Metode Algoritma dan Struktur Data di Java

Berikut adalah metode-metode umum dalam algoritma dan struktur data di Java, beserta fungsi dan contohnya:

1. Array dan ArrayList

Keterangan:
Array dan ArrayList adalah struktur data untuk menyimpan kumpulan elemen dengan tipe data yang sama. ArrayList adalah implementasi dinamis dari Array yang dapat mengubah ukurannya secara otomatis.

Metode:

- `add()` - Fungsi: Menambahkan elemen ke ArrayList
- `get()` - Fungsi: Mengambil elemen dari posisi tertentu
- `remove()` - Fungsi: Menghapus elemen tertentu
- `size()` - Fungsi: Mendapatkan ukuran ArrayList
- `sort()` - Fungsi: Mengurutkan elemen

Contoh:

java

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();

        // Menambahkan elemen
        numbers.add(10);
        numbers.add(5);
        numbers.add(20);

        // Mendapatkan elemen
        System.out.println("Elemen pada indeks 1: " + numbers.get(1));

        // Mengurutkan
        Collections.sort(numbers);
        System.out.println("Setelah diurutkan: " + numbers);

        // Menghapus elemen
        numbers.remove(0);
        System.out.println("Setelah menghapus: " + numbers);
    }
}
```

2. LinkedList

Keterangan:

LinkedList adalah implementasi struktur data doubly-linked list yang menyimpan elemen dalam

bentuk node yang terhubung. Cocok untuk operasi penyisipan dan penghapusan yang sering dilakukan.

Metode:

- `addFirst()` - Fungsi: Menambahkan di awal list
- `addLast()` - Fungsi: Menambahkan di akhir list
- `removeFirst()` - Fungsi: Menghapus elemen pertama
- `removeLast()` - Fungsi: Menghapus elemen terakhir
- `getFirst()` - Fungsi: Mengambil elemen pertama

Contoh:

java

```java
import java.util.LinkedList;

public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList<String> nama = new LinkedList<>();

        // Menambahkan elemen
        nama.addFirst("Budi");
        nama.addLast("Citra");
        nama.add("Dewi");

        System.out.println("LinkedList: " + nama);

        // Mengakses elemen pertama
        System.out.println("Elemen pertama: " + nama.getFirst());

        // Menghapus elemen terakhir
        nama.removeLast();
        System.out.println("Setelah menghapus terakhir: " + nama);
    }
}
```

3. Stack

Keterangan:

Stack adalah struktur data LIFO (Last-In-First-Out) di mana elemen yang terakhir masuk akan pertama keluar, seperti tumpukan buku.

Metode:

- `push()` - Fungsi: Memasukkan elemen ke stack

- `pop()` - Fungsi: Mengeluarkan elemen dari stack
- `peek()` - Fungsi: Melihat elemen teratas tanpa menghapus
- `empty()` - Fungsi: Memeriksa apakah stack kosong

Contoh:

```java
import java.util.Stack;

public class StackDemo {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        // Push elemen
        stack.push(10);
        stack.push(20);
        stack.push(30);

        // Peek elemen teratas
        System.out.println("Elemen teratas: " + stack.peek());

        // Pop elemen
        System.out.println("Elemen yang di-pop: " + stack.pop());

        // Cek apakah kosong
        System.out.println("Apakah stack kosong? " + stack.empty());
    }
}
```

4. Queue dan PriorityQueue

Keterangan:

Queue adalah struktur data FIFO (First-In-First-Out) dimana elemen pertama yang masuk akan pertama keluar, seperti antrian. PriorityQueue adalah implementasi Queue yang mengurutkan elemen berdasarkan prioritas.

Metode:

- `offer()` - Fungsi: Menambahkan elemen ke queue
- `poll()` - Fungsi: Mengambil dan menghapus elemen terdepan
- `peek()` - Fungsi: Melihat elemen terdepan tanpa menghapus
- `isEmpty()` - Fungsi: Memeriksa apakah queue kosong

Contoh:

java

```java
import java.util.PriorityQueue;

public class QueueDemo {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        // Menambahkan elemen
        pq.offer(30);
        pq.offer(10);
        pq.offer(20);

