JACIS : Journal Automation Computer Information System Volume: 5, Nomor **2**, **2025**, *Hlm*: 229 - 243

Perbandingan Kinerja Pencarian Berkas pada Struktur Direktori Tingkat Tunggal dan Hierarkis

Nafisa Devi Nur Rusydah *[1] dan Muhammad Ainul Yaqin [2]

[1] Universitas Islam Negeri Maulana Malik Ibrahim Malang; 230605110182@student.uin-malang.ac.id
[2] Universitas Islam Negeri Maulana Malik Ibrahim Malang; yaqinov@ti.uin-malang.ac.id

Abstrak: Pencarian berkas yang efisien menjadi tantangan dalam sistem komputer modern karena peningkatan jumlah dan kedalaman direktori dapat memperlambat akses data. Penelitian ini membandingkan kinerja struktur direktori tingkat tunggal dan hierarkis berdasarkan waktu pencarian berkas. Metode yang digunakan berupa simulasi eksperimental pada dua struktur dengan jumlah dan jenis berkas identik (1.000 e-book). Program uji Python melakukan pencarian terhadap 100 berkas acak menggunakan Linear Search, Breadth-First Search (BFS), dan Depth-First Search (DFS), dengan sepuluh kali pengulangan. Hasil menunjukkan bahwa struktur Single-Level Directory paling efisien, dengan Linear Search mencatat rata-rata 0,002162 detik per berkas. Pada struktur Hierarchical Directory, waktu pencarian meningkat seiring kedalaman. BFS naik dari 0,007890 detik (level 2) hingga 0,080534 detik (level 5), sedangkan DFS meningkat dari 0,011600 detik (level 2) hingga 0,049423 detik (level 4). Secara keseluruhan, BFS dan DFS masing-masing sekitar 20,26× dan 17,70× lebih lambat dibanding struktur tunggal. Temuan ini menegaskan bahwa kedalaman direktori merupakan faktor utama yang memengaruhi efisiensi pencarian, sehingga struktur tingkat tunggal lebih optimal untuk pencarian berbasis pencocokan nama berkas

Keywords: struktur direktori tunggal; struktur direktori hierarkis; pencarian berkas; kinerja sistem

DOI: https://doi.org/10.47134/jacis.v5i2.130
*Correspondensi: Nafisa Devi Nur Rusydah
Email: 230605110182@student.uin-malang.ac.id

Receive: 3 November 2025
Accepted: 23 November 2025
Published: 24 November 2025

Abstract: Efficient file searching remains a major challenge in modern computer systems, as increasing directory size and depth can significantly slow data access. This study compares the performance of single-level and hierarchical directory structures in terms of file search time. An experimental simulation was conducted using two directory models containing identical datasets of 1,000 e-book files. A Python-based testing program was developed to search 100 randomly selected files using three algorithms—Linear Search, Breadth-First Search (BFS), and Depth-First Search (DFS)—with each experiment repeated ten times. The results show that the Single-Level Directory provides the most efficient performance, with Linear Search achieving an average of 0.002162 seconds per file. In the Hierarchical Directory, search time increases with directory depth. BFS rises from 0.007890 seconds at level 2 to 0.080534 seconds at level 5, while DFS increases from 0.011600 seconds at level 2 to 0.049423 seconds at level 4. Overall, BFS and DFS are approximately 20.26× and 17.70× slower, respectively, compared to the single-level structure. These findings demonstrate that directory depth is the dominant factor affecting search efficiency and indicate that single-level directory structures are more optimal for full filename-based search scenarios

Keywords: single-level directory structure; hierarchical directory structure; file searching; system performance

PENDAHULUAN

Penataan berkas (*file organization*) merupakan komponen penting dalam sistem operasi karena berperan langsung dalam menentukan efisiensi akses, pengelolaan, dan pencarian data. Struktur direktori menentukan bagaimana sistem menyimpan dan menemukan informasi, yang secara langsung memengaruhi waktu respons (*response time*) dan performa sistem secara keseluruhan [1]. Terdapat dua pendekatan utama yang digunakan, yaitu struktur direktori tingkat tunggal (*single-level directory*) dan struktur direktori tingkat hierarkis (*hierarchical directory*). Struktur tingkat tunggal menempatkan seluruh berkas dalam satu folder yang sama, sehingga proses pencarian dapat berlangsung cepat karena sistem tidak perlu menelusuri subdirektori [1]. Sebaliknya, struktur tingkat hierarkis menyusun berkas dalam beberapa subdirektori berdasarkan kategori tertentu, yang memudahkan pengelolaan data dalam jumlah besar, namun berpotensi meningkatkan waktu pencarian akibat *path traversal* berlapis [2].

Kinerja pencarian berkas yang tidak efisien umumnya ditandai oleh peningkatan latensi akses, fluktuasi waktu pencarian, serta konsumsi sumber daya yang tinggi ketika tingkat kedalaman direktori bertambah [3], [4]. Masalah ini semakin relevan untuk dikaji karena sistem penyimpanan modern baik lokal maupun berbasis *cloud* mengelola jutaan hingga miliaran file dalam struktur direktori yang kompleks [5]. Oleh karena itu, penelitian mengenai pengaruh struktur direktori terhadap efisiensi pencarian berkas menjadi relevan untuk memahami bagaimana desain organisasi file memengaruhi performa sistem. Pencarian file dalam penelitian ini dilakukan berdasarkan pencocokan nama file secara penuh (*exact filename matching*), sehingga perbandingan benar-benar berfokus pada pengaruh struktur direktori terhadap waktu pencarian.

