LAPORAN PRAKTIKUM

Modul IX

GRAPH DAN TREE

Disusun oleh:

Muhammad Irsyad : **2211102048**

Dosen

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

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2023

BAB I

Tujuan Pembelajaran

a. Mahasiswa diharapkan mampu memahami graph dan tree
b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II

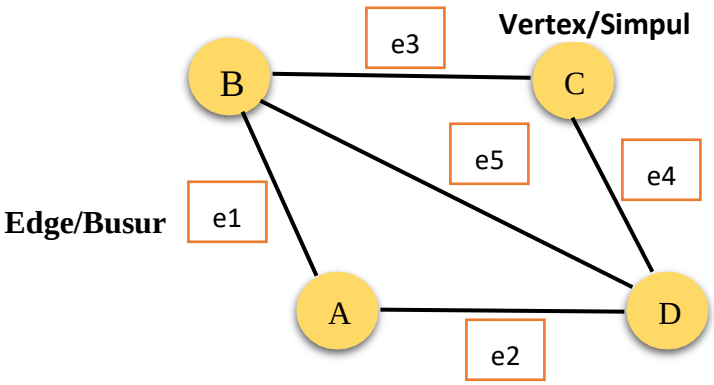
Dasar Teori

1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

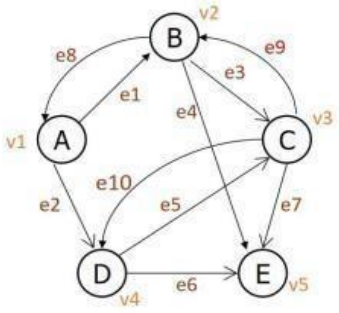
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:



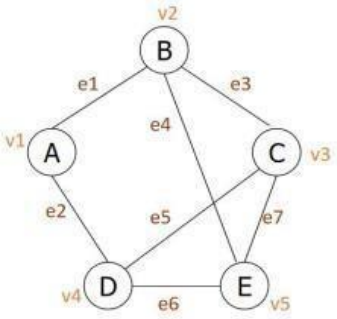
Gambar 1 Contoh Graph

Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

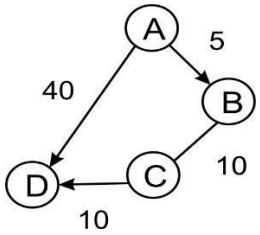
Jenis-jenis Graph



Directed Graph



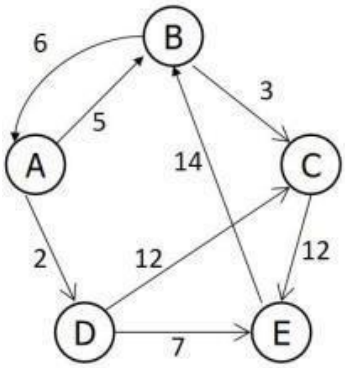
Undirected Graph



Weight Graph

a. **Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur ABadalah e1 sedangkan busur BA adalah e8.
b. **Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidakdiperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
c. **Weigth Graph :** Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks

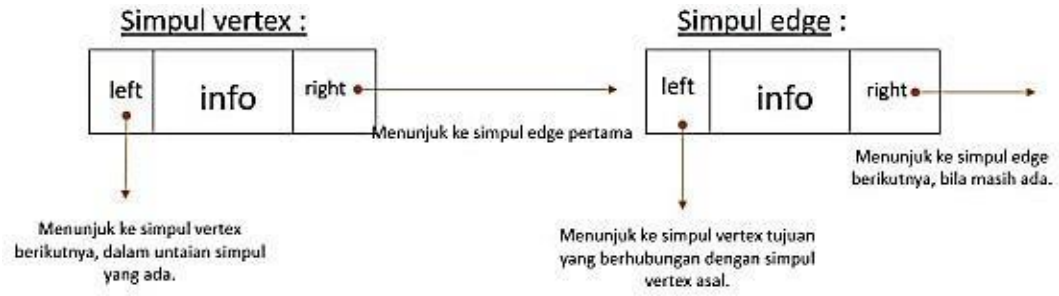


Ke →

Dari		A 0	B 1	C 2	D 3	E 4
A	0	0	5	0	2	0
B	1	6	0	3	0	0
C	2	0	0	0	0	9
D	3	0	0	12	0	7
E	4	0	14	0	0	0

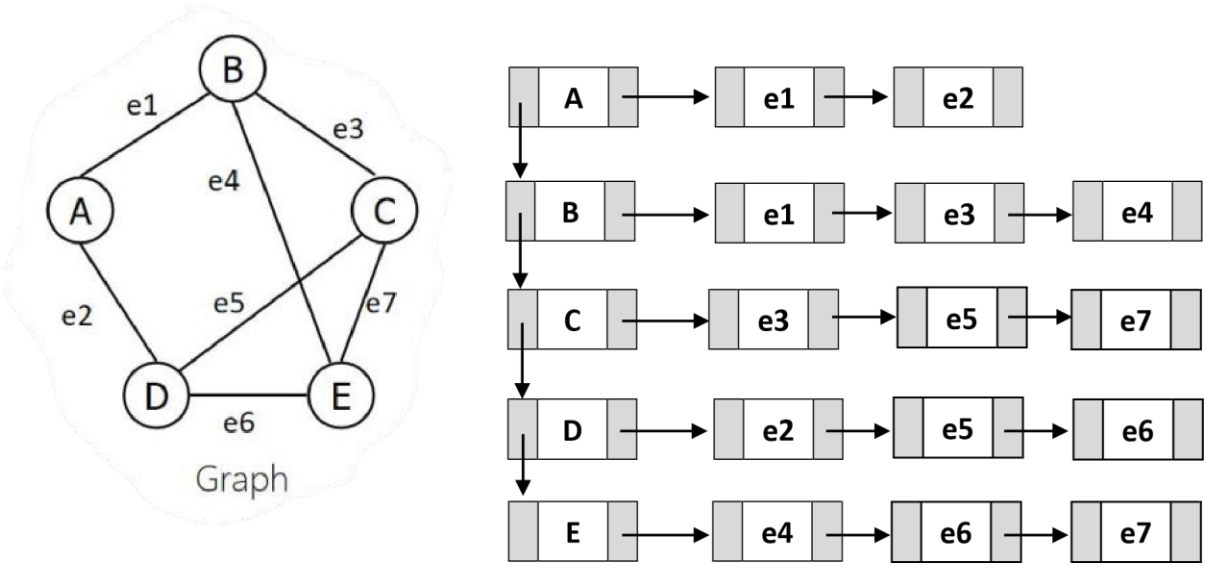
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

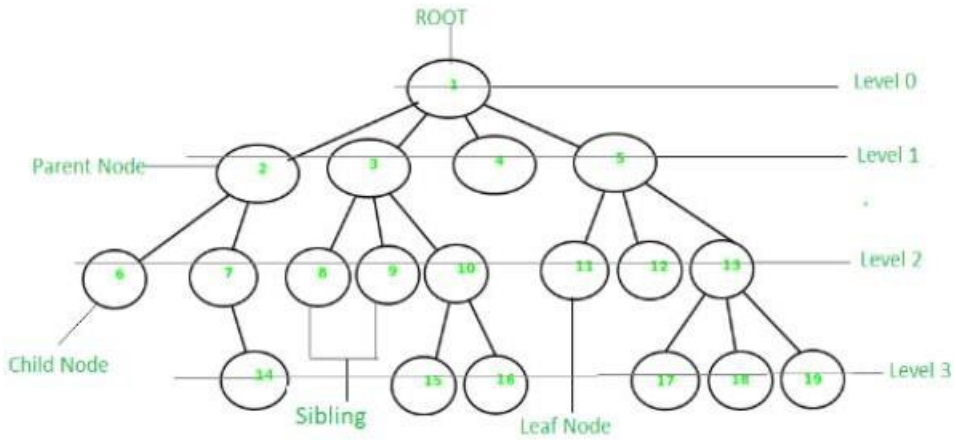
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

