

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 9
GRAPH DAN TREE**



DISUSUN OLEH:

NAMA : SYARIEF RENDI ADITYA ANTONIUS

NIM : 2311102072

S1 IF-11-B

DOSEN:

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

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

A. DASAR TEORI

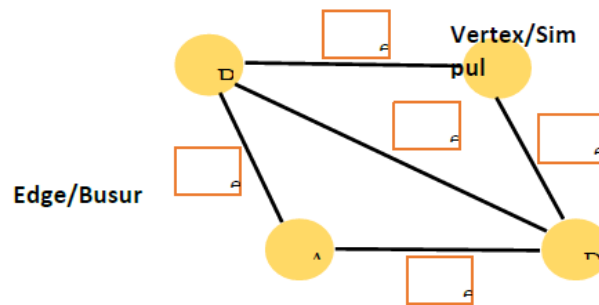
1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan 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 secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge.

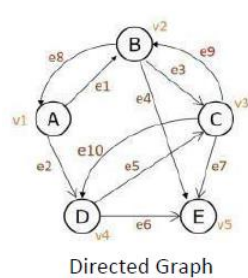
Dapat digambarkan:



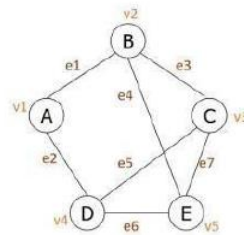
Gambar 1 Contoh Graph

Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

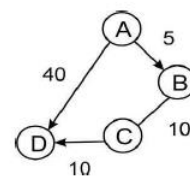
Jenis- jenis Graph



Directed Graph



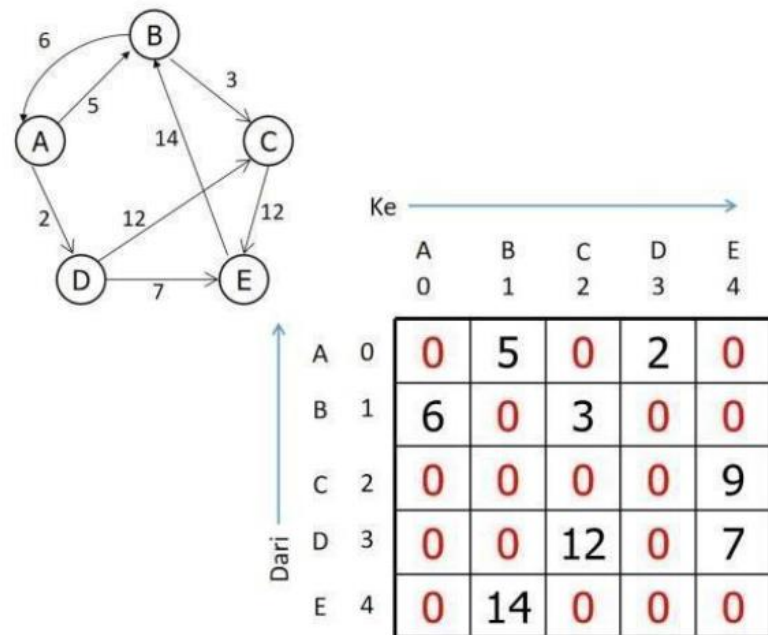
Undirected Graph



Weight Graph

- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.

- c. **Weight Graph** : Graph yang mempunyai nilai pada tiap edgenya.
Representasi Graph dengan Matriks



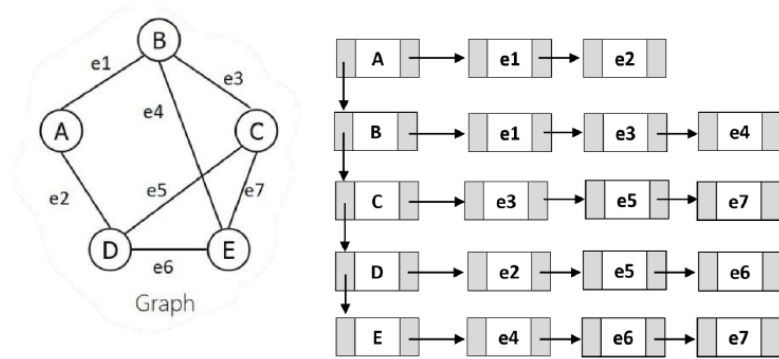
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

