LAPORAN PRAKTIKUM

MODUL IX
GRAPH AND TREE

Disusun oleh:
Geranada Saputra Priambudi
NIM: 2311102008

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M,Eng.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024

BAB I

TUJUAN PRAKTIKUM

a. Mahasiswa diharapkan mampu memahami graph dan tree.

b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman.

BAB II

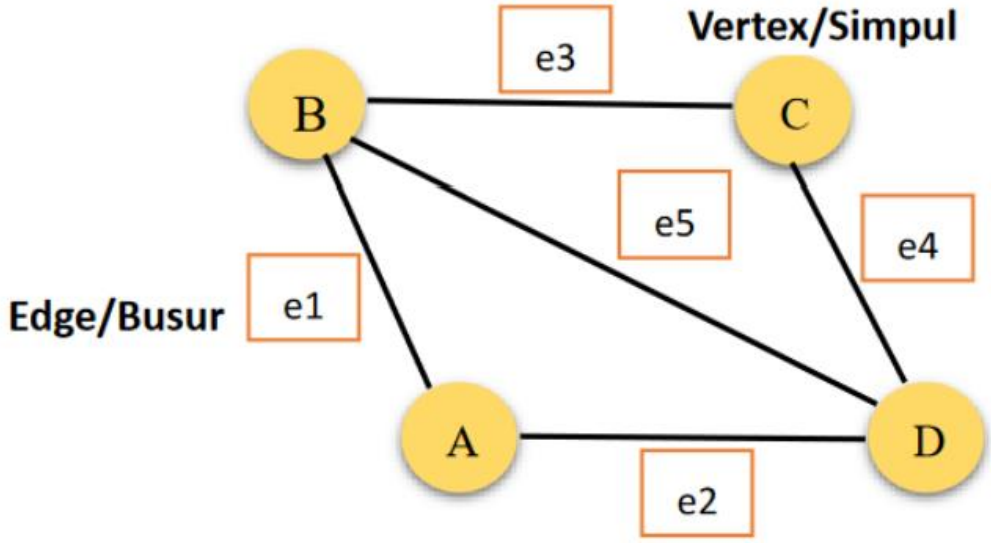
DASAR TEORI

- 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 :

$$G = (V, E)$$

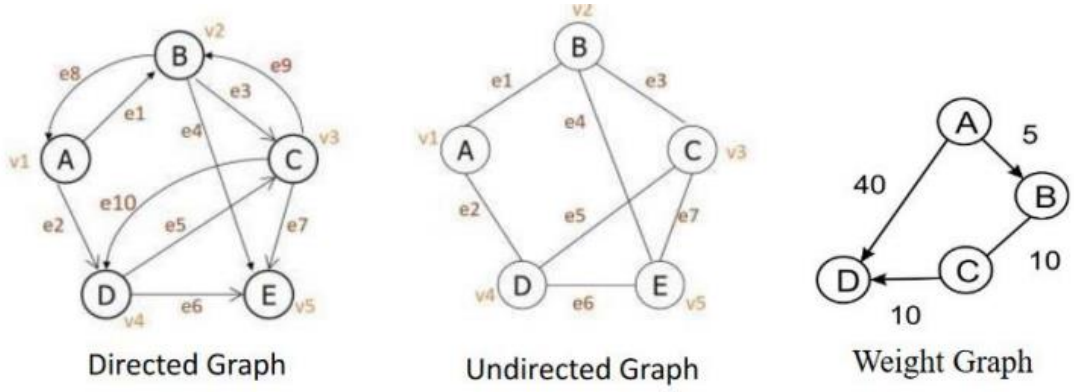
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data. Dapat digambarkan:



Gambar 1 Contoh Graph

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

Jenis-jenis Graph



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

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

c. Weigth Graph : Graph yang mempunyai nilai pada tiap edgenya.

- Tree

Dalam ilmu komputer, pohon 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, di mana 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 hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi.

