LAPORAN PRAKTIKUM

MODUL IX

GRAPH DAN TREE

Disusun Oleh:

Fahrur Rizqi

2311102059

Dosen

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

- Mahasiswa diharapkan mampu memahami Graph and Tree
- Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II

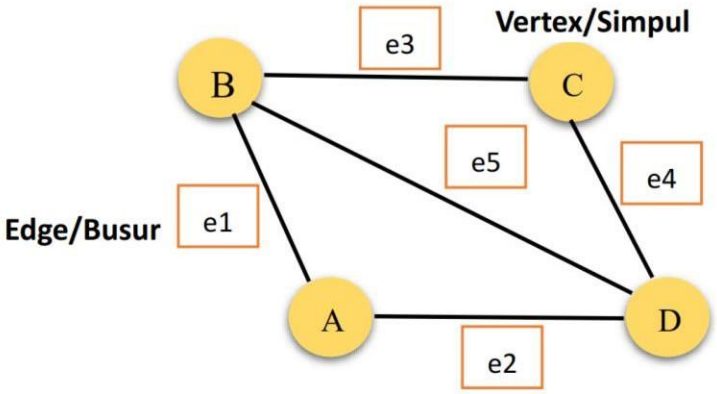
DASAR TEORI

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 edge 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 node sebagai titik atau egde dan digambarkan seperti berikut:

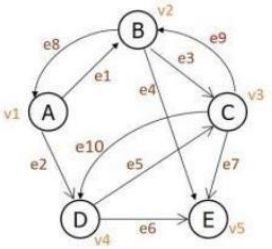


Graph juga dapat digunakan dalam berbagai bidang lainnya, seperti analisis jaringan untuk mempelajari pola koneksi dalam sistem kompleks, analisis peringkat dalam mesin pencari untuk menentukan relevansi dan otoritas halaman web. Dengan kemampuannya yang serbaguna, Graph menjadi alat yang sangat berharga dalam memahami dan menganalisis hubungan serta pola yang ada dalam berbagai konteks.

a. Jenis-jenis Graph

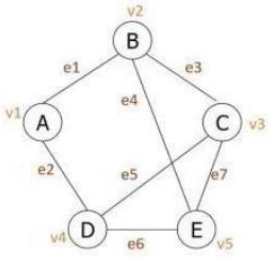
Graph memiliki berbagai jenis yang umumnya sering digunakan, antara lain:

1) Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.



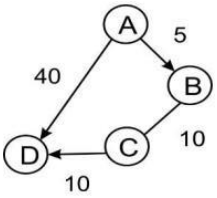
Directed Graph

2) Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur t idak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.



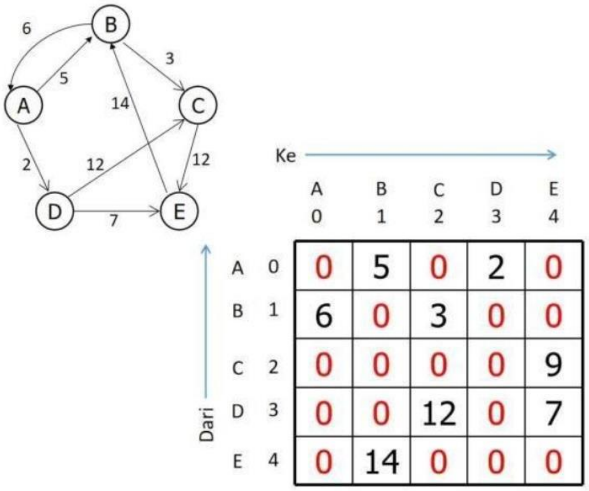
Undirected Graph

3) Weigth Graph : Graph yang mempunyai nilai pada tiap edgenya.