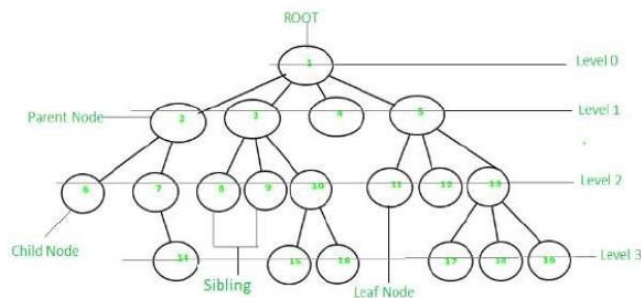
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. 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

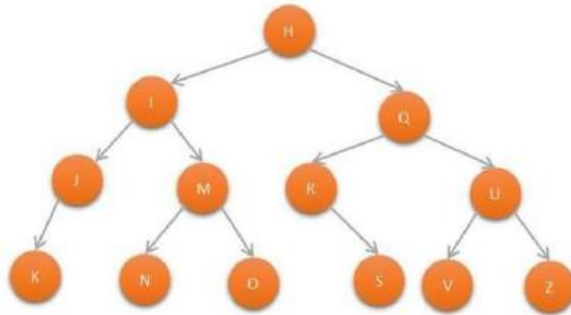
Dalam ilmu komputer, pohon/tree adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, dimana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



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

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.

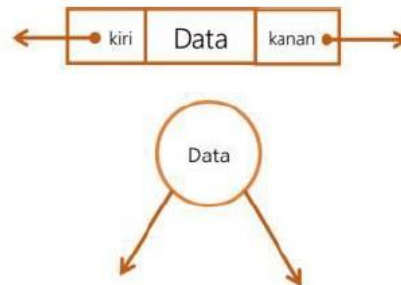


Gambar 1 Struktur Data Binary Tree

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;
  
```



Gambar 2 Ilustrasi Simpul 2 Pointer

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.

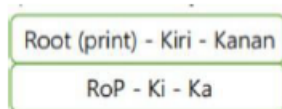
- f. **Update:** digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- g. **Retrive:** digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- h. **Delete Sub:** digunakan untuk menghapus 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 lenght-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

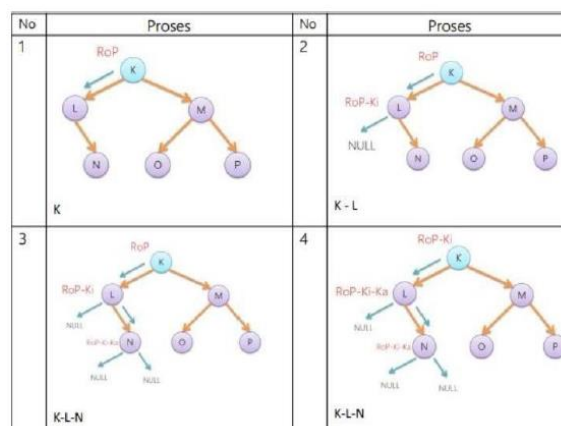
Penelusuran secara pre-order memiliki alur:

- 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:



Alur pre-order



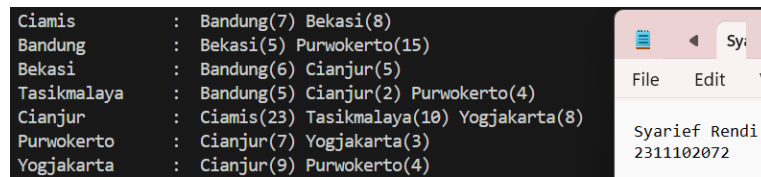
B. Guided

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;
} //2311102072
```


