TASK 1

Overview

- Domain : Searching, Strings
- Idea : String matching and searching

Introduction

Most of the data you come across is in the form of strings. Designing algorithms for searching in these is very important since we need to go through a lot of data quickly and gain meaningful information or for collecting data. Also it's important for manipulating file information. These algorithms are extensively used by twitter (for finding trending tweets) , google search engine, and many more.

What you'll learn

- Working with files
- Regular expressions and string searching/matching
- Designing new algorithms

Problem Statement

You are given a list of queries, and a book. Your task is to find:

- Line Number and Page number of ALL the occurrences of the given queries.

Instructions

PLEASE READ THESE INSTRUCTIONS VERY CAREFULLY

Donwloading Materials

- Go to this link to download the required materials for this task.

  > Task files

Contents

The downloaded .zip folder contains these files.

- `queries.txt`
- `page_*.txt` , these are 25 in number, from `page_1` to `page_25` .

The queries are located in a file called `queries.txt` , the file contains words written on a new line.

Then there are 25 pages of a very famous book, `Adventures of Huckleberry Finn` a classic from the olden era.

Setup for this task

To set up for this task extract the 2 files provided. Then follow these instructions

- Once you extract the folder, **WRITE YOUR CODE IN THE SAME FOLDER**, otherwise you will face a lot of issues when opening the files.

Working On the task

In order to work on this task you have to write a program that searches the pages for the given words `(in queries.txt)` and shows all occurrences of the given word.

Example

If `queries.txt` contains :

```
Queries.txt
~~~~~~~~~~~
Hello
World
Objects
Fun
```

And the `page_1.txt` contains:

```
Page_1.txt
~~~~~~~~~
Hello! Hello world, I am new to the world of
programming languages, but I am guessing it
is kind of fun!
```

Your output file should be of the format:

```
Output.txt
~~~~~~~~~~
Word: Hello
Occurrences:
Page 1, line 1
Page 1, line 1

Word: World
Occurrences:
Page 1, line 2

Word: Objects
Occurrences:
None

Word: Fun
Occurrences:
Page 1, line 3
```

> **NOTE: the word that you are finding must be an independent word**
> i.e. if you are searching for **round**, and the line contains
>
> - **around the globe**
>
> Then the word **round** has **NO MATCH**, even though ***round*** is a subword of ***around***.
>
> **NOTE: The queries are case sensitive.**

HINTS

- In order to efficiently read a page, try creating a function that takes a page number as an argument, and returns a list of the lines in that particular file.
- When returning a list of strings in C++, don't use:

  > `char mystrings[][]`

- Instead try using

  > `std::vector<std::string> mystrings` by including the headers
  > `#include<vector> and #include<string>`

- To access i^{th} string, it will be as simple as:

  > `mystrings[i]`

Precautions

Please follow the given precautions as the input is quite large

- Try to use data structures like `Sets`, `Vectors`, and `trees`, `Strings(not the char array from c)`.
- **NEVER!** read all the given files at once, read only **max 5 files** at a time.
- Try to use the `C++ STL library`
- Using primitive C style arrays and no proper algorithms will make the program run very slow, and will not generate correct output.
- Do not use Naive programming, it simply highlights your unwillingness to think for better and more optimized algorithms.
- Try to use the algorithms that we have mentioned below in the section Useful Materials.

HackerMode

Searching for strings using the KMP algorithm works fine, however, when it comes to very large files, and the search query also being large, a better and faster algorithm known as

> *Boyer - Moore Algorithm*

is used.
As a HackerMode task, your job is to implement the above mentioned algorithm instead of the KMP algorithm.

Submission

As usual, you have to submit your code and the output.txt file by creating a repo and then pushing code to it regularly. When you feel you have completed the task, submit the link of the repo on this website:

> Spider Inductions Website

Useful Materials

Linked Below are some algorithms, and data structures that will be necessary for you to complete this task.

- Bucket Sort
- Popular String Matching Algorithms
- KMP algorithm implementation in C++
- Read on what hashing is, if you can think of a way to incorporate hashing in your algorithm, you will see that your code will perform much better.

General Instructions

Please **READ ALL THE INSTRUCTIONS ON THIS PAGE CAREFULLY** before you begin your task. It is apparent when you ignore clear instructions and will reflect poorly in your evaluation.

- Write readable code. Write code with indentation and good readability. Pay attention to your variable names and object names.

- Write good commit messages.

- A good commit message should be short and to the point. It should give a <10 word overview of what exactly the commit does. Messages like Updated XYZ.html and Change #5 , Change #6 etc. are examples of poor commit messages. A good commit message would be Update ABC.js to add sin() function .

  > This is a great short guide on writing commit messages. Please go through it.

- Commit frequently, but not THAT frequently. A commit must have one definite basic logical change. A commit for changing a single character is not encouraged. Please do not flood your repositories with commits - a commit is not equal to a Ctrl+S. At the same time, don't make a single commit change 20 files and 2000 lines of code.

- Test your own code thoroughly. Your code should be working for you, at the least. Do not submit code that you have not completely tested.