LAPORAN PRAKTIKUM

MODUL V

HASH TABLE

Disusun oleh:

Reli Gita Nurhidayati

NIM: 2311102025

Dosen Pengampu:

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2024

BAB 1

TUJUAN PRAKTIKUM

Tujuan Praktikum

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

BAB II

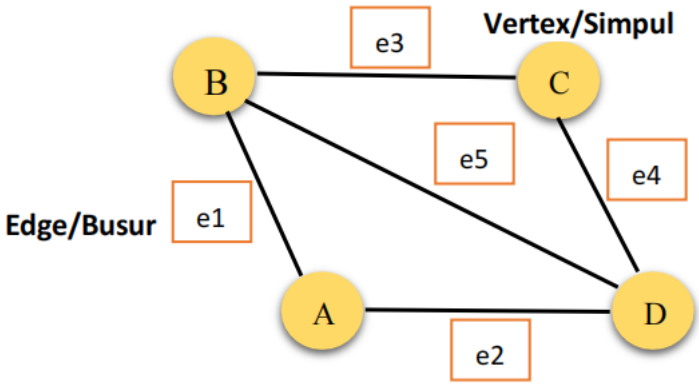
DASAR TEORI

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 :

$$G = (V, E)$$

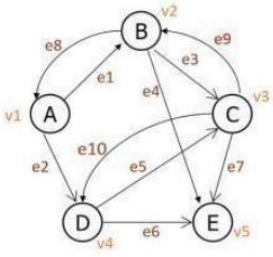
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:



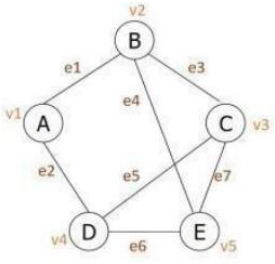
Gambar 1 Contoh Graph

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

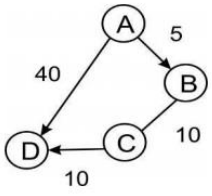
Jenis-jenis Graph:



Directed Graph



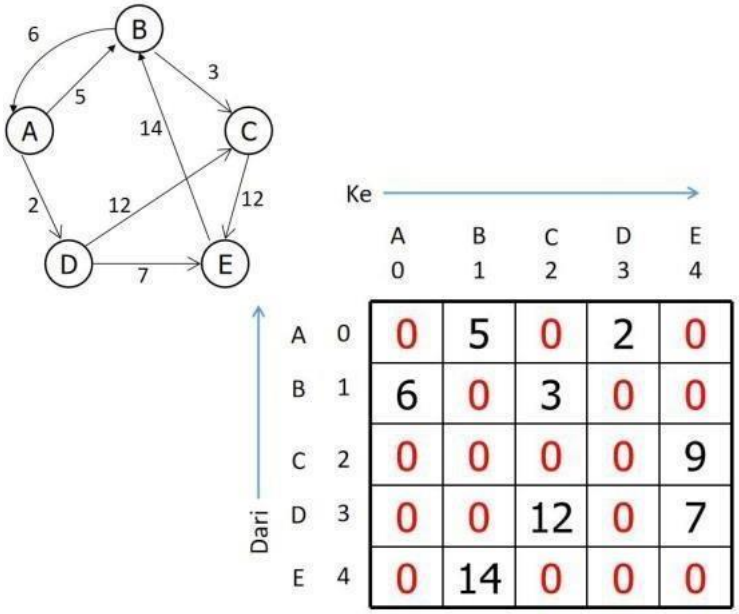
Undirected Graph



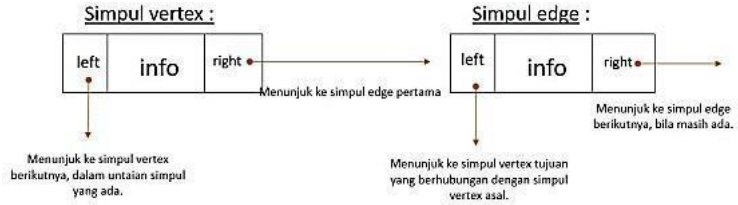
Weight Graph

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

Representasi Graph dengan Matriks

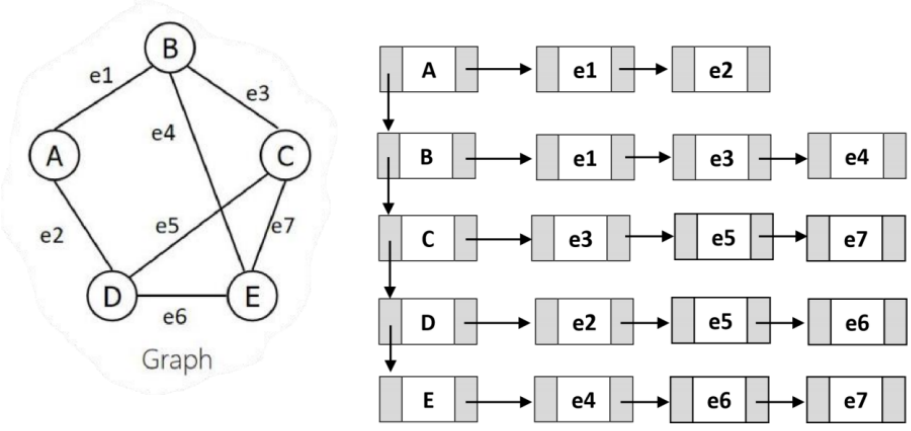


Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

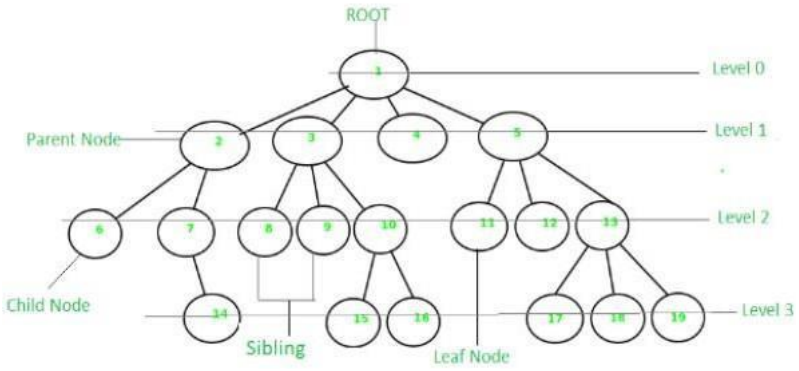
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, 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 :

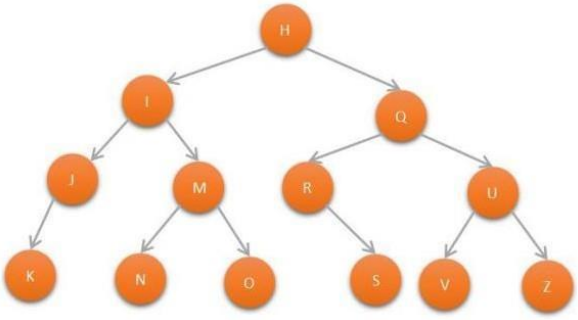


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor 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.

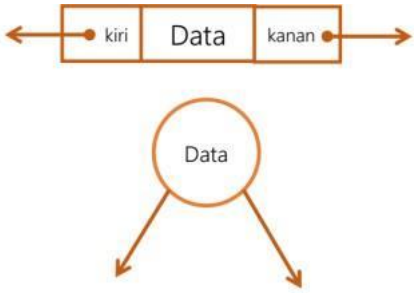
Gambar 1, menunjukkan contoh dari struktur data binary tree.



