

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

**MODUL IX
GRAPH AND TREE**



Disusun Oleh :
DIO GILBRAN PRAMANA
NIM : 2311102062

Dosen
WAHYU ANDI SAPUTRA, S.Pd., M.Eng.

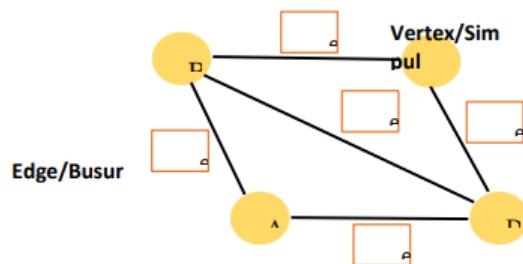
**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. DASAR TEORI

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 sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

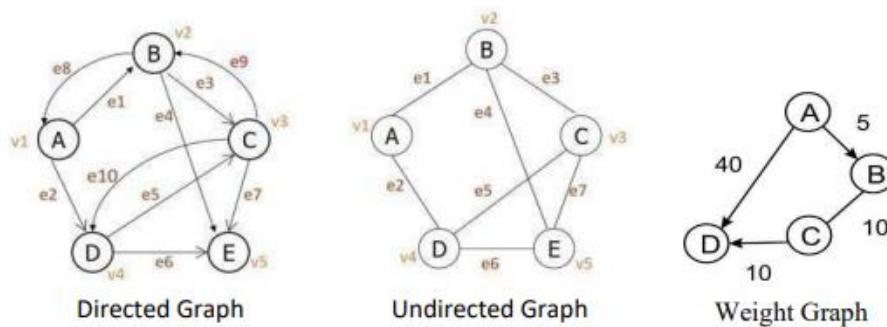
Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge. Dapat digambarkan:



Gambar 1 Contoh Graph

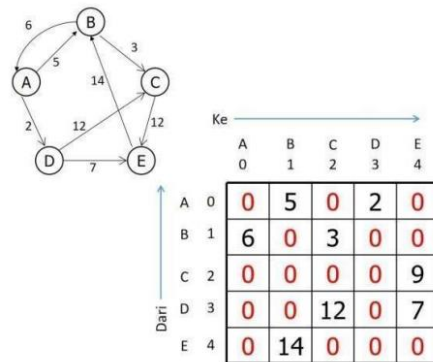
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph



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

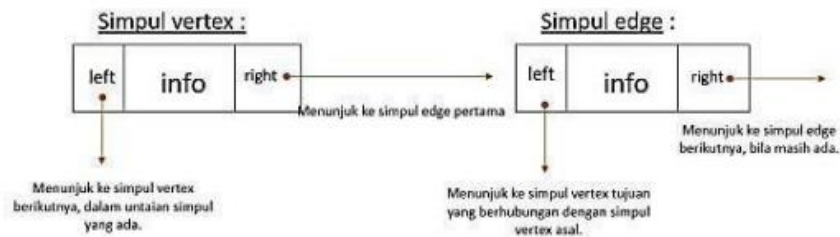
Representasi Graph dengan Matriks



Gambar 4 Representasi Graph dengan Matriks

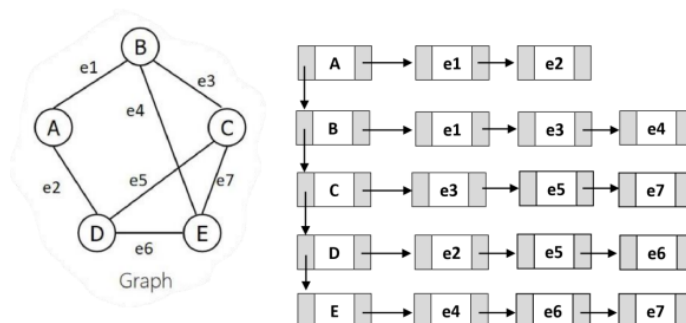
Representasi dengan Linked List

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

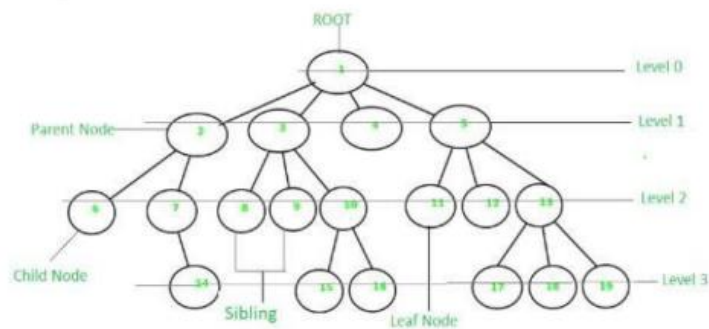
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

