CISC-235 Data Structures
Assignment 3

March 7, 2019

1 Problems

In this assignment, you will index the web and perform search queries using two data structures, i.e., AVL tree and MaxHeap. The goal of this assignment is to get familiar with the implementation of these data structures and create a simple web search engine.

1.1 Background

You probably use a search engine at least 10 times a day to dig through the vast amount of information on the web. But do you know how search engines store web pages and retrieve the most relevant web pages given a query? In this assignment, we'll be trying out one technique that is used by most search engines via **building an index of the words on a page**. For each word we encounter on a web page, we'll keep track of what order we encountered it in (0th, 1st, 2nd, 3rd, etc.) and keep a list of all the locations for each word (locations will help search engine locate words).

The query technique is related to the frequency of a query word on a given page. If you've got a page that has the word "data" occurring repeatedly, then it is quite likely to be about data. And a page that has 10 "data" is probably more relevant to a query on "data analysis" than a page that only contains 1 "data". Note that we are considering the very basic case. You can think of other ways to compute a relevance score that measures the similarity between the query and web page. For instance, normalized word count (divide by the total number of words in the web page).

Given the following paragraph of text: "A data structure is a way to store and organize data." If we go through and index the order of all the words (start from 0, lowercase), we will have Table 1.

1.2 AVLTreeMap: 50 points

Your first task is to implement a new data structure call **AVLTreeMap**. AVLTreeMap is just an AVL tree in disguise, where the nodes in the tree contain

Table 1: Table for Background Section

Word	Count	Positions
a	2	0,4
data	2	1,10
structure	1	2
is	1	3
way	1	5
to	1	6
store	1	7
and	1	8
organize	1	9

"key-value" pairs. You place items in the tree **ordered by their key**, and this key has an associated value tagging along with it. The key can be any reference type (e.g., integer or string), and so can the value (e.g., list).

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

1) Each node has fields left and right for subtrees, key and value for its key-value pairs, and height for adjusting tree structure.

2) Must have a **searchPath(key)** function that returns a list of all the keys visited on the search path.

3) Mush have a **get(key)** function that returns the value if the given key exists in the AVL tree. This function should be implemented in a recursive way. Return null if the key isn't in the tree, or the associated value if it is.

4) Must have a **put(key, value)** function (i.e., insert a new key-value pair). You may assume that the keys are integers or strings. Your put function should adjust the structure in order to maintain a valid AVL trees, i.e., perform rotations if needed and update height of each node if needed. Note that if the given key exists in the tree, you need to update the value. For example, previous you added 10(key)-david(value), now you want to add 10-bob, then you need to update the value with key 10 as bob.

Test: To test your put function, you can start with an empty tree and insert the key-value pairs: 15-bob, 20-anna, 24-tom, 10-david, 13-david, 7-ben, 30-karen, 36-erin, 25-david, 13-nancy. You can use your searchPath function and get function to confirm that the tree is correctly structured.

1.3 WebPageIndex: 20 points

Your second task is to 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, 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) Must have a constructor taking a file name as input.

2) Must have an instance variable specifying the corresponding file name.

3) Must convert all words to lower case.

4) Must have a **getCount(s)** function that returns the number of times the word s appeared on the page.

1.4 WebpagePriorityQueue: 50 points

Your second task is to write a heap-based implementation of a PriorityQueue namely **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 a constructor (i.e., _init_) which takes as input a query (string) and a set of WebpageIndex instances. This constructor 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 instance variable specify which query determines the current priority values for each WebPageIndex instances.

3) Must have a **peek** function. Return the highest priority (largest value) item in the WebpagePriorityQueue, without removing it.

4) Must have a **poll** function. Remove and return the highest priority item 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. Note that if the query is the same as the current one, you do not need to reheap the WebpagePriorityQueue.

1.5 Implement a simple web search engine: 20 points

Create a **ProcessQueries** class. This class implements a simple web search engine. The program has two basic parts. The first part is to build a list of WebPageIndex instances from a folder containing a set of web pages (txt files). The second part is to enter a loop to process a series of user queries. You need another file, i.e., query file, to save your queries, one query per line.

First, implement the part of your program that processes the txt files in the folder. For each file read in, use that file to construct a WebPageIndex instance. Put all of the WebPageIndex instances in a list. You should do this as a method and not just have all the code in main.

Second, you need to process the user queries using your implemented WebpagePriorityQueue. To find the results of the query in best-to-worst order, construct a WebpagePriorityQueue instance. The priority value should be computed by adding the counts of the words in the query. The WebpagePriorityQueue can be used to print out the names of matching files in order.

For every subsequent search query, use the new query to reheap your collection of WebPageIndex instances.

The program then prints out the matching filenames in order from best to worst match until there are no matches or the user specified limit is reached. Continue reading and processing queries in a loop until you reach end of query file.

Test: We will provide the input folder and query file.

1.6 Style: 10 points

You must comment your code.

2 Assignment Requirements

This assignment is to be completed individually (we have the right to run any clone detection tool). You need submit your documented source code (we don't need the input folder and query file). If you have multiple files, please combine all of them into a .zip archive and submit it to OnQ system. The .zip archive should contain your student number. Helper functions and classes are allowed as long as you write them by yourself. Deadline: Mar-18 12:00am, 2019.