Jurnal Riset Sistem Informasi dan Aplikasi Komputer (JRSIKOM)
Vol. 1, No. 4, Desember 2025
SK.ISSN: 3090076X/II.7.4/SK.ISSN/05/2025 e-ISSN: 3090-076X

Analisis Kinerja Linked List Dalam Pengolahan Data Dinamis: Studi Literatur

Nazriel Abdillah[1], Herdiansyah Ramzani[2], Sutresno[3], Diki Ananda[4], Muhammad Azyansah Putra Hadi[5], Indra Gunawan M.Kom[6]

[1]nazrielabdillah2@gmail.com
[2]ramzaniherdiansyah@gmail.com
[3]sutrisno02052006@gmail.com
[4]dddiki3937@gmail.com
[5]putrahadi4533@gmail.com
[6]indra@amiktunasbangsa.ac.id

Jurusan Teknik Informatika,STIKOM Tunas Bangsa Pematangsiantar,Medan,Sumatera Utara

ABSTRAK

Linked list merupakan struktur data fundamental yang banyak digunakan dalam pengolahan data dinamis karena fleksibilitasnya dalam alokasi memori dibandingkan dengan struktur statis seperti array. Penelitian ini bertujuan untuk menganalisis kinerja linked list dalam aspek efisiensi penyisipan dan penghapusan, manajemen memori, serta perbandingannya dengan struktur data lain seperti array, binary search tree (BST), dan hash table. Metode yang digunakan adalah studi literatur dengan menelaah sumber akademik, termasuk buku teks dan jurnal ilmiah, untuk mengevaluasi keunggulan dan keterbatasan linked list. Hasil penelitian menunjukkan bahwa linked list unggul dalam operasi penyisipan dan penghapusan dengan kompleksitas waktu O(1) jika posisi target sudah diketahui. Namun, linked list memiliki kelemahan berupa overhead memori tambahan akibat penyimpanan pointer serta waktu akses yang lebih lambat dibandingkan array. Dalam aplikasi dunia nyata, linked list banyak digunakan dalam implementasi antrian, tumpukan, sistem basis data, dan optimasi memori cache. Studi ini menyimpulkan bahwa meskipun linked list memiliki keunggulan dalam manipulasi data dinamis, performanya harus dievaluasi berdasarkan kebutuhan spesifik suatu sistem. Struktur alternatif seperti BST atau hash table mungkin lebih sesuai untuk skenario yang memerlukan pencarian data cepat.

Kata kunci: Linked List, Struktur Data, Manajemen Memori, Efisiensi Penyisipan, Kompleksitas Algoritma

PERFORMANCE ANALYSIS OF LINKED LIST IN DYNAMIC DATA PROCESSING: LITERATURE STUDY

ABSTRACT

Linked list is a fundamental data structure widely used in dynamic data processing due to its flexibility in memory allocation compared to static structures like arrays. This study aims to analyze the performance of linked lists in terms of insertion and deletion efficiency, memory management, and comparison with other data structures such as arrays, binary search trees (BST), and hash tables. A literature review method was employed by examining academic sources, including textbooks and scientific journals, to evaluate the strengths and limitations of linked lists. The findings indicate that linked lists excel in insertion and deletion operations, achieving O(1) time complexity when the target position is known. However, they suffer from additional memory overhead due to pointer storage and slower access time compared to arrays. In real-world applications, linked lists are widely used in queue and stack implementations, database management systems, and cache memory optimization. This study concludes that while linked lists offer advantages in dynamic data manipulation, their performance should be assessed based on specific application needs. Alternative structures like BSTs or hash tables may be more suitable for scenarios requiring fast data retrieval.

Keyword: *Linked List, Data Structure, Memory Management, Insertion Efficiency, Algorithm Complexity.*

I. PENDAHULUAN

Struktur data merupakan elemen fundamental dalam ilmu komputer yang mempengaruhi efisiensi algoritma dan manajemen sumber daya sistem. Salah satu struktur data yang sering digunakan dalam pengolahan data dinamis adalah linked list. Linked list memiliki keunggulan dalam fleksibilitas penyimpanan dibandingkan dengan struktur data lain seperti array, yang memiliki ukuran tetap dan dapat menyebabkan fragmentasi memori (Goodrich et al., 2014). Dengan kemampuannya untuk mengalokasikan memori secara dinamis, linked list menjadi pilihan utama dalam berbagai aplikasi komputasi modern yang memerlukan efisiensi dalam penyisipan dan penghapusan data.

