LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 9
GRAPH DAN TREE

Disusun Oleh :

Muhammad Hamzah Haifan Ma'ruf

231102091

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. **Dasar Teori**

Graph terdiri dari beberapa kumpulan titik (node) dan garis (edge). Node, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri. Edge, adalah penghubung antara satu node dengan node yang lain. Sebuah garis harus diawali dan diakhiri titik. Path adalah jalur dari satu titik ke titik lain. Sebuah path yang diawali dan diakhiri dengan titik yang sama disebut juga dengan simpul tertutup.

Berdasarkan orientasi arah sisi nya, graph dapat dibedakan menjadi 2 yaitu :

- Directed graph atau graf berarah adalah graph yang setiap sisi nya memiliki orientasi arah.
- Undirected graph atau graf tak berarah adalah graph yang sisi nya tidak memiliki orientasi arah.

Tree adalah struktur data non linier berbentuk hierarki yang terdiri dari sekumpulan node yang berbeda. Node, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri. Root, adalah sebuah node yang terletak di posisi tertinggi atau urutan pertama dari suatu tree. Depth, adalah jarak atau ketinggian antara root dan node. Degree, adalah banyaknya anak atau turunan dari suatu node.

Ada beberapa cara untuk menggambar sebuah tree, diantaranya dapat dengan Graph, Diagram Venn, Notasi Kurung, dan Identitas. Tree yang hanya memiliki maksimal dua child (anak) disebut dengan binary tree atau pohon biner.

Operasi yang terdapat pada binary tree

- Pre Order (DFS – Depth First Search)
- In Order
- Last Order
- Level Order (BFS – Bread First Search)

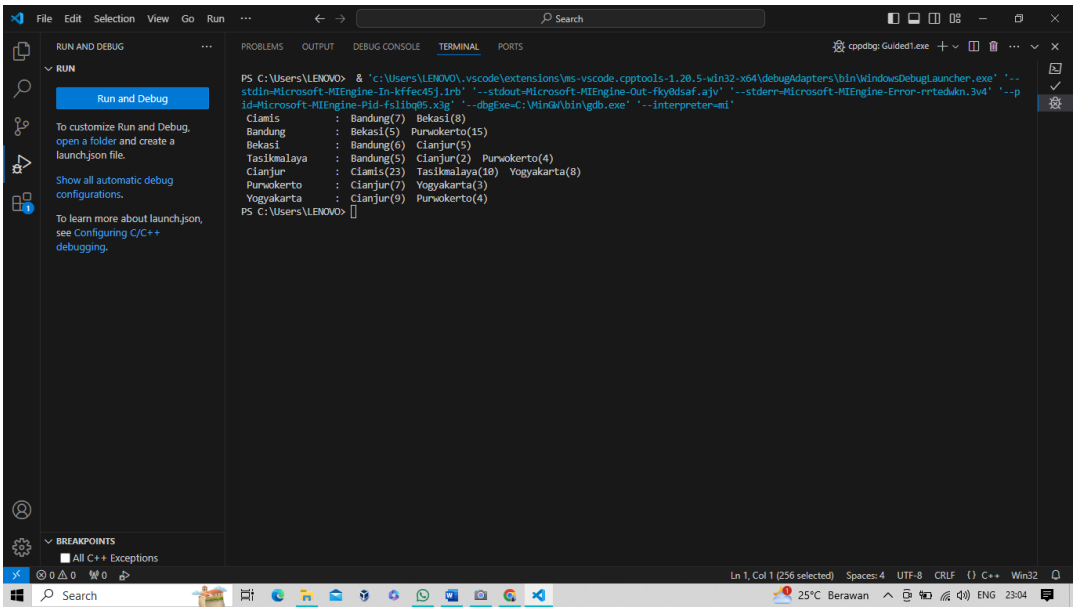
B. **Guided**

Guided 1

Source Code

```
#include <iostream>
#include <iomanip>

using namespace std;
string simpul[7] = {
    "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur",
"Purwokerto", "Yogyakarta"
};
int busur[7][7] = {
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}
};

void tampilGraph() {
    for (int baris=0; baris<7; baris++) {
        cout << " " << setiosflags(ios::left) << setw(15) <<
simpul[baris] << " : ";
        for (int kolom=0; kolom<7; kolom++) {
            if (busur[baris][kolom] != 0) {
                cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom] << ") ";
            }
        } cout << endl;
    }
}

int main() {
    tampilGraph();
    return 0;
}
```

Screenshots Output :



Deskripsi :

Program di atas adalah merepresentasikan tujuh simpul yang diberi nama Ciamis, Bandung, Bekasi, Tasikmalaya, Cianjur, Purwokerto, dan Yogyakarta. Matriks busur berukuran 7x7 digunakan untuk menyimpan bobot atau biaya dari satu simpul ke simpul lainnya. Nilai yang tidak nol dalam matriks busur menunjukkan adanya busur (edge) dari satu simpul ke simpul lainnya, sedangkan nilai nol menunjukkan bahwa tidak ada busur antara dua simpul tersebut. Simpul-simpul tersebut dihubungkan dengan busur yang memiliki bobot tertentu, misalnya busur dari simpul Ciamis ke Bandung memiliki bobot 7, dari Bandung ke Bekasi memiliki bobot 5, dan seterusnya.

