Laporan Hasil Praktikum

Algoritma Dan Struktur Data

Jobsheet 14



Angel Chelssa Leoniy Eka Permatasari

244107020202

1E

Program Studi Teknologi Informasi

Jurusan Teknik Informatika

POLINEMA

2025

**PERCOBAAN 1**

1. Di dalam class Mahasiswa03, deklarasikan atribut sesuai dengan diagram class Mahasiswa di atas. Tambahkan juga konstruktor dan method sesuai diagram di atas.
2. Di dalam class Node03, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.
3. Di dalam class BinaryTree03, tambahkan atribut root.
4. Tambahkan konstruktor dan method isEmpty() di dalam class BinaryTree03.
5. Tambahkan method add() di dalam class BinaryTree03. Node ditambahkan di binary search tree pada posisi sesuai dengan besar nilai IPK mahasiswa. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya dengan proses rekursif penulisan kode akan lebih efisien.
6. Tambahkan method find()
7. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam binary tree, baik dalam mode pre-order, in-order maupun post-order.
8. Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.
9. Tambahkan method delete(). Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak, cari posisi node yang akan dihapus.
10. Kemudian tambahkan proses penghapusan di dalam method delete() terhadap node current yang telah ditemukan.
11. Buka class BinaryTreeMain03 dan tambahkan method main() kemudian tambahkan kode berikut ini:

**KODE PROGRAM**

**KODE PROGRAM CLASS Mahasiswa03.Java**

|  |
| --- |
| public class Mahasiswa03 {      String nim;      String nama;      String kelas;      Double ipk;      public  Mahasiswa03() {      }      public Mahasiswa03(String *nim*, String *nama*, String *kelas*, Double *ipk*) {          this.nim = nim;          this.nama = nama;          this.kelas = kelas;          this.ipk = ipk;      }      public void tampilkanInformasi(){          System.out.println("NIM : " + this.nim + " " +          "Nama : " + this.nama + " " +          "Kelas : " + this.kelas + " " +          "IPK : " + this.ipk);      }  } |

**Kode program class Node03.java**

|  |
| --- |
| public class Node03 {      Mahasiswa03 mahasiswa;      Node03 left, right;      public Node03(){      }      public Node03(Mahasiswa03 *mahasiswa*){          this.mahasiswa = mahasiswa;          left = right = null;      }  } |

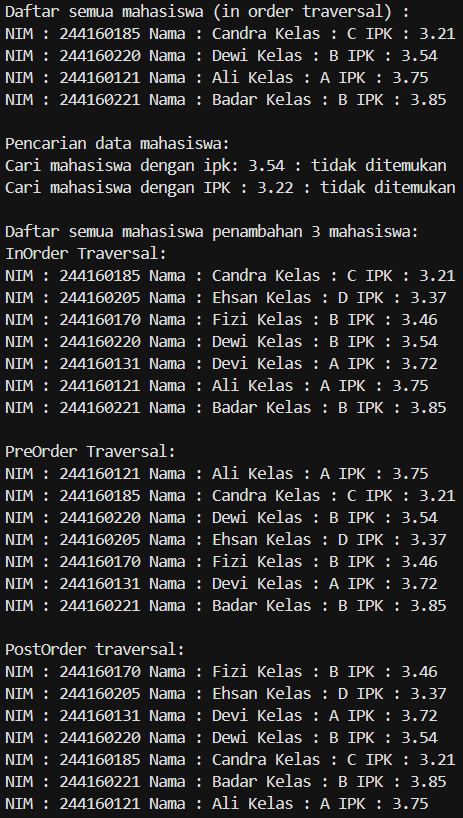
**KODE PROGRAM CLASS BinaryTree03.Java**