Weight Graph

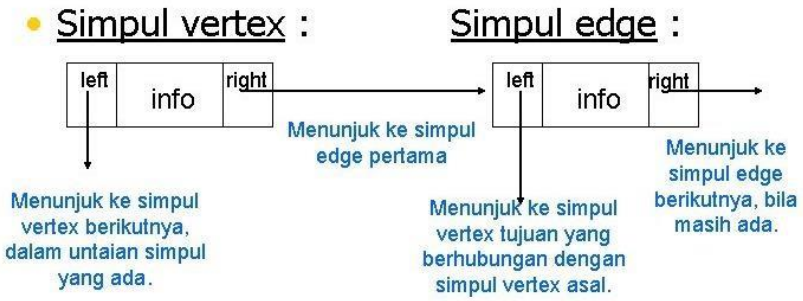
b. Representasi Graph dengan Matriks



Gambar Representasi Graph dengan Matriks

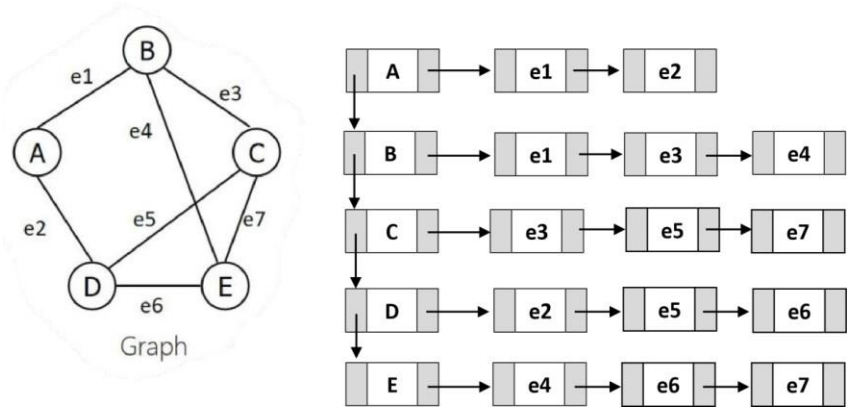
Representasi Graph dengan matriks, dikenal sebagai Matriks Adjasi, menggunakan matriks dua dimensi. Representasi Graph menggunakan matriks memiliki kelebihan akses efisien, penghitungan derajat simpul yang cepat, dan kesederhanaan. Namun, kekurangannya adalah penggunaan ruang yang besar, batasan skalabilitas, dan ketidakefisienan untuk Graph yang jarang terhubung. Representasi ini cocok untuk Graph kecil dengan hubungan yang padat, tetapi mungkin tidak efisien untuk Graph besar atau jarang terhubung.

c. Representasi Graph dengan Linked List



Dalam membuat representasi Graph dalam bentuk linked list, penting untuk memperhatikan perbedaan antara simpul vertex dan simpul edge. Simpul vertex merujuk pada simpul atau vertex dalam Graph, sedangkan simpul edge merujuk pada busur atau hubungan antar simbol. Meskipun struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, umumnya disamakan. Akan tetapi, perbedaan

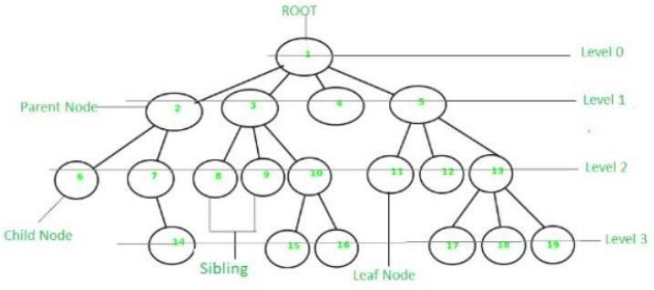
antara simpul vertex dan simpul edge terletak pada asumsi dan fungsi yang melekat pada masing-masing simpul.