Berbagai studi sebelumnya telah berupaya mengoptimalkan performa sistem file melalui perbaikan arsitektur metadata dan algoritma pencarian. FlatFS dan FlatFS+ mengusulkan pendekatan flat namespace untuk meratakan struktur hierarkis sehingga mengurangi biaya *path traversal* dan latensi pencarian [1], [2]. Pada sistem ext4 dan XFS, indeksasi direktori dioptimalkan menggunakan algoritma Hashed B-tree (H-Tree) yang mempercepat proses lookup tanpa harus memeriksa setiap entri secara linear [3]. Sementara itu, TABLEFS dan IndexFS memanfaatkan struktur data *Log-Structured Merge Tree* (LSM-Tree) serta table-based namespace untuk meningkatkan efisiensi metadata pada sistem file berskala besar [4], [5]. Pendekatan *speculative path resolution* dan pemisahan akses metadata juga diusulkan oleh InfiniFS dan PhatKV untuk mengurangi latensi path lookup pada sistem terdistribusi [6], [7]. Hu dan Wang [8] juga menunjukkan bahwa pengelolaan dentry yang efisien dalam struktur LSM-Tree dapat mempercepat proses lookup metadata secara signifikan. Dalam konteks sistem file terdistribusi, peneliti [9] menekankan bahwa desain namespace dan organisasi direktori berpengaruh besar terhadap latensi metadata.

Selanjutnya, peneliti [10] memperkenalkan DirectFS, sebuah sistem file terdistribusi berbasis RDMA yang memanfaatkan CPU-oblivious metadata indexing untuk mempercepat pencarian dan mengurangi overhead path-walk. Penelitian [11] turut memperlihatkan bahwa sistem metadata berskala besar sering kali terhambat oleh kompleksitas direktori dan jumlah

entri yang tinggi. Selain itu peneliti [12] juga mengatakan bahwa melalui tinjauan komprehensif pada sistem file terdistribusi juga menegaskan bahwa scalability dan efisiensi metadata sangat bergantung pada desain struktur direktori serta strategi pengindeksan yang digunakan. Dalam konteks lokal, algoritma *Breadth-First Search* (BFS) untuk mempercepat pencarian file pada direktori Windows dan menemukan bahwa BFS efektif dalam menelusuri hierarki direktori secara sistematis [13]. Sebagai pembanding terhadap metode BFS, algoritma *Depth-First Search* (DFS) yang secara luas digunakan dalam proses *traversal graf* [14] juga berpotensi diterapkan dalam penelusuran direktori sistem file karena kemampuannya menjelajahi struktur bertingkat secara mendalam. Penelitian ini menambahkan DFS untuk analisis komparatif efisiensi *traversal* pada direktori bertingkat. Meskipun struktur tunggal lebih cepat, struktur hierarkis tetap dominan pada sistem file modern karena kemampuannya mengorganisasi berkas secara logis pada skala penyimpanan besar.

Selain itu, penelitian [15] melakukan analisis menyeluruh terhadap efisiensi parallel Breadth-First Search, yang menunjukkan bahwa algoritma ini dapat dioptimalkan untuk *traversal* direktori berskala besar. Peneliti [16] juga melakukan studi perbandingan algoritma pencarian (termasuk *Linear Search*) pada struktur data berskala besar, dan menegaskan bahwa kompleksitas algoritma berpengaruh signifikan terhadap waktu pencarian. Selain itu struktur direktori memiliki dampak signifikan terhadap efisiensi pemrosesan file besar pada sistem terdistribusi[17]. Studi oleh [18] pada *Billion-files File Systems* (BfFS) mendukung temuan tersebut, di mana peningkatan kedalaman direktori menyebabkan kenaikan signifikan pada waktu read/write dan metadata overhead. Hal ini menegaskan bahwa hierarki direktori memiliki dampak langsung terhadap performa pencarian berkas.

Meskipun berbagai penelitian telah meneliti optimasi metadata dan *traversal* direktori, hingga saat ini belum terdapat penelitian yang secara empiris membandingkan kinerja waktu pencarian berkas antara struktur direktori tingkat tunggal dan struktur direktori tingkat hierarkis menggunakan kondisi jumlah dan jenis berkas yang identik serta melibatkan lebih dari satu algoritma pencarian secara bersamaan. Berdasarkan telaah literatur, belum ditemukan penelitian yang secara simultan menguji algoritma *Linear Search*, *Breadth-First Search* (BFS), dan *Depth-First Search* (DFS) pada dua tipe struktur direktori dengan kondisi pengujian yang seragam. Oleh karena itu, penelitian ini berupaya mengisi kesenjangan tersebut dengan melakukan simulasi terkontrol untuk membandingkan efisiensi waktu pencarian berkas antara dua struktur direktori tersebut menggunakan algoritma *Linear Search*, *Breadth-First Search* (BFS), dan *Depth-First Search* (DFS) berbasis Python.

Berdasarkan uraian latar belakang tersebut, permasalahan utama yang diangkat dalam penelitian ini adalah menilai sejauh mana struktur organisasi direktori berpengaruh terhadap efisiensi pencarian berkas pada sistem penyimpanan lokal. Permasalahan ini penting karena efisiensi pencarian berkas berimplikasi langsung terhadap performa sistem file dan produktivitas pengguna, khususnya pada lingkungan dengan jumlah berkas yang besar dan kompleksitas hierarki yang tinggi.

Kebaruan (*novelty*) penelitian ini terletak pada tiga aspek utama. Pertama, penelitian ini melakukan pengujian eksperimental langsung terhadap dua struktur direktori tingkat tunggal dan hierarkis dengan kondisi jumlah dan jenis berkas yang identik, sesuatu yang masih jarang dilakukan secara empiris dalam penelitian terdahulu, sehingga hasil dapat dibandingkan secara adil. Kedua, eksperimen dilakukan per tingkat kedalaman (level) hingga

lima level hierarki untuk mengevaluasi dampak kedalaman terhadap waktu pencarian berkas. Ketiga, penelitian ini menggunakan tiga algoritma sekaligus (*Linear Search*, BFS dan DFS) dengan pendekatan *paired sampling* dan analisis statistik *paired t-test* guna memastikan bahwa perbedaan waktu pencarian yang diperoleh bersifat signifikan secara ilmiah. Penelitian ini dibatasi pada analisis efisiensi waktu pencarian berkas dengan pencocokan nama file secara penuh (*exact filename matching*). Aspek lain seperti keamanan berkas, organisasi folder, hak akses, maupun kinerja baca/tulis tidak menjadi fokus penelitian. Selain itu, penelitian ini tidak mengevaluasi aspek pengalaman pengguna (*user perspective*) seperti preferensi organisasi berkas, kemudahan navigasi, atau praktik penggunaan direktori, karena aspek tersebut memerlukan pendekatan survei atau studi pengguna yang tidak termasuk dalam ruang lingkup eksperimen teknis ini.

