Published online on the journal website: https://publikasi.mercubuana.ac.id/index.php/fasilkom/

Collabits Journal

Vol 1, No 1 (2024) ISSN :

FILE SEARCH ON AN FTP SERVER USING THE FUZZYWUZZY METHOD AND THE DFS ALGORITHM

Bambang Jokonowo[1], Agil Dwiki Wijaya[2], Naufal Saidhus Syuhur[3]

[1,2,3]Fakultas Ilmu Komputer, Universitas Mercubuana

Bambang.jokonowo@mercubuana.ac.id[1], 41522110068@student.mercubuana.ac.id[3]

Abstract

The Depth-First Search (DFS) algorithm can be used to search for files on an FTP server. The fuzzy-wuzzy method can be applied to perform fuzzy string matching for the file names. To implement this, one can establish a connection to the FTP server using a programming language that supports FTP operations, retrieve the file listings using DFS, apply fuzzy matching to the file names, and retrieve the matching files. The fuzzy-wuzzy method is a string-matching library that uses the Levenshtein Distance to calculate the differences between sequences. While there are no specific examples of using DFS with fuzzy-wuzzy on an FTP server, the general approach can be implemented using FTP in Python for FTP operations and the fuzzy-wuzzy library for fuzzy string matching.

Keywords : *Depth-First Search (DFS), fuzzy-wuzzy, FTP*

1. INTRODUCTION

The Depth-First Search (DFS) algorithm can be used to search for files on an FTP server. This is a helpful way to navigate the directory structure on a server that has different files and folders. The File Transfer Protocol (FTP) is a protocol that facilitates file transfers across a network between computers, while the Deep Folding Search (DFS) algorithm is a search algorithm designed to thoroughly examine a data structure.

By using DFS in file searches on an FTP server, users can locate desired files more efficiently without needing to have a detailed understanding of the folder structure. Users can more effectively maintain and access existing files on an FTP server by utilizing this technique.

Increased efficiency in file management and access to a variety of data stored on the FTP server are made possible by this more focused and effective search feature. The effective and efficient management of files and information kept on FTP servers is greatly aided by the utilization of the DFS algorithm in FTP file searches.

2. METHOD

Research approaches that can be carried out regarding file searches using the DFS algorithm on an FTP server employing the fuzzy-wuzzy method are as other data structure, like a tree or graph, can have all of its nodes explored or searched using the Depth-First Search (DFS) algorithm. Before moving on to a new branch, DFS searches as far as it can on a current branch. DFS, in theory, starts its exploration of a branch at its deepest node and works its way out.

a) Utilizing the Fuzzywuzzy Technique
   To do fuzzy string matching against file names in the list of retrieved files, use the fuzzy-wuzzy technique. A Python library called FuzzyWuzzy can be used to compare a string to a set of strings in order to determine how similar they are.
b) Assessment of the Outcomes
   Evaluate search results by comparing found files with predefined search criteria.
c) Implementing Fuzzy-Wuzzy and DFS
   On the used FTP server, implement the DFS algorithm and fuzzywuzzy method, then assess the server's performance.
d) Evaluation via Comparison
   To ascertain the benefits and drawbacks of each approach, compare the DFS algorithm with the fuzzy-wuzzy method and the BFS algorithm with alternative search techniques..

other data structure, like a tree or graph, can have all of its nodes explored or searched using the Depth-First Search (DFS) algorithm. Before moving on to a new branch, DFS searches as far as it can on a current branch. DFS, in theory, starts its exploration of a branch at its deepest node and works its way out.

e) Utilizing the Fuzzywuzzy Technique
To do fuzzy string matching against file names in the list of retrieved files, use the fuzzy-wuzzy technique. A Python library called FuzzyWuzzy can be used to compare a string to a set of strings in order to determine how similar they are.

f) Assessment of the Outcomes
Evaluate search results by comparing found files with predefined search criteria.

g) Implementing Fuzzy-Wuzzy and DFS
On the used FTP server, implement the DFS algorithm and fuzzywuzzy method, then assess the server's performance.

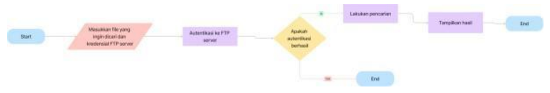
h) Evaluation via Comparison
To ascertain the benefits and drawbacks of each approach, compare the DFS algorithm with the fuzzy-wuzzy method and the BFS algorithm with alternative search techniques..

The Python programming language, the FTP module for FTP operations, and the fuzzy-wuzzy library for fuzzy string matching can all be used to conduct the aforementioned research approach.The Python programming language, the FTP module for FTP operations, and the fuzzy-wuzzy library for fuzzy string matching can all be used to conduct the aforementioned research approach.

3. RESULT AND DISCUSSION

3.1 Primary Flowchart

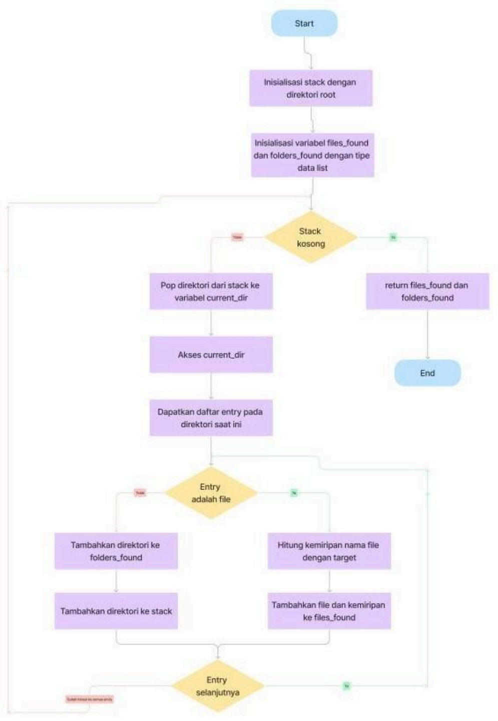
The primary flowchart for the software that will be developed is as follows.


