Laporan Praktikum STRUKTUR DATA
MODUL IX TREE

Disusun Oleh :

Dwi Candra Pratama/2211104035 SE 07 02

Asisten Praktikum :

Aldi Putra

Andini Nur Hidayah

Dosen Pengampu :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA TELKOM UNIVERSITY PURWOKERTO

2024

A. GUIDED

1. Tujuan Praktikum
   a. Memahami konsep penggunaan fungsi rekursif.
   b. Mengimplementasikan bentuk-bentuk fungsi rekursif.
   c. Mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
   d. Mengimplementasikan struktur data tree, khususnya Binary Tree.
2. Pengertian Tree

   Tree dalam C++ adalah struktur data hierarkis yang menyerupai pohon dengan akar (root) sebagai titik awal. Tree terdiri dari kumpulan node, di mana setiap node memiliki data dan dapat terhubung ke node lain melalui edge (sisi). Node yang tidak memiliki parent disebut sebagai root, sedangkan node tanpa anak disebut sebagai leaf (daun). Struktur ini sangat berguna untuk representasi data yang memiliki hubungan hierarkis, seperti organisasi, file sistem, atau ekspresi matematika.

   Komponen Utama Tree
   a. Root (Akar): Node teratas dalam tree.
   b. Parent (Orang Tua): Node yang memiliki anak.
   c. Child (Anak): Node yang berasal dari node lain (parent).
   d. Leaf (Daun): Node yang tidak memiliki anak (anak kiri dan kanan null).
   e. Subtree: Tree kecil yang merupakan bagian dari tree besar.
   f. Height (Tinggi): Jumlah level dari node paling atas hingga paling bawah.
   g. Degree: Jumlah child dari suatu node.

   Jenis-jenis Tree
   a. Binary Tree: Setiap node memiliki maksimal dua child (anak kiri dan kanan).
   b. Binary Search Tree (BST):
      - Sebuah binary tree di mana:
        - Semua nilai di subtree kiri lebih kecil daripada parent.
        - Semua nilai di subtree kanan lebih besar daripada parent.
   c. AVL Tree: Binary search tree yang selalu balanced.
   d. Heap Tree:
      - Max-Heap: Parent lebih besar dari semua anak.
      - Min-Heap: Parent lebih kecil dari semua anak.
   e. N-ary Tree:
      - Setiap node bisa memiliki maksimal N anak.

   Operasi pada Tree

   Traversal (Penelusuran):
   1. Pre-order: Root → Left → Right.
   2. In-order: Left → Root → Right.
   3. Post-order: Left → Right → Root.
   4. Level-order: Penelusuran berdasarkan level menggunakan queue.

