

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



Disusun Oleh :

NAMA : RIFKY DWI MAHARDIKA
NIM : 2311102043

Dosen

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

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

A. Dasar Teori

1. Graph

Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). Graph dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graph adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge).

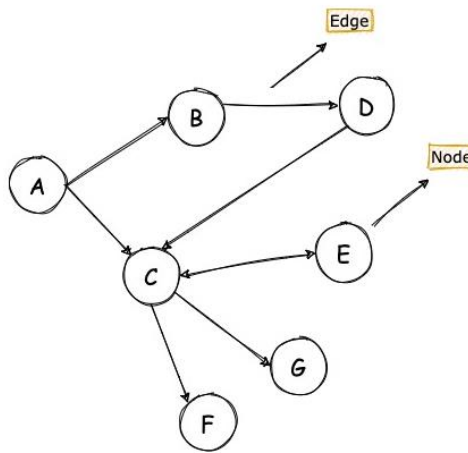
$$G = (V, E)$$

Dimana :

G = Graph

V = Simpul atau Vertex, atau Node, atau Titik

E = Busur atau Edge, atau arc



Gambar 1.0, Contoh Representasi Graf

Dalam pemrograman, agar data yang ada dalam graph dapat diolah, maka graph harus dinyatakan dalam suatu struktur data yang dapat mewakili graph tersebut. Dalam hal ini graph perlu direpresentasikan kedalam bentuk array dan dimensi yang sering disebut matrix atau direpresentasikan dalam bentuk linked list. Bentuk mana yang dipilih biasanya tergantung kepada efisiensi dan kemudahan dalam membuat program.

Representasi dalam bentuk matrix, antara lain:

- Adjacency Matrix Graph tak berarah

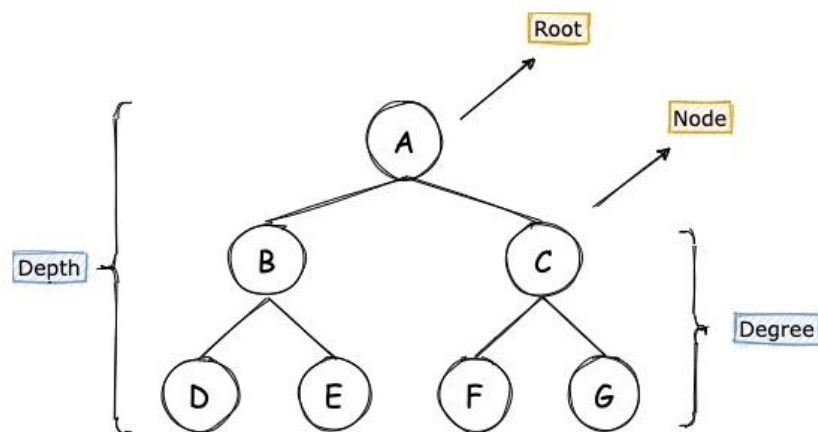
- Adjacency Matrix Graf Berarah
- Adjacency Matrix Graph berbobot tak Berarah

Representasi dalam bentuk Linked List, antara lain:

- Adjacency List

2. Tree

Tree adalah struktur data non linier berbentuk hierarki yang terdiri dari sekumpulan node yang berbeda.



Gambar 2.0, Contoh Representasi Tree

- Node, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri.
- Root, adalah sebuah node yang terletak di posisi tertinggi atau urutan pertama dari suatu tree.
- Depth, adalah jarak atau ketinggian antara root dan node.
- Degree, adalah banyaknya anak atau turunan dari suatu node.