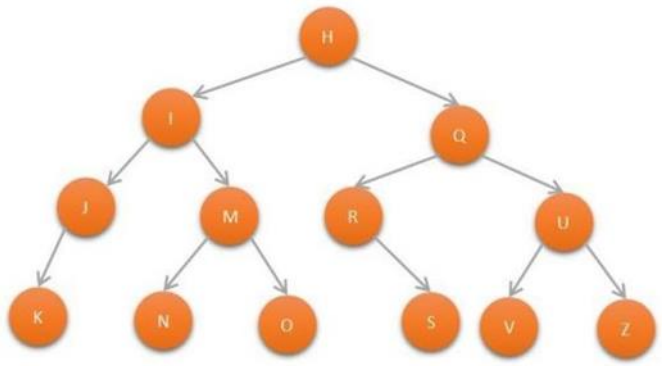
Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

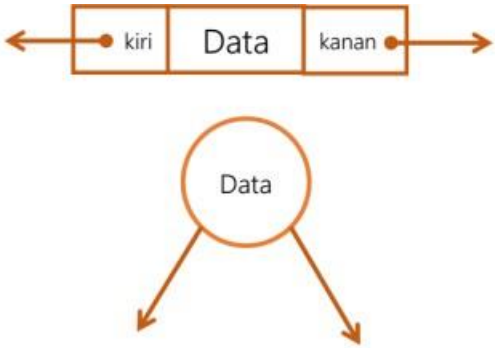
Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



Gambar 1 Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

1. **Create**: digunakan untuk membentuk binary tree baru yang masih kosong.

2. **Clear**: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
3. **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
4. **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.
5. **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong**.**
6. **Update**: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
7. **Retrive**: digunakan untuk mengetahui isi dari node yang ditunjuk pointercurrent dengan syarat tree tidak boleh kosong.
8. **Delete Sub**: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
9. **Characteristic**: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
10. **Traverse**: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.:

    1. Pre-Order

    Penelusuran secara pre-order memiliki alur:

    a. Cetak data pada simpul root

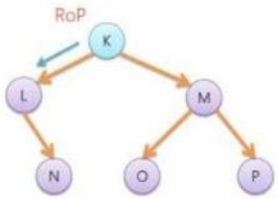
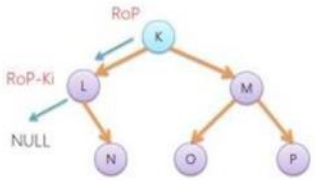
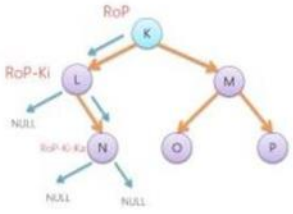
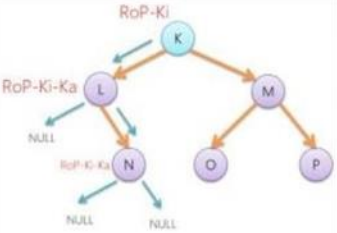
    b. Secara rekursif mencetak seluruh data pada subpohon kiri

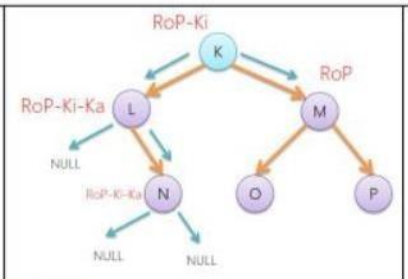
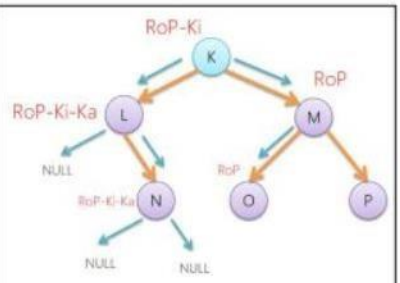
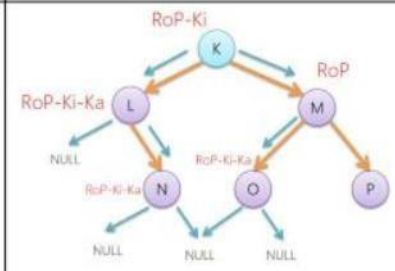
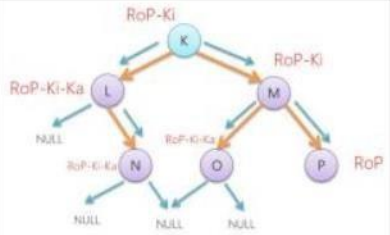
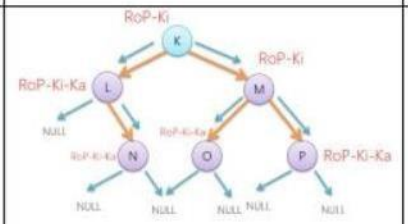
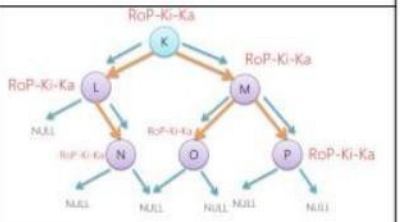
    c. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Alur pre-order

No	Proses	No	Proses
1	 K	2	 K - L
3	 K-L-N	4	 K-L-N

5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

2. In-Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Cetak data pada root

c. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

Ro - KiP - Ka

3. **Post Order**

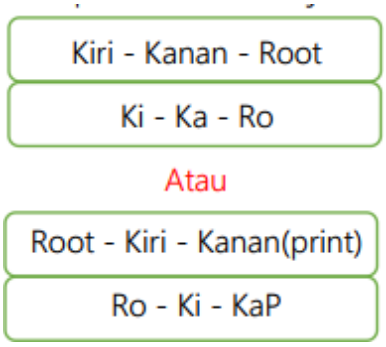
Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Secara rekursif mencetak seluruh data pada subpohon kanan

