**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

****

**DISUSUN OLEH :**

**MUHAMMAD NURIL ANWARI**

**362458302106**

**TEKNOLOGI REKAYASA PERANGKAT LUNAK**

**POLITEKNIK NEGERI BANYUWANGI**

**TAHUN 2025**

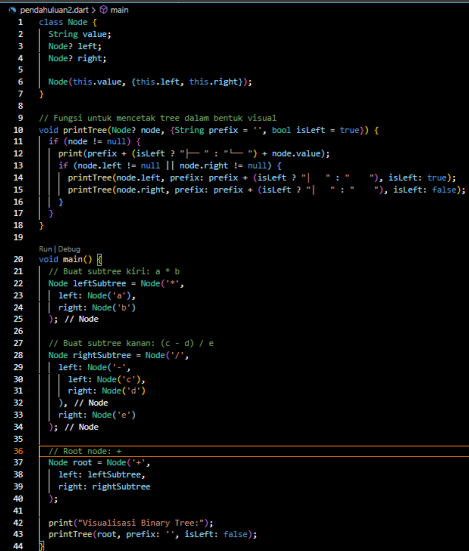
**BINARY TREE**

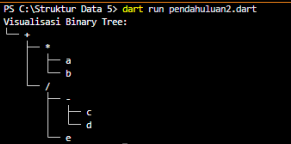
**TUGAS PENDAHULUAN**

**I.** Jelaskan istilah-istilah: Binary Tree, Root, Parent, Child, Sibling, Leaf Node, Level Node, dan Kedalaman Binary Tree

1. Binary Tree Binary Tree adalah struktur data pohon di mana setiap node maksimal memiliki dua anak (child), yaitu: • anak kiri (left child) • anak kanan (right child) Struktur ini sering digunakan dalam berbagai aplikasi seperti pencarian data, ekspresi matematika, dan lainnya.
2. Root Root adalah node paling atas dalam pohon. • Merupakan awal dari seluruh struktur tree. • Tidak memiliki parent.
3. Parent Parent adalah node yang memiliki anak (child). • Node ini terhubung langsung ke satu atau dua node di bawahnya.
4. Child Child adalah node yang berasal atau diturunkan dari node lain (parent). • Bisa berupa left child atau right child. • Setiap node hanya memiliki satu parent (kecuali root, yang tidak punya parent).
5. SiblingSibling adalah node-node yang memiliki parent yang sama. • Contoh: jika dua node adalah anak dari node yang sama, maka mereka adalah saudara (siblings).
6. Leaf Node Leaf Node adalah node yang tidak memiliki anak sama sekali. • Letaknya biasanya di bagian paling bawah dari pohon. • Leaf node sering disebut juga daun pohon.
7. Level Node Level Node mengacu pada tingkatan atau kedalaman sebuah node dalam pohon: • Root berada di level 0. • Anak dari root di level 1, dan seterusnya. • Menunjukkan jarak vertikal dari root ke node tersebut.
8. Kedalaman Binary Tree (Depth) Kedalaman binary tree adalah panjang jalur terpanjang dari root ke leaf node. • Atau dapat dikatakan sebagai level maksimal dalam pohon tersebut. • Depth bisa dihitung dengan jumlah level + 1, tergantung definisinya

**II.** Terdapat Binary Tree yang merepresentasikan sebuah notasi aritmatika a \* b + (c – d) / e Gambar 8. Binary Tree dengan notasi aritmatika a \* b + (c – d) / e Buatlah digram rekursif untuk pembacaan Binary Tree menggunakan Traversal PostOrder

****

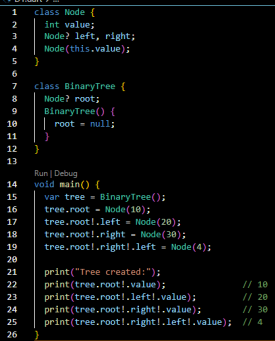
****

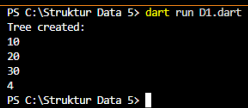
Kiri dari +: - Kiri dari \*: a → print a - Kanan dari \*: b → print b - print \*