Fungsi tampilGraph digunakan untuk menampilkan graf tersebut dalam bentuk yang lebih mudah dipahami. Fungsi ini menggunakan dua loop bersarang untuk mengiterasi setiap baris dan kolom dari matriks busur. Untuk setiap baris (simpul), ia mencetak nama simpul tersebut, diikuti oleh daftar simpul-simpul tujuan yang terhubung dengannya beserta bobot busurnya. Fungsi setw dari pustaka iomanip digunakan untuk menetapkan lebar kolom agar output lebih rapi. Program ini kemudian menjalankan fungsi tampilGraph dalam fungsi main untuk menampilkan struktur graf ke layar saat program dieksekusi. Hasilnya adalah representasi visual dari konektivitas antara simpul-simpul tersebut, bersama dengan bobot yang menunjukkan biaya atau jarak antara simpul-simpul tersebut.

Guided 2

Source Code

```
#include <iostream>
#include <iomanip>

using namespace std;
```

```
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

void init()
{
    root = NULL;
}

bool isEmpty()
{
    return root == NULL;
}

void buatNode(char data)
{
    if (isEmpty())
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat sebagai root."
             << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->data << " sudah ada child kiri !"
<< endl;
```

```
            return NULL;
        }
        else
        {
            Pohon *baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke child
kiri " << baru->parent->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\n Node " << node->data << " sudah ada child kanan
!" << endl;
            return NULL;
        }
        else
        {
            Pohon *baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke child
kanan " << baru->parent->data << endl;
            return baru;
        }
    }
}

void update(char data, Pohon *node)
{
```

```
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\n Node yang ingin diganti tidak ada!!" << endl;
        }
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah menjadi "
                 << data << endl;
        }
    }
}

void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

void find(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
```

```
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left->data <<
endl;
            else if (node->parent != NULL && node->parent->right != node
&& node->parent->left == node)
                cout << " Sibling : " << node->parent->right->data <<
endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << " Child Kiri : (tidak punya Child kiri)" << endl;
            else
                cout << " Child Kiri : " << node->left->data << endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child kanan)" <<
endl;
            else
                cout << " Child Kanan : " << node->right->data << endl;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
```

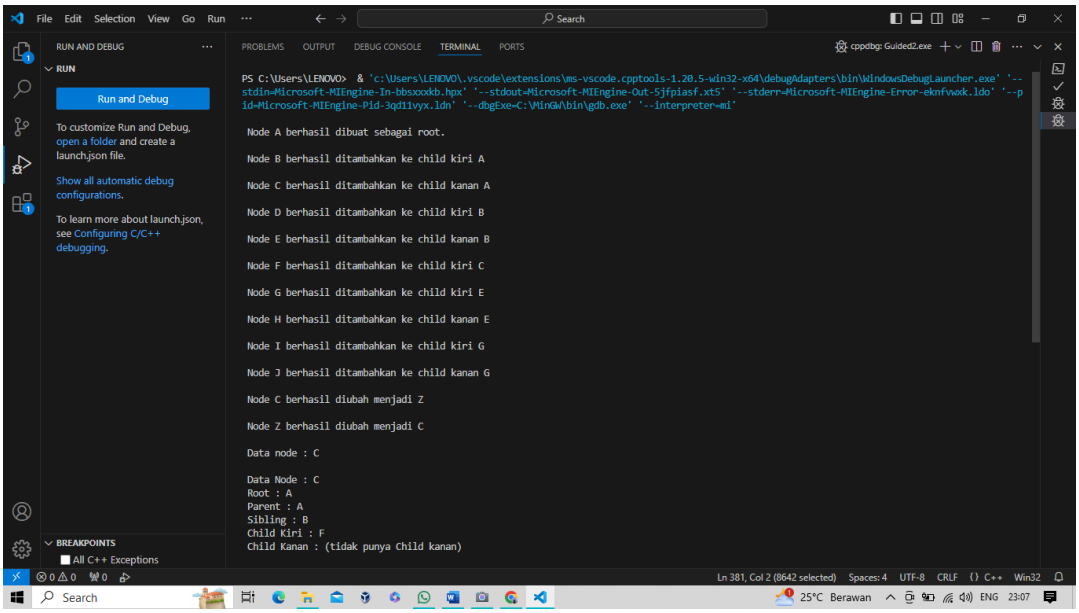
```
    }
}

// inOrder
void inOrder(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
```

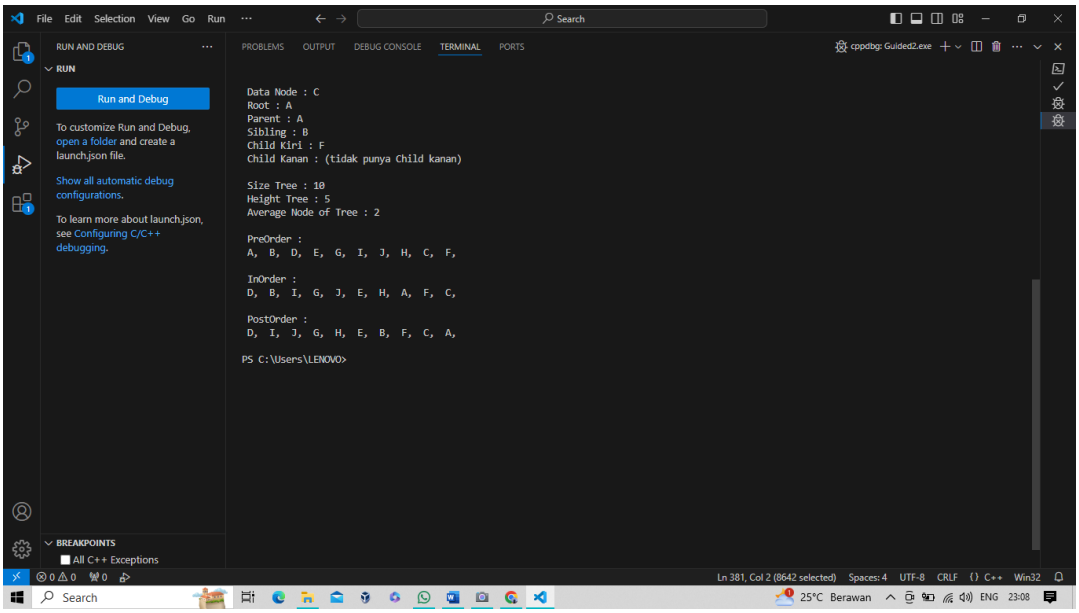
```
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil dihapus."
<< endl;
    }
}

void clear()
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    }
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
```

```
// Cek Size Tree
int size(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
```

```
    }
}

// Karakteristik Tree
void characteristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI,
*nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    characteristic();

    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"<< endl;

    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;

    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;
}
```

