

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA**

**MODUL 9  
GRAPH AND TREE**



**Dosen : Wahyu Andi Saputra, S.Pd., M.Eng.**

**Disusun oleh:**

**MUHAMMAD AULIA MUZZAKI NUGRAHA**

**(2311102051)**

**IF-11-B**

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

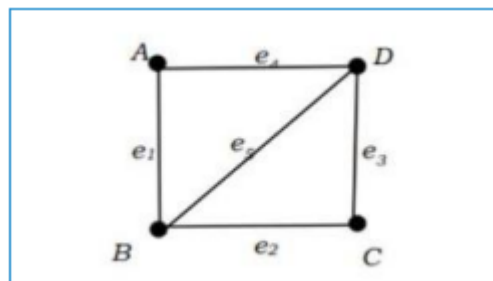
**2024**

# BAB I

## DASAR TEORI

### 1. Teori Graph

Graf adalah himpunan  $G = (V, E)$  di mana  $V$  adalah himpunan simpul dan  $E$  adalah himpunan sisi yang menghubungkan simpul-simpul tersebut. Simpul diberi label seperti A, B, C atau 1, 2, 3, dan sisi yang menghubungkan simpul  $u$  dan  $v$  dinyatakan dengan  $(u, v)$  atau  $e_1, e_2, \dots, e_n$ . Secara geometris, graf digambarkan sebagai simpul-simpul di bidang dua dimensi yang dihubungkan oleh garis-garis. (Munir, 2012; Sembiring, 2022).

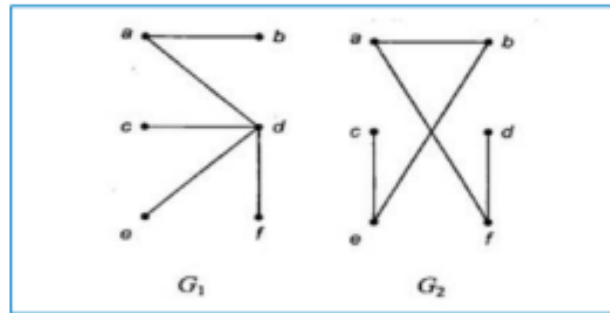


**Gambar 1.** Suatu Graf  $G(4,5)$  (Munir, 2010)

Graf  $G$  pada gambar di atas dapat dinyatakan sebagai  $G(4,5)$  dengan  $V(G) = \{A, B, C, D\}$  dan  $E(G) = \{(A, B), (A, D), (B, C), (C, D), (B, D)\}$ . Dengan kata lain, himpunan sisi  $E(G) = \{e_1, e_2, e_3, e_4, e_5\}$  untuk  $e_1 = (A, B)$ ,  $e_2 = (B, C)$ ,  $e_3 = (C, D)$ ,  $e_4 = (A, D)$ , dan  $e_5 = (B, D)$

### 2. Teori Tree

Pohon adalah graf terhubung yang tidak memiliki sirkuit dan tidak memuat sisi paralel atau loop, menjadikannya graf sederhana. Dua sifat penting pohon adalah terhubung dan tidak mengandung sirkuit. Gambar  $G_1$  dan  $G_2$  pada Gambar 2 adalah pohon karena keduanya terhubung dan tidak memiliki loop. Meskipun terlihat berbeda,  $G_1$  dan  $G_2$  sebenarnya sama. Bentuk pohon tidak harus menyerupai tanaman dengan akar dan cabang (Sembiring, 2022).



**Gambar 2.**  $G_1$  dan  $G_2$  adalah Pohon

## BAB II

### GUIDED

#### LATIHAN – GUIDED

##### 1. Guided 1

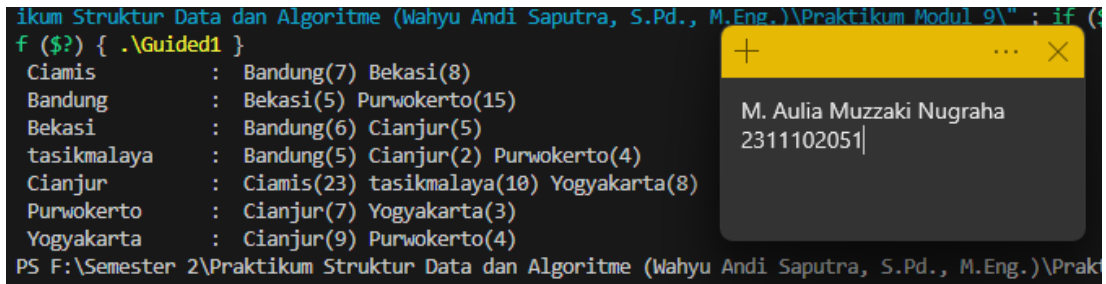
Program Graph

**Source Code**

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {
    "Ciamis",
    "Bandung",
    "Bekasi",
    "tasikmalaya",
    "Cianjur",
    "Purwokerto",
    "Yogyakarta"};
int busur[7][7] = {
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
            << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom]
                << ")";
            }
        }
        cout << endl;
    }
}
```

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

## Screenshoot program