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;
```



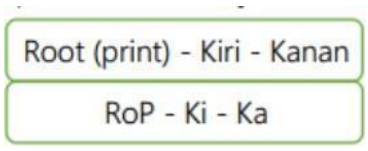
Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

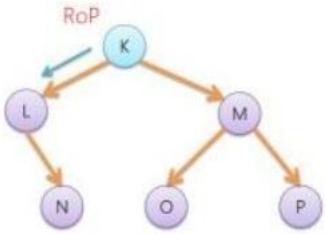
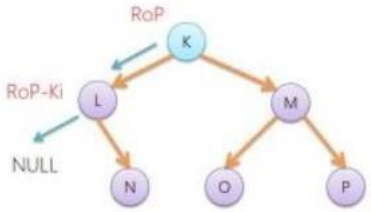
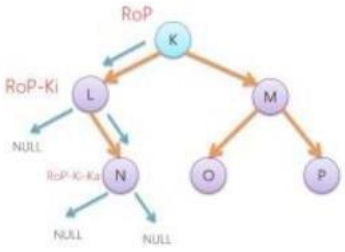
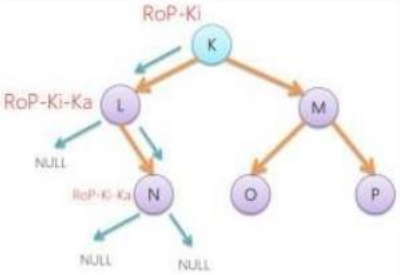
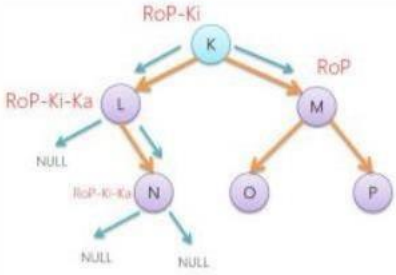
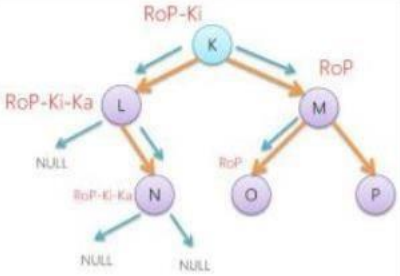
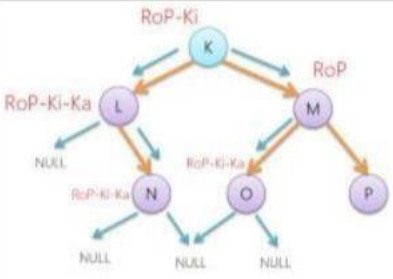
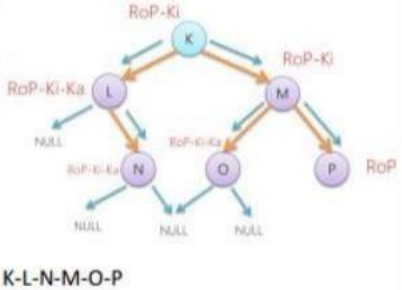
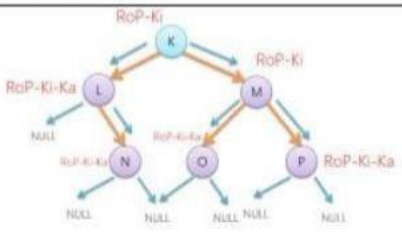
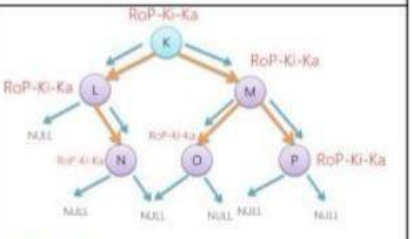
a. Create: digunakan untuk membentuk binary tree baru yang masih kosong.
b. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
c. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
d. Insert: digunakan untuk memasukkan sebuah node kedalam tree.
e. Find: digunakan untuk mencari root, parent, left child, atau right child darisuatu node dengan syarat tree tidak boleh kosong**.**
f. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong
g. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
h. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
i. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
j. 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:



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:

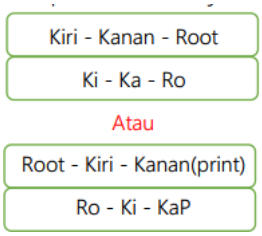
3. Post Order

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

GUIDED

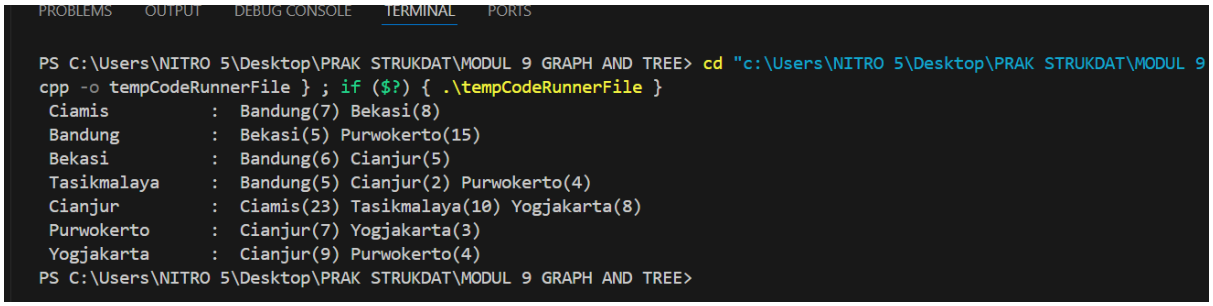
Program Graph

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;
}
```

Screenshoot Program



Deskripsi Program

Program ini adalah program C++ yang menampilkan representasi graf dengan menggunakan matriks ketetanggaan (adjacency matrix). Tujuan utama dari program ini adalah menampilkan daftar kota (simpul) dan jarak antara kota-kota tersebut (busur) dalam bentuk matriks.

Ketika Program dijalankan, outputnya akan berupa daftar kota-kota dan kota-kota yang dapat dijangkau secara langsung dari setiap kota beserta jarak antara keduanya. Ini membantu dalam melihat hubungan antara kota-kota dalam bentuk graf.

Program Tree

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;
    else
        return 0;
    // true
    // 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();
}
```

Screenshoot Program

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 9 GRAPH AND TREE> cd "c:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 9
cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

 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\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 9 GRAPH AND TREE>
```

Deskripsi Program:

Program c++ berikut ini mendefinisikan struktur `Pohon` yang terdiri dari data bertipe `char` dan tiga pointer yaitu `left`, `right`, dan `parent` yang digunakan untuk menghubungkan setiap node dalam pohon.

Program ini dapat digunakan untuk memahami konsep pohon biner dan mengimplementasikan operasi-operasi dasar pada struktur data tersebut dalam bahasa C++.