Start
Masukkan file yang ingin dicari dan kredensial FTP server
Autentikasi ke FTP server
Apakah autentikasi berhasil
Lakukan pencarian
Tampilkan hasil
End
End

An FTP server's information search procedure is shown in the flowchart above. Entering the credentials needed to access the FTP server is the first step in the process. If the authentication process is successful, the next step is to see if the requested information is on the server. In that case, search results will be shown; if not, the procedure is completed.

3.2 Flowchart for DFS Search

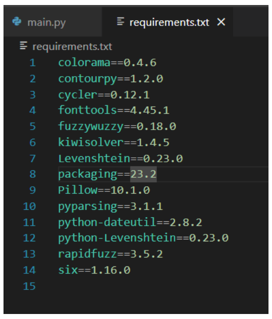
Here is the flowchart DFS Search utilizes metode fuzzywuzzy.


Start
Inisialisasi stack dengan direktori root
Inisialisasi variabel files_found dan folders_found dengan tipe data list
Stack kosong
Pop direktori dari stack ke variabel current_dir
return files_found dan folders_found
End
Akses current_dir
Dapatkan daftar entry pada direktori saat ini
Entry adalah file
Tambahkan direktori ke folders_found
Hitung kemiripan nama file dengan target
Tambahkan direktori ke stack
Tambahkan file dan kemiripan ke files_found
Entry selanjutnya

The procedure for looking for files and folders within the directory hierarchy is shown in the flowchart above. First, the stack is initialized with a root directory and variables for files and directories that are found. After that, the stack is examined to determine if it is empty; if not, the current directory is removed from the stack and its contents are examined to see if they satisfy the requirements for files or folders. The directory is returned to the stack for additional processing after matching entries have been added to the discovered list. It returns the files and folders found if the stack is empty.The procedure for looking for files and folders within the directory hierarchy is shown in the flowchart above. First, the stack is initialized with a root directory and variables for files and directories that are found. After that, the stack is examined to determine if it is empty; if not, the current directory is removed from the stack and its contents are examined to see if they satisfy the requirements for files or folders. The directory is returned to the stack for additional processing after matching entries have been added to the discovered list. It returns the files and folders found if the stack is empty.

3.3 Supporting Packages

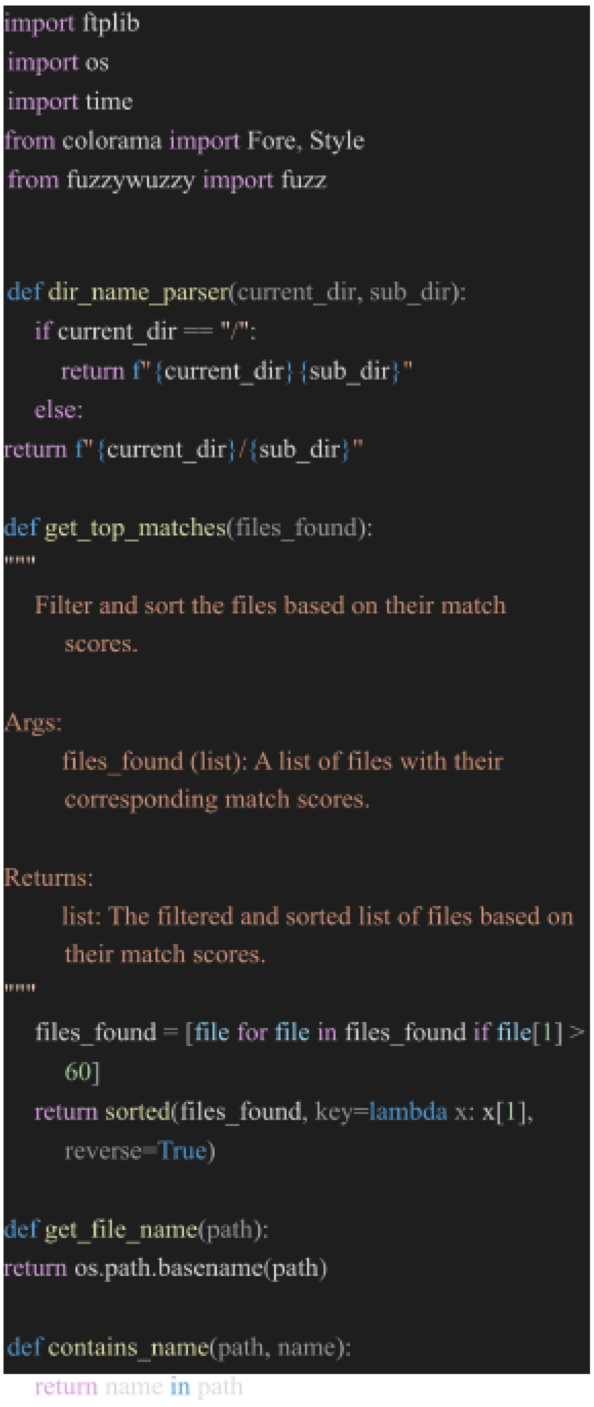
Create a file called requirements.txt which will later act as a file containing a list of packages or libraries needed to work on a project, all of which can be installed with this file.