Gambar Representasi Graph dengan Linked List

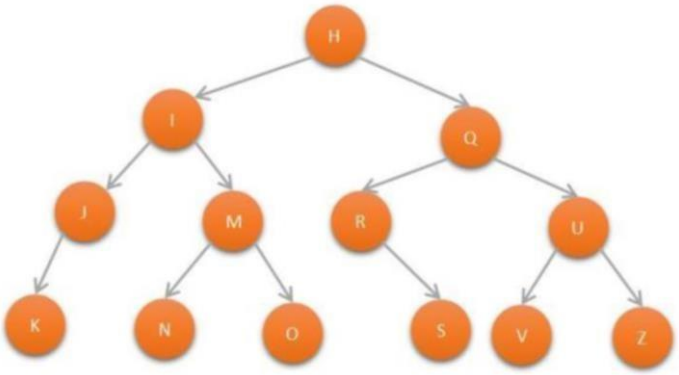
2. Tree atau Pohon

Dalam ilmu komputer, pohon merupakan struktur data yang umum dan kuat yang menyerupai pohon dalam kehidupan nyata. Struktur pohon terdiri dari kumpulan node yang saling terhubung, di mana setiap node memiliki paling banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data pohon digunakan untuk menyimpan data secara hierarkis, seperti pohon keluarga, skema pertandingan, dan struktur organisasi. Istilah-istilah yang terkait dengan struktur data pohon meliputi:



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

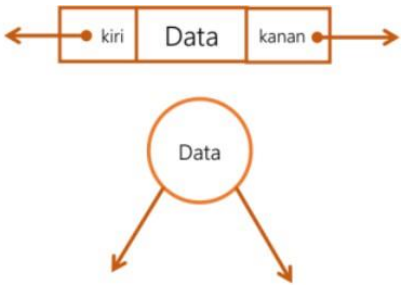
Binary Tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.



Gambar Struktur Data Binary Tree

Untuk 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 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

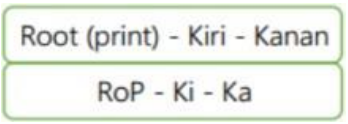
Penelusuran secara pre-order memiliki alur:

a. Cetak data pada simpul root

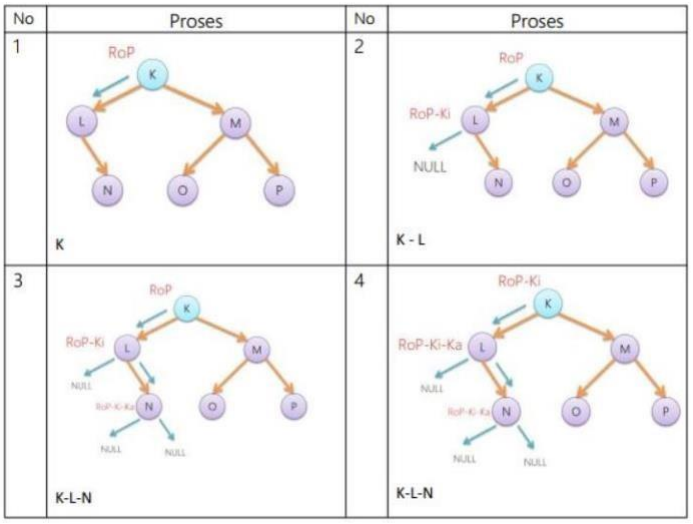
b. Secara rekursif mencetak seluruh data pada subpohon kiri

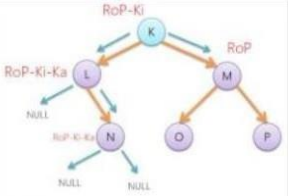
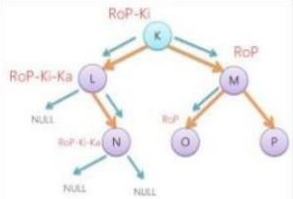
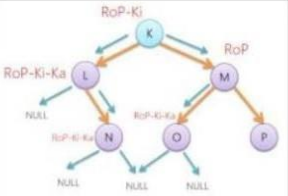
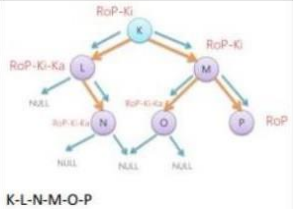
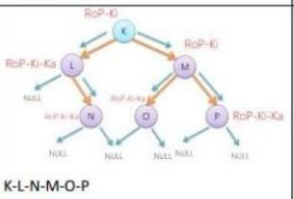
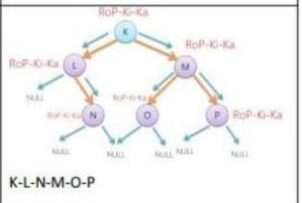
c. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:



