LAPORAN PRAKTIKUM
STURKTUR DATA DAN ALGORITMA

MODUL IX
GRAPH & TREE

Disusun Oleh :
Muhammad Dani Ayubi
2311102003

Dosen Pengmapu:
Wahyu Andi Syahputra, S.pd.,M.Eng

LABORATORIUM MULTIMEDIA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO
2024

BAB I
TUJUAN PRAKTIKUM

1. Mahasiswa diharapkan mampu memahami graph dan tree.
2. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II
DASAR TEORI

Graf dan pohon (tree) adalah dua struktur data penting dalam ilmu komputer dan matematika diskret. Mereka digunakan untuk merepresentasikan hubungan antara objek atau entitas dalam bentuk yang terstruktur. Berikut adalah dasar teori mengenai
materi graf dan pohon:

1. Graf: Graf adalah himpunan objek yang disebut simpul (node) yang terhubung melalui sisi (edge). Graf digunakan untuk merepresentasikan hubungan antara entitas atau objek. Graf dapat digunakan dalam berbagai aplikasi, seperti jaringan komputer, jaringan sosial, algoritma penelusuran, pemodelan hubungan antar data, dan banyak lagi. Graf dapat digolongkan menjadi beberapa jenis, termasuk:
    - **Graf Berarah (Directed Graph):** Graf di mana setiap sisi memiliki arah. Artinya, hubungan antara simpul-simpul dalam graf memiliki orientasi tertentu.
    - **Graf Tidak Berarah (Undirected Graph):** Graf di mana setiap sisi tidak memiliki arah. Hubungan antara simpul-simpul dalam graf adalah saling mengarah.
    - **Graf Terhubung (Connected Graph):** Graf di mana ada setidaknya satu jalur yang menghubungkan setiap pasang simpul dalam graf.
    - **Graf Berbobot (Weighted Graph):** Graf di mana setiap sisi memiliki bobot atau nilai terkait yang menunjukkan biaya atau jarak antara simpul simpul yang terhubung.

    Beberapa konsep dan algoritma yang terkait dengan graf meliputi:

    a) Representasi Graf: Ada beberapa cara untuk merepresentasikan graf, termasuk representasi matriks ketetanggaan, representasi daftar ketetanggaan, dan representasi matriks penyebaran.
    b) Traversal Graf: Algoritma yang digunakan untuk mengunjungi semua simpul dalam graf, seperti Depth-First Search (DFS) dan Breadth-First Search (BFS).
    c) Menemukan Jarak Terpendek: Algoritma yang digunakan untuk menemukan jarak terpendek antara dua simpul dalam graf, seperti Algoritma Dijkstra dan Algoritma Bellman-Ford.

d) Minimum Spanning Tree: Pohon yang menghubungkan semua simpul dalam graf dengan total bobot yang minimal.
e) Algoritma Jaringan: Algoritma yang digunakan dalam jaringan, seperti Algoritma Max Flow-Min Cut dan Algoritma PageRank.

2. Tree : Pohon adalah struktur data hirarkis yang terdiri dari simpul-simpul terhubung satu sama lain. Pada pohon, terdapat simpul khusus yang disebut akar (root), simpul-simpul lain yang terhubung ke simpul di atasnya disebut anak (child), dan simpul yang berada di atas simpul anak disebut induk (parent).

   Konsep dan sifat yang terkait dengan pohon antara lain:
   a. Pohon Terurut (Ordered Tree): Pohon di mana anak-anak setiap simpul memiliki urutan tertentu.
   b. Pohon Biner (Binary Tree): Pohon di mana setiap simpul memiliki paling banyak dua anak, yaitu anak kiri dan anak kanan.

