

LAPORAN PRAKTIKUM

MODUL IX GRAPH AND TREE



**Disusun oleh:
Ilhan Sahal Mansiz
2311102029**

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

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023**

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

Graph adalah struktur data non-linier yang terdiri dari kumpulan node atau simpul (vertices) yang dihubungkan oleh garis atau sisi (edges). Node dapat mewakili objek dalam data sementara edges mewakili hubungan antara objek-objek tersebut. Graph digunakan untuk memodelkan hubungan yang kompleks dan saling terkait, seperti jaringan sosial, peta jalan, atau sistem rekomendasi. Ada beberapa jenis graph berdasarkan karakteristiknya, seperti graph berarah (directed graph) di mana hubungan memiliki arah, dan graph tidak berarah (undirected graph) di mana hubungan tidak memiliki arah tertentu.

Graph dapat diklasifikasikan ke dalam berbagai jenis berdasarkan sifatnya. Graph berarah (directed graph) memiliki arah pada setiap edges yang menunjukkan hubungan dari satu node ke node lainnya. Graph tidak berarah (undirected graph) tidak memiliki arah khusus pada edges. Selain itu, ada graph berbobot (weighted graph) di mana setiap edge memiliki nilai atau bobot yang menunjukkan kekuatan atau biaya hubungan. Graph juga dapat berupa graph siklik (cyclic graph) jika memiliki jalur yang membentuk lingkaran, atau graph asiklik (acyclic graph) jika tidak memiliki lingkaran.

Tree adalah struktur data hierarkis yang terdiri dari node yang dihubungkan dalam bentuk cabang-cabang seperti pohon. Setiap tree memiliki satu node akar (root) dari mana semua node lainnya bercabang. Tree digunakan untuk menyimpan data yang memiliki hubungan hierarkis, seperti struktur folder di komputer atau data silsilah keluarga. Jenis-jenis tree termasuk binary tree, di mana setiap node memiliki paling banyak dua anak, dan binary search tree, yang menjaga urutan elemen sehingga memudahkan pencarian.

Tree sangat berguna dalam banyak aplikasi pemrograman karena struktur hierarkisnya mempermudah proses pencarian dan pengelolaan data. Misalnya, dalam algoritma pencarian biner, tree memungkinkan pencarian yang cepat dengan membagi ruang pencarian secara berulang. Selain itu, balanced tree, seperti AVL tree dan red-black tree, memastikan operasi tetap efisien bahkan dalam kasus terburuk. Dalam sistem database, tree digunakan untuk mengindeks data dan mempercepat akses ke informasi yang diinginkan.

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;