SCREENSHOT 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)
```

DESKRIPSI PROGRAM

Program di atas menggunakan array dua dimensi untuk merepresentasikan sebuah graf dengan tujuh simpul yang terhubung melalui busur dengan bobot tertentu. Program ini bertujuan untuk menampilkan representasi graf tersebut dalam bentuk teks. Setiap simpul direpresentasikan dengan sebuah string yang menunjukkan nama kota, dan setiap busur memiliki bobot yang menunjukkan jarak atau hubungan antara dua simpul.

GUIDED 2 : Program Tree

SOURCE CODE

```
#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();
} //2311102072

```

SCREENSHOT 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

Data node : C

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

PreOrder :
A, B, D, E, G, I, J, H, C, F,

InOrder :
D, B, I, G, J, E, H, A, F, C,

PostOrder :
D, I, J, G, H, E, B, F, C, A,

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

Node subtree E berhasil dihapus.

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

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

DESKRIPSI PROGRAM

Program diatas menggunakan struktur data pohon biner untuk menyimpan dan memanipulasi data dalam bentuk node. Program tersebut menyediakan berbagai fungsi untuk menginisialisasi pohon, menambahkan node ke pohon, mengubah data node, menampilkan isi data node, mencari data dalam pohon, melakukan penelusuran pohon dengan berbagai metode (preorder, inorder, postorder), menghapus node dan subpohon, serta menghitung ukuran dan tinggi pohon. Program juga menampilkan karakteristik dari pohon biner yang dibuat.

C. UNGUIDED

*Cantumkan NIM pada salah satu variabel di dalam program.

Contoh : int nama_22102003;

UNGUIDED 1 :

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

SOURCE CODE

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

using namespace std;

int main()
{
    int simpul_2311102072;
    cout << "Silakan masukkan jumlah simpul : ";
    cin >> simpul_2311102072;

    string simpul[simpul_2311102072];
    int bobot[simpul_2311102072][simpul_2311102072];
    cout << "Silakan masukkan nama simpul\n";
    for (int i = 0; i < simpul_2311102072; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    cout << "Silakan masukkan bobot antar simpul\n";
    for (int i = 0; i < simpul_2311102072; i++) {
        for (int j = 0; j < simpul_2311102072; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " : ";
            cin >> bobot[i][j];
        }
    }

    cout << endl << setw(10) << " ";
    for (int i = 0; i < simpul_2311102072; i++) {
        cout << setw(10) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < simpul_2311102072; i++) {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < simpul_2311102072; j++) {
```

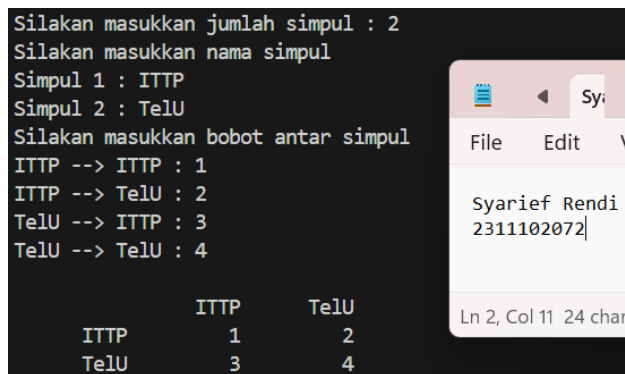
```

        cout << setw(10) << bobot[i][j];
    }
    cout << endl;
}

return 0;
} //2311102072

```

SCREENSHOT OUTPUT



```

Silakan masukkan jumlah simpul : 2
Silakan masukkan nama simpul
Simpul 1 : ITTP
Simpul 2 : TelU
Silakan masukkan bobot antar simpul
ITTP --> ITTP : 1
ITTP --> TelU : 2
TelU --> ITTP : 3
TelU --> TelU : 4

```

	ITTP	TelU
ITTP	1	2
TelU	3	4

DESKRIPSI PROGRAM

Program di atas membuat dan menampilkan graf berbobot. Program tersebut meminta pengguna untuk memasukkan jumlah simpul dalam graf, nama setiap simpul, dan bobot antar simpul. Setelah data dimasukkan, program akan menampilkan matriks bobot yang menunjukkan bobot antar setiap pasangan simpul. Setiap elemen dalam matriks merepresentasikan bobot dari simpul baris ke simpul kolom. Program ini menggunakan array dua dimensi untuk menyimpan bobot antar simpul dan array satu dimensi untuk menyimpan nama simpul, serta menggunakan manipulasi keluaran seperti setw untuk merapikan tampilan matriks.