Screenshots Output :





Deskripsi :

Program di atas adalah sebuah implementasi pohon biner dalam bahasa C++ yang menyediakan berbagai fungsi untuk membuat, memodifikasi, menelusuri, dan menghapus pohon biner. Struktur Pohon digunakan untuk merepresentasikan setiap node dalam pohon, yang menyimpan data karakter dan pointer ke node anak kiri, anak kanan, dan parent. Program ini mencakup berbagai fungsi untuk memanipulasi pohon seperti buatNode untuk membuat node root, insertLeft dan insertRight untuk menambahkan node ke kiri atau kanan dari node tertentu, update untuk mengubah data node, dan retrieve untuk mengambil data dari node tertentu.

Selain itu, program ini juga menyediakan fungsi untuk menampilkan informasi detail tentang node tertentu dengan find, serta berbagai jenis penelusuran seperti preOrder, inOrder, dan postOrder. Fungsi deleteTree dan deleteSub digunakan untuk menghapus seluruh pohon atau subtree tertentu. Fungsi size dan height

menghitung ukuran dan tinggi pohon, dan fungsi characteristic menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Program utama (main) mendemonstrasikan penggunaan berbagai fungsi ini dengan membuat dan memodifikasi pohon, serta menampilkan hasil penelusuran dan karakteristik pohon.

C. Unguided

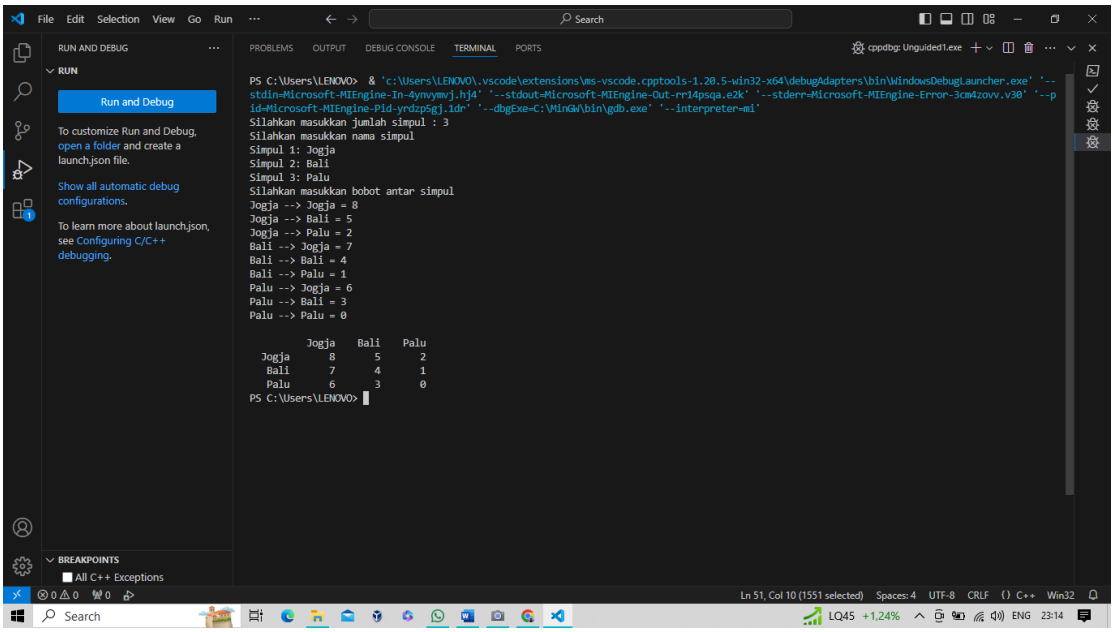
Unguided 1

Source Code

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

int main()
{
    int Haifan_2311102091;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> Haifan_2311102091;
    vector<string> simpul(Haifan_2311102091);
    vector<vector<int>> busur(Haifan_2311102091,
    vector<int>(Haifan_2311102091, 0));

    cout << "Silahkan masukkan nama simpul " << endl;

    for (int i = 0; i < Haifan_2311102091; i++) {
        cout << "Simpul " << (i + 1) << ": ";
        cin >> simpul[i];
        }
        cout << "Silahkan masukkan bobot antar simpul" << endl;

    for (int i = 0; i < Haifan_2311102091; i++) {

        for (int j = 0; j < Haifan_2311102091; j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> busur[i][j];
             }
        }
             cout << endl;
             cout << setw(7) << " ";
             for (int i = 0; i < Haifan_2311102091; i++) {

                cout << setw(8) << simpul[i];
            }
                cout << endl;
```

```
            for (int i = 0; i < Haifan_2311102091; i++)
            {
                cout << setw(7) << simpul[i];

                for (int j = 0; j < Haifan_2311102091; j++)

