

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



**Disusun Oleh :
WIDARI DWI HAYATI
2311102060**

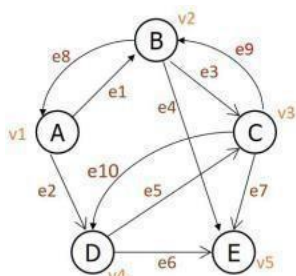
**Dosen :
WAHYU ANDI SAPUTRA, S.Pd., M.Eng.**

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

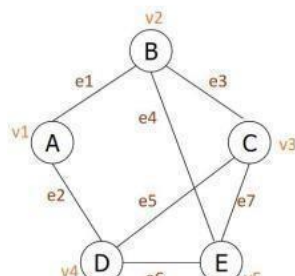
A. Dasar Teori

Graf atau graph adalah struktur data yang digunakan untuk menggambarkan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Graf terdiri dari simpul dan busur yang dapat diwakili secara matematis sebagai $G = (V, E)$, dimana G adalah Graph, V adalah simpul atau vertex, dan E adalah sisi atau edge. Graf dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

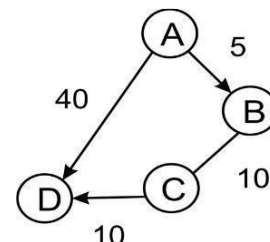
Terdapat beberapa jenis graf, yaitu graf berarah, graf tak berarah, dan graf berbobot. Graf berarah (directed graph) memiliki urutan simpul yang penting, sedangkan graf tak berarah (undirected graph) tidak memperhatikan urutan simpul. Graf berbobot (weight graph) memiliki nilai pada tiap edgenya.



Directed Graph

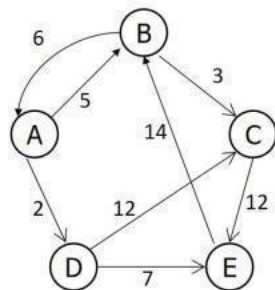


Undirected Graph



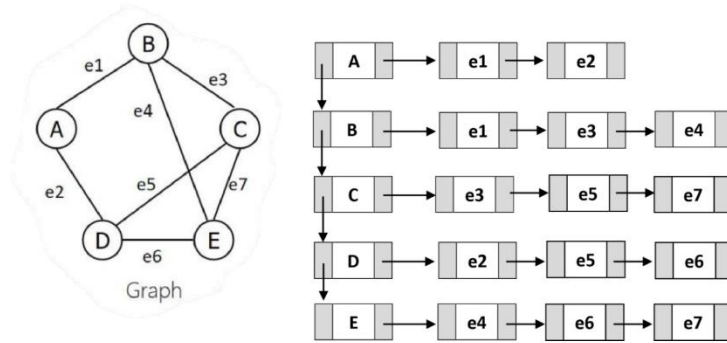
Weight Graph

Representasi graph dalam bentuk matriks

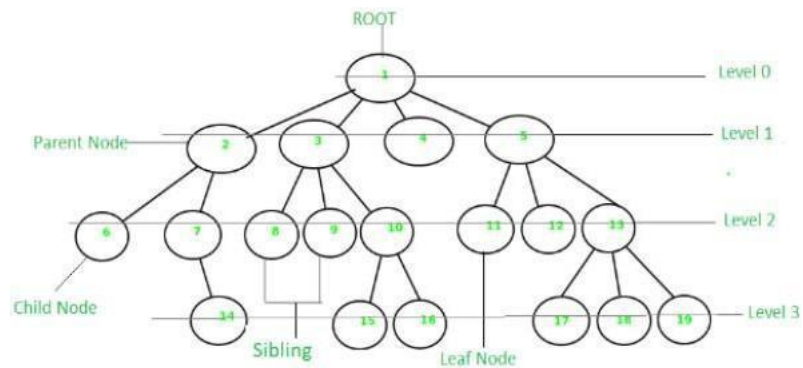


		Ke →				
		A	B	C	D	E
Dari ↑	A 0	0	5	0	2	0
	B 1	6	0	3	0	0
	C 2	0	0	0	0	9
	D 3	0	0	12	0	7
	E 4	0	14	0	0	0

Representasi graf dalam bentuk linked list memerlukan pemahaman yang baik tentang perbedaan antara simpul vertex dan simpul edge. Simpul vertex mewakili titik atau simpul dalam graf, sedangkan simpul edge mewakili hubungan antara simpul-simpul tersebut.