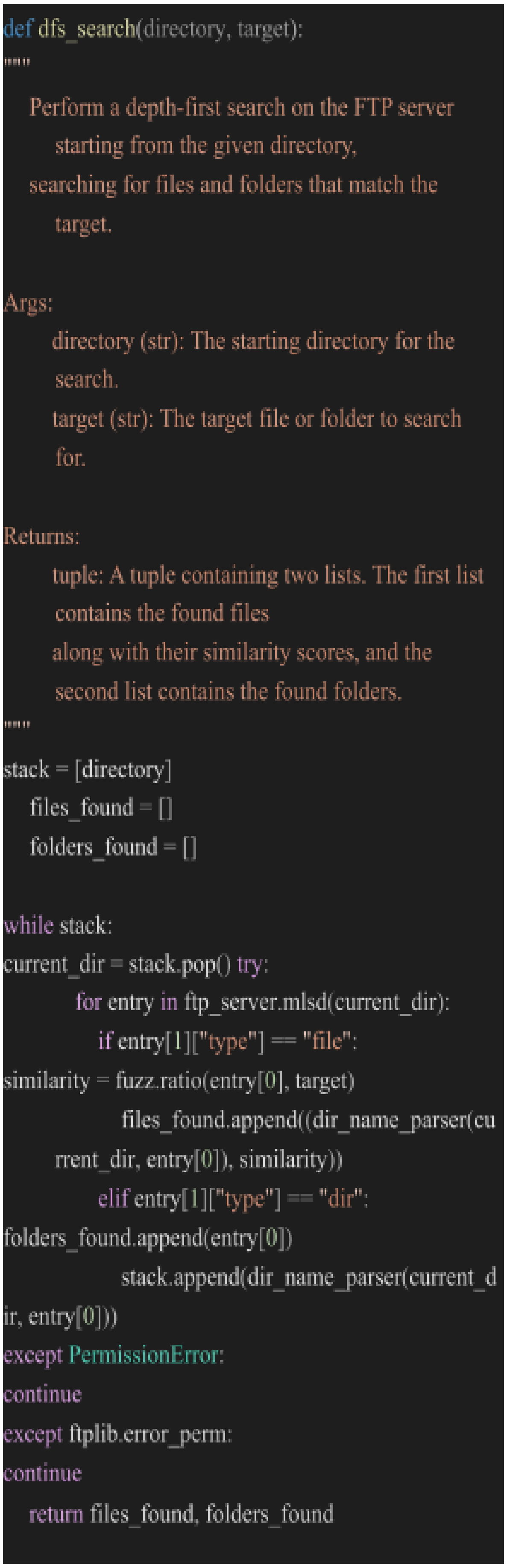
```
main.py    requirements.txt ×
requirements.txt
colorama==0.4.6
contourpy==1.2.0
cycler==0.12.1
fonttools==4.45.1
fuzzywuzzy==0.18.0
kiwisolver==1.4.5
Levenshtein==0.23.0
packaging==23.2
Pillow==10.1.0
pyparsing==3.1.1
python-dateutil==2.8.2
python-Levenshtein==0.23.0
rapidfuzz==3.5.2
six==1.16.0
```

The packages in file

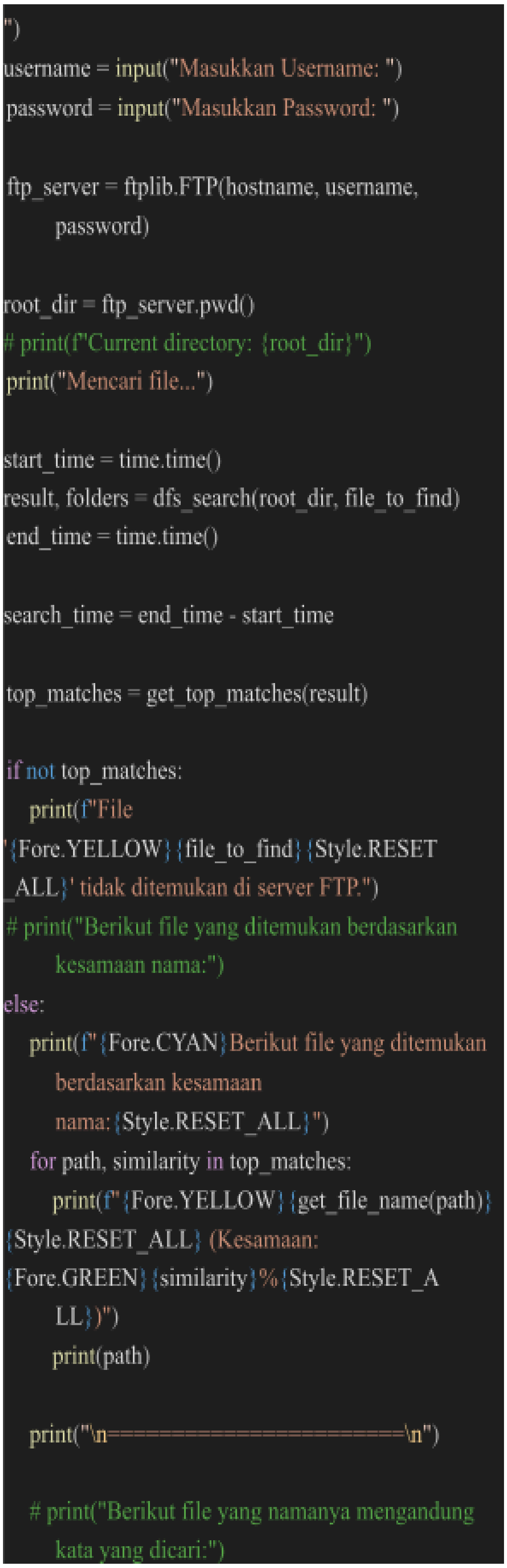
1) Colorama==0.4.6 The Library to enable text display modification (including color) in the Python terminal are explained in multiple ways.
2) Contourpy==1.2.0: Python library for generating contour graphs or plot visualizations from data.
3) Cycler = 0.12.1 A Python library that makes it easier to create cycles; this library is often used to set the cycle attribute in matplotlib.
4) Fonttools==4.45.1: Font manipulation library, supporting both OpenType (OTF) and TrueType (TTF) formats.
5) The fuzzy fuzzy = 0.18.0 a library offering a fuzzy string matching technique for identifying similarities (matching) and matching strings.
6) Kiwisolver==1.4.5 Library, frequently used to handle matplotlib layout issues, solves value search problems in mathematical optimization problems.
7) Levenshtein = 0.223-0 a library that offers a way to compute the distance between two strings using the Levenshtein distance algorithm.
8) Packaging==23.2 Libraries that offer resources for managing dependencies, distribution, and working with Python packages.
9) Pillow==10.1.0: A Python image processing package that is typically used to access, edit, and store different image formats.
10) Use the Pyparsing==3.1.1 Library to simplify Python text or string parsing.
11) Python-dateutil == 2.8.1 a library that offers more features from Python's built-in datetime module, making it easier to interact with dates and times in Python.
12) Kiwisolver==1.4.5 Library, frequently used to handle matplotlib layout issues, solves value search problems in mathematical optimization problems.
13) Levenshtein = 0.223-0: a library that offers a way to compute the distance between two strings using the Levenshtein distance algorithm.
14) Packaging==23.2 Libraries that offer resources for managing dependencies, distribution, and working with Python packages.
15) Pillow==10.1.0: A Python image processing package that is typically used to access, edit, and store different image formats.
16) Use the Pyparsing==3.1.1 Library to simplify Python text or string parsing.
17) Python-dateutil == 2.8.1 A library that offers more features from Python's built-in datetime module, making it easier to interact with dates and times in Python.



