JRIIN: Jurnal Riset Informatika dan Inovasi
Volume 1, No. 12 Mei 2024
ISSN 3025-0919 (media online)
Hal 1329-1331

Penggunaan Algoritma Doubly Linked List Untuk Insertion Dan Deletion

Agung Wijoyo, Ahmad Farhan, Lalu Akbar Prayudi, Muhamad Fiqih, Rendi Dwi Santoso, Ricky Tri Setiawan Putra, Teguh Arifin

Program Studi S1 Teknik Informatika, Fakultas Ilmu Komputer, Universitas Pamulang, Kota Tangerang
Selatan, Banten, Indonesia
Email : teguharifin172202@gmail.com

Abstrak - doubly Linked List (DLL) merupakan struktur data linier yang terdiri dari node-node yang saling berhubungan, dimana setiap node mempunyai dua pointer yang menunjuk ke node sebelumnya dan node berikutnya. Berbeda dengan Single Linked List (SLL) yang hanya memiliki satu pointer yang menunjuk ke node berikutnya. Keunggulan DLL dibandingkan SLL terletak pada kemudahan dalam melakukan operasi penyisipan dan penghapusan data, karena DLL memungkinkan navigasi dua arah dalam daftar. penggunaan algoritma DLL untuk operasi insertion dan deletion. Algoritma insertion pada DLL dapat dilakukan di awal list (depan), di akhir list (belakang), atau di tengah list. Algoritma deletion pada DLL juga dapat dilakukan di awal list, di akhir list, atau di tengah list.

Kata kunci : doubly linked list, struktur data, penyisipan dan Penghapusan, linked list, previous pointer

Abstract - Double Linked List (DLL) is a linear data structure consisting of interconnected nodes connected, where each node has two pointers that point to the previous node and the next node. In contrast to Single Linked List (SLL) which only has one pointer that points to the next node. The advantage of DLL over SLL lies in the ease of performing data insertion and deletion operations, because DLL allows two-way navigation in the list. use of DLL algorithms for insertion and deletion operations. Inserting an algorithm in a DLL can be done at the beginning of the list (front), at the end of the list (back), or in the middle of the list. Deleting algorithms in DLLs can also be done at the beginning of the list, at the end of the list, or in the middle of the list.

Keyword : doubly linked list, Struktur Data, Insertion and Deletion, linked list, previous pointer

1. PENDAHULUAN

Doubly Linked List (DLL) merupakan struktur data yang terdiri dari node-node yang terhubung secara berpasangan, di mana setiap node memiliki pointer ke node sebelumnya (prev) dan node berikutnya (next). DLL menawarkan beberapa keunggulan dibandingkan dengan Single Linked List (SLL), terutama dalam hal operasi insertion dan deletion. Operasi insertion pada DLL dapat dilakukan lebih efisien di berbagai posisi, baik di awal, tengah, maupun akhir list. Hal ini karena DLL memungkinkan akses data dari dua arah, sehingga memungkinkan untuk menyisipkan node baru tanpa harus melalui seluruh list.

Operasi deletion pada DLL juga lebih fleksibel dibandingkan dengan SLL. DLL memungkinkan untuk menghapus node di berbagai posisi, termasuk node pertama, node terakhir, dan node di tengah list. Selain itu, DLL juga memungkinkan untuk menghapus node tanpa harus mengetahui nilai node tersebut, hanya dengan mengetahui pointer ke node tersebut.

2. TINJAUAN PUSTAKA

Doubly Linked List (DLL) merupakan struktur data dinamis yang terdiri dari node-node yang saling terhubung dua arah. Kemampuannya dalam operasi penyisipan dan penghapusan data yang fleksibel dan efisien menjadikannya pilihan populer untuk berbagai aplikasi :

2.1 Algoritma Dasar Penyisipan dan Penghapusan pada DLL

algoritma dasar penyisipan dan penghapusan pada DLL. Penyisipan dapat dilakukan di awal, tengah, atau akhir daftar. Kompleksitas waktu rata-rata untuk operasi ini adalah O(1), yang berarti operasi dapat dilakukan dengan waktu yang konstan, terlepas dari jumlah node dalam daftar. Penghapusan juga dapat dilakukan di awal, tengah, atau akhir daftar dengan kompleksitas waktu rata-rata O(1).

JRIIN: Jurnal Riset Informatika dan Inovasi
Volume 1, No. 12 Mei 2024
ISSN 3025-0919 (media online)
Hal 1329-1331

2.2 Implementasi DLL dalam Bahasa Pemrograman

Banyak pustaka dan contoh kode yang tersedia untuk implementasi DLL dalam berbagai bahasa pemrograman. Pustaka standar C++ menyediakan kelas list yang dapat digunakan untuk implementasi DLL. Bahasa pemrograman seperti Python, Java, dan C# juga menyediakan pustaka bawaan atau pihak ketiga untuk implementasi DLL.

