**LAPORAN PRAKTIKUM 9**

**STRUKTUR DATA**

**A logo of a university

AI-generated content may be incorrect.**

Dosen Pengampu:  
Dr. Wahyudi, S.T, M.Kom.

Disusun Oleh:  
Muhammad Luthfi Kautsar Rizata – 2311532020

**DEPARTEMEN INFORMATIKA**

**FAKULTAS TEKNOLOGI INFORMASI**

**UNIVERSITAS ANDALAS**

**2025**

A screen shot of a computer program

AI-generated content may be incorrect.**1. Class Node**

Bagian kode ini mendeklarasikan tiga atribut utama dalam class Node, yaitu int data, Node left, dan Node right. Atribut data digunakan untuk menyimpan nilai dari setiap simpul dalam pohon biner—dalam konteks ini bertipe int, jadi nilai yang disimpan berupa bilangan bulat. Kemudian left dan right masing-masing adalah referensi ke node anak kiri dan anak kanan. Karena struktur yang digunakan adalah binary tree, setiap node hanya bisa memiliki maksimal dua anak, yaitu satu di kiri dan satu di kanan. Ketiga atribut ini termasuk instance variable, artinya setiap objek Node yang dibuat akan menyimpan nilai data-nya sendiri serta tahu siapa anak kirinya (left) dan anak kanannya (right), kalau ada. Awalnya referensi anak kiri dan kanan ini biasanya di-set ke null saat node dibuat, dan bisa diatur nanti melalui method setter.

constructor Node(int data) otomatis dipanggil saat objek Node baru dibuat. Fungsinya adalah untuk menginisialisasi nilai awal dari objek tersebut. Dalam constructor ini, nilai yang diberikan melalui parameter data akan disimpan ke dalam atribut this.data, yang merupakan data utama dari node tersebut. Sementara itu, left dan right di-set ke null, yang artinya node ini belum memiliki anak kiri maupun anak kanan saat pertama kali dibuat. Jadi, saat kamu menulis Node nodeBaru = new Node(10);, node tersebut akan punya nilai 10, tanpa anak kiri atau kanan dulu. Struktur ini penting untuk memastikan bahwa setiap node dimulai dalam keadaan "kosong" dan bisa dikembangkan kemudian dengan menambahkan anak sesuai kebutuhan.

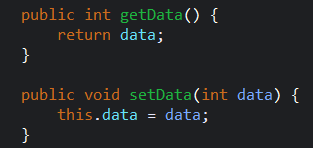
A computer screen with text

AI-generated content may be incorrect.

Selanjutnya, dua method yaitu setLeft(Node node) dan setRight(Node node) digunakan untuk menetapkan anak kiri dan anak kanan dari sebuah node, namun dengan kondisi tambahan: penetapan hanya dilakukan jika anak tersebut belum ada sebelumnya. Maksudnya, jika left atau right masih null, barulah node baru bisa ditetapkan ke posisi tersebut. Kalau anaknya sudah terisi, maka method ini tidak akan mengubahnya. Ini semacam bentuk proteksi agar struktur pohon tidak ditimpa sembarangan setelah anaknya terisi

A screen shot of a computer

AI-generated content may be incorrect.Method getLeft() dan getRight() berfungsi sebagai akses untuk mengambil referensi anak kiri dan kanan dari sebuah node. Keduanya merupakan getter sederhana yang tidak menerima parameter dan hanya mengembalikan nilai dari atribut left dan right secara langsung. Dengan menggunakan method ini, bagian lain dari program bisa mengetahui siapa anak kiri atau kanan dari suatu node tanpa mengakses atributnya secara langsung.

Method getData() dan setData(int data) adalah aksesori untuk menangani nilai yang disimpan dalam suatu node. getData() digunakan untuk mengambil nilai data dari node, sedangkan setData(int data) digunakan untuk menggantinya dengan nilai baru. Ini memberikan fleksibilitas jika suatu saat data dalam node perlu diubah tanpa membuat node baru.