   Operasi yang dapat dilakukan pada pohon :
   1. Pembuatan (create): Operasi ini digunakan untuk membuat pohon baru. Pohon dapat dibuat dengan menginisialisasi simpul akar dan menghubungkannya dengan simpul-simpul anak sesuai dengan struktur hierarki yang diinginkan.
   2. Penghapusan (delete): Operasi ini digunakan untuk menghapus simpul tertentu beserta semua simpul anak yang terkait dengannya. Penghapusan simpul dapat dilakukan dengan menghapus referensi ke simpul tersebut dan membebaskan memori yang digunakan.
   3. Penelusuran (traversal): Operasi ini digunakan untuk mengunjungi setiap simpul dalam pohon.

   Terdapat beberapa metode penelusuran yang umum digunakan, antara lain:
   • Pre-order traversal: Mengunjungi simpul akar, kemudian secara rekursif melakukan penelusuran pre-order pada subpohon kiri, dan terakhir melakukan penelusuran pre-order pada subpohon kanan.
   • In-order traversal: Secara rekursif melakukan penelusuran in-order pada subpohon kiri, kemudian mengunjungi simpul akar, dan terakhir melakukan penelusuran in-order pada subpohon kanan.

   • Post-order traversal: Secara rekursif melakukan penelusuran post-order pada subpohon kiri, kemudian melakukan penelusuran post-order pada subpohon kanan, dan terakhir mengunjungi simpul akar.

4. Pencarian (search): Operasi ini digunakan untuk mencari simpul dengan nilai tertentu dalam pohon. Pencarian dapat dilakukan dengan melakukan penelusuran pohon secara rekursif atau menggunakan metode khusus seperti pohon biner pencarian.°
5. Penyisipan (insertion): Operasi ini digunakan untuk menyisipkan simpul baru ke dalam pohon. Penyisipan simpul dapat dilakukan dengan menentukan posisi yang sesuai sesuai dengan aturan struktur pohon, seperti menyisipkan simpul sebagai anak dari simpul tertentu atau sebagai simpul baru dengan menghubungkannya dengan simpul yang sudah ada.

BAB III

GUIDED

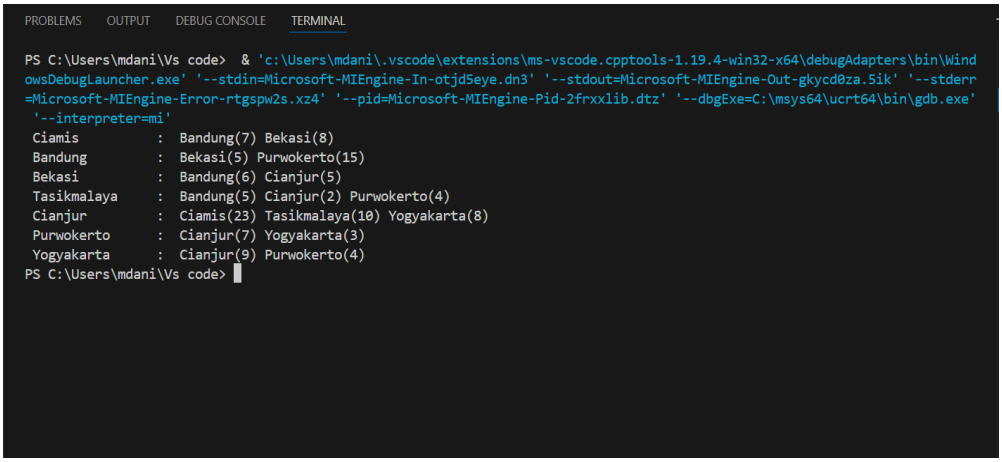
1. Guided 1

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;
}
```

Screenshoot program



Deskripsi program

Kode di atas adalah contoh implementasi graf dalam bentuk adjacency matrix menggunakan bahasa pemrograman C++. Terdapat array simpul yang berisi menggunakan bahasa pemrograman C++. Terdapat array simpul yang berisi simpul-simpul graf, serta array dua dimensi busur yang merepresentasikan matriks ketetanggaan (adjacency matrix). Fungsi tampilGraph() digunakan untuk menampilkan graf ke layar. Dalam fungsi main(), fungsi tampilGraph() dipanggil untuk menampilkan graf tersebut**.**

2. 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)
{
```