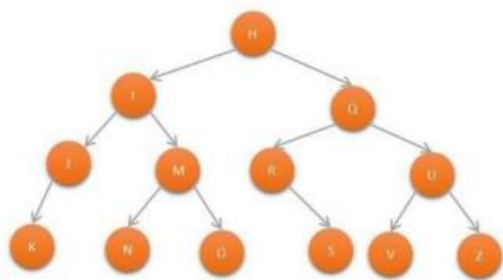
Dalam ilmu komputer, pohon/tree adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, dimana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hirarki 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

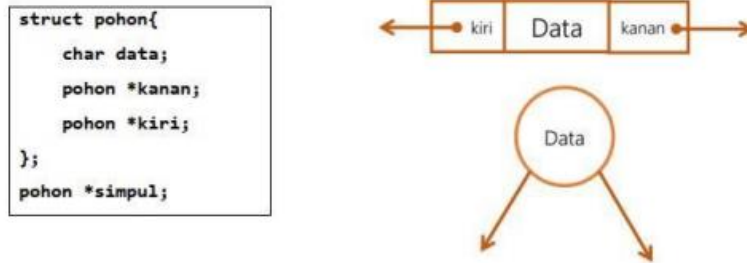
Tabel 1 Terminologi dalam Struktur Data Tree

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.



