Assignment 6: Document Processing Tool

COL106: Data Structures and Algorithms, Semester-I 2023–2024

Submission Deadline: 27th October, 5:00 PM

Honor Code

Following is the Honor Code for this assignment.

- Copying programming code in whole or in part from another or allowing your own code (in whole or in part) to be used by another in an assignment meant to be done individually is a serious offence.
- The use of publicly available codes on the internet is strictly not allowed for this assignment.
- The use of ChatGPT and other AI tools is strictly forbidden for writing code for this assignment. If any student has been found guilty, it will result into disciplinary action.
- Collaboration with any other team would be construed as academic misconduct.
- Outsourcing the assignment to another person or "service" is a very serious academic offence and could result in disciplinary action.
- Sharing of passwords and login ids is explicitly disallowed in this Institute and any instance of it is academic misconduct. Such sharing only compromises your own privacy and the security of our Dept./Institute network.
- Please ensure that your assignment directories and files are well-protected against copying by others. Deliberate or inadvertent copying of code will result in penalty for all parties who have "highly similar" code. Note that all the files of an assignment will be screened manually or by a program for similarity before being evaluated. In case such similarities are found, the origin as well as destination of the code will be held equally responsible (as the software cannot distinguish between the two).

Introduction

Assignments 6 and 7 will allow you to use the concepts used in the data structures course to build a real-world system to solve a real world problem. In particular, in these assignments you will build a natural language-based question answering system that will process given corpus of documents (say complete works of Mahatama Gandhi) and answer any question posed by the user in natural language. Your system is expected to answer the questions using the corpus of documents provided to you.

This assignment is to be done in **teams of two**. You are free to choose your own teams. Two teams of Assignment 6, will merge to form a team of 4 for doing the Assignment 7. For assignment 7, you are expected to re-use some of your work done done in this assignment. Please plan accordingly.

In Assignment 6, you will create a *search engine* that will return all matching answers to an input query in the corpus provided, similar to they way search engines return the results. Your job is to make the search engine more robust than Google or Bing. The input query may start from middle of a word and end at the middle of a word, yet you are expected to return all the sentences in the given corpus that match your input query. There will be marks for correctness as well as efficiency if your implementation. In addition, you are expected to create a dictionary that counts the number of occurrences of each word in the given corpus.

In Assignment 7, you will develop an algorithm to rank your search results and select top sentences or paragraphs for a given query and feed those to ChatGPT API (or and open source such API as Hugging Face) along with the query and get the required results which you will display to the user.

Please download the Startercode for Assignment 6 from this link.

1 Dictionary (50 Marks)

Your task in this part is to create a dictionary data structure for storing the word counts of the words in the document. You are free to choose any implementation of your choice. A few ideas which you may consider:

- Red-Black Trees
- AVL Trees
- Hashtables
- Skip Lists (See Lab Sheet 2)
- Any other data structure you would like to use

Your submissions will be graded on the following criteria:

Correctness (30 Marks)

The correctness of your implementation will be checked by making several insert and count queries. The marks for this section will be determined by the fraction of evaluation test cases which your implementation passes.

Competitive Part (20 Marks)

Since the task is fairly straightforward, if your implementation passes all the evaluation tests, it will become eligible for the competitive part. In this part, we will rank your submissions by the runtime on a set of testcases, and marks will be given both by comparing your runtime to a baseline implementation, and according to the ranking of your submission in the leaderboard.

2 Pattern Search (50 Marks)

In the second part, your task is to implement a *search mechanism* to search for a string in the document. Given any pattern p, you must return a list of *all* the offsets in the document at which the pattern is present in the text. The pattern p can be present anywhere in the document, i.e. it can start and end in the middle of words in the document.

Once again, in this problem, you have been given a free hand - you can implement any algorithm of your choice using any data structure you feel would be efficient for the given problem. Some extra reading never hurts! Here are some possibilities you can think about:

- Algorithms using tries
- Algorithms involving hashing, eg. Rabin-Karp Algorithm
- Any other algorithm of your choice

Your submissions will be graded on the following criteria:

Correctness (30 Marks)

The correctness of your implementation will be checked by calling the search function of your Document Tool several times. The marks for this section will be determined by the fraction of evaluation test cases which your implementation passes.

Competitive Part (20 Marks)

If your implementation passed higher than a threshold of evaluation test cases (this will be decided later, may be 100%), it will become eligible for the competitive part. In this part, we will rank your submissions by the runtime on a set of testcases, and marks will be given both by comparing your runtime to a baseline implementation, and according to the ranking of your submission in the leaderboard.