Pohon atau tree adalah struktur data yang sangat umum dan kuat yang menyerupai pohon nyata. Pohon terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, dimana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Pohon digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi.



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

Pohon biner adalah struktur data pohon yang setiap simpul dalam pohon diprasyarkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul. Operasi pada pohon biner meliputi :

- a. Create: Membuat pohon baru.
- b. Clear: Mengosongkan pohon.
- c. isEmpty: Memeriksa apakah pohon kosong.
- d. Insert: Memasukkan node ke pohon.
- e. Find: Mencari node tertentu.
- f. Update: Mengubah isi node.
- g. Retrieve: Mengambil isi node.
- h. Delete Sub: Menghapus subtree.
- i. Characteristic: Mengetahui karakteristik pohon (ukuran, ketinggian, rata-rata panjang).
- j. Traverse adalah operasi yang digunakan untuk mengunjungi seluruh node-node pada pohon dengan cara traversal. Terdapat 3 metode traversal yang dibahas, yaitu :
 - 1) Pre-Order: Kunjungi root, lalu kiri, lalu kanan.
 - 2) In-Order: Kunjungi kiri, root, kanan.
 - 3) Post-Order: Kunjungi kiri, kanan, root.

B. Guided

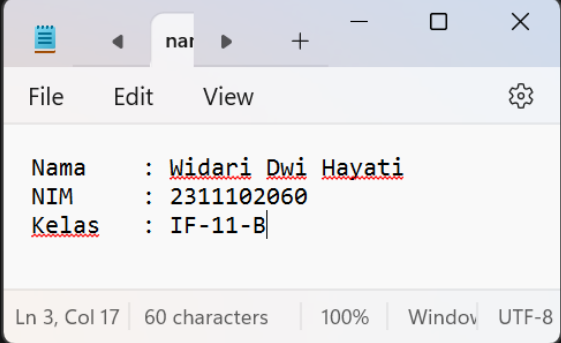
Program Graph

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;
}
```

Screenshots Output

```
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 C:\Users\ADVAN>
```



The screenshot shows a Notepad++ window with a single tab titled 'nar'. The window contains the following text:

```
Nama      : Widari Dwi Hayati
NIM       : 2311102060
Kelas    : IF-11-B
```

The status bar at the bottom of the window indicates 'Ln 3, Col 17', '60 characters', '100%', 'Window', and 'UTF-8'.

Deskripsi:

Program diatas mendefinisikan graf dengan 7 node, yang diwakili oleh array string simpul, yang berisi nama-nama kota di Indonesia. Matriks adjasensi graf diwakili oleh array 2D busur, di mana setiap elemen busur[i][j] merepresentasikan bobot edge antara node i dan j. Jika tidak ada edge antara dua node, elemen yang sesuai dalam matriks diatur ke 0. Program ini mendefinisikan fungsi tampilGraph() yang mencetak matriks adjasensi graf dan koneksi node-nya ke konsol. Fungsi ini mengiterasi setiap baris matriks, mencetak nama node dan node yang terkoneksi, serta bobot edge yang sesuai.

Dalam fungsi main(), program memanggil fungsi tampilGraph() untuk menampilkan representasi graf. Output program akan menampilkan matriks adjasensi graf dan koneksi node-nya, dengan setiap node terdaftar pada baris yang terpisah, diikuti oleh node yang terkoneksi dan bobot edge yang sesuai.

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;
}

```

Screenshots Output

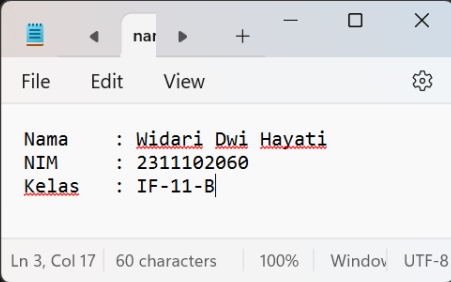
```
Node A berhasil dibuat sebagai root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kanan A
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kanan B
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri 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,

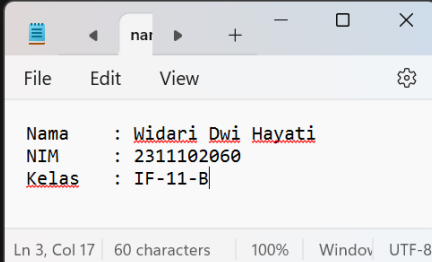
PS C:\Users\ADVAN>
```



The screenshot shows a Notepad window titled 'nar' with the following text:

```
Nama : Widari Dwi Hayati
NIM : 2311102060
Kelas : IF-11-B|
```

The status bar at the bottom of the Notepad window indicates: Ln 3, Col 17 | 60 characters | 100% | Window | UTF-8.