            {
                cout << setw(8) << busur[i][j];
            }
            cout << endl;
        }
    }
```

Screenshots Output :



Deskripsi :

Program di atas adalah implementasi C++ untuk membuat dan menampilkan matriks ketetanggaan (adjacency matrix) dari sebuah graf berbobot. Matriks ketetanggaan adalah representasi graf yang menyimpan bobot atau jarak antar simpul dalam bentuk matriks dua dimensi. Setiap elemen matriks menunjukkan bobot dari satu simpul ke simpul lainnya.

Pertama-tama, program meminta pengguna untuk memasukkan jumlah simpul yang ada dalam graf. Informasi ini digunakan untuk mendeklarasikan dua vektor: satu untuk menyimpan nama-nama simpul dan satu lagi untuk menyimpan matriks ketetanggaan. Matriks ketetanggaan diinisialisasi dengan nilai nol, yang menunjukkan bahwa pada awalnya tidak ada hubungan antar simpul.

Selanjutnya, program meminta pengguna untuk memasukkan nama-nama simpul. Nama-nama ini disimpan dalam vektor simpul. Setelah itu, program meminta pengguna untuk memasukkan bobot antar simpul. Pengguna diminta untuk memasukkan bobot dari setiap simpul ke setiap simpul lainnya, dan nilai-nilai ini disimpan dalam matriks busur.

Terakhir, program menampilkan matriks ketetanggaan dalam format yang rapi. Bagian ini dimulai dengan mencetak nama-nama simpul sebagai header kolom. Kemudian, untuk setiap simpul, program mencetak nama simpul sebagai header baris dan bobot yang sesuai dari matriks ketetanggaan. Penggunaan fungsi setw dari pustaka iomanip membantu dalam penyusunan kolom sehingga output menjadi lebih mudah dibaca.

Unguided 2

Source Code

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

class BinarySearchTree
{
private:
    struct nodeTree
    {
        nodeTree *left;
        nodeTree *right;
        int data_091;
    };
        nodeTree *root;
public:
    BinarySearchTree()
    {
        root = NULL;
    }

    bool isEmpty() const { return root == NULL; }
    void print_inorder();
    void inorder(nodeTree *);
    void print_preorder();
    void preorder(nodeTree *);
    void print_postorder();
    void postorder(nodeTree *);
    void insert(int);
    void remove(int);
};
void BinarySearchTree::insert(int a)
{
    nodeTree *t = new nodeTree;
```

```
    nodeTree *parent;
    t->data_091 = a;
    t->left = NULL;
    t->right = NULL;
    parent = NULL;

    if (isEmpty())
        root = t;
    else
    {
        nodeTree *current;
        current = root;
        while (current)
        {
            parent = current;
            if (t->data_091 > current->data_091)
                current = current->right;
            else
                current = current->left;
        }

        if (t->data_091 < parent->data_091)
            parent->left = t;
        else
            parent->right = t;
    }
}
void BinarySearchTree::remove(int a)
{
    // Mencari elemen yang akan dihapus
    bool found = false;
    if (isEmpty())
    {
        cout << "Pohon ini kosong!" << endl;
        return;
    }

    nodeTree *current;
    nodeTree *parent;
    current = root;

    while (current != NULL)
    {
        if (current->data_091 == a)
        {
            found = true;
            break;
    } else {
        parent = current;
```

```
        if (a > current->data_091)
            current = current->right;
        else
            current = current->left;

        }
    }
    if (!found)
    {
        cout << "Data tidak ditemukan!" << endl;
        return;
    }

    // Node dengan satu anak
    if ((current->left == NULL && current->right != NULL) || (current-
>left != NULL && current->right == NULL)) {

        if (current->left == NULL && current->right != NULL) {

            if (parent->left == current)
            {
                parent->left = current->right;
                delete current;
            }
            else
            {
                parent->right = current->right;
                delete current;
            }
        }
        else
        {
            if (parent->left == current)
            {
                parent->left = current->left;
                delete current;
            }
            else
            {
                parent->right = current->left;
                delete current;
            }
        }
        return;
    }

    // Node tanpa anak
    if (current->left == NULL && current->right == NULL)
    {
```