        // Elemen terdepan (terkecil karena PriorityQueue)
        System.out.println("Elemen terdepan: " + pq.peek());

        // Mengambil dan menghapus elemen terdepan
        System.out.println("Elemen yang diambil: " + pq.poll());

        // Cek setelah pengambilan
        System.out.println("Elemen terdepan baru: " + pq.peek());
    }
}
```

5. HashMap dan HashSet

Keterangan:

HashMap menyimpan pasangan key-value, memungkinkan pengambilan nilai berdasarkan kunci. HashSet adalah implementasi Set yang menyimpan elemen unik tanpa duplikat.

Metode HashMap:

- `put()` - Fungsi: Menyimpan pasangan key-value

- `get()` - Fungsi: Mengambil nilai berdasarkan key
- `remove()` - Fungsi: Menghapus elemen berdasarkan key
- `containsKey()` - Fungsi: Memeriksa keberadaan key

Contoh HashMap:

```java
import java.util.HashMap;

public class HashMapDemo {
    public static void main(String[] args) {
        HashMap<String, Integer> nilai = new HashMap<>();

        // Menambahkan data
        nilai.put("Ani", 90);
        nilai.put("Budi", 85);
        nilai.put("Cici", 95);

        // Mengambil nilai
        System.out.println("Nilai Budi: " + nilai.get("Budi"));

        // Memeriksa keberadaan key
        System.out.println("Ada Dedi? " + nilai.containsKey("Dedi"));

        // Menghapus data
        nilai.remove("Ani");
        System.out.println("Setelah menghapus: " + nilai);
    }
}
```

6. Collections

Keterangan:

Collections adalah framework yang menyediakan metode-metode untuk memanipulasi dan mengelola kumpulan objek secara efisien.

Metode:

- `sort()` - Fungsi: Mengurutkan elemen dalam collection
- `reverse()` - Fungsi: Membalikkan urutan elemen
- `shuffle()` - Fungsi: Mengacak urutan elemen
- `binarySearch()` - Fungsi: Mencari elemen dalam collection yang sudah terurut
- `max()` - Fungsi: Mencari nilai maksimum dalam collection
- `min()` - Fungsi: Mencari nilai minimum dalam collection
- `frequency()` - Fungsi: Menghitung frekuensi elemen dalam collection
- `disjoint()` - Fungsi: Memeriksa apakah dua collection tidak memiliki elemen yang sama
- `unmodifiableList()` - Fungsi: Membuat list yang tidak dapat dimodifikasi

Contoh:

java

```java
import java.util.*;

public class CollectionsDemo {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Mengurutkan
        Collections.sort(fruits);
        System.out.println("Setelah diurutkan: " + fruits);

        // Membalikkan urutan
        Collections.reverse(fruits);
        System.out.println("Setelah dibalik: " + fruits);

        // Mengacak urutan
        Collections.shuffle(fruits);
        System.out.println("Setelah diacak: " + fruits);

        // Mencari nilai maksimum dan minimum
        String max = Collections.max(fruits);
        String min = Collections.min(fruits);
        System.out.println("Nilai maksimum: " + max);
        System.out.println("Nilai minimum: " + min);
    }
}
```

7. Algoritma Sorting

Keterangan:

Algoritma pengurutan untuk mengurutkan elemen dalam array atau collection.

Metode:

- `Arrays.sort()` - Fungsi: Mengurutkan array
- `Collections.sort()` - Fungsi: Mengurutkan collection

Contoh:

```java
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;

public class SortingDemo {
    public static void main(String[] args) {
        // Mengurutkan array
        int[] numbers = {5, 2, 9, 1, 3};
        Arrays.sort(numbers);
        System.out.println("Array terurut: " + Arrays.toString(numbers));

        // Mengurutkan ArrayList
        ArrayList<String> names = new ArrayList<>();
        names.add("Zaki");
        names.add("Andi");
        names.add("Yudi");

        Collections.sort(names);
        System.out.println("ArrayList terurut: " + names);
    }
}
```

8. Binary Search

Keterangan:

Algoritma pencarian untuk menemukan posisi elemen dalam array atau collection yang sudah terurut, dengan kompleksitas waktu O(log n).

Metode:

- `Arrays.binarySearch()` - Fungsi: Pencarian pada array terurut
- `Collections.binarySearch()` - Fungsi: Pencarian pada collection terurut

Contoh:

```java
import java.util.Arrays;

public class BinarySearchDemo {
    public static void main(String[] args) {
        int[] numbers = {1, 3, 5, 7, 9, 11};

        // Mencari elemen 7
        int index = Arrays.binarySearch(numbers, 7);
        System.out.println("Elemen 7 ditemukan pada indeks: " + index);

        // Mencari elemen yang tidak ada
        int notFound = Arrays.binarySearch(numbers, 8);
        System.out.println("Elemen 8 seharusnya pada indeks: " + (-notFound - 1));
    }
}
```

9. Graph

Keterangan:

Graph adalah struktur data yang terdiri dari simpul (vertex) dan sisi (edge) yang menghubungkan antar simpul, berguna untuk merepresentasikan jaringan dan relasi.

Implementasi dengan adjacency list:

```
java
```

```java
import java.util.ArrayList;
import java.util.LinkedList;

public class GraphDemo {
    public static void main(String[] args) {
        // Mengimplementasikan graph dengan adjacency list
        int vertices = 5;
        ArrayList<LinkedList<Integer>> adjList = new ArrayList<>(vertices);

        // Inisialisasi adjacency list
        for (int i = 0; i < vertices; i++) {
            adjList.add(new LinkedList<>());
        }

        // Menambahkan edge
        addEdge(adjList, 0, 1);
        addEdge(adjList, 0, 4);
        addEdge(adjList, 1, 2);
        addEdge(adjList, 1, 3);
        addEdge(adjList, 1, 4);

        // Menampilkan graph
        printGraph(adjList);
    }

    // Fungsi untuk menambahkan edge
    static void addEdge(ArrayList<LinkedList<Integer>> adjList, int u, int v) {
        adjList.get(u).add(v);
        adjList.get(v).add(u); // Untuk graph tidak terarah
    }

    // Fungsi untuk menampilkan graph
    static void printGraph(ArrayList<LinkedList<Integer>> adjList) {
```

```
        for (int i = 0; i < adjList.size(); i++) {
            System.out.print("Vertex " + i + " terhubung ke: ");
            for (Integer neighbor : adjList.get(i)) {
                System.out.print(neighbor + " ");
            }
            System.out.println();
        }
    }
}
```

10. Tree

Keterangan:

Tree adalah struktur data hierarkis yang terdiri dari node yang terhubung dalam bentuk pohon. Binary Search Tree (BST) adalah jenis khusus tree di mana nilai di kiri lebih kecil dari node induk dan nilai di kanan lebih besar.

Implementasi Binary Search Tree:

```
java
```

```java
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinarySearchTreeDemo {
    Node root;

    BinarySearchTreeDemo() {
        root = null;
    }

    // Menambahkan node baru
    void insert(int data) {
        root = insertRec(root, data);
    }

    Node insertRec(Node root, int data) {
        if (root == null) {
            root = new Node(data);
            return root;
        }

        if (data < root.data)
            root.left = insertRec(root.left, data);
        else if (data > root.data)
            root.right = insertRec(root.right, data);
```

```
        return root;
    }

    // Menelusuri tree secara inorder
    void inorder() {
        inorderRec(root);
    }

    void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.data + " ");
            inorderRec(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTreeDemo tree = new BinarySearchTreeDemo();

        // Menambahkan node
        tree.insert(50);
        tree.insert(30);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.insert(60);
        tree.insert(80);

        // Menampilkan tree dengan traversal inorder
        System.out.print("Inorder traversal: ");
        tree.inorder();
    }
}
```

11. Algoritma Pencarian Jalur

Keterangan:

Algoritma pencarian jalur digunakan untuk menemukan jalur terpendek dalam graph. Algoritma Dijkstra mencari jalur terpendek dari satu simpul sumber ke semua simpul lainnya.

Implementasi Algoritma Dijkstra:

```
java
```

```java
import java.util.*;

public class DijkstraDemo {
    static final int V = 9; // Jumlah vertex

    // Menemukan vertex dengan nilai jarak minimum
    int minDistance(int dist[], Boolean sptSet[]) {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }
        }

        return min_index;
    }

    // Mencetak array jarak
    void printSolution(int dist[]) {
        System.out.println("Vertex \t Jarak dari Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " \t\t " + dist[i]);
    }

    // Algoritma Dijkstra
    void dijkstra(int graph[][], int src) {
        int dist[] = new int[V]; // Array untuk menyimpan jarak terpendek
        Boolean sptSet[] = new Boolean[V]; // Vertex yang sudah diproses

        // Inisialisasi
        for (int i = 0; i < V; i++) {
```

```
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Jarak dari source ke dirinya sendiri selalu 0
        dist[src] = 0;

        // Mencari jalur terpendek untuk semua vertex
        for (int count = 0; count < V - 1; count++) {
            int u = minDistance(dist, sptSet);
            sptSet[u] = true;

            for (int v = 0; v < V; v++) {
                if (!sptSet[v] && graph[u][v] != 0 &&
                    dist[u] != Integer.MAX_VALUE &&
                    dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
            }
        }

        printSolution(dist);
    }

    public static void main(String[] args) {
        int graph[][] = new int[][] {
            { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
```

```
            { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
        };

        DijkstraDemo t = new DijkstraDemo();
        t.dijkstra(graph, 0);
    }
}
```

12. Heap

Keterangan:

Heap adalah struktur data pohon khusus yang memenuhi properti heap, dimana nilai node induk selalu lebih besar (max heap) atau lebih kecil (min heap) dari node anaknya.

Implementasi Max Heap:

```
java
```

```java
public class MaxHeapDemo {
    private int[] Heap;
    private int size;
    private int maxsize;

    // Constructor untuk inisialisasi
    public MaxHeapDemo(int maxsize) {
        this.maxsize = maxsize;
        this.size = 0;
        Heap = new int[this.maxsize + 1];
        Heap[0] = Integer.MAX_VALUE;
    }

    // Fungsi untuk mengembalikan posisi parent
    private int parent(int pos) {
        return pos / 2;
    }

    // Fungsi untuk mengembalikan posisi left child
    private int leftChild(int pos) {
        return (2 * pos);
    }

    // Fungsi untuk mengembalikan posisi right child
    private int rightChild(int pos) {
        return (2 * pos) + 1;
    }

    // Fungsi untuk swap dua node
    private void swap(int fpos, int spos) {
        int tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
```

```
    Heap[spos] = tmp;
}

// Fungsi untuk heapify-down
private void maxHeapify(int pos) {
    if (pos >= (size / 2) && pos <= size)
        return;

    if (Heap[pos] < Heap[leftChild(pos)] ||
        Heap[pos] < Heap[rightChild(pos)]) {

        if (Heap[leftChild(pos)] > Heap[rightChild(pos)]) {
            swap(pos, leftChild(pos));
            maxHeapify(leftChild(pos));
        } else {
            swap(pos, rightChild(pos));
            maxHeapify(rightChild(pos));
        }
    }
}

// Fungsi untuk insert elemen
public void insert(int element) {
    Heap[++size] = element;
    int current = size;

    while (Heap[current] > Heap[parent(current)]) {
        swap(current, parent(current));
        current = parent(current);
    }
}

// Fungsi untuk extract max
```

```java
public int extractMax() {
    int popped = Heap[1];
    Heap[1] = Heap[size--];
    maxHeapify(1);
    return popped;
}

// Fungsi untuk print heap
public void print() {
    for (int i = 1; i <= size / 2; i++) {
        System.out.print("Parent: " + Heap[i] +
                         " Left Child: " + Heap[2 * i] +
                         " Right Child: " + Heap[2 * i + 1]);
        System.out.println();
    }
}

public static void main(String[] args) {
    MaxHeapDemo maxHeap = new MaxHeapDemo(15);

    maxHeap.insert(5);
    maxHeap.insert(3);
    maxHeap.insert(17);
    maxHeap.insert(10);
    maxHeap.insert(84);
    maxHeap.insert(19);
    maxHeap.insert(6);
    maxHeap.insert(22);
    maxHeap.insert(9);

    maxHeap.print();

