

Circular Linked List

A. Pengenalan

Circular linked list adalah jenis struktur data linked list di mana setiap node terhubung ke node berikutnya dan node terakhir terhubung kembali ke node pertama, membentuk siklus tertutup. Dalam circular linked list, tidak ada akhir yang sebenarnya atau null pointer pada node terakhir, karena node terakhir mengacu kembali ke node pertama.

Pada circular linked list, traversal (penjelajahan) dapat dimulai dari mana saja dalam siklus. Misalnya, jika kita memulai penjelajahan dari node pertama, kita dapat mengunjungi setiap node dalam siklus sampai kembali ke node pertama lagi. Dalam hal ini, penjelajahan dapat berlangsung secara tak terbatas.

Keuntungan dari circular linked list adalah kemampuannya untuk mendukung operasi seperti iterasi berkelanjutan tanpa memerlukan perulangan atau periksa akhir. Ini berguna dalam implementasi antrian berputar, pemrosesan sirkular, atau pemecahan masalah dengan siklus alami.

Namun, perlu diingat bahwa penggunaan circular linked list membutuhkan perhatian lebih pada manajemen memori dan terminasi loop. Dalam beberapa kasus, jika tidak dikelola dengan baik, itu dapat menyebabkan masalah seperti kebocoran memori atau masalah yang berhubungan dengan penghentian loop tak terbatas. Oleh karena itu, perlu diimplementasikan dengan hati-hati dan memastikan bahwa siklusnya benar-benar terputus saat diperlukan.

B. Kapan Harus Menggunakan

Circular linked list adalah jenis struktur data linked list di mana setiap elemen (node) memiliki referensi atau pointer ke node berikutnya dalam daftar, dan node terakhir memiliki referensi kembali ke node pertama. Dalam struktur data ini, kita dapat menjelajahi elemen-elemen dalam siklus terus-menerus, karena tidak ada akhir yang jelas seperti pada linked list linear.


Circular linked list sering digunakan dalam situasi di mana kita perlu menjelajahi elemen-elemen dalam suatu struktur data secara berkelanjutan tanpa memerlukan perulangan atau pengecekan akhir. Contohnya, dalam implementasi antrian berputar (circular queue), elemen-elemen masuk dan keluar dari antrian pada posisi yang berbeda-beda. Dengan menggunakan circular linked list, elemen baru dapat ditambahkan di ujung belakang antrian dan elemen dapat dihapus dari ujung depan antrian dengan efisien.

Selain itu, circular linked list juga cocok untuk pemrosesan sirkular, seperti pemrosesan data sensor yang berputar atau algoritma yang melibatkan perulangan siklus. Dalam kasus seperti itu, circular linked list memungkinkan kita untuk secara efisien menjelajahi elemen-elemen dalam siklus tanpa harus memeriksa akhir atau menghentikan perulangan.


Namun, penggunaan circular linked list perlu dipertimbangkan dengan hati-hati. Mereka dapat memperkenalkan kompleksitas tambahan dalam manajemen memori dan menghentikan perulangan. Oleh karena itu, penting untuk mempertimbangkan kebutuhan spesifik dari masalah yang sedang dihadapi dan memastikan bahwa manfaat yang diberikan oleh circular linked list sebanding dengan kompleksitas yang diperlukan.

C. Contoh Implementasi

Membuat class Node, merupakan atribut setiap Node yang akan digunakan untuk setiap elemen pada Circular Linked List.