Kanan dari +: - Kiri dari /: - Kiri dari -: c → print c - Kanan dari -: d → print d - print - - Kanan dari /: e → print e - print /

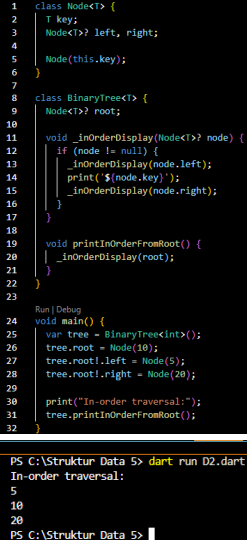
**PERCOBAAN**

1. Membangun Binary Tree seperti gambar 9 secara manual

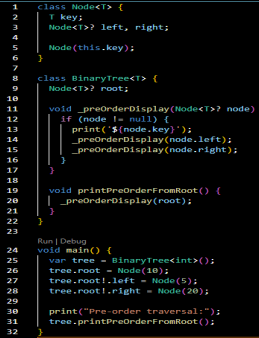
****

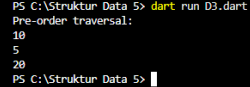
****

1. Pembacaan Binary Tree dengan Traversal Inorder

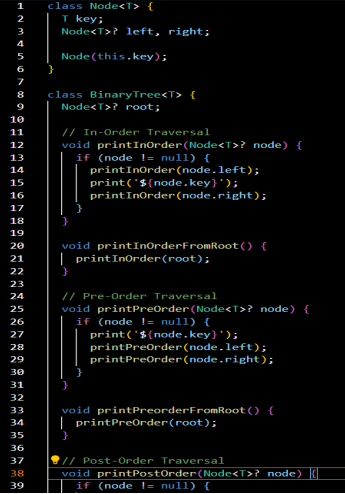
****

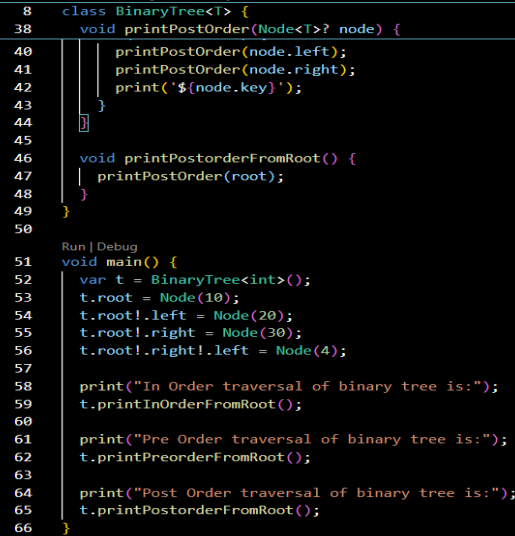
1. Pembacaan Binary Tree dengan Traversal Preorder

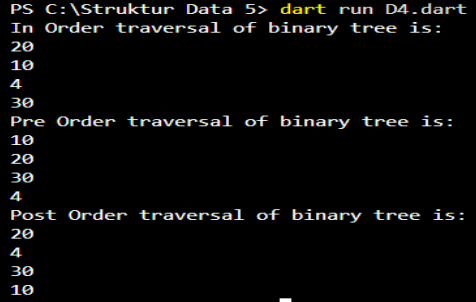
****

****

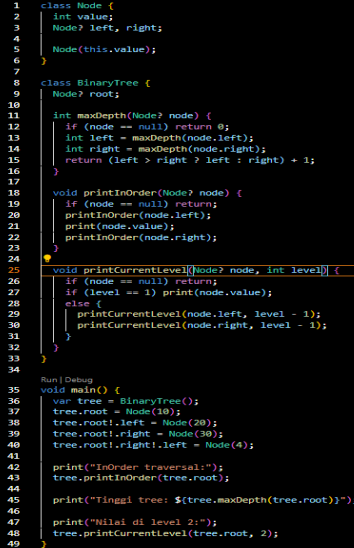
1. Pembacaan Binary Tree dengan Traversal Postorder