```
ikun Struktur Data dan Algoritme (Wahyu Andi Saputra, S.Pd., M.Eng.)\Praktikum Modul 9\" ; if (
f ($?) { .\Guided1 }
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 F:\Semester 2\Praktikum Struktur Data dan Algoritme (Wahyu Andi Saputra, S.Pd., M.Eng.)\Prak
```

## Deskripsi program

Pada awalnya, program mendefinisikan dua array, yaitu simpul yang berisi daftar nama-nama kota atau tempat, dan busur yang berisi jarak atau bobot antara setiap pasangan kota. Array simpul memiliki tujuh elemen, yang berarti graf tersebut terdiri dari tujuh simpul atau node. Array busur memiliki tujuh baris dan tujuh kolom, yang mewakili jarak antara setiap pasangan kota. Jika nilai pada busur[i][j] adalah nol, maka tidak ada jalur langsung antara kota i dan kota j. Selanjutnya, program mendefinisikan sebuah fungsi bernama tampilGraph() yang bertanggung jawab untuk menampilkan representasi graf tersebut ke layar. Fungsi ini menggunakan dua perulangan for bersarang untuk menelusuri setiap baris dan kolom pada array busur. Untuk setiap baris, fungsi tersebut mencetak nama kota yang sesuai dari array simpul, diikuti dengan daftar kota-kota yang terhubung langsung beserta jaraknya dalam bentuk kota(jarak).

## 2. Guided 2

Program tree

### Source Code

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

#include <iostream>
using namespace std;

// PROGRAM BINARY TREE
```

```

// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent; // pointer
};
// pointer global
Pohon *root;
// Inisialisasi
void init()
{
    root = NULL;
}
bool isEmpty()
{
    return root == NULL;
}
Pohon *newPohon(char data)
{
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}
void buatNode(char data)
{
    if (isEmpty())
    {
        root = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root."
<< endl;
    }
    else
    {
        cout << "\nPohon sudah dibuat" << endl;
    }
}
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {

```

```

        if (node->left != NULL)
        {
            cout << "\nNode " << node->data << " sudah ada child
kiri!"

            << endl;

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

Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\nNode " << node->data << " sudah ada child
kanan!"

            << endl;

            return NULL;
        }
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kanan " << node->data << endl;
            return baru;
        }
    }
}

```

```

}
void update(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah menjadi "
<<
            data << endl;
        }
    }
}
void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData node : " << node->data << endl;
        }
    }
}
void find(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
    }
}

```



```

        else
        {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->data << endl;
            if (!node->parent)
                cout << "Parent : (tidak punya parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)

                cout << "Sibling : " << node->parent->left->data <<
endl;
            else if (node->parent != NULL && node->parent->right !=
node
                && node->parent->left == node)

                cout << "Sibling : " << node->parent->right->data <<
                endl;

            else
                cout << "Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << "Child Kiri : (tidak punya Child kiri)" <<
endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child kanan)" <<
                endl;

            else
                cout << "Child Kanan : " << node->right->data <<
endl;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else

```

```

    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

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

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

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

```

```

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

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

// Hapus Tree
void clear()
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node)

```

```

{
    if (isEmpty())
    {
        cout << "\nBuat 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)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree

```

```

void characteristic()
{
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

int main()
{
    init();
    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 << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n"
        << endl;
    cout << "InOrder :" << endl;
    inOrder(root);
    cout << "\n"
        << endl;
    cout << "PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
        << endl;
    characteristic();
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n"