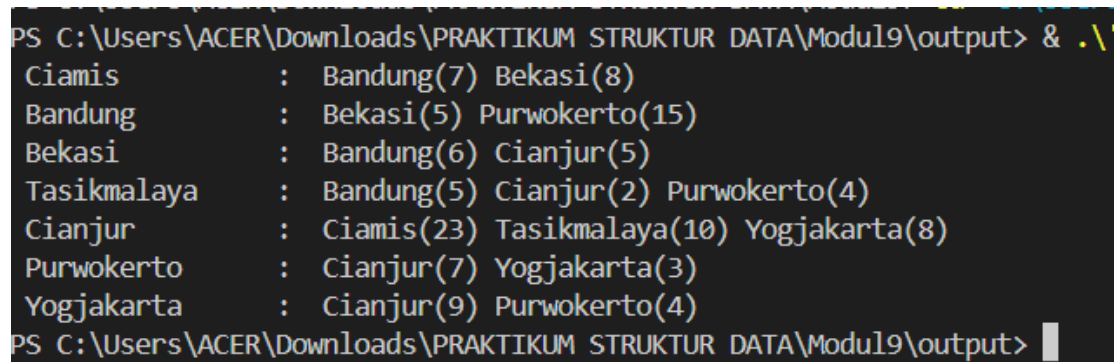
}

}

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

```

Screenshoot program



```

PS C:\Users\ACER\Downloads\PRAKTIKUM STRUKTUR DATA\Modul9\output> & .\
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS C:\Users\ACER\Downloads\PRAKTIKUM STRUKTUR DATA\Modul9\output>

```

Deskripsi program

Program ini mengimplementasikan dan menampilkan representasi dari sebuah graph berarah (directed graph) yang terdiri dari 7 node atau simpul. Setiap simpul diwakili oleh nama kota di Jawa Barat dan sekitarnya, yaitu: "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogyakarta". Hubungan antara kota-kota ini direpresentasikan dalam sebuah matriks dua dimensi bernama `busur`, yang mencatat bobot atau jarak antara dua simpul. Matriks ini menunjukkan adanya jalan langsung (busur) dari satu kota ke kota lain dan bobot busur tersebut yang bisa menggambarkan jarak atau waktu tempuh. Nilai nol

dalam matriks ``busur`` menunjukkan bahwa tidak ada hubungan langsung antara dua simpul.

Fungsi ``tampilGraph()`` bertugas untuk menampilkan graph ini dalam format yang mudah dibaca. Fungsi ini melakukan iterasi melalui setiap simpul dan mencetak nama kota tersebut. Untuk setiap kota, fungsi ini juga mencetak semua koneksi atau busur yang keluar dari kota tersebut beserta bobotnya. Misalnya, jika terdapat busur dari "Ciamis" ke "Bandung" dengan bobot 7, output akan menunjukkan "Bandung(7)". Fungsi ini menggunakan manipulasi string untuk memastikan bahwa outputnya rapi dan mudah dibaca, menggunakan ``setw`` dan ``setiosflags`` untuk menyelaraskan teks. Ketika fungsi ``main()`` dijalankan, ia memanggil ``tampilGraph()``, sehingga seluruh struktur graph ditampilkan di konsol.

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;
    else
        return 0;
    // true
    // 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
kechild 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

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan kechild kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan kechild kiri C

Node G berhasil ditambahkan kechild kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan kechild 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
```

Deskripsi program

Program ini adalah implementasi dasar dari pohon biner (binary tree) dalam bahasa C++. Struktur dasar pohon diwakili oleh `struct Pohon`, yang terdiri dari elemen data, serta pointer ke node anak kiri (`left`), anak kanan (`right`), dan node induk (`parent`). Program dimulai dengan menginisialisasi pohon menggunakan fungsi `init()`, yang mengatur `root` atau akar pohon menjadi `NULL`. Fungsi `isEmpty()` digunakan untuk memeriksa apakah pohon kosong, yaitu jika `root` masih `NULL`.

Fungsi utama untuk manipulasi pohon meliputi `buatNode()`, yang membuat node baru dan mengaturnya sebagai akar pohon jika pohon belum ada. Fungsi `insertLeft()` dan `insertRight()` menambahkan node baru sebagai anak kiri atau kanan dari node yang diberikan, dengan pengecekan apakah node anak kiri atau kanan sudah ada. Program juga memiliki kemampuan untuk memperbarui data dalam node dengan `update()`, menampilkan data node dengan `retrieve()`, dan mencari serta menampilkan karakteristik node dan hubungannya dengan node lain dalam pohon menggunakan `find()`.

Selain itu, program mendukung beberapa metode penelusuran pohon biner seperti `preOrder`, `inOrder`, dan `postOrder`, yang menelusuri dan mencetak data dari setiap node sesuai urutan penelusuran yang ditentukan. Program ini juga menyediakan fungsi untuk menghapus subtree dengan `deleteSub()`, menghapus keseluruhan pohon dengan `clear()`, dan menghitung ukuran (`size()`) serta tinggi pohon (`height()`). Fungsi `characteristic()` memberikan informasi tentang ukuran, tinggi, dan rata-rata node dari pohon. Pada bagian `main()`, program menunjukkan bagaimana node baru ditambahkan ke pohon, memperbarui node, dan menelusuri pohon dengan berbagai cara.