A screen shot of a computer code

AI-generated content may be incorrect.Pada traversal preorder, urutannya adalah: cetak data node saat ini, lalu kunjungi anak kiri, lalu anak kanan. Ini cocok digunakan jika ingin memproses node sebelum anak-anaknya, misalnya dalam proses penyalinan struktur pohon. Traversal postorder kebalikannya: kunjungi anak kiri dulu, lalu anak kanan, terakhir baru node itu sendiri. Ini berguna kalau ingin menghapus pohon dari bawah ke atas. Terakhir, traversal inorder mengunjungi anak kiri dulu, lalu node itu sendiri, lalu anak kanan. Traversal ini sangat berguna khususnya jika pohonnya adalah binary search tree, karena akan mencetak nilai secara berurutan dari kecil ke besar.



Method print() tanpa parameter adalah metode bantu untuk mencetak struktur pohon secara visual, dimulai dari node saat ini (biasanya root). Fungsi ini memanggil versi print(String prefix, boolean isTail, String sb) dengan nilai awal prefix kosong, isTail bernilai true, dan string kosong sb. Meskipun print() ini secara teknis mengembalikan String, pada implementasinya ia lebih menekankan pada output langsung ke konsol melalui System.out.println() daripada membangun string dalam sb.

Method print(String prefix, boolean isTail, String sb) bertugas membentuk tampilan visual dari tree menggunakan karakter khusus seperti \-- dan | untuk menggambarkan level dan arah cabang. Secara logika, method ini bekerja dari kanan ke kiri: pertama dia memanggil dirinya sendiri ke anak kanan, lalu mencetak node saat ini, lalu memanggil dirinya ke anak kiri. Parameter prefix berfungsi sebagai indentasi supaya struktur kelihatan bertingkat. isTail menandakan apakah node saat ini adalah anak terakhir (biasanya anak kiri), agar bisa menyesuaikan karakter visual yang dipakai.

**2. Class BTree**

Class BTree memiliki dua atribut utama, yaitu root dan currentNode, keduanya bertipe Node. Atribut root merepresentasikan akar dari pohon biner, yaitu titik awal atau simpul utama dari seluruh struktur pohon. Semua traversal dan operasi pada pohon dimulai dari node ini. Sedangkan currentNode berfungsi sebagai penanda simpul yang sedang aktif atau difokuskan saat ini. Meskipun tidak selalu digunakan dalam semua operasi pohon, currentNode bisa sangat berguna dalam konteks interaktif, seperti saat membuat antarmuka pengguna, simulasi pohon, atau operasi berbasis navigasi manual.

Constructor BTree() digunakan untuk membuat objek BTree baru dengan menginisialisasi akar pohon sebagai null. Ini berarti saat pohon pertama kali dibuat, belum ada node di dalamnya. Penetapan awal ini penting untuk memastikan bahwa pohon tidak langsung memiliki isi sebelum ada data yang dimasukkan secara eksplisit, dan juga jadi dasar untuk menentukan apakah pohon masih kosong dengan method isEmpty().

A computer code with colorful text

AI-generated content may be incorrect.Method search(int data) adalah pintu masuk untuk melakukan pencarian data dalam pohon. Method ini tidak langsung melakukan pencarian, tapi meneruskannya ke method private search(Node node, int data) dengan memberikan root sebagai titik awal. Tujuannya adalah agar pemanggilan dari luar class tetap sederhana, cukup dengan menyebut data yang ingin dicari tanpa perlu tahu struktur internal pohonnya.

Sementara itu, method search(Node node, int data) melakukan pencarian secara rekursif dengan cara traversal preorder (kunjungi node sekarang, lalu anak kiri, lalu anak kanan). Pertama-tama, dia membandingkan data di node saat ini dengan nilai yang dicari. Kalau sama, langsung mengembalikan true. Kalau tidak, dia akan memeriksa anak kiri dengan cara memanggil dirinya sendiri secara rekursif. Jika data belum ditemukan, dia lanjut memeriksa anak kanan. Kalau dari semua jalur data tidak ditemukan, maka false akan dikembalikan.

