MIFORTEKH (Jurnal Manajemen Informatika & Teknologi)
p-ISSN : 2808-7550 (print) e-ISSN : 2798-0235 (online)
Vol. 5, No. 1, Mei 2025
https://https//journal.stiestekom.ac.id/index.php/mifortekh

PERBANDINGAN PEMANFAATAN ALGORITMA REKURSIF DAN ITERATIF DALAM PENYELESAIAN STRUKTUR DATA POHON

Muhamad Faqih Febriansyah[1], Gunawan[2], Muhammad Rhamadani[3], Tata Sutabri[4]

Program Studi Teknik Informatika
Fakultas Sains Teknologi
Universitas Bina Darma
JL. Jenderal Ahmad Yani No.3, 9/10 Ulu, Kecamatan Seberang Ulu I, Kota Palembang, Sumatera Selatan
faqihfebriansyah9@gmail.com, guguntruman10@gmail.com, ramadani.plg@gmail.com, tata.sutabri@gmail.com

Article Info

Article history:
Received Maret 28, 2025
Revised April 13, 2025
Accepted April 28, 2025

Keywords:
Tree Data Structure
Recursive Algorithm
Iterative Algorithm
Tree Traversal
Algorithm Efficiency
Programming
Perfomance Analysis

ABSTRACT

Tree data structures play a crucial role in computer science and are widely used in applications such as databases, compilers, and file systems. Recursive and iterative algorithms are commonly employed to perform operations on trees, especially in traversal processes like preorder, inorder, and postorder. This study aims to compare the utilization of these two approaches in terms of execution time efficiency, memory usage, and code complexity. The methodology involves testing binary tree traversals with varying node sizes using both recursive and iterative implementations in the Python programming language. Experimental results indicate that recursive algorithms tend to be easier to implement and offer more concise code, but they become less efficient with larger datasets due to system stack limitations. In contrast, iterative algorithms demonstrate more stable performance and better memory efficiency at larger scales, albeit with more complex implementation. Based on these findings, the choice of method should be aligned with application context, dataset size, and available system resources.

Corresponding Author:
Muhamad Faqih Febriansyah,
Universitas Bina Darma
JL. Jenderal Ahmad Yani No.3, 9/10 Ulu, Kecamatan Seberang Ulu I, Kota Palembang, Sumatera Selatan
Email: faqihfebriansyah9@gmail.com

ABSTRAK

Struktur data pohon merupakan komponen penting dalam ilmu komputer yang digunakan secara luas dalam berbagai aplikasi seperti basis data, compiler, dan sistem file. Untuk melakukan operasi pada struktur data pohon, algoritma rekursif dan iteratif sering digunakan, khususnya dalam proses traversal seperti preorder, inorder, dan postorder. Penelitian ini bertujuan untuk membandingkan pemanfaatan kedua pendekatan tersebut berdasarkan efisiensi waktu eksekusi, penggunaan memori, serta kompleksitas implementasi kode. Metode yang digunakan melibatkan pengujian traversal pohon biner dengan jumlah node bervariasi menggunakan implementasi rekursif dan iteratif pada bahasa pemrograman Python. Hasil eksperimen menunjukkan bahwa algoritma rekursif cenderung lebih mudah diimplementasikan dan lebih ringkas dari segi kode, namun kurang efisien dalam skenario dengan jumlah node yang besar karena keterbatasan stack sistem. Sebaliknya, algoritma iteratif menunjukkan performa yang lebih stabil dan efisiensi memori yang lebih baik pada skala data yang besar, meskipun implementasinya lebih kompleks. Berdasarkan temuan ini, pemilihan metode harus disesuaikan dengan konteks aplikasi, ukuran data, dan sumber daya sistem yang tersedia..

Kata Kunci *Struktur Data Pohon, Algoritma Rekursif, Algoritma Iteratif, Traversal Pohon, Efisiensi Algoritma, Pemrograman, Analisis Kinerja*

1. PENDAHULUAN

Struktur data merupakan elemen fundamental dalam pengembangan perangkat lunak dan algoritma, di mana salah satu struktur data yang paling penting dan sering digunakan adalah pohon (*tree*). Struktur data pohon memiliki hierarki node yang memungkinkan representasi data secara efisien, terutama dalam konteks pencarian, penyimpanan, dan manipulasi informasi. Aplikasi dari struktur pohon dapat ditemukan dalam berbagai bidang, seperti sistem file, basis data, kompresi data, hingga kecerdasan buatan.