The screenshot shows a Notepad window titled 'nar' with the following text:

```
Nama : Widari Dwi Hayati
NIM : 2311102060
Kelas : IF-11-B|
```

The status bar at the bottom of the Notepad window indicates: Ln 3, Col 17 | 60 characters | 100% | Window | UTF-8.

Deskripsi:

Program ini mendefinisikan struktur Pohon untuk merepresentasikan node dalam pohon biner, yang memiliki field data karakter, dan pointer ke node anak kiri dan kanan, serta node induk.

Program ini menyediakan beberapa fungsi untuk beroperasi pada pohon biner, yaitu `init()`: Menginisialisasi pohon dengan mengatur node root ke NULL. `isEmpty()`: Memeriksa apakah pohon kosong. `buatNode(char data)`: Membuat node baru dengan data yang diberikan dan mengatur sebagai node root jika pohon kosong. `insertLeft(char data, Pohon *node)`: Menambahkan node baru dengan data yang diberikan sebagai anak kiri dari node yang ditentukan. `insertRight(char data, Pohon *node)`: Menambahkan node baru dengan data yang diberikan sebagai anak kanan dari node yang ditentukan. `update(char data, Pohon *node)`: Mengupdate data dari node yang ditentukan. `retrieve(Pohon *node)`: Mengambil data dari node yang ditentukan. `find(Pohon *node)`: Mencari node yang ditentukan dan menampilkan data, induk, saudara, dan anak node. `preOrder(Pohon *node)`, `inOrder(Pohon *node)`, dan `postOrder(Pohon *node)`: Melakukan traversal pre-order, in-order, dan post-order pada pohon, masing-masing. `deleteTree(Pohon *node)`: Menghapus node yang ditentukan dan subtree-nya. `deleteSub(Pohon *node)`: Menghapus subtree yang berakar pada node yang ditentukan. `clear()`: Menghapus pohon secara keseluruhan. `size(Pohon *node)`: Mengembalikan jumlah node dalam pohon. `height(Pohon *node)`: Mengembalikan tinggi pohon. `characteristic()`: Menampilkan ukuran, tinggi, dan derajat node rata-rata dari pohon.

Dalam fungsi `main()`, program ini membuat pohon biner dengan beberapa node dan melakukan berbagai operasi pada pohon, termasuk menambahkan node, mengupdate data node, mengambil data node, mencari node, dan menelusuri pohon. Output program akan menampilkan hasil dari berbagai operasi yang dilakukan pada pohon biner, termasuk struktur pohon, data node, dan hasil traversal.

C. Unguided/Tugas

*Cantumkan NIM pada salah satu variabel di dalam program.

Contoh : int nama_22102003;

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;

vector<string> simpul;
vector<vector<int>> busur;

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

int main()
{
```

```

int n, i, j;
string nama_2311102060;

cout << "Silakan masukan jumlah simpul : ";
cin >> n;

// Input nama simpul
for (i = 0; i < n; i++) {
    cout << "Simpul " << i + 1 << " : ";
    cin >> nama_2311102060;
    simpul.push_back(nama_2311102060);
}

// Inisialisasi matriks bobot
busur.resize(n, vector<int>(n, 0));

// Input bobot antar simpul
cout << "Silakan masukkan bobot antar simpul" << endl;
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        cout << simpul[i] << "--> " << simpul[j] << " = ";
        cin >> busur[i][j];
    }
}

// Tampilkan matriks bobot
tampilGraph();

// Hitung jarak antar simpul
string asal, tujuan;
cout << "Masukkan kota asal : ";
cin >> asal;
cout << "Masukkan kota tujuan : ";
cin >> tujuan;

int origin, destination;
for (i = 0; i < n; i++) {
    if (simpul[i] == asal) {
        origin = i;
        break;
    }
}
for (i = 0; i < n; i++) {
    if (simpul[i] == tujuan) {
        destination = i;
        break;
    }
}

```

