CISC-235 Data Structures W22
Assignment 3

March 3, 2022

How to Submit

You need to submit your python code for question 1 and your textual answer to question 2. You must comment your code. Name your folder as **your netid-A3**, zip the folder and upload it to OnQ. Check whether your code is runable before submitting.

1 Implementing a simple web search engine: 70 points

We decompose the task into three parts, they are connected. Thus we recommend you to read the whole question first and then start implementing components.

1.1 WebPageIndex: 20 points

Write a **WebPageIndex** class that will contain the index representation of a web page. This class will help you transfer a web page (text file in this assignment) into an AVLTreeMap (you can reuse the AVLTreeMap you implemented from A2), where each key refers to each word appearing in the document, and each value represents a list containing the positions of this word in the file.

Your WebPageIndex class should satisfy the following requirements (you can create more instance variables/functions if you want):

1) Implement a **__init__(self, file)** function that initilizes an instance of WebPageIndex from a txt file. Must have an attribute specifying the path of the file used for initilizing the WebPageIndex instance. Lower case all words appearing in the input file.

2) Implement a **getCount(s)** function that returns the number of times the word s appeared on the page.

3) Test your WebPageIndex class using data/doc1.txt.

1.2 WebpagePriorityQueue: 40 points

Your task is to write a heap-based implementation of a PriorityQueue named **WebpagePriorityQueue**. It should contain an array-based list to hold the data items in the priority queue. Your WebpagePriorityQueue should be a maxheap. The most relevant web page given a specific query will have the highest priority value so that this web page can be retrieved first.

Your WebpagePriorityQueue class should satisfy the following requirements (you can create more instance variables/functions if you want):

1) Must have an initilization function which takes a query (string) and a set of WebpageIndex instances as input. This function will create a max heap where each node in the max heap represents a WebpageIndex instance. The priority of a WebpageIndex instance is defined as **the sum of the word counts of the page for the words in the query**. For instance, given a query "data structures", and a WebpageIndex instance, the priority value of this web page should be: sum(number of times "data" appears in the page + number of times "structures" appears in the page).

2) Must contain an attribute specifying the query string that determines the current WebpagePriorityQueue.

3) Must have a **peek** function. Return the WebpageIndex with the highest priority in the WebpagePriorityQueue, without removing it.

4) Must have a **poll** function. Remove and return the WebpageIndex with the highest priority in the WebpagePriorityQueue.

5) Must have a **reheap** function which takes a new query as input and reheap the WebpagePriorityQueue. This is required as the priority value of each web page is query-dependent.

1.3 Implement a simple web search engine: 10 points

Create a python file named "processQueries.py". This script implements a simple web search engine.

First, implement a function named **readFiles**, which takes a folder path as input, and returns a list of WebPageIndex instances, representing the txt files in the given input folder.

In the main function, given a query file (each line represents one search query), create a WebpagePriorityQueue instance to process queries in the query file in a loop. For every subsequent search query, use the new query to reheap your collection of WebPageIndex instances in the created WebpagePriorityQueue instance. Then prints out the matching filenames in order from best to worst match until there are no matches(no words in the query appear in the txt file) or the user specified limit (e.g., return at most 3 matched files) is reached.

Test your web search engine using the provided data folder and query file.

2 Hashing for URL Retrieval: 30 points

Given 10,000 URL visit records, design a hashing-based method so that we can quickly retrieve URLs that are most frequently visited. Describe your design concerns in a txt file. Like how you would design the data structure and how you plan to process input and generate output.

For instance, your input is a file containing 10,000 rows, where each row contains: URLaddress,visit_timestamp

Note that one URLaddress can be visited multiple times.