UNGUIDED 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 diinputkan!

SOURCE CODE

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

// Deklarasi Pohon
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;

// Inisialisasi
void init() {
    root = NULL;
}

// Cek Node
int isEmpty() {
    return (root == NULL);
}

// Buat Node Baru
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 menjadi"
root." << endl;
    } else {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Cari Node Berdasarkan Data
Pohon* findNode(Pohon* node, char data) {
    if (node == NULL) return NULL;
    if (node->data == data) return node;
```

```

        Pohon* foundNode = findNode(node->left, data);
        if (foundNode == NULL) foundNode = findNode(node->right,
data);
        return foundNode;
    }

// Tambah Kiri
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) {
            // 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 (isEmpty()) {
        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()) {
        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;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon* node) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

```

```

}

// inOrder
void inOrder(Pohon* node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

// postOrder
void postOrder(Pohon* node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Hapus Node Tree
void deleteTree(Pohon* node) {
    if (node != NULL) {
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root) {
            delete root;
            root = NULL;
        } else {
            delete node;
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon* node) {
    if (node != NULL) {
        deleteTree(node->left);
        deleteTree(node->right);
        node->left = NULL;
        node->right = NULL;
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree

```

```

void clear() {
    deleteTree(root);
    cout << "\n Pohon berhasil dihapus." << endl;
}

// Cek Size Tree
int size(Pohon* node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

// Cek Height Level Tree
int height(Pohon* node) {
    if (node == NULL) {
        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return max(heightKiri, heightKanan) + 1;
    }
}

// Karakteristik Tree
void characteristic() {
    cout << "\n Size Tree : " << size(root) << endl;
    cout << " Height Tree : " << height(root) << endl;
    cout << " Average Node of Tree : " << (height(root) == 0 ?
0 : size(root) / height(root)) << endl;
}

// Menampilkan Child Node
void showChildren(Pohon* node) {
    if (node) {
        if (node->left)
            cout << " Child Kiri: " << node->left->data <<
endl;
        else
            cout << " Child Kiri: (tidak punya Child kiri)" <<
endl;
        if (node->right)
            cout << " Child Kanan: " << node->right->data <<
endl;
        else

```

```

        cout << " Child Kanan: (tidak punya Child kanan)"
<< endl;
    }
}

// Menampilkan Descendants Node
void showDescendants(Pohon* node) {
    if (node) {
        cout << " Descendants of Node " << node->data << ": ";
        preOrder(node);
        cout << endl;
    }
}

void menu() {
    int pilihan;
    char data;
    char parentData_2311102072;
    Pohon* temp = nullptr;
    do {
        cout << "\nMENU:\n";
        cout << "1. Buat Node Root\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Update Node\n";
        cout << "5. Retrieve Node\n";
        cout << "6. Find Node\n";
        cout << "7. Tampilkan PreOrder\n";
        cout << "8. Tampilkan InOrder\n";
        cout << "9. Tampilkan PostOrder\n";
        cout << "10. Tampilkan Characteristic\n";
        cout << "11. Hapus SubTree\n";
        cout << "12. Hapus Tree\n";
        cout << "13. Tampilkan Children\n";
        cout << "14. Tampilkan Descendants\n";
        cout << "0. Keluar\n";
        cout << "Masukkan pilihan: ";
        cin >> pilihan;
        switch (pilihan) {
            case 1:
                if (isEmpty()) {
                    cout << "Masukkan data root: ";
                    cin >> data;
                    buatNode(data);
                } else {
                    cout << "\n Root sudah ada!" << endl;
                }
            default:
                break;
        }
    } while (pilihan != 0);
}

```

```

        break;
    case 2:
        if (!isEmpty()) {
            cout << "Masukkan data node kiri: ";
            cin >> data;
            cout << "Masukkan data parent: ";
            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            insertLeft(data, temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 3:
        if (!isEmpty()) {
            cout << "Masukkan data node kanan: ";
            cin >> data;
            cout << "Masukkan data parent: ";
            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            insertRight(data, temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 4:
        if (!isEmpty()) {
            cout << "Masukkan data baru: ";
            cin >> data;
            cout << "Masukkan data node yang akan diupdate:
";