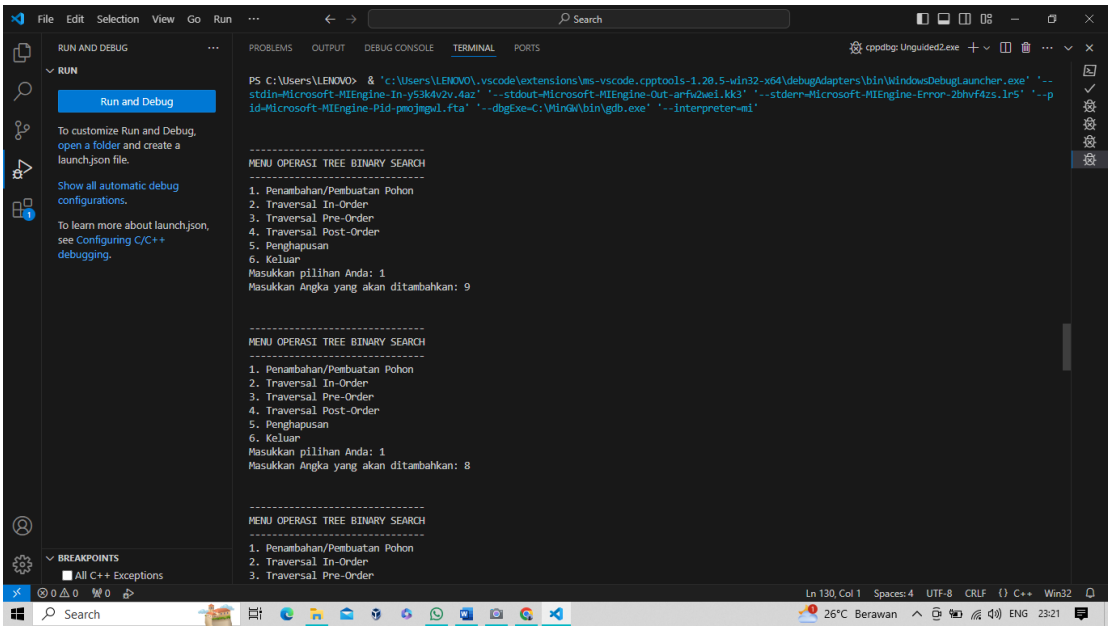
```
        if (parent->left == current)
            parent->left = NULL;
        else
            parent->right = NULL;
        delete current;
        return;
    }

    // Node dengan dua anak
    // Ganti node dengan nilai terkecil pada subtree sebelah kanan
    if (current->left != NULL && current->right != NULL)
    {
        nodeTree *temp;
        temp = current->right;
        if ((temp->left == NULL) && (temp->right == NULL))
        {
            current = temp;
            delete temp;
            current->right = NULL;
        }
        else
        {
            if ((current->right)->left != NULL)
            {
                nodeTree *lcurrent;
                nodeTree *lcurrp;
                lcurrp = current->right;
                lcurrent = (current->right)->left;
                while (lcurrent->left != NULL)
                {
                    lcurrp = lcurrent;
                    lcurrent = lcurrent->left;
                }
                current->data_091 = lcurrent->data_091;
                delete lcurrent;
                lcurrp->left = NULL;

            }
            else
            {
                nodeTree *tmp2;
                tmp2 = current->right;
                current->data_091 = tmp2->data_091;
                current->right = tmp2->right;
                delete tmp2;
            }
        }
        return;
    }
```

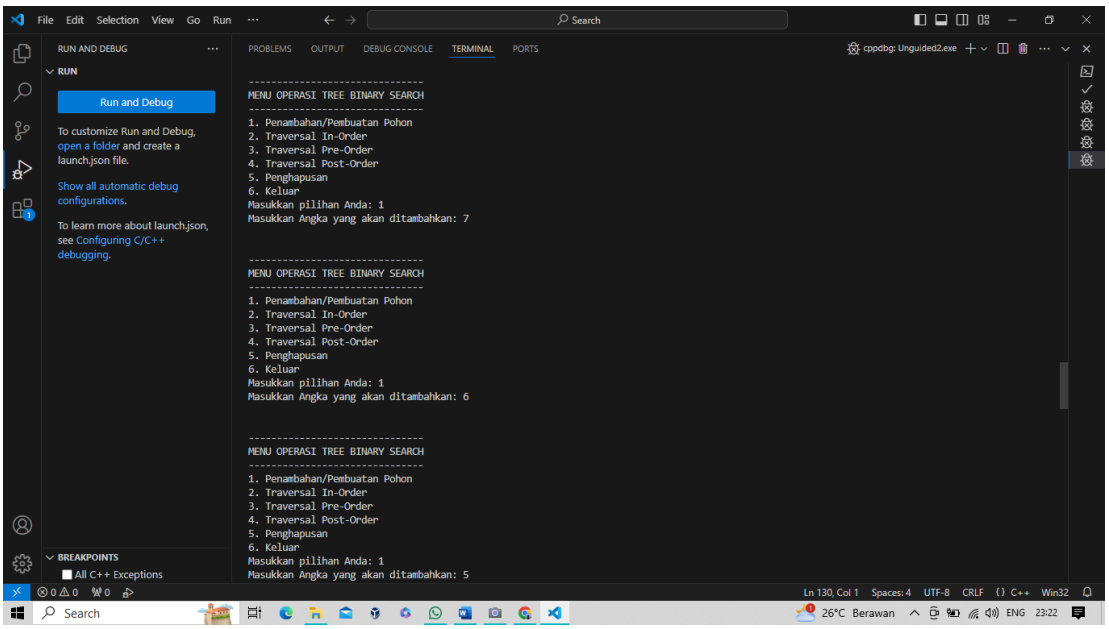
```
}
void BinarySearchTree::print_inorder()
{
    inorder(root);
}
void BinarySearchTree::inorder(nodeTree *b)
{
    if (b != NULL)
    {
        if (b->left)
            inorder(b->left);
        cout << " " << b->data_091 << " ";
        if (b->right)
            inorder(b->right);
    }
    else
        return;
}
void BinarySearchTree::print_preorder()
{
    preorder(root);
}
void BinarySearchTree::preorder(nodeTree *b)
{
    if (b != NULL)
    {
        cout << " " << b->data_091 << " ";
        if (b->left)
            preorder(b->left);
        if (b->right)
            preorder(b->right);

    }
    else
        return;
}
void BinarySearchTree::print_postorder()
{
    postorder(root);
}
void BinarySearchTree::postorder(nodeTree *b)
{
    if (b != NULL)
    {
        if (b->left)
            postorder(b->left);
        if (b->right)
            postorder(b->right);
        cout << " " << b->data_091 << " ";
```