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;
}
```

Screenshots Output

```
PS C:\Users\ASUS\Documents\semester 2\struktur data algoritma\laprak 9> cd "c:\Users\ASUS\Documents\semester 2\struktur data algoritma\laprak 9\" ; if ($?) { g++ guided1.cpp -o guided1 } ; if ($?) { .\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)
```

+

NAMA : Rifky Dwi Mahardika
NIM : 2311102043

Deskripsi :

Program tersebut merepresentasikan dan menampilkan graf berarah berbobot menggunakan array dua dimensi dan array string. Dengan pustaka iostream untuk input-output dan iomanip untuk format keluaran, program ini mendeklarasikan simpul-simpul graf dalam array “simpul[7]” dan bobot busur dalam array “busur[7][7]”. Fungsi “tampilGraph()” menampilkan graf dengan menunjukkan setiap simpul beserta busur dan bobotnya, menggunakan loop bersarang untuk memeriksa setiap koneksi antara simpul-simpul. Fungsi main() memanggil “tampilGraph()” untuk menampilkan hasilnya, yang menunjukkan koneksi antara simpul-simpul seperti "Ciamis : Bandung(7) Bekasi(8)" dan seterusnya, memberikan representasi visual yang jelas dari graf yang telah didefinisikan.

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();
}

```

Screenshots Output

```

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kananA

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kananB

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kananE

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kananG

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

```

+

NAMA : Rifky Dwi Mahardika
NIM : 2311102043

```
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.
```

+

NAMA : Rifky Dwi Mahardika
NIM : 2311102043

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
```

+

NAMA : Rifky Dwi Mahardika
NIM : 2311102043|

Deskripsi

Program ini untuk berbagai operasi pada struktur data pohon biner (binary tree). Program ini mencakup fungsi-fungsi untuk inisialisasi, penambahan node, pembaruan data node, pencarian node, traversal pohon, penghapusan pohon atau subtree, serta pengecekan ukuran dan tinggi pohon. Dimulai dengan deklarasi struktur Pohon yang memiliki data karakter dan pointer ke anak kiri, anak kanan, serta parent. Fungsi init menginisialisasi pohon menjadi kosong, dan isEmpty memeriksa apakah pohon kosong. Fungsi buatNode membuat node baru sebagai root jika pohon kosong, sedangkan insertLeft dan insertRight menambahkan node di anak kiri atau kanan dari node tertentu. Fungsi update memperbarui data pada node tertentu, retrieve menampilkan data node, dan find menampilkan detail

node termasuk parent, sibling, dan child. Program juga menyediakan fungsi traversal preOrder, inOrder, dan postOrder untuk mengunjungi node dalam urutan tertentu. Fungsi deleteTree dan deleteSub menghapus seluruh pohon atau subtree, sementara clear menghapus seluruh pohon. Fungsi size dan height mengembalikan ukuran dan tinggi pohon. Fungsi characteristic menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node per level. Program ini diakhiri dengan beberapa operasi pada pohon yang dimulai dengan root 'A' dan diikuti oleh penambahan beberapa node lainnya, pembaruan node, pencarian node, traversal, dan penghapusan subtree, serta menampilkan karakteristik pohon sebelum dan setelah penghapusan subtree.

C. UNGUIDED

Unguided 1

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

using namespace std;

int main()
{
    int verCount;

    cout << "Masukkan jumlah simpul : ";
    cin >> verCount;

    string vertecies[verCount];
    int edgeValues[verCount][verCount];
    cout << "Masukkan nama simpul,\n";
    for (int i = 0; i < verCount; i++)
    {
        cout << "Simpul " << i + 1 << " : ";
        cin >> vertecies[i];
    }

    cout << "Masukkan bobot antar simpul,\n";
    for (int i = 0; i < verCount; i++)
    {
        for (int j = 0; j < verCount; j++)
        {
            cout << vertecies[i] << "->" << vertecies[j] << " : ";
            cin >> edgeValues[i][j];
        }
    }

    cout << endl
         << setw(10) << " ";
    for (int i = 0; i < verCount; i++)
    {
        cout << setw(10) << vertecies[i];
    }
    cout << endl;

    for (int i = 0; i < verCount; i++)
    {
        cout << setw(10) << vertecies[i];
```

```

        for (int j = 0; j < verCount; j++)
        {
            cout << setw(10) << edgeValues[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Screenshots Output