Pengelolaan memori merupakan salah satu tantangan utama dalam implementasi struktur data. Linked list menawarkan solusi dengan menghindari masalah alokasi memori statis yang umum terjadi pada array. Dalam implementasinya, setiap elemen dalam linked list, yang disebut node, memiliki referensi ke node berikutnya, memungkinkan efisiensi dalam operasi penyisipan dan penghapusan dibandingkan dengan array yang membutuhka n pergeseran elemen (Cormen et al., 2009). Namun, linked list juga memiliki keterbatasan, seperti penggunaan memori tambahan untuk penyimpanan pointer dan waktu akses yang lebih lambat dibandingkan array karena tidak mendukung akses langsung ke elemen tertentu.

Dalam konteks efisiensi penyisipan dan penghapusan data, linked list memiliki beberapa keunggulan dibandingkan dengan struktur data lain. Pada array, penyisipan atau penghapusan elemen memerlukan pergeseran elemen lainnya untuk menjaga keterurutan, yang dapat menyebabkan kompleksitas waktu O(n). Sebaliknya, linked list hanya perlu mengubah referensi pointer, yang memungkinkan penyisipan dan penghapusan dilakukan dalam kompleksitas waktu O(1) pada posisi tertentu (Weiss, 2012). Efisiensi ini menjadikan linked list lebih unggul dalam skenario yang membutuhkan manipulasi data secara dinamis.

Selain efisiensi dalam penyisipan dan penghapusan, linked list juga memiliki implikasi dalam manajemen memori yang lebih baik. Dengan alokasi dinamis, linked list dapat menghindari pemborosan memori akibat alokasi statis yang berlebih. Namun, penggunaan pointer dalam setiap node mengakibatkan overhead memori yang lebih besar dibandingkan array (Knuth, 1998). Oleh karena itu, penting untuk mengevaluasi penggunaan linked list dalam berbagai skenario guna memahami kompromi antara fleksibilitas dan efisiensi memori.

Dalam aplikasi dunia nyata, linked list sering digunakan dalam implementasi antrian (queue), tumpukan (stack), serta sistem penyimpanan yang memerlukan modifikasi data secara dinamis, seperti dalam sistem basis data dan pengelolaan buffer jaringan (Tanenbaum & Austin, 2016). Efisiensi linked list dalam operasi tersebut menunjukkan relevansinya dalam pengolahan data modern, khususnya dalam sistem dengan kebutuhan manipulasi data yang tinggi.

Namun, performa linked list dalam pengolahan data dinamis tetap harus dibandingkan dengan struktur data lain, seperti array, tree, atau hash table, untuk menentukan penggunaannya dalam skenario tertentu. Faktor seperti kecepatan akses, konsumsi memori, dan kemudahan implementasi menjadi pertimbang an penting dalam pemilihan struktur data yang optimal (Sedgewick & Wayne, 2011).

Penelitian ini bertujuan untuk menganalisis kinerja linked list dalam pengola han data dinams dengan metode studi literatur. Studi literatur dilakukan dengan menelaah berbagai sumber akademik yang relevan, seperti buku teks, jurnal ilmiah, dan publikasi terkini, guna memperoleh pemaham an mendalam mengenai efisiensi linked list dalam aspek manajemen memori serta proses penyisipan dan penghapusan data. Studi ini juga akan membandingkan linked list dengan struktur data lain guna memperoleh pemahaman yang lebih komprehensif mengenai keunggulan dan keterbatasannya.

Dengan memahami kinerja linked list dalam berbagai konteks pengolahan data, diharapkan penelitian ini dapat memberikan kontribusi bagi pengembang perangkat lunak dan akademisi dalam memilih struktur data yang paling sesuai dengan kebutuhan spesifik mereka. Selain itu, penelitian ini dapat menjadi referensi bagi studi lanjutan mengenai optimasi penggunaan linked list dalam sistem komputasi yang semakin kompleks.

II. TINJAUAN PUSTAKA

Linked list sebagai struktur data dinamis telah menjadi fokus kajian dalam berbagai literatur ilmu komputer. Menurut Goodrich et al. (2014), linked list menawarkan fleksibilitas alokasi memori dinamis yang mengatasi keterbatasan struktur statis seperti array. Array memiliki ukuran tetap, sehingga sering menyebabkan *memory over-allocation* atau fragmentasi memori ketika kapasitas tidak sepenuhnya terisi. Sebaliknya,

linked list mengalokasikan memori secara *on-demand* melalui pembentukan node-node yang terhubung via pointer, menghindari pemborosan memori (Knuth, 1998). Konsep ini didukung oleh teori *dynamic memory management*, di mana setiap node dialokasikan di heap memory, memungkinkan operasi penyisipan dan penghapusan tanpa mengganggu struktur data secara keseluruhan (Cormen et al., 2009).