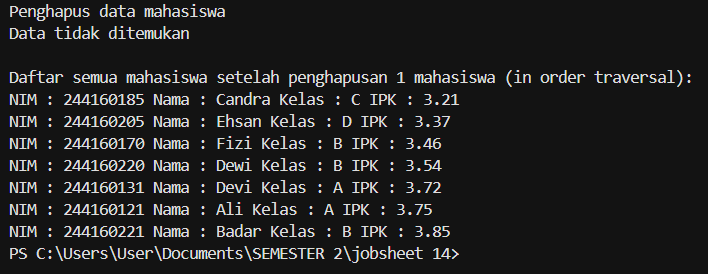
|  |
| --- |
| public class BinaryTree03 {      Node03 root;      public BinaryTree03() {          root = null;      }      public boolean isEmpty() {          return root == null;      }      public void add(Mahasiswa03 *mahasiswa*) {          Node03 newNode = new Node03(*mahasiswa*);          if (root == null) {              root = newNode;          } else {              Node03 current = root;              Node03 parent;              while (true) {                  parent = current;                  if (*mahasiswa*.ipk < current.mahasiswa.ipk) {                      current = current.left;                      if (current == null) {                          parent.left = newNode;                          return;                      }                  } else {                      current = current.right;                      if (current == null) {                          parent.right = newNode;                          return;                      }                  }              }          }      }      public boolean find(double *ipk*) {          Node03 current = root;          while (current != null) {              if (current.mahasiswa.ipk == *ipk*) {                  return true;              } else if (*ipk* > current.mahasiswa.ipk) {                  current = current.right;              } else {                  current = current.left;              }          }          return false;      }      public void traversePreOrder(Node03 *node*) {          if (*node* != null) {  *node*.mahasiswa.tampilkanInformasi();              traversePreOrder(*node*.left);              traversePreOrder(*node*.right);          }      }      public void traverseInOrder(Node03 *node*) {          if (*node* != null) {              traverseInOrder(*node*.left);  *node*.mahasiswa.tampilkanInformasi();              traverseInOrder(*node*.right);          }      }      public void traversePostOrder(Node03 *node*) {          if (*node* != null) {              traversePostOrder(*node*.left);              traversePostOrder(*node*.right);  *node*.mahasiswa.tampilkanInformasi();          }      }      public Node03 getSuccessor(Node03 *del*) {          Node03 successor = *del*.right;          Node03 successorParent = *del*;          while (successor.left != null) {              successorParent = successor;              successor = successor.left;          }          if (successor != *del*.right) {              successorParent.left = successor.right;              successor.right = *del*.right;          }          return successor;      }      public void delete(double *ipk*) {          if (isEmpty()) {              System.out.println("Binary tree kosong");              return;          }          Node03 parent = root;          Node03 current = root;          boolean isLeftChild = false;          while (current != null && current.mahasiswa.ipk != *ipk*) {              parent = current;              if (*ipk* < current.mahasiswa.ipk) {                  isLeftChild = true;                  current = current.left;              } else {                  isLeftChild = false;                  current = current.right;              }          }          if (current == null) {              System.out.println("Data tidak ditemukan");              return;          }          if (current.left == null && current.right == null) {              if (current == root) {                  root = null;              } else if (isLeftChild) {                  parent.left = null;              } else {                  parent.right = null;              }          }          else if (current.left == null) {              if (current == root) {                  root = current.right;              } else if (isLeftChild) {                  parent.left = current.right;              } else {                  parent.right = current.right;              }          }          else if (current.right == null) {              if (current == root) {                  root = current.left;              } else if (isLeftChild) {                  parent.left = current.left;              } else {                  parent.right = current.left;              }          }          else {              Node03 successor = getSuccessor(current);              System.out.println("Jika 2 anak, current = ");              successor.mahasiswa.tampilkanInformasi();              if (current == root) {                  root = successor;              } else if (isLeftChild) {                  parent.left = successor;              } else {                  parent.right = successor;              }              successor.left = current.left;          }      }  } |