A screen shot of a computer program

AI-generated content may be incorrect.method printInorder(), printPreOrder(), dan printPostOrder() merupakan interface publik untuk mencetak isi pohon menggunakan tiga jenis traversal yang berbeda. Ketiganya bekerja dengan cara memanggil method traversal yang sudah didefinisikan sebelumnya di class Node, yaitu printInorder(Node node), printPreorder(Node node), dan printPostorder(Node node), dengan root sebagai parameter awal.

printInorder() akan mencetak node secara urut: anak kiri → node saat ini → anak kanan. Ini cocok jika ingin melihat data dalam urutan yang terurut (khususnya kalau pohon berupa Binary Search Tree). printPreOrder() mencetak dengan urutan node saat ini → anak kiri → anak kanan, cocok untuk membaca struktur dari atas ke bawah. Sedangkan printPostOrder() mencetak dalam urutan anak kiri → anak kanan → node saat ini, biasanya digunakan dalam operasi yang sifatnya bottom-up, seperti penghapusan node. Dengan menyediakan tiga method ini secara terpisah, pengguna class bisa memilih traversal sesuai kebutuhan tanpa harus tahu detail teknis cara kerja masing-masing.

A computer screen with text

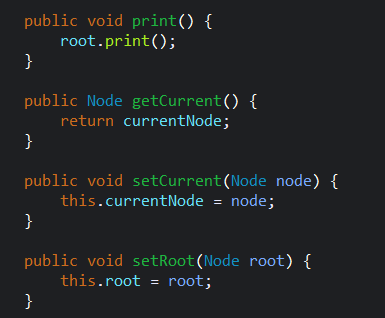
AI-generated content may be incorrect.Method getRoot() berfungsi untuk mengambil node akar dari pohon, yaitu node yang menjadi titik awal semua cabang. Dengan menyediakan akses ke root, method ini memungkinkan bagian lain dari program untuk membaca atau memanipulasi struktur pohon secara langsung, misalnya untuk menambahkan anak atau mengatur ulang posisi node.

Sementara itu, method isEmpty() digunakan untuk mengecek apakah pohon masih kosong atau tidak. Logikanya sangat sederhana: jika root masih null, maka artinya pohon belum memiliki node sama sekali, dan method ini akan mengembalikan true. Kalau sudah ada node (meskipun hanya satu), maka hasilnya adalah false. Method ini berguna sebagai kondisi awal sebelum melakukan operasi yang bergantung pada keberadaan isi pohon, seperti pencarian, penambahan, atau pencetakan struktur.

A computer code on a black background

AI-generated content may be incorrect.Method countNodes() digunakan untuk menghitung jumlah total node yang ada di dalam pohon. Sama seperti method pencarian sebelumnya, method ini hanya bertugas sebagai interface publik yang memanggil method countNodes(Node node) secara rekursif, dengan memberikan root sebagai titik awal perhitungan.

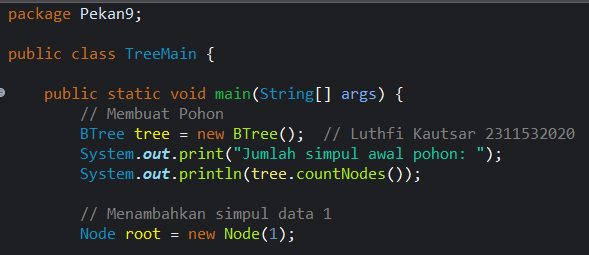
Di dalam countNodes(Node node), perhitungan dilakukan secara rekursif dengan menjelajahi seluruh pohon. Kalau node saat ini adalah null, maka akan langsung mengembalikan nilai 0 yang mana adalah kondisi dasar dari rekursi. Jika node tidak null, maka perhitungan dimulai dari 1 (untuk node itu sendiri), lalu ditambahkan dengan hasil dari menghitung node di anak kirinya (countNodes(node.getLeft())) dan anak kanannya (countNodes(node.getRight())). Dengan pendekatan ini, semua node dalam pohon akan dijelajahi dan dihitung, terlepas dari urutan atau bentuk struktur pohonnya. Hasil akhirnya adalah jumlah total node yang dimiliki oleh pohon biner tersebut.