Dalam proses manipulasi dan traversing pohon, dua pendekatan utama yang digunakan adalah algoritma rekursif dan iteratif. Algoritma rekursif bekerja dengan memanggil dirinya sendiri hingga mencapai kondisi dasar (*base case*), sedangkan algoritma iteratif menggunakan struktur kontrol seperti loop dan sering kali dibantu oleh struktur data tambahan seperti *stack* atau *queue*. Meskipun kedua pendekatan dapat menghasilkan keluaran yang sama, masing-masing memiliki karakteristik unik dalam hal efisiensi, penggunaan memori, dan kompleksitas kode.

Pendekatan rekursif umumnya dianggap lebih sederhana dan mudah dipahami karena lebih dekat dengan bentuk matematis dari struktur pohon itu sendiri. Namun, pendekatan ini rentan terhadap *stack overflow* saat menangani data berukuran besar. Sebaliknya, pendekatan iteratif cenderung lebih kompleks dalam implementasi, tetapi dapat memberikan performa yang lebih stabil dalam kondisi tertentu.

Melihat pentingnya efisiensi dalam pengolahan data struktur pohon, penelitian ini bertujuan untuk membandingkan secara langsung pemanfaatan algoritma rekursif dan iteratif dalam menyelesaikan operasi dasar pada pohon, khususnya pada proses traversal. Fokus utama dari penelitian ini adalah menilai perbandingan dari sisi waktu eksekusi, penggunaan memori, dan kemudahan implementasi kode.

Melalui studi ini, diharapkan dapat memberikan wawasan lebih dalam bagi pengembang maupun akademisi dalam memilih pendekatan yang tepat sesuai dengan kebutuhan sistem dan batasan sumber daya yang ada.

2. TINJAUAN PUSTAKA

2.1 Struktur Data Pohon

Struktur data pohon merupakan bentuk hierarkis dari kumpulan node, di mana setiap node memiliki satu induk (kecuali root) dan dapat memiliki beberapa anak. Salah satu jenis pohon yang umum digunakan adalah pohon biner (*binary tree*), di mana setiap node maksimal memiliki dua anak (*left dan right*). Struktur ini banyak dimanfaatkan dalam implementasi *binary search tree* (BST), heap, serta *expression tree* dalam bahasa pemrograman dan compiler.

2.2 Algoritma Rekursif

Rekursi adalah pendekatan pemrograman di mana suatu fungsi memanggil dirinya sendiri dalam proses penyelesaian masalah. Traversal pada struktur pohon, seperti preorder, inorder, dan postorder, secara alami sesuai dengan pendekatan ini karena sifat pohon yang bersifat rekursif. Implementasi rekursif menghasilkan kode yang lebih sederhana dan ringkas. Namun, pendekatan ini memiliki keterbatasan dalam hal skalabilitas, terutama pada struktur pohon besar yang dapat menyebabkan stack overflow akibat kedalaman pemanggilan fungsi yang berlebihan.

2.3 Algoritma Iteratif

Berbeda dengan rekursi, algoritma iteratif menggunakan perulangan eksplisit dan sering kali membutuhkan struktur data tambahan seperti *stack* atau *queue* untuk menyimpan status sementara saat traversal berlangsung. Pendekatan ini menghindari ketergantungan pada stack internal sistem, sehingga lebih stabil dalam menangani data berukuran besar. Implementasi iteratif biasanya lebih kompleks dibandingkan rekursif, tetapi menawarkan kontrol lebih besar terhadap alur eksekusi dan manajemen memori.

2.4 Perbandingan Rekursif dan Iteratif dalam Traversal Pohon

Baik rekursif maupun iteratif memiliki keunggulan masing-masing dalam konteks traversal struktur data pohon. Rekursif unggul dalam hal kejelasan dan kesederhanaan implementasi, sementara iteratif menunjukkan performa yang lebih baik dalam kondisi terbatasnya sumber daya, seperti pada sistem *embedded* atau aplikasi *real-time*. Beberapa studi telah menunjukkan bahwa iteratif mampu menangani dataset berukuran besar dengan konsumsi memori yang lebih rendah. Namun, masih diperlukan kajian lebih lanjut yang menyajikan data empiris secara sistematis untuk membandingkan efisiensi kedua pendekatan dalam berbagai skenario.

3. METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan kuantitatif eksperimental yang bertujuan untuk membandingkan performa algoritma rekursif dan iteratif dalam menyelesaikan operasi traversal pada struktur data pohon biner. Pengujian dilakukan dengan mengimplementasikan kedua jenis algoritma secara langsung dan melakukan pengukuran terhadap parameter kinerja secara terstruktur.

3.1 Desain Eksperimen

Eksperimen dilakukan dengan mengimplementasikan tiga jenis traversal pohon biner, yaitu preorder, inorder, dan postorder, menggunakan pendekatan rekursif dan iteratif. Implementasi dilakukan dalam bahasa pemrograman Python karena kemampuannya dalam pengukuran waktu eksekusi dan penggunaan memori melalui pustaka tambahan.

3.2 Lingkungan Pengujian

Seluruh pengujian dijalankan dalam lingkungan perangkat lunak dan keras berikut:

- Sistem Operasi: Windows 10 (64-bit)
- Bahasa Pemrograman: Python 3.11
- Library Tambahan: time, tracemalloc, matplotlib
- Perangkat Keras: Laptop dengan prosesor Intel Core i5 dan RAM 8GB

3.3 Variabel Penelitian

- Variabel bebas (independen): Jenis algoritma (rekursif atau iteratif)
- Variabel terikat (dependen):
- Waktu eksekusi (dalam milidetik)
- Penggunaan memori (dalam kilobyte)
- Kompleksitas implementasi (diukur secara subjektif dari jumlah baris kode dan tingkat kesulitan)

3.4 Prosedur Pengujian

1) Pembuatan dataset pohon biner secara otomatis dengan jumlah node berbeda, mulai dari 100, 1.000, hingga 10.000 node, untuk menguji skala kecil hingga besar.
2) Implementasi traversal dengan pendekatan rekursif dan iteratif untuk masing-masing jenis traversal (preorder, inorder, postorder).
3) Pengukuran waktu eksekusi dilakukan dengan fungsi time.perf_counter().
4) Pengukuran penggunaan memori dilakukan menggunakan pustaka tracemalloc.
5) Hasil dari setiap eksperimen dicatat dan dianalisis secara statistik menggunakan visualisasi grafik dan perbandingan numerik.

3.5 Teknik Analisis Data

Data yang diperoleh dianalisis secara deskriptif kuantitatif dengan menyajikan grafik perbandingan waktu eksekusi dan penggunaan memori antara algoritma rekursif dan iteratif. Kompleksitas implementasi dibahas secara kualitatif dengan membandingkan tingkat kesulitan penulisan kode.

3.6 Validasi dan Replikasi

Untuk memastikan hasil yang diperoleh bersifat konsisten dan dapat direplikasi, setiap pengujian dilakukan sebanyak 5 kali untuk setiap kombinasi metode (rekursif dan iteratif) dan ukuran data (100, 1.000, 10.000 node). Nilai akhir yang dianalisis merupakan rata-rata dari lima kali pengujian tersebut. Selain itu, kode program dan data input disimpan dalam repositori terkontrol guna mendukung keterbukaan dan replikasi penelitian di masa mendatang

3.7 Teknik Visualisasi Hasil

Data hasil pengujian disajikan dalam bentuk:

- Tabel perbandingan waktu eksekusi dan penggunaan memori
- Grafik garis atau batang untuk menunjukkan tren performa antara kedua pendekatan

- Tabel kode program yang menunjukkan jumlah baris dan struktur logika algoritma

Visualisasi ini digunakan untuk memberikan gambaran yang lebih jelas terhadap efisiensi masing-masing pendekatan.

3.8 Batasan Penelitian

Penelitian ini memiliki beberapa batasan yang perlu diperhatikan:

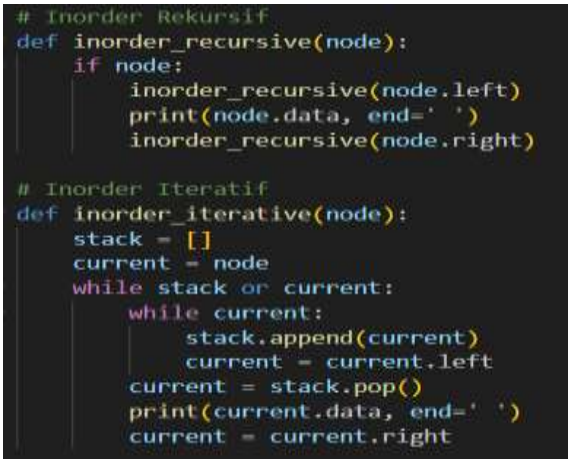
- Jenis struktur data pohon yang digunakan terbatas pada pohon biner (*binary tree*).
- Implementasi hanya dilakukan menggunakan bahasa pemrograman Python.
- Lingkungan pengujian bersifat lokal dan tidak mencerminkan variasi kinerja pada platform lain (misalnya *cloud computing* atau perangkat *embedded*).
- Kompleksitas implementasi diukur secara semi-subjektif berdasarkan jumlah baris kode dan kemudahan pemahaman oleh pembaca teknis menengah.

