A report on the performance of various file search methods and algorithms

SEARCHING FILES USING PYTHON

There are many methods of implementing a file search in python. After conducting extensive research on many of these methods, I decided to look further into ten of these methods. They are of varying functionalities, some are more suitable for a particular input type than others, and some of them can only be used on linux computers. The ten methods that I eventually settled on are:

- Built-in search method
- Memory Map (Mmap) search method
- Grep search method
- Silver searcher (ag) method
- Awk search method
- Regular Expression (regex/re) method
- Aho-Corasick search method/algorithm
- Knuth-Morris-Pratt (KMP) search method/algorithm
- Bloom Filter search method
- Finite Automata search method

I will now examine each of them in detail

Built-in Search Method

This is the easiest file search method to set-up or code, as it just involves opening the text file in a readable format, or using the 'r' mode, and looping through the file line-by-line to find which line corresponds with the searched string or query. Nevertheless, don't be fooled by this ease of set-up, it is the second fastest file search method I researched, clocking in at 0.0037s for a file length of 100,000 lines, and 0.0043s for a file length of 200,000 lines.

Memory Map (Mmap) search method

This is a very unique file search method because it utilizes low-level programming to assign a segment of virtual memory a byte-for-byte correlation with some portion of a file or file-like resource. Once present, this correlation between the file and memory space permits applications to treat the mapped portion as if it were primary memory. The main benefit of memory mapping a file is to increase input/output performance. This method was the fastest file search method that I researched, clocking in at 0.0013s for a file length of 100,000 lines and 0.0019s for a file length of 200,000 lines.

Grep search method

This is an in-built linux command that prints lines that contain a match for one or more patterns. Given one or more patterns, grep searches input files for matches to the patterns. When it finds a match in a line, it copies the line to standard output (by default), or produces whatever other sort of output you have requested with options. Though grep expects to do the matching on text, it has no limits on input line length other than available memory, and it can match arbitrary characters within a line. This file search method clocked in at a speed of 0.0096s for a file length of 100,000 lines, and 0.008s for a file length of 200,000 lines.

Awk search method

Awk is a scripting language used for manipulating data and generating reports. The awk command requires no compiling and allows the user to use variables, numeric functions, string functions, and

logical operators. Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform the associated actions. This file search method clocked in at a speed of 0.011s for a file length of 100,000 lines, and 0.0218s for a file length of 200,000 lines.

Silver searcher (ag) method

This is also an inbuilt linux command (although it can be installed on other operating systems) that is used to search for patterns in text. It is similar to the awk search method, but about 30 times faster. This method utilizes memory mapping to read files, and searching is done with the use of the Boyer-Moore string search algorithm. However, it did not do so well in searching through the given text file, and was one of the slower file search methods, clocking in at 0.198s for a file length of 100,000 lines, and 0.0262s for a file length of 200,000 lines.

Regular Expression (regex/re) method

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the ‘re’ module. The regular expression language is relatively small and restricted, so not all possible string processing tasks can be done using regular expressions. Some of the processing task that they are normally used to perform are matching characters and searching for patterns in strings or text files. This is done by using this language to specify the rules for the set of possible strings that you want to match, and then asking questions such as “Does this string match the pattern?”, or “Is there a match for the pattern anywhere in this string? This file search method clocked in at a speed of 0.0343s for a file length of 100,000 lines, and 0.028s for a file length of 200,000 lines.

Aho-Corasick search method/algorithm

This is a type of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. Informally, the algorithm constructs a finite-state machine that resembles a trie with additional links between the various internal nodes. These extra internal links allow fast transitions between failed string matches to other branches of the trie that share a common suffix. This allows the automaton to transition between string matches without the need for backtracking. The Aho–Corasick algorithm assumes that the set of search strings is fixed. It does not directly apply to applications in which new search strings are added during application of the algorithm. The Aho–Corasick string matching algorithm formed the basis of the original Unix command fgrep. This file search method clocked in at a speed of 0.0105s for a file length of 100,000 lines, and 0.0108s for a file length of 200,000 lines.

KMP search method

The Knuth-Morris-Pratt (KMP) algorithm is a string-searching algorithm that searches for occurences of word within a main text string or file by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. KMP algorithm has a better worst case performance than the brute force or strainghtforward search because it makes use of previous match information, while the others do not. This file search method clocked in at a speed of 0.2734s for a file length of 100,000 lines, and 0.2042s for a file length of 200,000 lines.