BAB IV

UNGUIDED

1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

Output Program

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukkan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

         BALI     PALU
  BALI      0     3
  PALU      4     0

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.
```

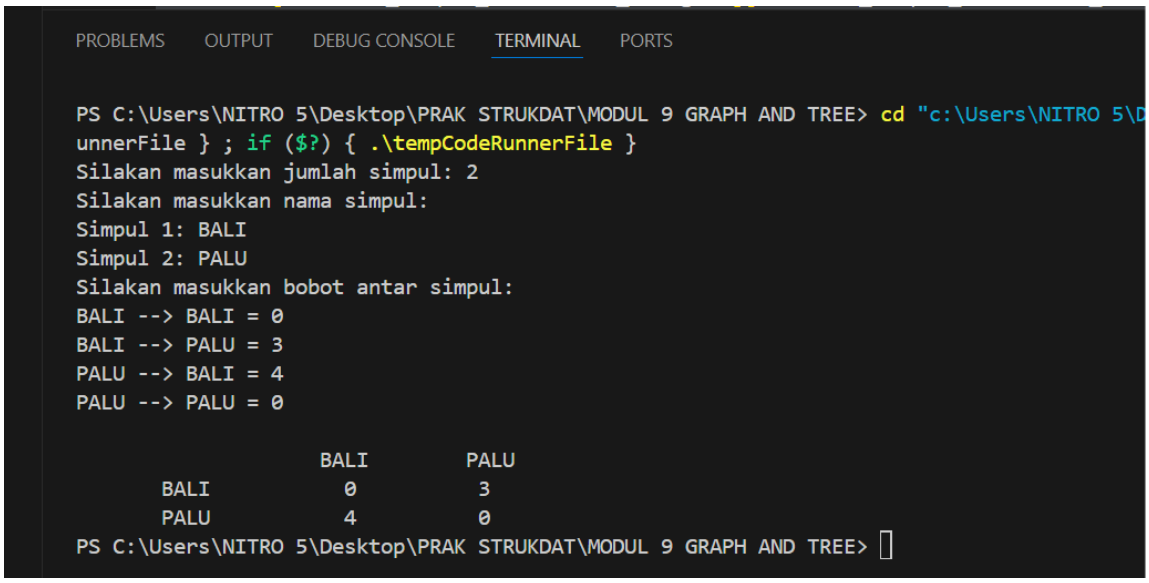
Source Code

```
#include <iostream> // Library untuk input-output standar
#include <iomanip> // Library untuk manipulasi output
#include <string> // Library untuk menggunakan tipe data string
using namespace std;

const int Maksimal_Simpul_2311102025_Religita = 2; // Konstanta untuk jumlah
maksimal simpul

// Deklarasi array string yang berisi nama-nama kota
string simpul_025[Maksimal_Simpul_2311102025_Religita];

// Deklarasi matriks busur yang berisi bobot dari setiap busur antar simpul
(nama kota)
int
busur[Maksimal_Simpul_2311102025_Religita][Maksimal_Simpul_2311102025_Religita
];

// Fungsi untuk menampilkan graf
void tampilGraph_2311102025_Religita ()
{
    cout << setw(12) << " "; // Menampilkan spasi kosong dengan lebar 12 kolom
    for (int i = 0; i < Maksimal_Simpul_2311102025_Religita; i++)
    {
        cout << setw(12) << simpul_025[i]; // Menampilkan nama simpul dengan
lebar 12 kolom
    }
    cout << endl; // Pindah ke baris baru

    // Menampilkan isi graf
```

```
    for (int i = 0; i < Maksimal_Simpul_2311102025_Religita; i++)
    {
        cout << setw(11) << simpul_025[i] << " "; // Menampilkan nama simpul
dengan lebar 12 kolom
        for (int j = 0; j < Maksimal_Simpul_2311102025_Religita; j++)
        {
            cout << setw(11) << busur[i][j]; // Menampilkan bobot busur dengan
lebar 12 kolom
        }
        cout << endl; // Pindah ke baris baru
    }
}

int main()
{
    cout << "Silakan masukkan jumlah simpul: "; // Meminta pengguna untuk
memasukkan jumlah simpul
    int numVertices;
    cin >> numVertices; // Membaca jumlah simpul dari input pengguna

    if (numVertices != Maksimal_Simpul_2311102025_Religita)
    {
        cout << "Jumlah simpul tidak sesuai dengan konstanta MAX_VERTICES." <<
endl; // Memberi pesan jika jumlah simpul tidak sesuai dengan konstanta
        return 1; // Menghentikan program dengan kode kesalahan
    }

    cout << "Silakan masukkan nama simpul:\n"; // Meminta pengguna untuk
memasukkan nama simpul
    for (int i = 0; i < numVertices; ++i)
    {
        cout << "Simpul " << i + 1 << ": "; // Menampilkan pesan untuk
memasukkan nama simpul
        cin >> simpul_025[i]; // Membaca nama simpul dari input pengguna
    }

    cout << "Silakan masukkan bobot antar simpul:\n"; // Meminta pengguna
untuk memasukkan bobot antar simpul
    for (int i = 0; i < numVertices; ++i)
    {
        for (int j = 0; j < numVertices; ++j)
        {
            cout << simpul_025[i] << " --> " << simpul_025[j] << " = "; //
Menampilkan pesan untuk memasukkan bobot busur
            cin >> busur[i][j]; // Membaca bobot busur dari input pengguna
        }
    }
```

```
    cout << endl; // Pindah ke baris baru
    tampilGraph_2311102025_Religita(); // Memanggil fungsi untuk menampilkan
graf

    return 0; // Mengembalikan nilai 0 yang menandakan program berakhir dengan
sukses
}
```

Screenshoot Program



Deskripsi Program

Program C++ ini adalah implementasi dari sebuah graf yang merepresentasikan jarak/bobot antar kota-kota. Berikut adalah deskripsi dari program tersebut:

Program mendefinisikan konstanta `Maksimal_Simpul_2311102025_Religita` yang menyatakan jumlah maksimal simpul (kota) yang dapat ditampung di dalam program.

Program ini memungkinkan pengguna untuk merepresentasikan jarak atau bobot antar kota-kota dalam bentuk graf dengan menggunakan matriks ketetanggaan (adjacency matrix). Program dapat digunakan untuk memodelkan berbagai masalah yang melibatkan jaringan kota atau lokasi, seperti transportasi, logistik, atau jaringan komunikasi.

2. Modifikasi unguided tree diatas dengan program menu menggunakan input data

tree dari user!

Source Code

```
#include <iostream> // Memasukkan library iostream untuk input-output
#include <iomanip>  // Memasukkan library iomanip untuk manipulasi input-
output
using namespace std;

// Struktur data Pohon untuk merepresentasikan node dalam tree
struct Pohon
{
    char data_gita_2311102025; // Data yang disimpan di node
    Pohon *left;               // Pointer ke anak kiri
    Pohon *right;              // Pointer ke anak kanan
    Pohon *parent;             // Pointer ke parent (induk)
};

// Deklarasi pointer root dan baru untuk tree
Pohon *root, *baru;

// Fungsi untuk menginisialisasi tree
void init()
{
    root = NULL; // Mengatur root menjadi NULL
}

// Fungsi untuk memeriksa apakah tree kosong
bool isEmpty()
{
    return root == NULL; // Mengembalikan true jika root NULL (tree kosong)
}

// Fungsi untuk membuat node baru sebagai root
void buatNode(char data)
{
    if (isEmpty())
    {
        root = new Pohon();                 // Membuat node baru
        root->data_gita_2311102025 = data; // Mengisi data node
        root->left = NULL;                  // Mengatur anak kiri menjadi NULL
        root->right = NULL;                 // Mengatur anak kanan menjadi
NULL
        root->parent = NULL;                // Mengatur parent menjadi NULL
        cout << "\n Node " << data << " berhasil dibuat sebagai root." <<
endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl; // Pesan jika tree sudah ada
```