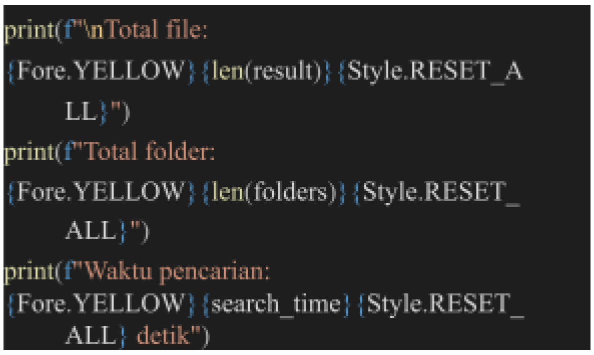
```
import ftplib
import os
import time
from colorama import Fore, Style
from fuzzywuzzy import fuzz


def dir_name_parser(current_dir, sub_dir):
  if current_dir == "/":
    return f"{current_dir}{sub_dir}"
  else:
return f"{current_dir}/{sub_dir}"

def get_top_matches(files_found):
"""
  Filter and sort the files based on their match
    scores.

Args:
    files_found (list): A list of files with their
    corresponding match scores.

Returns:
    list: The filtered and sorted list of files based on
    their match scores.
"""
  files_found = [file for file in files_found if file[1] >
    60]
  return sorted(files_found, key=lambda x: x[1],
    reverse=True)

def get_file_name(path):
return os.path.basename(path)

def contains_name(path, name):
  return name in path
```



```
def dfs_search(directory, target):
"""
    Perform a depth-first search on the FTP server
        starting from the given directory,
    searching for files and folders that match the
        target.

Args:
        directory (str): The starting directory for the
        search.
        target (str): The target file or folder to search
        for.

Returns:
        tuple: A tuple containing two lists. The first list
        contains the found files
        along with their similarity scores, and the
        second list contains the found folders.
"""
stack = [directory]
    files_found = []
    folders_found = []

while stack:
current_dir = stack.pop() try:
            for entry in ftp_server.mlsd(current_dir):
                if entry[1]["type"] == "file":
similarity = fuzz.ratio(entry[0], target)
                    files_found.append((dir_name_parser(cu
        rrent_dir, entry[0]), similarity))
                elif entry[1]["type"] == "dir":
folders_found.append(entry[0])
                    stack.append(dir_name_parser(current_d
ir, entry[0]))
except PermissionError:
continue
except ftplib.error_perm:
continue
    return files_found, folders_found
```