Bloom Filter search method

A bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a menber of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either “possibly in set”, or “definitely not in set”. Elements can be added to the set, and the more items added, the larger the probability of false positives. Bloom filters have a substantial space advantage over other data structures and file search methods. This file search method clocked in at a speed of 0.9417s for a file length of 100,000 lines, and 1.4299s for a file length of 200,000 lines.

Finite Automata Search Method

The Finite Automaton-based pattern searching algorithm utilizes the concept of a deterministic finite automaton (DFA) to efficiently search for patterns within text. It involves two main steps: constructing the Transition Function (TF) table and traversing the DFA over the text. The idea of this approach is to build finite automata to scan text T for finding all occurrences of pattern P. This approach examines each character of text exactly once to find the pattern. Thus it takes linear time for matching but preprocessing time may be large. These string matching automaton are very efficient because they examine each text character exactly once, taking constant time per text character. The matching time used is O(n) where n is the length of Text string. This file search method clocked in at a speed of 0.0064s for a file length of 100,000 lines, and 0.0066s for a file length of 200,000 lines.

Table showing different search methods and their speeds as a function of file size

File Search Method	100k file Speed (seconds)	200k file speed (seconds)	Sample text file Speed (seconds)
Memory Map Method	0.0013	0.0019	0.0012
Built-in Method	0.0037	0.0043	0.0044
Finite Automata Method	0.0064	0.0066	0.0064
Grep search method	0.0096	0.008	0.01
Aho-Corasick algorithm	0.0105	0.0108	0.0093
Awk search method	0.011	0.0218	0.0247
Regular Expression method	0.0343	0.028	0.0306
Silver searcher method	0.198	0.0262	0.0368
KMP algorithm	0.2734	0.2042	0.2057
Bloom Filter method	0.9417	1.4299	1.8475

