

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA**

**MODUL 9  
GRAPH AND TREE**



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

**Disusun oleh:**

**FALAH ASBI PANGESTU**

**2311102045**

**IF-11-B**

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

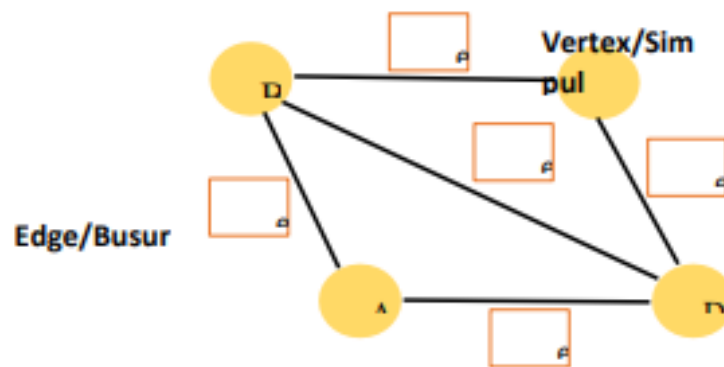
**2024**

# 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).

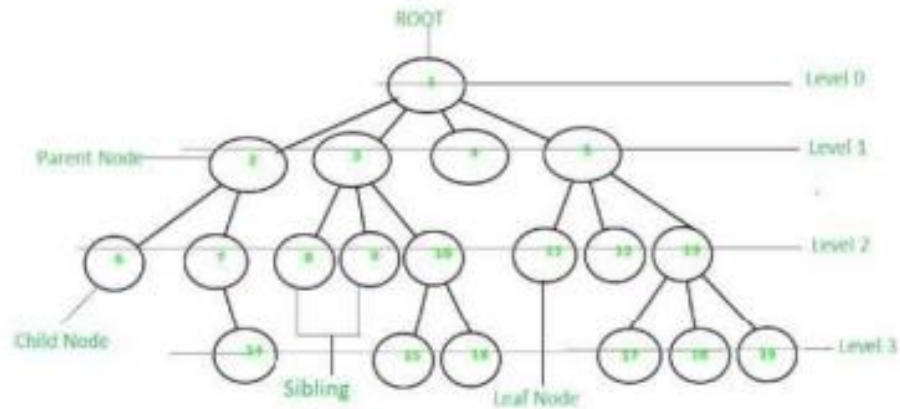


Gambar 1 Contoh Graph

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

### 2. Teori Tree

Pohon (tree) 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.



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

Tabel 1 Terminologi dalam Struktur Data Tree

## BAB II

### GUIDED

#### LATIHAN – GUIDED

##### 1. Guided 1

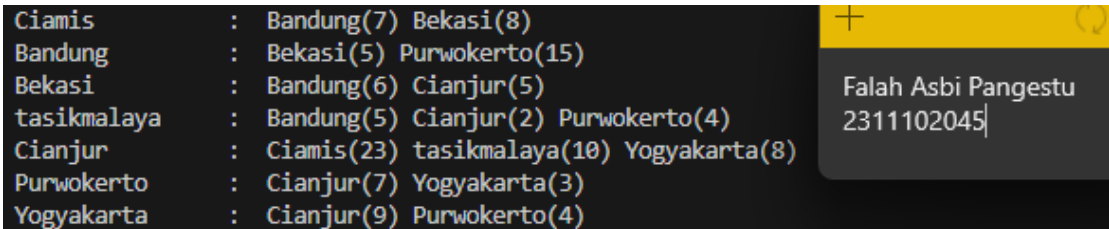
Program Graph

**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



```
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)
```

### Deskripsi program

Awalnya, program ini mendefinisikan dua array: satu untuk simpul yang berisi daftar nama kota atau tempat, dan satu lagi untuk busur yang mencakup jarak atau bobot antara setiap pasangan kota. Array simpul terdiri dari tujuh elemen, yang berarti graf memiliki tujuh simpul atau node. Sementara itu, array busur memiliki tujuh baris dan tujuh kolom, yang menunjukkan jarak antara setiap pasangan kota. Jika nilai pada busur[i][j] adalah nol, ini berarti tidak ada jalur langsung antara kota i dan kota j. Selanjutnya, program ini mendefinisikan fungsi bernama tampilGraph(), yang bertugas menampilkan representasi graf tersebut di layar. Fungsi ini menggunakan dua perulangan for bersarang untuk menelusuri setiap baris dan kolom dalam array busur. Untuk setiap baris, fungsi ini mencetak nama kota yang sesuai dari array simpul, diikuti dengan daftar kota-kota yang terhubung langsung beserta jaraknya dalam format 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
```

+

...

×

Falah Asbi Pangestu  
2311102045