Berikut Implementasi Tree yang dilakukan:

```
#include <iostream>
using namespace std;

/// PROGRAM BINARY TREE

// Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri
struct Pohon {
   char data;
   Pohon *left, *right;
   Pohon *parent;
};

// Variabel global unutk menyimpan root (akar) pohon dan node baru
Pohon *root, *baru;

// Inisialisasi pohon agar kosong
void init() {
   root = NULL;
}

// Mengecek apakah pohon kosong
bool isEmpty() {
   return root == NULL;
}

// Membuat node baru sebagai root pohon
void buatNode(char data) {
   if (isEmpty()) {
      root = new Pohon{data, NULL, NULL, NULL};
      cout << "\nNode " << data << " Berhasil dibuat menjadi root." << endl; //
Menambahkan spasi
   } else {
      cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak
   }
}

// Menambahkan node baru sebagai anak kiri dari node tertentu
Pohon* insertLeft(char data, Pohon *node) {
   if (node->left != NULL) {
      cout << "\nNode " << node->data << " sudah ada di child kiri!" << endl;
      return NULL;
   }
   baru = new Pohon{data, NULL, NULL, node};
   node->left = baru;
   cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data <<
endl;
   return baru;
}
```

```
Pohon* insertRight(char data, Pohon *node) {
   if (node->right != NULL) {
      cout << "\nNode " << node->data << " sudah ada di child kanan!" << endl;
      return NULL;
   }
   baru = new Pohon{data, NULL, NULL, node};
   node->right = baru;
   cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data
<< endl;
   return baru;
}

void update(char data, Pohon *node) {
   if (!node) {
      cout << "\nNode yang ingin diubah tidak ditemukan " << endl;
      return;
   }
   char temp = node->data;
   node->data = data;
   cout << "\nData node " << temp << " berhasil diubah menjadi " << data << endl;
}

void find(char data, Pohon *node) {
   if (!node) return;

   if (node->data == data) {
      cout << "\nNode ditemukan " << data << endl;
      return;
   }

   find(data, node->left);
   find(data, node->right);
}

void preOrder(Pohon *node) {
   if (!node) return;
   cout << node->data << " ";
   preOrder(node->left);
   preOrder(node->right);
}

void inOrder(Pohon *node) {
   if (!node) return;
   inOrder(node->left);
   cout << node->data << " ";
   inOrder(node->right);
}
```

```
void postOrder(Pohon *node) {
   if (!node) return;
   postOrder(node->left);
   postOrder(node->right);
   cout << node->data << " ";
}

Pohon* deleteNode(Pohon *node, char data) {
   if (!node) return NULL;

   if (data < node->data) {
      node->left = deleteNode(node->left, data);
   } else if (data > node->data) {
      node->right = deleteNode(node->right, data);
   } else {
      if (!node->left) {
         Pohon *temp = node->right;
         delete node;
         return temp;
      } else if (!node->right) {
         Pohon *temp = node->left;
         delete node;
         return temp;
      }

      Pohon *successor = node->right;
      while (successor->left) successor = successor->left;
      node->data = successor->data;
      node->right = deleteNode(node->right, successor->data);
   }
   return node;
}

Pohon* mostLeft(Pohon *node) {
   if (!node) return NULL;
   while (node->left) node = node->left;
   return node;
}

Pohon* mostRight(Pohon *node) {
   if (!node) return NULL;
   while (node->right) node = node->right;
   return node;
}

int main() {
   init();
   buatNode('F');
   insertLeft('B', root);
   insertRight('G', root);
```

```
    insertLeft('A', root->left);
    insertRight('D', root->left);
    insertLeft('C', root->left->right);
    insertRight('E', root->left->right);

    cout << "\nPre-Order traversal: ";
    preOrder(root);
    cout << "\nIn-Order traversal: ";
    inOrder(root);
    cout << "\nPost-Order traversal: ";
    postOrder(root);

    cout << "\nMenghapus node D.";
    root = deleteNode(root, 'D');
    cout << "\nPre-Order traversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}
```

```
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 9\output> & .\'Guided.exe'

Node F Berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri F

Node G berhasil ditambahkan ke child kanan F

Node A berhasil ditambahkan ke child kiri B

Node D berhasil ditambahkan ke child kanan B

Node C berhasil ditambahkan ke child kiri D

Node E berhasil ditambahkan ke child kanan D

Pre-Order traversal: F B A D C E G
In-Order traversal: A B C D E F G
Post-Order traversal: A C E D B G F
Menghapus node D.
Pre-Order traversal setelah penghapusan: A B C E F G
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 9\output>
```