Charts/Graph of different search methods plotted against their speeds

100k text file

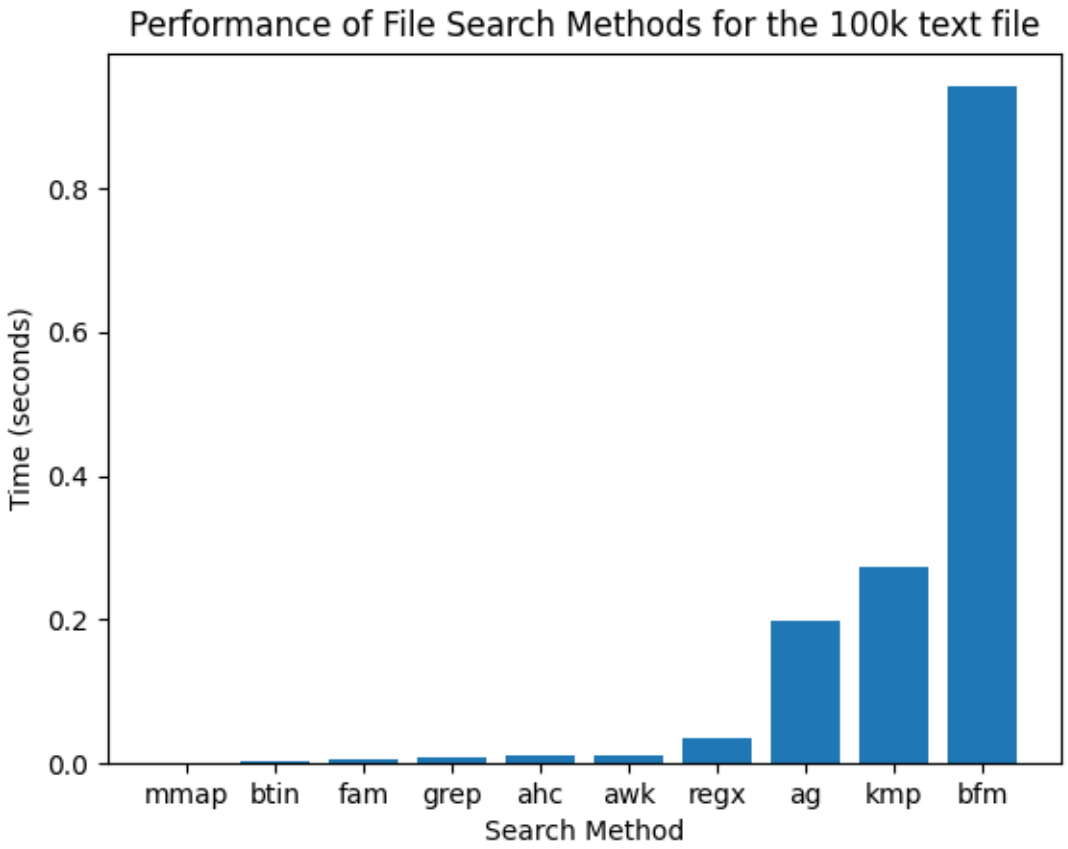

Chart Keys
mmap : **Memory Map Method**
btin : **Built−in Method**
fam: **Finite Automata Method**
ahc: **Aho−Corasick Algorithm**
grep: **Grep Search Method**
awk: **Awk Search Method**
regx: **Regular Expression Method**
ag: **Silver Searcher Method**
kmp: **Knuth−Morris−Pratt Method**
bfm: **Bloom Filter Method**

200k text file

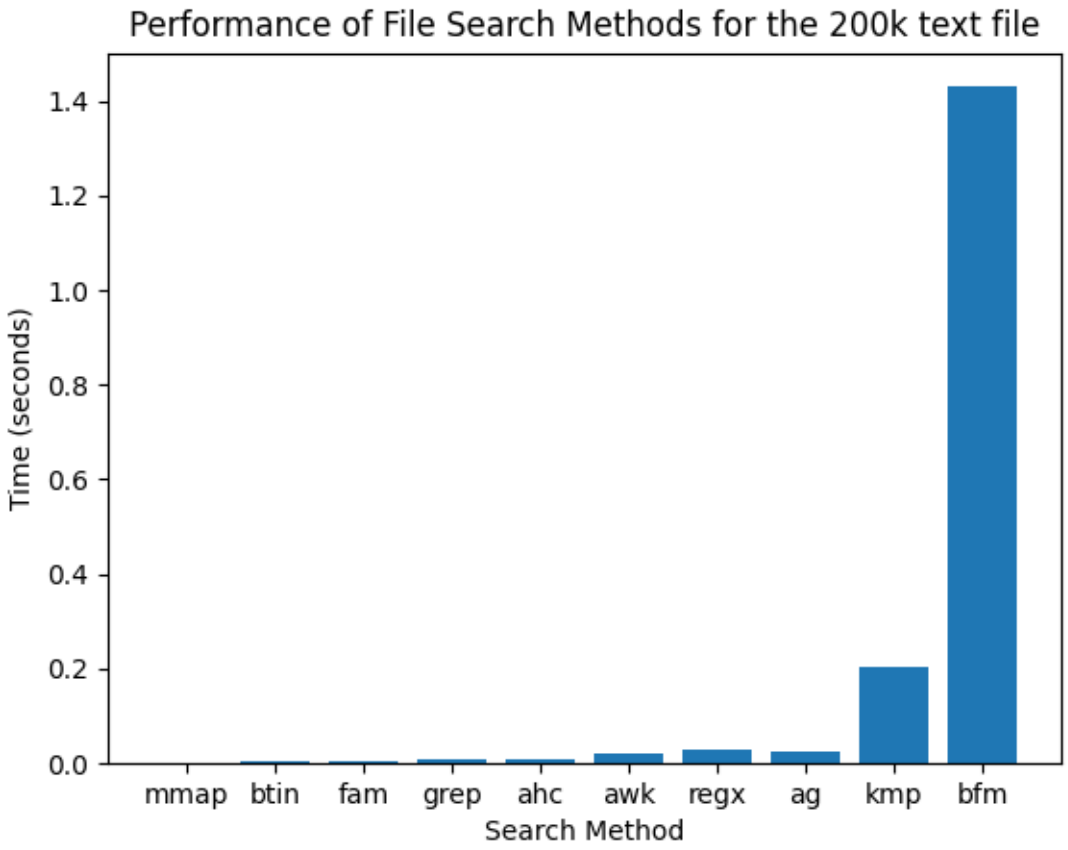

Chart Keys
mmap : **Memory Map Method**
btin : **Built−in Method**
fam: **Finite Automata Method**
ahc: **Aho−Corasick Algorithm**
grep: **Grep Search Method**
awk: **Awk Search Method**
regx: **Regular Expression Method**
ag: **Silver Searcher Method**
kmp: **Knuth−Morris−Pratt Method**
bfm: **Bloom Filter Method**