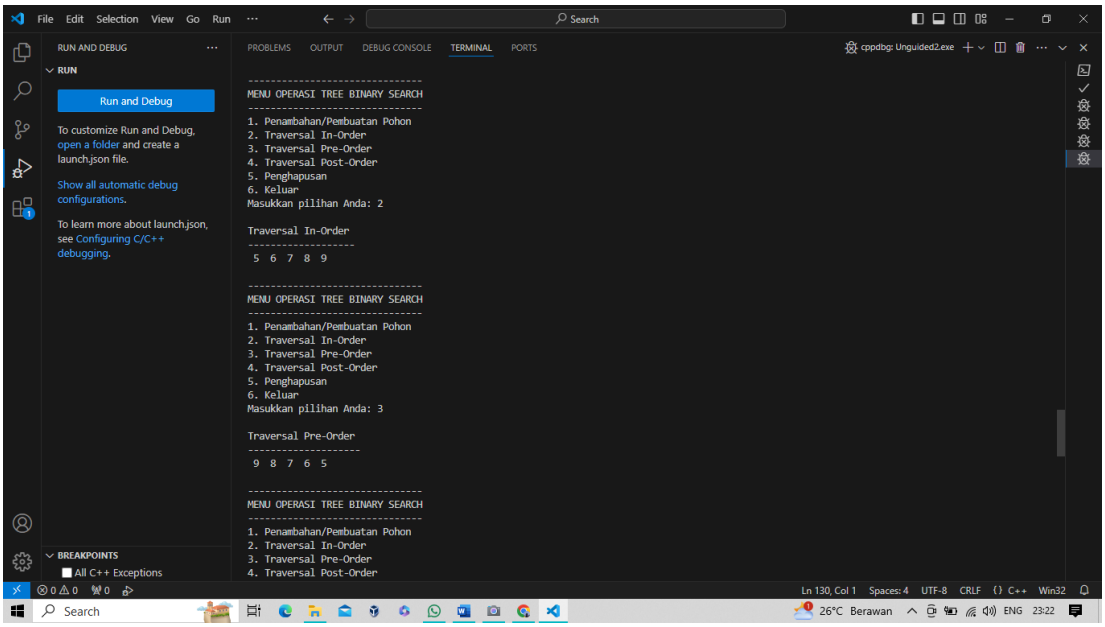
```
    }
    else
        return;
}

int main()
{
    BinarySearchTree b;
    int ch, tmp, tmp1;
    while (1)
    {
        cout << endl << endl; cout << "------------------------------"
<< endl;
        cout << "MENU OPERASI TREE BINARY SEARCH" << endl;
        cout << "------------------------------" << endl;
        cout << "1. Penambahan/Pembuatan Pohon" << endl;
        cout << "2. Traversal In-Order" << endl;
        cout << "3. Traversal Pre-Order" << endl;
        cout << "4. Traversal Post-Order" << endl;
        cout << "5. Penghapusan" << endl;
        cout << "6. Keluar" << endl;
        cout << "Masukkan pilihan Anda: ";
        cin >> ch;
        switch (ch)
        {
            case 1: cout << "Masukkan Angka yang akan ditambahkan: ";
            cin >> tmp;
            b.insert(tmp);
            break;

            case 2:
            cout << endl;
            cout << "Traversal In-Order" << endl;
            cout << "------------------" << endl;
            b.print_inorder();
            break;

            case 3:
            cout << endl;
            cout << "Traversal Pre-Order" << endl;
            cout << "-------------------" << endl;
            b.print_preorder();
            break;

            case 4:
            cout << endl;
            cout << "Traversal Post-Order" << endl;
            cout << "--------------------" << endl;
            b.print_postorder();
```

```
            break;

            case 5:
            cout << "Masukkan angka yang akan dihapus: ";
            cin >> tmp1;
            b.remove(tmp1);
            break;

            case 6:
            return 0;
        }
    }
}
```

Screenshots Output :