Alur Pre-Order



5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

2. In-Order

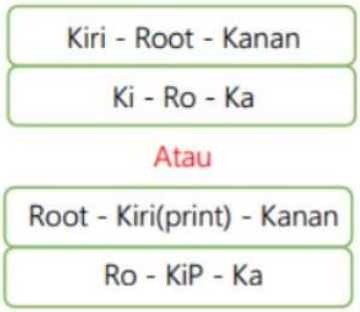
Penelusuran secara in-order memiliki alur:

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:

BAB III

GUIDED

Guided 1

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;
}
```

Output

```
PS C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\Laprak Struktur Data dan Algoritma\MODUL 9> cd
cuments\File Fahrur\SEMESTER 2\Laprak Struktur Data dan Algoritma\MODUL 9\" ; if ($?) { g++ guid
 } ; if ($?) { .\guided1 }
 Ciamis          :  Bandung(7) Bekasi(8)
 Bandung         :  Bekasi(5) Purwokerto(15)
 Bekasi          :  Bandung(6) Cianjur(5)
 tasikmalaya     :  Bandung(5) Cianjur(2) Purwokerto(4)
 Cianjur         :  Ciamis(23) tasikmalaya(10) Yogyakarta(8)
 Purwokerto      :  Cianjur(7) Yogyakarta(3)
 Yogyakarta      :  Cianjur(9) Purwokerto(4)
PS C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\Laprak Struktur Data dan Algoritma\MODUL 9>
```

Deskripsi program

Program tersebut adalah implementasi representasi graph menggunakan matriks ketetanggaan yang terdiri dari 7 simpul.

Guided 2

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();
}
```

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
PS C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\
```

Deskripsi Program

Program tersebut adalah implementasi dari pohon biner, program ini memungkinkan pembuatan, pemodifikasi, dan penelusuran pohon biner.

UNGUIDED

1. **Unguided 1**
   Source code

```
#include <iostream>
#include <vector>
#include <map>
#include <iomanip>
using namespace std;

typedef map<string, int> DistanceMap;
typedef map<string, DistanceMap> Graph;

void addEdge(Graph &graph, const string &source, const string
&destination, int distance)
{
    graph[source][destination] = distance;
}

void printGraph(const Graph &graph)
{
    vector<string> nodes;
    for (const auto &pair : graph)
    {
        nodes.push_back(pair.first);
    }

    cout << setw(10) << " ";
    for (const auto &node : nodes)
    {
        cout << setw(10) << node;
    }
    cout << endl;

    for (const auto &source : nodes)
    {
        cout << setw(10) << source;
        for (const auto &destination : nodes)
        {
            cout << setw(10) << graph.at(source).at(destination);
        }
        cout << endl;
    }
}

int main()
{
    int numNodes;
    cout << "Silahkan masukkan jumlah simpul: ";
```

```
    cin >> numNodes;
    vector<string> nodes(numNodes);
    cout << "Silahkan masukkan nama simpul:\n";
    for (int i = 0; i < numNodes; ++i)
    {
        cout << "Simpul " << i + 1 << ": ";
        cin >> nodes[i];
    }

    Graph graph;
    cout << "Silahkan masukkan bobot antar simpul\n";
    for (int i = 0; i < numNodes; ++i)
    {
        for (int j = 0; j < numNodes; ++j)
        {
            string source = nodes[i];
            string destination = nodes[j];
            int distance;
            cout << source << " => " << destination << " = ";
            cin >> distance;
            addEdge(graph, source, destination, distance);
        }
    }