    System.out.println("Elemen terbesar: " + maxHeap.extractMax());
```

```
        }
    }
```

13. Trie

Keterangan:
Trie adalah struktur data pohon yang digunakan untuk menyimpan kumpulan string, berguna untuk pencarian prefix dan autocomplete.

Implementasi Trie:

```
java
```

```
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;

    TrieNode() {
        isEndOfWord = false;
        for (int i = 0; i < 26; i++)
            children[i] = null;
    }
}

public class TrieDemo {
    static TrieNode root;

    // Fungsi untuk insert kata ke trie
    static void insert(String key) {
        int level;
        int length = key.length();
        int index;

        TrieNode pCrawl = root;

        for (level = 0; level < length; level++) {
            index = key.charAt(level) - 'a';
            if (pCrawl.children[index] == null)
                pCrawl.children[index] = new TrieNode();

            pCrawl = pCrawl.children[index];
        }

        pCrawl.isEndOfWord = true;
    }
```

```java
// Fungsi untuk mencari kata dalam trie
static boolean search(String key) {
    int level;
    int length = key.length();
    int index;
    TrieNode pCrawl = root;

    for (level = 0; level < length; level++) {
        index = key.charAt(level) - 'a';

        if (pCrawl.children[index] == null)
            return false;

        pCrawl = pCrawl.children[index];
    }

    return (pCrawl.isEndOfWord);
}

public static void main(String args[]) {
    // Input keys (kata-kata yang akan disimpan)
    String keys[] = {"the", "a", "there", "answer", "any", "by", "bye", "their"};

    root = new TrieNode();

    // Construct trie
    for (int i = 0; i < keys.length; i++)
        insert(keys[i]);

    // Search for different keys
    if(search("the"))
        System.out.println("the --- Present in trie");
    else
```

```
            System.out.println("the --- Not present in trie");

        if(search("these"))
            System.out.println("these --- Present in trie");
        else
            System.out.println("these --- Not present in trie");
    }
}
```

14. Union-Find

Keterangan:

Union-Find atau Disjoint-Set adalah struktur data yang melacak elemen-elemen yang dipartisi menjadi satu set atau lebih yang terpisah, berguna untuk algoritma Kruskal dan deteksi siklus dalam graph.

Implementasi Union-Find:

```
java
```

```java
public class UnionFindDemo {
    int[] parent, rank;
    int n;

    // Constructor
    public UnionFindDemo(int n) {
        this.n = n;
        parent = new int[n];
        rank = new int[n];

        // Set setiap elemen sebagai parent dari dirinya sendiri
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    // Fungsi untuk menemukan set yang memiliki elemen x
    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }

    // Fungsi untuk menggabungkan dua set
    void union(int x, int y) {
        int xRoot = find(x);
        int yRoot = find(y);

        if (xRoot == yRoot)
            return;

        // Menggabungkan berdasarkan rank
        if (rank[xRoot] < rank[yRoot])
```

```
            parent[xRoot] = yRoot;
        else if (rank[xRoot] > rank[yRoot])
            parent[yRoot] = xRoot;
        else {
            parent[yRoot] = xRoot;
            rank[xRoot]++;
        }
    }

    public static void main(String[] args) {
        UnionFindDemo uf = new UnionFindDemo(5);

        // Gabungkan beberapa elemen
        uf.union(0, 2);
        uf.union(4, 2);
        uf.union(3, 1);

        // Cek apakah 4 dan 0 berada dalam set yang sama
        if (uf.find(4) == uf.find(0))
            System.out.println("4 dan 0 berada dalam set yang sama");
        else
            System.out.println("4 dan 0 tidak berada dalam set yang sama");

        // Cek apakah 1 dan 0 berada dalam set yang sama
        if (uf.find(1) == uf.find(0))
            System.out.println("1 dan 0 berada dalam set yang sama");
        else
            System.out.println("1 dan 0 tidak berada dalam set yang sama");
    }
}
```

Ini adalah beberapa metode dan struktur data dasar dalam Java yang umum digunakan dalam algoritma dan struktur data. Setiap metode memiliki fungsi spesifik yang membantu dalam

implementasi berbagai algoritma.