```

PS C:\Users\ASUS\Documents\semester 2\struktur data algoritma\1
ak 9\" ; if ($?) { g++ unguided1.cpp -o unguided1 } ; if ($?) {
Silahkan masukkan jumlah simpul : 2
Silahkan masukkan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Masukkan bobot antar simpul
BALI--->BALI : 0
BALI--->PALU : 3
PALU--->BALI : 4
PALU--->PALU : 0

```

	BALI	PALU
BALI	0	3
PALU	4	0

Deskripsi

Program ini memungkinkan user untuk membuat dan mengisi matriks bobot graf berarah. Dimulai dengan meminta jumlah simpul yang diinginkan, pengguna kemudian memasukkan nama setiap simpul yang disimpan dalam array. Selanjutnya, pengguna diminta untuk memasukkan bobot antar setiap pasangan simpul, termasuk dari simpul ke dirinya sendiri, yang disimpan dalam matriks. Setelah semua data dimasukkan, program menampilkan matriks bobot dalam format tabel yang rapi, dengan nama simpul sebagai header kolom dan baris. Penggunaan fungsi “setw” dari pustaka “iomanip” membantu meratakan kolom untuk tampilan yang lebih terstruktur dan mudah dibaca, menyediakan representasi visual dari graf berarah berbobot yang telah dibuat oleh user.

Unguided 2

```
#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;
        else
            cout << "Child kiri dari " << parent << ": (tidak ada)" <<
endl;
        if (root->right != NULL)
            cout << "Child kanan dari " << parent << "Child kanan dari "
<< parent << ": " << root->right->data << endl;
        else
            cout << "Child kanan dari " << parent << ": (tidak ada)" <<
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 opsi()
{
    Node *root = NULL;
    int choice;
    string data, parent;

    do
    {
        cout << "\n===== \n";
        cout << "                MENU                \n";
        cout << "===== \n";
        cout << "1. Tambah node\n";
        cout << "2. Tampilkan traversal inorder\n";
        cout << "3. Tampilkan child dari suatu node\n";
        cout << "4. Tampilkan descendant dari suatu node\n";
        cout << "5. Keluar\n";
        cout << "----- \n";
        cout << "Pilih operasi yang ingin dilakukan: ";
        cin >> choice;

        switch (choice)
        {
            case 1:
                cout << "\nMasukkan data untuk node baru: ";
                cin >> data;
                root = insertNode(root, data);
                cout << "Node " << data << " berhasil ditambahkan.\n";
                break;
            case 2:
                cout << "\nInorder traversal tree: ";
                inorderTraversal(root);
                cout << endl;
                break;
            case 3:
                cout << "\nMasukkan nama node yang ingin ditampilkan child-
nya: ";

                cin >> parent;
                displayChild(root, parent);
                break;

```

```

        case 4:
            cout << "\nMasukkan nama node yang ingin ditampilkan
descendant-nya: ";
            cin >> parent;
            displayDescendant(root, parent);
            break;
        case 5:
            cout << "\nAnda telah keluar dari aplikasi.Terima kasih!\n";
            break;
        default:
            cout << "\nPilihan tidak valid! Silakan coba lagi.\n";
        }
    } while (choice != 5);
}

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

```

Screenshots Output

- **Menu 1**

The screenshot shows the output of a C++ program. It displays a menu with five options: 1. Tambah node, 2. Tampilkan traversal inorder, 3. Tampilkan child dari suatu node, 4. Tampilkan descendant dari suatu node, and 5. Keluar. The user has selected option 1. The program then prompts the user to enter data for a new node, and the user has entered 'a'. The program confirms that node 'a' has been successfully added.

```

=====
                        MENU
=====
1. Tambah node
2. Tampilkan traversal inorder
3. Tampilkan child dari suatu node
4. Tampilkan descendant dari suatu node
5. Keluar
-----
Pilih operasi yang ingin dilakukan: 1

Masukkan data untuk node baru: a
Node a berhasil ditambahkan.

```

- Menu 2

```
=====
                        MENU
=====
1. Tambah node
2. Tampilkan traversal inorder
3. Tampilkan child dari suatu node
4. Tampilkan descendant dari suatu node
5. Keluar
-----
Pilih operasi yang ingin dilakukan: 2

Inorder traversal tree: a b c d e
```

- Menu 3

```
=====
                        MENU
=====
1. Tambah node
2. Tampilkan traversal inorder
3. Tampilkan child dari suatu node
4. Tampilkan descendant dari suatu node
5. Keluar
-----
Pilih operasi yang ingin dilakukan: 3