```

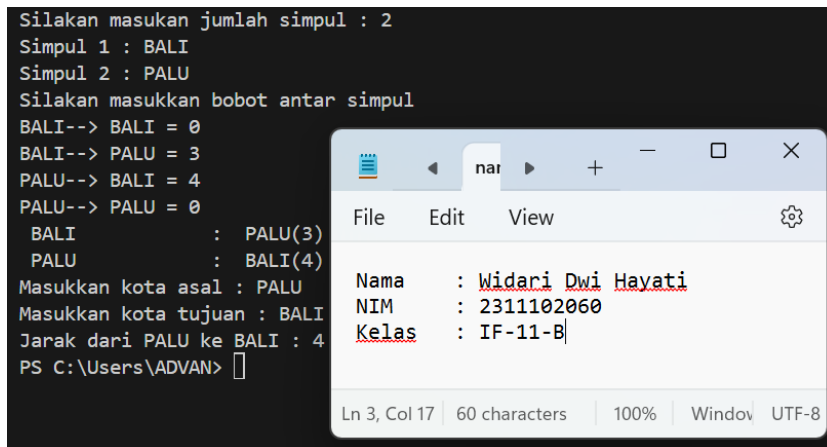
    }
    for (int k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (busur[i][k] + busur[k][j] < busur[i][j]) {
                    busur[i][j] = busur[i][k] + busur[k][j];
                }
            }
        }
    }
}

// Tampilkan jarak
cout << "Jarak dari " << asal << " ke " << tujuan << " : " << busur[origin][destination]
<< endl;

return 0;
}

```

Screenshots Output



```

Silakan masukan jumlah simpul : 2
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0
BALI      : PALU(3)
PALU      : BALI(4)
Masukkan kota asal : PALU
Masukkan kota tujuan : BALI
Jarak dari PALU ke BALI : 4
PS C:\Users\ADVAN>

```

Inset window content:

```

Nama      : Widari Dwi Hayati
NIM       : 2311102060
Kelas    : IF-11-B

```

Status bar: Ln 3, Col 17 | 60 characters | 100% | Window | UTF-8

Deskripsi:

Program ini menghitung jarak terpendek antara dua node dalam graf berbobot. Graf direpresentasikan sebagai matriks adjacency, di mana setiap elemen dalam matriks tersebut mewakili bobot dari edge antara dua node. Input berupa jumlah node (simpul) dalam graf, nama setiap node, bobot edge antara setiap pasangan node, node asal dan tujuan untuk menghitung jarak terpendek. Output berupa matriks adjacency graf, jarak terpendek antara node asal dan tujuan.

Program pertama-tama menginisialisasi graf kosong dengan jumlah node yang ditentukan. Kemudian, program meminta pengguna untuk memasukkan nama setiap node dan bobot edge antara setiap pasangan node. Program menyimpan graf sebagai matriks adjacency, di mana setiap elemen dalam matriks tersebut mewakili bobot dari edge antara dua node. Program kemudian meminta pengguna untuk memasukkan node asal dan tujuan untuk menghitung jarak terpendek. Program akan menghitung jarak terpendek antara setiap pasangan node dalam graf. Akhirnya, program mengeluarkan jarak terpendek antara node asal dan tujuan.

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 <iomanip>

using namespace std;

struct Pohon
{
    char data_2311102060;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

void init()
{
    root = NULL;
}

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

void buatNode(char data_2311102060)
{
    if (isEmpty())
    {
        root = new Pohon();
        root->data_2311102060 = data_2311102060;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data_2311102060 << " berhasil dibuat sebagai root."
            << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

Pohon *insertLeft(char data_2311102060, Pohon *node)
```

```

{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->data_2311102060 << " sudah ada child kiri !" <<
endl;
            return NULL;
        }
        else
        {
            Pohon *baru = new Pohon();
            baru->data_2311102060 = data_2311102060;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data_2311102060 << " berhasil ditambahkan ke child kiri
" << baru->parent->data_2311102060 << endl;
            return baru;
        }
    }
}

```

Pohon *insertRight(char data_2311102060, Pohon *node)

```

{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\n Node " << node->data_2311102060 << " sudah ada child kanan !"
<< endl;
            return NULL;
        }
        else
        {
            Pohon *baru = new Pohon();