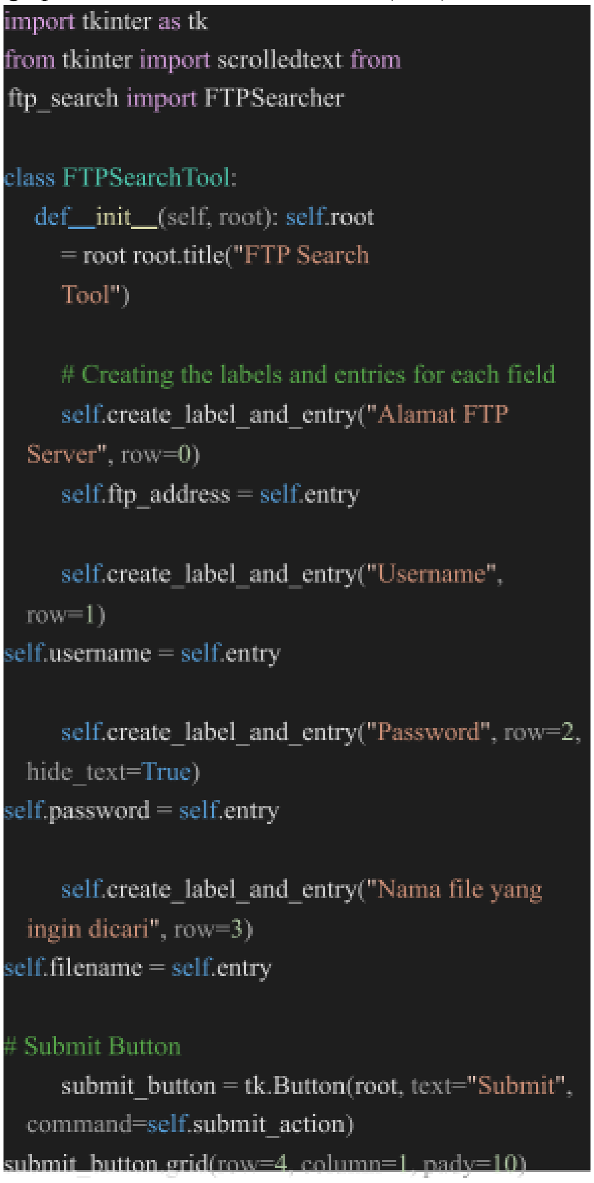


```
")
username = input("Masukkan Username: ")
password = input("Masukkan Password: ")

ftp_server = ftplib.FTP(hostname, username,
        password)

root_dir = ftp_server.pwd()
# print(f"Current directory: {root_dir}")
print("Mencari file...")

start_time = time.time()
result, folders = dfs_search(root_dir, file_to_find)
end_time = time.time()

search_time = end_time - start_time

top_matches = get_top_matches(result)

if not top_matches:
    print(f"File
'{Fore.YELLOW}{file_to_find}{Style.RESET
_ALL}' tidak ditemukan di server FTP.")
# print("Berikut file yang ditemukan berdasarkan
        kesamaan nama:")
else:
    print(f"{Fore.CYAN}Berikut file yang ditemukan
        berdasarkan kesamaan
        nama:{Style.RESET_ALL}")
    for path, similarity in top_matches:
        print(f"{Fore.YELLOW}{get_file_name(path)}
{Style.RESET_ALL} (Kesamaan:
{Fore.GREEN}{similarity}%{Style.RESET_A
        LL})")
        print(path)

    print("\n=======================\n")

    # print("Berikut file yang namanya mengandung
        kata yang dicari:")
```

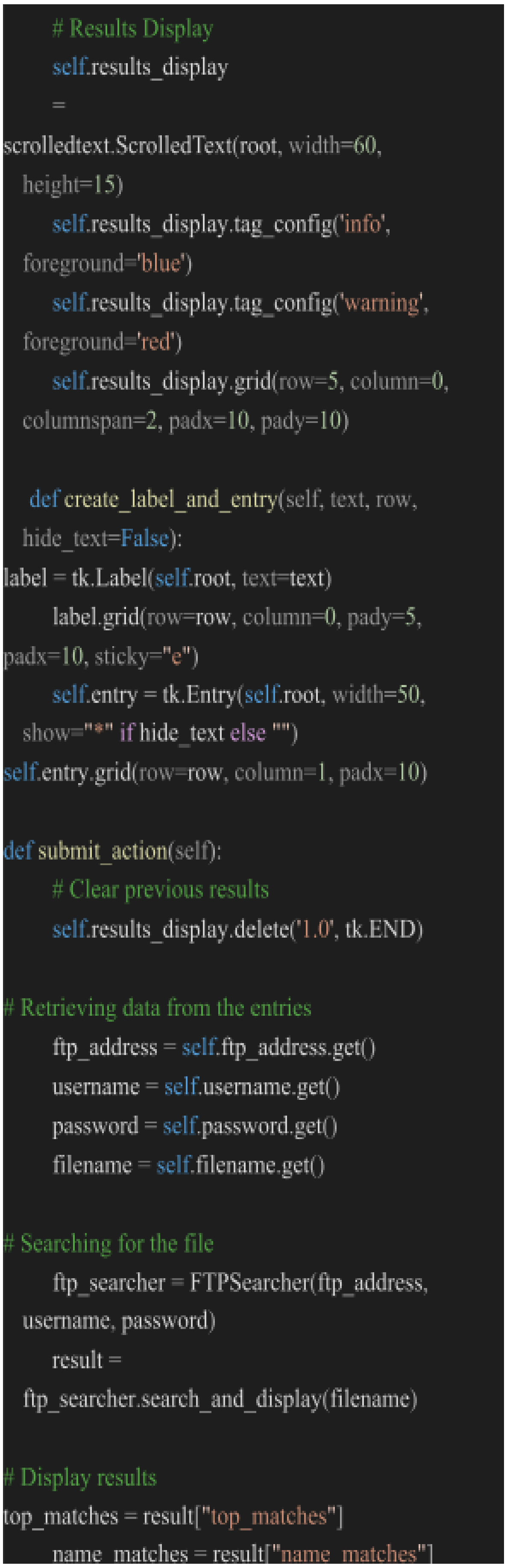


```
print(f"\nTotal file:
{Fore.YELLOW}{len(result)}{Style.RESET_A
        LL}")
print(f"Total folder:
{Fore.YELLOW}{len(folders)}{Style.RESET_
        ALL}")
print(f"Waktu pencarian:
{Fore.YELLOW}{search_time}{Style.RESET_
        ALL} detik")
```