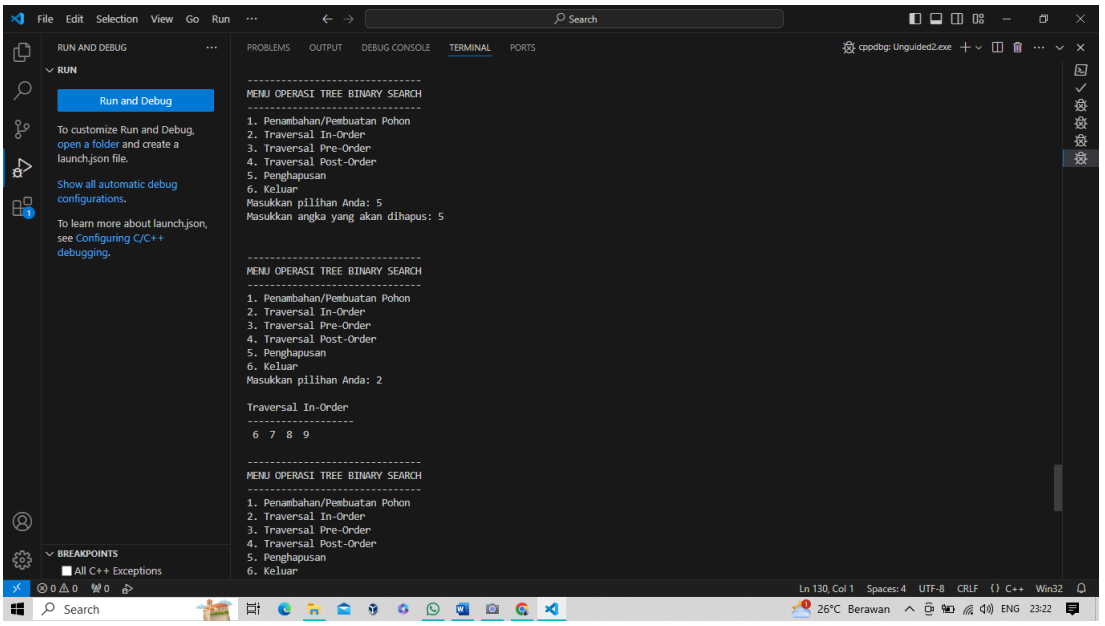



File Edit Selection View Go Run
Search
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
To learn more about launch.json, see Configuring C/C++ debugging.
BREAKPOINTS
All C++ Exceptions
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
cppdbg: Unguided2.exe

MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Masukkan pilihan Anda: 2
Traversal In-Order
---
5 6 7 8 9

MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Masukkan pilihan Anda: 3
Traversal Pre-Order
---
9 8 7 6 5

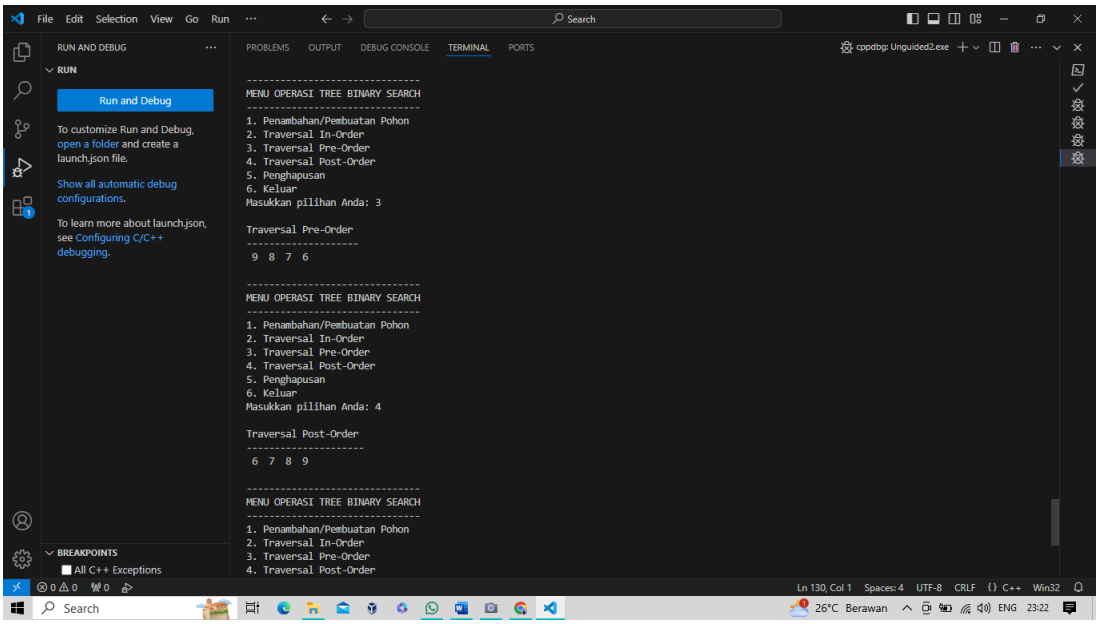
MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
Ln 130, Col 1 Spaces: 4 UTF-8 CRLF C++ Win32
Search
26°C Berawan
ENG 23:22