```
    }
}

// Fungsi untuk menambahkan node di anak kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL; // Mengembalikan NULL jika tree kosong
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->data_gita_2311102025 << " sudah ada
child kiri !" << endl;
            return NULL; // Mengembalikan NULL jika anak kiri sudah ada
        }
        else
        {
            Pohon *baru = new Pohon();          // Membuat node baru
            baru->data_gita_2311102025 = data; // Mengisi data node
            baru->left = NULL;                  // Mengatur anak kiri menjadi
NULL
            baru->right = NULL;                 // Mengatur anak kanan menjadi
NULL
            baru->parent = node;                // Mengatur parent menjadi
node
            node->left = baru;                  // Mengatur anak kiri dari
node
            cout << "\n Node " << data << " berhasil ditambahkan ke child kiri
" << baru->parent->data_gita_2311102025 << endl;
            return baru; // Mengembalikan pointer ke node baru
        }
    }
}

// Fungsi untuk menambahkan node di anak kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL; // Mengembalikan NULL jika tree kosong
    }
    else
    {
```

```
        if (node->right != NULL)
        {
            cout << "\n Node " << node->data_gita_2311102025  << " sudah ada
child kanan !" << endl;
            return NULL; // Mengembalikan NULL jika anak kanan sudah ada
        }
        else
        {
            Pohon *baru = new Pohon();          // Membuat node baru
            baru->data_gita_2311102025 = data; // Mengisi data node
            baru->left = NULL;                  // Mengatur anak kiri menjadi
NULL
            baru->right = NULL;                 // Mengatur anak kanan menjadi
NULL
            baru->parent = node;                // Mengatur parent menjadi
node
            node->right = baru;                 // Mengatur anak kanan dari
node
            cout << "\n Node " << data << " berhasil ditambahkan ke child
kanan " << baru->parent->data_gita_2311102025  << endl;
            return baru; // Mengembalikan pointer ke node baru
        }
    }
}

// Fungsi untuk mengupdate data di node
void update(char data, Pohon *node)
{
    if (isEmpty()) // Memeriksa apakah tree kosong/tersedia
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Menampilkan pesan
jika tree kosong
    }
    else // Jika tree tidak kosong
    {
        if (!node) // Memeriksa apakah node yang ditunjuk ada
        {
            cout << "\n Node yang ingin diganti tidak ada!!" << endl; //
Menampilkan pesan jika node tidak ada
        }
        else // Jika node ada
        {
            char temp = node->data_gita_2311102025
;                                   // Menyimpan data lama
            node->data_gita_2311102025  =
data;                                       // Mengganti data dengan data
baru
```

```
            cout << "\n Node " << temp << " berhasil diubah menjadi " << data
<< endl; // Menampilkan pesan sukses
        }
    }
}

// Fungsi untuk menampilkan data di node
void retrieve(Pohon *node) // Fungsi retrieve dengan parameter pointer ke node
dari Pohon
{
    if (isEmpty()) // Mengecek apakah tree kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Jika tree kosong,
tampilkan pesan
    }
    else // Jika tree tidak kosong
    {
        if (!node) // Mengecek apakah node yang ditunjuk tidak ada (NULL)
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl; // Jika node
tidak ada, tampilkan pesan
        }
        else // Jika node ada
        {
            cout << "\n Data node : " << node->data_gita_2311102025  << endl;
// Tampilkan data node yang ditunjuk
        }
    }
}

void find(Pohon *node) // Fungsi untuk menemukan dan menampilkan informasi
tentang node dalam pohon
{
    if (isEmpty()) // Periksa apakah pohon kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Tampilkan pesan
jika pohon kosong
    }
    else
    {
        if (!node) // Jika node yang ditunjuk NULL
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl; // Tampilkan
pesan bahwa node tidak ada
        }
        else
        {
```

```
            cout << "\n Data Node : " << node->data_gita_2311102025 << endl;
// Tampilkan data dari node yang ditunjuk
            cout << " Root : " << root->data_gita_2311102025  <<
endl;         // Tampilkan data dari root node

            // Tampilkan informasi tentang sibling node
            if (node->parent != NULL && node->parent->left != node && node-
>parent->right == node)
                cout << " Sibling : " << node->parent->left-
>data_gita_2311102025  << endl;
            else if (node->parent != NULL && node->parent->right != node &&
node->parent->left == node)
                cout << " Sibling : " << node->parent->right-
>data_gita_2311102025  << endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;

            // Tampilkan informasi tentang child node kiri
            if (!node->left)
                cout << " Child Kiri : (tidak punya Child kiri)" << endl;
            else
                cout << " Child Kiri : " << node->left-
>data_gita_2311102025  << endl;

            // Tampilkan informasi tentang child node kanan
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right-
>data_gita_2311102025  << endl;
        }
    }
}

void preOrder(Pohon *node = root) // Fungsi untuk melakukan traversal PreOrder
pada pohon
{
    if (isEmpty()) // Periksa apakah pohon kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Tampilkan pesan
jika pohon kosong
    }
    else
    {
        if (node != NULL) // Jika node tidak NULL
        {
            cout << node->data_gita_2311102025  << ", "; // Tampilkan data
dari node
```

```
            preOrder(node->left);                          // Lakukan traversal
PreOrder pada subtree kiri
            preOrder(node->right);                         // Lakukan traversal
PreOrder pada subtree kanan
        }
    }
}

void inOrder(Pohon *node = root) // Fungsi untuk melakukan traversal InOrder
pada pohon
{
    if (isEmpty()) // Periksa apakah pohon kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Tampilkan pesan
jika pohon kosong
    }
    else
    {
        if (node != NULL) // Jika node tidak NULL
        {
            inOrder(node->left);                           // Lakukan traversal
InOrder pada subtree kiri
            cout << node->data_gita_2311102025  << ", "; // Tampilkan data
dari node
            inOrder(node->right);                          // Lakukan traversal
InOrder pada subtree kanan
        }
    }
}

void postOrder(Pohon *node = root) // Fungsi untuk melakukan traversal
PostOrder pada pohon
{
    if (isEmpty()) // Periksa apakah pohon kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Tampilkan pesan
jika pohon kosong
    }
    else
    {
        if (node != NULL) // Jika node tidak NULL
        {
            postOrder(node->left);                         // Lakukan traversal
PostOrder pada subtree kiri
            postOrder(node->right);                        // Lakukan traversal
PostOrder pada subtree kanan
            cout << node->data_gita_2311102025  << ", "; // Tampilkan data
dari node
```

```
        }
    }
}

void deleteSubtree(Pohon *node) // Fungsi untuk menghapus subtree yang dimulai
dari node yang diberikan
{
    if (node == NULL) // Jika node NULL, kembalikan
        return;

    // Hapus subtree secara rekursif
    deleteSubtree(node->left);
    deleteSubtree(node->right);

    // Hapus hubungan parent dengan node
    if (node->parent != NULL && node->parent->left == node)
    {
        node->parent->left = NULL;
    }
    else if (node->parent != NULL && node->parent->right == node)
    {
        node->parent->right = NULL;
    }

    // Tampilkan pesan bahwa node berhasil dihapus
    cout << "\n Node " << node->data_gita_2311102025  << " berhasil dihapus."
<< endl;