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 pointercurrent dengan syarat tree tidak boleh kosong.

h. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruhdescendant-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, InOrder, dan Post-Order.

BAB III

GUIDED

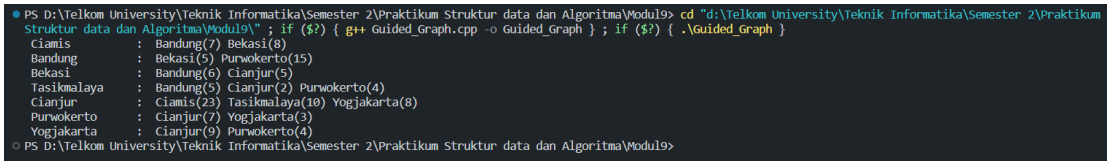
1. Guided 1 (Graph)

Source code

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

using namespace std;

string simpul[7] ={"Ciamis", "Bandung", "Bekasi", "Tasikmalaya",
"Cianjur", "Purwokerto", "Yogjakarta"};
int busur[7][7] =
{
    {0,7,8,0,0,0,0},
    {0,0,5,0,0,15,0},
    {0,6,0,0,5,0,0},
    {0,5,0,0,2,4,0},
    {23,0,0,10,0,0,8},
    {0,0,0,0,7,0,3},
    {0,0,0,0,9,4,0}
};

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

```
            cout << endl;
        }
}


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

Screenshoot program



Deskripsi program

Program diatas adalah implementasi dasar dari sebuah graf terarah menggunakan C++ yang menunjukkan simpul (node) dan busur (edge) yang menghubungkan mereka. Secara keseluruhan, program ini memberikan representasi visual dari sebuah graf terarah yang menunjukkan koneksi antar simpul beserta berat dari masing-masing koneksi tersebut.

2. Guided 2 (Tree)

Source Code

```
#include <iostream>
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() {
    if (root == NULL)
        return 1;
    else
        return 0;
}

// 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 {
        if( node->left != NULL ) {
            cout << "\n Node "<< node->data << " sudah ada 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;
        }
    }
}

// Tambah Kanan
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 ada 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;
        }
    }
}

// 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();
}
```

Screenshot Program

```
PS D:\Telkom University\Teknik Informatika\Semester 2\Praktikum Struktur data dan Algoritma\Modul9> cd "d:\Telkom University\Teknik Informatika\Semester 2\Praktikum Struktur data dan Algoritma\Modul9\" ; if ($?) { g++ Guided_Tree.cpp -o Guided_Tree } ; if ($?) { .\Guided_Tree }