    printGraph(graph);
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\
cuments\File Fahrur\SEMESTER 2\Laprak Struktur Dat
ded1 } ; if ($?) { .\unguided1 }
Silahkan masukkan jumlah simpul: 2
Silahkan masukkan nama simpul:
Simpul 1: Bali
Simpul 2: Palu
Silahkan 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 C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\
```

Deskripsi Program

Program tersebut adalah implementasi sederhana dari struktur data graf, program ini memungkinkan pengguna untuk memasukkan jumlah simpul, nama simpul, dan bobot antar simpul dalam graf. Kemudian, graf yang dibuat akan dicetak ke layar.

2. **Unguided 2**

Source code

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

// PROGRAM BINARY TREE
struct Pohon
{
    char data;
    Pohon *left, *right, *parent; // pointer
};

// pointer global
Pohon *root;

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

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

Pohon *newPohon(char data)
{
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data)
{
    if (isEmpty())
    {
        root = newPohon(data);
```

```
        cout << "\nAlhamdulillah, Node " << data << " berhasil dibuat
menjadi root." << endl;
    }
    else
    {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nPliss! Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\nUyy Node " << node->data << " sudah ada child
kiri!" << endl;
            return NULL;
        }
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nMantapp!! Node " << data << " berhasil
ditambahkan ke child kiri " << node->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nOyyy!! buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\nUyy Node " << node->data << " sudah ada child
kanan!" << endl;
            return NULL;
        }
```

```
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nAlhamdulillah!! Node " << data << " berhasil
ditambahkan ke child kanan " << node->data << endl;
            return baru;
        }
    }
}

void update(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nOyy!! buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\nAlhamdulillah!! Node " << temp << " berhasil
diubah menjadi " << data << endl;
        }
    }
}

void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nOyy!! Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

void find(Pohon *node)
{
```

```
    if (isEmpty())
    {
        cout << "\nOyy!! Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->data << endl;
            if (!node->parent)
                cout << "Parent : (tidak punya parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node &&
node->parent->right == node)
                cout << "Sibling : " << node->parent->left->data <<
endl;
            else if (node->parent != NULL && node->parent->right !=
node && node->parent->left == node)
                cout << "Sibling : " << node->parent->right->data <<
endl;
            else
                cout << "Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << "Child Kiri : (tidak punya Child kiri)" <<
endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child kanan)" <<
endl;
            else
                cout << "Child Kanan : " << node->right->data << endl;
        }
    }
}

void showChild(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nOyy!! Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
```

```
        }
        else
        {
            cout << "\nNode " << node->data << " Child: " << endl;
            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;
        }
    }
}

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

void showDescendants(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nOyy!! Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\nDescendants of node " << node->data << " : ";
            preOrder(node);
            cout << endl;
        }
    }
}
```

```
// Penelusuran (Traversal)
// 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)
    {
        if (node != root)
        {
            if (node->parent->left == node)
                node->parent->left = NULL;
            else if (node->parent->right == node)
                node->parent->right = NULL;
        }
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root)
        {
            delete root;
            root = NULL;
        }
        else
        {
            delete node;
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
```

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

// Hapus Tree
void clear()
{
    if (isEmpty())
    {
        cout << "\nOyy!! Buat tree terlebih dahulu!!" << endl;
    }
    else
    {
        deleteTree(root);
        cout << "\nAlhamdulillah!! Pohon berhasil dihapus." << endl;
    }
}

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

// Cek Height Level Tree
int height(Pohon *node)
{
    if (!node)
    {
        return 0;
    }
    else
    {
        int heightKiri = height(node->left);
```

```
        int heightKanan = height(node->right);
        if (heightKiri >= heightKanan)
        {
            return heightKiri + 1;
        }
        else
        {
            return heightKanan + 1;
        }
    }
}

// Karakteristik Tree
void characteristic()
{
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

int main()
{
    int choice;
    char fahrur_2311102059 , parentData, direction;
    Pohon *parentNode, *tempNode;

    do
    {
        cout << "\n[[[======Selamat Datang di Node Pohon=====]]]\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. Show Child\n";
        cout << "8. Show Descendants\n";
        cout << "9. PreOrder Traversal\n";
        cout << "10. InOrder Traversal\n";
        cout << "11. PostOrder Traversal\n";
        cout << "12. Show Characteristics\n";
        cout << "13. Clear Tree\n";
        cout << "14. Exit\n";
        cout << "Pilih menu: ";
        cin >> choice;
```