Dengan demikian, tujuan utama penelitian ini adalah untuk (1) membandingkan waktu pencarian berkas pada struktur direktori tingkat tunggal dan hierarkis, (2) menganalisis pengaruh kedalaman direktori terhadap waktu pencarian, serta (3) memberikan rekomendasi pemilihan struktur direktori yang optimal berdasarkan efisiensi pencarian.

Penelitian ini diharapkan dapat memberikan kontribusi empiris bagi pengembang sistem file, administrator sistem, maupun akademisi di bidang sistem operasi yang ingin memahami hubungan antara struktur organisasi berkas dan performa pencarian data di lingkungan komputasi modern.

METODE

Desain Penelitian

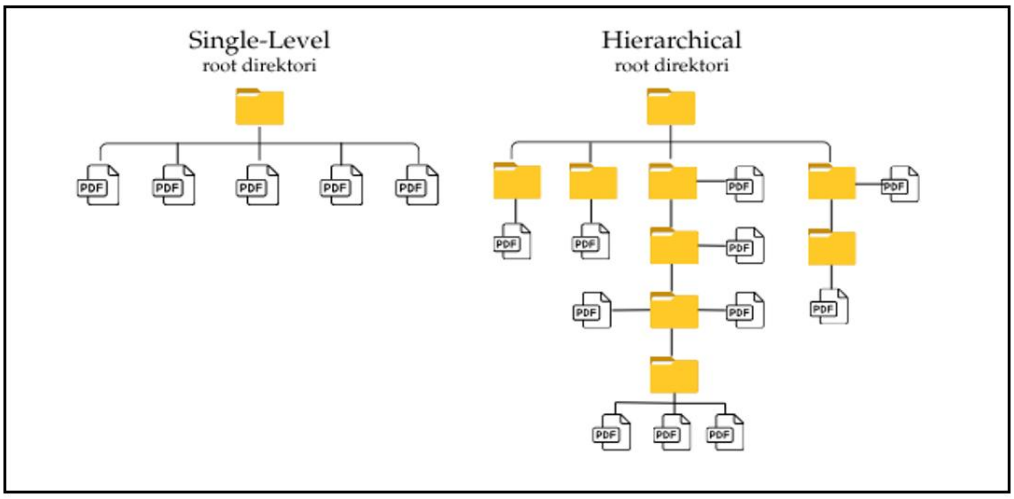
Penelitian ini menerapkan desain eksperimen simulatif komparatif untuk mengevaluasi perbedaan kinerja dua model struktur direktori dalam proses pencarian berkas digital. Desain ini memungkinkan pengujian terkontrol terhadap variabel bebas berupa jenis struktur direktori (*single level* dan *hierarchical*), dengan variabel terikat berupa waktu pencarian per berkas. Seluruh eksperimen dijalankan pada media penyimpanan eksternal (flashdisk) untuk memastikan lingkungan pengujian yang terisolasi dari sistem operasi utama.

Struktur direktori yang diuji meliputi:

1. *Single-Level* (*Flat Directory*): 1000 berkas e-book disimpan dalam satu direktori tanpa subfolder.
2. *Hierarchical* (*Multi-Level Directory*): 1000 berkas yang sama didistribusikan ke beberapa subfolder dengan kedalaman hingga lima level.

Seluruh struktur uji ditempatkan pada direktori D: (*flashdisk*), yang terdiri atas:

1. D:/SingleLevel, memuat seluruh berkas dalam satu folder.
2. D:/Hierarchical, memuat berkas yang sama dalam struktur bertingkat hingga level ke-5.
3. D:/ProgramUji, berisi skrip uji_pencarian.py dan file hasil pengujian (CSV).



Gambar 1. Struktur direktori tingkat tunggal dan hierarkis

Prosedur pengujian

Eksperimen dilakukan menggunakan skrip Python *uji_pencarian.py* untuk mengukur performa tiga algoritma pencarian *Linear Search, Breadth-First Search* (BFS), dan *Depth-First Search* (DFS) pada dua struktur direktori (*single-level* dan *hierarchical*) dengan dataset identik. Langkah-langkah pengujian meliputi:

1. Pemilihan *Sampel* File

   Dari 1000 e-book pada kedua struktur, sistem memilih 100 file secara acak untuk setiap siklus uji menggunakan `random.sample()` tanpa penggantian, sehingga setiap file memiliki peluang yang sama dan tidak terulang pada siklus yang sama.

2. Pencarian File di Struktur *Single-Level*

   Pencarian dilakukan menggunakan `linear_search()` yang menelusuri seluruh file secara sekuensial. BFS (`bfs_search()`) dan DFS (`dfs_search()`) tetap dijalankan untuk menjaga konsistensi perbandingan meskipun direktori hanya memiliki satu tingkat. Waktu eksekusi setiap algoritma dicatat menggunakan `time.perf_counter()`.

3. Pencarian File di Struktur *Hierarchical*

   Linear Search Hierarchical diterapkan melalui `linear_search_hierarchical()` berbasis os.walk() untuk menelusuri subdirektori hingga lima level. BFS memanfaatkan struktur *queue*, sedangkan DFS menggunakan *stack* untuk menelusuri direktori secara mendalam.

4. Pengukuran Waktu Eksekusi

   Durasi pencarian dihitung dari selisih waktu mulai dan selesai yang diperoleh melalui `time.perf_counter()`, menghasilkan nilai waktu dalam satuan detik.

5. Repetisi dan Perekaman Data

6. Setiap siklus pengujian diulang 10 kali untuk meningkatkan reliabilitas dan meminimalkan *random noise*. Hasil pengujian disimpan dalam tiga berkas CSV:

a. `hasil_uji_final.csv` untuk mencatat waktu pencarian seluruh algoritma pada kedua struktur.

b. `per_file_summary.csv` untuk merangkum rata-rata waktu pencarian per file.

c. `per_level_summary.csv` untuk merangkum waktu pencarian berdasarkan kedalaman direktori.

Seluruh berkas hasil disimpan pada direktori D:/ProgramUji/.

