

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



Disusun Oleh :

NAMA : ANISA YASAROH

NIM : 2311102063

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

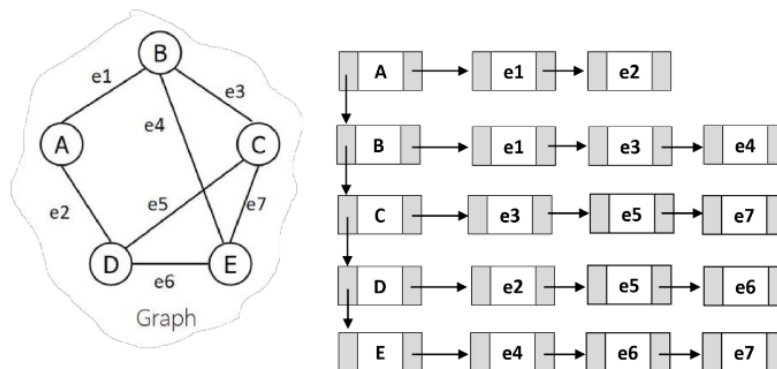
Graf merupakan struktur data nonlinier yang terdiri dari simpul dan sisi. Simpul terkadang juga disebut sebagai simpul dan sisinya adalah garis atau busur yang menghubungkan dua simpul mana pun dalam grafik. Secara lebih formal, Graf terdiri dari himpunan simpul (V) dan himpunan sisi (E). Grafik tersebut dilambangkan dengan $G(V, E)$.

- 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. 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

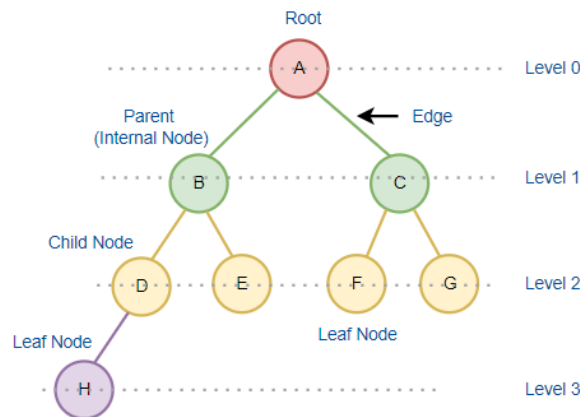
Tree adalah struktur data yang terdiri dari simpul-simpul yang saling terhubung melalui edge, menyimpan nilai, dan dapat diakses melalui satu simpul induk. Jika simpul tidak memiliki anak, disebut leaf node. Ini memungkinkan pengaturan data yang efektif di komputer.

- Istilah-istilah pada Tree

Layaknya sebuah pohon yang memiliki akar, cabang, dan daun yang terhubung satu sama lain, pada struktur data tree terdapat beberapa istilah, antara lain :

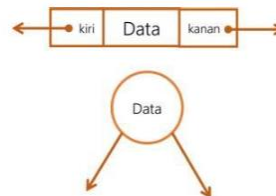
1. Node
2. Child node
3. Leaf Node

4. Root
5. Internal node
6. Edge
7. Height of node
8. Depth of node
9. Height of tree
10. Degree of node
11. Subtree



Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



- 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.

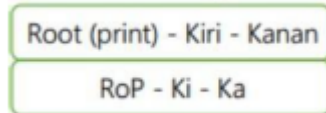
- i. **Characteristic:** digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.
- j. **Traverse:** digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, 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:

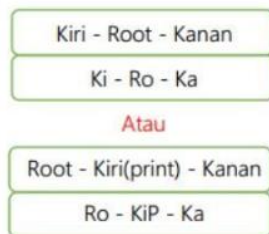


2. In-Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
- b. Cetak data pada root
- c. Secara rekursif mencetak seluruh data pada subpohon kanan.

