The American University in Cairo
CSCE 3401 - Operating systems
Fall 2022
Project : Disk Analyzer Report

Salma Zaghloul - 900183256
Allaa Elkhouly- 900201771

Table of Contents

Summary 3

Features 3

Implementation 3

Back End 3

Data Structures 4

Engineering Decisions 5

Algorithms 5

GUI Screenshots 5

Summary

- *Boba Disk* is a disk analyzer implemented using **Rust** programming language. Boba Disk is a hard disk space usage analysis and management software. It will search local hard disk drives for files and folders consuming your disk space. It includes features such as, but not limited to, tree-view of folders, the largest files, Images and videos, and installer packages.

- The goal of this project is to help the user, using a user-friendly interface, to better manage their hard disk by finding large files and remove unneeded "space eaters"

Features

- A GUI that is accessible and user-friendly

- View size-sorted folders and files

- Option to select/ select all files to delete.

- Distribute files into categories (images, videos, downloads).

- Automatically detect junk files based on certain characteristics (installer packages)

- Option to view space, space used and space free.

Implementation

Back End

- The back-end program was written using **Rust**. It included the following functions:
    - **view_and_sort:** this function iterates through the directories on the disk, then it views the top largest 50 files with their sizes.
    - **create_directory:** this function is used to help the user create a new directory in a given path.
    - **delete_directory**: this function is used to help the user delete a directory.
    - **rename_directory:** this function is used to help the user rename a directory.

    - **move_directory:** this function is used to help the user to move multiple files or multiple directories to another directory.
    - **size_bar:** this function is used to view the total used and free space on the disk. The goal of this function was to be used to display this data on the front-end as a size bar.
    - **detect_junk:** this function is used to display the files that usually takes up unnecessary space and can be deleted to free space such as installer packages.
    - **videos:** this function is used to collect the total sizes of the videos, as this category takes up huge space on the disk.
    - **Images:** this function is used to collect the total sizes of the images with different extensions such as png, jpg, jpeg,...etc, as this category usually takes up huge space on the disk.
    - **downloads**: this function is used to display the contents of the downloads directory. The downloads directory usually contains unneeded files that can be deleted to free-up space.
- All these functions aim to help the users to have better management of their disk space.

Data Structures

1. **Vectors:** Since we have used Taurie to link the Rust-based back end with the JavaScript-based front end, data was sent between the two ends through vectors. Thus, a vector was created for each function to store the returned data of the function that will be displayed on the GUI.

2. **Structs:** We have created a parent struct that contains two variables (Name, Size), this struct has children in each function. As we traverse through the directories to collect data a child struct is created in which we stored the data of the current iteration. Then we push this struct into the vector. Finally, we iterate over the parent struct to store and output the desired data.

Engineering Decisions

- One of the decisions that we took very early on is to make sure to use the powerful libraries and crates provided by Rust instead of trying to come up with code solutions to problems we face. An example of this is our usage of the Walkdir() which traversed directories on their own.
- We choose not to find duplicate files since that would have required hashing which would not have been efficient considering all the functionalities that we have implemented.
- We choose to show the largest 40 files so as to not overwhelm the user with the output.
- While we chose to use Rust libraries in the backend, we opted out of using Rust libraries in the front-end mainly because of their simplicity. They weren't really visually appealing. We ended up choosing Tauri to help us implement a front end consisting of HTML and CSS linked with JS files that has a RUST file running in the background. That allowed us to implement the intricate design we had in mind.

Algorithms

- Most of the algorithms we used were simple.
    - **Traversal Algorithm:** The first being a simple traversing function. The function would be given a directory then it would go through all the files inside. If it found any more folders, it would go through them, etc. This is similar to traversing a tree, where each path is like a node and each node may represent a subtree itself.
    - **File Extension:** Another algorithm we had was one to check the file type through checking its extension. The function would be given the path and the extension that we need to find, for example, in order to detect images we search for extensions such as JPG, PNG, …etc. Then, it would go through the files and compare them with the extension of said file.