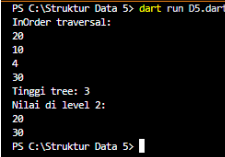
****

****

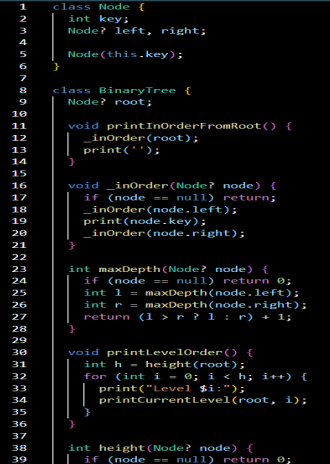
****

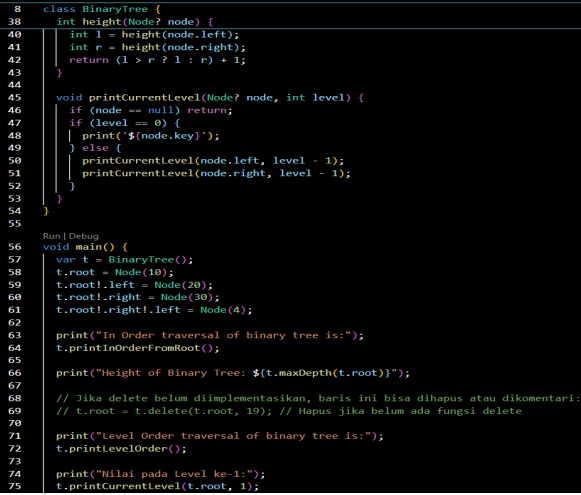
1. Untuk mendapatkan kedalaman dari Binary Tree

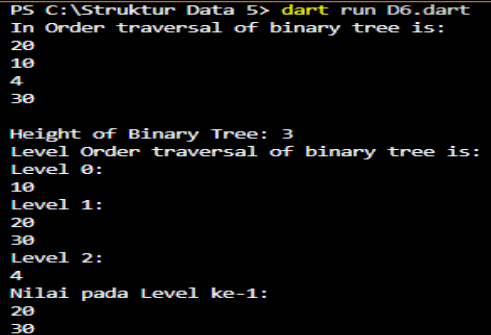
****

****

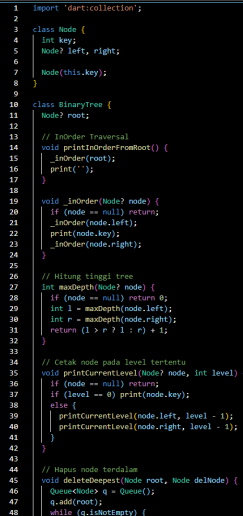
1. Untuk Mencetak Level Order Traversal Binary Tree

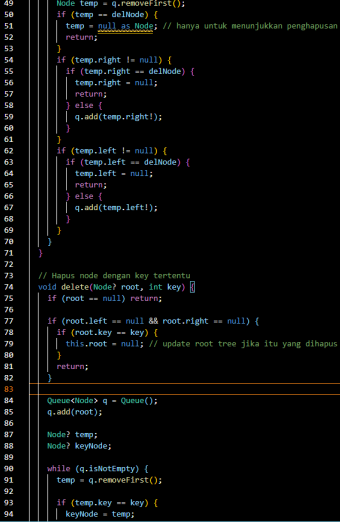
****

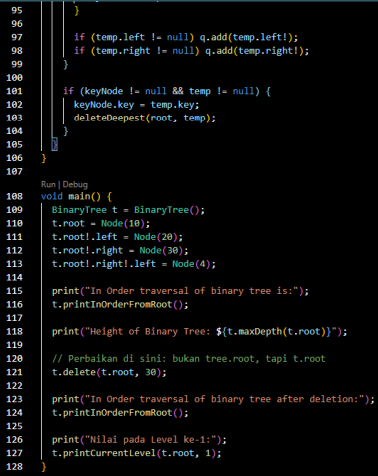
****

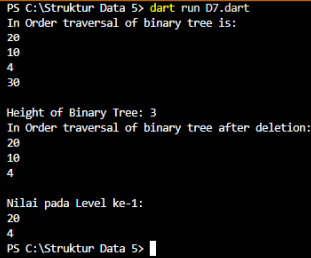
****

1. Untuk menghapus Binary Tree

****

****

****

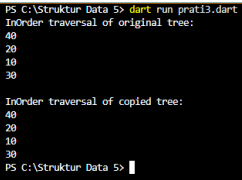
****

**LATIHAN**

**No. 3**

Buatlah method copyTree(TNode? t). Dan buatlah Class void main untuk menguji method tersebut