    // Hapus node dari memory
    delete node;
}

// Fungsi untuk menampilkan informasi tentang child dan descendant
void showChildAndDescendant(Pohon *node) // Fungsi untuk menampilkan informasi
tentang child dan descendant dari suatu node
{
    if (isEmpty()) // Periksa apakah pohon kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Tampilkan pesan
jika pohon kosong
    }
    else
    {
        if (!node) // Jika node yang ditunjuk NULL
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl; // Tampilkan
pesan bahwa node tidak ada
        }
```

```
        else
        {
            cout << "\n Data Node : " << node->data_gita_2311102025 << endl;
// Tampilkan data dari node yang ditunjuk
            if (!node-
>left)                                                  // Jika tidak ada
child kiri
            {
                cout << " Child Kiri : (tidak punya Child kiri)" << endl; //
Tampilkan pesan bahwa tidak ada child kiri
            }
            else
            {
                cout << " Child Kiri : " << node->left->data_gita_2311102025
<< endl; // Tampilkan data dari child kiri
                cout << " Descendant dari Child Kiri : ";
                inOrder(node->left); // Tampilkan descendant dari child kiri
dengan traversal InOrder
                cout << endl;
            }

            if (!node->right) // Jika tidak ada child kanan
            {
                cout << " Child Kanan : (tidak punya Child kanan)" << endl; //
Tampilkan pesan bahwa tidak ada child kanan
            }
            else
            {
                cout << " Child Kanan : " << node->right-
>data_gita_2311102025  << endl; // Tampilkan data dari child kanan
                cout << " Descendant dari Child Kanan : ";
                inOrder(node->right); // Tampilkan descendant dari child kanan
dengan traversal InOrder
                cout << endl;
            }
        }
    }
}

// Deklarasi fungsi findParent
void findParent(Pohon *node, char parentData, Pohon *&parentNode) // Fungsi
untuk mencari parent dari node dengan data tertentu
{
    if (node == NULL) // Jika node NULL, kembalikan
        return;

    if (node->data_gita_2311102025 == parentData) // Jika data node sama
dengan data parent yang dicari
```

```
    {
        parentNode = node; // Simpan node sebagai parent yang ditemukan
        return;
    }

    // Telusuri subtree secara rekursif
    findParent(node->left, parentData, parentNode);
    findParent(node->right, parentData, parentNode);
}

// Fungsi untuk menghitung ukuran tree
int size(Pohon *node = root) // Fungsi size dengan parameter default node yang
menunjuk ke root
{
    if (isEmpty()) // Mengecek apakah tree kosong
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl; // Jika tree kosong,
tampilkan pesan dan kembalikan ukuran 0
        return 0;
    }
    else // Jika tree tidak kosong
    {
        if (!node) // Mengecek apakah node NULL
        {
            return 0; // Jika node NULL, kembalikan ukuran 0
        }
        else // Jika node tidak NULL
        {
            return 1 + size(node->left) + size(node->right); // Kembalikan
jumlah node, termasuk node saat ini dan semua node di subtree kiri dan kanan
        }
    }
}


// Fungsi untuk menghitung tinggi tree
int height(Pohon *node = root) // Fungsi height dengan parameter default node
yang menunjuk ke root
{
    if (isEmpty()) // Mengecek apakah tree kosong
    {
        cout << "\n Buat tree terlebih dahulu!" << endl; // Jika tree kosong,
tampilkan pesan dan kembalikan tinggi 0
        return 0;
    }
    else // Jika tree tidak kosong
    {
        if (!node) // Mengecek apakah node NULL
        {
```

```
            return 0; // Jika node NULL, kembalikan tinggi 0
        }
        else // Jika node tidak NULL
        {
            int heightKiri = height(node->left); // Hitung tinggi subtree kiri
            int heightKanan = height(node->right); // Hitung tinggi subtree
kanan
            if (heightKiri >= heightKanan) // Bandingkan tinggi subtree kiri
dan kanan
            {
                return heightKiri + 1; // Kembalikan tinggi subtree kiri
ditambah 1
            }
            else
            {
                return heightKanan + 1; // Kembalikan tinggi subtree kanan
ditambah 1
            }
        }
    }
}
// Fungsi untuk menampilkan karakteristik tree
void characteristic() // Fungsi characteristic tanpa parameter
{
    cout << "\n Size Tree : " << size() << endl; // Tampilkan ukuran tree
menggunakan fungsi size()
    cout << " Height Tree : " << height() << endl; // Tampilkan tinggi tree
menggunakan fungsi height()
    cout << " Average Node of Tree : " << size() / height() << endl; //
Tampilkan rata-rata node dari tree
}

int main()
{
    int pil;   // Variabel untuk menyimpan pilihan menu
    char data; // Variabel untuk menyimpan data node atau parent node

    // Loop utama program
    do
    {
        cout << "\n ====================================";
        cout << "\n        ==  Program Tree C++ ==       ";
        cout << "\n ====================================";
        cout << "\n Menu :                      ";
        cout << "\n 1. Tambah Node Root";
        cout << "\n 2. Tambah Kiri";
        cout << "\n 3. Tambah Kanan";
        cout << "\n 4. Update Data";
```

```
        cout << "\n 5. Lihat/Retrive Data";
        cout << "\n 6. Cari Data";
        cout << "\n 7. Traversal PreOrder";
        cout << "\n 8. Traversal InOrder";
        cout << "\n 9. Traversal PostOrder";
        cout << "\n 10. Hapus Subtree";
        cout << "\n 11. Hapus Seluruh Tree";
        cout << "\n 12. Karakteristik Tree";
        cout << "\n 13. Tampilkan Child dan Descendant";
        cout << "\n 0. Exit";
        cout << "\n ====================================";
        cout << "\n Pilih menu : ";

        cin >> pil; // Masukkan pilihan menu

        switch (pil)
        {
        case 1: // Menu untuk menambahkan node root
            cout << "\n Input data : ";
            cin >> data;
            buatNode(data);
            break;

        case 2: // Menu untuk menambahkan node sebelah kiri
        {
            cout << "\n Input data : ";
            cin >> data;
            char parentData;
            cout << " Input parent data: ";
            cin >> parentData;
            Pohon *parentNode = NULL;

            // Cari parent node berdasarkan parentData
            findParent(root, parentData, parentNode);

            if (parentNode)
            {
                insertLeft(data, parentNode);
            }
            else
            {
                cout << "\n Parent node tidak ditemukan!" << endl;
            }
            break;
        }
        case 3: // Menu untuk menambahkan node sebelah kanan
        {
            cout << "\n Input data : ";
```

```
            cin >> data;
            char parentData;
            cout << " Input parent data: ";
            cin >> parentData;
            Pohon *parentNode = NULL;

            // Cari parent node berdasarkan parentData
            findParent(root, parentData, parentNode);

            if (parentNode)
            {
                insertRight(data, parentNode);
            }
            else
            {
                cout << "\n Parent node tidak ditemukan!" << endl;
            }
            break;
        }

        case 4: // Menu untuk mengupdate data pada node
        {
            cout << "\n Input data baru : ";                    // Meminta input
untuk data baru
            cin >> data;                                        // Membaca data
baru dari pengguna
            char nodeData;                                      // Variabel
untuk menyimpan data node yang ingin diupdate
            cout << " Input data node yang ingin diupdate: "; // Meminta input
data node yang ingin diupdate
            cin >> nodeData;                                    // Membaca data
node yang ingin diupdate dari pengguna
            Pohon *node = root;                                 // Inisialisasi
node yang akan digunakan untuk mencari node yang ingin diupdate
            // Loop untuk mencari node yang ingin diupdate berdasarkan datanya
            while (node && node->data_gita_2311102025 != nodeData)
            {
                if (node->left && node->left->data_gita_2311102025  ==
nodeData)
                {
                    node = node->left; // Jika node pada left child sesuai
dengan nodeData, pindahkan node ke left child
                }
                else if (node->right && node->right->data_gita_2311102025  ==
nodeData)
                {
                    node = node->right; // Jika node pada right child sesuai
dengan nodeData, pindahkan node ke right child
```