Efisiensi operasi penyisipan dan penghapusan pada linked list dibahas secara mendalam oleh Weiss (2012). Dalam teorinya, operasi tersebut memiliki kompleksitas waktu O(1) jika posisi target sudah diketahui, berbeda dengan array yang memerlukan pergeseran elemen (O(n)). Misalnya, penyisipan di tengah array memerlukan pergeseran *k* elemen, sementara linked list hanya mengubah referensi pointer dari node sebelumnya dan berikutnya. Namun, kelemahan utama linked list adalah ketidakmampuan melakukan *random access*, sehingga akses ke elemen ke-*n* memerlukan traversal linear (O(n)) (Sedgewick & Wayne, 2011). Hal ini menegaskan pentingnya pemilihan struktur data berdasarkan skenario penggunaan.

Dari perspektif manajemen memori, linked list menghasilkan *overhead* akibat penyimpanan pointer. Setiap node memerlukan ruang tambahan untuk menyimpan alamat node berikutnya (dan sebelumnya pada *doubly linked list*). Menurut Knuth (1998), overhead ini dapat menjadi signifikan jika data yang disimpan berukuran kecil, sehingga rasio metadata terhadap data aktual meningkat. Sebagai contoh, jika sebuah node menyimpan integer 4-byte tetapi memerlukan pointer 8-byte, penggunaan memori menjadi tidak efisien. Oleh karena itu, struktur seperti array tetap unggul dalam skenario penyimpanan data homogen berukuran kecil.

Dalam aplikasi sistem, linked list digunakan untuk mengimplementasikan antrian (*queue*) dan tumpukan (*stack*), di mana operasi dinamis seperti *enqueue*/*dequeue* atau *push*/*pop* sering dilakukan (Tanenbaum & Austin, 2016). Misalnya, antrian prioritas pada sistem operasi menggunakan linked list untuk mengelola proses yang memerlukan penyisipan cepat. Selain itu, linked list juga digunakan dalam *garbage collection* dan alokasi memori *block-based*, di mana manipulasi blok memori harus fleksibel (Cormen et al., 2009).

Perbandingan dengan struktur data lain seperti *hash table* atau *tree* menunjukkan bahwa linked list lebih cocok untuk operasi sekuensial, sementara struktur lain unggul dalam pencarian cepat. Sedgewick & Wayne (2011) menjelaskan bahwa *binary search tree* memiliki kompleksitas O(log n) untuk pencarian, tetapi memerlukan rebalancing yang rumit, sedangkan linked list tidak. Adapun *hash table* mampu mencapai O(1) untuk pencarian rata-rata, tetapi tidak efisien dalam penyisipan berurutan. Dengan demikian, linked list tetap relevan dalam konteks tertentu, seperti pengelolaan cache LRU (*Least Recently Used*) atau implementasi adjacency list dalam graf (Goodrich et al., 2014).

Studi empiris oleh Cormen et al. (2009) juga menunjukkan bahwa performa linked list sangat bergantung pada optimasi manajemen pointer dan lokalisasi memori. Pada sistem dengan *cache memory* terbatas, traversal linked list dapat menyebabkan *cache miss* yang tinggi karena node tersebar di heap memory. Hal ini berbeda dengan array yang memiliki *spatial locality*, sehingga lebih ramah cache. Temuan ini menegaskan bahwa efisiensi linked list tidak hanya ditentukan oleh kompleksitas algoritmik, tetapi juga faktor arsitektur komputer.