****

****

Method copyTree(t) adalah sebuah fungsi yang digunakan untuk menyalin seluruh isi pohon biner (binary tree) secara rekursif. Artinya, setiap node dalam pohon asli akan dibuatkan salinan baru (bukan hanya menyalin alamat atau referensinya), sehingga pohon hasil salinan benar-benar terpisah dari pohon aslinya. Fungsi ini bekerja dengan cara: jika node saat ini kosong (null), maka dikembalikan null juga. Tetapi jika node ada isinya, maka dibuat node baru dengan nilai yang sama, lalu memanggil kembali copyTree untuk anak kiri dan anak kanannya. Dengan cara ini, seluruh struktur pohon disalin dari atas hingga ke bawah. Di dalam main(), kita membuat sebuah pohon biner awal (originalTree) dan mengisinya dengan beberapa node. Setelah itu, kita memanggil copyTree() untuk menyalin pohon tersebut ke dalam copiedTree. Kita bisa melihat bahwa hasil in-order traversal dari kedua pohon ini sama persis, yang menunjukkan bahwa isi pohon telah berhasil disalin. Karena setiap node yang disalin dibuat baru, perubahan pada salah satu pohon tidak akan memengaruhi pohon lainnya. Inilah yang disebut sebagai deep copy, bukan shallow copy. Dengan cara ini, kita bisa menyalin pohon tanpa khawatir terjadi konflik data antara salinan dan aslinya.

**BINARY SEARCH TREE**

**TUGAS PENDAHULUAN**

1. Buatlah resume 1 halaman mengenai Binary Search Tree

Binary Search Tree adalah salah satu struktur data pohon biner yang digunakan secara luas dalam ilmu komputer untuk menyimpan data secara terstruktur agar pencarian, penyisipan, dan penghapusan data dapat dilakukan secara efisien. Ciri khas dari BST adalah setiap node hanya memiliki maksimal dua anak, yaitu anak kiri dan anak kanan. Nilai pada anak kiri selalu lebih kecil dari nilai induknya, sedangkan nilai pada anak kanan selalu lebih besar. Struktur ini memungkinkan operasi pencarian dilakukan lebih cepat dibandingkan dengan pencarian linier, terutama jika pohon seimbang. Operasi dasar dalam BST mencakup pencarian (search), penyisipan (insert), dan penghapusan (delete). Ketiga operasi ini, dalam kondisi ideal saat pohon seimbang, memiliki kompleksitas waktu rata-rata O(log n). Namun, pada kasus terburuk (misalnya, jika data yang dimasukkan secara berurutan), BST dapat berubah menjadi struktur linear seperti linked list, dan kompleksitasnya menjadi O(n). Oleh karena itu, penting untuk menjaga keseimbangan pohon, misalnya dengan menggunakan variasi seperti AVL Tree atau Red-Black Tree. BST juga mendukung tiga jenis penelusuran (traversal), yaitu inorder, preorder, dan postorder. Traversal inorder digunakan untuk menghasilkan data secara urut, traversal preorder berguna dalam proses duplikasi atau ekspor struktur pohon, dan traversal postorder cocok digunakan ketika semua node perlu dihapus. Implementasi BST dapat dilakukan dalam berbagai bahasa pemrograman, dan umumnya cukup sederhana, seperti menggunakan rekursi dalam Python untuk menyisipkan node baru. Beberapa keunggulan BST adalah kemampuannya dalam menyimpan data secara terstruktur, efisiensi dalam pencarian, dan mendukung traversal secara terurut. Namun, kelemahannya terletak pada potensi ketidakseimbangan yang dapat memperlambat kinerja. BST digunakan dalam berbagai aplikasi seperti sistem indeks pada basis data, algoritma kompresi data seperti Huffman coding, serta pengurutan dan pencarian data yang efisien.