Method print() berfungsi untuk menampilkan struktur pohon secara visual langsung di konsol. Ia memanggil method print() milik objek root, yang sebelumnya sudah dijelaskan pada class Node. Hasilnya akan berupa tampilan hierarki pohon menggunakan simbol seperti \-- dan indentasi, sehingga bentuk cabang-cabang pohon bisa terlihat lebih jelas. Ini sangat berguna untuk memahami struktur dan posisi tiap node dalam pohon tanpa harus melihat data satu per satu lewat traversal biasa.

Method getCurrent() dan setCurrent(Node node) digunakan untuk mengakses dan menetapkan nilai dari atribut currentNode. getCurrent() akan mengembalikan node yang sedang aktif atau terfokus, sedangkan setCurrent() memungkinkan kamu menetapkan node tertentu sebagai yang sedang difokuskan.

Method setRoot(Node root) digunakan untuk menetapkan akar pohon secara eksplisit yang mana bisa dipakai kalau membangun tree secara manual, misalnya dengan membuat node-node satu per satu lalu menyusunnya dari luar class BTree. Dengan method ini, kita bisa mengatur node mana yang jadi titik awal pohon, menggantinya, atau membangun ulang tree dari struktur yang sudah ada sebelumnya.

Bottom of Form

**3. Class MainTree**

Program dimulai dengan membuat objek dari class BTree menggunakan BTree tree = new BTree();. Ini berarti sebuah pohon kosong telah dibuat, di mana root-nya masih null. Setelah itu, program mencetak jumlah simpul awal dari pohon menggunakan tree.countNodes(). Karena pohon masih kosong, maka yang dicetak adalah angka 0.

Setelah itu, sebuah simpul baru dibuat menggunakan Node root = new Node(1);, yang berarti node dengan data bernilai 1 telah dibuat, namun belum dimasukkan ke dalam pohon. Supaya node tersebut menjadi bagian dari pohon, dipanggillah tree.setRoot(root);, yang menetapkan node 1 sebagai root dari pohon. Setelah itu, program kembali mencetak jumlah simpul dalam pohon

A screen shot of a computer program

AI-generated content may be incorrect.Pada bagian ini, program membuat enam node tambahan yang masing-masing menyimpan data dari 2 hingga 7: node2, node3, node4, node5, node6, dan node7. Masing-masing node dibuat secara terpisah menggunakan konstruktor Node(int data), yang artinya setiap node langsung memiliki data integer yang spesifik dan belum terhubung ke node lainnya. Meskipun sudah dibuat, keenam node ini masih berdiri sendiri-sendiri dan belum memiliki hubungan sebagai anak atau induk.

A screenshot of a computer program

AI-generated content may be incorrect.Setelah semua node dibuat, bagian ini menyusun struktur pohon dengan menentukan hubungan antar simpul menggunakan method setLeft() dan setRight(). Pertama, root.setLeft(node2); menyambungkan node 2 sebagai anak kiri dari root (node 1), lalu node2.setLeft(node4); dan node2.setRight(node5); menjadikan node 4 dan 5 sebagai anak kiri dan kanan dari node 2. Selanjutnya, node3.setLeft(node6); mengatur node 6 sebagai anak kiri dari node 3, dan node3.setRight(node7); menjadikan node 7 sebagai anak kanannya. Terakhir, root.setRight(node3); menyambungkan node 3 sebagai anak kanan dari root. Hasil dari proses ini adalah terbentuknya pohon biner yang strukturnya simetris, dengan node 1 sebagai akar dan node-node lain terhubung sesuai dengan posisinya dalam pohon.