```
                }
                else if (node->left)
                {
                    node = node->left; // Jika node memiliki left child,
pindahkan node ke left child
                }
                else if (node->right)
                {
                    node = node->right; // Jika node memiliki right child,
pindahkan node ke right child
                }
                else
                {
                    node = NULL; // Jika tidak ada kondisi di atas yang
terpenuhi, atur node menjadi NULL (node tidak ditemukan)
                }
            }
            if (node)
            {
                update(data, node); // Jika node ditemukan, panggil fungsi
update untuk mengupdate datanya
            }
            else
            {
                cout << "\n Node tidak ditemukan!" << endl; // Jika node tidak
ditemukan, tampilkan pesan kesalahan
            }
            break; // Keluar dari case 4
        }
        case 5: // Menu untuk melihat informasi data pada node
        {
            char nodeData;                                  // Variabel untuk
menyimpan data node yang ingin dilihat
            cout << "Input data node yang ingin dilihat: "; // Meminta input
data node yang ingin dilihat
            cin >> nodeData;                                // Membaca data
node yang ingin dilihat dari pengguna

            Pohon *node = root; // Inisialisasi node yang akan digunakan untuk
mencari node yang ingin dilihat

            // Loop untuk mencari node yang ingin dilihat berdasarkan datanya
            while (node && node->data_gita_2311102025 != nodeData)
            {
                if (nodeData < node->data_gita_2311102025 )
                {
                    node = node->left; // Pindah ke left child jika data node
lebih kecil dari node saat ini
```

```
                }
                else
                {
                    node = node->right; // Pindah ke right child jika data
node lebih besar atau sama dengan node saat ini
                }
            }

            if (node)
            {
                // Tampilkan informasi tentang node dan strukturnya
                cout << "Data node: " << node->data_gita_2311102025  << endl;
            }
            else
            {
                // Jika node tidak ditemukan, tidak melakukan apa-apa
            }

            break; // Keluar dari case 5
        }

        case 6: // Menu untuk mencari data pada node
        {
            char nodeData;                                  // Variabel untuk
menyimpan data node yang ingin dicari
            cout << " Input data node yang ingin dicari: "; // Meminta input
data node yang ingin dicari
            cin >> nodeData;                                // Membaca data
node yang ingin dicari dari pengguna
            Pohon *node = root;                             // Inisialisasi
node yang akan digunakan untuk mencari node yang ingin dicari
            // Loop untuk mencari node yang ingin dicari berdasarkan datanya
            while (node && node->data_gita_2311102025  != nodeData)
            {
                if (node->left && node->left->data_gita_2311102025  ==
nodeData)
                {
                    node = node->left; // Jika node pada left child sesuai
dengan nodeData, pindahkan node ke left child
                }
                else if (node->right && node->right->data_gita_2311102025 ==
nodeData)
                {
                    node = node->right; // Jika node pada right child sesuai
dengan nodeData, pindahkan node ke right child
                }
                else if (node->left)
                {
```

```
                    node = node->left; // Jika node memiliki left child, pindahkan node ke left child
                }
                else if (node->right)
                {
                    node = node->right; // Jika node memiliki right child, pindahkan node ke right child
                }
                else
                {
                    node = NULL; // Jika tidak ada kondisi di atas yang terpenuhi, atur node menjadi NULL (node tidak ditemukan)
                }
            }
            if (node)
            {
                // Panggil fungsi find untuk menampilkan informasi tentang node
                find(node);
            }
            else
            {
                cout << "\n Node tidak ditemukan!" << endl; // Jika node tidak ditemukan, tampilkan pesan kesalahan
            }
            break; // Keluar dari case 6
        }

        case 7: // Menu untuk traversal PreOrder
            cout << "\n Traversal PreOrder : ";
            preOrder(root); // Panggil fungsi preOrder untuk melakukan traversal PreOrder
            break;

        case 8: // Menu untuk traversal InOrder
            cout << "\n Traversal InOrder : ";
            inOrder(root); // Panggil fungsi inOrder untuk melakukan traversal InOrder
            break;

        case 9: // Menu untuk traversal PostOrder
            cout << "\n Traversal PostOrder : ";
            postOrder(root); // Panggil fungsi postOrder untuk melakukan traversal PostOrder
            break;

        case 10: // Menu untuk menghapus subtree dari sebuah node
        {
```

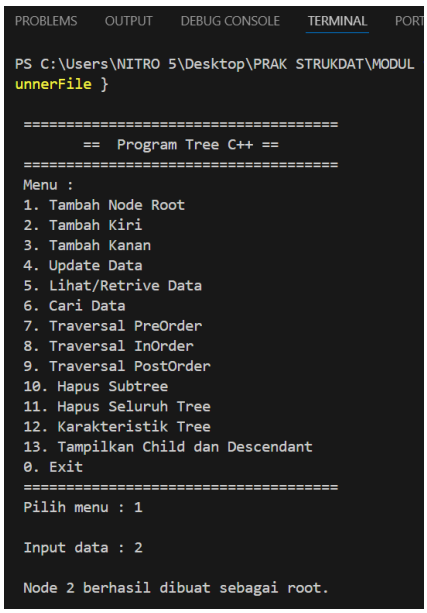
```
            char nodeData;                                                  //
Variabel untuk menyimpan data node yang ingin dihapus subtree-nya
            cout << " Input data node yang ingin dihapus subtree-nya: "; //
Meminta input data node yang ingin dihapus subtree-nya
            cin >> nodeData;                                                //
Membaca data node yang ingin dihapus subtree-nya dari pengguna
            Pohon *node = root;                                             //
Inisialisasi node yang akan digunakan untuk mencari node yang ingin dihapus
subtree-nya
            // Loop untuk mencari node yang ingin dihapus subtree-nya
berdasarkan datanya
            while (node && node->data_gita_2311102025  != nodeData)
            {
                if (node->left && node->left->data_gita_2311102025  ==
nodeData)
                {
                    node = node->left; // Jika node pada left child sesuai
dengan nodeData, pindahkan node ke left child
                }
                else if (node->right && node->right->data_gita_2311102025  ==
nodeData)
                {
                    node = node->right; // Jika node pada right child sesuai
dengan nodeData, pindahkan node ke right child
                }
                else if (node->left)
                {
                    node = node->left; // Jika node memiliki left child,
pindahkan node ke left child
                }
                else if (node->right)
                {
                    node = node->right; // Jika node memiliki right child,
pindahkan node ke right child
                }
                else
                {
                    node = NULL; // Jika tidak ada kondisi di atas yang
terpenuhi, atur node menjadi NULL (node tidak ditemukan)
                }
            }
            if (node)
            {
                // Panggil fungsi deleteSubtree untuk menghapus subtree
                deleteSubtree(node);
            }
            else
            {
```

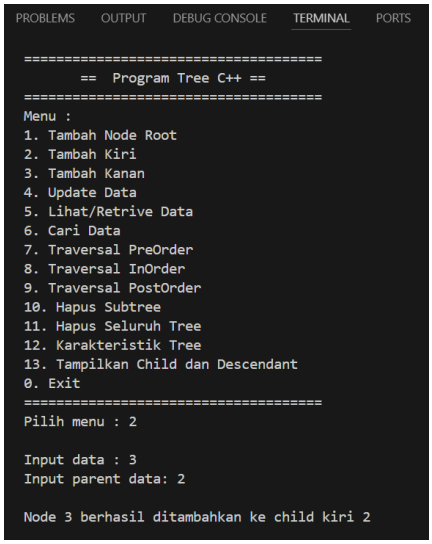
```
                cout << "\n Node tidak ditemukan!" << endl; // Jika node tidak
ditemukan, tampilkan pesan kesalahan
            }
            break; // Keluar dari case 10
        }