c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



BAB III

LATIHAN KELAS – GUIDED

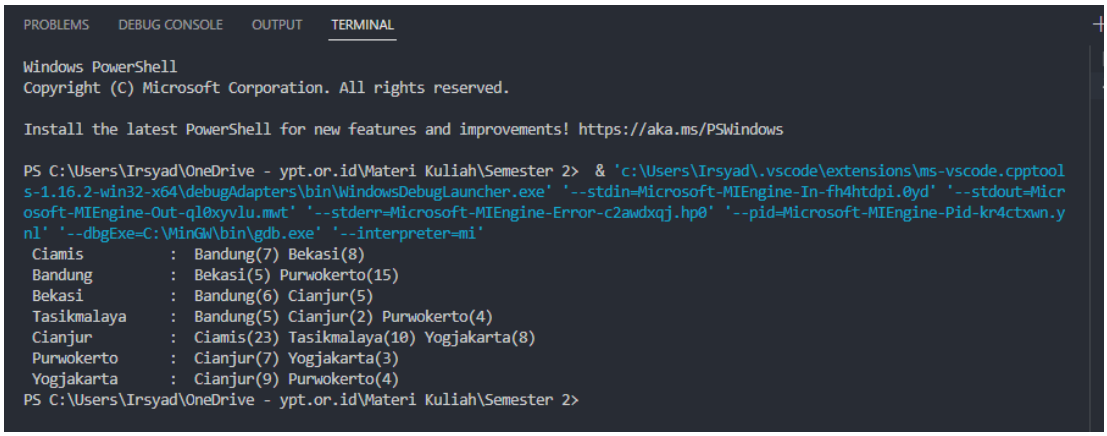
Guided 1

Source Code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                    "Bandung",
                    "Bekasi",
                    "Tasikmalaya",
                    "Cianjur",
                    "Purwokerto",
                    "Yogjakarta"};
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;
}
```

Screenshot Program



Deskripsi Program

Implementasi sederhana dari representasi graf menggunakan matriks ketetanggaan. Program ini menampilkan graf yang terdiri dari simpul-simpul dan busur-busur yang menghubungkan simpul-simpul tersebut.

Pada kode program ini, terdapat beberapa elemen penting, yaitu:

- Library:
    - `<iostream>`: Library input-output standar yang digunakan untuk operasi masukan dan keluaran.
    - `<iomanip>`: Library yang digunakan untuk mengatur format output, seperti lebar kolom dan presisi bilangan.
- Namespace:
    - `using namespace std;`: Digunakan untuk menghindari penulisan `std::` di depan fungsi dan objek standar.
- Variabel dan Array:
    - `simpul`: Array berisi 7 string yang mewakili nama simpul-simpul graf.
    - `busur`: Array 2 dimensi berukuran 7x7 yang mewakili matriks ketetanggaan graf. Nilai `busur[i][j]` menyatakan bobot busur yang menghubungkan simpul `i` dan `j`. Jika nilai `busur[i][j]` adalah 0, maka tidak ada busur yang menghubungkan simpul tersebut.

- Fungsi `tampilGraph()`:
    - Fungsi ini digunakan untuk menampilkan graf ke layar.
    - Menggunakan perulangan untuk mengiterasi melalui setiap baris dan kolom matriks ketetanggaan.
    - Untuk setiap baris, mencetak nama simpul pada kolom pertama.
    - Jika terdapat busur yang menghubungkan simpul pada baris dan kolom tertentu, mencetak nama simpul tujuan beserta bobot busurnya.
- Fungsi `main()`:
    - Fungsi utama program.
    - Memanggil fungsi `tampilGraph()` untuk menampilkan graf.

Dalam program ini, graf direpresentasikan menggunakan matriks ketetanggaan, di mana setiap elemen matriks merepresentasikan bobot busur yang menghubungkan dua simpul. Jika nilai elemen matriks adalah 0, maka tidak ada busur yang menghubungkan simpul-simpul tersebut. Program ini kemudian mencetak graf tersebut ke layar dengan format yang sesuai.

Guided 2

Source Code

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
```

```
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
         cout << "\n Node " << data << " berhasil dibuat menjadi
root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
```

```
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
             cout << "\n Node " << node->data << " sudah ada child
kiri!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            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;
        }
    }
}
// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
```

```
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
             cout << "\n Node " << node->data << " sudah ada child
kanan!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            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;
        }
    }
}
// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        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;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        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;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
```

```
    if (!root)
    {
        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;
        }
    }
}
// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        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 (!root)
        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 (!root)
        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 (!root)
        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 (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
          cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        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 (!root)
    {
        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 (!root)
    {
        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 charateristic()
{
    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);
    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;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n" << endl;
    charateristic();
}
```

Screenshot Program

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

```
Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kanan G

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C

Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)

PreOrder :
A,  B,  D,  E,  G,  I,  J,  H,  C,  F,

InOrder :
D,  B,  I,  G,  J,  E,  H,  A,  F,  C,
```

```
PostOrder :
D,  I,  J,  G,  H,  E,  B,  F,  C,  A,


Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

PreOrder :
A,  B,  D,  E,  C,  F,


Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\Users\Irsyad\OneDrive - ypt.or.id\Materi Kuliah\Semester 2>
```

Ln 362, Col 2 (8982 selected) Spaces: 4 UTF-8 CRLF {} C++

Deskripsi Program

implementasi sederhana dari struktur data binary tree. Program ini menyediakan berbagai fungsi untuk membuat, mengubah, mencari, dan menghapus node dalam binary tree. Berikut ini adalah penjelasan rinci mengenai setiap bagian program:

- Library:
  - `<iostream>`: Library input-output standar yang digunakan untuk operasi masukan dan keluaran.
- Struktur Pohon:
  - `Pohon`: Struktur yang mendefinisikan node dalam binary tree. Setiap node memiliki karakter data, pointer ke anak kiri (`left`), pointer ke anak kanan (`right`), dan pointer ke induk (`parent`).
- Variabel dan Pointer:
  - `root`: Pointer yang menunjuk ke root (akar) dari binary tree.
  - `baru`: Pointer yang digunakan untuk menunjuk ke node baru yang dibuat dalam binary tree.
- Fungsi `init()`:
  - Menginisialisasi pointer `root` menjadi `NULL`, menandakan bahwa binary tree kosong.
- Fungsi `isEmpty()`:
  - Memeriksa apakah binary tree kosong atau tidak.
  - Mengembalikan nilai 1 (true) jika `root` adalah `NULL`, dan 0 (false) jika `root` tidak `NULL`.
- Fungsi `buatNode()`:
  - Membuat node baru dan menetapkannya sebagai root jika binary tree masih kosong.
  - Menetapkan karakter data untuk node baru, dan mengatur pointer anak kiri, anak kanan, dan induk menjadi `NULL`.
  - Jika binary tree tidak kosong, fungsi ini mencetak pesan kesalahan.

- Fungsi `insertLeft()`:
    - Menambahkan node baru sebagai anak kiri dari node yang ditentukan.
    - Memeriksa apakah binary tree sudah dibuat atau belum, apakah node tersebut sudah memiliki anak kiri, dan membuat node baru dengan karakter data yang ditentukan.
    - Mengatur pointer anak kiri, anak kanan, dan induk untuk node baru dan mencetak pesan kesuksesan atau kesalahan.
- Fungsi `insertRight()`:
    - Menambahkan node baru sebagai anak kanan dari node yang ditentukan.
    - Memeriksa apakah binary tree sudah dibuat atau belum, apakah node tersebut sudah memiliki anak kanan, dan membuat node baru dengan karakter data yang ditentukan.
    - Mengatur pointer anak kiri, anak kanan, dan induk untuk node baru dan mencetak pesan kesuksesan atau kesalahan.
- Fungsi `update()`:
    - Mengubah karakter data dari node yang ditentukan.
    - Memeriksa apakah binary tree sudah dibuat atau belum, dan apakah node yang ingin diubah ada.
    - Mengganti karakter data dengan karakter baru dan mencetak pesan kesuksesan atau kesalahan.
- Fungsi `retrieve()`:
    - Menampilkan karakter data dari node yang ditentukan.
    - Memeriksa apakah binary tree sudah dibuat atau belum, dan apakah node yang ditunjuk ada.
    - Mencetak karakter data dari node.
- Fungsi `find()`:

    - Menampilkan informasi tentang node yang ditentukan, termasuk karakter data, root, induk, saudara, dan anak.
    - Memeriksa apakah binary tree sudah dibuat atau belum, dan apakah node yang ditunjuk ada.
    - Mencetak informasi tentang node yang ditunjuk.
- Fungsi Traversal:
    - `preOrder()`: Menampilkan data dari setiap node dalam pre-order traversal (akar-kiri-kanan).
    - `inOrder()`: Menampilkan data dari setiap node dalam in-order traversal (kiri-akar-kanan).
    - `postOrder()`: Menampilkan data dari setiap node dalam post-order traversal (kiri-kanan-akar).
    - Setiap fungsi traversal memeriksa apakah binary tree sudah dibuat atau belum sebelum melakukan traversal.
- Fungsi `deleteTree()`:
    - Menghapus seluruh binary tree, termasuk semua node.
    - Memeriksa apakah binary tree sudah dibuat atau belum.
    - Menghapus setiap node secara rekursif dan membebaskan memori yang dialokasikan.
    - Setelah binary tree dihapus, pointer `root` diatur menjadi `NULL`.
- Fungsi `deleteSub()`:
    - Menghapus subtree yang dimulai dari node yang ditentukan.
    - Memeriksa apakah binary tree sudah dibuat atau belum.
    - Menghapus semua node dalam subtree secara rekursif dan mencetak pesan kesuksesan.
- Fungsi `clear()`:
    - Menghapus seluruh binary tree menggunakan fungsi `deleteTree()`.
    - Memeriksa apakah binary tree sudah dibuat atau belum.

- Fungsi `size()`:
    - Menghitung jumlah node dalam binary tree.
    - Memeriksa apakah binary tree sudah dibuat atau belum.
    - Mengembalikan jumlah node dalam binary tree dengan melakukan perhitungan secara rekursif.
- Fungsi `height()`:
    - Menghitung tinggi level (kedalaman maksimum) dari binary tree.
    - Memeriksa apakah binary tree sudah dibuat atau belum.
    - Mengembalikan tinggi level dari binary tree dengan melakukan perhitungan secara rekursif.
- Fungsi `charateristic()`:
    - Menampilkan karakteristik dari binary tree, termasuk ukuran (jumlah node), tinggi level, dan rata-rata node per level.
    - Menggunakan fungsi `size()` dan `height()` untuk menghitung nilai-nilai tersebut.
- Fungsi `main()`:
    - Fungsi utama program.
    - Memanggil berbagai fungsi untuk membuat binary tree, mengubah node, mencari node, melakukan traversal, menghapus subtree, dan menampilkan karakteristik.
    - Beberapa node ditambahkan ke binary tree dengan menggunakan fungsi `buatNode()`, `insertLeft()`, dan `insertRight()`.
    - Beberapa node diubah dengan menggunakan fungsi `update()`.
    - Beberapa operasi traversal dan penampilan karakteristik dilakukan.
    - Subtree dihapus menggunakan fungsi `deleteSub()`.
    - Operasi traversal dan penampilan karakteristik dilakukan lagi setelah subtree dihapus.

TUGAS – UNGUIDED

Unguided 1

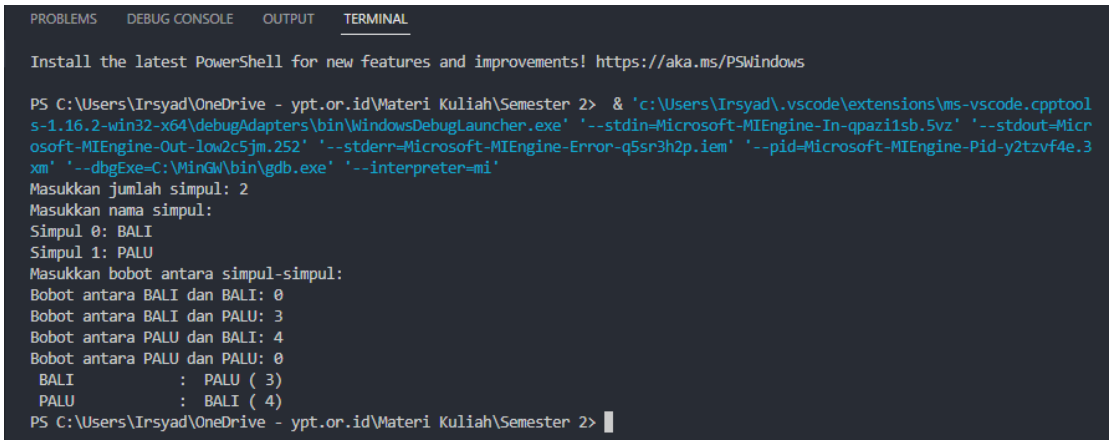
Source Code

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

const int MAX_SIMPUL = 7;

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

void inputGraph(string simpul[], int busur[][MAX_SIMPUL], int
jumlahSimpul)
{
    cout << "Masukkan nama simpul:" << endl;
    for (int i = 0; i < jumlahSimpul; i++)
```

```
    {
        cout << "Simpul " << i << ": ";
        cin >> simpul[i];
    }

    cout << "Masukkan bobot antara simpul-simpul:" << endl;
    for (int i = 0; i < jumlahSimpul; i++)
    {
        for (int j = 0; j < jumlahSimpul; j++)
        {
              cout << "Bobot antara " << simpul[i] << " dan " <<
simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }
}

int main()
{
    int MuhammadIrsyad_2211102048;
    cout << "Masukkan jumlah simpul: ";
    cin >> MuhammadIrsyad_2211102048;

    if (MuhammadIrsyad_2211102048 > MAX_SIMPUL)
    {
        cout << "Jumlah simpul melebihi batas maksimum." << endl;
        return 0;
    }

    string simpul[MAX_SIMPUL];
    int busur[MAX_SIMPUL][MAX_SIMPUL];

    inputGraph(simpul, busur, MuhammadIrsyad_2211102048);
```

```
    tampilGraph(simpul, busur, MuhammadIrsyad_2211102048);