III. METODOLOGI

Penelitian ini menggunakan metode studi literatur dengan pendekatan sistematis untuk menganalisis kinerja linked list dalam pengolahan data dinamis. Tahap pertama melibatkan pengumpulan sumber akademis terkait, seperti buku teks (mis., Cormen et al., 2009; Goodrich et al., 2014), jurnal ilmiah, dan publikasi terbaru, yang dipilih berdasarkan relevansi dengan topik struktur data, manajemen memori, dan kompleksitas algoritma. Kriteria seleksi mencakup otoritas penulis, reputasi penerbit, dan tahun terbit (maksimal 10 tahun terakhir, kecuali karya fundamental seperti Knuth, 1998). Data dianalisis secara kualitatif melalui teknik *thematic analysis* untuk mengidentifikasi pola tematik seperti efisiensi operasi penyisipan/penghapusan, overhead memori, dan perbandingan kinerja dengan struktur data lain. Teori kompleksitas algoritma (O-notasi) dan prinsip manajemen memori dinamis menjadi kerangka analisis utama, sementara temuan empiris dari studi kasus (mis., implementasi antrian, *cache management*) dievaluasi untuk mengonfirmasi keunggulan dan keterbatasan linked list. Hasil sintesis literatur kemudian disusun secara komparatif guna menjawab tujuan penelitian dan memberikan rekomendasi kontekstual.

IV. HASIL DAN PEMBAHASAN

1. Efisiensi Penyisipan dan Penghapusan Data pada Linked List

Berdasarkan analisis literatur, salah satu keunggulan utama linked list adalah efisiensinya dalam operasi penyisipan dan penghapusan data. Seperti yang dijelaskan oleh Weiss (2012), linked list memungkinkan penyisipan dan penghapusan dengan kompleksitas waktu O(1) jika posisi target sudah diketahui. Hal ini dikarenakan setiap node hanya perlu mengubah referensi pointer tanpa harus menggeser elemen lain, seperti yang terjadi pada array yang memerlukan O(n) untuk operasi serupa.

Hasil studi menunjukkan bahwa pada skenario yang membutuhkan modifikasi data secara dinamis, seperti antrian proses dalam sistem operasi atau manipulasi daftar data dalam basis data, linked list memiliki keunggulan dibandingkan array. Misalnya, dalam implementasi antrian pada sistem operasi, setiap proses yang masuk dapat ditambahkan dengan cepat tanpa harus memindahkan elemen yang ada, sehingga meningkatkan efisiensi waktu eksekusi. Tanenbaum & Austin (2016) menekankan bahwa struktur ini sangat efektif dalam aplikasi yang memerlukan operasi penyisipan dan penghapusan yang sering.

Namun, hasil penelitian juga menunjukkan bahwa efisiensi ini bergantung pada cara pengelolaan pointer dalam implementasi linked list. Jika tidak dioptimalkan dengan baik, linked list dapat mengalami fragmentasi memori akibat alokasi dinamis yang tidak terstruktur, seperti yang dijelaskan oleh Knuth (1998). Oleh karena itu, dalam beberapa skenario, struktur lain seperti tree atau hash table mungkin lebih disarankan tergantung pada pola akses dan manipulasi data yang dibutuhkan.

2. Overhead Memori dan Efisiensi Manajemen Memori

Dalam analisis manajemen memori, linked list menunjukkan keunggulan dalam menghindari pemborosan akibat alokasi statis seperti yang terjadi pada array. Goodrich et al. (2014) menjelaskan bahwa linked list memungkinkan alokasi memori secara on-demand, yang berarti hanya memori yang dibutuhkan yang akan digunakan. Hal ini dapat mengurangi fragmentasi internal yang umum terjadi pada struktur data statis.

Namun, penelitian ini juga mengonfirmasi bahwa linked list memiliki overhead memori tambahan akibat penggunaan pointer. Seperti yang diungkapkan oleh Knuth (1998), setiap node dalam linked list memerlukan ruang tambahan untuk menyimpan alamat node berikutnya (atau sebelumnya dalam kasus doubly linked list). Jika data yang disimpan kecil, seperti integer atau karakter, overhead ini menjadi signifikan. Sebagai contoh, pada sistem dengan arsitektur 64-bit, setiap pointer bisa berukuran 8 byte, sehingga total memori yang digunakan meningkat dibandingkan dengan array yang hanya menyimpan data tanpa tambahan pointer.

Selain itu, dalam sistem dengan cache memory terbatas, linked list memiliki kelemahan dalam hal efisiensi cache. Seperti yang dijelaskan oleh Cormen et al. (2009), karena elemen dalam linked list tersebar di heap memory dan tidak dalam lokasi bersebelahan seperti array, traversal linked list sering menyebabkan cache miss, yang dapat memperlambat akses data secara keseluruhan. Hal ini membuktikan bahwa meskipun linked list unggul dalam fleksibilitas memori, performanya dalam pengaksesan data dapat lebih lambat dibandingkan dengan array yang memiliki keuntungan dalam spatial locality.