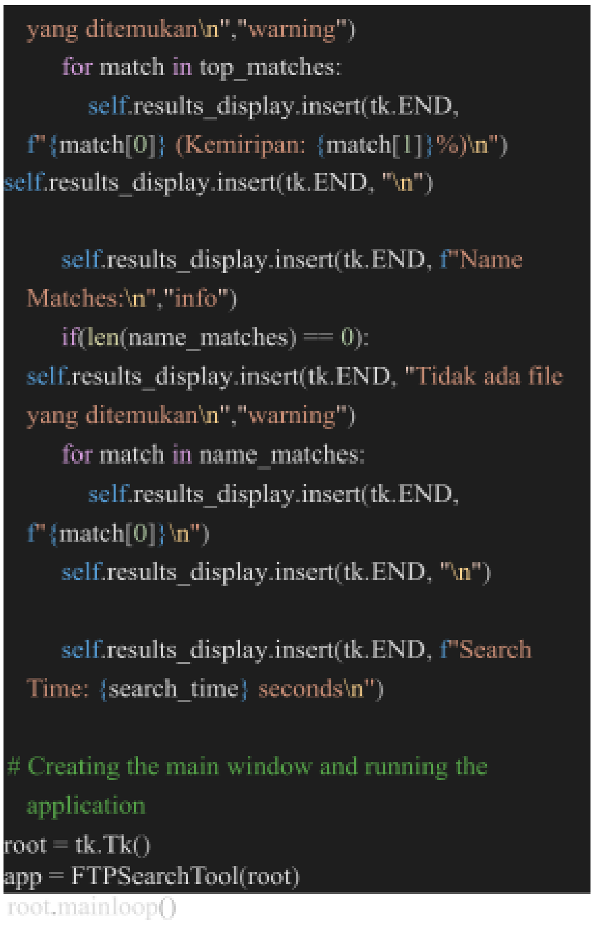
Based on the file name input that the user provides, the aforementioned program looks for files and directories on an FTP server. The program makes use of the FTP module to communicate with the FTP server, OS to manipulate file paths, time to determine the duration of the search, Colorama to produce colored output in the terminal, and fuzzy-wuzzy to determine how similar the file name being searched is to the one on the server. Another tool for making graphical user interfaces is Tkinter (GUI).



```
import tkinter as tk
from tkinter import scrolledtext from
ftp_search import FTPSearcher

class FTPSearchTool:
    def__init__(self, root): self.root
        = root root.title("FTP Search
        Tool")

        # Creating the labels and entries for each field
        self.create_label_and_entry("Alamat FTP
    Server", row=0)
        self.ftp_address = self.entry

        self.create_label_and_entry("Username",
    row=1)
self.username = self.entry

        self.create_label_and_entry("Password", row=2,
    hide_text=True)
self.password = self.entry

        self.create_label_and_entry("Nama file yang
    ingin dicari", row=3)
self.filename = self.entry

# Submit Button
        submit_button = tk.Button(root, text="Submit",
    command=self.submit_action)
submit_button.grid(row=4, column=1, pady=10)
```



```
        # Results Display
        self.results_display
        =
scrolledtext.ScrolledText(root, width=60,
   height=15)
        self.results_display.tag_config('info',
   foreground='blue')
        self.results_display.tag_config('warning',
   foreground='red')
        self.results_display.grid(row=5, column=0,
   columnspan=2, padx=10, pady=10)

    def create_label_and_entry(self, text, row,
   hide_text=False):
label = tk.Label(self.root, text=text)
        label.grid(row=row, column=0, pady=5,
padx=10, sticky="e")
        self.entry = tk.Entry(self.root, width=50,
   show="*" if hide_text else "")
self.entry.grid(row=row, column=1, padx=10)

def submit_action(self):
        # Clear previous results
        self.results_display.delete('1.0', tk.END)

# Retrieving data from the entries
        ftp_address = self.ftp_address.get()
        username = self.username.get()
        password = self.password.get()
        filename = self.filename.get()

# Searching for the file
        ftp_searcher = FTPSearcher(ftp_address,
   username, password)
        result =
   ftp_searcher.search_and_display(filename)

# Display results
top_matches = result["top_matches"]
        name_matches = result["name_matches"]
```



```
    yang ditemukan\n","warning")
        for match in top_matches:
            self.results_display.insert(tk.END,
    f"{match[0]} (Kemiripan: {match[1]}%)\n")
self.results_display.insert(tk.END, "\n")

        self.results_display.insert(tk.END, f"Name
    Matches:\n","info")
        if(len(name_matches) == 0):
    self.results_display.insert(tk.END, "Tidak ada file
    yang ditemukan\n","warning")
        for match in name_matches:
            self.results_display.insert(tk.END,
    f"{match[0]}\n")
        self.results_display.insert(tk.END, "\n")

        self.results_display.insert(tk.END, f"Search
    Time: {search_time} seconds\n")

# Creating the main window and running the
    application
root = tk.Tk()
app = FTPSearchTool(root)
root.mainloop()
```

A summary of each program component is provided below: Here is a summary of each part of the program::

1. Bring in Libraries and Modules: To develop a GUI, import the tkinter modules.

- To construct a text area that can scroll, import the scrolledtext class from the tkinter module.
- To search files on the FTP server, import the FTPSearcher class from the ftp search module.

2. **FTPSearchTool Class:**
   - The GUI elements and user interface are initialized using this class's constructor.
3. **Method Create label and entry:**
   - For every parameter, labels and input entry components are created using this method (FTP Server Address, Username, Password, Name of the file you want to search for).
4. **The method of Submit action:**
   - When the "Submit" button is clicked, this procedure is triggered.
   - Gets values out of inputs (input) that the user enters.
   - Use the search and display method to create an FTPSearcher object and conduct a file search.
   - Shows search results in a text field that may be scrolled.
5. **GUI Display:**
   - Each parameter's labels and entries (FTP Server Address, Username, Password, Name of the file you want to search for).
   - Click "Submit" to begin the search.
   - Search results can be viewed by scrolling over the text box.
6. **Main Window dan Loop:**
   - As the primary window, create an instance of the Tk class.
   - Use the main window as a parameter when creating an instance of the FTPSearchTool class.
   - To maintain the window open, the main loop () is executed using root.