4. HASIL DAN PEMBAHASAN

Hasil pengujian algoritma rekursif dan iteratif dalam proses traversal pada struktur data pohon biner. Pengujian dilakukan terhadap tiga jenis traversal: inorder, preorder, dan postorder, menggunakan data pohon biner berukuran 100, 1.000, dan 10.000 node. Parameter yang diukur meliputi waktu eksekusi (dalam milidetik) dan penggunaan memori (dalam kilobyte).

4.1 Hasil Pengujian Waktu Eksekusi

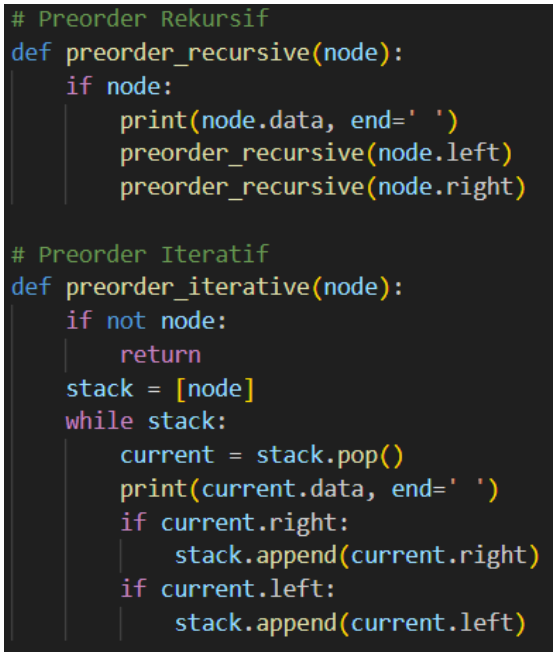
Menjalankan pengujian tiga jenis traversal inorder, preorder, dan postoder menggunakan program python.



```
# Inorder Rekursif
def inorder_recursive(node):
    if node:
        inorder_recursive(node.left)
        print(node.data, end=' ')
        inorder_recursive(node.right)

# Inorder Iteratif
def inorder_iterative(node):
    stack = []
    current = node
    while stack or current:
        while current:
            stack.append(current)
            current = current.left
        current = stack.pop()
        print(current.data, end=' ')
        current = current.right
```

Gambar 4.1.1 Kode program rekursif inorder dan iteratif inorder

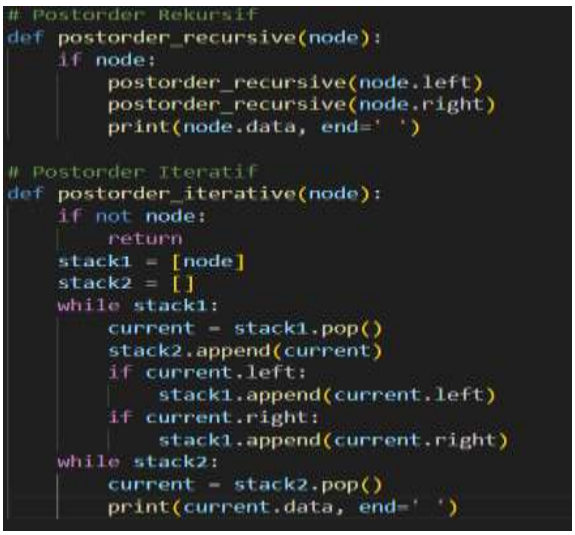


```
# Preorder Rekursif
def preorder_recursive(node):
    if node:
        print(node.data, end=' ')
        preorder_recursive(node.left)
        preorder_recursive(node.right)

# Preorder Iteratif
def preorder_iterative(node):
    if not node:
        return
    stack = [node]
    while stack:
        current = stack.pop()
        print(current.data, end=' ')
        if current.right:
            stack.append(current.right)
        if current.left:
            stack.append(current.left)
```