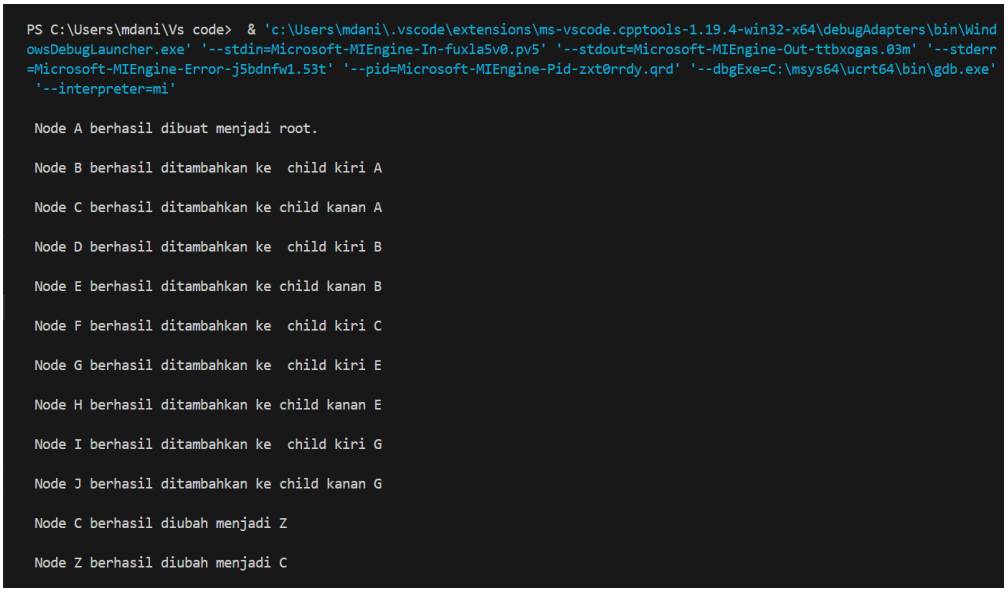
```
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}
// postOrder
void postOrder(Pohon *node = 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;
```

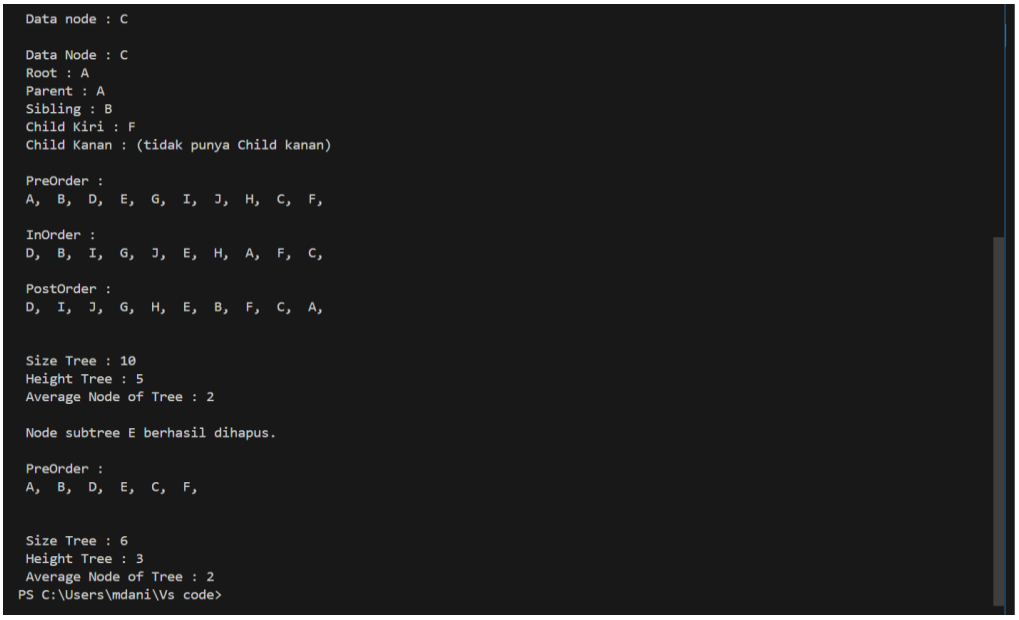
```
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!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();
}
```