**KODE PROGRAM CLASS BinaryTreeMain03.Java**

|  |
| --- |
| public class BInaryTreeMain03 {      public static void main(String[] *args*) {          BinaryTree03 bst = new BinaryTree03();          bst.add(new Mahasiswa03("244160121", "Ali", "A", 3.75));          bst.add(new Mahasiswa03("244160221", "Badar", "B", 3.85));          bst.add(new Mahasiswa03("244160185", "Candra", "C", 3.21));          bst.add(new Mahasiswa03("244160220", "Dewi", "B", 3.54));          System.out.println("Daftar semua mahasiswa (in order traversal) : ");          bst.traverseInOrder(bst.root);          System.out.println("\nPencarian data mahasiswa: ");          System.out.print("Cari mahasiswa dengan ipk: 3.54 : ");          String hasilCari = bst.find(5.4)?"Ditemukan " : "tidak ditemukan";          System.out.println(hasilCari);          System.out.print("Cari mahasiswa dengan IPK : 3.22 : ");          hasilCari = bst.find(3.22)?"Ditemukan" : "tidak ditemukan";          System.out.println(hasilCari);            bst.add(new Mahasiswa03("244160131", "Devi", "A", 3.72));          bst.add(new Mahasiswa03("244160205", "Ehsan", "D", 3.37));          bst.add(new Mahasiswa03("244160170", "Fizi", "B", 3.46));          System.out.println("\nDaftar semua mahasiswa penambahan 3 mahasiswa: ");          System.out.println("InOrder Traversal: ");          bst.traverseInOrder(bst.root);          System.out.println("\nPreOrder Traversal: ");          bst.traversePreOrder(bst.root);          System.out.println("\nPostOrder traversal: ");          bst.traversePostOrder(bst.root);          System.out.println("\nPenghapus data mahasiswa");          bst.delete(3.57);          System.out.println("\nDaftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal): ");          bst.traverseInOrder(bst.root);      }  } |

**HASIL KODE PROGRAM**

****

****

**PERTANYAAN**

1. Binary Search Tree (BST) lebih efisien dalam pencarian karena memiliki aturan bahwa nilai di cabang kiri selalu lebih kecil dari node induk, dan nilai di cabang kanan lebih besar. Dengan struktur ini, proses pencarian bisa langsung mengarah ke sisi yang relevan, mempercepat waktu pencarian hingga logaritmik (O(log n)) pada tree yang seimbang.
2. Atribut left dan right pada setiap node digunakan untuk menghubungkan node tersebut ke anak kirinya dan anak kanannya. Hal ini membentuk struktur pohon biner di mana setiap node bisa memiliki maksimal dua anak.
3. Jawaban:
   1. Atribut root berfungsi sebagai titik awal atau node utama dalam tree. Semua proses seperti pencarian, penambahan, maupun penghapusan data dimulai dari root.
   2. Ketika tree baru dibuat dan belum berisi data apapun, nilai dari root adalah null, menandakan bahwa tree masih kosong.
4. Pada saat tree masih belum memiliki node, node pertama yang ditambahkan akan langsung menjadi root, tanpa perlu dilakukan proses perbandingan terlebih dahulu.
5. Baris parent = current digunakan untuk menyimpan referensi node induk sebelum pindah ke anaknya. Proses ini digunakan untuk menentukan ke mana node baru akan ditempatkan ke kiri jika IPK lebih kecil, ke kanan jika IPK lebih besar. Jika ditemukan child yang kosong, maka node baru ditambahkan di sana sebagai anak dari parent.
6. Ketika menghapus node yang memiliki dua anak, pertama-tama kita mencari node pengganti yang disebut **successor**, yaitu node dengan nilai terkecil di subtree kanan. Setelah itu, node yang ingin dihapus diganti dengan successor tersebut. Method getSuccessor() digunakan untuk menemukan node tersebut dan mengatur ulang koneksi antar node agar struktur BST tetap valid.

**PERCOBAAN 2**

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2. Buatlah class BinaryTreeArray00 dan BinaryTreeArrayMain00. Ganti 00 dengan nomer absen Anda. Buat atribut data dan idxLast di dalam class
3. BinaryTreeArray00. Buat juga method populateData() dan traverseInOrder().
4. Kemudian dalam class BinaryTreeArrayMain00 buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method main().