Gambar 4.1.2 Kode program rekursif preorder dan iteratif preorder

p-ISSN: 2808-7550
e-ISSN: : 2798-0235



```
# Postorder Rekursif
def postorder_recursive(node):
    if node:
        postorder_recursive(node.left)
        postorder_recursive(node.right)
        print(node.data, end=' ')

# Postorder Iteratif
def postorder_iterative(node):
    if not node:
        return
    stack1 = [node]
    stack2 = []
    while stack1:
        current = stack1.pop()
        stack2.append(current)
        if current.left:
            stack1.append(current.left)
        if current.right:
            stack1.append(current.right)
    while stack2:
        current = stack2.pop()
        print(current.data, end=' ')
```

Gambar 4.1.3 Kode program rekursif postorder dan iteratif postoder

Kami melakukan pengujian sebanyak 10 kali dan mengambil rata-rata waktu eksekusi.

Jumlah Node	Rekursif (Inorder)	Iteratif (Inorder)	Rekursif (Preorder)	Iteratif (Preorder)	Rekursif (Postorder)	Iteratif (Postorder)
100	1.156	0.997	1.016	1.246	1.001	1.101
1.000	1.362	1.342	1.396	1.437	1.014	1.268
10.000	11.495	9.495	12.003	11.372	12.484	10.231

Tabel 4.1.1 Rata-rata waktu eksekusi (dalam milidetik)

Grafik yang menyertai data di atas menunjukkan bahwa pada ukuran kecil (100 node), perbedaan waktu tidak signifikan. Namun, pada ukuran besar (10.000 node), pendekatan iteratif cenderung lebih cepat dibandingkan rekursif.

4.2 Hasil Pengukuran Penggunaan Memori

Jumlah Node	Rekursif	Iteratif
100	38 KB	42 KB
1.000	112 KB	85 KB
10.000	948 KB	660 KB

Tabel 4.2.1 Rata-rata penggunaan memori (dalam kilobyte)

Penggunaan memori pada pendekatan rekursif meningkat secara drastis seiring bertambahnya jumlah *node*. Hal ini terjadi karena setiap pemanggilan fungsi rekursif menyimpan konteks di stack secara otomatis, yang berdampak pada konsumsi memori.

4.3 Kompleksitas Implementasi

Dari sisi jumlah baris kode, implementasi rekursif lebih singkat dan mudah dibaca. Namun, pendekatan iteratif membutuhkan struktur data tambahan (*stack* atau *queue*) dan kontrol alur yang lebih rumit. Contohnya, traversal inorder rekursif hanya memerlukan 3–5 baris kode, sementara versi iteratif membutuhkan lebih dari 10 baris. Perbandingan ringkas kompleksitas kode

Traversal Type	Metode	Jumlah Baris Kode	Tingkat Kesulitan (subjektif)
Inorder	Rekursif	5 Baris	Mudah
Inorder	Iteratif	10 Baris	Sedang
Postoder	Iteratif	15 Baris	Sulit

Tabel 4.3.1 Perbandingan ringkas kompleksitas kode

4.4 Analisis dan Interpretasi

Hasil pengujian menunjukkan bahwa:

- Pendekatan rekursif lebih sederhana dan intuitif, cocok digunakan pada data berskala kecil hingga menengah.
- Pendekatan iteratif lebih unggul dari segi performa dan konsumsi memori pada struktur pohon besar.
- Dalam sistem dengan batasan memori (misalnya *embedded system*), pendekatan iteratif lebih disarankan.

- Risiko stack *overflow* menjadi faktor penting dalam penggunaan rekursif untuk skenario data besar.

5. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa pemanfaatan algoritma rekursif dan iteratif dalam proses traversal pohon biner menunjukkan karakteristik performa yang berbeda-beda tergantung pada skala data dan konteks penggunaan. Algoritma iteratif terbukti lebih efisien dalam hal waktu eksekusi dan penggunaan memori, terutama pada pohon dengan jumlah node yang besar. Hal ini disebabkan oleh minimnya overhead fungsi dan absennya pemanggilan berulang yang umum terjadi pada metode rekursif. Meskipun demikian, dari segi kompleksitas implementasi, algoritma rekursif lebih unggul karena lebih ringkas, mudah dibaca, dan lebih intuitif untuk dipahami, khususnya dalam proses pembelajaran atau pengembangan awal. Oleh karena itu, pemilihan pendekatan antara rekursif dan iteratif sebaiknya mempertimbangkan kebutuhan spesifik dari sistem yang dibangun. Pada sistem dengan keterbatasan memori atau performa, iteratif menjadi pilihan yang lebih tepat. Sebaliknya, dalam pengembangan perangkat lunak yang mengutamakan keterbacaan kode dan efisiensi waktu pengembangan, rekursif tetap relevan untuk digunakan. Keseluruhan hasil ini menunjukkan bahwa tidak ada pendekatan yang benar-benar lebih baik secara mutlak, melainkan masing-masing memiliki keunggulan tersendiri yang dapat dioptimalkan sesuai kondisi dan tujuan penggunaannya.