```

```
<< endl;  
characteristic();  
}
```

### Screenshoot program

```
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
```



M. Aulia Muzzaki Nugraha  
2311102051

```

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

```

## Deskripsi program

Program juga menyediakan fungsi `insertLeft(char data, Pohon *node)` dan `insertRight(char data, Pohon *node)` untuk menambahkan node baru sebagai anak kiri atau anak kanan dari node yang ditentukan. Fungsi `update(char data, Pohon *node)` digunakan untuk mengubah nilai data pada node tertentu. Fungsi `retrieve(Pohon *node)` digunakan untuk mengambil nilai data dari node tertentu, sedangkan fungsi `find(Pohon *node)` memberikan informasi lebih lengkap tentang node tersebut, seperti induk, saudara, dan anak-anaknya. Selain itu, program juga menyediakan fungsi untuk melakukan traversal (penelusuran) pada pohon biner, yaitu `preOrder(Pohon *node)`, `inOrder(Pohon *node)`, dan `postOrder(Pohon *node)`. Traversal ini berguna untuk mengakses setiap node dalam pohon biner dengan urutan yang berbeda-beda.

## BAB III

### UNGUIDED

#### TUGAS – UNGUIDED

##### 1. Unguided 1

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

##### Source Code

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

using namespace std;

void Zaki_2311102051() {
    int jumlahSimpul;

    // Meminta pengguna memasukkan jumlah simpul
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    string *simpul = new string[jumlahSimpul];
    int **bobot = new int*[jumlahSimpul];
    for (int i = 0; i < jumlahSimpul; ++i) {
        bobot[i] = new int[jumlahSimpul];
    }

    // Meminta pengguna memasukkan nama-nama simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    // Meminta pengguna memasukkan bobot antar simpul
    cout << "\nSilakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    // Menampilkan hasil input pengguna
    cout << "\n";
    cout << setw(15) << " ";
    for (int i = 0; i < jumlahSimpul; i++) {
```



```

        cout << setw(15) << simpul[i];
    }
    cout << "\n";

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(15) << bobot[i][j];
        }
        cout << endl;
    }

    // Menghapus memori yang dialokasikan secara dinamis
    delete[] simpul;
    for (int i = 0; i < jumlahSimpul; ++i) {
        delete[] bobot[i];
    }
    delete[] bobot;
}

int main() {
    Zaki_2311102051();
    return 0;
}

```

### Screenshot Program



### Deskripsi program

Pada awalnya, program meminta pengguna untuk memasukkan jumlah simpul yang diinginkan. Setelah itu, program akan meminta pengguna untuk memasukkan nama-nama simpul secara berurutan. Kemudian, program akan meminta pengguna untuk memasukkan bobot antar setiap pasangan simpul. Setelah semua data dimasukkan, program akan menampilkan tabel yang menunjukkan bobot antar setiap pasangan

simpul. Tabel ini akan disajikan dalam format yang rapi dan mudah dibaca, dengan nama simpul sebagai baris dan kolom, serta nilai bobot sebagai isi sel-sel tabel. Setelah menampilkan tabel, program akan membersihkan memori yang telah dialokasikan secara dinamis untuk menyimpan data simpul dan bobot. Proses ini dilakukan untuk mencegah kebocoran memori (memory leak) yang dapat terjadi jika memori yang dialokasikan tidak dibebaskan dengan benar.

## 2. Guided 2

Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

### Source code

```
#include <iostream>
#include <string>

using namespace std;

// Node tree
struct Node {
    string data;
    Node* left;
    Node* right;
};

// Fungsi untuk membuat node baru
Node* createNode(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Fungsi untuk menambahkan node ke tree
Node* insertNode(Node* root, string data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data <= root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}
```

```

}

// Fungsi untuk menampilkan inorder traversal tree
void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    cout << root->data << " ";
    inorderTraversal(root->right);
}

// Fungsi untuk menampilkan child dari suatu node
void displayChild(Node* root, string parent) {
    if (root == NULL) return;
    if (root->data == parent) {
        if (root->left != NULL)
            cout << "Child kiri dari " << parent << ": " << root->
left->data << endl;
        if (root->right != NULL)
            cout << "Child kanan dari " << parent << ": " << root->
right->data << endl;
        return;
    }
    displayChild(root->left, parent);
    displayChild(root->right, parent);
}

// Fungsi untuk menampilkan descendant dari suatu node
void displayDescendant(Node* root, string parent) {
    if (root == NULL) return;
    if (root->data == parent) {
        cout << "Descendant dari " << parent << ": ";
        inorderTraversal(root->left);
        inorderTraversal(root->right);
        cout << endl;
        return;
    }
    displayDescendant(root->left, parent);
    displayDescendant(root->right, parent);
}