Dapat kita turunkan rumus penelusuran menjadi:

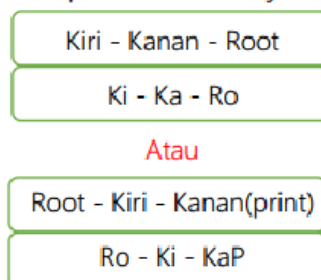


3. Post Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
- b. Secara rekursif mencetak seluruh data pada subpohon kanan
- c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



1. GUIDED

1. Guided 1 – Program Graph

Source Code

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

using namespace std;

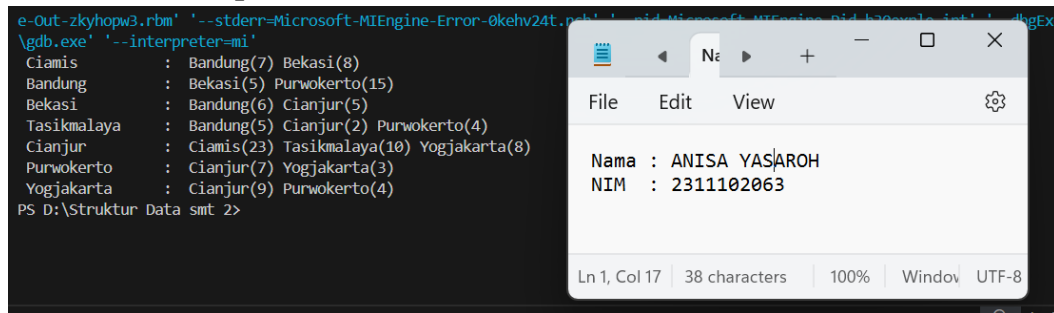
string simpul [7] =
    {"Ciamis", "Bandung", "Bekasi", "Tasikmalaya",
"Cianjur", "Purwokerto", "Yogjakarta"};

int busur[7][7] = {
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}
};

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

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

Screenshoot Output



Deskripsi Program

Program di atas yaitu mendefinisikan sebuah graph menggunakan matriks ketetanggaan yang menggambarkan beberapa kota dan jarak antar kota dalam sebuah array dua dimensi bernama 'busur'. Terdapat juga array satu dimensi 'simpul' yang memuat nama-nama kota. Fungsi 'tampilGraph' bertugas menampilkan graf dengan mengiterasi setiap baris dan kolom dari matriks 'busur', lalu mencetak nama kota beserta jarak (bobot) ke kota-kota yang terhubung. Pada fungsi 'main', fungsi 'tampilGraph' dipanggil untuk menampilkan representasi graf tersebut di layar. Hasilnya adalah daftar kota dengan koneksi dan bobot jarak ke kota lain yang terhubung.

2. Guided 2 – Program Tree

Source Code

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

using namespace std;

struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

void init()
{
    root = NULL;
}

bool isEmpty()
{
    return root == NULL;
}
```

```

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
sebagai root."
        << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

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)
        {
            //kalo ada
            cout << "\n Node " << node->data << " sudah ada
child kiri !" << endl;
            return NULL;
        }
        else
        {
            //kalo gada
            Pohon *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
    {
        //cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            //kalo ada
            cout << "\n Node " << node->data << " sudah ada
child kanan !" << endl;
            return NULL;
        }
        else
        {
            //kalo gada
            Pohon *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;
        }
    }
}

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;
    }
}
}

void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

void find(Pohon *node)
{
    if (isEmpty())
    {
        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;
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (isEmpty())
    {
        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 (isEmpty())
    {
        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 (isEmpty())
    {
        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 (isEmpty())
    {

```

```

        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 (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

void clear()
{
    if (isEmpty())
    {
        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 (isEmpty())
    {
        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 (isEmpty())
    {
        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 characteristic()
{
    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);
    characteristic();

    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;
}

```

Screenshoot Output

```

PS D:\Struktur Data smt 2> & 'c:\Users\ASUS\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\ft-MIEngine-In-ummcogd.rys' '--stdout=Microsoft-MIEngine-Out-35defqxo.xa0' '--stderr=Microsoft-MIEngine-Error-0errp4r23.oix' '--dbgExe=C:\Program Files\CodeBlocks\MinGW\bin\gdb.exe' '--interpreter=mi'

Node A berhasil dibuat sebagai root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kanan G

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C

Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)

Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

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,

PS D:\Struktur Data smt 2>

```

Deskripsi Program

Program ini yaitu mendefinisikan dan mengimplementasikan berbagai operasi pada struktur data pohon biner, termasuk pembuatan node baru, penambahan child kiri dan kanan, pengubahan dan pengambilan nilai node, pencarian informasi tentang node tertentu, traversal pohon menggunakan metode pre-order, in-order, dan post-order, penghapusan pohon dan subtree, serta menghitung ukuran dan tinggi pohon, lalu mengujinya dengan membuat, memodifikasi, dan menampilkan informasi tentang sebuah pohon biner yang terdiri dari beberapa node.

2. UNGUIDED

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

Output Program

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

      BALI    PALU
BALI    0      3
PALU    4      0

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

Source Code

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

using namespace std;

int main(){
    int Anisa_2311102063;
    cout << "Silakan masukkan jumlah simpul : ";
    cin >> Anisa_2311102063;

    vector<string> simpul(Anisa_2311102063);
    vector<vector<int>> busur(Anisa_2311102063,
vector<int>(Anisa_2311102063, 0));
    cout << "Silakan masukkan nama simpul " << endl;
    for (int a = 0; a < Anisa_2311102063; a++){
        cout << "Simpul " << (a + 1) << " : ";
        cin >> simpul[a];
    }
    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int a = 0; a < Anisa_2311102063; a++){
        for (int b = 0; b < Anisa_2311102063; b++){
            cout << simpul[a] << " --> " << simpul[b] << " = ";
            cin >> busur[a][b];
        }
    }
    cout << endl;
    cout << setw(7) << " ";
    for (int a = 0; a < Anisa_2311102063; a++){
        cout << setw(8) << simpul[a];
```



```

    }
    cout << endl;
    for (int a = 0; a < Anisa_2311102063; a++){
        cout << setw(7) << simpul[a];
        for (int b = 0; b < Anisa_2311102063; b++){
            cout << setw(8) << busur[a][b];
        }
        cout << endl;
    }
}

```

Screenshoot Output

The screenshot shows a terminal window on the left and a Notepad++ window on the right. The terminal window displays the following output:

```

PS D:\Struktur Data smt 2> & 'c:\Users\ASUS\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\bin\code.cmd' --stdin=Microsoft-MIEngine-In-e13hyv24.coa' '--stdout=Microsoft-MIEngine-Out-pm1qyky1.b5k' '--stderr=Microsoft-MIEngine-Pid-lqgsuxsi.vvz' '--dbgExe=C:\Program Files\CodeBlocks\MinGW\bin\gdb.exe' '--i
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 D:\Struktur Data smt 2>

```

The Notepad++ window shows the following text:

```

Nama : ANISA YASAROH
NIM  : 2311102063

```

Deskripsi Program

Program di atas yaitu implementasi graf berbobot. Pengguna diminta untuk memasukkan jumlah simpul (node) pada graf dan kemudian memberikan nama untuk setiap simpul tersebut. Setelah itu, pengguna memasukkan bobot (weight) antara setiap pasangan simpul, yang disimpan dalam matriks dua dimensi `busur`. Program kemudian menampilkan matriks ketetanggaan (adjacency matrix) dari graf, yang menunjukkan bobot antara setiap pasangan simpul. Matriks ini ditampilkan dalam bentuk tabel yang rapi menggunakan fungsi `setw` untuk mengatur lebar kolomnya.

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 Anisa_2311102063() {
    Node* root = NULL;
    int choice;
    string data, parent;

    do {
        cout << "\n-----" << endl;
        cout << "                MENU                " << endl;
        cout << "-----" << endl;
        cout << "1. Masukan Node " << endl;
        cout << "2. Menampilkan inorder traversal " << endl;
        cout << "3. Menampilkan child of a node " << endl;
        cout << "4. Menampilkan descendant of a node " << endl;
        cout << "5. Keluar " << endl;
        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() {
    Anisa_2311102063();
    return 0;
}

```

Screenshoot Output

```

PS D:\Struktur Data smt 2> & 'c:\Users\ASUS\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\deb
wsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-vrogf0hk.4me' '--stdout=Microsoft-MIEngine-Out-323y
Microsoft-MIEngine-Error-qecq4lce.lma' '--pid=Microsoft-MIEngine-Pid-cuj24vrr.pth' '--dbgExe=C:\Program
nGW\bin\gdb.exe' '--interpreter=mi'