```

```

        baru->data_2311102060 = data_2311102060;
        baru->left = NULL;
        baru->right = NULL;
        baru->parent = node;
        node->right = baru;
        cout << "\n Node " << data_2311102060 << " berhasil ditambahkan ke child
kanan " << baru->parent->data_2311102060 << endl;
        return baru;
    }
}

void update(char data_2311102060, 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_2311102060;
            node->data_2311102060 = data_2311102060;
            cout << "\n Node " << temp << " berhasil diubah menjadi "
                << data_2311102060 << 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_2311102060 << 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_2311102060 << endl;
            cout << " Root : " << root->data_2311102060 << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data_2311102060 << endl;
            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left->data_2311102060 << endl;
            else if (node->parent != NULL && node->parent->right != node && node->parent->left == node)
                cout << " Sibling : " << node->parent->right->data_2311102060 << endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << " Child Kiri : (tidakpunya Child kiri)" << endl;
            else
                cout << " Child Kiri : " << node->left->data_2311102060 << endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right->data_2311102060 << endl;
        }
    }
}

void displayChild(Pohon *node)

```



```

{
    if (node->left)
        cout << "Child kiri: " << node->left->data_2311102060 << endl;
    if (node->right)
        cout << "Child kanan: " << node->right->data_2311102060 << endl;
}

void displayDescendant(Pohon *node)
{
    if (node->left)
    {
        cout << "Descendant kiri: ";
        displayDescendant(node->left);
        cout << endl;
    }
    if (node->right)
    {
        cout << "Descendant kanan: ";
        displayDescendant(node->right);
        cout << endl;
    }
    if (!node->left && !node->right)
        cout << node->data_2311102060 << " ";
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data_2311102060 << " ";
            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_2311102060 << " ";
            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_2311102060 << " ";
        }
    }
}

// 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_2311102060 << " 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()
{
    init();
    char data_2311102060;
    cout << "Masukkan data root: ";
    cin >> data_2311102060;
    buatNode(data_2311102060);
    Pohon *node;
    int pilihan;
    while (true)
    {
        cout << "\n Menu:" << endl;
        cout << "1. Tambah node kiri" << endl;
        cout << "2. Tambah node kanan" << endl;
        cout << "3. Update node" << endl;
        cout << "4. Retrieve node" << endl;
        cout << "5. Find node" << endl;
        cout << "6. Display child" << endl;
        cout << "7. Display descendant" << endl;
        cout << "8. PreOrder" << endl;
        cout << "9. InOrder" << endl;
        cout << "10. PostOrder" << endl;
        cout << "11. Delete subtree" << endl;
        cout << "12. Clear tree" << endl;
        cout << "13. Characteristic" << endl;
        cout << "14. Exit" << endl;
        cout << "Pilihan: ";
        cin >> pilihan;
        switch (pilihan)
        {
            case 1:
            {
                char dataKiri;
                cout << "Masukkan data kiri: ";
                cin >> dataKiri;
                node = insertLeft(dataKiri, root);
            }
        }
    }
}

```

```
        break;
    }
    case 2:
    {
        char dataKanan;
        cout << "Masukkan data kanan: ";
        cin >> dataKanan;
        node = insertRight(dataKanan, root);
        break;
    }
    case 3:
    {
        char dataUpdate;
        cout << "Masukkan data update: ";
        cin >> dataUpdate;
        update(dataUpdate, root);
        break;
    }
    case 4:
    {
        retrieve(root);
        break;
    }
    case 5:
    {
        find(root);
        break;
    }
    case 6:
    {
        displayChild(root);
        break;
    }
    case 7:
    {
        displayDescendant(root);
        break;
    }
    case 8:
    {
        preOrder();
        break;
    }
    case 9:
    {
        inOrder();
        break;
    }
```

```
}  
case 10:  
{  
    postOrder();  
    break;  
}  
case 11:  
{  
    deleteSub(root);  
    break;  
}  
case 12:  
{  
    clear();  
    break;  
}  
case 13:  
{  
    characteristic();  
    break;  
}  
case 14:  
{  
    return 0;  
}  
default:  
{  
    cout << "Pilihan tidak tersedia!" << endl;  
    break;  
}  
}  
return 0;  
}
```