```
        switch (choice)
        {
        case 1:
            cout << "Masukkan data root: ";
            cin >> fahrur_2311102059;
            buatNode(fahrur_2311102059);
            break;
        case 2:
            cout << "Masukkan data parent: ";
            cin >> parentData;
            cout << "Masukkan data node kiri: ";
            cin >> fahrur_2311102059;
            parentNode = root;
            while (parentNode && parentNode->data != parentData)
            {
                if (parentNode->left && parentNode->left->data ==
parentData)
                    parentNode = parentNode->left;
                else if (parentNode->right && parentNode->right->data
== parentData)
                    parentNode = parentNode->right;
                else if (parentNode->left)
                    parentNode = parentNode->left;
                else if (parentNode->right)
                    parentNode = parentNode->right;
                else
                    parentNode = NULL;
            }
            if (parentNode)
            {
                insertLeft(fahrur_2311102059, parentNode);
            }
            else
            {
                cout << "Parent tidak ditemukan!" << endl;
            }
            break;
        case 3:
            cout << "Masukkan data parent: ";
            cin >> parentData;
            cout << "Masukkan data node kanan: ";
            cin >> fahrur_2311102059;
            parentNode = root;
            while (parentNode && parentNode->data != parentData)
            {
                if (parentNode->left && parentNode->left->data ==
parentData)
                    parentNode = parentNode->left;
                else if (parentNode->right && parentNode->right->data
== parentData)
                    parentNode = parentNode->right;
```

```
                else if (parentNode->left)
                    parentNode = parentNode->left;
                else if (parentNode->right)
                    parentNode = parentNode->right;
                else
                    parentNode = NULL;
            }
            if (parentNode)
            {
                insertRight(fahrur_2311102059, parentNode);
            }
            else
            {
                cout << "Parent tidak ditemukan!" << endl;
            }
            break;
        case 4:
            cout << "Masukkan data node yang ingin diupdate: ";
            cin >> parentData;
            cout << "Masukkan data baru: ";
            cin >>fahrur_2311102059;
            tempNode = root;
            while (tempNode && tempNode->data != parentData)
            {
                if (tempNode->left && tempNode->left->data ==
parentData)
                    tempNode = tempNode->left;
                else if (tempNode->right && tempNode->right->data ==
parentData)
                    tempNode = tempNode->right;
                else if (tempNode->left)
                    tempNode = tempNode->left;
                else if (tempNode->right)
                    tempNode = tempNode->right;
                else
                    tempNode = NULL;
            }
            if (tempNode)
            {
                update(fahrur_2311102059, tempNode);
            }
            else
            {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
        case 5:
            cout << "Masukkan data node yang ingin diretrieve: ";
            cin >> fahrur_2311102059;
            tempNode = root;
            while (tempNode && tempNode->data != fahrur_2311102059)
```

```
            {
                if (tempNode->left && tempNode->left->data ==
fahrur_2311102059)
                    tempNode = tempNode->left;
                else if (tempNode->right && tempNode->right->data ==
fahrur_2311102059)
                    tempNode = tempNode->right;
                else if (tempNode->left)
                    tempNode = tempNode->left;
                else if (tempNode->right)
                    tempNode = tempNode->right;
                else
                    tempNode = NULL;
            }
            if (tempNode)
            {
                retrieve(tempNode);
            }
            else
            {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
        case 6:
            cout << "Masukkan data node yang ingin dicari: ";
            cin >> fahrur_2311102059;
            tempNode = root;
            while (tempNode && tempNode->data != fahrur_2311102059)
            {
                if (tempNode->left && tempNode->left->data ==
fahrur_2311102059)
                    tempNode = tempNode->left;
                else if (tempNode->right && tempNode->right->data ==
fahrur_2311102059)
                    tempNode = tempNode->right;
                else if (tempNode->left)
                    tempNode = tempNode->left;
                else if (tempNode->right)
                    tempNode = tempNode->right;
                else
                    tempNode = NULL;
            }
            if (tempNode)
            {
                find(tempNode);
            }
            else
            {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
```