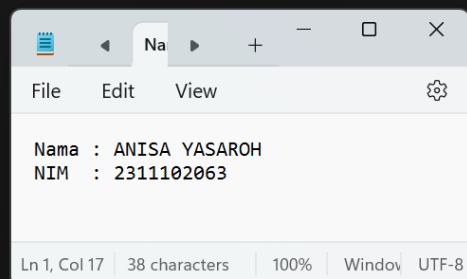
```

```

-----
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: Pekalongan

-----
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: Semarang

```

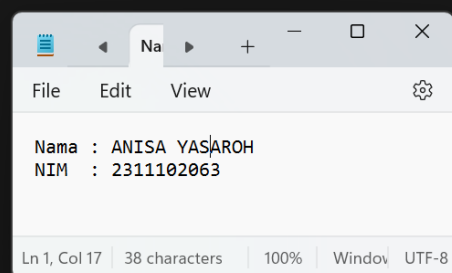


```

-----
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: Yogyakarta

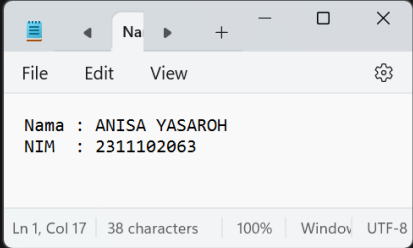
-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 2
Inorder traversal tree: Pekalongan Semarang Yogyakarta

```

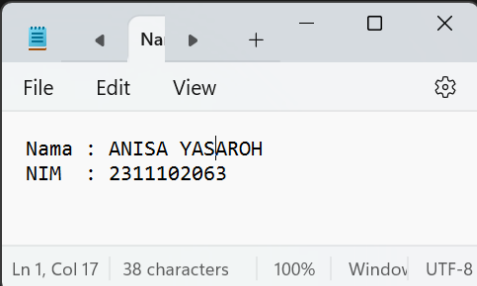


```
-----
                        MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 3
Masukkan nama node yang ingin ditampilkan child-nya: Semarang
Child kanan dari Semarang: Yogyakarta

-----
                        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: Pekalongan
Descendant dari Pekalongan: Semarang Yogyakarta
```



```
-----
                        MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 5
Terima kasih!
PS D:\Struktur Data smt 2> █
```



Deskripsi Program

Program diatas yaitu pengguna untuk melakukan berbagai operasi pada pohon. Pengguna dapat menambahkan node baru, menampilkan traversal inorder dari pohon, serta melihat anak-anak (child) dan keturunan (descendant) dari suatu node tertentu. Struktur `Node` digunakan untuk merepresentasikan setiap node dalam pohon, dengan fungsi-fungsi `createNode`, `insertNode`, `inorderTraversal`, `displayChild`, dan `displayDescendant` untuk menangani berbagai operasi pada pohon tersebut. Program ini juga menyediakan menu interaktif yang memungkinkan pengguna memilih operasi yang diinginkan hingga memilih opsi untuk keluar dari program.

3. KESIMPULAN

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa :

1. Graph adalah struktur data yang terdiri dari simpul (node) yang terhubung oleh edge (sisi).
2. Graph dapat digunakan untuk memodelkan berbagai masalah, seperti jaringan komputer, jejaring sosial, rute perjalanan, dan lainnya.
3. Graph dapat digunakan untuk melakukan berbagai operasi, seperti traversal (melintasi) graph, mencari jalur terpendek, menemukan siklus, dan lainnya.
4. Graph dapat diimplementasikan menggunakan representasi adjacency matrix atau adjacency list, tergantung pada kebutuhan dan jenis operasi yang akan dilakukan.

4. REFERENSI

Alia, P. A., & S ST, M. T. (2023). Dasar-Dasar Pemrograman.

Anita Sindar, R. M. S. (2019). Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY.

FIRLIANA, R., & Kasih, P. (2019). Algoritma dan Pemrograman C++.

Hanief, S., Jepriana, I. W., & Kom, S. (2020). Konsep Algoritme dan Aplikasinya dalam Bahasa Pemrograman C++. Penerbit Andi.

Nurhadi, N., Jatmiko, A. R., Legito, L., Saputra, E. A., Surianto, D. F., Komalasari, R., ... & Zain, N. N. L. E. (2023). BUKU AJAR LOGIKA & ALGORITMA. PT. Sonpedia Publishing Indonesia.