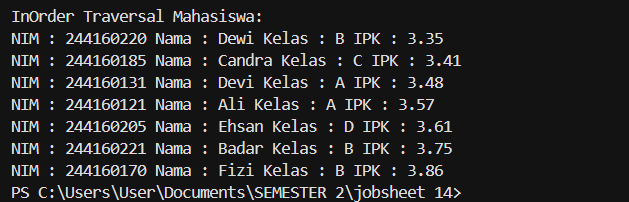
**KODE PROGRAM BinaryTreeArray03.java**

|  |
| --- |
| public class BinaryTreeArray03 {      Mahasiswa03[] dataMahasiswa03;      int idxLast;      public BinaryTreeArray03() {          this.dataMahasiswa03 = new Mahasiswa03[10];      }      void populateData (Mahasiswa03 *dataMhs*[], int *idxLast*){          this.dataMahasiswa03 = *dataMhs*;          this.idxLast = *idxLast*;      }      void traverseInOrder(int *idxStart*) {          if (*idxStart* <= idxLast) {              if (dataMahasiswa03[*idxStart*] != null) {                  traverseInOrder(2\**idxStart*+1);                  dataMahasiswa03[*idxStart*].tampilkanInformasi();                  traverseInOrder(2\**idxStart*+2);              }          }      }  } |

**KODE PROGRAM BinarryTreeMain03.Java**

|  |
| --- |
| public class BInaryTreeMain03 {      public static void main(String[] *args*) {          BinaryTree03 bst = new BinaryTree03();          bst.add(new Mahasiswa03("244160121", "Ali", "A", 3.75));          bst.add(new Mahasiswa03("244160221", "Badar", "B", 3.85));          bst.add(new Mahasiswa03("244160185", "Candra", "C", 3.21));          bst.add(new Mahasiswa03("244160220", "Dewi", "B", 3.54));          System.out.println("Daftar semua mahasiswa (in order traversal) : ");          bst.traverseInOrder(bst.root);          System.out.println("\nPencarian data mahasiswa: ");          System.out.print("Cari mahasiswa dengan ipk: 3.54 : ");          String hasilCari = bst.find(5.4)?"Ditemukan " : "tidak ditemukan";          System.out.println(hasilCari);          System.out.print("Cari mahasiswa dengan IPK : 3.22 : ");          hasilCari = bst.find(3.22)?"Ditemukan" : "tidak ditemukan";          System.out.println(hasilCari);            bst.add(new Mahasiswa03("244160131", "Devi", "A", 3.72));          bst.add(new Mahasiswa03("244160205", "Ehsan", "D", 3.37));          bst.add(new Mahasiswa03("244160170", "Fizi", "B", 3.46));          System.out.println("\nDaftar semua mahasiswa penambahan 3 mahasiswa: ");          System.out.println("InOrder Traversal: ");          bst.traverseInOrder(bst.root);          System.out.println("\nPreOrder Traversal: ");          bst.traversePreOrder(bst.root);          System.out.println("\nPostOrder traversal: ");          bst.traversePostOrder(bst.root);          System.out.println("\nPenghapus data mahasiswa");          bst.delete(3.57);          System.out.println("\nData semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal): ");          bst.traverseInOrder(bst.root);          BinaryTreeArray03 bta = new BinaryTreeArray03();          Mahasiswa03 mhs1= new Mahasiswa03("244160121", "Ali", "A", 3.57);          Mahasiswa03 mhs2= new Mahasiswa03("244160185", "Candra", "C", 3.41);          Mahasiswa03 mhs3= new Mahasiswa03("244160221", "Badar", "B", 3.75);          Mahasiswa03 mhs4= new Mahasiswa03("244160220", "Dewi", "B", 3.35);            Mahasiswa03 mhs5= new Mahasiswa03("244160131", "Devi", "A", 3.48);          Mahasiswa03 mhs6= new Mahasiswa03("244160205", "Ehsan", "D", 3.61);          Mahasiswa03 mhs7= new Mahasiswa03("244160170", "Fizi", "B", 3.86);          Mahasiswa03[] dataMahasiswa03 = {mhs1, mhs2, mhs3, mhs4, mhs5, mhs6, mhs7, null, null, null};          int idxLast = 6;          bta.populateData(dataMahasiswa03, idxLast);          System.out.println("\nInOrder Traversal Mahasiswa: ");          bta.traverseInOrder(0);      }  } |

**HASIL KODE**



**PERTANYAAN**

1. data: Merupakan array yang digunakan untuk menampung seluruh elemen dalam struktur pohon.

idxLast: Menyatakan indeks terakhir dalam array yang masih berisi data aktif atau valid.