3.1 Process Workflow for Programs

This is the program's primary workflow:

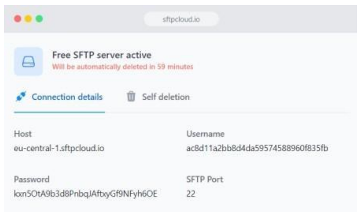
1. Users are asked to input the name of the file they want to look for on the FTP server, the Hostname of the destination FTP Server, the username they will use and the password they will use.
2. Using the input credentials, the script will establish a connection to the designated FTP server.
3. Starting from the first directory, the application will use the DFS (Depth-First Search) method to look for files and directories on the FTP server.
4. Using the fuzz.ratio() function from fuzzywuzzy, the search will compare each file or folder found with the name of the file searched using string similarity calculations.
5. Following the completion of the search, the application will show the results in the form of files with the highest degree of name similarity to the file being looked for, files whose names contain the words being looked for, the total number of files and folders found, and the amount of time the search took..

The fuzzy string matching, or fuzzy-wuzzy, technique is used by this program to locate files that share the same name as the file that must be found.

3.2 Outcomes of the Program

The following points is the output :

1. The FTP server that will be utilized in this test is listed below.



Free SFTP server active
Will be automatically deleted in 59 minutes

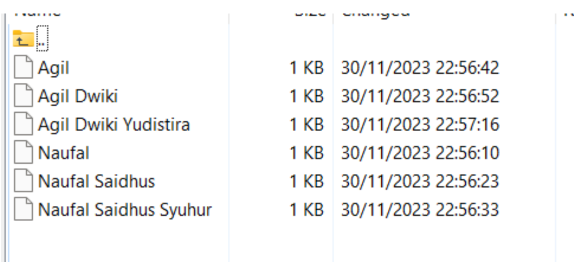
Host
eu-central-1.sftpcloud.io

Username
ac8d11a2bb8d4da59574588960f835fb

Password
kxn5OtA9b3d8PnbqJAftxyGf9NFyh6OE

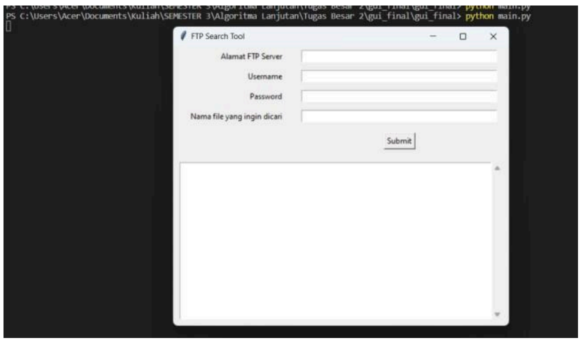
SFTP Port
22

2. The following are some files that exist on the FTP server.



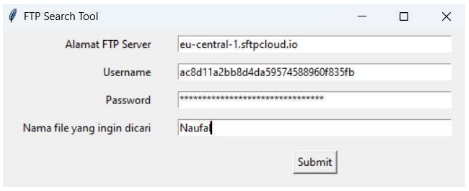
..		
Agil	1 KB	30/11/2023 22:56:42
Agil Dwiki	1 KB	30/11/2023 22:56:52
Agil Dwiki Yudistira	1 KB	30/11/2023 22:57:16
Naufal	1 KB	30/11/2023 22:56:10
Naufal Saidhus	1 KB	30/11/2023 22:56:23
Naufal Saidhus Syuhur	1 KB	30/11/2023 22:56:33

3. Run the python program.



```
PS C:\Users\Acer\Documents\Kuliah\SEMESTER 3\Algoritma Lanjutan\Tugas Besar 2\gui_final\gui_final> python main.py
```

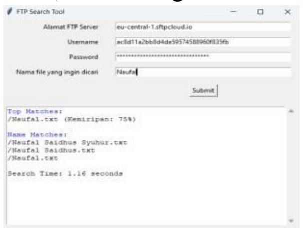
4. Type in the file name you wish to look for, the hostname of the FTP server, the username, and the password..



FTP Search Tool

Alamat FTP Server: eu-central-1.sftpcloud.io
Username: ac8d11a2bb8d4da59574588960f835fb
Password: ********************************
Nama file yang ingin dicari: Naufal

5. If that's the case, the application will search for files that include terms similar to the ones we provide; the findings are shown below.

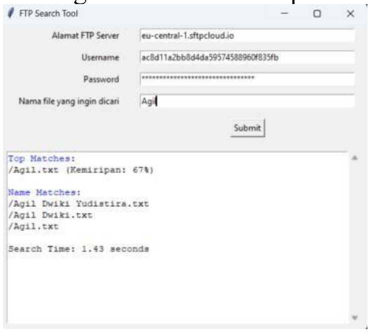


FTP Search Tool

Alamat FTP Server: eu-central-1.sftpcloud.io
Username: ac8d11a2bb8d4da59574588960f835fb
Password: ********************************
Nama file yang ingin dicari: Naufal

```
Top Matches:
/Naufal.txt (Kemiripan: 75%)

Name Matches:
/Naufal Saidhus Syuhur.txt
/Naufal Saidhus.txt
/Naufal.txt

Search Time: 1.16 seconds
```