3 Implementation Details

You have been given a header file `dict.h` containing the declaration of the class `Dict`, and a header file `search.h`, containing the declaration of the class `SearchEngine`. You must implement the following functions in `dict.cpp`:

- `Dict()`: Create a Dict instance. In the constructor, you may initialize the data structures you wish to use for storing word counts
- `void insert_sentence(int book_code, int page, int paragraph, int sentence_no, string s)`: Split the given sentence `s` into its constituent words and increment the word count of the words.
- `int get_word_count(string word)`: Given a word, return its count in the document. Note that you should not distinguish between upper case and lower case letters. The word "And" and "and" should both increment the count of the word "and".
- `void dump_dictionary(string filename)`: Dump the dictionary in the given file name. Do not use any upper-case letter while writing the word. All the words must be converted into lower case for counting as well as dumping. *Each line* of the output file must be a word, followed by a comma, and its count. For example, if the dictionary is `{"and": 5, "the": 10, "time":2}`, then running the cat command on the output file after dumping the dictionary should be:

  ```
  $ cat dict.txt
  and, 5
  the, 10
  time, 2
  ```

- $\sim$ `Dict()`: Destroy the Dict instance created for a document.

For the search engine, you must implement the following functions in

- `SearchEngine()`: Create a SearchEngine instance. In the constructor, you may initialize the data structures you wish to use for pattern search
- `void insert_sentence(int book_code, int page, int paragraph, int sentence_no, string s)`: Pre-process the given sentence `s` (if needed by your algorithm) for search queries.
- `Node* search(string pattern, int &n_matches)`: Given a pattern, create a linked list of nodes containing data about each match of the pattern in the text. The nodes should contain:
  - Book Code
  - Page Number
  - Paragraph

- Sentence Number
- Starting Offset, relative to the beginning of the sentence. For example, if the match starts at the first letter of the sentence, the starting offset must be 0. **Note:** Patterns will not span across sentence boundaries. Also, you must not distinguish between upper-case and lower case letters. All the patterns and sentences may be converted into strings with only lower case letters.

For this, you must use the Node class declared in `Node.h`. You must return the head of the linked list which you create. Further, in the variable `n_matches` (passed by reference), store the length of the linked list, i.e. the number of matches.

- $\sim$ `SearchEngine()`: Destroy the SearchEngine instance created for a document.

This time you are allowed to modify the header file and add attributes and helper functions. However, you should not remove the existing functions and attributes of the classes, or your submission will not run. Further, you may not add any more include statements in both the files.

We will soon release a preliminary testercode, as well as the corpora on which we will test your submissions. In the meanwhile, you are welcome to test your code on testcases your create yourself.

4 Submission Instructions

You are required to submit `dict.h`, `dict.cpp`, `search.h` and `search.cpp` on Gradescope. Please ensure you submit both files, and no extra files.

Please note that as usual, the submission deadline is 5 PM, and not midnight.

5 Todo before Assignment 7

Apart from this, you must ensure that you have (atleast) $5 ChatGPT API credits associated with your OpenAI account. OpenAI grants $5 worth of free credits to new users.

It is likely that your credits are expired if you created your OpenAI account more than 3 months ago. In this case, you must do the following:

- Fill up the following form: Quota Increase Form
- Send a message to OpenAI support via the chat interface on the website. In billing related issues, mention that you would like a renewal of the credits as yours expired prematurely. Also read: How can I contact support about billing?

The approval of credits **may take up to a week**, so you are strongly encouraged to apply for renewal of credits as soon as possible. Please note that this is an essential process to the successful completion of Assignment 7. If you face any issues, please reach out to us on Piazza.

6 Points to Note

1. Your submissions will also be tested for plagiarism against existing online implementations, so please do not get carried away by the competitive part and copy code from the internet.
2. Try not to use more than $O(|\text{text}|)$ extra space for any of the algorithms. Some of the corpuses on which your code will be tested may be huge.
3. As mentioned earlier, using any external libraries which have not been allowed to gain an edge in the competitive part will be considered a violation of the honour code.

4. Follow good memory management practices. When an instance of `Dict` and `SearchEngine` is destroyed, it should clean up all memory used by it.

5. Remember, the debug test cases are not representative of the evaluation test cases. You must test your submissions rigorously and ensure it runs on all cases, not just the debug test cases.