Masukkan nama node yang ingin ditampilkan child-nya: a
Child kiri dari a: (tidak ada)
Child kanan dari aChild kanan dari a: b
```

- Menu 4


```
=====
                        MENU
=====
1. Tambah node
2. Tampilkan traversal inorder
3. Tampilkan child dari suatu node
4. Tampilkan descendant dari suatu node
5. Keluar
-----
Pilih operasi yang ingin dilakukan: 4

Masukkan nama node yang ingin ditampilkan descendant-nya: c
Descendant dari c: d e
```

- **Menu 5**

```
=====
                        MENU
=====
1. Tambah node
2. Tampilkan traversal inorder
3. Tampilkan child dari suatu node
4. Tampilkan descendant dari suatu node
5. Keluar
-----
Pilih operasi yang ingin dilakukan: 5

Anda telah keluar dari aplikasi.Terima kasih!
```



NAMA : Rifky Dwi Mahardika
NIM : 2311102043

Deskripsi

Program ini memungkinkan user untuk berinteraksi dengan tree melalui berbagai pilihan menu yang disediakan. Setiap node dalam tree direpresentasikan oleh struktur “Node” yang memiliki data dan pointer ke child kiri dan kanan. Fungsi utama “opsi()” menyediakan menu interaktif di mana pengguna dapat melakukan operasi seperti menambah node baru, menampilkan traversal inorder tree, serta menampilkan child dan descendant dari suatu node. Pengguna dapat memilih operasi yang diinginkan dengan memasukkan nomor menu yang sesuai. Pada fungsi utama opsi(), pengguna diberikan beberapa opsi menu yang terdiri dari:

1. Menambahkan node baru ke pohon.
2. Menampilkan traversal inorder dari pohon.
3. Menampilkan child dari suatu node.
4. Menampilkan descendant dari suatu node.
5. Keluar dari aplikasi.

Setiap opsi akan mengarahkan pengguna ke fungsi yang sesuai dan melakukan operasi yang diminta. Program akan terus berjalan sampai pengguna memilih untuk keluar dari aplikasi.

D. Kesimpulan

Graph dan tree adalah dua konsep penting dalam pemrograman. Graph adalah struktur data yang terdiri dari simpul yang terhubung oleh sisi, yang dapat memiliki berbagai jenis, seperti directed, undirected, weighted, cyclic, atau acyclic. Digunakan untuk merepresentasikan jaringan, relasi, atau struktur data non-linear lainnya, graf memiliki beragam algoritma seperti DFS, BFS, Dijkstra, Prim, Kruskal, dan lainnya untuk penelusuran, pencarian jalur terpendek, atau pembentukan minimum spanning tree. Di sisi lain, tree adalah struktur data hierarkis yang terdiri dari simpul yang terhubung dalam sebuah hirarki, di mana setiap simpul memiliki satu induk kecuali simpul root, dan dapat berbagai macam, termasuk binary tree, binary search tree, balanced tree, dan heap. Pohon digunakan untuk merepresentasikan hierarki data seperti struktur folder, dokumen XML, dan struktur data terorganisir lainnya, dengan algoritma seperti traversal, pencarian biner, dan pengelolaan BST. Pemahaman yang kuat tentang graph dan tree penting dalam merancang dan menerapkan algoritma yang efisien dalam pemrograman.

E. Referensi

[1] Asisten Praktikum, “Modul 9 Graph dan Tree”

[2] ABHISHEK SHARMA. 18 JANUARI 2023. Perbedaan Antara Pohon dan Grafik dalam Struktur Data. Diakses pada 9 Juni 2024, dari https://www-prepbytes-com.translate.goog/blog/data-structure/difference-between-tree-and-graph-in-data-structure/? x tr sl=en& x tr tl=id& x tr hl=id& x tr_pto=tc

[3] Hadari Blog. 31 Mei 2019. Graf (Graph) dan Pohon (Tree) - Algoritma Pemrograman. Diakses pada 9 Juni 2024, dari <http://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html>

[4] Aside. 10 NOVEMBER 2020. Data Structure : Mengenal Graph & Tree.

Diakses pada 9 Juni 2024, dari <https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>