 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 D:\Telkom University\Teknik Informatika\Semester 2\Praktikum Struktur data dan Algoritma\Modul9>
```

Deskripsi program

Program tersebut adalah implementasi dasar dari struktur data pohon biner menggunakan bahasa pemrograman C++. Program ini mendefinisikan berbagai fungsi untuk manipulasi pohon biner, seperti membuat node baru, menambahkan node ke kiri atau kanan, mengubah data node, melakukan traversal, dan menghapus node.

LATIHAN KELAS - UNGUIDED

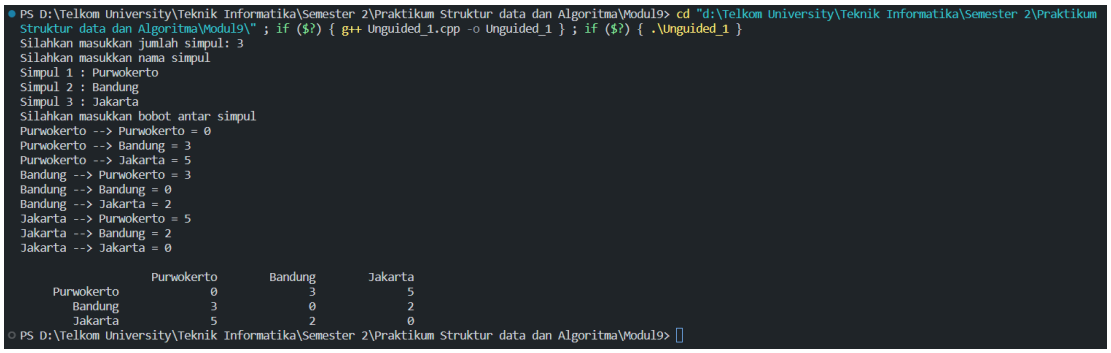
1. Unguided 1

Source code

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

using namespace std;

int main() {
    int jumlahSimpul;

    // Meminta pengguna untuk memasukkan jumlah simpul
    cout << "Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cin.ignore();  //  Mengabaikan  newline  setelah  memasukkan
angka

    vector<string> namaSimpul(jumlahSimpul);
    vector<vector<int>>                    bobot(jumlahSimpul,
vector<int>(jumlahSimpul, 0));

    // Meminta pengguna untuk memasukkan nama-nama simpul
    cout << "Silahkan masukkan nama simpul\n";
    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << "Simpul " << i + 1 << " : ";
        getline(cin, namaSimpul[i]);
    }

    // Meminta pengguna untuk memasukkan bobot antar simpul
    cout << "Silahkan masukkan bobot antar simpul\n";
```

```
    for (int i = 0; i < jumlahSimpul; ++i) {
        for (int j = 0; j < jumlahSimpul; ++j) {
            cout << namaSimpul[i] << " --> " << namaSimpul[j] <<
" = ";
            cin >> bobot[i][j];
        }
    }

    // Menampilkan matriks jarak antar simpul
    cout << "\n";
    cout << setw(15) << "";
    for (const auto& nama : namaSimpul) {
        cout << setw(15) << nama;
    }
    cout << "\n";

    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << setw(15) << namaSimpul[i];
        for (int j = 0; j < jumlahSimpul; ++j) {
            cout << setw(15) << bobot[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Screenshoot program



Deskripsi program

Program tersebut adalah sebuah program yang meminta pengguna untuk memasukkan informasi tentang graf berbobot. Program ini meminta pengguna untuk memasukkan jumlah simpul, nama-nama simpul, serta bobot antar simpul. Setelah itu, program menampilkan matriks jarak antar simpul. Program ini sangat berguna untuk merepresentasikan graf berbobot dan dapat digunakan untuk berbagai keperluan seperti analisis jaringan, rute terpendek, atau visualisasi hubungan antar node.

2. Unguided 2

Source code

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

using namespace std;

void          inputTreeData(vector<string>&          namaSimpul,
vector<vector<int>>& bobot, int& jumlahSimpul) {
    cout << "Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cin.ignore();  //  Mengabaikan  newline  setelah  memasukkan
angka

    namaSimpul.resize(jumlahSimpul);
    bobot.assign(jumlahSimpul, vector<int>(jumlahSimpul, 0));

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

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

```
void displayMatrix(const vector<string>& namaSimpul, const
vector<vector<int>>& bobot) {
    int jumlahSimpul = namaSimpul.size();

    cout << "\n";
    cout << setw(15) << "";
    for (const auto& nama : namaSimpul) {
        cout << setw(15) << nama;
    }
    cout << "\n";

    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << setw(15) << namaSimpul[i];
        for (int j = 0; j < jumlahSimpul; ++j) {
            cout << setw(15) << bobot[i][j];
        }
        cout << "\n";
    }
}

int main() {
    vector<string> namaSimpul;
    vector<vector<int>> bobot;
    int jumlahSimpul = 0;
    int pilihan;

    do {
        cout << "\nMenu:\n";
        cout << "1. Masukkan data tree\n";
        cout << "2. Tampilkan matriks jarak antar simpul\n";
        cout << "3. Keluar\n";
        cout << "Pilih opsi: ";
        cin >> pilihan;
```