1. Buatlah Binary Search Tree dari node-node 30, 15,17, 36, 67, 30, 69

Berikut adalah Binary Search Tree yang dibentuk dari urutan penyisipan node: 30, 15, 17, 36, 67, 30, 69.

Catatan: Dalam BST standar, data tidak boleh duplikat. Namun jika 30 muncul dua kali seperti di sini, kita bisa memilih salah satu strategi penanganan,

misalnya:

• Menolak duplikat (standar BST)

• Menyimpan duplikat di anak kanan

• Menggunakan counter di node Untuk contoh ini, kita akan menyimpan duplikat di anak kanan agar tetap bisa divisualisasikan dalam bentuk BST.

Langkah-langkah membentuk BST :

1. 30 → menjadi root

2. 15 < 30 → masuk ke kiri dari 30

3. 17 > 15 → masuk ke kanan dari 15

4. 36 > 30 → masuk ke kanan dari 30

5. 67 > 36 → masuk ke kanan dari 36

6. 30 (duplikat) = 30 → masuk ke kanan dari 30 (asumsi simpan duplikat di kanan)

7. 69 > 67 → masuk ke kanan dari 67

Struktur BST-nya

30

/ \

15 36

\ \

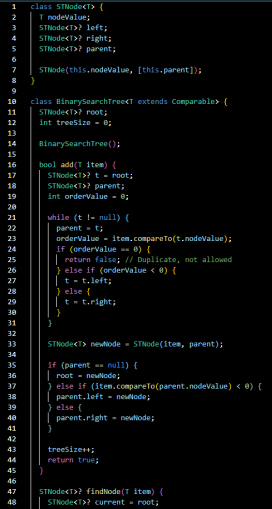
17 67

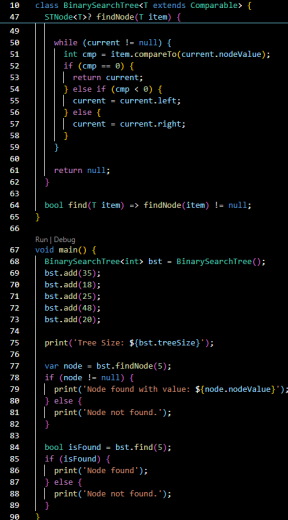
/ \

30 69

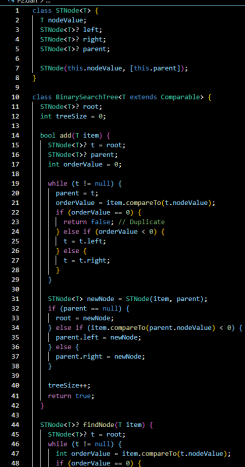
**PERCOBAAN**

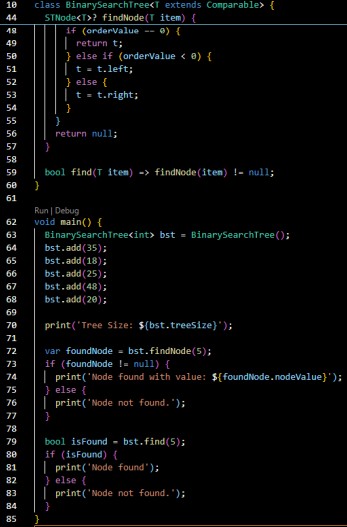
1. Membuat class BinarySearchTree

****

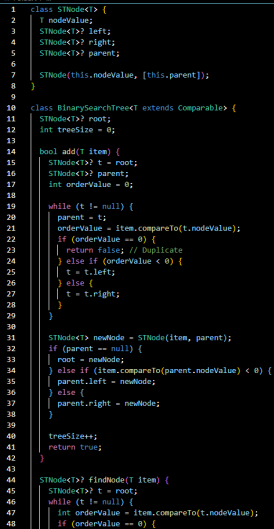
****

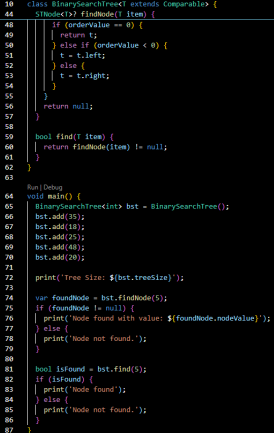
1. Membuat method findNode()