Screenshoot program



```
PS C:\Users\mdani\Vs code>  & 'c:\Users\mdani\.vscode\extensions\ms-vscode.cpptools-1.19.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-fuxla5v0.pv5' '--stdout=Microsoft-MIEngine-Out-ttbxogas.03m' '--stderr=Microsoft-MIEngine-Error-j5bdnfw1.53t' '--pid=Microsoft-MIEngine-Pid-zxt0rrdy.qrd' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'

 Node A berhasil dibuat menjadi 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)

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\mdani\Vs code>
```

Deskripsi program

Kode di atas adalah contoh implementasi pohon biner menggunakan bahasa pemrograman C++. Program ini memungkinkan pembuatan, pemrosesan, dan penghapusan pohon biner. Fungsi-fungsi yang disediakan termasuk pembuatan simpul, penambahan simpul anak, pemutakhiran data, penelusuran pohon, penghapusan pohon, dan perhitungan karakteristik pohon seperti ukuran dan tinggi. Pada main(), pohon dibuat dan beberapa simpul ditambahkan. Informasi dan hasil penelusuran pohon ditampilkan, serta dilakukan penghapusan subtree. Karakteristik pohon dicetak sebelum dan setelah penghapusan.

LATIHAN KELAS - UNGUIDED

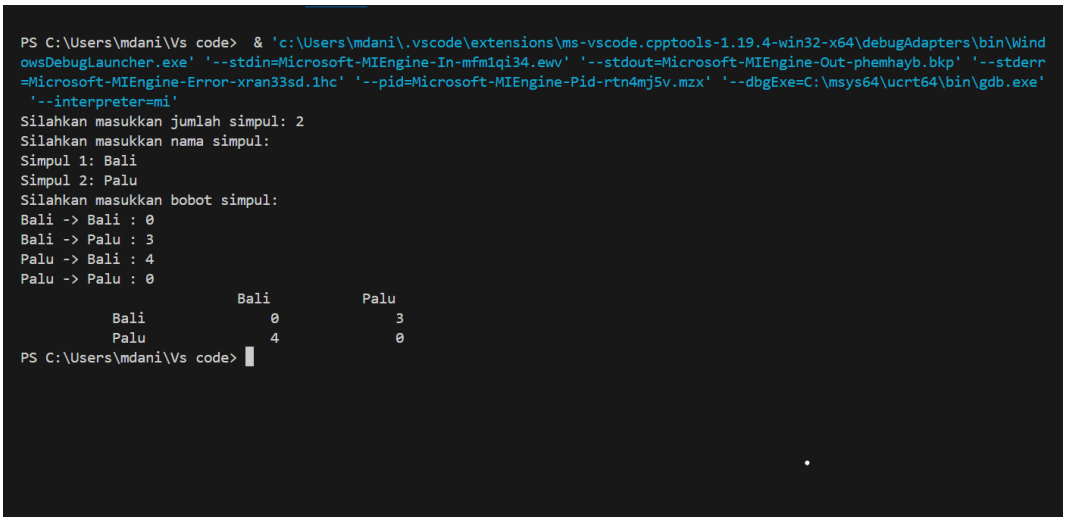
1. Unguided 1

Source code

```
#include <iostream>
#include <iomanip>
using namespace std;//Arif Pramudia Wardana_2211102149

const int MAX_SIMPUL = 10;

void tampilGraph(string simpul[], int busur[][MAX_SIMPUL], int
jumlahSimpul)
{
    cout << setw(15) << " ";
    for (int kolom = 0; kolom < jumlahSimpul; kolom++) {
        cout << setw(15) << simpul[kolom];
    }
    cout << endl;

    for (int baris = 0; baris < jumlahSimpul; baris++) {
        cout << setw(15) << simpul[baris] << " ";
        for (int kolom = 0; kolom < jumlahSimpul; kolom++) {
            cout << setw(15) << busur[baris][kolom];
        }
        cout << endl;
    }
}

void inputGraph(string simpul[], int busur[][MAX_SIMPUL], int&
jumlahSimpul)
{
    cout << "Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

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

    cout << "Silahkan masukkan bobot simpul:\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << " -> " << simpul[j] << " : ";
            cin >> busur[i][j];
        }
```

```
    }
}

int main()
{
    string simpul[MAX_SIMPUL];
    int busur[MAX_SIMPUL][MAX_SIMPUL];
    int jumlahSimpul;

    inputGraph(simpul, busur, jumlahSimpul);
    tampilGraph(simpul, busur, jumlahSimpul);

    return 0;
}
```

Screenshoot program



Deskripsi program

Kode di atas adalah program yang mengimplementasikan representasi grafik menggunakan struktur data map dan vector. Program ini memungkinkan pengguna untuk memasukkan jumlah simpul dan bobot antar simpul dalam grafik. Setelah memasukkan informasi tersebut, program akan mencetak grafik berisi bobot antar simpul. Program menggunakan struktur data map untuk menyimpan bobot antar simpul berdasarkan pasangan simpul yang terhubung.

2. Unguided 2

Source code

```
//Muhammad Dani Ayubi_2311102003
#include <iostream>
using namespace std;
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

void init()
{
    root = NULL;
}

int isEmpty()
{
    if (root == NULL)
        return 1;
    else
        return 0;
}

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;
    }
}

Pohon *insertLeft(char data, Pohon *node)
{
```

```
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->data << " sudah
memiliki child kiri!"
<< endl;
            return NULL;
        }
        else
        {
            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;
        }
    }
}

Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\n Node " << node->data << " sudah
memiliki child kanan!"
<< endl;
            return NULL;
        }
        else
```

```
        {
            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() == 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;
        }
    }
}

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;
        }
    }
}

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;
        }
    }
}

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);
        }
    }
}

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);
        }
    }
}

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 << ", ";
        }
    }
}

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;
            }
        }
    }
}

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;
    }
```

```
}

void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

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);
        }
    }
}

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;
            }
        }
    }
}

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

int main()
{
    int choice;
    char data;
    init();
    cout << "PROGRAM BINARY TREE" << endl;
    while (true)
    {
        cout << "\nMenu:" << endl;
        cout << "1. Buat Node Baru" << endl;
        cout << "2. Tambah Kiri" << endl;
        cout << "3. Tambah Kanan" << endl;
        cout << "4. Ubah Data Tree" << endl;
        cout << "5. Lihat Isi Data Tree" << endl;
        cout << "6. Cari Data Tree" << endl;
        cout << "7. Penelusuran Pre-order" << endl;
        cout << "8. Penelusuran In-order" << endl;
        cout << "9. Penelusuran Post-order" << endl;
        cout << "10. Hapus Node Tree" << endl;
        cout << "11. Hapus SubTree" << endl;
        cout << "12. Hapus Tree" << endl;
        cout << "13. Karakteristik Tree" << endl;
        cout << "14. Keluar" << endl;
        cout << "Pilihan: ";
```

```
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "\nMasukkan data untuk root: ";
            cin >> data;
            buatNode(data);
            break;
        case 2:
            cout << "\nMasukkan data baru: ";
            cin >> data;
            insertLeft(data, root);
            break;
        case 3:
            cout << "\nMasukkan data baru: ";
            cin >> data;
            insertRight(data, root);
            break;
        case 4:
            cout << "\nMasukkan data yang ingin diubah: ";
            cin >> data;
            update(data, root);
            break;
        case 5:
            retrieve(root);
            break;
        case 6:
            find(root);
            break;
        case 7:
            cout << "\nPenelusuran Pre-order: ";
            preOrder(root);
            cout << endl;
            break;
        case 8:
            cout << "\nPenelusuran In-order: ";
            inOrder(root);
            cout << endl;
            break;
        case 9:
            cout << "\nPenelusuran Post-order: ";
            postOrder(root);
            cout << endl;
            break;
        case 10:
            deleteTree(root);
            cout << "\nPohon berhasil dihapus." << endl;
```

```
            break;
        case 11:
            deleteSub(root);
            break;
        case 12:
            clear();
            break;
        case 13:
            characteristic();
            break;
        case 14:
            return 0;
        default:
            cout << "\nInput salah, silakan masukkan pilihan
yang benar!" <<
endl;
        }
    }
    return 0;
}
```

