls in a BFS wave number fashion, processing entire waves by priority and next crawl time before moving on.**
        ii. **Priority score is calculated using (Lower priority = better):**
            1. **keyword score (keyword matching within anchor text and url)**
            2. **Preferring certain preset domains**
            3. **Recency score (using last modified determine recency)**
            4. **Inlink score (the number of inlinks, dynamically updates inlink scores when more are found)**
        iii. **Next crawl time is calculated when adding the url to the frontier (by adding the delay, determined from default or robots.txt, to the current time), this way when we retrieve urls to be crawled we check this value to ensure we are complying with the crawl delay and politeness policy.**

d. **Politeness Policy**
    i. **Using the robotparser library, parsing every domain's robots.txt to comply with politeness policy, checking if we are allowed to fetch, and getting the specific delay time.**
    ii. **Robotparsers are cached to avoid having to re-process these when encountering the same domain several times.**

e. **Document Processing**
    i. **Document processing is done in two ways**
        1. **The first method is I directly index to an elastic cloud index, indexing documents by their canonical url as id with the following fields: url, content (same format as AP89), inlinks, outlinks, authors (lists). When I first index these documents the inlink list won't be final, therefore, when I encounter new inlinks I update the document in the index by adding the existing inlinks with the new ones.**
        2. **The second method is more rudimentary, simply storing the index locally in a dictionary, then saving the index to a pkl file, cleaning the index (removing entires with no content, because of inlink method), then looping through the index to store them.**
        3. **For the indexing and updating, I check if a document already exists, if it does I merge the existing document with the new one efficiently, ensuring not to overwrite any date, if one does not exist I simply index it.**

2. **Vertical Search (FOR CS6200 only)**
    a. **Add a Screenshot of your Vertical Search UI**
    b. **Explain briefly how you implemented it.**

3. **Extra Credits Done [Note down what was done for each extra credit]**
    a. **Crawling directly into the merged index**
    b. **Experimented with different frontier management techniques, for example using next crawl time before priority to prioritize politeness over self calculated score, dynamically updating queue when encountering new inlinks, etc**
    c. **Speed optimizations: threading, tried out async requests using asyncio, memory improvements (storing more things in ES), caching robotparsers, reducing multiple http requests, reducing duplicate methods.**