CS 378h, Spring 2026, Laboratory 2
String Search

Assigned: Thursday, February 26, 2026
Due: Thursday, March 12, 2025, by 9:30 am

1 Introduction

In this lab, you will learn about the search performance of a x86-compatible microprocessor. This laboratory asks you to write C-language code to search for strings within another string. This laboratory is two part (see below). The challenge in this laboratory is to understand the impact of search algorithms on memory performance.

2 Logistics

You are expected to work on this lab alone. However, you may communicate with others concerning your understanding of C code and the various tools, e.g., the compiler, assembler, linker, loader, and other systems issues. The results that you submit in response to laboratory must be created and provided by you alone.

Any clarifications and revisions to the laboratory will be posted on the top-level course webpage. By Friday, March 6, 2026, we will post several files to serve as a test corpus and also a collection of search patterns. For each (search) pattern, your program needs to return the number of times the search pattern appears in the test file, and you must capture the time your combined searches require. Note that your program should read in the entire file prior to performing all of the search requests – so your results are not affected by file-system performance. It's a good idea to read every location (character) in the files that you read into memory so as to make sure that your string search tests are not waiting on the file system.

Your program should be called as follows:

```
pat_search <src-filename> <pattern-file> <type>
```

The `<src-filename>` argument should be the name of a Linux file; the `pattern-file>` should be a string (enclosed in double quotes), and the type (the numerals **1** or **2**) refers to the algorithm to use. Any `<pattern-file>` will be a file with a collection of strings (one string per line) for which your programs

should be able to search. The ¡type¿ argument should be **1** or **2** to indicate whether the simple search algorithm (**1**) is to be used of the Boyer-Moore search algorithm **2** is to be used.

3 Algorithm

You are to implement two algorithms:

- a simple algorithm that compares characters one-by-one, and
- the Boyer-Moore string search algorithm.

Your two string-search implementation should return the same answers. We expect the Boyer-Moore algorithm to be faster, but how much faster? This is something for you to determine.

We will provide a collection of strings on which you will need to run your two codes. For each test file and for each search algorithm, your program should return the amount of time required to find the total number of string matches for a given search pattern file. Note, before you start your timing, you should read the entire test files into main memory, so as to avoid Linux loading your test file incrementally,. Also note that one test file will be large – say several hundred megabytes.

Thus, for the "file" below:

```
"abcdabceabc"
```

with the two search strings (taken from a file)

```
"abc"
"dab"
```

with either (search) type (**1** or **2**), your program should return **4**.

You may assume that the files and search strings provided will only contain printable ASCII characters (along with line feed characters).

4 Evaluation

You are expected to write a short report that explains timing results of running your programs using both algorithms.

Your entire write-up should be quite succinct. In addition, you must also submit your C-language code. And, you must include your test timings and results. The class TA will post how to submit your program. Your write-up should be included as a C-language-style comment at the front of your solution – and it must in ASCII and no more than 100 lines or less in length. Longer write-ups will have points deducted!

On March 5, the TA will post test input.

The maximum score for this laboratory is 100 points. The value of the individual components is as follows.

- Timing results (meaning byte copying rates) for solutions to Parts 1 and 2 (60 points). The timing results for each test file should include just four numbers: the number of matches for each algorithm (which should be the same); the total time to perform all of the searches with the simple algorithm; and the total time to perform all of the searches using the Boyer-Moore string search algorithm.
- An explanation of your timing results (10 points).
- An explanation of the code that you wrote that documents your approach and rationale for text search solutions (30 points). This explanation should include a discussion of how one can assure the correctness of your implementation.

5 Hints

Work incrementally. Don't try to do everything at once. Part 1 is straightforward – get it working first. Part 2 is quite a bit more work because of the tabular information that needs to be created for each search string. For each Boyer-Moore search, your timing should include the time to create the search movement tables.

6 Hand In Instructions

Please follow the instructions below for turning in your work.

- Make sure you have included your identifying information in your submission. In this case, you will update the file you submit with your name, UTID. Submit only a single C-language file that contains both your report (in ASCII) and your code. Your timings for both searches on each example file should be included with your comments.
- The solution file you submit must be named:

```
<YourUTID>-search.c.
```

The **CS378h** TA will post instructions as to how to Submit your laboratory report by using CANVAS.