        case 11: // Menu untuk menghapus seluruh tree
            cout << "\n Hapus Seluruh Tree : ";
            init();                                         // Inisialisasi ulang
tree
            cout << "\n Tree berhasil dihapus!" << endl; // Tampilkan pesan
berhasil menghapus tree
            break;                                          // Keluar dari case
11

        case 12: // Menu untuk menampilkan karakteristik tree
            characteristic(); // Memanggil fungsi characteristic()

            break; // Keluar dari case 12

        case 13: // Menu untuk menampilkan child dan descendant sebuah node
        {
            char
nodeData;                                                                //
Variabel untuk menyimpan data node yang ingin dilihat child dan descendant-nya
            cout << " Input data node yang ingin dilihat child dan descendant-
nya: "; // Meminta input data node yang ingin dilihat child dan descendant-nya
            cin >>
nodeData;                                                              // Membaca
data node yang ingin dilihat child dan descendant-nya dari pengguna
            Pohon *node =
root;                                                    // Inisialisasi
node yang akan digunakan untuk mencari node yang ingin dilihat child dan
descendant-nya
            // Loop untuk mencari node yang ingin dilihat child dan
descendant-nya berdasarkan datanya
            while (node && node->data_gita_2311102025  != nodeData)
            {
                if (node->left && node->left->data_gita_2311102025  ==
nodeData)
                {
                    node = node->left; // Jika node pada left child sesuai
dengan nodeData, pindahkan node ke left child
                }
                else if (node->right && node->right->data_gita_2311102025  ==
nodeData)
                {
```

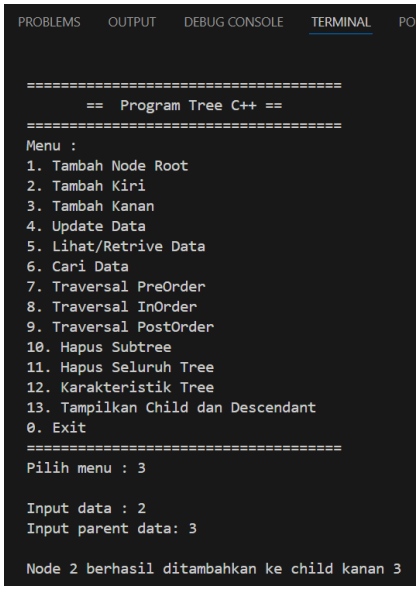
```
                    node = node->right; // Jika node pada right child sesuai
dengan nodeData, pindahkan node ke right child
                }
                else if (node->left)
                {
                    node = node->left; // Jika node memiliki left child,
pindahkan node ke left child
                }
                else if (node->right)
                {
                    node = node->right; // Jika node memiliki right child,
pindahkan node ke right child
                }
                else
                {
                    node = NULL; // Jika tidak ada kondisi di atas yang
terpenuhi, atur node menjadi NULL (node tidak ditemukan)
                }
            }
            if (node)
            {
                // Panggil fungsi showChildAndDescendant untuk menampilkan
child dan descendant
                showChildAndDescendant(node);
            }
            else
            {
                cout << "\n Node tidak ditemukan!" << endl; // Jika node tidak
ditemukan, tampilkan pesan kesalahan
            }
            break; // Keluar dari case 13
        }

        case
0:                                                          // Menu untuk
keluar dari program
            cout << "\n Terima kasih telah menggunakan program ini!" << endl;
// Tampilkan pesan terima kasih
            break;
// Keluar dari case 0

        default:                                            // Default case
untuk pilihan menu tidak valid
            cout << "\n Pilihan menu tidak valid!" << endl; // Tampilkan pesan
kesalahan untuk pilihan menu tidak valid
        }
```

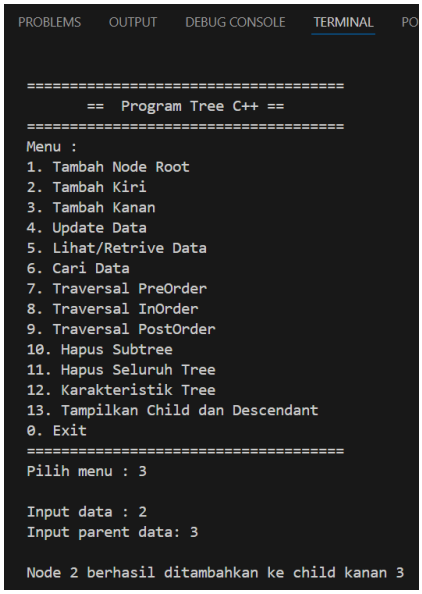
```
    } while (pil != 0); // Loop akan terus berjalan selama pilihan menu yang dimasukkan tidak sama dengan 0, menandakan pengguna masih ingin menggunakan program.