3. METODOLOGI

3.1 Insertion

1. Membuat Node Baru: Alokasikan memori untuk node baru dan inisialisasi nilai data di dalamnya.
2. Menentukan Posisi Penyisipan:
   a) Di Depan:
   a) Ubah pointer 'head' untuk menunjuk ke node baru.
   b) Atur pointer 'prev' dari node baru ke NULL (sebagai node pertama).
   c) Atur pointer 'next' dari node baru ke node yang sebelumnya merupakan 'head'.
   d) Perbarui pointer 'prev' dari node yang sebelumnya merupakan 'head' untuk menunjuk ke node baru.

b) Di Tengah:
   a) Temukan node sebelum node target di mana penyisipan akan dilakukan.
   b) Ubah pointer 'next' dari node sebelum ke node baru.
   c) Ubah pointer 'prev' dari node baru ke node sebelum.
   d) Ubah pointer 'next' dari node baru ke node target.
   e) Ubah pointer 'prev' dari node target ke node baru.

c) Di Belakang:
   a) Temukan node terakhir (tail) dalam DLL.
   b) Ubah pointer 'next' dari node terakhir ke node baru.
   c) Atur pointer 'prev' dari node baru ke node terakhir.
   d) Atur pointer 'next' dari node baru ke NULL (sebagai node terakhir).

3.2 Deletion

1. Menemukan Node yang Akan Dihapus:
   a) Gunakan nilai data sebagai kunci pencarian.
   b) Lintasi DLL dari awal hingga node yang sesuai ditemukan.
2. Menghapus Node:

a) Kasus Pertama: Node Pertama (Head):
   a) Ubah pointer 'head' untuk menunjuk ke node berikutnya.
   b) Jika DLL hanya memiliki satu node, ubah 'head' dan 'tail' menjadi NULL.

b) Kasus Kedua: Node Terakhir (Tail):
   a) Ubah pointer 'next' dari node sebelum tail ke NULL.
   b) Ubah 'tail' untuk menunjuk ke node sebelum tail.

JRIIN: Jurnal Riset Informatika dan Inovasi
Volume 1, No. 12 Mei 2024
ISSN 3025-0919 (media online)
Hal 1329-1331

c) Kasus Ketiga: Node di Tengah:

a) Ubah pointer 'next' dari node sebelum ke node setelah node yang dihapus.

b) Ubah pointer 'prev' dari node setelah ke node sebelum node yang dihapus.

4. ANALISA DAN PEMBAHASAN

4.1 Hasil

a) Penyisipan dan penghapusan data di sembarang posisi: DLL memungkinkan penambahan atau penghapusan node di mana saja dalam daftar, tidak hanya di awal atau akhir seperti pada SLL.
b) Traversal maju dan mundur: DLL mendukung traversal daftar ke arah depan dan belakang, memudahkan akses data dari berbagai arah.
c) Modifikasi data yang mudah: DLL memungkinkan modifikasi data dalam node dengan mudah karena aksesibilitasnya yang lebih fleksibel.

4.2 Pembahasan

a) operasi penyisipan di awal dan akhir daftar, serta penghapusan di awal dan akhir daftar.

b) Operasi lain seperti penyisipan dan penghapusan di tengah daftar dapat diimplementasikan dengan modifikasi yang serupa.

c) DLL menawarkan fleksibilitas dan efisiensi dalam operasi manipulasi data, membuatnya cocok untuk aplikasi yang membutuhkan akses data acak dan modifikasi daftar yang sering.

5. KESIMPULAN

Algoritma Doubly Linked List sangat berguna untuk operasi penyisipan dan penghapusan karena fleksibilitas akses dua arah. Untuk penyisipan, algoritma ini memungkinkan penambahan simpul baru di awal, tengah, atau akhir daftar dengan mudah karena setiap simpul memiliki penunjuk ke simpul sebelumnya dan berikutnya. Proses penyisipan hanya memerlukan penyesuaian penunjuk pada simpul yang terlibat tanpa harus menelusuri daftar dari awal, sehingga efisien. Dalam penghapusan, kelebihan utama Doubly Linked List adalah kemampuannya untuk mengakses simpul yang akan dihapus dan simpul-simpul di sekitarnya dengan mudah, membuat penghapusan menjadi lebih cepat.

Penyesuaian penunjuk previous dan next memungkinkan penghapusan simpul tanpa traversal panjang. Dengan demikian, Doubly Linked List memberikan efisiensi dan fleksibilitas yang lebih tinggi dalam operasi penyisipan dan penghapusan dibandingkan dengan Singly Linked List, meskipun dengan penggunaan memori tambahan dan kompleksitas yang lebih tinggi.

REFERENCES

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). Cambridge, MA: MIT Press.
Kumar, S., & Singhal, M. (2004). A comparative performance analysis of single linked list and double linked list. International Journal of Computer Science and Network Security, 4(10), 248-254.
Reddy, S. K., & Ramakrishnan, C. R. (2007). Performance comparison of various data structures for implementing symbol table. International Journal of Computer Science and Network Security, 7(10), 167-172.
Singhal, M., & Gupta, N. (2010). Performance improvement of doubly linked list using caching technique. International Journal of Computer Applications, 1(9), 33-36.
Willoughby, R. A., & Guyton, G. D. (2008). Indexing techniques for doubly linked lists. ACM SIGCSE Bulletin,