LATIHAN KELAS - UNGUIDED

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

Source code

```
#include <iostream>
#include <string>
using namespace std;
const int max_vertices = 100;
int ilhan_2311102029;

int main(){
    int numvertices;
    cout << "Silahkan Masukan jumlah simpul : ";
    cin >> numvertices;

    string vertices[max_vertices];
    int weight[max_vertices][max_vertices];

    cout << "Silahkan masukan nama simpul" << endl;
    for (int i = 0; i < numvertices; i++){
        cout << "simpul" << i+1 << ": ";
        cin >> vertices[i];
    }

    cout << "silahkan masukan bobot antar simpul" << endl;
    for(int i=0; i<numvertices; i++){
        for(int h=0; h<numvertices; h++){
            cout <<vertices[i] << " - " <<vertices[h]<<": ";
            cin >> weight[i][h];
        }
    }
    cout<<endl<<"bobot: " << endl;
```

```

cout << "          ";
for(int i=0; i<numvertices; i++){
    cout<<vertices[i]<<" ";
}
cout<<endl;

for (int i=0; i<numvertices; i++){
    cout<<vertices[i]<<" ";
    cout<< string(10- vertices[i].size(), ' ');

    for (int h=0; h<numvertices; h++){
        cout << weight[i][h]<< " ";
    }

    cout << endl;
}
return 0;
} // ILHAN SAHAL MANSIZ // 2311102029

```

Screenshot Program

```

Silahkan Masukan jumlah simpul : 2
Silahkan masukan nama simpul
simpul1: BALI
simpul2: PALU
silahkan masukan bobot antar simpul
BALI - BALI: 0
BALI - PALU: 3
PALU - BALI: 4
PALU - PALU: 0

bobot:
          BALI PALU
BALI      0 3
PALU      4 0

```

Deskripsi Program

Program ini adalah aplikasi C++ untuk mendefinisikan dan mencetak representasi dari sebuah graph berbobot (weighted graph). Program dimulai dengan meminta pengguna untuk memasukkan jumlah simpul atau node yang akan digunakan dalam graph. Kemudian, pengguna diminta untuk memberikan nama untuk setiap simpul tersebut. Nama-nama simpul disimpan dalam sebuah array string bernama `vertices`. Dengan demikian, program dapat menangani hingga 100 simpul yang berbeda, sesuai dengan batas maksimum yang ditentukan oleh konstanta `max_vertices`.

Setelah nama-nama simpul diinput, program meminta pengguna untuk memasukkan bobot atau nilai dari setiap hubungan antara pasangan simpul. Bobot ini disimpan dalam sebuah matriks dua dimensi bernama `weight`, di mana `weight[i][j]` mewakili bobot dari simpul `i` ke simpul `j`. Matriks ini memungkinkan representasi dari semua hubungan yang mungkin antara simpul-simpul dalam graph, termasuk kemungkinan tidak adanya hubungan (yang dapat diindikasikan dengan bobot nol atau nilai yang sesuai dengan kebutuhan aplikasi).

Terakhir, program menampilkan matriks bobot yang sudah diisi pengguna dalam format tabel. Bagian ini menampilkan nama-nama simpul di sepanjang sumbu horizontal dan vertikal dari tabel. Setiap elemen dalam tabel menunjukkan bobot dari hubungan antara dua simpul tertentu. Misalnya, entri di baris ke-`i` dan kolom ke-`j` menunjukkan bobot dari simpul `i` ke simpul `j`. Penggunaan manipulasi string dalam bagian ini memastikan bahwa tampilan tabel tetap rapi dan mudah dibaca, dengan kolom yang sejajar dengan benar terlepas dari panjang nama simpul yang bervariasi. Program ini memberikan visualisasi yang jelas dari struktur graph berbobot yang telah ditentukan pengguna.

2. Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!