****

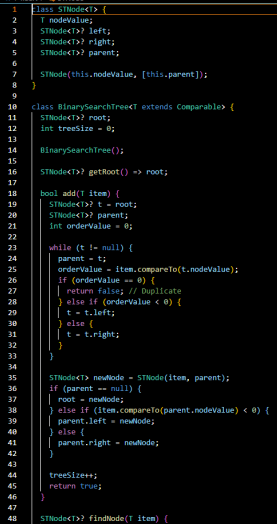
****

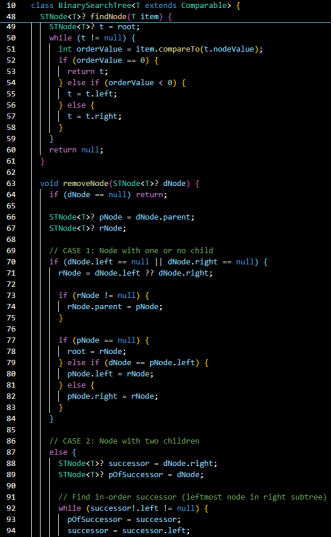
1. Membuat method find()

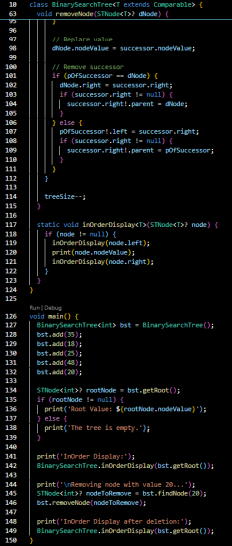
****

****

1. Method untuk menghapus sebuah Node

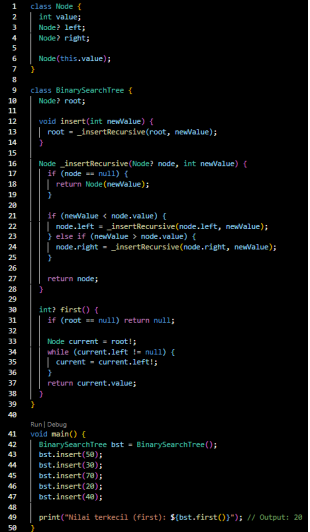
****

****

****

**LATIHAN**

Buatlah method first() untuk mendapatkan nilai terkecil dari node-node yang terdapat pada Binary search tree

****

**C:\Users\user\Pictures\Screenshots\Screenshot (469).png**

Method first() pada Binary Search Tree (BST) berfungsi untuk mendapatkan nilai terkecil dari seluruh node yang ada di dalam pohon tersebut. Dalam struktur BST, setiap node memiliki aturan bahwa nilai pada child kiri (left) selalu lebih kecil daripada node induknya, dan nilai pada child kanan (right) selalu lebih besar. Oleh karena itu, untuk menemukan nilai terkecil, kita cukup mengikuti jalur dari root ke arah kiri secara terus-menerus hingga mencapai node paling kiri yang tidak memiliki child kiri lagi. Node ini adalah node dengan nilai terkecil di seluruh pohon. Implementasi first() dalam Dart dilakukan dengan cara memeriksa apakah root kosong, kemudian menggunakan perulangan untuk bergerak ke child kiri sampai null ditemukan. Nilai dari node terakhir sebelum null itulah yang dikembalikan sebagai hasil dari method first(). Pendekatan ini efisien karena hanya menjelajahi sebagian kecil dari pohon, yaitu hanya node-node sebelah kiri, tanpa perlu melakukan traversal seluruh pohon.