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

 CodelImage

Membuat class CircularLinkedList, sebagai tempat untuk mengatur logika, agar bisa memodifikasi CircularLinkedList

```
public class CircularLinkedList {

    private Node head;

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

    public boolean isEmpty() {
        return head == null;
    }

    public void insert(int data) {
        Node newNode = new Node(data);

        if (isEmpty()) {
            head = newNode;
            head.next = head; // Menghubungkan node pertama dengan dirinya sendiri
        } else {
            Node current = head;

            // Menemukan node terakhir dan menghubungkannya dengan node baru
            while (current.next != head) {
                current = current.next;
            }

            current.next = newNode;
            newNode.next = head; // Menghubungkan node baru dengan node pertama
        }

        System.out.println("Node with data " + data + " inserted");
    }

    public void update(int oldData, int newData) {
        if (isEmpty()) {
            System.out.println("Circular linked list is empty");
            return;
        }

        Node current = head;

        // Mencari node dengan data yang akan diupdate
        do {
            if (current.data == oldData) {
                current.data = newData;
                System.out.println("Node with data " + oldData + " updated to " + newData);
                return;
            }
            current = current.next;
        } while (current != head);

        System.out.println("Node with data " + oldData + " not found");
    }

    public void delete(int data) {
        if (isEmpty()) {
            System.out.println("Circular linked list is empty");
            return;
        }

        Node current = head;
        Node previous = null;

        // Mencari node dengan data yang dihapus
        do {
            if (current.data == data) {
                break;
            }
            previous = current;
            current = current.next;
        } while (current != head);

        // Menghapus node yang ditemukan
        if (current == head && current.next == head) {
            head = null; // Jika hanya ada satu node dalam circular linked list
        } else if (current == head) {
            Node lastNode = head;
            while (lastNode.next != head) {
                lastNode = lastNode.next;
            }
            head = head.next;
            lastNode.next = head;
        } else if (current == head && previous != null) {
            previous.next = head.next;
            head = head.next;
        } else if (current != head) {
            previous.next = current.next;
        }

        System.out.println("Node with data " + data + " deleted");
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Circular linked list is empty");
            return;
        }

        Node current = head;

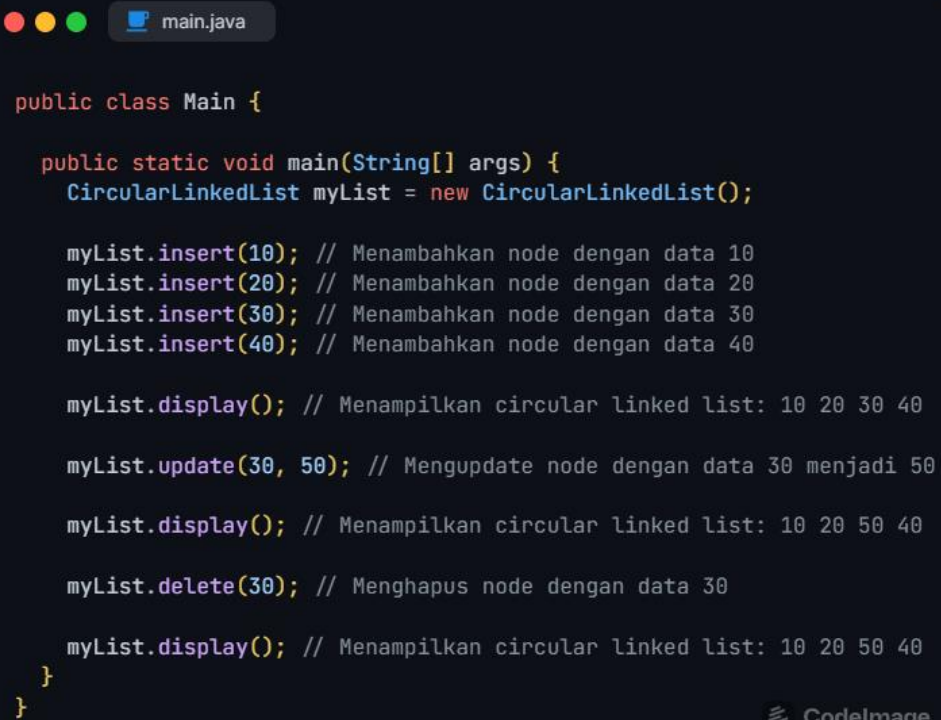
        System.out.print("Circular linked list: ");

        // Menampilkan semua elemen dalam circular linked list
        do {
            System.out.print(current.data + " ");
            current = current.next;
        } while (current != head);

        System.out.println();
    }
}
```

CodeImage

Membuat class Main, tempat aplikasi dieksekusi



```
public class Main {  
  
    public static void main(String[] args) {  
        CircularLinkedList myList = new CircularLinkedList();  
  
        myList.insert(10); // Menambahkan node dengan data 10  
        myList.insert(20); // Menambahkan node dengan data 20  
        myList.insert(30); // Menambahkan node dengan data 30  
        myList.insert(40); // Menambahkan node dengan data 40  
  
        myList.display(); // Menampilkan circular linked list: 10 20 30 40  
  
        myList.update(30, 50); // Mengupdate node dengan data 30 menjadi 50  
  
        myList.display(); // Menampilkan circular linked list: 10 20 50 40  
  
        myList.delete(30); // Menghapus node dengan data 30  
  
        myList.display(); // Menampilkan circular linked list: 10 20 50 40  
    }  
}
```

CodeImage

D. Studi Kasus

Buatlah program untuk memodelkan manajemen pesanan makanan di sebuah restoran menggunakan circular linked list. Setiap pesanan makanan memiliki nomor pesanan dan nama pelanggan. Implementasikan fungsi-fungsi `addOrder` untuk menambahkan pesanan ke daftar pesanan, `removeOrder` untuk menghapus pesanan yang telah selesai diproses, dan `displayOrders` untuk menampilkan seluruh daftar pesanan saat ini.