    return 0; // Mengembalikan nilai 0 sebagai tanda bahwa program telah berakhir dengan sukses.
}
```

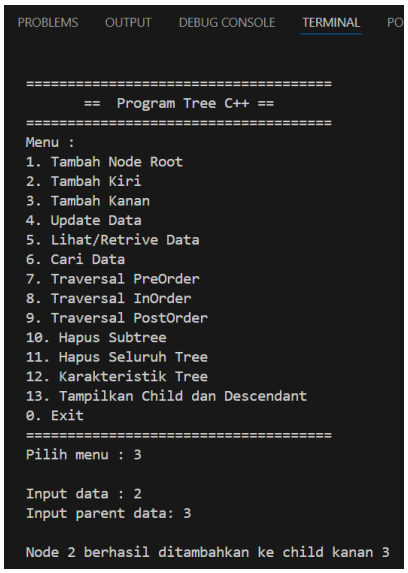
Screenshoot Program

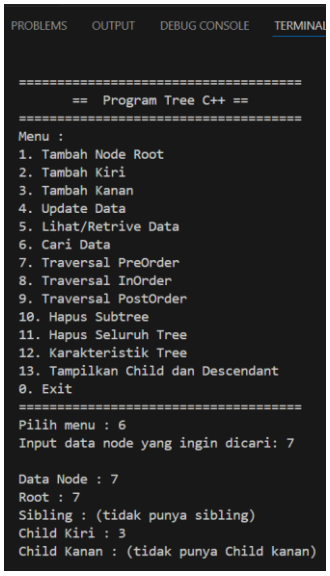


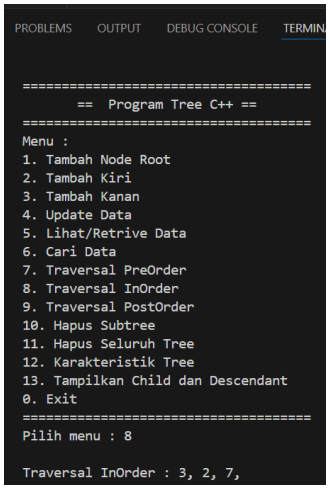



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PO
==========
== Program Tree C++ ==
==========
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
==========
Pilih menu : 3
Input data : 2
Input parent data: 3
Node 2 berhasil ditambahkan ke child kanan 3

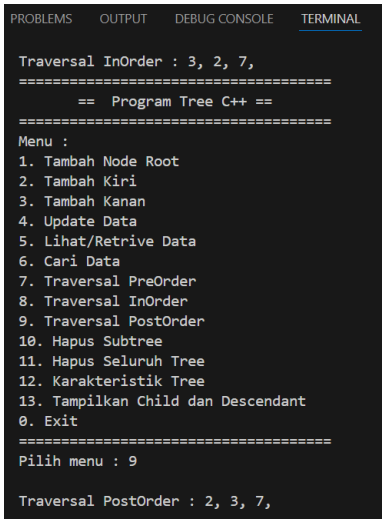

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PO
==========
== Program Tree C++ ==
==========
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
==========
Pilih menu : 3
Input data : 2
Input parent data: 3
Node 2 berhasil ditambahkan ke child kanan 3

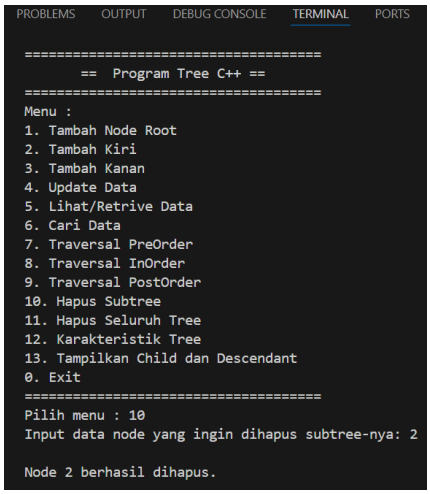

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PO
==========
== Program Tree C++ ==
==========
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
==========
Pilih menu : 3
Input data : 2
Input parent data: 3
Node 2 berhasil ditambahkan ke child kanan 3

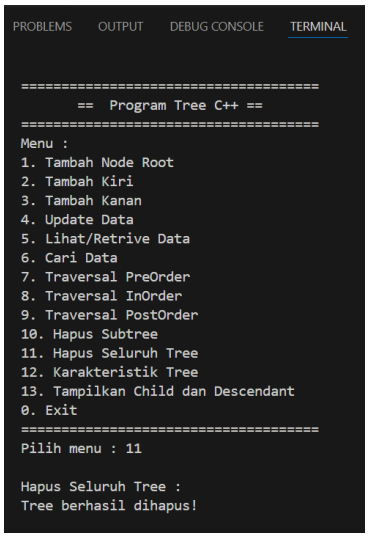

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 6
Input data node yang ingin dicari: 7
Data Node : 7
Root : 7
Sibling : (tidak punya sibling)
Child Kiri : 3
Child Kanan : (tidak punya Child kanan)


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 8
Traversal InOrder : 3, 2, 7,


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 8
Traversal InOrder : 3, 2, 7,


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Traversal InOrder : 3, 2, 7,
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 9
Traversal PostOrder : 2, 3, 7,


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 10
Input data node yang ingin dihapus subtree-nya: 2
Node 2 berhasil dihapus.


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
== Program Tree C++ ==
Menu :
1. Tambah Node Root
2. Tambah Kiri
3. Tambah Kanan
4. Update Data
5. Lihat/Retrive Data
6. Cari Data
7. Traversal PreOrder
8. Traversal InOrder
9. Traversal PostOrder
10. Hapus Subtree
11. Hapus Seluruh Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendant
0. Exit
Pilih menu : 11
Hapus Seluruh Tree :
Tree berhasil dihapus!

Deskripsi Program

Program C++ ini adalah implementasi struktur data Pohon Biner (Binary Tree) yang menyediakan berbagai operasi untuk mengelola pohon biner seperti menambahkan node, menghapus node, memperbaharui data node, melakukan traversal (PreOrder, InOrder, dan PostOrder), serta menghitung ukuran dan tinggi pohon.

Program mendefinisikan struktur Pohon yang terdiri dari data bertipe char dan tiga pointer, yaitu left, right, dan parent, yang digunakan untuk menghubungkan setiap node dalam pohon.

Program menyediakan menu utama yang memungkinkan pengguna untuk memilih operasi yang ingin dilakukan pada pohon biner, seperti menambahkan node root, menambahkan node kiri atau kanan, mengupdate data node, melihat data node, mencari data node, melakukan traversal, menghapus subtree, menghapus seluruh pohon, melihat karakteristik pohon, dan menampilkan child serta descendant dari suatu node.

Program ini dapat digunakan untuk memahami konsep pohon biner dan mengimplementasikan berbagai operasi pada struktur data tersebut dalam bahasa C++. Program ini juga menyediakan antarmuka menu yang memudahkan pengguna untuk berinteraksi dengan program dan melakukan operasi yang diinginkan.

BAB V

KESIMPULAN

Setelah mempelajari materi Graf dan Pohon (Graph and Tree), ini Kesimpulan yang saya dapatkan:

 Graf dan Pohon adalah struktur data yang sangat penting dan banyak digunakan dalam pemrograman. Struktur data ini memungkinkan representasi dan pengelolaan data yang saling terhubung atau memiliki hierarki.
 Graf terdiri dari kumpulan simpul (node) yang dihubungkan oleh sisi (edge). Graf dapat direpresentasikan menggunakan matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list). Operasi yang umum dilakukan pada graf antara lain menambahkan atau menghapus simpul dan sisi, serta melakukan traversal (penelusuran) pada graf.
 Pohon adalah struktur data hierarkis khusus yang merupakan kasus khusus dari graf. Setiap simpul dalam pohon, kecuali root, memiliki tepat satu simpul induk (parent). Pohon memiliki jenis-jenis seperti pohon biner (binary tree), pohon biner pencarian (binary search tree), dan pohon AVL (AVL tree).
 Operasi yang umum dilakukan pada pohon antara lain menambahkan node baru, menghapus node, melakukan traversal (penelusuran) dengan metode seperti pre-order, in-order, dan post-order, serta mencari node tertentu dalam pohon.
  Implementasi Graf dan Pohon dalam C++ memerlukan penggunaan pointer dan struktur data dinamis seperti linked list atau array dinamis. Penggunaan pointer memungkinkan pembuatan dan penghapusan simpul atau sisi secara dinamis sesuai kebutuhan.
 Algoritma-algoritma penting seperti pencarian jalur terpendek (shortest path), penjelajahan graf (graph traversal), dan penyusunan secara hierarkis (hierarchical organization) dapat diimplementasikan dengan menggunakan struktur data Graf dan Pohon.

 Penguasaan konsep dan implementasi Graf dan Pohon dalam C++ sangat penting dalam pengembangan aplikasi yang melibatkan pengelolaan data yang saling terhubung atau memiliki hierarki, seperti sistem navigasi, jaringan sosial, struktur organisasi, atau aplikasi pengolahan file dan direktori.

Secara keseluruhan, mempelajari materi Graf dan Pohon dalam C++ memberikan pemahaman yang mendalam tentang bagaimana merepresentasikan dan mengelola data yang saling terhubung atau memiliki hierarki secara efisien. Penerapan konsep-konsep ini memungkinkan pengembangan solusi yang lebih efektif dan optimal untuk berbagai permasalahan dalam bidang pemrograman.