CS 201 - Data Structures II:
Assignment #4

Muhammad Mobeen Movania (L1),
Syeda Saleha Raza (L2),
Faisal Alvi (L3, L4),
Abdullah Zafar (L5).

Due on April 20, 2024, 11.59pm

Student 1 Name, ID **Student 2 Name, ID**

Instructions

This assignment document consists of two problems.

- Problem 1 is a programming based question which requires implementation. It must be submitted by pushing all your code files to the Github repository. This problem is worth 40 points.
- Problem 2 is a theoretical question which requires analysis. It should be completed and submitted within this document as a pdf on Canvas. This problem is worth 20 points.

Problem 1

(40 points) [**Creating a Search Story Using Inverted Index**]
In this assignment you will be implementing an inverted index on a corpus of resumes taken from Resume Dataset - Kaggle and creating a story based on the data.

Understanding the Data Format

The data is divided into two folders called "text" and "html", containing text files and HTML files respectively. Each text file contains simplified/cleaned text that corresponds to an HTML file of the same name containing the original resume text. You need to use the text files to create an inverted index and perform queries on it. The HTML files are given to help you explore the data better.

Create a Positional Index

Create a positional index of the following format using all the data from the "text" folder. Table 1 shows an entry and some of its values in the positional index.

Filename	[DocID, Frequency in Doc: [pos1, pos2, …], weight=1 …]
⋮	⋮
inspired, 16	[30, 1: [3036], 1 51, 1: [1452], 1 1440, 2: [2697, 5123], 1 ⋮]

Table 1: Entry for the term "inspired", which appears a total of 16 times in all documents combined. The Doc. ID corresponds to the number appended to the last underscore in the filename. A default weight of 1 is added to each document entry as a placeholder to be replaced with a weight when implementing ranked retrieval.

A positional index allows proximity and phrase search of variable length. You will use the positional index to create a search story that incorporates both proximity and phrase search. In order to create the required index, you need to consider:

1. *Text Extraction:* The entire data needs to be tokenized over whitespace. Each token will be treated as a term. The terms will probably need to be collated in some form before being added to the inverted index.

2. *Choice of Data Structure:* The two common choices are hash tables and tries. Deciding whether to use one or the other depends on the specific requirements of your application. First understand all the requirements of this assignment, and then do your own research before deciding on the appropriate data structure to use. You are free to use any data structure to implement the inverted index for the purposes of grading.

Add Ranked Retrieval Functionality

The inverted index needs to support ranked retrieval, i.e. queried documents should be assigned scores and sorted in order of highest score to lowest score. The Term Frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF) are two commonly used metrics to evaluate how important a term is, but

Problem 1 continued on next page…

they differ significantly in how they measure this importance. You are only required to implement the TF-IDF metric on your terms. You may decide to use a different metric, such as TF, in the next part of the assignment.

Add Boolean and Proximity Search Functionality

Three types of boolean queries and two types of proximity queries, represented by strings, should be supported by your positional inverted index. These are:

1. t_1 AND t_2: returns the postings that match both terms t_1 and t_2.
2. t_1 OR t_2: returns the postings that match either terms t_1 or t_2.
3. NOT t: returns the postings that do not match term t.
4. t_1 BEFORE t_2: returns postings where term t_1 appears immediately before term t_2.
5. t_1 NEAR t_2n: returns postings where terms t_1 and t_2 appear within n positions of each other.

Furthermore, the inverted index needs to support the intersection and union of results obtained from boolean and proximity searches.

1. If q_1 and q_2 are queries, then $q_1 \cap q_2$ should return the intersection of the postings obtained by searches q_1 and q_2.
2. If q_1 and q_2 are queries, then $q_1 \cup q_2$ should return the union of the postings obtained by searches q_1 and q_2.

Add function specifications here

Required Tasks

Grading Criteria

The rubric for this homework is as follows:

- 10 points
- 15 points
- 15 points

Penalties

- (-5) Code compiles with warnings
- (-5) OOP Inheritance not applied correctly between AVL and BST
- (-5) Implementation does not match the provided function signature.
- (-5) GitHub repository does not follow appropriate structure. All the code files should be in the code folder and output files in the output folder.

Compilation Guidelines: Before proceeding with submission, kindly verify that your code compiles utilizing the **C++17** standard.

Problem 2

(20 points) [**Analysis: Create a Search Story**]
Once you have implemented the required functionality of the positional index, you need to create your own search story. Think of the search story as the description of a search task that you perform on the inverted index. Here are the steps you will need to undertake:

1. *Explore the dataset:*
2. *Identify a Task:*
3. *Choose 3 distinct Queries for your Task:*
4. *Choose and briefly justify your Scoring Metric:*
5. *Summarize Results:*

Due date: March 30, 2024, 11:59 PM