LAPORAN PRAKTIKUM PEKAN 7 STRUKTUR DATA



NAMA : IRGI FATIHUL IHSAN

NIM : 2311533010

DOSEN PENGAMPU : DR. WAHYUDI,S.T,MT.

DEPARTEMEN INFORMATIKA FAKULTAS TEKNOLOGI INFORMASI UNIVERSITAS ANDALAS

PADANG

1. Tujuan
   1. Memahami cara penggunaan Node dan Tree.
   2. Memahami cara kerja Node dan Tree pada data.
   3. Membuat program menggunakan Node dan Tree.

1. Kajian teori

Dalam ilmu komputer, struktur data merupakan komponen penting yang memungkinkan efisiensi dalam penyimpanan, pengambilan, dan pengelolaan data. Dua struktur data yang sering digunakan dan memiliki peran krusial dalam berbagai aplikasi adalah node dan tree. Pemahaman yang mendalam tentang kedua konsep ini sangat penting bagi para pengembang perangkat lunak dan peneliti di bidang ilmu computer.

* 1. Definisi Node

Node adalah unit dasar dari struktur data yang lebih kompleks. Setiap node biasanya terdiri dari dua komponen utama:

* + - Data: Informasi atau nilai yang disimpan dalam node.
    - Pointer atau Referensi: Alamat atau referensi yang mengarah ke node lain.

Dalam konteks yang lebih luas, node dapat ditemukan dalam berbagai struktur data seperti linked list, tree, graph, dan lain-lain. Dalam linked list, node dihubungkan secara linear, sedangkan dalam tree dan graph, node dihubungkan secara hierarkis atau non-linear.

* 1. Definisi Tree

Tree adalah struktur data yang terdiri dari node-node yang diorganisasikan dalam hierarki. Tree memiliki beberapa karakteristik unik:

* + - Akar (Root): Node paling atas dari tree.
    - Cabang (Branch): Node yang memiliki anak (child).
    - Daun (Leaf): Node yang tidak memiliki anak.
    - Tingkat (Level): Posisi node dalam hierarki tree, yang dimulai dari root pada level 0.

Tree digunakan dalam berbagai aplikasi seperti pengolahan database, sistem file, dan algoritma pencarian.

* 1. Jenis-jenis Tree

Ada berbagai jenis tree yang sering digunakan dalam ilmu komputer, antara lain:

* + - Binary Tree: Setiap node memiliki paling banyak dua anak.
    - Binary Search Tree (BST): Binary tree yang terurut sehingga memudahkan pencarian.
    - AVL Tree: Binary search tree yang seimbang.
    - Heap: Tree yang memenuhi sifat heap, di mana nilai parent selalu lebih besar atau lebih kecil dari nilai anak-anaknya.
    - B-Tree: Tree yang digunakan dalam sistem database dan sistem file untuk menjaga data tetap terurut dan mudah diakses.

* 1. Pentingnya Node dan Tree dalam Ilmu Komputer

Node dan tree adalah konsep fundamental yang mendasari banyak algoritma dan struktur data dalam ilmu komputer. Penggunaan tree memungkinkan pengolahan data yang efisien dalam berbagai aplikasi seperti:

* + - Pencarian dan Pengurutan: Algoritma pencarian seperti Binary Search Tree dan AVL Tree.
    - Pemrosesan Data Hierarkis: Struktur organisasi data dalam sistem file.
    - Database: B-Tree digunakan dalam indeks database untuk memudahkan pencarian data.

1. Langkah Kerja 1. Class Node Input:

package Pekan7;

public class Node {

int data;

Node left;

Node right;

public Node (int data) {

this.data = data;

left = null;

right = null;

}

public void setLeft(Node node) {

if (left == null) left = node;

}

public void setRight (Node node) {

if(right == null) right = node;

}

public Node getLeft() {

return left;

}

public Node getRight() {

return right;

}

public int getData() {

return data;

}

public void setData (int data) {

this.data = data;

}

void printPreorder (Node node) {

if (node == null) return;

System.***out***.print(node.data + " ");

printPreorder(node.left);

printPreorder(node.right);

}

void printPostorder(Node node) {

if (node == null) return;

printPostorder(node.left);

printPostorder(node.right);

System.***out***.print(node.data + " ");

}

void printInorder(Node node) {

if (node == null) return;

printInorder(node.left);

System.***out***.print(node.data + " ");

printInorder(node.right);

}

public String print() {

return this.print("",true,"");

}

public String print(String prefix, boolean isTail, String sb) {

if (right != null) {

right.print(prefix + (isTail ? "| " : " "), false, sb);

}

System.***out***.println(prefix+(isTail ? "\\-- " : "/-- ")+data);

if (left != null) {

left.print(prefix+(isTail ? " " : "| "), true, sb);

}

return sb;

}

}

* 1. Class BTree

Input:

package Pekan7;

public class BTree {

private Node root;

private Node currentNode;

public BTree() {

root = null;

}

public boolean search (int data) {

return search (root, data);

}

private boolean search (Node node, int data) {

if (node.getData() == data) return true;

if (node.getLeft() != null)

if(search(node.getLeft(), data))

return true;

if (node.getRight() != null)

if (search(node.getRight(), data))

return true;

return false;