```
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 ini juga memiliki 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 telah ditentukan. Fungsi `update(char data, Pohon *node)` digunakan untuk mengubah nilai data pada node tertentu. Untuk mengambil nilai data dari node tertentu, digunakan fungsi `retrieve(Pohon *node)`, sementara fungsi `find(Pohon *node)` memberikan informasi lebih detail tentang node tersebut, termasuk induk, saudara, dan anak-anaknya. Selain itu, program ini juga memiliki fungsi untuk melakukan traversal pada pohon biner, yaitu `preOrder(Pohon *node)`, `inOrder(Pohon *node)`, dan `postOrder(Pohon *node)`. Traversal ini bermanfaat untuk mengakses setiap node dalam pohon biner dengan urutan yang berbeda.

## 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 asbi_2311102045() {
    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() {
    asbi_2311102045();
    return 0;
}

```

### Screenshot Program

```

Silakan masukkan jumlah simpul: 2
Simpul 1: bandung
Simpul 2: purwokerto

Silakan masukkan bobot antar simpul
bandung--> bandung = 0
bandung--> purwokerto = 5
purwokerto--> bandung = 5
purwokerto--> purwokerto = 0

          bandung    purwokerto
bandung      0         5
purwokerto   5         0

```

### Deskripsi program

Pada awalnya, program meminta pengguna untuk memasukkan jumlah simpul yang diinginkan. Setelah itu, pengguna diminta untuk memasukkan nama-nama simpul satu per satu. Kemudian, pengguna harus menginput bobot untuk setiap pasangan simpul.

Setelah semua data dimasukkan, program akan menampilkan tabel yang menunjukkan bobot antara setiap pasangan simpul. Tabel ini disusun dengan rapi dan mudah dibaca, menampilkan nama-nama simpul pada baris dan kolom, serta bobot di dalam sel. Setelah tabel ditampilkan, program akan membersihkan memori yang digunakan untuk menyimpan data simpul dan bobot, untuk mencegah kebocoran memori yang mungkin terjadi jika memori 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 asbi_2311102045() {
    Node* root = NULL;
    int choice;
    string data, parent;

    do {
        cout << "\nMenu:\n";
        cout << "1. Masukkan Node\n";
        cout << "2. Menampilkan inorder traversal\n";
    } while (true);
}

```

```

        cout << "3. Menampilkan child of a node\n";
        cout << "4. Menampilkan descendant of a node\n";
        cout << "5. Keluar\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() {
    asbi_2311102045();
    return 0;
}

```

**Screenshoot program**

```
Menu:
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Banyumas
```

Falah Asbi Pangestu  
2311102045

```
Menu:
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 2
Inorder traversal tree: Banyumas Cilacap Kebumen Purwokerto
```

Falah Asbi Pangestu  
2311102045

```
Menu:
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 3
Masukkan nama node yang ingin ditampilkan child-nya: Banyumas
Child kanan dari Banyumas: Purwokerto
```

Falah Asbi Pangestu  
2311102045

```
Menu:
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 4
Masukkan nama node yang ingin ditampilkan descendant-nya: Purwokerto
Descendant dari Purwokerto: Cilacap Kebumen
```

Falah Asbi Pangestu  
2311102045

### Deskripsi program

Program dimulai dengan mendefinisikan struktur Node yang mewakili setiap node dalam pohon biner. Setiap node memiliki tiga anggota data: data (nilai yang disimpan dalam node), left (pointer ke anak kiri), dan right (pointer ke anak kanan). Kemudian, program mendefinisikan beberapa fungsi untuk melakukan operasi-operasi pada pohon biner, seperti createNode untuk membuat node baru, insertNode untuk menyisipkan node baru ke dalam pohon, inorderTraversal untuk menampilkan traversal inorder dari pohon, displayChild untuk menampilkan anak-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.

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] Asisten Praktikum. Modul 9 Graph dan Tree, Learning Management System 2024