Populasi, Sampel dan Teknik Sampling

Populasi

Populasi penelitian terdiri atas 1000 file e-book pada Drive D:, dengan ukuran 50 KB hingga 93 MB dan berasal dari berbagai kategori (*Programming*, *Mathematics*, *Science*, *Computer Networks*, dan lainnya) dalam format PDF. Seluruh file diuji kelayakannya terlebih dahulu (tidak rusak, dapat diakses, dan memiliki izin baca). File yang tidak memenuhi syarat dikeluarkan dari populasi.

Sampel

Sebanyak 100 file dipilih secara acak dari populasi menggunakan fungsi `random.sample()` tanpa penggantian untuk mencegah duplikasi antar-uji. Daftar sampel digunakan secara konsisten pada seluruh 10 repetisi pengujian untuk memastikan kesetaraan kondisi bagi ketiga algoritma (Linear Search, BFS, dan DFS) pada kedua struktur direktori. Pemilihan sampel disimpan untuk menjamin replikasi. Ukuran sampel 100 mengacu pada prinsip *large-sample approximation* dalam *Central Limit Theorem* (CLT), sehingga estimasi rata-rata stabil dan memadai untuk analisis statistik seperti *paired t-test*, sekaligus efisien untuk pengujian berulang.

Teknik Sampling

Penelitian menggunakan teknik *paired* sampling, yaitu file yang sama diuji pada dua kondisi struktur direktori (*single-level* dan *hierarchical*). Pendekatan ini meminimalkan variabilitas antar-file, sehingga perbedaan waktu pencarian lebih merefleksikan pengaruh struktur direktori daripada karakteristik file lainnya.

Instrumen Penelitian

1. Perangkat Keras

   Eksperimen dijalankan pada laptop MSI Cyborg 15 A13UCK dengan spesifikasi:

   a. Intel® Core™ i7-13620H (10 core, 16 thread, hingga 4.9 GHz)

   b. RAM: 16 GB DDR5 5200 MHz

   c. NVMe SSD 512 GB Micron 2400

   d. Windows 11 Home 64-bit

   e. GPT (GUID Partition Table)

   Spesifikasi dicatat untuk memastikan replikasi hasil dan mengidentifikasi potensi pengaruh performa perangkat keras terhadap waktu pencarian. Seluruh pengujian

dilakukan pada flashdisk Drive D: sebagai media penyimpanan sekunder untuk mensimulasikan kondisi penggunaan nyata.

2. Perangkat Lunak

   Eksperimen dilakukan menggunakan Python 3.11 dengan pustaka standar, meliputi:

   a. *os* untuk membaca isi direktori,

   b. *collections.deque* untuk implementasi antrian pada BFS,

   c. *csv* untuk menyimpan hasil uji ke file,

   d. *random* untuk pengambilan sampel file acak,

   e. *time* untuk mengukur durasi eksekusi,

   f. *statistics* untuk menghitung rata-rata dan simpangan baku hasil uji.

   g. *matplotlib.pyplot* → pembuatan grafik analisis level

   Pengujian dilakukan dalam kondisi sistem yang stabil tanpa beban latar belakang signifikan untuk menjaga konsistensi waktu eksekusi.

3. Skrip Uji

   Seluruh proses penelitian diimplementasikan dalam skrip utama `uji_pencarian.py`, yang mencakup:

   a. `linear_search()` untuk pencarian sekuensial pada struktur *single level*

   b. `linear_search_hierarchical()` untuk pencarian rekursif pada struktur *hierarchical* berbasis `os.walk`

   c. `bfs_search()` untuk pencarian berbasis BFS pada kedua struktur

   d. `dfs_search()` sebagai pembanding tambahan pada kedua struktur

   e. `get_random_files()` untuk pemilihan 100 file sampel,

   f. `main()` sebagai pengendali eksekusi, repetisi, dan perekaman hasil ke file CSV

Untuk memperjelas implementasi algoritma pencarian yang digunakan, Algoritma 1 berikut menampilkan potongan kode utama dari skrip `uji_pencarian.py`.

Algoritma 1 Potongan kode fungsi pencarian linear, BFS dan DFS pada skrip uji_pencarian.py

```
def linear_search(folder_path, target_file):
    try:
        files = os.listdir(folder_path)
        for f in files:
            if f == target_file:
        return os.path.join(folder_path, f), 1  # level 1
    return None, 1
  except (FileNotFoundError, PermissionError):
        return None, 1
def bfs_search(start_path, target_file):
    queue = deque([(start_path, 1)])
    while queue:
        current_dir, level = queue.popleft()
        try:
            for item in os.listdir(current_dir):
                item_path = os.path.join(current_dir, item)
```

```
                if os.path.isfile(item_path) and item == target_file:
                    return item_path, level
                elif os.path.isdir(item_path) and level < 5:
                    queue.append((item_path, level + 1))
        except PermissionError:
            continue
    return None, "-"
def linear_search_hierarchical(start_path, target_file):
    try:
        for root, dirs, files in os.walk(start_path):
            for f in files:
                if f == target_file:
                    level = root.replace(start_path, "").count(os.sep) + 1
                    return os.path.join(root, f), level
        return None, "-"
    except PermissionError:
        return None, "-"
def dfs_search(start_path, target_file):
    stack = [(start_path, 1)]
    while stack:
        current_dir, level = stack.pop()
        try:
            for item in os.listdir(current_dir):
                item_path = os.path.join(current_dir, item)
                if os.path.isfile(item_path) and item == target_file:
                    return item_path, level
                elif os.path.isdir(item_path) and level < 5:
                    stack.append((item_path, level + 1))
        except PermissionError:
            continue
    return None, "-"
```

Berdasarkan Algoritma 1, skrip *uji_pencarian.py* mengimplementasikan tiga algoritma pencarian *Linear Search, Breadth-First Search (BFS),* dan *Depth-First Search (DFS)* yang digunakan secara konsisten pada kedua struktur direktori sesuai kebutuhan pengujian.

Algoritma Pencarian yang Digunakan

1. *Linear Search*

   Linear Search merupakan algoritma pencarian dasar yang menelusuri setiap elemen secara berurutan hingga menemukan target atau mencapai akhir daftar. Algoritma ini memiliki kompleksitas waktu O(n), dengan n sebagai jumlah file dalam direktori. Metode ini efektif untuk pencarian sederhana karena tidak memerlukan struktur data tambahan.