    return 0;
}
```

Screenshot Program



Deskripsi Program

Kode program di atas adalah implementasi sederhana dari representasi grafik menggunakan matriks ketetanggaan. Program ini memungkinkan pengguna untuk memasukkan simpul-simpul dan bobot antara simpul-simpul dalam sebuah grafik, dan kemudian menampilkan grafik tersebut.

Berikut adalah penjelasan rinci mengenai setiap bagian program:

- Library:
    - `<iostream>`: Library input-output standar yang digunakan untuk operasi masukan dan keluaran.
    - `<iomanip>`: Library yang digunakan untuk mengatur format output.
- Konstanta:
    - `MAX_SIMPUL`: Konstanta yang menentukan batas maksimum jumlah simpul dalam grafik. Dalam program ini, nilainya adalah 7.
- Fungsi `tampilGraph()`:
    - Menampilkan grafik yang telah dimasukkan oleh pengguna.

    - Menerima argumen berupa array `simpul[]` yang berisi nama-nama simpul, matriks `busur[][]` yang berisi bobot antara simpul-simpul, dan `jumlahSimpul` yang merupakan jumlah simpul dalam grafik.
    - Melakukan iterasi untuk setiap simpul dan mencetak nama simpul beserta bobot antara simpul tersebut dengan simpul lain yang terhubung.
    - Bobot yang tidak nol menunjukkan adanya hubungan antara simpul-simpul tersebut.
- Fungsi `inputGraph()`:
    - Meminta pengguna untuk memasukkan nama-nama simpul dan bobot antara simpul-simpul dalam grafik.
    - Menerima argumen yang sama dengan fungsi `tampilGraph()`.
    - Melakukan iterasi untuk setiap simpul dan meminta pengguna memasukkan nama simpul.
    - Melakukan iterasi untuk setiap pasangan simpul dan meminta pengguna memasukkan bobot antara simpul-simpul tersebut.
- Fungsi `main()`:
    - Fungsi utama program.
    - Meminta pengguna memasukkan jumlah simpul dalam grafik.
    - Memeriksa apakah jumlah simpul melebihi batas maksimum `MAX_SIMPUL`. Jika melebihi, program memberikan pesan kesalahan dan berakhir.
    - Membuat array `simpul[]` yang akan menampung nama-nama simpul dan matriks `busur[][]` yang akan menampung bobot antara simpul-simpul.
    - Memanggil fungsi `inputGraph()` untuk memasukkan nama simpul dan bobot antara simpul-simpul.

- Memanggil fungsi `tampilGraph()` untuk menampilkan grafik yang telah dimasukkan.

Dengan demikian, program ini memungkinkan pengguna untuk memasukkan grafik menggunakan matriks ketetanggaan dan menampilkan grafik tersebut sesuai dengan input pengguna.

Unguided 2

Source Code

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

struct TreeNode
{
    char MuhammadIrsyad_2211102048;
    TreeNode *left;
    TreeNode *right;
};

TreeNode *root;

void createTree(char data)
{
    if (root != NULL)
    {
        cout << "Pohon sudah dibuat." << endl;
        return;
    }

    root = new TreeNode;
    root->MuhammadIrsyad_2211102048 = data;
    root->left = NULL;
    root->right = NULL;
```

```
      cout << "Pohon dibuat dengan simpul akar: " << root->MuhammadIrsyad_2211102048 << endl;
}

TreeNode *insertLeft(char data, TreeNode *node)
{
    if (node == NULL)
    {
        cout << "Simpul induk tidak valid." << endl;
        return NULL;
    }

    if (node->left != NULL)
    {
        cout << "Simpul sudah memiliki anak kiri." << endl;
        return NULL;
    }

    TreeNode *newNode = new TreeNode;
    newNode->MuhammadIrsyad_2211102048 = data;
    newNode->left = NULL;
    newNode->right = NULL;

    node->left = newNode;
     cout << "Simpul " << data << " ditambahkan sebagai anak kiri dari " << node->MuhammadIrsyad_2211102048 << endl;

    return newNode;
}

TreeNode *insertRight(char data, TreeNode *node)
{
    if (node == NULL)
    {
```

```
        cout << "Simpul induk tidak valid." << endl;
        return NULL;
    }

    if (node->right != NULL)
    {
        cout << "Simpul sudah memiliki anak kanan." << endl;
        return NULL;
    }

    TreeNode *newNode = new TreeNode;
    newNode->MuhammadIrsyad_2211102048 = data;
    newNode->left = NULL;
    newNode->right = NULL;

    node->right = newNode;
    cout << "Simpul " << data << " ditambahkan sebagai anak kanan
dari " << node->MuhammadIrsyad_2211102048 << endl;

    return newNode;
}

void updateNode(char newData, TreeNode *node)
{
    if (node == NULL)
    {
        cout << "Simpul tidak valid." << endl;
        return;
    }

    node->MuhammadIrsyad_2211102048 = newData;
    cout << "Data simpul diperbarui menjadi " << newData << endl;
}
```

```
void traversePreOrder(TreeNode *node)
{
    if (node == NULL)
        return;

    cout << node->MuhammadIrsyad_2211102048 << " ";
    traversePreOrder(node->left);
    traversePreOrder(node->right);
}

void traverseInOrder(TreeNode *node)
{
    if (node == NULL)
        return;

    traverseInOrder(node->left);
    cout << node->MuhammadIrsyad_2211102048 << " ";
    traverseInOrder(node->right);
}

void traversePostOrder(TreeNode *node)
{
    if (node == NULL)
        return;

    traversePostOrder(node->left);
    traversePostOrder(node->right);
    cout << node->MuhammadIrsyad_2211102048 << " ";
}

int calculateSize(TreeNode *node)
{
    if (node == NULL)
        return 0;
```

```
     return 1 + calculateSize(node->left) + calculateSize(node->right);
}

int calculateHeight(TreeNode *node)
{
    if (node == NULL)
        return 0;

    int leftHeight = calculateHeight(node->left);
    int rightHeight = calculateHeight(node->right);

    return (leftHeight > rightHeight) ? leftHeight + 1 : rightHeight + 1;
}

void displayData(TreeNode *node, int level)
{
    if (node != NULL)
    {
        displayData(node->right, level + 1);

        for (int i = 0; i < level; i++)
            cout << "     ";
        cout << node->MuhammadIrsyad_2211102048 << endl;

        displayData(node->left, level + 1);
    }
}

void displayData()
{
    cout << "Data yang ada dalam pohon:" << endl;
```

```
    displayData(root, 0);
    cout << endl;
}

int main()
{
    root = NULL;

    char rootData;
    cout << "Masukkan data untuk simpul akar: ";
    cin >> rootData;
    createTree(rootData);

    char choice;
    do
    {
        cout << "\nMENU\n";
        cout << "1. Tambahkan anak kiri\n";
        cout << "2. Tambahkan anak kanan\n";
        cout << "3. Perbarui data simpul\n";
        cout << "4. Traversal pre-order\n";
        cout << "5. Traversal in-order\n";
        cout << "6. Traversal post-order\n";
        cout << "7. Tampilkan data pohon\n";
        cout << "8. Keluar\n";
        cout << "Masukkan pilihan Anda: ";
        cin >> choice;

        switch (choice)
        {
        case '1':
        {
            char parentData, newData;
            cout << "Masukkan data simpul induk: ";
```

```
            cin >> parentData;
            cout << "Masukkan data simpul baru: ";
            cin >> newData;

            TreeNode *parent = root;
            while (parent != NULL)
            {
                if (parent->MuhammadIrsyad_2211102048 == parentData)
                    break;

                    if (parent->left != NULL && parent->left-
>MuhammadIrsyad_2211102048 == parentData)
                {
                    parent = parent->left;
                    break;
                }
                  else if (parent->right != NULL && parent->right-
>MuhammadIrsyad_2211102048 == parentData)
                {
                    parent = parent->right;
                    break;
                }

                if (parent->left != NULL)
                    parent = parent->left;
                else if (parent->right != NULL)
                    parent = parent->right;
                else
                    parent = NULL;
            }

            if (parent != NULL)
                insertLeft(newData, parent);
            else
```