            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            update(data, temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 5:
        if (!isEmpty()) {
            cout << "Masukkan data node yang akan dilihat:
";

            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);

```

```

        retrieve(temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" <<
endl;

    }
    break;
case 6:
    if (!isEmpty()) {
        cout << "Masukkan data node yang akan dicari:
";

        cin >> parentData_2311102072;
        temp = findNode(root, parentData_2311102072);
        find(temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" <<
endl;

    }
    break;
case 7:
    if (!isEmpty()) {
        cout << "\n PreOrder :" << endl;
        preOrder(root);
        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" <<
endl;

    }
    break;
case 8:
    if (!isEmpty()) {
        cout << "\n InOrder :" << endl;
        inOrder(root);
        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" <<
endl;

    }
    break;
case 9:
    if (!isEmpty()) {
        cout << "\n PostOrder :" << endl;
        postOrder(root);
        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" <<
endl;

    }
}

```

```

        break;
    case 10:
        if (!isEmpty()) {
            characteristic();
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 11:
        if (!isEmpty()) {
            cout << "Masukkan data node yang subtreenya
akan dihapus: ";
            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            deleteSub(temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 12:
        clear();
        break;
    case 13:
        if (!isEmpty()) {
            cout << "Masukkan data node yang akan
ditampilkan childnya: ";
            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            showChildren(temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
        break;
    case 14:
        if (!isEmpty()) {
            cout << "Masukkan data node yang akan
ditampilkan descendantnya: ";
            cin >> parentData_2311102072;
            temp = findNode(root, parentData_2311102072);
            showDescendants(temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" <<
endl;
        }
    }
}

```



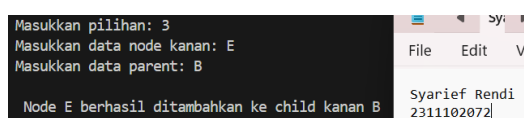
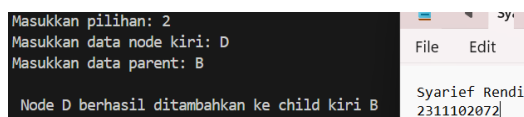
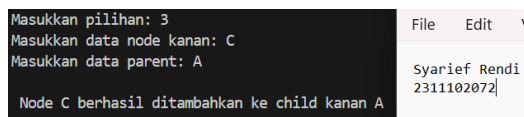
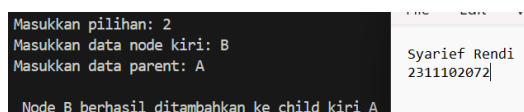
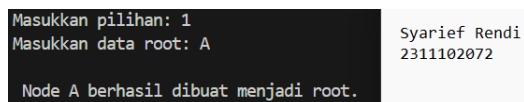
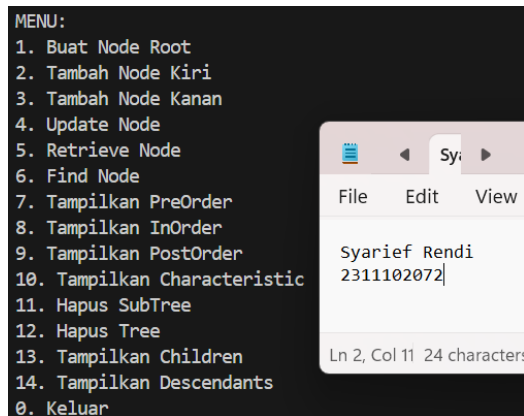
```

        break;
    case 0:
        cout << "\n Keluar dari program..." << endl;
        break;
    default:
        cout << "\n Pilihan tidak valid!" << endl;
    }
} while (pilihan != 0);
}