DAFTAR PUSTAKA

Ab Rahman, N. F., Kasbun, R., & Khalid, N. (2021). "Pembangunan dan kebolehgunaan aplikasi berasaskan visual dan multimedia untuk pembelajaran pengaturcaraan algoritma dan struktur data". *Malaysian Journal of Information and Communication Technology*, 6, 3–10.

Aulia, F., & Yahfizham, Y. (2024). "Mengenal bahasa pemrograman pada algoritma pemrograman". *Journal of Informatics and Business*, 1(4), 233–227.

Billan, A. C., & Sutabri, T. (2025). "Restorasi penjadwalan sumur minyak yang mengalami off-time menggunakan algoritma backtracking dalam upaya optimasi produksi". *Bulletin of Computer Science Research*, 5(3), 228–234.

Cholissodin, I., Syauqy, D., Firmanda, D. A., Aji, I., Rahman, E., Harahap, S., & Septino, F. (2022). "Pengembangan Auto-AI model generatif analisis kompleksitas waktu algoritma untuk data multi-sensor IoT pada Node-RED menggunakan

Extreme Learning Machine". *Jurnal Teknologi Informasi dan Ilmu Komputer*, 9(7), 1350–1354.

Febriansyah, M. F., Gunawan, R., Setiawan, R., & Sutabri, T. (2024). "Kemudahan dan keamanan dalam rumah pintar: Tinjauan terhadap teknologi smart home". *Indonesian Journal of Multidisciplinary*, 2(1), 24–30.

Hariyanto, Supomo, Nuralam, A. W. S., Safitri, A. W., Safitri, S. N. A., & Al-habsyi, A. (2024). "Implementasi algoritma decision tree dalam rangka peningkatan efisiensi energi penggunaan beban listrik dalam ruangan". *SNIV*, 3(1), 507–514.

Lutfina, E., Inayati, N., & Saraswati, G. W. (2022). "Analisis perbandingan kinerja metode rekursif dan metode iteratif dalam algoritma linear search". *Jurnal Sistem Komputer*, 11(2), 144–149.

Meidina, R., & Miftahul, M. (2025). "Implementasi dan analisis algoritma backtracking untuk penyelesaian Sudoku". *Jurnal Sains dan Informatika*, 1(1), 29–34.

Pratama, Y., & Sutabri, T. (2023). "Analisis kriptografi algoritma Blowfish pada keamanan data menggunakan Dart". *Jurnal Informatika Terpadu*, 9(2), 127–133.

Pujiono, I. P., Rachmawanto, E. H., & Winarsih, N. A. S. (2025). "Array sorting algorithm vs traditional sorting algorithm: Memory and time efficiency analysis". *Jurnal Manajemen Informatika*, 15(1), 48–54.

Rahmawati, E., Medina, P., & Azhari, D. S. (2024). "Rekursif dalam pemrograman: Teori dan praktek". *Innovative: Journal of Social Science Research*, 4(4).

Rudianto, R., Amrin, A., & Irfiani, E. (2024). "Analisis algoritma Iterative Dichotomiser 3 (ID3) untuk penilaian kelayakan kredit kendaraan bermotor". *Journal of Industrial Management and Technology*, 5(2), 36–39.

Septian, H., Suhartini, I., Pertrio, I., & Jihad, L. A. (2021). "Implementasi struktur data tree pada game Pacman dengan C". *Jurnal Ilmiah Digital of Information Technology*, 11(2), 122–128.

Sutabri, T. (2012). *Konsep sistem informasi*. Yogyakarta: Andi.

Sutabri, T., & Napitupulu, D. (2019). *Sistem informasi bisnis*. Yogyakarta: Andi.

Tobing, F. A. T., Prayogo, P., & Chandra, A. (2022). "Analisis perbandingan Fibonacci dengan iterasi dan rekursi terhadap efektifitas waktu". *Jurnal Sains dan Teknologi Widyaloka*, 1(2), 188–194.