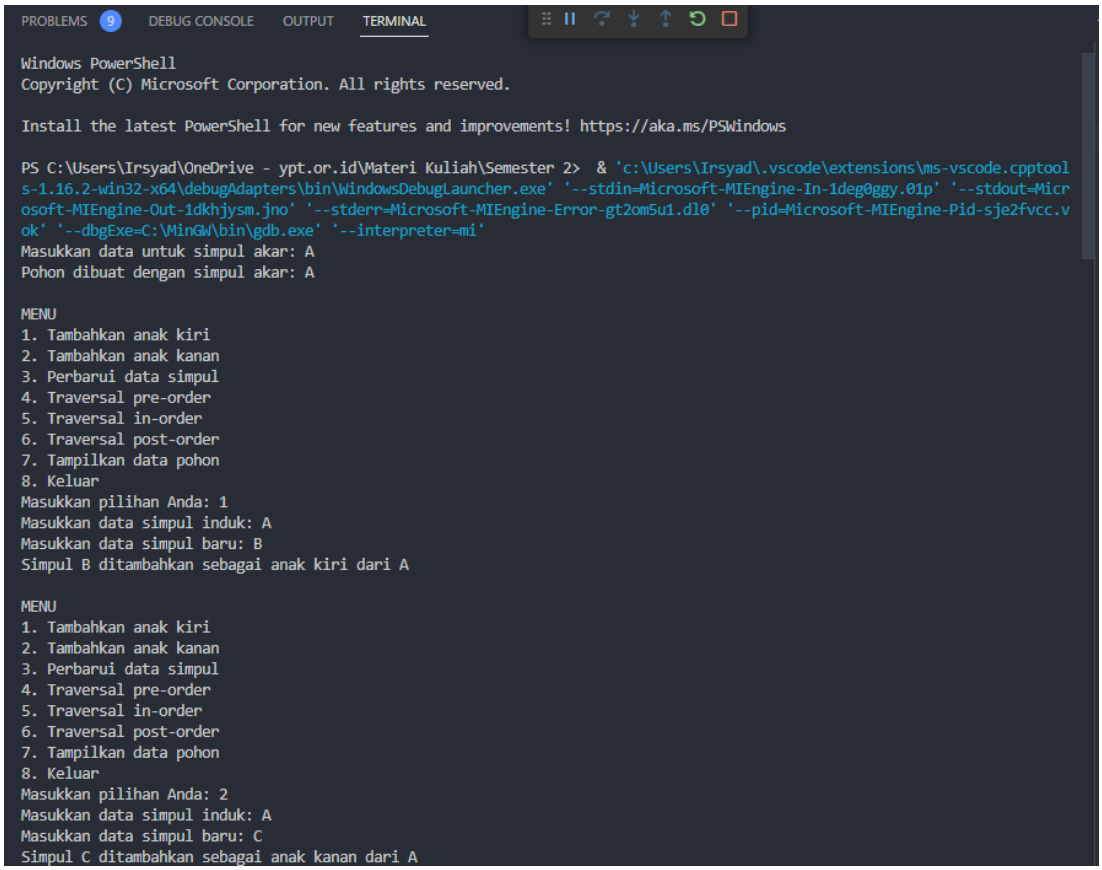
```
                cout << "Simpul induk tidak ditemukan." << endl;

            break;
        }
        case '2':
        {
            char parentData, newData;
            cout << "Masukkan data simpul induk: ";
            cin >> parentData;
            cout << "Masukkan data simpul baru: ";
            cin >> newData;

            TreeNode *parent = root;
            while (parent != NULL)
            {
               if (parent->MuhammadIrsyad_2211102048 == parentData)
                    break;

                      if (parent->left != NULL && parent->left-
>MuhammadIrsyad_2211102048 == parentData)
                {
                    parent = parent->left;
                    break;
                }
                  else if (parent->right != NULL && parent->right-
>MuhammadIrsyad_2211102048 == parentData)
                {
                    parent = parent->right;
                    break;
                }

                if (parent->left != NULL)
                    parent = parent->left;
                else if (parent->right != NULL)
```

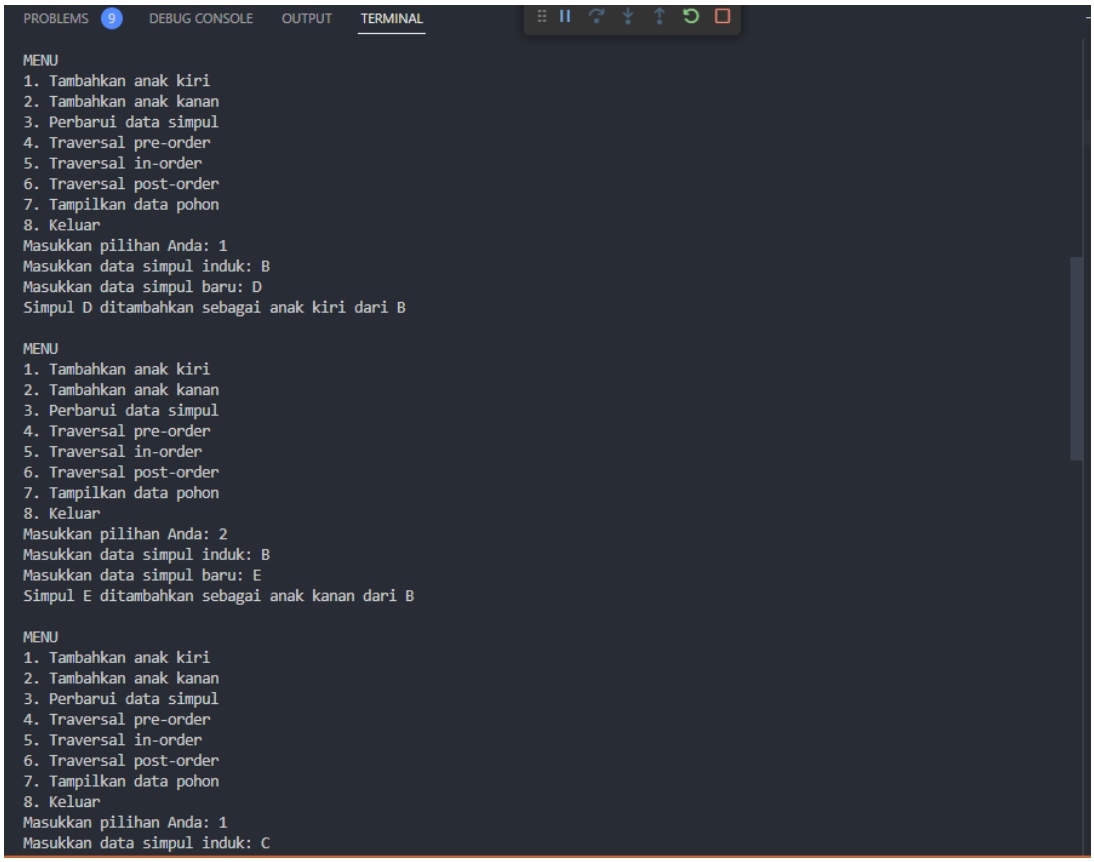
```
                    parent = parent->right;
                else
                    parent = NULL;
            }

            if (parent != NULL)
                insertRight(newData, parent);
            else
                cout << "Simpul induk tidak ditemukan." << endl;

            break;
        }
        case '3':
        {
            char nodeData, newData;
            cout << "Masukkan data simpul yang akan diperbarui: ";
            cin >> nodeData;
            cout << "Masukkan data baru: ";
            cin >> newData;

            TreeNode *node = root;
            while (node != NULL)
            {
                if (node->MuhammadIrsyad_2211102048 == nodeData)
                    break;

                    if (node->left != NULL && node->left-
>MuhammadIrsyad_2211102048 == nodeData)
                {
                    node = node->left;
                    break;
                }
                    else if (node->right != NULL && node->right-
>MuhammadIrsyad_2211102048 == nodeData)
```