A screen shot of a computer code

AI-generated content may be incorrect.Pada bagian ini, program menetapkan kembali node root sebagai node yang sedang aktif atau difokuskan dengan memanggil tree.setCurrent(tree.getRoot());. Ini berarti currentNode dalam objek tree sekarang menunjuk ke node 1 (akar pohon). Langkah ini sebenarnya tidak mengubah struktur pohon apa pun, tapi bisa berguna untuk simulasi, navigasi interaktif, atau sekadar pelacakan posisi saat ini dalam pohon. Setelah itu, program mencetak data dari simpul yang sedang aktif dengan tree.getCurrent().getData();, yang hasilnya tentu adalah angka 1 karena node 1 adalah root dan baru saja ditetapkan sebagai currentNode.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.Setelah struktur pohon selesai dibentuk, program mencetak jumlah simpul yang ada di pohon menggunakan tree.countNodes();. Karena seluruh node dari 1 hingga 7 sudah ditambahkan dan terhubung satu sama lain, maka hasilnya adalah 7. Ini menunjukkan bahwa pohon sekarang sudah lengkap dan terdiri dari tujuh simpul yang saling terhubung dengan struktur bercabang.

Langkah selanjutnya adalah menampilkan isi pohon dengan tiga jenis traversal: Inorder, Preorder, dan Postorder. Masing-masing traversal dilakukan dengan cara memanggil method yang sesuai: tree.printInorder();, tree.printPreOrder();, dan tree.printPostOrder();. Setiap metode akan mencetak data simpul berdasarkan urutan yang berbeda. Inorder mencetak anak kiri dulu, lalu simpul saat ini, baru anak kanan; Preorder mencetak simpul saat ini dulu, lalu anak kiri dan kanan; sementara Postorder mencetak anak kiri dan kanan dulu, baru simpul saat ini.

A screen shot of a computer code

AI-generated content may be incorrect.**4. Class GraphTraversal**

Struktur data utama yang digunakan dalam program ini adalah Map<String, List<String>>, yang merepresentasikan graf dalam bentuk adjacency list. Di sini, String berperan sebagai nama simpul (node), dan List<String> menyimpan semua tetangga atau node yang terhubung langsung dengan node tersebut. Misalnya, jika node "A" terhubung ke "B" dan "C", maka dalam map-nya akan disimpan sebagai "A" -> ["B", "C"]. Pemilihan struktur ini sangat efisien untuk menyimpan dan menelusuri graf yang tidak terlalu padat (sparse graph), karena hanya mencatat koneksi yang benar-benar ada, tanpa membuang memori untuk koneksi yang tidak ada. Selain itu, Map memungkinkan pencarian node secara cepat berdasarkan nama, sementara List menyimpan tetangganya dalam urutan penambahan, yang akan memengaruhi urutan saat penelusuran DFS atau BFS dilakukan.

A screen shot of a computer code

AI-generated content may be incorrect.Method addEdge() digunakan untuk menambahkan sisi (edge) antara dua node dalam graf yang tidak berarah. Artinya, jika node A terhubung ke B, maka secara otomatis B juga dianggap terhubung ke A. Di dalam method ini, pertama-tama dicek apakah masing-masing node (node1 dan node2) sudah ada di dalam graph. Jika belum, maka akan ditambahkan ke dalam map dengan nilai awal berupa list kosong (new ArrayList<>()). Setelah dipastikan kedua node ada, langkah berikutnya adalah menambahkan node2 ke dalam daftar tetangga node1, dan sebaliknya, menambahkan node1 ke dalam daftar tetangga node2. Dengan demikian, hubungan dua arah antar simpul bisa terbentuk secara otomatis.