2. *Breadth-First Search* (BFS)

   BFS merupakan algoritma pencarian berbasis *graph traversal* yang memanfaatkan struktur data *queue* untuk menjelajahi direktori secara bertingkat, dimulai dari level teratas sebelum memasuki subdirektori yang lebih dalam. Kompleksitas waktu algoritma ini mendekati O(V + E), dengan V sebagai jumlah folder dan E sebagai jumlah relasi antar-folder. BFS dipilih karena mampu menelusuri struktur direktori yang kompleks tanpa risiko *stack overflow* serta memberikan kontrol yang baik terhadap tingkat kedalaman pencarian.

3. *Depth-First Search* (DFS)

   DFS adalah algoritma pencarian mendalam yang menelusuri cabang direktori hingga titik terdalam sebelum kembali naik ke level sebelumnya. Pada penelitian ini digunakan implementasi iteratif berbasis *stack,* sehingga aman dari risiko *stack*

overflow. DFS efektif untuk eksplorasi menyeluruh terhadap struktur direktori multi-level.

Validitas dan Reliabilitas Instrumen

Untuk memastikan keandalan hasil, penelitian menerapkan beberapa prosedur validasi, yaitu

1. Setiap pengujian diulang sebanyak 10 kali untuk mengurangi variabilitas acak dan memperoleh hasil yang stabil.
2. File yang diuji identik di kedua struktur, sehingga faktor *eksternal* (ukuran dan nama file) tidak memengaruhi hasil.
3. Seluruh proses dijalankan pada kondisi sistem *idle* untuk mencegah gangguan performa dari proses lain.
4. Data hasil disimpan secara otomatis dalam file CSV agar dapat diverifikasi dan dianalisis ulang dengan perangkat statistik seperti Excel.
5. Sampel file dan waktu pencarian diverifikasi manual pada sebagian data untuk memastikan validitas *fungsional skrip*.

Nilai simpangan baku antar-pengulangan digunakan sebagai indikator *reliabilitas*. Nilai simpangan baku berada pada rentang yang konsisten dan tidak menunjukkan anomali, sehingga instrumen pengujian dinyatakan stabil dan *reprodusibel*.

Analisis Data

Analisis dilakukan secara kuantitatif untuk mengukur performa kedua struktur. Langkah-langkah analisis meliputi:

1. Menghitung rata-rata waktu pencarian (*mean*) dan simpangan baku (*standard deviation*) dari setiap pengujian.
2. Melakukan uji-t *paired* untuk menguji perbedaan waktu pencarian antara struktur direktori tingkat tunggal dan *hierarkis*.
3. Menganalisis hubungan antara kedalaman direktori dan waktu pencarian rata-rata per level.
4. Menampilkan hasil analisis dalam bentuk grafik boxplot dan tabel perbandingan untuk memperjelas *tren* performa.

HASIL DAN PEMBAHASAN

Hasil Eksperimen

Eksperimen dilakukan dalam sepuluh siklus pengujian untuk membandingkan kinerja pencarian berkas pada dua struktur direktori, yaitu *Single-Level Directory* dan *Hierarchical Directory*. Dataset terdiri atas 1.000 berkas e-book dengan ukuran 50 KB hingga 93 MB. Pada setiap siklus, sistem secara acak memilih 100 berkas dan melakukan pencarian menggunakan tiga algoritma *Linear Search, Breadth-First Search* (BFS), dan *Depth-First Search* (DFS) yang diterapkan pada kedua struktur direktori untuk memastikan perbandingan yang konsisten.

Waktu eksekusi setiap pencarian diukur menggunakan `time.perf_counter()` dengan presisi mikrodetik, kemudian dirata-ratakan untuk memperoleh *average search time per file*.

Analisis Rata-Rata Berdasarkan Level Hierarki

Rangkuman hasil pengujian waktu pencarian berdasarkan tingkat kedalaman folder (*directory level*) disajikan pada Tabel 1. Data menunjukkan kecenderungan bahwa semakin dalam posisi file dalam struktur hierarki, semakin besar pula waktu pencarian yang dibutuhkan.

Tabel 1. Rata-rata waktu pencarian per level direktori

Struktur Direktori	Algoritma	Level Direktori	Rata-rata Waktu Pencarian (s/file)	Standar Deviasi	Efisiensi Relatif terhadap Single-Level
Single-Level	Linear Search	Level 1	0.002162	0.000376	100% (acuan)
Hierarchical	BFS	Level 2	0.007890	0.004514	3.65× lebih lambat
Hierarchical	BFS	Level 3	0.027233	0.003975	12.60× lebih lambat
Hierarchical	BFS	Level 4	0.058550	0.013842	27.09× lebih lambat
Hierarchical	BFS	Level 5	0.080534	0.007210	37.26× lebih lambat
Hierarchical (hingga 5 level)	BFS	Rata-rata keseluruhan	0.043787	0.025032	20.26× lebih lambat
Hierarchical	DFS	Level 2	0.011600	0.023454	5.37× lebih lambat
Hierarchical	DFS	Level 3	0.039051	0.015523	18.07× lebih lambat
Hierarchical	DFS	Level 4	0.049423	0.017039	22.86× lebih lambat
Hierarchical	DFS	Level 5	0.022490	0.003415	10.40× lebih lambat
Hierarchical (hingga 5 level)	DFS	Rata-rata keseluruhan	0.038272	0.023267	17.70× lebih lambat

Sumber: hasil eksekusi skrip uji_pencarian.py.

Hasil pengujian menunjukkan bahwa struktur *Single-Level Directory* yang diuji menggunakan algoritma *Linear Search* memiliki waktu pencarian rata-rata sebesar 0.002162 detik per berkas dan dijadikan sebagai acuan efisiensi 100%. Pada struktur hierarkis, waktu pencarian meningkat secara konsisten seiring bertambahnya kedalaman direktori. Pada algoritma *Breadth-First Search* (BFS), rata-rata waktu pencarian meningkat dari 0.007890 detik (level 2), menjadi 0.027233 detik (level 3), 0.058550 detik (level 4), dan mencapai 0.080534 detik pada level 5. Secara keseluruhan, BFS mencatat rata-rata 0.043787 detik per file, atau sekitar 20.26 kali lebih lambat dibanding pencarian pada struktur tunggal.