Screenshoot program

```
Menu:
1. Buat Node Baru
2. Tambah Kiri
3. Tambah Kanan
4. Ubah Data Tree
5. Lihat Isi Data Tree
6. Cari Data Tree
7. Penelusuran Pre-order
8. Penelusuran In-order
9. Penelusuran Post-order
10. Hapus Node Tree
11. Hapus SubTree
12. Hapus Tree
13. Karakteristik Tree
14. Keluar
Pilihan: 2

Masukkan data baru: E

 Node E berhasil ditambahkan ke child kiri A
```

```
Menu:
1. Buat Node Baru
2. Tambah Kiri
3. Tambah Kanan
4. Ubah Data Tree
5. Lihat Isi Data Tree
6. Cari Data Tree
7. Penelusuran Pre-order
8. Penelusuran In-order
9. Penelusuran Post-order
10. Hapus Node Tree
11. Hapus SubTree
12. Hapus Tree
13. Karakteristik Tree
14. Keluar
Pilihan: 3

Masukkan data baru: I

 Node I berhasil ditambahkan ke child kananA
```

```
Menu:
1. Buat Node Baru
2. Tambah Kiri
3. Tambah Kanan
4. Ubah Data Tree
5. Lihat Isi Data Tree
6. Cari Data Tree
7. Penelusuran Pre-order
8. Penelusuran In-order
9. Penelusuran Post-order
10. Hapus Node Tree
11. Hapus SubTree
12. Hapus Tree
13. Karakteristik Tree
14. Keluar
Pilihan: 9

Penelusuran Post-order:  E,  I,  A,
```

Deskripsi program

Program di atas adalah implementasi sederhana dari struktur data pohon biner menggunakan bahasa pemrograman C++. Pada program ini, user dapat membuat node baru, menambahkan node kiri atau kanan, mengubah data node, melihat isi data node, mencari node, melakukan penelusuran preOrder, inOrder, dan postOrder, menghapus subtree, menghapus seluruh pohon, dan melihat karakteristik pohon seperti ukuran (jumlah node), tinggi, dan rata-rata node per level.

BAB IV
KESIMPULAN

Berikut adalah kesimpulan yang dapat diambil dari laporan praktikum kali ini:

Graph:

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

Tree:

- Tree adalah struktur data hierarkis yang terdiri dari simpul (node) yang terhubung dengan cara khusus.
- Setiap simpul dalam tree memiliki tepat satu simpul yang menjadi parent (induk) kecuali root, dan dapat memiliki banyak anak (child) atau tidak sama sekali.
- Tree digunakan untuk merepresentasikan hubungan hierarkis, seperti struktur organisasi, struktur direktori pada sistem file, struktur sintaksis dalam pemrograman, dan lainnya.
- Tree memiliki operasi khusus seperti traversal (melintasi) tree (Pre-order, In-order, dan Post-order), pencarian, penambahan, penghapusan, dan lainnya.
- Beberapa jenis tree yang populer meliputi Binary Tree, Binary Search Tree (BST), AVL Tree, Red-Black Tree, dan lainnya. Setiap jenis tree memiliki karakteristik dan aturan tertentu.