```
        case 7:
            cout << "Masukkan data node yang ingin ditampilkan child-
nya: ";
            cin >>fahrur_2311102059;
            tempNode = root;
            while (tempNode && tempNode->data != fahrur_2311102059)
            {
                if (tempNode->left && tempNode->left->data
==fahrur_2311102059)
                    tempNode = tempNode->left;
                else if (tempNode->right && tempNode->right->data
==fahrur_2311102059)
                    tempNode = tempNode->right;
                else if (tempNode->left)
                    tempNode = tempNode->left;
                else if (tempNode->right)
                    tempNode = tempNode->right;
                else
                    tempNode = NULL;
            }
            if (tempNode)
            {
                showChild(tempNode);
            }
            else
            {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
        case 8:
            cout << "Masukkan data node yang ingin ditampilkan
descendant-nya: ";
            cin >> fahrur_2311102059;
            tempNode = root;
            while (tempNode && tempNode->data != fahrur_2311102059)
            {
                if (tempNode->left && tempNode->left->data ==
fahrur_2311102059)
                    tempNode = tempNode->left;
                else if (tempNode->right && tempNode->right->data
==fahrur_2311102059)
                    tempNode = tempNode->right;
                else if (tempNode->left)
                    tempNode = tempNode->left;
                else if (tempNode->right)
                    tempNode = tempNode->right;
                else
                    tempNode = NULL;
            }
            if (tempNode)
            {
```

```
                showDescendants(tempNode);
            }
            else
            {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
        case 9:
            cout << "\nPreOrder :" << endl;
            preOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 10:
            cout << "InOrder :" << endl;
            inOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 11:
            cout << "PostOrder :" << endl;
            postOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 12:
            characteristic();
            break;
        case 13:
            clear();
            break;
        case 14:
            cout << "Bye byee!!" << endl;
            break;
        default:
            cout << "Pilihan tidak valid!" << endl;
        }
    } while (choice != 14);

    return 0;
}
```

Output

```
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 1
Masukkan data root: a

Alhamdulillah, Node a berhasil dibuat menjadi root.
```

```
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 2
Masukkan data parent: a
Masukkan data node kiri: c

Mantapp!! Node c berhasil ditambahkan ke child kiri a
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 3
Masukkan data parent: a
Masukkan data node kanan: s

Alhamdulillah!! Node s berhasil ditambahkan ke child kanan a
```

```
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 7
Masukkan data node yang ingin ditampilkan child-nya: a

Node a Child:
Child Kiri : c
Child Kanan : s

[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 8
Masukkan data node yang ingin ditampilkan descendant-nya: a

Descendants of node a :  a,  c,  s,
```

```
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 12

Size Tree : 3
Height Tree : 2
Average Node of Tree : 1
[[[======Selamat Datang di Node Pohon=====]]]
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Child
8. Show Descendants
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Show Characteristics
13. Clear Tree
14. Exit
Pilih menu: 14
Bye byee!!
PS C:\Users\ASUS\Documents\File Fahrur\SEMESTER 2\
```

Deskripsi Program

Program tersebut adalah implementasi sederhana dari struktur data pohon (tree), program ini memungkinkan pengguna untuk menambahkan simpul (node) ke dalam pohon dan mencetak pohon tersebut.

BAB IV
KESIMPULAN

Program-program di atas, yaitu Program Graf dan Program Pohon, adalah contoh implementasi struktur data graf dan pohon dalam bahasa pemrograman C++. Program Graf memungkinkan pengguna untuk membangun graf dengan memasukkan bobot antar simpul, sementara Program Pohon memungkinkan pengguna untuk membangun pohon dengan menambahkan simpul. Kedua program tersebut menggunakan konsep dasar seperti map, vector, dan fungsi. Program Pohon juga memiliki fitur interaktif yang memungkinkan pengguna untuk memilih menu dan melakukan operasi pada pohon yang dibuat. Dengan menggunakan program-program ini, pengguna dapat memahami konsep dan penggunaan struktur data graf dan pohon, serta menerapkannya dalam aplikasi yang lebih kompleks. Program-program tersebut memberikan pengguna kontrol penuh untuk membuat dan memanipulasi struktur data graf dan pohon sesuai kebutuhan mereka. Meskipun sederhana, program-program ini membantu pengguna mempelajari dan mengimplementasikan struktur data graf dan pohon dalam pemrograman.