Pada algoritma *Depth-First Search* (DFS), pola yang dihasilkan sedikit berbeda. Waktu pencarian meningkat dari 0.011600 detik (level 2), menjadi 0.039051 detik (level 3), dan 0.049423 detik (level 4), namun kemudian turun secara signifikan pada level 5 menjadi 0.022490 detik per file. Secara keseluruhan, DFS menghasilkan rata-rata waktu pencarian 0.038272 detik, atau sekitar 17.70 kali lebih lambat dibanding struktur tunggal. Temuan ini menegaskan bahwa kedalaman direktori merupakan faktor dominan yang memengaruhi latensi pencarian berkas, dan pola kinerja masing-masing algoritma konsisten dengan teori kompleksitasnya.

Analisis Rata-Rata per File

Rincian waktu pencarian untuk setiap file terekam dalam berkas `hasil_uji_final.csv`. Sebagai representasi, Tabel 2 berikut menampilkan sepuluh contoh berkas pertama dengan variasi level direktori.

Tabel 2. Contoh hasil rata-rata waktu pencarian per file (10 sampel pertama)

No.	Nama File	Level Hierarki	Rata-rata Waktu Single-Level (s)	Rata-rata Waktu Hierarchical (s)	Keterangan
1	Mathematics_of_Finance,_Seventh_Edition_Robert_L_Brown;_Steve_Kopp.pdf	3	0.002253	0.020574	Level 3 waktu hierarki moderat
2	How to Teach AI.pdf	4	0.002170	0.083786	Level 4 waktu hierarki tinggi
3	INTRODUCTION TO DESIGN AND MAKING.pdf	2	0.002320	0.007685	Level 2 waktu cepat
4	Methods_Of_Geometry_In_The_Theory_Of_Partial_Differential_Equations.pdf	4	0.002141	0.058220	Level 4 meningkat signifikan
5	Data Insight Foundations _ Step-by-Step Data Analysis with R.pdf	3	0.002224	0.026599	Level 3 waktu sedang
6	Concept_and_Reality_in_Early_Buddhist_Thought_An_Essay_on_Papanca.pdf	2	0.002140	0.004654	Level 2 yang paling cepat
7	Financial_Instrument_Pricing_Using_C++_2018,_Wiley_libgen_li.pdf	4	0.002148	0.056002	Level 4 tinggi namun stabil
8	Opening the Hand of Thought, Revised and Expanded Edition.pdf	2	0.002104	0.009488	Level 2 waktu rendah
9	Demystifying_Deep_Learning_An_introduction_to_the_mathematics_of.pdf	4	0.002130	0.037359	Level 4 meningkat
10	Mastering_DeepSeek_v3_Unlocking_Advanced_Features_for_Expert_Users.pdf	4	0.002180	0.076045	Level 4 salah satu tertinggi

Sumber: hasil_uji_final.csv.

Hasil pada Tabel 2 menunjukkan bahwa waktu pencarian pada struktur *hierarkis* meningkat seiring dengan bertambahnya kedalaman direktori. File yang berada pada level rendah (level 2) memiliki waktu pencarian relatif cepat, yaitu di bawah 0.01 detik, sedangkan file pada level yang lebih dalam, seperti level 4, menunjukkan peningkatan waktu hingga kisaran 0.05–0.08 detik. Pola ini menegaskan bahwa semakin dalam lokasi file dalam struktur folder, semakin besar biaya *traversal* yang harus dilakukan algoritma dalam menemukan berkas tersebut.

Hasil Pencarian File pada 10 Pengulangan

Setiap pengujian dilakukan sebanyak sepuluh kali untuk memastikan stabilitas hasil. File uji yang digunakan pada Tabel 3 sama dengan sepuluh file representatif yang tercantum pada Tabel 2.

Tabel 3. Hasil waktu pencarian per file pada 10 pengulangan (10 sampel pertama)

No	Uji-1	Uji-2	Uji-3	Uji-4	Uji-5	Uji-6	Uji-7	Uji-8	Uji-9	Uji-10	Rata-rata(s)
1	0.01977	0.02107	0.01918	0.01821	0.03173	0.01852	0.01816	0.01859	0.02091	0.01960	0.02057
2	0.07792	0.10841	0.06956	0.06323	0.09916	0.06700	0.06919	0.07020	0.06767	0.08607	0.07792
3	0.00768	0.00780	0.00752	0.00789	0.00935	0.00774	0.00773	0.00770	0.00791	0.00801	0.00784
4	0.05822	0.05168	0.05851	0.05815	0.07968	0.05802	0.06889	0.05770	0.05850	0.05547	0.06058
5	0.02660	0.02801	0.02678	0.02688	0.04238	0.02659	0.02655	0.02668	0.02651	0.02680	0.02838
6	0.00465	0.00449	0.00447	0.00448	0.00579	0.00435	0.00461	0.00438	0.00572	0.00464	0.00470
7	0.05600	0.06229	0.05621	0.05635	0.07436	0.05607	0.05622	0.05633	0.05632	0.05825	0.05801
8	0.00949	0.00940	0.00943	0.00939	0.01287	0.00940	0.00941	0.00941	0.00951	0.00949	0.00963
9	0.03736	0.03503	0.03625	0.03720	0.05200	0.03628	0.03650	0.03614	0.03748	0.03635	0.03736
10	0.07466	0.07996	0.07348	0.07280	0.08549	0.07441	0.07499	0.07022	0.07463	0.07981	0.07604

Sumber: hasil_uji_final.csv

Hasil pada Tabel 3 menunjukkan bahwa variasi waktu pencarian antarpengulangan relatif kecil pada file yang berada di level rendah (level 2), dengan deviasi sekitar ±2%–5% dari nilai rata-ratanya. Hal ini menunjukkan bahwa proses pencarian pada kedalaman dangkal berlangsung stabil dan konsisten. Pada file yang berada pada level yang lebih dalam (level 3–4), deviasi waktu pengulangan meningkat hingga sekitar ±10–20%, akibat jumlah subdirektori lebih banyak sehingga variasi *traversal* lebih besar. Meskipun demikian, pola kinerja tetap konsisten: file yang berada pada level lebih dalam secara konsisten membutuhkan waktu pencarian lebih lama daripada file level 2. Hal ini menegaskan bahwa kedalaman direktori merupakan faktor utama yang menyebabkan peningkatan latensi pencarian.