The given text file

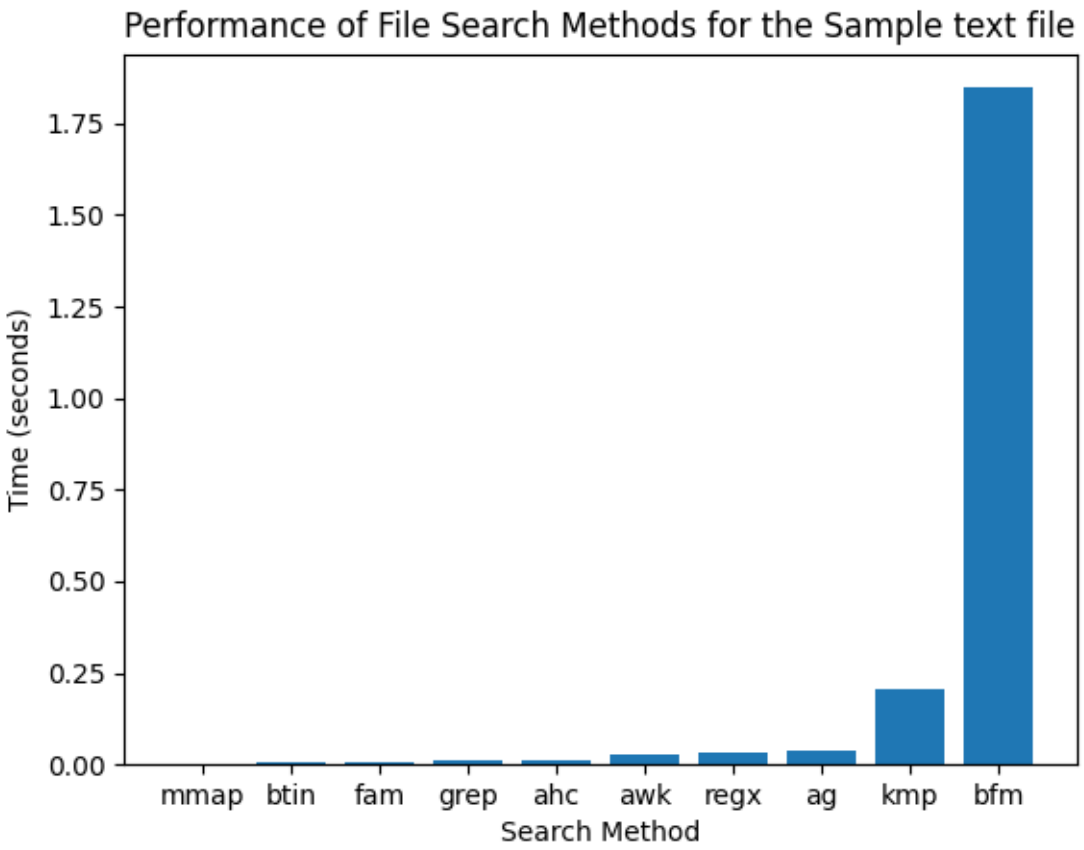

Chart Keys
mmap : **Memory Map Method**
btin : **Built−in Method**
fam: **Finite Automata Method**
ahc: **Aho−Corasick Algorithm**
grep: **Grep Search Method**
awk: **Awk Search Method**
regx: **Regular Expression Method**
ag: **Silver Searcher Method**
kmp: **Knuth−Morris−Pratt Method**
bfm: ***Bloom Filter Method***

Instructions for running the server as a linux daemon

1. Place the Python script (e.g., server.py) in your preferred directory
2. Create a systemd service file in /etc/systemd/system/myserver.service with the following content:

```
[Unit]
Description=My Python Server
After=network.target

[Service]
ExecStart=/usr/bin/python3/path/to/your/directory/server.py
WorkingDirectory=/path/to/your/directory
Restart=always
User=yourusername
Group=yourgroup
StandardOutput=syslog
StandardError=syslog

[Install]
WantedBy=multi-user.target
```

Explanation:
ExecStart: Specifies the command to run the server. Replace /path/to/your/directory/server.py with the path to your Python script.

WorkingDirectory: Directory where the server will be run.

Restart=always: Restarts the server if it crashes.

User and Group: Specifies which user and group should run the service. It's good practice to run the service as a non-root user.

Environment: Sets environment variables (optional). PYTHONUNBUFFERED=1 ensures Python logs are printed immediately without buffering.

StandardOutput and StandardError: Redirects logs to the system's syslog (you can change this to log files if needed).

3. Reload the systemd configuration with: “sudo systemctl daemon-reload”

4. Start the service: “sudo systemctl start myserver.service”

5. Enable the service to start on boot: “sudo systemctl enable myserver.service”

6. You can check if the service is running by using: “sudo systemctl status myserver.service”

7. If you make changes to your Python code and want to restart the service, use:
“sudo systemctl restart myserver.service”