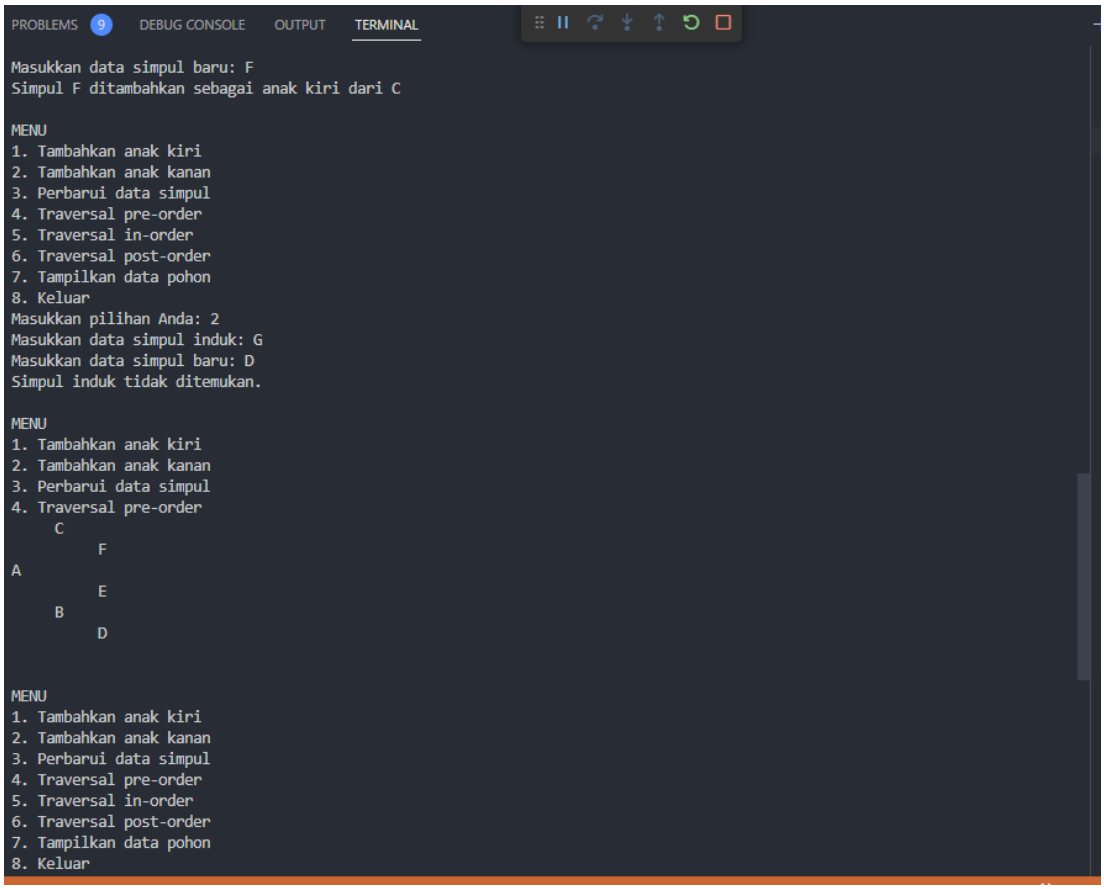
```
                {
                    node = node->right;
                    break;
                }

                if (node->left != NULL)
                    node = node->left;
                else if (node->right != NULL)
                    node = node->right;
                else
                    node = NULL;
            }

            if (node != NULL)
                updateNode(newData, node);
            else
                cout << "Simpul tidak ditemukan." << endl;

            break;
        }
        case '4':
            cout << "Traversal pre-order: ";
            traversePreOrder(root);
            cout << endl;
            break;
        case '5':
            cout << "Traversal in-order: ";
            traverseInOrder(root);
            cout << endl;
            break;
        case '6':
            cout << "Traversal post-order: ";
            traversePostOrder(root);
            cout << endl;
```

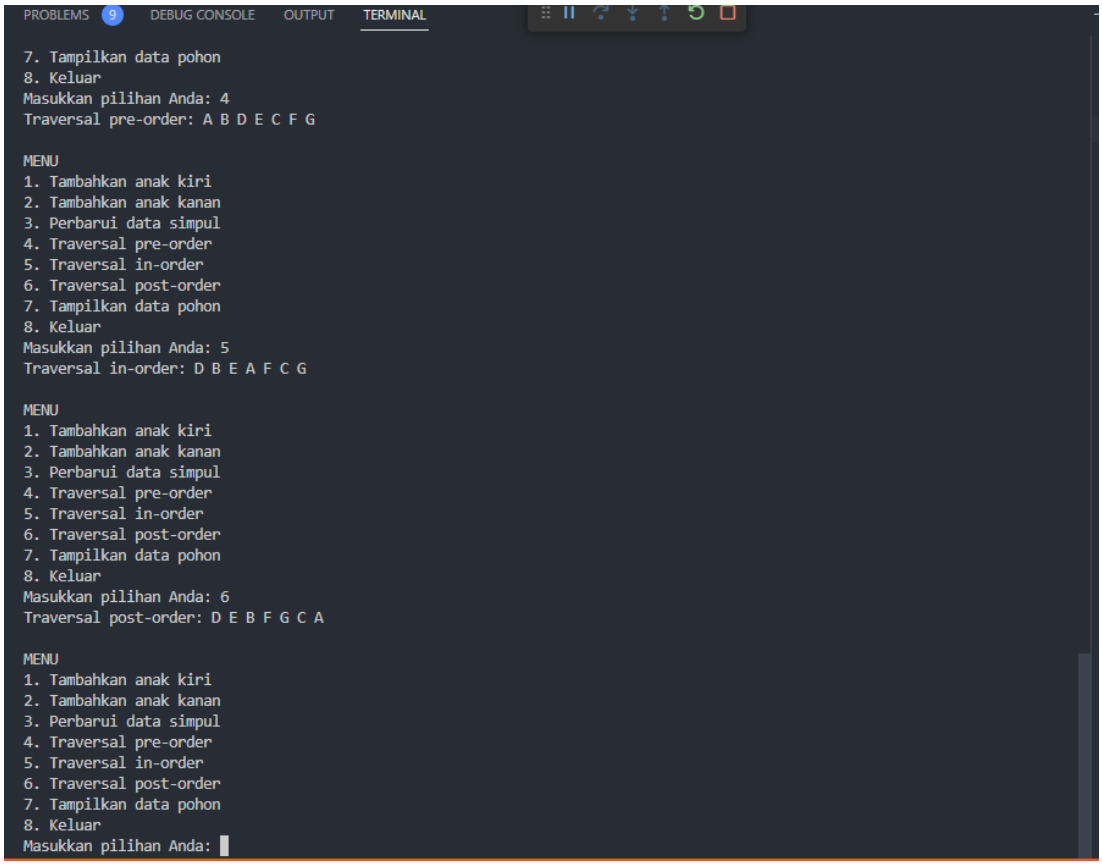
```
            break;
        case '7':
            displayData();
            break;
        case '8':
            cout << "Terima kasih!" << endl;
            break;
        default:
            cout << "Pilihan tidak valid." << endl;
        }
    } while (choice != '8');

    return 0;
}
```