Gambar 2 Ilustrasi Simpul 2 Pointer

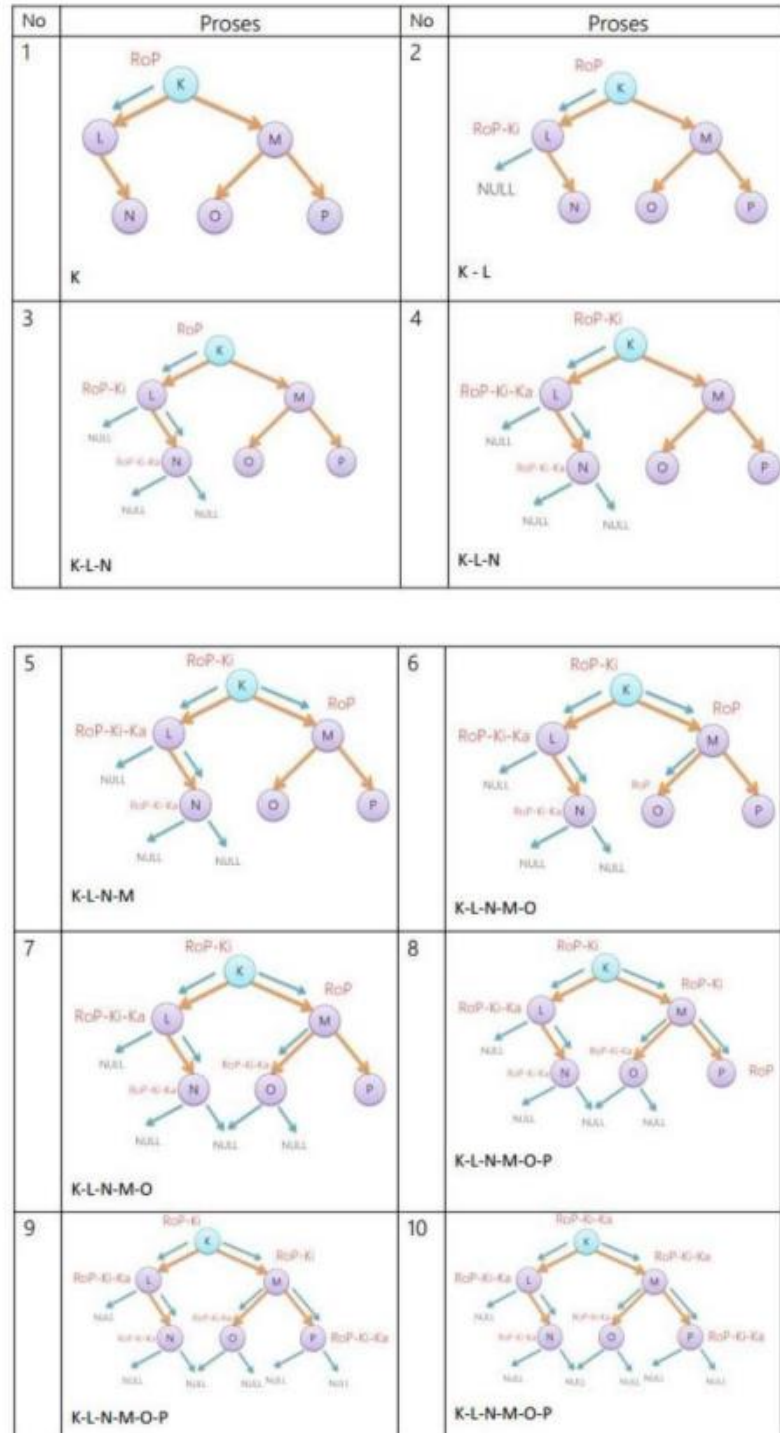
Operasi pada Tree

- 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 dari suatu 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:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Alur pre-order



2. In-Order Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
 - Cetak data pada root
 - Secara rekursif mencetak seluruh data pada subpohon kanan
- Dapat kita turunkan rumus penelusuran menjadi



3. Post Order Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan
- Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



B. GUIDED

GUIDED 1

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

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

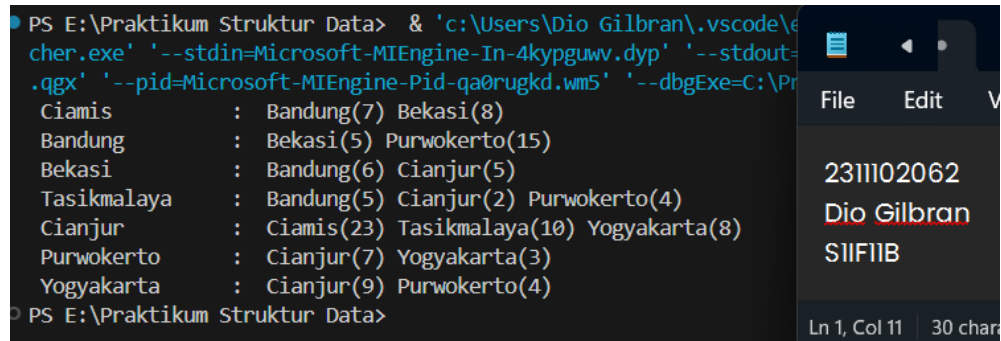
```

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

int main()
{
    tampilgraph();
    return 0;
}

```

SCREENSHOT OUTPUT



```

PS E:\Praktikum Struktur Data> & 'c:\Users\Dio Gilbran\.vscode\ch
cher.exe' '--stdin=Microsoft-MIEngine-In-4kypguwv.dyp' '--stdout=
.qgx' '--pid=Microsoft-MIEngine-Pid-qa0rugkd.wm5' '--dbgExe=C:\Pr
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS E:\Praktikum Struktur Data>

```

DESKRIPSI

Program ini adalah aplikasi untuk menampilkan representasi graf berbentuk matriks ketetanggaan (adjacency matrix) dalam bahasa C++. Program ini memiliki array simpul yang berisi nama-nama kota dan matriks busur yang berisi jarak antar kota. Fungsi tampilgraph menampilkan setiap kota beserta kota-kota tetangganya dan jarak ke kota-kota tersebut, dengan memanfaatkan format yang teratur. Program ini membantu memvisualisasikan koneksi antar kota berdasarkan jarak yang sudah ditentukan dalam matriks busur.