6. The following uses another example.

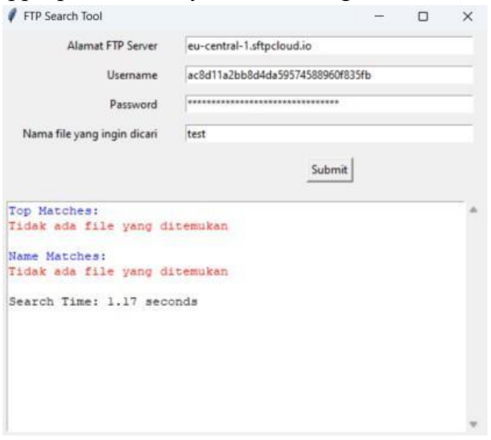


FTP Search Tool

Alamat FTP Server: eu-central-1.sftpcloud.io
Username: ac8d11a2bb8d4da59574588960f835fb
Password: ********************************
Nama file yang ingin dicari: Agil

```
Top Matches:
/Agil.txt (Kemiripan: 67%)

Name Matches:
/Agil Dwiki Yudistira.txt
/Agil Dwiki.txt
/Agil.txt

Search Time: 1.43 seconds
```

7. If the software is unable to locate an appropriate file, try the following.



FTP Search Tool

Alamat FTP Server: eu-central-1.sftpcloud.io
Username: ac8d11a2bb8d4da59574588960f835fb
Password: ********************************
Nama file yang ingin dicari: test

```
Top Matches:
Tidak ada file yang ditemukan

Name Matches:
Tidak ada file yang ditemukan

Search Time: 1.17 seconds
```

4. CONCLUSION

Applying the fuzzywuzzy technique with the DFS (Depth-First Search) Algorithm to File Search on an FTP Server yields numerous significant findings:

1. Search Efficiency: When looking for files on the FTP server, the DFS algorithm is employed. This method traverses the full directory structure to provide thorough searches, but it may be less effective for particularly large directory structures.
2. **Matching Accuracy:** More adaptable matching is possible with the fuzzywuzzy approach. This enables the adjustment of typos, small variations in file names, or spelling variances in the context of file searches, allowing relevant results to be found despite variations in writing or spelling..
3. **Accuracy of Results:** By taking into account the similarity between the searched string and the file name on the server, the DFS and fuzzywuzzy combo improves the accuracy of search results. Even if there are small variations in the file name or mistakes in the search query, this aids in locating the needed file.
4. **Performance Limitations:** Using the fuzzywuzzy approach might raise the computing strain on the server, especially when searching on huge datasets, even though it helps boost search accuracy. Improper management of this could have an impact on server performance.
5. **Adaptability:** In an FTP Server environment, changes or variations in file names are frequently encountered. This approach enables good adaption to these situations. It is a more dynamic approach because of its capacity to handle mistakes or variances in file naming.

In conclusion, using the fuzzywuzzy approach in conjunction with the DFS algorithm on the FTP server helps improve file discovery; nevertheless, system performance must be balanced with search accuracy.

REFERENCE

[1] Alabi, T. H. (2023, September 11). *Python Requirements.txt – How to Create and Pip Install Requirements.txt in Python*. Retrieved from www.freecodecamp.org: https://www.freecodecamp.org/news/python-requirementstxt-explained/

[2] Bachmann, M. (2023, October 11). *pypi.org*. Retrieved from python-Levenshtein 0.23.0: https://pypi.org/project/python-Levenshtein/

[3] Bachmann, M. (2023, November 3). *pypi.org*. Retrieved from rapidfuzz 3.5.2: https://pypi.org/project/rapidfuzz/

[4] Caswell, T. A. (2023, October 7). *pypi.org*. Retrieved from cycler 0.12.1:https://pypi.org/project/cycler/

[5] Clark, J. A. (2023, October 15). *pypi.org*. Retrieved from Pillow 10.1.0: https://pypi.org/project/Pillow/

[6] Cohen, A. (2020, February 14). *pypi.org*. Retrieved from fuzzywuzzy 0.18.0: https://pypi.org/project/fuzzywuzzy/

[7] Cynthia Kustanto, R. M. (n.d.). Penerapan Algoritma Breadth-first Search dan Depth-first Search. *Laboratorium Ilmu dan Rekayasa Komputasi*.

[8] Hartley, J. (2022, October 25). *pypi.org*. Retrieved from colorama 0.4.6: https://pypi.org/project/colorama/ianthomas23. (2023, November 3). *pypi.org*. Retrieved fromcontourpy1.2.0:https://pypi.org/project/contourpy/

[9] J. Postel, J. R. (1985). RFC 959: File Transfer Protocol. *IETF (Internet Engineering Task Force)*.

[10] McGuire, P. (2023, July 30). *pypi.org*. Retrieved from pyparsing 3.1.1: https://pypi.org/project/pyparsing/

[11] Niemeyer, G. (2021, July 14). *python.org*. Retrieved from python-dateutil 2.8.2:https://pypi.org/project/python-dateutil/

[12] Peterson, B. (2021, May 5). *pypi.org*. Retrieved from six 1.16.0: https://pypi.org/project/six/

[13] Pratama, L. G. (2020, December 4). *medium.com*. Retrieved from Contoh implementasi "FuzzyWuzzy" pada pandas

[14] Rossum, J. v. (2023, December 19). *pypi.org*. Retrieved from fonttools 4.47.0:https://pypi.org/project/fonttools/

*[15]*Sedgewick, R. &. (2011). Algorithms (4th ed.). In Addison-Wesley, *Algorithms (4th ed.)*.

[16] Stufft, D. (2023, October 1). *pypi.org*. Retrieved from packaging 23.2:https://pypi.org/project/packaging/

[17] The Nucleic Development Team. (2023, August 24). *pypi.org*. Retrieved from kiwisolver 1.4.5: https://pypi.org/project/kiwisolver/

[18] van Rossum, G. D. (2009). *The Python Language Reference Manual.* Python Software Foundation.