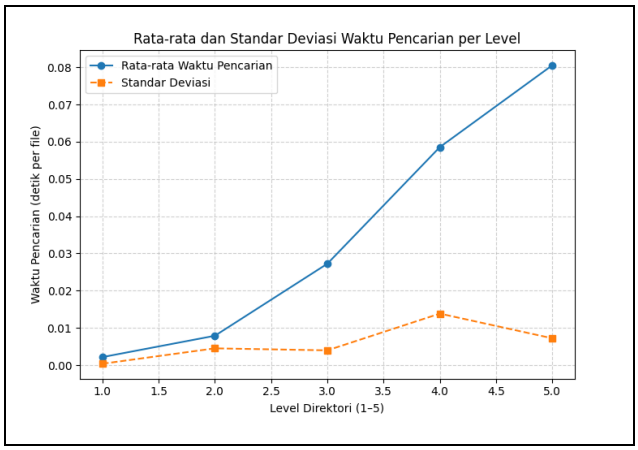
Pembahasan

Hasil pengujian menunjukkan bahwa pada struktur *Single-Level Directory* yang diuji menggunakan algoritma *Linear Search*, waktu pencarian rata-rata hanya 0.002162 detik per file, sehingga digunakan sebagai acuan efisiensi 100%. Sebaliknya, pada struktur *Hierarchical Directory*, baik algoritma *Breadth-First Search* (BFS) maupun *Depth-First Search* (DFS) menunjukkan peningkatan waktu pencarian yang cukup signifikan seiring bertambahnya kedalaman direktori.

Pada algoritma BFS, rata-rata waktu pencarian meningkat dari 0.007890 detik (level 2), menjadi 0.027233 detik (level 3), 0.058550 detik (level 4), dan mencapai 0.080534 detik (level 5). Secara keseluruhan, BFS mencatat rata-rata 0.043787 detik per file, atau sekitar 20.26 kali lebih lambat dibandingkan struktur tunggal. Sementara itu, pada algoritma DFS, waktu pencarian mencapai 0.011600 detik (level 2), meningkat menjadi 0.039051 detik (level 3) dan 0.049423 detik (level 4), kemudian menurun pada level 5 menjadi 0.022490 detik, dengan rata-rata keseluruhan 0.038272 detik atau sekitar 17.70 kali lebih lambat dari struktur tunggal.

Gambar 2 memperlihatkan bahwa waktu pencarian meningkat seiring bertambahnya kedalaman direktori. Series 1 (garis biru) menunjukkan rata-rata waktu pencarian per level, sedangkan Series 2 (garis oranye) menunjukkan nilai standar deviasi. Rata-rata waktu pencarian meningkat dari sekitar 0.003 detik pada *single-level* menjadi 0.008 detik pada level 2, kemudian naik lebih tajam pada level 3 (≈0.028 detik), level 4 (≈0.059 detik), dan mencapai

nilai tertinggi pada level 5 (≈0.081 detik). Tren ini menegaskan bahwa semakin dalam struktur folder, semakin besar jumlah node yang harus ditelusuri sehingga beban *traversal* meningkat.



Gambar 2. Rata-rata dan standar deviasi waktu pencarian per level direktori.

Nilai standar deviasi juga meningkat dari sekitar 0.001 detik pada *single-level* menjadi 0.005 detik pada level 2, relatif stabil pada level 3 (≈0.004 detik), kemudian naik pada level 4 (≈0.014 detik) dan sedikit menurun pada level 5 (≈0.008 detik). Peningkatan deviasi menunjukkan bahwa variasi waktu antar-pengulangan semakin besar pada level yang lebih dalam karena struktur direktori yang lebih kompleks.

Fenomena ini menunjukkan bahwa efisiensi pencarian tidak hanya dipengaruhi oleh algoritma, tetapi juga oleh morfologi struktur penyimpanan yang menentukan jumlah node dan kedalaman *traversal* yang harus dilalui sistem. Semakin dalam struktur direktori, sistem harus melakukan lebih banyak operasi penelusuran (*traversal*) terhadap node-folder sebelum mencapai file target. Hal ini selaras dengan kompleksitas teoretis BFS dan DFS pada struktur pohon, yaitu O(V + E), di mana *V* merupakan jumlah direktori dan *E* adalah jumlah relasi antar-folder. Temuan empiris ini juga konsisten dengan penelitian terdahulu pada sistem berkas seperti NTFS dan ext4, yang menunjukkan bahwa struktur penyimpanan bertingkat cenderung meningkatkan latensi pencarian akibat jalur penelusuran yang lebih panjang.

SIMPULAN

Berdasarkan sepuluh kali pengujian terhadap 100 file acak, hasil eksperimen menunjukkan bahwa struktur direktori dan algoritma pencarian berpengaruh signifikan terhadap waktu pencarian berkas. Pada struktur *Single-Level Directory*, ketiga algoritma *Linear Search*, *BFS*, dan *DFS* menunjukkan performa yang jauh lebih cepat, dengan *Linear Search* sebagai acuan utama yang mencatat rata-rata 0.002162 detik per file (100% efisien). Sebaliknya, pada *Hierarchical Directory*, waktu pencarian meningkat seiring bertambahnya kedalaman direktori. BFS mengalami kenaikan dari 0.007890 detik (level 2) hingga 0.080534 detik (level 5), sedangkan DFS meningkat dari 0.011600 detik (level 2) hingga 0.049423 detik (level 4) sebelum menurun pada level 5 menjadi 0.022490 detik. Secara keseluruhan, BFS dan DFS tercatat masing-masing 20.26× dan 17.70× lebih lambat dibanding pencarian pada struktur tunggal. Peningkatan waktu pencarian ini menunjukkan bahwa penambahan kedalaman direktori memperluas ruang *traversal* yang harus dilalui algoritma, sehingga menambah latensi pencarian. Dengan demikian, efisiensi tidak hanya ditentukan oleh algoritma yang digunakan, tetapi juga oleh

morfologi struktur penyimpanan, khususnya jumlah node dan kedalaman hierarki. Struktur Single-Level Directory terbukti paling efisien karena tidak memerlukan penelusuran berlapis.