}

public void printInorder() {

root.printInorder(root);

}

public void printPreOrder() {

root.printPreorder(root);

}

public void printPostOrder() {

root.printPostorder(root);

}

public Node getRoot() {

return root;

}

public boolean isEmpty() {

return root == null;

}

public int countNodes() {

return countNodes(root);

}

private int countNodes(Node node) {

int count = 1;

if (node == null) {

return 0;

} else {

count += countNodes(node.getLeft());

count += countNodes(node.getRight());

return count;

}

}

public void print() {

root.print();

}

public Node getCurrent() {

return currentNode;

}

public void setCurrent(Node node) {

this.currentNode = node;

}

public void setRoot(Node root) {

this.root = root;

}

}

* 1. Class TreeMain Input:

package Pekan7;

public class TreeMain {

public static void main (String[] args) {

// Membuat pohon

BTree tree = new BTree();

System.***out***.print("Jumlah simpul pohon: ");

System.***out***.println(tree.countNodes());

// menambah simpul data 1

Node root = new Node(1);

// menjadikan simpul 1 sebagai root

tree.setRoot(root);

System.***out***.println("Jumlah simpuljika hanya ada root");

System.***out***.println(tree.countNodes());

Node node2 = new Node (2);

Node node3 = new Node (3);

Node node4 = new Node (4);

Node node5 = new Node (5);

Node node6 = new Node (6);

Node node7 = new Node (7);

root.setLeft(node2);

node2.setLeft(node4);

node2.setRight(node5);

node5.setLeft(node7);

root.setRight(node3);

node3.setRight(node6);

// Set root

tree.setCurrent(tree.getRoot());

System.***out***.println("menampilkan simpul terakhir: ");

System.***out***.println(tree.getCurrent().getData());

System.***out***.println("Jumlah simpul ; setelah simpul 7 ditambahkan");

System.***out***.println(tree.countNodes());

System.***out***.println("InOrder: ");

tree.printInorder();

System.***out***.println("\nPreorder: ");

tree.printPreOrder();

System.***out***.println("\nPostorder: ");

tree.printPostOrder();

System.***out***.println("\nMenampilkan simpul dalam bentuk pohon");

tree.print();

}

}

Output:

Jumlah simpul pohon: 0

Jumlah simpuljika hanya ada root

1

menampilkan simpul terakhir:

1

Jumlah simpul ; setelah simpul 7 ditambahkan

7

InOrder:

4 2 7 5 1 3 6

Preorder:

1 2 4 5 7 3 6

Postorder:

4 7 5 2 6 3 1

Menampilkan simpul dalam bentuk pohon

| /-- 6

| /-- 3

\-- 1

| /-- 5

| | \-- 7

\-- 2

\-- 4

1. Kesimpulan

Dalam laporan ini, kita telah mengkaji secara mendalam konsep dan aplikasi dari node dan tree dalam ilmu komputer. Beberapa poin penting yang dapat disimpulkan adalah:

* 1. Pentingnya Node: Node merupakan unit dasar dari berbagai struktur data yang lebih kompleks. Setiap node berisi data dan pointer yang mengarah ke node lain, memungkinkan pembentukan struktur data seperti linked list, tree, dan graph.
  2. Struktur dan Fungsi Tree: Tree adalah struktur data hierarkis yang terdiri dari node- node yang saling terhubung. Tree memungkinkan pengolahan data secara efisien dengan karakteristik seperti root, branch, leaf, dan level. Tree digunakan dalam berbagai aplikasi mulai dari database hingga sistem file dan algoritma pencarian.
  3. Jenis-jenis Tree: Terdapat berbagai jenis tree yang memiliki karakteristik dan kegunaan spesifik, seperti Binary Tree, Binary Search Tree (BST), AVL Tree, Heap, dan B-Tree. Masing-masing jenis tree memiliki keunggulan dalam aplikasi tertentu, seperti pencarian, pengurutan, dan penyimpanan data yang terstruktur.
  4. Aplikasi Praktis: Implementasi tree dalam pemrograman memberikan solusi yang efisien untuk banyak masalah komputasi. Misalnya, BST mempermudah pencarian data yang cepat, AVL Tree menjaga keseimbangan untuk performa optimal, dan B- Tree digunakan dalam indeks database untuk akses data yang cepat.
  5. Analisis Performa: Tree memungkinkan algoritma yang efisien dalam hal waktu dan ruang. Dengan pemilihan jenis tree yang tepat, performa aplikasi dapat ditingkatkan secara signifikan. Analisis kompleksitas waktu dan ruang pada berbagai operasi tree seperti pencarian, penyisipan, dan penghapusan sangat penting untuk memahami efisiensi algoritma.

Secara keseluruhan, pemahaman yang mendalam mengenai node dan tree serta aplikasinya merupakan dasar yang kuat dalam pengembangan perangkat lunak dan penelitian ilmu komputer. Kemampuan untuk mengimplementasikan dan mengoptimalkan struktur data ini membuka peluang untuk menciptakan solusi yang lebih efisien dan efektif dalam berbagai bidang teknologi.