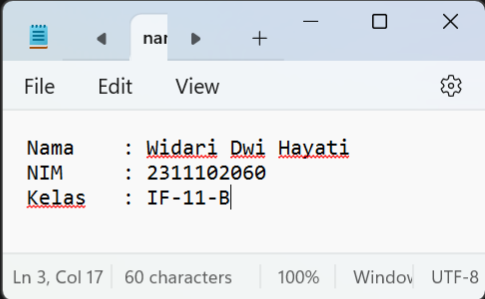
Screenshots Output

```
Masukkan data root: A

Node A berhasil dibuat sebagai root.

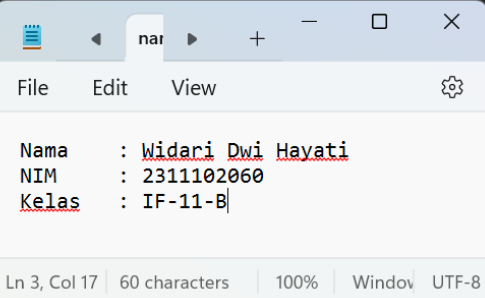
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 1
Masukkan data kiri: B

Node B berhasil ditambahkan ke child kiri A
```



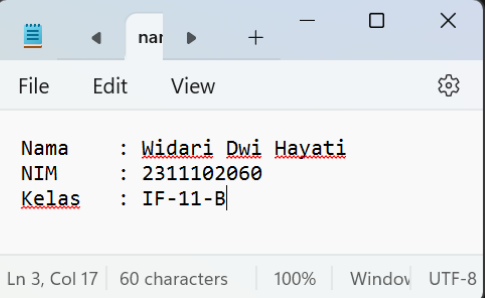
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 2
Masukkan data kanan: C

Node C berhasil ditambahkan ke child kanan A
```



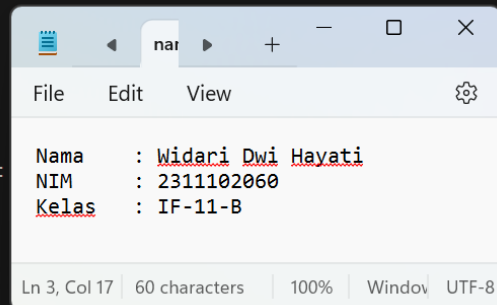
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 1
Masukkan data kiri: D

Node A sudah ada child kiri !
```



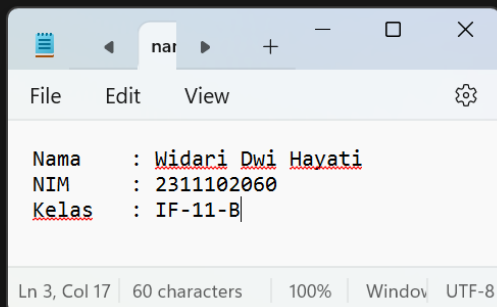

```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 2
Masukkan data kanan: E

Node A sudah ada child kanan !
```



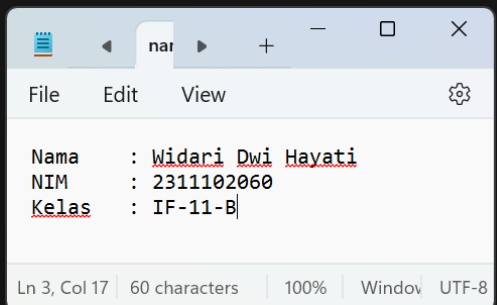
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 3
Masukkan data update: F

Node A berhasil diubah menjadi F
```



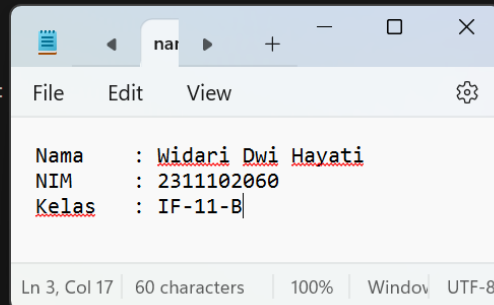
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 4

Data node : F
```