Temuan ini memberikan kontribusi empiris terhadap pemahaman hubungan antara kompleksitas struktur direktori dan performa algoritma pencarian file. Penelitian selanjutnya disarankan mengeksplorasi metode hibrid, seperti optimasi *directory pruning* atau *metadata caching*, untuk meningkatkan performa pencarian pada struktur *hierarkis* tanpa mengurangi akurasi maupun cakupan pencarian

DAFTAR PUSTAKA

[1] M. Cai, J. Shen, B. Tang, H. Huang, dan B. Ye, "FlatFS: Flatten Hierarchical File System Namespace on Non-volatile Memories," dalam *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, Carlsbad, CA: USENIX Association, Jul 2022, hlm. 899–914.

[2] M. Cai, J. Shen, B. Tang, H. Huang, dan B. Ye, "Exploiting Flat Namespace to Improve File System Metadata Performance on Ultra-Fast, Byte-Addressable NVMs," *ACM Trans Storage*, vol. 20, no. 1, Jan 2024, doi: 10.1145/3620673.

[3] A. Dewald dan S. Seufert, "What is New in ext4 for Incident Analysis and Digital Forensics," ERNW Enno Rey Netzwerke GmbH, Whitepaper, 2017. [Daring]. Tersedia pada: https://www.ernw.de/download/ext4_incident_analysis_digital_forensics.pdf

[4] K. Ren dan G. Gibson, "TABLEFS: Enhancing Metadata Efficiency in the Local File System," dalam *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, San Jose, CA: USENIX Association, 2013, hlm. 145–156. [Daring]. Tersedia pada: https://www.usenix.org/conference/atc13/technical-sessions/presentation/ren

[5] K. Ren, Q. Zheng, S. Patil, dan G. Gibson, "IndexFS: Scaling File System Metadata Performance with Table-Based Namespace," dalam *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, ACM, 2014, hlm. 237–248. doi: 10.1109/SC.2014.25.

[6] W. Lv, Y. Lu, Y. Zhang, P. Duan, dan J. Shu, "InfiniFS: An Efficient Metadata Service for Large-Scale Distributed Filesystems," dalam *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST '22)*, 2022. doi: 10.5555/3517268.3523105.

[7] Y. Zhang, G. Li, K. Lu, dan J. Wan, "PhatKV: Towards an Efficient Metadata Engine for KV-based File Systems on Modern SSD," dalam *Proceedings of the 21st International Conference on Mass Storage Systems and Technologies (MSST)*, 2024. [Daring]. Tersedia pada: https://www.msstconference.org/MSST-history/2024/Papers/msst24-9.3.pdf

[8] Y. Hu dan C. Wang, "Accelerating LSM-Tree with the Dentry Management of File System," 28 September 2021, *arXiv*:2109.13142. doi: 10.48550/arXiv.2109.13142.

[9] J.-Y. Lee, M.-H. Kim, S. A. Raza Shah, S.-U. Ahn, H. Yoon, dan S.-Y. Noh, "Performance Evaluations of Distributed File Systems for Scientific Big Data in FUSE Environment," *Electronics*, vol. 10, no. 12, hlm. 1471, Jun 2021, doi: 10.3390/electronics10121471.

[10] L. Jiang, Z. Zhang, R. Ni, dan M. Cai, "DirectFS: An RDMA-Accelerated Distributed File System with CPU-Oblivious Metadata Indexing," *Electronics*, vol. 14, no. 19, hlm. 3778, Sep 2025, doi: 10.3390/electronics14193778.

[11] T. J. Khoo *dkk.*, "Constraints on Future Analysis Metadata Systems in High Energy Physics," *Comput. Softw. Big Sci.*, vol. 6, no. 1, hlm. 13, Des 2022, doi: 10.1007/s41781-022-00086-2.

[12] H. Dai, Y. Wang, K. B. Kent, L. Zeng, dan C. Xu, "The State of the Art of Metadata Managements in Large-Scale Distributed File Systems — Scalability, Performance and Availability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, hlm. 3850–3869, 2022, doi: 10.1109/TPDS.2022.3170574.

[13] K. Sitompul dan A. P. Harinja, "Rancangan Aplikasi Pencarian File pada Direktori Windows Dengan Menggunakan Metode Breadth First Search," *KAKIFIKOM J Ilmu Komput. Dan Inform.*, vol. 03, no. 01, hlm. 28–39, 2021, doi: 10.54367/kakifikom.v3i1.1198.

[14] M. S. Rumetna, T. N. Lina, A. B. Santoso, R. Komansilan, dan J. B. Karay, "Implementasi Algoritma Depth First Search dalam Penyelesaian Permasalahan Lintasan dan Sirkuit Euler," *J. Komtika Komputasi Dan Inform.*, vol. 7, no. 1, hlm. 12–21, 2023, doi: 10.31603/komtika.v7i1.8672.

[15] J. Li, "Efficient parallelism in Breadth-First Search: A comprehensive analysis and implementation," *Appl. Comput. Eng.*, vol. 36, no. 1, hlm. 185–191, Feb 2024, doi: 10.54254/2755-2721/36/20230443.

[16] N. Purnama, "Comparative Performance Study of Search Algorithms on Large-Scale Data Structures," *J Ilm Pengetah. Dan Teknol. Komput. JITK*, vol. 11, no. 1, 2022, doi: 10.33480/jitk.v11i1.6592.

[17] E. C. Da Silva, L. M. Sato, dan E. T. Midorikawa, "Distributed File System to Leverage Data Locality for Large-File Processing," *Electronics*, vol. 13, no. 1, hlm. 106, Des 2023, doi: 10.3390/electronics13010106.

[18] S. Shaikh, "Billion-files File Systems (BfFS): A Comparison," *ArXiv Prepr. ArXiv240801805*, Agu 2024, doi: 10.48550/arXiv.2408.01805.