```
        switch (pilihan) {
            case 1:
                inputTreeData(namaSimpul, bobot, jumlahSimpul);
                break;
            case 2:
                displayMatrix(namaSimpul, bobot);
                break;
            case 3:
                cout << "Keluar dari program.\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silahkan coba
lagi.\n";
                break;
        }
    } while (pilihan != 3);

    return 0;
}
```

Screenshot program

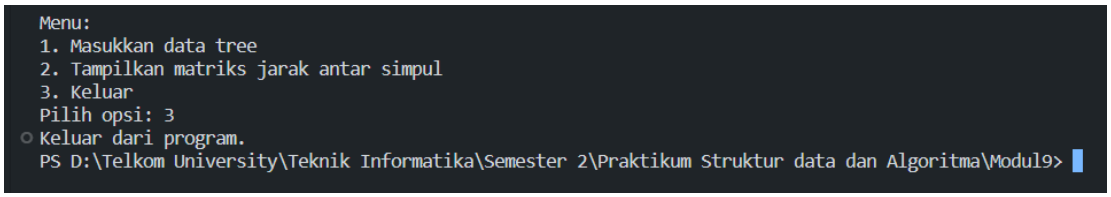
Menu 1

```
Menu:
1. Masukkan data tree
2. Tampilkan matriks jarak antar simpul
3. Keluar
Pilih opsi: 1
Silahkan masukkan jumlah simpul: 2
Silahkan masukkan nama simpul
Simpul 1 : Medan
Simpul 2 : Tangerang
Silahkan masukkan bobot antar simpul
Medan --> Medan = 3
Medan --> Tangerang = 8
Tangerang --> Medan = 7
Tangerang --> Tangerang = 4
```

Menu 2

```
Menu:
1. Masukkan data tree
2. Tampilkan matriks jarak antar simpul
3. Keluar
Pilih opsi: 2

                  Medan      Tangerang
       Medan          3              8
   Tangerang          7              4
```

Menu 3



Deskripsi program

Program tersebut adalah program interaktif yang memungkinkan pengguna untuk memasukkan data graf berbobot, menyimpan data tersebut, dan kemudian menampilkan matriks jarak antar simpul. Program ini memiliki menu interaktif dengan beberapa pilihan, yaitu memasukkan data tree, menampilkan matriks jarak, dan keluar dari program. Program ini berguna untuk memasukkan dan menampilkan data graf berbobot. Dengan menggunakan menu interaktif, pengguna dapat dengan mudah memasukkan data dan melihat matriks jarak antar simpul. Program ini mengajarkan dasar-dasar manipulasi vektor, pengolahan input/output, serta penggunaan loop dan kondisi.

BAB IV

KESIMPULAN

1. Graf (Graph):
   - Graf adalah struktur data yang merepresentasikan hubungan antara objek dalam bentuk node (simpul) dan sambungan antara node tersebut dalam bentuk edge (busur).
   - Graf dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.
   - Dalam bahasa pemrograman C++, graf dapat diimplementasikan menggunakan pendekatan berbasis objek dengan menggunakan kelas dan struktur data yang sesuai, seperti menggunakan representasi adjacency list atau adjacency matrix.

2. Pohon (Tree):
   - Pohon adalah struktur data yang terdiri dari satu set node yang terhubung secara hierarkis, di mana setiap node memiliki paling banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu.
   - Pohon digunakan untuk memodelkan hubungan hierarkis antara data, seperti struktur direktori pada sistem operasi atau representasi hierarki dalam struktur data.
   - Dalam bahasa pemrograman C++, pohon dapat diimplementasikan menggunakan pendekatan berbasis objek dengan menggunakan kelas dan pointer yang sesuai untuk menghubungkan antara simpul-simpul dalam pohon.

Dalam bahasa pemrograman C++, baik graf maupun pohon dapat diimplementasikan menggunakan struktur data dan konsep-konsep yang disediakan oleh bahasa tersebut. Implementasi yang baik dan efisien dari graf dan pohon membutuhkan pemahaman yang baik tentang algoritma dan struktur data yang sesuai, serta penggunaan.