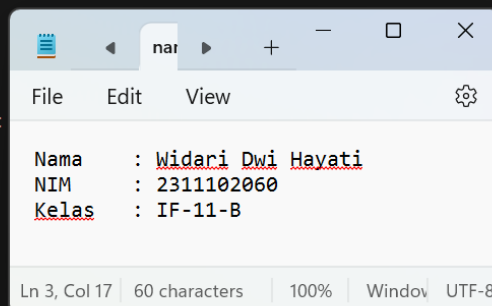


```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 5

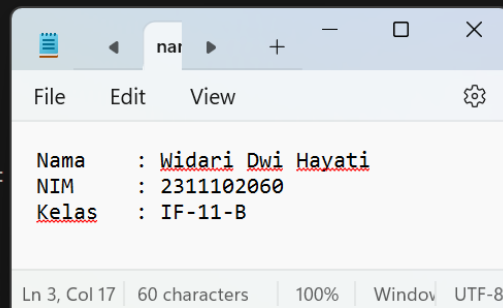
Data Node : F
Root : F
Parent : (tidak punya parent)
Sibling : (tidak punya sibling)
Child Kiri : B
Child Kanan : C
```



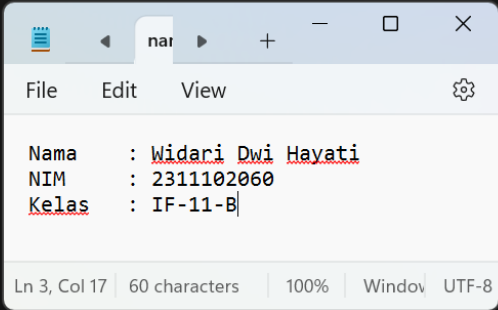
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 6
Child kiri: B
Child kanan: C
```



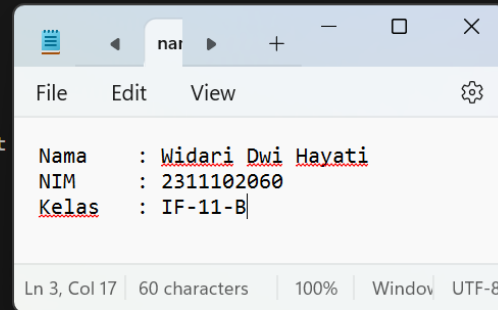
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 7
Descendant kiri: B
Descendant kanan: C
```



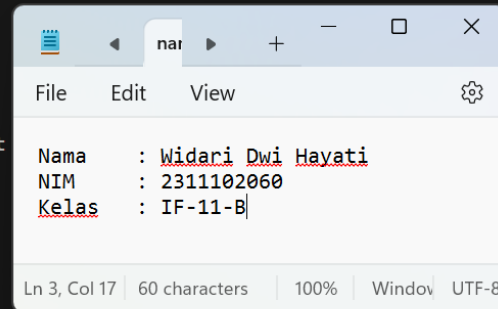
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 8
F, B, C,
```



```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 9
B, F, C,
```

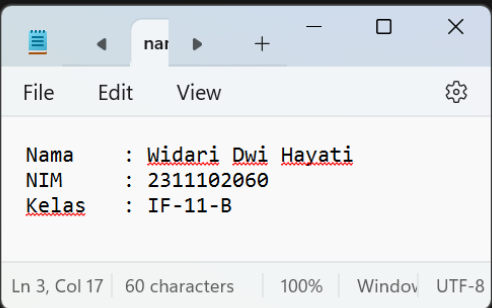


```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 10
B, C, F,
```



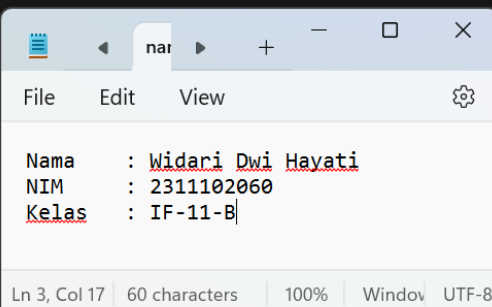
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 11

Node subtree F berhasil dihapus.
```