1. Method ini digunakan untuk mengisi tree dengan elemen-elemen dari array yang telah disiapkan, sekaligus menetapkan nilai idxLast sebagai batas akhir data yang sah.
2. Method ini berfungsi untuk menelusuri semua node dalam pohon secara in-order, yaitu dimulai dari anak kiri, kemudian node utama, lalu ke anak kanan, dengan pendekatan berbasis array.
3. Jika suatu node berada pada indeks ke-2 dalam array, maka:

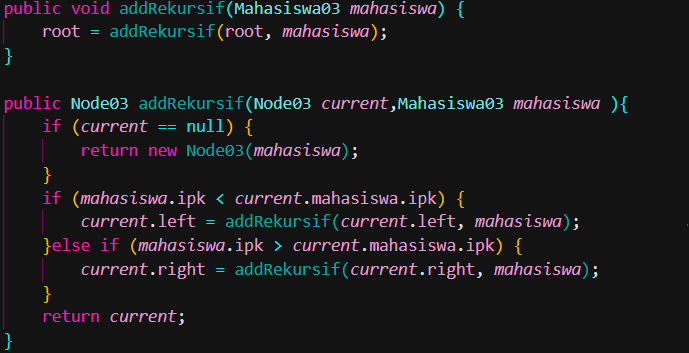
Anak kiri berada pada indeks 2 × 2 + 1 = 5

Anak kanan berada pada indeks 2 × 2 + 2 = 6

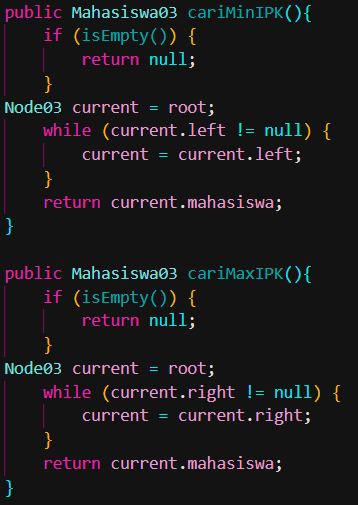
1. Nilai idxLast = 6 menandakan bahwa elemen yang digunakan atau diisi dalam array hanya sampai indeks ke-6 (total 7 data). Sedangkan elemen setelahnya (indeks 7, 8, 9) dianggap kosong atau tidak terpakai.
2. Rumus 2 \* idx + 1 untuk anak kiri dan 2 \* idx + 2 untuk anak kanan adalah cara umum untuk menentukan posisi anak dari node tertentu dalam array. Rumus ini mendukung penyimpanan pohon biner secara berurutan dan efisien dalam bentuk array, terutama pada pohon biner lengkap.

**TUGAS PRAKTIKUM**

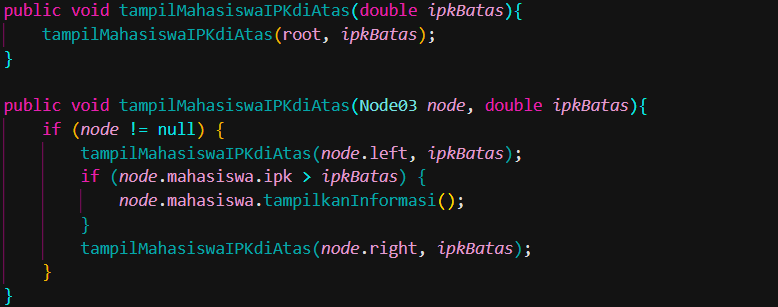
1. Buat method di dalam class BinaryTree00 yang akan menambahkan node dengan cara rekursif (addRekursif()).



1. Buat method di dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (cariMinIPK() dan cariMaxIPK()) yang ada di dalam binary search tree.



1. Buat method dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK di atas suatu batas tertentu, misal di atas 3.50 (tampilMahasiswaIPKdiAtas(double ipkBatas)) yang ada di dalam binary search tree.



1. Modifikasi class BinaryTreeArray00 di atas, dan tambahkan :

method add(Mahasiswa data) untuk memasukan data ke dalam binary tree

method traversePreOrder()