3. Perbandingan Kinerja Linked List dengan Struktur Data Lain

Dalam studi ini, linked list dibandingkan dengan struktur data lain seperti array, binary search tree (BST), dan hash table untuk memahami keunggulan dan keterbatasannya dalam skenario yang berbeda. Berdasarkan penelitian Sedgewick & Wayne (2011), hasil perbandingan menunjukkan bahwa:

- **Linked List vs. Array**
  Linked list unggul dalam penyisipan dan penghapusan data (O(1)), sementara array lebih unggul dalam akses data secara langsung (O(1)). Untuk skenario yang memerlukan banyak penyisipan/penghapusan, seperti sistem buffer jaringan, linked list lebih efektif. Namun, dalam kasus akses data acak, array lebih optimal.
- **Linked List vs. BST**
  BST menawarkan waktu pencarian yang lebih cepat (O(log n)) dibandingkan dengan linked list (O(n)). Namun, BST memerlukan mekanisme rebalancing untuk menjaga efisiensi pencarian. Linked list lebih cocok untuk skenario penyimpanan data sekuensial yang sering dimodifikasi.
- **Linked List vs. Hash Table**
  Hash table memiliki waktu pencarian rata-rata O(1), lebih cepat

dibandingkan linked list yang harus melakukan traversal linier. Namun, hash table tidak cocok untuk operasi penyisipan berurutan seperti yang terjadi dalam adjacency list pada graf, di mana linked list lebih unggul (Goodrich et al., 2014).

Hasil studi menunjukkan bahwa pilihan struktur data sangat bergantung pada kebutuhan spesifik aplikasi. Linked list sangat cocok untuk sistem dengan kebutuhan modifikasi data tinggi, sementara array, BST, dan hash table lebih optimal untuk skenario dengan akses data cepat.

4. Implementasi Linked List dalam Aplikasi Dunia Nyata

Dalam berbagai aplikasi komputasi, linked list digunakan untuk implementasi antrian (queue), tumpukan (stack), serta struktur data dinamis lainnya. Tanenbaum & Austin (2016) menjelaskan bahwa linked list sering digunakan dalam sistem basis data, di mana perubahan data secara dinamis sering terjadi, seperti pada sistem manajemen buffer atau tabel indeks dinamis.

Selain itu, linked list digunakan dalam implementasi cache LRU (Least Recently Used) untuk mengelola memori dalam sistem operasi. Dalam skenario ini, linked list memungkinkan penghapusan cepat dari elemen yang paling jarang digunakan dan menambah elemen baru dengan efisiensi tinggi. Penelitian oleh Cormen et al. (2009) menunjukkan bahwa efisiensi linked list dalam cache management sangat dipengaruhi oleh strategi alokasi memori dan pengelolaan pointer yang optimal.

Dalam konteks sistem jaringan, linked list juga digunakan dalam pengelolaan buffer jaringan, di mana data masuk dan keluar secara dinamis. Efisiensi linked list dalam menangani antrian data menjadikannya pilihan utama dalam sistem komunikasi berbasis paket.

V. KESIMPULAN

Dari hasil studi literatur ini, dapat disimpulkan bahwa linked list memiliki keunggulan utama dalam fleksibilitas alokasi memori dan efisiensi penyisipan serta penghapusan data dibandingkan array. Namun, kelemahan utamanya terletak pada overhead memori akibat penggunaan pointer serta waktu akses yang lebih lambat karena traversal linier. Dalam aplikasi dunia nyata, linked list sangat berguna dalam pengelolaan antrian, cache memory, serta sistem basis data yang membutuhkan perubahan data secara dinamis.

Meskipun linked list memiliki keunggulan dalam manipulasi data dinamis, performanya harus dievaluasi berdasarkan kebutuhan spesifik suatu sistem. Untuk operasi pencarian cepat, struktur seperti BST atau hash table lebih direkomendasikan. Dengan memahami kelebihan dan keterbatasan linked list, pengembang perangkat lunak dan akademisi dapat memilih struktur data yang paling sesuai untuk kebutuhan mereka.

V. DAFTAR PUSTAKA

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.

Knuth, D. E. (1998). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.

Tanenbaum, A. S., & Austin, T. (2016). Structured Computer Organization (6th ed.). Pearson.

Weiss, M. A. (2012). Data Structures and Algorithm Analysis in Java (3rd ed.). Pearson..