Linked List

A. Pengenalan

LinkedList adalah struktur data linier yang terdiri dari serangkaian simpul yang terhubung satu sama lain melalui referensi. Setiap simpul dalam linked list menyimpan elemen data dan referensi ke simpul berikutnya dalam urutan. Tidak seperti array, elemen-elemen dalam linked list tidak disimpan secara berurutan dalam memori, melainkan tersebar di lokasi-lokasi yang berbeda di memori.

Setiap linked list terdiri dari dua bagian utama: kepala (head) dan ekor (tail). Kepala adalah simpul pertama dalam linked list, sedangkan ekor adalah simpul terakhir. Jika sebuah linked list kosong, maka kepala dan ekor akan menunjuk ke null.

LinkedList memiliki beberapa kelebihan dan kekurangan dibandingkan dengan struktur data lain seperti array:

Kelebihan

1. Kemampuan Penyisipan dan Penghapusan: Operasi penyisipan dan penghapusan elemen di linked list dapat dilakukan dengan cepat dan efisien, karena hanya perlu memperbarui referensi simpul terdekat.
2. Ukuran Dinamis: Linked list dapat tumbuh dan menyusut secara dinamis saat elemen-elemen ditambahkan atau dihapus, tanpa memerlukan alokasi ulang memori seperti pada array.

Kekurangan

1. Akses Acak yang Lambat: Dalam linked list, akses ke elemen-elemen tidak dapat dilakukan secara langsung seperti pada array. Untuk mengakses elemen di indeks tertentu, perlu dilakukan pencarian dari awal linked list hingga mencapai indeks yang diinginkan.
2. Penggunaan Memori yang Lebih Besar: Linked list memerlukan lebih banyak ruang memori daripada array untuk menyimpan referensi antar simpul.

B. Kapan Harus Menggunakan

LinkedList adalah struktur data yang berguna dalam beberapa situasi. Pertama, ketika kita perlu sering menyisipkan atau menghapus elemen di tengah-tengah koleksi data, LinkedList dapat menjadi pilihan yang baik. Operasi ini dapat dilakukan dengan cepat dan efisien, tanpa memerlukan pergeseran elemen lain seperti yang terjadi pada array. Selain itu, jika ukuran koleksi data dapat berubah secara dinamis, LinkedList dapat digunakan untuk menambah dan menghapus elemen dengan mudah, tanpa perlu melakukan alokasi ulang memori seperti pada array.

Namun, perlu diingat bahwa penggunaan LinkedList juga memiliki beberapa kekurangan. Pertama, LinkedList menggunakan lebih banyak ruang memori daripada array, karena setiap simpul memerlukan ruang tambahan untuk menyimpan referensi. Jadi, jika memori sangat terbatas, penggunaan LinkedList mungkin tidak efisien. Selain itu, jika akses acak atau pencarian elemen sering terjadi, LinkedList mungkin tidak menjadi pilihan terbaik, karena akses langsung ke elemen di LinkedList tidak efisien. LinkedList lebih cocok digunakan untuk akses sekuensial atau akses ke elemen awal atau akhir.

Dalam beberapa kasus, penggunaan LinkedList dapat mempermudah implementasi struktur data lain yang lebih kompleks. Misalnya, dengan bantuan LinkedList, kita dapat mengimplementasikan struktur data seperti stack atau queue dengan mudah.

Pilihan struktur data, termasuk penggunaan LinkedList, harus didasarkan pada kebutuhan khusus dari aplikasi atau masalah yang dihadapi. Terkadang, struktur data lain seperti array dapat lebih efisien tergantung pada skenario penggunaan yang diinginkan.

C. Contoh Implementasi

Menggunakan LinkedList dengan metode insert, update dan delete.

main.java

```
class Node {

  int data; // Nilai yang disimpan dalam simpul
  Node next; // Referensi ke simpul berikutnya

  public Node(int data) {
    this.data = data;
    this.next = null;
  }
}

class LinkedList {

  Node head; // Simpul pertama dalam LinkedList

  public LinkedList() {
    this.head = null;
  }

  // Menyisipkan elemen baru ke LinkedList
  public void insert(int data) {
    Node newNode = new Node(data);

    if (head == null) {
      head = newNode; // Jika LinkedList kosong, elemen baru menjadi simpul pertama (head)
    } else {
      Node current = head;
      while (current.next != null) {
        current = current.next;
      }
      current.next = newNode; // Menyisipkan elemen baru di simpul terakhir
    }
  }

  // Menghapus elemen dengan nilai tertentu dari LinkedList
  public void delete(int data) {
    if (head == null) {
      System.out.println("LinkedList kosong.");
      return;
    }

    if (head.data == data) {
      head = head.next; // Jika elemen yang akan dihapus adalah elemen pertama (head)
      System.out.println(data + " dihapus dari LinkedList.");
      return;
    }

    Node current = head;
    Node prev = null;
    while (current != null && current.data != data) {
      prev = current;
      current = current.next;
    }

    if (current == null) {
      System.out.println(data + " tidak ditemukan dalam LinkedList.");
      return;
    }

    prev.next = current.next; // Menghapus elemen dengan mengubah referensi node sebelumnya
    System.out.println(data + " dihapus dari LinkedList.");
  }

  // Mengubah nilai elemen dengan nilai lama tertentu menjadi nilai baru dalam LinkedList
  public void update(int oldData, int newData) {
    if (head == null) {
      System.out.println("LinkedList kosong.");
      return;
    }

    Node current = head;
    while (current != null && current.data != oldData) {
      current = current.next;
    }

    if (current == null) {
      System.out.println(oldData + " tidak ditemukan dalam LinkedList.");
      return;
    }

    current.data = newData; // Mengubah nilai elemen
    System.out.println(oldData + " diubah menjadi " + newData + " dalam LinkedList.");
  }

  // Menampilkan isi LinkedList
  public void display() {
    if (head == null) {
      System.out.println("LinkedList kosong.");
      return;
    }

    Node current = head;
    System.out.print("LinkedList: ");
    while (current != null) {
      System.out.print(current.data + " ");
      current = current.next;
    }
    System.out.println();
  }
}

public class Main {

  public static void main(String[] args) {
    LinkedList list = new LinkedList();

    list.insert(10);
    list.insert(20);
    list.insert(30);

    list.display(); // Output: LinkedList: 10 20 30

    list.delete(20); // Output: 20 dihapus dari LinkedList
    list.display(); // Output: LinkedList: 10 30

    list.update(10, 50); // Output: 10 diubah menjadi 50 dalam LinkedList
    list.display(); // Output: LinkedList: 50 30
  }
}
```

CodeImage

D. Studi Kasus

Implementasi daftar kontak dalam sebuah aplikasi telepon. Dalam implementasi ini, setiap kontak akan direpresentasikan sebagai simpul dalam LinkedList. Setiap simpul akan memiliki atribut-atribut seperti nama, nomor telepon, dan alamat email.