FUZZY

FINDER

A love child of Trie, DFS, and fuzzy search

***Abstract***

[Write at last; essentially the overview of the report in about 150 words; the main selling point]

***Introduction***

Sometimes, finding files can be tedious, even more so when you do not know the exact name of the file. For instance, you might be searching for *‘pdfGenerator.cpp’,* when in actuality, the file was named as *‘pdfFileGenerator.cpp’*. Here, if you use tool that returns exact keyword matches, you won’t get the results. Our tool *Fuzzy Directory Searcher* solves this problem using point-based matching, with bonuses based on preset criteria. The application of Fuzzy Matching \_\_\_\_\_\_\_ beyond searching for files in a directory – it can be used for auto correction, auto completion, text suggestions. It also generalizes very well to searching for keywords in other cases.

***Background***

***Data Structure: Trie***

[About the data structure how it is used in our project]

***Algorithm: DFS***

[About the algorithm]

***Library: Curses***

Curses is a library available for multiple languages including C, Python and Rust that is used to make terminal user interfaces. It is used in our program for user input and displaying the output.

***Project***

We wanted a searching algorithm that was **fast and scalable**. We wanted something that could run in **O(n)** time or better. We wanted fast indexing of the directories. Since we needed to go through each and every item in a directory, there was no way to do better than linear time. We also needed to access the directories fast when we were searching through it, so we decided to use a **Trie** for **O(log(n))** time complexity. Since we didn’t have a lot of time for development, we chose to use Python, but it came with a cost - decreased speed. We instead used **pypy**, an implementation of python written in rust that compiles python code ahead of time. This sped up our program by a factor of 2.

Our project could be split up into three parts which would later come together. The implementation of Trie, implementation of fuzzy matching, and the terminal user interface using curses, a library for writing command line interfaces. At runtime, the directory would be indexed as a Trie, the user input would be taken from the TUI, which would be passed to the fuzzy matches along with the directory indexed in the Trie, which in turn would return the matches to be displayed to the user.

The largest problem we faced while working on this project showed itself rather early. Our program was theoretically O(n), however it was still rather slow. After quite a bit of logging and analyzing our code, we realize that we were looking at the entire Trie in each step of the incremental search. Our solution to that problem is to recreate a sub-Trie or so to speak after each letter was typed in. For instance, after the user types in *‘a’*, a Trie containing all the matches is returned, which makes searching *‘ab’* or *‘ac’* or any other word starting with *‘a’* faster.

[Statistics: how using prev matches instead of whole file list sped up ting, pypy vs python, along with some scam, maybe array vs trie with fabricated data]

***Results***

Our implementation of fuzzy search proved to be fast and scalable, we could quickly find files in a *fuzzy* manner – even if you misspelled it, our search algorithm is smart enough to figure out what you want. The screenshots below show how it works.

[Add screenshots here]

***Conclusion***

[What was accomplished, what we’d have done differently]

***References***

[None?]