Screenshot Program


PROBLEMS 9
DEBUG CONSOLE
OUTPUT
TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\Irsyad\OneDrive - ypt.or.id\Materi Kuliah\Semester 2> & 'c:\Users\Irsyad\.vscode\extensions\ms-vscode.cpptools-1.16.2-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1deg0ggy.01p' '--stdout=Microsoft-MIEngine-Out-1dkhjysm.jno' '--stderr=Microsoft-MIEngine-Error-gt2om5u1.dl0' '--pid=Microsoft-MIEngine-Pid-sje2fvcc.vok' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'
Masukkan data untuk simpul akar: A
Pohon dibuat dengan simpul akar: A
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 1
Masukkan data simpul induk: A
Masukkan data simpul baru: B
Simpul B ditambahkan sebagai anak kiri dari A
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 2
Masukkan data simpul induk: A
Masukkan data simpul baru: C
Simpul C ditambahkan sebagai anak kanan dari A


PROBLEMS 9
DEBUG CONSOLE
OUTPUT
TERMINAL
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 1
Masukkan data simpul induk: B
Masukkan data simpul baru: D
Simpul D ditambahkan sebagai anak kiri dari B
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 2
Masukkan data simpul induk: B
Masukkan data simpul baru: E
Simpul E ditambahkan sebagai anak kanan dari B
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 1
Masukkan data simpul induk: C


PROBLEMS 9
DEBUG CONSOLE
OUTPUT
TERMINAL
Masukkan data simpul baru: F
Simpul F ditambahkan sebagai anak kiri dari C
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar
Masukkan pilihan Anda: 2
Masukkan data simpul induk: G
Masukkan data simpul baru: D
Simpul induk tidak ditemukan.
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
C
F
A
E
B
D
MENU
1. Tambahkan anak kiri
2. Tambahkan anak kanan
3. Perbarui data simpul
4. Traversal pre-order
5. Traversal in-order
6. Traversal post-order
7. Tampilkan data pohon
8. Keluar



Deskripsi Program

- Program tersebut merupakan implementasi sederhana dari struktur data pohon biner menggunakan bahasa pemrograman C++. Berikut adalah deskripsi singkat dari kode program tersebut:

- Program dimulai dengan mendefinisikan struktur TreeNode yang memiliki tiga anggota:
    - char MuhammadIrsyad_2211102048: Menyimpan data karakter untuk simpul pohon.
    - TreeNode *left: Menunjuk ke simpul anak kiri.
    - TreeNode *right: Menunjuk ke simpul anak kanan.

- Deklarasi variabel global root bertipe TreeNode*, yang akan menunjukkan ke simpul akar dari pohon.

- Terdapat beberapa fungsi yang digunakan dalam program ini:
    - createTree(char data): Membuat pohon baru dengan simpul akar berisi data yang diberikan.
    - insertLeft(char data, TreeNode *node): Menambahkan simpul anak kiri dengan data yang diberikan ke simpul yang ditunjukkan oleh node.
    - insertRight(char data, TreeNode *node): Menambahkan simpul anak kanan dengan data yang diberikan ke simpul yang ditunjukkan oleh node.
    - updateNode(char newData, TreeNode *node): Memperbarui data simpul yang ditunjukkan oleh node dengan data baru yang diberikan.
    - traversePreOrder(TreeNode *node): Melakukan traversal pre-order pada pohon yang dimulai dari simpul yang ditunjukkan oleh node.
    - traverseInOrder(TreeNode *node): Melakukan traversal in-order pada pohon yang dimulai dari simpul yang ditunjukkan oleh node.
    - traversePostOrder(TreeNode *node): Melakukan traversal post-order pada pohon yang dimulai dari simpul yang ditunjukkan oleh node.
    - calculateSize(TreeNode *node): Menghitung jumlah simpul dalam pohon yang dimulai dari simpul yang ditunjukkan oleh node.
    - calculateHeight(TreeNode *node): Menghitung tinggi pohon yang dimulai dari simpul yang ditunjukkan oleh node.
    - displayData(TreeNode *node, int level): Menampilkan data dalam pohon secara terurut, dengan tingkat yang ditentukan untuk setiap simpul.
    - displayData(): Memanggil displayData(root, 0) untuk menampilkan data dalam pohon secara keseluruhan.

- Di dalam fungsi main(), program meminta pengguna memasukkan data untuk simpul akar dan kemudian membuat pohon dengan data tersebut menggunakan fungsi createTree().

- Program menyediakan menu dengan pilihan berikut:
    - Tambahkan anak kiri: Meminta pengguna memasukkan data simpul induk dan data simpul baru, kemudian menambahkan simpul baru sebagai anak kiri dari simpul induk yang sesuai.
    - Tambahkan anak kanan: Meminta pengguna memasukkan data simpul induk dan data simpul baru, kemudian menambahkan simpul baru sebagai anak kanan dari simpul induk yang sesuai.
    - Perbarui data simpul: Meminta pengguna memasukkan data simpul yang akan diperbarui dan data baru, kemudian memperbarui data simpul yang sesuai.
    - Traversal pre-order: Menampilkan data dalam pohon menggunakan traversal pre-order.
    - Traversal in-order: Menampilkan data dalam pohon menggunakan traversal in-order.
    - Traversal post-order: Menampilkan data dalam pohon menggunakan traversal post-order.
    - Tampilkan data pohon: Menampilkan data dalam pohon secara terurut.

    - Keluar: Mengakhiri program.

- Program menggunakan switch-case untuk memproses pilihan pengguna dan memanggil fungsi yang sesuai.

- Program akan terus menampilkan menu dan meminta input pengguna sampai pengguna memilih untuk keluar dengan memilih pilihan 8.

- Ketika program berakhir, memori yang dialokasikan untuk simpul-simpul pohon akan dibebaskan.

- Program ini memungkinkan pengguna untuk membuat pohon biner, menambahkan simpul anak kiri dan anak kanan, memperbarui data simpul, serta melakukan traversal dan tampilan data dalam pohon.

BAB IV

KESIMPULAN

Kesimpulan dari modul ini adalah sebuah implementasi dari Binary Tree menggunakan bahasa pemrograman C++. Program ini menyediakan berbagai fungsi untuk mengelola pohon biner, termasuk pembuatan node, penambahan node anak kiri dan anak kanan, pembaruan data node, penelusuran pohon, penghapusan pohon, perhitungan ukuran dan tinggi pohon, serta tampilan karakteristik pohon.

Dengan menggunakan struktur data pohon biner, program ini memungkinkan pengguna untuk membuat dan mengelola struktur data pohon biner dengan mudah. Beberapa fungsi yang disediakan, seperti penelusuran pohon dan perhitungan karakteristik pohon, memungkinkan pengguna untuk memahami struktur dan informasi yang terkandung dalam pohon.

Kode program ini memberikan fleksibilitas kepada pengguna untuk melakukan berbagai operasi pada pohon biner, termasuk menambahkan dan menghapus node, memperbarui data node, serta menampilkan informasi yang berguna tentang pohon tersebut. Dengan adanya fungsi-fungsi tersebut, program ini dapat digunakan untuk

berbagai keperluan, seperti pemodelan struktur data, analisis data, atau implementasi algoritma yang melibatkan pohon biner.

Keseluruhan, kode program ini memberikan pengguna sebuah kerangka kerja yang solid untuk mengelola pohon biner dan melakukan operasi pada struktur data tersebut dengan mudah.