B. UNGUIDED

```
 #include <iostream>
#include <climits>
using namespace std;

// Struktur data pohon biner
struct Pohon {
   char data;
   Pohon *left, *right, *parent;
};

// Variabel global
Pohon *root = NULL, *baru;

// Inisialisasi pohon
void init() {
   root = NULL;
}

// Mengecek apakah pohon kosong
bool isEmpty() {
   return root == NULL;
}

// Membuat node baru
void buatNode(char data) {
   if (isEmpty()) {
      root = new Pohon{data, NULL, NULL, NULL};
      cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
   } else {
      cout << "\nRoot sudah dibuat." << endl;
   }
}

// Menambahkan node sebagai anak kiri
Pohon* insertLeft(char data, Pohon *node) {
   if (node->left != NULL) {
      cout << "\nNode " << node->data << " sudah memiliki anak kiri!" <<
endl;
      return NULL;
   }
   baru = new Pohon{data, NULL, NULL, node};
   node->left = baru;
   cout << "\nNode " << data << " berhasil ditambahkan ke anak kiri " <<
node->data << endl;
   return baru;
}

// Menambahkan node sebagai anak kanan
Pohon* insertRight(char data, Pohon *node) {
```

```
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah memiliki anak kanan!"
<< endl;
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke anak kanan " <<
node->data << endl;
    return baru;
}

// Menampilkan anak langsung dari node
void tampilkanChild(Pohon *node) {
    if (!node) {
        cout << "\nNode tidak ditemukan." << endl;
        return;
    }
    cout << "\nAnak langsung dari " << node->data << ": ";
    if (node->left) cout << "Kiri: " << node->left->data << " ";
    if (node->right) cout << "Kanan: " << node->right->data << " ";
    if (!node->left && !node->right) cout << "Tidak memiliki anak.";
    cout << endl;
}

// Menampilkan semua keturunan dari node
void tampilkanDescendant(Pohon *node) {
    if (!node) return;
    if (node->left || node->right) cout << node->data << " -> ";
    if (node->left) {
        cout << node->left->data << " ";
        tampilkanDescendant(node->left);
    }
    if (node->right) {
        cout << node->right->data << " ";
        tampilkanDescendant(node->right);
    }
}

// Memeriksa apakah pohon valid BST
bool is_valid_bst(Pohon *node, char min_val, char max_val) {
    if (!node) return true;
    if (node->data <= min_val || node->data >= max_val) return false;
    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}

// Menghitung jumlah simpul daun
int cari_simpul_daun(Pohon *node) {
    if (!node) return 0;
```

```
    if (!node->left && !node->right) return 1;
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

// Menu program
void menu() {
    char pilihan, data, parent;
    Pohon *current;
    do {
        cout << "\nMenu Program Binary Tree:"
             << "\n1. Buat Node Root"
             << "\n2. Tambah Anak Kiri"
             << "\n3. Tambah Anak Kanan"
             << "\n4. Tampilkan Child"
             << "\n5. Tampilkan Descendant"
             << "\n6. Cek Valid BST"
             << "\n7. Hitung Jumlah Simpul Daun"
             << "\n8. Keluar"
             << "\nPilih: ";
        cin >> pilihan;
        switch (pilihan) {
            case '1':
                cout << "Masukkan data root: ";
                cin >> data;
                buatNode(data);
                break;
            case '2':
                cout << "Masukkan parent: ";
                cin >> parent;
                current = root; // Cari node parent
                while (current && current->data != parent) {
                    if (parent < current->data)
                        current = current->left;
                    else
                        current = current->right;
                }
                if (current) {
                    cout << "Masukkan data anak kiri: ";
                    cin >> data;
                    insertLeft(data, current);
                } else {
                    cout << "Node parent tidak ditemukan.\n";
                }
                break;
            case '3':
                cout << "Masukkan parent: ";
                cin >> parent;
                current = root; // Cari node parent
                while (current && current->data != parent) {
                    if (parent < current->data)
```

```
                    current = current->left;
                else
                    current = current->right;
            }
            if (current) {
                cout << "Masukkan data anak kanan: ";
                cin >> data;
                insertRight(data, current);
            } else {
                cout << "Node parent tidak ditemukan.\n";
            }
            break;
        case '4':
            cout << "Masukkan node untuk menampilkan child: ";
            cin >> parent;
            current = root;
            while (current && current->data != parent) {
                if (parent < current->data)
                    current = current->left;
                else
                    current = current->right;
            }
            tampilkanChild(current);
            break;
        case '5':
            cout << "Masukkan node untuk menampilkan descendant: ";
            cin >> data;
            current = root;
            while (current && current->data != data) {
                if (data < current->data)
                    current = current->left;
                else
                    current = current->right;
            }
            if (current) {
                cout << "Descendant dari " << data << ": ";
                tampilkanDescendant(current);
            } else {
                cout << "Node tidak ditemukan.\n";
            }
            break;
        case '6':
            if (is_valid_bst(root, CHAR_MIN, CHAR_MAX))
                cout << "\nPohon adalah Binary Search Tree.\n";
            else
                cout << "\nPohon BUKAN Binary Search Tree.\n";
            break;
        case '7':
            cout << "\nJumlah simpul daun: " << cari_simpul_daun(root) <<
endl;
```

```
                    break;
                case '8':
                    cout << "Keluar dari program.\n";
                    break;
                default:
                    cout << "Pilihan tidak valid.\n";
            }
        } while (pilihan != '8');
    }

    // Fungsi utama
    int main() {
        init();
        menu();
        return 0;
}
```

1. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinputkan!
   (ini setelah melakukan inputan)

```
Menu Program Binary Tree:
1. Buat Node Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Cek Valid BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 4
Masukkan node untuk menampilkan child: F

Masukkan node untuk menampilkan child: F
Masukkan node untuk menampilkan child: F

Anak langsung dari F: Kiri: B Kanan: G

Menu Program Binary Tree:
1. Buat Node Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Cek Valid BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 5
Masukkan node untuk menampilkan descendant: F
Descendant dari F: F -> B B -> A D G
```

2. Buatlah fungsi rekursif is_valid_bst(node, min_val, max_val) untuk memeriksa apakah suatu pohon memenuhi properti Binary Search Tree. Uji fungsi ini pada berbagai pohon, baik yang valid maupun tidak valid sebagai BST.

```
Menu Program Binary Tree:
1. Buat Node Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Cek Valid BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 6

Pohon adalah Binary Search Tree.
```

3. Buatlah fungsi rekursif cari_simpul_daun(node) untuk menghitung jumlah simpul daun dalam Binary Tree. Simpul daun adalah node yang tidak memiliki anak kiri maupun kanan.

```
Menu Program Binary Tree:
1. Buat Node Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Cek Valid BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 7

Jumlah simpul daun: 3
```