GUIDED 2

```
#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"

```

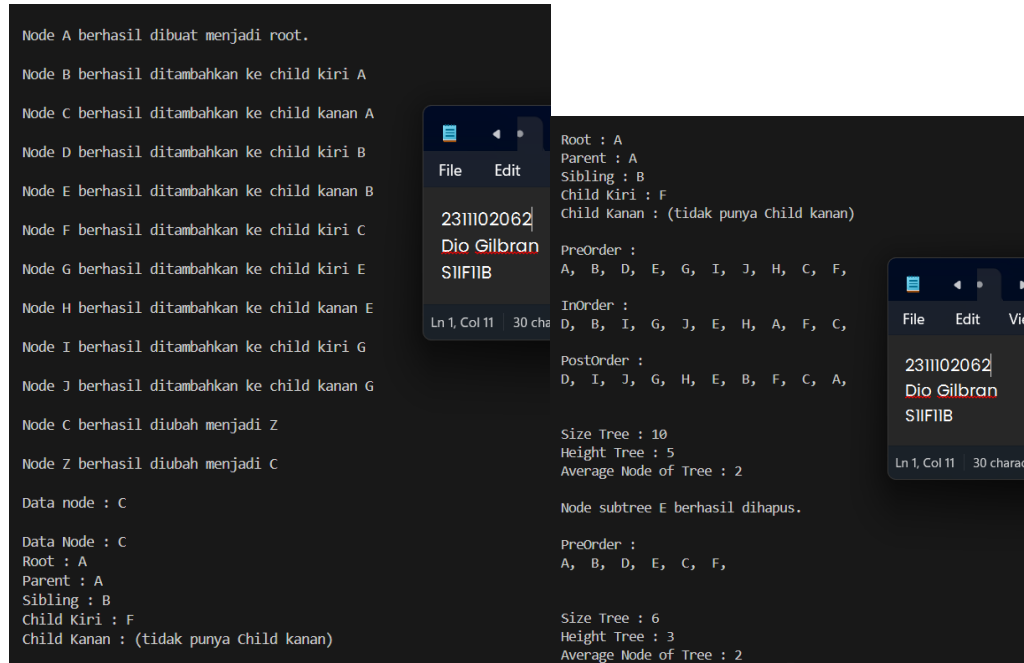


```

        << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
        << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
        << endl;
    charateristic();
}

```

SCREENSHOT OUTPUT



DESKRIPSI

Program ini merupakan implementasi dari struktur data pohon biner menggunakan bahasa C++. Program ini mencakup berbagai operasi pada pohon biner seperti inisialisasi, penambahan node kiri dan kanan, pembaruan data node, penelusuran (pre-order, in-order, post-order), pencarian node, dan penghapusan node atau sub-pohon. Selain itu, program ini juga menyediakan fitur untuk menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Dengan menjalankan fungsi-fungsi ini, pengguna dapat membuat, memodifikasi, menelusuri, dan mengelola pohon biner secara interaktif.

C. UNGUIDED

UNGUIDED 1

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

int main()
{
    int simpul_2311102062;

    cout << "Silakan masukkan jumlah simpul : ";
    cin >> simpul_2311102062;

    string simpul[simpul_2311102062];
    int bobot[simpul_2311102062][simpul_2311102062];

    cout << "Silakan masukkan nama simpul\n";
    for (int i = 0; i < simpul_2311102062; i++)
    {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    cout << "Silakan masukkan bobot antar simpul\n";
    for (int i = 0; i < simpul_2311102062; i++)
    {
        for (int j = 0; j < simpul_2311102062; j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << " : ";
            cin >> bobot[i][j];
        }
    }

    cout << endl << setw(10) << " ";
    for (int i = 0; i < simpul_2311102062; i++)
    {
        cout << setw(10) << simpul[i];
    }
    cout << endl;
    for (int i = 0; i < simpul_2311102062; i++)
    {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < simpul_2311102062; j++)
        {
            cout << setw(10) << bobot[i][j];
        }
        cout << endl;
    }
}
```

```

    }
    return 0;
}

```

SCREENSHOT OUTPUT

```

t-13hviclt.wap' '--stderr=Microsoft-MIEngine-Error-rjbm95
soft-MIEngine-Pid-is4w5m4d.g5h' '--dbgExe=C:\Program Files
in\gdb.exe' '--interpreter=mi'
Silakan masukkan jumlah simpul : 2
Silakan masukkan 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
PS E:\Praktikum Struktur Data>

```

DESKRIPSI

Program ini meminta pengguna untuk memasukkan jumlah simpul (nodes) dalam sebuah graph, kemudian nama-nama simpul tersebut, dan bobot (weight) atau jarak antar simpul. Setelah itu, program mencetak tabel adjacency matrix yang menunjukkan bobot antara setiap pasangan simpul. Dalam tabel ini, setiap baris dan kolom mewakili simpul, dan nilai di persimpangan baris dan kolom menunjukkan bobot antar simpul tersebut.

UNGUIDED 2