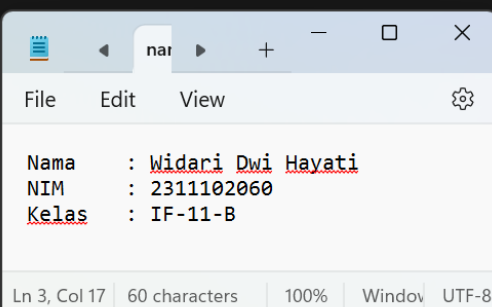
```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 13

Size Tree : 1
Height Tree : 1
Average Node of Tree : 1
```



```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 5

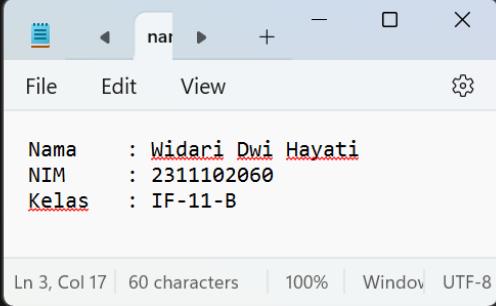
Data Node : F
Root : F
Parent : (tidak punya parent)
Sibling : (tidak punya sibling)
Child Kiri : (tidak punya Child kiri)
Child Kanan : (tidak punya Child kanan)
```



```
Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 12

Pohon berhasil dihapus.

Menu:
1. Tambah node kiri
2. Tambah node kanan
3. Update node
4. Retrieve node
5. Find node
6. Display child
7. Display descendant
8. PreOrder
9. InOrder
10. PostOrder
11. Delete subtree
12. Clear tree
13. Characteristic
14. Exit
Pilihan: 14
PS C:\Users\ADVAN>
```



Deskripsi:

Program ini mengimplementasikan struktur data pohon biner dan menyediakan berbagai operasi untuk memanipulasi dan menelusuri pohon.

Fitur berupa membuat pohon biner baru dengan node akar, menambahkan node baru ke kiri atau kanan node yang ada, memperbarui nilai node yang ada, mengambil nilai node, mencari node dalam pohon dan menampilkan propertinya (orang tua, saudara, node anak), menampilkan node anak dari node, menampilkan node keturunan dari node, melakukan traversal pre-order, in-order, dan post-order pohon, menghapus subtree yang berakar pada node tertentu, menghapus pohon secara keseluruhan, menampilkan karakteristik pohon (ukuran, tinggi, derajat node rata-rata).

Program ini menyediakan antarmuka menu-driven untuk melakukan operasi di atas. Pengguna dapat memilih dari opsi berupa tambah node kiri (Tambah node anak kiri), tambah node kanan (Tambah node anak kanan), perbarui node (Perbarui nilai node), ambil node (Ambil nilai node), cari node (Cari node dalam pohon dan tampilkan propertinya), tampilkan anak (Tampilkan node anak dari node), tampilkan keturunan (Tampilkan node keturunan dari node), PreOrder (Lakukan traversal pre-order pohon), InOrder (Lakukan traversal in-order pohon), PostOrder (Lakukan traversal post-order pohon), hapus subtree (Hapus subtree yang berakar pada node tertentu), hapus pohon (Hapus pohon secara keseluruhan), karakteristik (Tampilkan karakteristik pohon), keluar (Keluar dari program).

Program ini menggunakan struct Pohon untuk merepresentasikan node dalam pohon biner, dengan field untuk nilai node, node anak kiri dan kanan, dan node orang tua. Program ini menyediakan berbagai fungsi untuk melakukan operasi di atas, termasuk init untuk menginisialisasi pohon, buatNode untuk membuat node baru, insertLeft dan insertRight untuk menambahkan node baru, update untuk memperbarui nilai node, retrieve untuk mengambil nilai node, find untuk mencari node dalam pohon, dan sebagainya. Program ini juga menyediakan fungsi untuk melakukan traversal pohon dan menghapus subtree atau menghapus pohon secara keseluruhan.

D. Kesimpulan

Graf adalah struktur data yang digunakan untuk menggambarkan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Pohon biner adalah struktur data pohon yang setiap simpul dalam pohon diprasyartkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul.

Program ini menyediakan berbagai operasi untuk memanipulasi dan menelusuri pohon biner, seperti membuat pohon baru, menambahkan node baru, memperbarui nilai node, mengambil nilai node, mencari node dalam pohon, menampilkan node anak dan keturunan, melakukan traversal pre-order, in-order, dan post-order pohon, menghapus subtree, menghapus pohon secara keseluruhan, dan menampilkan karakteristik pohon.

E. Referensi (APA)

Stroustrup, B. (2021). A Tour of C++ (2nd ed.). Addison-Wesley Professional.

Lippman, S. L., Lajoie, J., & Moo, B. (2020). C++ Primer (6th ed.). Addison-Wesley Professional.

Malik, D. S. (2014). C++ Programming: From Problem Analysis to Program Design (7th ed.). Cengage Learning.