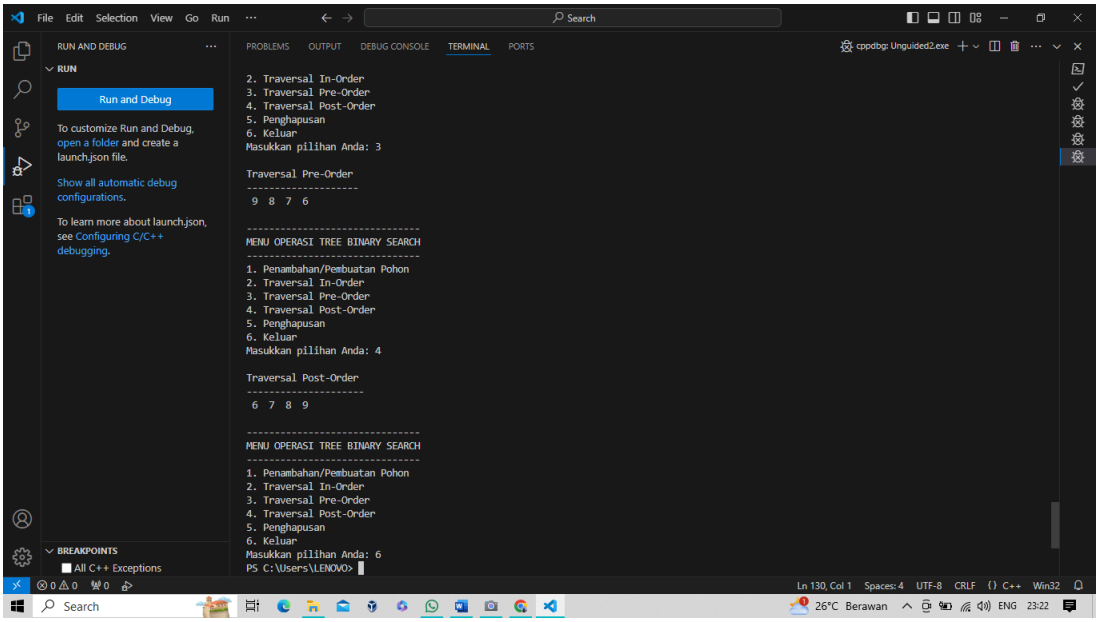

File Edit Selection View Go Run
Search
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
To learn more about launch.json, see Configuring C/C++ debugging.
BREAKPOINTS
All C++ Exceptions
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
cppdbg: Unguided2.exe

MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Masukkan pilihan Anda: 5
Masukkan angka yang akan dihapus: 5

MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Masukkan pilihan Anda: 2
Traversal In-Order
---
6 7 8 9

MENU OPERASI TREE BINARY SEARCH
---
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Ln 130, Col 1 Spaces: 4 UTF-8 CRLF C++ Win32
Search
26°C Berawan
ENG 23:22





Deskripsi :

Program ini adalah pohon biner di mana setiap node memiliki paling banyak dua anak. Kelas `BinarySearchTree` digunakan untuk mendefinisikan BST dengan struktur `nodeTree` yang mewakili setiap node dalam pohon. Setiap nodeTree memiliki data, pointer ke anak kiri, pointer ke anak kanan, dan pointer ke root. Kelas ini juga memiliki metode untuk menyisipkan, menghapus, dan melakukan traversal (penelusuran) dalam urutan in-order, pre-order, dan post-order.

Metode `insert` digunakan untuk menambahkan elemen baru ke dalam BST. Jika pohon kosong, elemen baru menjadi root. Jika tidak, program mencari posisi yang sesuai untuk menyisipkan elemen baru berdasarkan aturan BST di mana elemen yang lebih kecil ditempatkan di kiri dan elemen yang lebih besar di kanan. Metode `remove` digunakan untuk menghapus elemen dari BST. Proses penghapusan menangani tiga kasus utama: node dengan satu anak, node tanpa anak, dan node

dengan dua anak. Dalam kasus node dengan dua anak, node yang dihapus diganti dengan nilai terkecil dari subtree kanan.

Selain itu, kelas `BinarySearchTree` menyediakan metode untuk melakukan traversal pada pohon. Metode `print_inorder` mencetak elemen dalam urutan in-order (kiri, root, kanan), `print_preorder` mencetak elemen dalam urutan pre-order (root, kiri, kanan), dan `print_postorder` mencetak elemen dalam urutan post-order (kiri, kanan, root). Setiap metode traversal memanggil fungsi rekursif yang sesuai untuk mencetak elemen dalam urutan yang benar.

Bagian `main` dari program ini menyediakan menu interaktif untuk pengguna. Menu ini memungkinkan pengguna untuk menambahkan elemen ke dalam pohon, menghapus elemen dari pohon, dan melakukan traversal dalam urutan in-order, pre-order, dan post-order. Pengguna dapat memilih operasi yang diinginkan dengan memasukkan angka yang sesuai dari menu. Program ini memberikan cara yang efektif untuk mengelola data dalam struktur pohon biner pencarian, lengkap dengan operasi dasar dan antarmuka interaktif yang mudah digunakan.

D. Kesimpulan

BinarySearchTree, program ini memungkinkan operasi dasar seperti menyisipkan (insert), menghapus (remove), dan traversal (in-order, pre-order, post-order) pada pohon biner. Dengan menggunakan menu interaktif dalam fungsi main, pengguna dapat dengan mudah berinteraksi dengan BST untuk melakukan berbagai operasi. Implementasi ini memberikan pemahaman yang mendalam tentang struktur data BST. Kita memahami bagaimana elemen-elemen disimpan dan diorganisir dalam pohon berdasarkan nilai mereka, serta aturan dasar BST di mana elemen yang lebih kecil ditempatkan di kiri dan elemen yang lebih besar ditempatkan di kanan. Melalui operasi dasar seperti menyisipkan dan menghapus elemen, serta traversal pohon dalam berbagai urutan, kita belajar bagaimana mengelola data dalam BST dengan efisien.

Pentingnya pengelolaan memori juga menjadi fokus, terutama dalam alokasi dan penghapusan node-node dalam pohon. Program ini juga menunjukkan cara membuat antarmuka pengguna sederhana melalui menu interaktif, memungkinkan pengguna untuk dengan mudah memilih operasi yang ingin dilakukan. Selain itu, penanganan kasus-kasus khusus, seperti pohon kosong atau node dengan satu atau dua anak, juga diperhatikan agar operasi BST berjalan dengan benar.

E. Referensi

Trivusi (2022a) 'Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya,' *Trivusi*, 16 September.

https://www.trivusi.web.id/2022/07/struktur-data-graph.html.

Nurul, R. (2021) *Data structure : Mengenal graph & tree*. https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/.

Trivusi (2022d) 'Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya,' *Trivusi*, 16 September. https://www.trivusi.web.id/2022/07/struktur-data-tree.html.

Profil, K. (no date) *Graf (Graph) dan Pohon (Tree) - Algoritma Pemrograman 2*. http://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html.