```

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

// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;

void init()
{
    root = NULL;
}

// Cek Node
int isEmpty()
{
    return (root == NULL);
}

```

```

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

// Cari Node Berdasarkan Data
Pohon *findNode(Pohon *node, char data)
{
    if (node == NULL)
        return NULL;
    if (node->data == data)
        return node;
    Pohon *foundNode = findNode(node->left, data);
    if (foundNode == NULL)
        foundNode = findNode(node->right,
                             data);
    return foundNode;
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        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 (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // mengecek apakah child kanan ada atau tidak dahulu
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan oyy!" << 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())
    {
        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)
{
    if (node != NULL)
    {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// inOrder
void inOrder(Pohon *node)
{
    if (node != NULL)
    {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

// postOrder
void postOrder(Pohon *node)
{
    if (node != NULL)
    {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (node != NULL)
    {
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root)
        {
            delete root;
            root = NULL;
        }
        else

```



```

        {
            delete node;
        }
    }
}

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

// Hapus Tree
void clear()
{
    deleteTree(root);
    cout << "\n Pohon berhasil dihapus." << endl;
}

// Cek Size Tree
int size(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        return 1 + size(node->left) + size(node->right);
    }
}

// Cek Height Level Tree
int height(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
}

```

```

        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            return max(heightKiri, heightKanan) + 1;
        }
    }

// Karakteristik Tree
void characteristic()
{
    cout << "\n Size Tree : " << size(root) << endl;
    cout << " Height Tree : " << height(root) << endl;
    cout << " Average Node of Tree : " << (height(root) == 0 ? 0 :
size(root) / height(root)) << endl;
}

// Menampilkan Child Node
void showChildren(Pohon *node)
{
    if (node)
    {
        if (node->left)
            cout << " Child Kiri: " << node->left->data << endl;
        else
            cout << " Child Kiri: (tidak punya Child kiri)" <<
endl;
        if (node->right)
            cout << " Child Kanan: " << node->right->data << endl;
        else
            cout << " Child Kanan: (tidak punya Child kanan)"
<< endl;
    }
}

// Menampilkan Descendants Node
void showDescendants(Pohon *node)
{
    if (node)
    {
        cout << " Descendants of Node " << node->data << ": ";
        preOrder(node);
        cout << endl;
    }
}

void menu()
{

```

```

int pilihan;
char data;
char parentdata2311102062;
Pohon *temp = nullptr;
do
{
    cout << "\nMENU:\n";
    cout << "1. Buat Node Root\n";
    cout << "2. Tambah Node Kiri\n";
    cout << "3. Tambah Node Kanan\n";
    cout << "4. Update Node\n";
    cout << "5. Retrieve Node\n";
    cout << "6. Find Node\n";
    cout << "7. Tampilkan PreOrder\n";
    cout << "8. Tampilkan InOrder\n";
    cout << "9. Tampilkan PostOrder\n";
    cout << "10. Tampilkan Characteristic\n";
    cout << "11. Hapus SubTree\n";
    cout << "12. Hapus Tree\n";
    cout << "13. Tampilkan Children\n";
    cout << "14. Tampilkan Descendants\n";
    cout << "0. Keluar\n";
    cout << "Masukkan pilihan: ";
    cin >> pilihan;
    switch (pilihan)
    {
    case 1:
        if (isEmpty())
        {
            cout << "Masukkan data root: ";
            cin >> data;
            buatNode(data);
        }
        else
        {
            cout << "\n Root sudah ada!" << endl;
        }
        break;
    case 2:
        if (!isEmpty())
        {
            cout << "Masukkan data node kiri: ";
            cin >> data;
            cout << "Masukkan data parent: ";
            cin >> parentdata2311102062;
            temp = findNode(root, parentdata2311102062);
            insertLeft(data, temp);
        }
    }
}

```