int main() {
    init();
    menu();
    return 0;
} //2311102072

```

SCREENSHOT OUTPUT



Masukkan pilihan: 2 Masukkan data node kiri: F Masukkan data parent: C Node F berhasil ditambahkan ke child kiri C	File Edit Vie Syarief Rendi 2311102072
Masukkan pilihan: 2 Masukkan data node kiri: G Masukkan data parent: E Node G berhasil ditambahkan ke child kiri E	File Edit Vie Syarief Rendi 2311102072
Masukkan pilihan: 3 Masukkan data node kanan: H Masukkan data parent: E Node H berhasil ditambahkan ke child kanan E	File Edit V Syarief Rendi 2311102072
Masukkan pilihan: 2 Masukkan data node kiri: I Masukkan data parent: G Node I berhasil ditambahkan ke child kiri G	File Edit Syarief Rendi 2311102072
Masukkan pilihan: 3 Masukkan data node kanan: J Masukkan data parent: G Node J berhasil ditambahkan ke child kanan G	File Edit V Syarief Rendi 2311102072
Masukkan pilihan: 4 Masukkan data baru: Z Masukkan data node yang akan diupdate: C Node C berhasil diubah menjadi Z	File Edit V Syarief Rendi 2311102072
Masukkan pilihan: 4 Masukkan data baru: C Masukkan data node yang akan diupdate: Z Node Z berhasil diubah menjadi C	File Edit Syarief Rendi 2311102072
Masukkan pilihan: 5 Masukkan data node yang akan dilihat: C Data node : C	Syarief Rendi 2311102072
Masukkan pilihan: 6 Masukkan data node yang akan dicari: C Data Node : C Root : A Parent : A Sibling : B Child Kiri : F Child Kanan : (tidak punya Child kanan)	Syarief Rendi 2311102072 Ln 2, Col 11 24 char
Masukkan pilihan: 7 PreOrder : A, B, D, E, G, I, J, H, C, F,	Syarief Rendi 2311102072
Masukkan pilihan: 8 InOrder : D, B, I, G, J, E, H, A, F, C,	Syarief Rendi 2311102072
Masukkan pilihan: 9 PostOrder : D, I, J, G, H, E, B, F, C, A,	Syarief Rendi 2311102072

Masukkan pilihan: 10	Syarief Rendi 2311102072
Size Tree : 10 Height Tree : 5 Average Node of Tree : 2	
Masukkan pilihan: 13 Masukkan data node yang akan ditampilkan childnya: A Child Kiri: B Child Kanan: C	Syarief Rendi 2311102072
Masukkan pilihan: 14 Masukkan data node yang akan ditampilkan descendantnya: A Descendants of Node A: A, B, D, E, G, I, J, H, C, F,	Syarief Rendi 2311102072
Masukkan pilihan: 11 Masukkan data node yang subtreenya akan dihapus: E Node subtree E berhasil dihapus.	Syarief Rendi 2311102072
Masukkan pilihan: 12 Pohon berhasil dihapus.	Syarief Rendi 2311102072
Masukkan pilihan: 0 Keluar dari program...	Syarief Rendi 2311102072

DESKRIPSI PROGRAM

Program di atas memberi opsi kepada user untuk melakukan berbagai operasi pada pohon biner melalui menu. Pengguna dapat membuat node root, menambahkan node anak kiri dan kanan, memperbarui data node, melihat dan mencari node, serta menampilkan traversal pre-order, in-order, dan post-order. Selain itu, program ini juga menyediakan fungsi untuk menampilkan karakteristik pohon seperti ukuran dan tinggi, menghapus subtree atau seluruh pohon, serta menampilkan anak dan keturunan dari suatu node tertentu. Program ini dimulai dengan inisialisasi pohon dan diakhiri dengan menjalankan menu utama hingga pengguna memilih untuk keluar.

D. Kesimpulan

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Sedangkan pohon atau tree adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini 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. Struktur data tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi.

E. Referensi

- [1] Asisten Praktikum, "Modul Algoritma GRAPH DAN TREE", 2024.
- [2] Ramdannur. 2020. "Data Structure : Mengenal Graph & Tree". Diakses pada 10 Juni 2024, dari <https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>
- [3] Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.
- [4] Budi Raharjo. 2015. *Pemrograman C++*. Bandung: Informatika.