// Fungsi utama sesuai NIM
void zaki_2311102051() {
    Node* root = NULL;
    int choice;
    string data, parent;

    do {
        cout << "\nMenu:\n";

```

```

        cout << "1. Insert node\n";
        cout << "2. Display inorder traversal\n";
        cout << "3. Display child of a node\n";
        cout << "4. Display descendant of a node\n";
        cout << "5. Exit\n";
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data untuk node baru: ";
                cin >> data;
                root = insertNode(root, data);
                break;
            case 2:
                cout << "Inorder traversal tree: ";
                inorderTraversal(root);
                cout << endl;
                break;
            case 3:
                cout << "Masukkan nama node yang ingin ditampilkan
child-nya: ";
                cin >> parent;
                displayChild(root, parent);
                break;
            case 4:
                cout << "Masukkan nama node yang ingin ditampilkan
descendant-nya: ";
                cin >> parent;
                displayDescendant(root, parent);
                break;
            case 5:
                cout << "Terima kasih!\n";
                break;
            default:
                cout << "Pilihan tidak valid!\n";
        }
    } while (choice != 5);
}

int main() {
    zaki_2311102051();
    return 0;
}

```

**Screenshoot program**

```
Menu:
1. Insert node
2. Display inorder traversal
3. Display child of a node
4. Display descendant of a node
5. Exit
Pilihan Anda: 1
Masukkan data untuk node baru: Purwokerto
```

M. Aulia Muzzaki Nugraha  
2311102051

```
Menu:
1. Insert node
2. Display inorder traversal
3. Display child of a node
4. Display descendant of a node
5. Exit
Pilihan Anda: 2
Inorder traversal tree: Bumiayu Jogja Purwokerto
```

M. Aulia Muzzaki Nugraha  
2311102051

```
Menu:
1. Insert node
2. Display inorder traversal
3. Display child of a node
4. Display descendant of a node
5. Exit
Pilihan Anda: 3
Masukkan nama node yang ingin ditampilkan child-nya: Purwokerto
Child kiri dari Purwokerto: Bumiayu
```

M. Aulia Muzzaki Nugraha  
2311102051

```
Menu:
1. Insert node
2. Display inorder traversal
3. Display child of a node
4. Display descendant of a node
5. Exit
Pilihan Anda: 4
Masukkan nama node yang ingin ditampilkan descendant-nya: Bumiayu
Descendant dari Bumiayu: Jogja
```

M. Aulia Muzzaki Nugraha  
2311102051

### Deskripsi program

Di awal program, terdapat definisi struktur Node yang mewakili setiap node dalam binary tree. Setiap Node memiliki tiga bagian: data (nilai yang disimpan dalam node), left (pointer ke anak kiri), dan right (pointer ke anak kanan). Kemudian, terdapat beberapa fungsi seperti createNode untuk membuat node baru, insertNode untuk memasukkan node baru ke dalam binary tree, inorderTraversal untuk menampilkan

urutan traversal inorder, displayChild untuk menampilkan anak dari suatu node, dan displayDescendant untuk menampilkan descendant dari suatu node.

## **BAB IV**

### **KESIMPULAN**

Graf merupakan struktur data yang terdiri dari node (simpul) dan edge (sisi) yang menghubungkan antar node tersebut. Dalam praktikum, kita mempelajari bagaimana merepresentasikan graf dalam program, baik menggunakan matriks ketetanggaan maupun daftar ketetanggaan. Kita juga mempelajari algoritma-algoritma seperti pencarian lintasan terpendek dan traversal graf (BFS dan DFS).

Sementara itu, pohon (tree) merupakan jenis khusus dari graf yang tidak mengandung siklus. Dalam praktikum, kita mempelajari bagaimana struktur data pohon diimplementasikan dalam program, seperti binary tree, binary search tree, dan lain-lain. Kita juga mempelajari operasi-operasi dasar pada pohon seperti pencarian, penyisipan, dan penghapusan node.

Secara umum, praktikum ini memberikan pemahaman yang lebih baik tentang bagaimana graf dan pohon direpresentasikan dalam program dan bagaimana algoritma-algoritma yang terkait dengan keduanya bekerja. Praktikum ini juga meningkatkan kemampuan dalam mengimplementasikan struktur data dan algoritma dalam kode program.

## **DAFTAR PUSTAKA**

- [1] Munir, R. (2012). Matematika Diskrit. Bandung : Informatika.
- [2] Sembiring, R. R. (2022). Penerapan Algoritma Prim Dalam Menentukan Minimum Spanning Tree (MST).