```

    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 3:
    if (!isEmpty())
    {
        cout << "Masukkan data node kanan: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        insertRight(data, temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 4:
    if (!isEmpty())
    {
        cout << "Masukkan data baru: ";
        cin >> data;
        cout << "Masukkan data node yang akan
diupdate: ";
        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        update(data, temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 5:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan dilihat: ";
        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        retrieve(temp);
    }
    else
    {

```

```

        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 6:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan dicari: ";
        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        find(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 7:
    if (!isEmpty())
    {
        cout << "\n PreOrder :" << endl;
        preOrder(root);
        cout << "\n"
             << endl;
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 8:
    if (!isEmpty())
    {
        cout << "\n InOrder :" << endl;
        inOrder(root);
        cout << "\n"
             << endl;
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 9:
    if (!isEmpty())
    {
        cout << "\n PostOrder :" << endl;
        postOrder(root);
    }

```

```

        cout << "\n"
            << endl;
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 10:
    if (!isEmpty())
    {
        characteristic();
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 11:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang subtreenya akan
dihapus : ";

        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        deleteSub(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 12:
    clear();
    break;
case 13:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan ditampilkan
childnya : ";

        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        showChildren(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
}

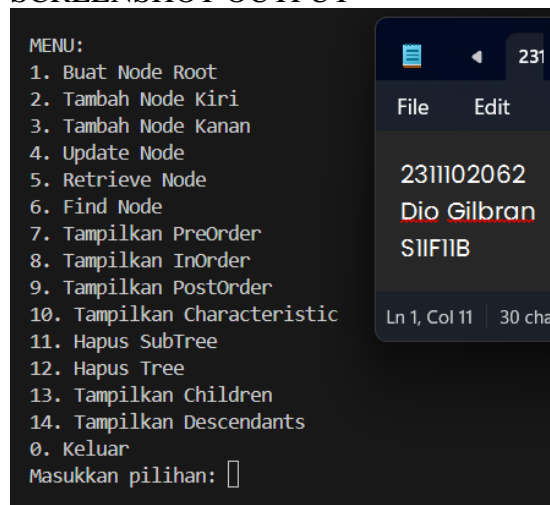
```

```

    }
    break;
case 14:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan ditampilkan
descendantnya : ";
        cin >> parentdata2311102062;
        temp = findNode(root, parentdata2311102062);
        showDescendants(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 0:
    cout << "\n Keluar dari program..." << endl;
    break;
default:
    cout << "\n Pilihan tidak valid!" << endl;
}
} while (pilihan != 0);
}
int main()
{
    init();
    menu();
    return 0;
}

```

SCREENSHOT OUTPUT



<p>0. Keluar</p> <p>Masukkan pilihan: 1</p> <p>Masukkan data root: A</p> <p>Node A berhasil dibuat menjadi root.</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 2</p> <p>Masukkan data node kiri: B</p> <p>Masukkan data parent: A</p> <p>Node B berhasil ditambahkan ke child kiri A</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 3</p> <p>Masukkan data node kanan: C</p> <p>Masukkan data parent: A</p> <p>Node C berhasil ditambahkan ke child kanan A</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 2</p> <p>Masukkan data node kiri: D</p> <p>Masukkan data parent: B</p> <p>Node D berhasil ditambahkan ke child kiri B</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 3</p> <p>Masukkan data node kanan: E</p> <p>Masukkan data parent: B</p> <p>Node E berhasil ditambahkan ke child kanan B</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 2</p> <p>Masukkan data node kiri: F</p> <p>Masukkan data parent: C</p> <p>Node F berhasil ditambahkan ke child kiri C</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 2</p> <p>Masukkan data node kiri: G</p> <p>Masukkan data parent: E</p> <p>Node G berhasil ditambahkan ke child kiri E</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>