A computer screen with text

AI-generated content may be incorrect.Method printGraph() digunakan untuk menampilkan isi dari graf dalam bentuk adjacency list, yaitu daftar yang menunjukkan node-node mana saja yang terhubung ke setiap simpul. Prosesnya dimulai dengan mencetak judul “Graf Awal (Adjacency List):” agar output mudah dikenali. Setelah itu, dilakukan iterasi terhadap setiap kunci (node) dalam graph, lalu dicetak node tersebut diikuti oleh tanda panah (->) dan daftar tetangganya, yang diambil dari graph.get(node). List tetangga ini kemudian digabungkan menjadi satu baris string menggunakan String.join(", ", neighbors) supaya tampil rapi dan mudah dibaca.

A computer screen shot of colorful text

AI-generated content may be incorrect.Method dfs() berfungsi sebagai pintu masuk untuk melakukan penelusuran graf menggunakan algoritma Depth-First Search (DFS) secara rekursif. Penelusuran dimulai dari node yang diberikan sebagai parameter start. Pertama-tama, method ini membuat struktur Set<String> visited untuk mencatat simpul-simpul yang sudah dikunjungi agar tidak dikunjungi berulang. Setelah itu, pemanggilan rekursif dimulai dengan memanggil dfsHelper().

Method dfsHelper() adalah inti dari algoritma DFS rekursif. Ia menerima dua parameter: current (simpul yang sedang dikunjungi) dan visited (set simpul yang sudah dikunjungi sejauh ini). Jika simpul saat ini sudah pernah dikunjungi (dicek dengan visited.contains(current)), maka fungsi langsung mengembalikan tanpa melakukan apa-apa. Jika belum, maka simpul dimasukkan ke dalam visited dan dicetak ke layar. Setelah itu, method ini akan memanggil dirinya sendiri untuk setiap tetangga dari simpul saat ini (diambil dari graph.getOrDefault(current, new ArrayList<>())), selama tetangga tersebut belum dikunjungi.

A computer screen shot of text

AI-generated content may be incorrect.Method bfs() digunakan untuk melakukan penelusuran graf menggunakan algoritma Breadth-First Search (BFS) secara iteratif. Penelusuran dimulai dari simpul yang diberikan sebagai parameter start. Pertama, dibuat struktur Set<String> visited untuk melacak simpul yang sudah dikunjungi, serta Queue<String> queue untuk menyimpan simpul-simpul yang akan dikunjungi berikutnya. Simpul awal langsung dimasukkan ke queue dan ke dalam visited.

Selama queue belum kosong, program terus melakukan perulangan: mengambil simpul paling depan dari queue menggunakan poll(), mencetak simpul tersebut, lalu memeriksa semua tetangganya. Jika ada tetangga yang belum dikunjungi, maka tetangga itu dimasukkan ke queue dan ditandai sebagai sudah dikunjungi. Proses ini t erus berlanjut hingga semua simpul yang terhubung dari simpul awal telah dijelajahi.

A screen shot of a computer program

AI-generated content may be incorrect.Dalam method main(), program pertama-tama membuat objek graf baru dengan GraphTraversal graph = new GraphTraversal();. Setelah itu, graf dibentuk dengan menambahkan beberapa edge: A-B, A-C, B-D, dan B-E, menggunakan method addEdge(). Karena graf ini tidak berarah, setiap edge akan otomatis ditambahkan dua arah (misalnya A terhubung ke B, dan B juga ke A).

A screenshot of a computer program

AI-generated content may be incorrect.Setelah graf selesai dibuat, method printGraph() dipanggil untuk menampilkan struktur graf dalam bentuk adjacency list, agar kita bisa melihat hubungan antar simpul sebelum dilakukan penelusuran. Kemudian, program melakukan penelusuran graf dua kali: pertama menggunakan Depth-First Search (DFS) dengan graph.dfs("A"), lalu menggunakan Breadth-First Search (BFS) dengan graph.bfs("A"). Keduanya dimulai dari simpul "A", dan hasilnya akan menunjukkan urutan simpul yang dikunjungi sesuai dengan strategi penelusuran masing-masing.