<p>Masukkan pilihan: 3</p> <p>Masukkan data node kanan: H</p> <p>Masukkan data parent: E</p> <p>Node H berhasil ditambahkan ke child kanan E</p>	<p>2311102062</p> <p><u>Dio Gilbran</u></p> <p>SIIFIIB</p>

<p>Masukkan pilihan: 2 Masukkan data node kiri: 1 Masukkan data parent: G</p> <p>Node 1 berhasil ditambahkan ke child kiri G</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 3 Masukkan data node kanan: J Masukkan data parent: G</p> <p>Node J berhasil ditambahkan ke child kanan G</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 4 Masukkan data baru: Z Masukkan data node yang akan diupdate: 1</p> <p>Node 1 berhasil diubah menjadi Z</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 4 Masukkan data baru: I Masukkan data node yang akan diupdate: Z</p> <p>Node Z berhasil diubah menjadi I</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 5 Masukkan data node yang akan dilihat: C</p> <p>Data node : C</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 6 Masukkan data node yang akan dicari: C</p> <p>Data Node : C Root : A Parent : A Sibling : B Child Kiri : F Child Kanan : (tidak punya Child kanan)</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>
<p>Masukkan pilihan: 7</p> <p>PreOrder : A, B, D, E, G, I, J, H, C, F,</p>	<p>2311102062 Dio Gilbran SIIFIIB</p>

Masukkan pilihan: 8	
InOrder : D, B, I, G, J, E, H, A, F, C,	2311102062 <u>Dio Gilbran</u> SIIF11B
Masukkan pilihan: 9	
PostOrder : D, I, J, G, H, E, B, F, C, A,	2311102062 <u>Dio Gilbran</u> SIIF11B
Masukkan pilihan: 10	
Size Tree : 10 Height Tree : 5 Average Node of Tree : 2	2311102062 <u>Dio Gilbran</u> SIIF11B
0. Keluar	
Masukkan pilihan: 13	2311102062
Masukkan data node yang akan ditampilkan childnya : A	<u>Dio Gilbran</u>
Child Kiri: B	SIIF11B
Child Kanan: C	
0. Keluar	
Masukkan pilihan: 11	2311102062
Masukkan data node yang subtreenya akan dihapus : E	<u>Dio Gilbran</u>
Node subtree E berhasil dihapus.	SIIF11B
0. Keluar	
Masukkan pilihan: 14	2311102062
Masukkan data node yang akan ditampilkan descendantnya : A	<u>Dio Gilbran</u>
Descendants of Node A: A, B, D, E, C, F,	SIIF11B
0. Keluar	
Masukkan pilihan: 12	2311102062
Pohon berhasil dihapus.	<u>Dio Gilbran</u> SIIF11B

DESKRIPSI

Program ini merupakan implementasi pohon biner dengan berbagai fitur untuk manipulasi data. Pengguna dapat membuat pohon, menambah node kiri atau kanan, mengupdate node, serta melihat data node tertentu. Program ini juga menyediakan fungsi untuk mencari node berdasarkan data, menampilkan traversal preorder, inorder, dan postorder, serta karakteristik pohon seperti

ukuran dan tinggi. Selain itu, pengguna dapat menghapus subtree atau seluruh pohon, menampilkan anak-anak dan keturunan dari sebuah node. Menu interaktif memandu pengguna dalam melakukan operasi-operasi tersebut.

D. KESIMPULAN

Pada bagian graph, kami mempelajari berbagai jenis graf seperti graf berarah (directed graph), graf tak berarah (undirected graph), dan graf berbobot (weighted graph). Implementasi graf menggunakan matriks ketetanggaan (adjacency matrix) membantu memvisualisasikan koneksi antar node berdasarkan bobot atau jarak yang sudah ditentukan.

Sedangkan pada bagian tree, fokus utama kami adalah pada pohon biner (binary tree) dan operasinya. Kami mempelajari cara membuat pohon, menambah node pada posisi kiri atau kanan, melakukan penelusuran (traversal) seperti pre-order, in-order, dan post-order, serta menghapus node atau sub-pohon. Program interaktif yang dikembangkan membantu dalam memahami karakteristik pohon seperti ukuran, tinggi, dan rata-rata node.

Untuk semua bagiannya, praktikum ini memberikan kami pengalaman praktis dalam mengimplementasikan struktur data graf dan pohon, serta memahami bagaimana kedua struktur data ini dapat digunakan untuk memecahkan berbagai masalah dalam ilmu komputer.

E. REFERENSI

Asisten Praktikum, "Modul Algoritma GRAPH DAN TREE", 2024.

Karumanchi, N. (2016). Data Structures and algorithms made easy: Concepts, problems, Interview Questions. CareerMonk Publications.

Budi Raharjo. 2015. Pemrograman C++. Bandung: Informatika.