Source Code

```
#include <iostream>
#include <string>
using namespace std;
const int MAX_VERTICES = 100;
int ilhan_2311102029;

void printWeights(const string vertices[], const int
weights[][MAX_VERTICES],int numVertices){
    cout << "          ";
    for (int a=0; a<numVertices; a++){
        cout << vertices[a] << " ";
    }
    cout << endl;

    for (int a=0; a< numVertices; a++){
        cout << vertices[a] << " ";
        cout << string(10 - vertices[a].size(), ' ');

        for (int c=0; c<numVertices; c++){
            cout << weights[a][c] << " ";
        }
        cout << endl;
    }
}

int main(){
    string vertices[MAX_VERTICES];
    int weights[MAX_VERTICES][MAX_VERTICES];
    int numVertices = 0;

    int choice;
    while(true){
        cout << "1.Tambah data simpul" << endl;
        cout << "2.Tambah data bobot antar simpul" << endl;
        cout << "3.Tampilkan data bobot antar simpul" <<
endl;
        cout << "4.Keluar" << endl;
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                if (numVertices >= MAX_VERTICES){
                    cout << "Jumlah simpul maksimum telah
tercapai!" << endl;
                }else {
```

```

        cout << "Simpul " << numVertices + 1 << ":
";
        cin >> vertices[numVertices];
        numVertices++;
    }
    break;

    case 2:
    if (numVertices == 0){
        cout << "simpul belum ada yang ditambahkan!"
<< endl;
    }else {
        cout << "Silahkan masukan bobot antar
simpul:" << endl;
        for (int a=0; a<numVertices; a++){
            for (int c=0; c<numVertices; c++){
                cout << vertices[a] << " - " <<
vertices[c] << ": ";
                cin >> weights[a][c];
            }
        }
        break;

    case 3:
    if (numVertices == 0){
        cout << "simpul belum ada yang ditambahkan!"
<< endl;
    }else{
        cout << endl << "Bobot :" << endl;
        printWeights(vertices, weights,
numVertices);
    }
    break;

    case 4:
    cout << "Terima kasih!" << endl;
    return 0;

    default:
    cout << "Pilihan tidak valid" << endl;
    break;
    }
    cout << endl;
}
return 0;
} // ILHAN SAHAL M // 2311102029

```

Screenshot Program

```
1.Tambah data simpul
2.Tambah data bobot antar simpul
3.Tampilkan data bobot antar simpul
4.Keluar
```

Pilihan Anda: 1

Simpul 1: BALI

```
1.Tambah data simpul
2.Tambah data bobot antar simpul
3.Tampilkan data bobot antar simpul
4.Keluar
```

Pilihan Anda: 1

Simpul 2: PALU

```
1.Tambah data simpul
2.Tambah data bobot antar simpul
3.Tampilkan data bobot antar simpul
4.Keluar
```

Pilihan Anda: 2

Silahkan masukan bobot antar simpul:

BALI - BALI: 0

BALI - PALU: 3

PALU - BALI: 4

PALU - PALU: 0

```
1.Tambah data simpul
2.Tambah data bobot antar simpul
3.Tampilkan data bobot antar simpul
4.Keluar
```

Pilihan Anda: 3

Bobot :

	BALI	PALU
BALI	0 3	
PALU	4 0	

Deskripsi Program

Program ini adalah sebuah aplikasi interaktif dalam bahasa C++ yang berfungsi untuk membuat dan mengelola graph berbobot. Graph tersebut terdiri dari simpul-simpul (vertices) yang saling terhubung oleh busur (edges) dengan bobot tertentu. Pengguna dapat secara dinamis menambahkan simpul ke graph, mengisi bobot antar simpul, dan menampilkan tabel bobot antar simpul dengan menggunakan antarmuka menu yang tersedia dalam program. Struktur dasar dari graph ini dikelola menggunakan dua array: `vertices` untuk menyimpan nama-nama simpul dan `weights` untuk menyimpan bobot hubungan antara pasangan simpul dalam bentuk matriks dua dimensi.

Pada awal eksekusi, program meminta pengguna untuk memilih salah satu dari beberapa opsi menu: menambahkan simpul baru, memasukkan bobot antar simpul, menampilkan tabel bobot antar simpul, atau keluar dari program. Setiap opsi menu diproses melalui pernyataan switch-case. Jika pengguna memilih untuk menambahkan simpul baru, mereka akan diminta untuk memasukkan nama simpul, yang kemudian disimpan dalam array `vertices`. Namun, jika jumlah simpul yang ditambahkan sudah mencapai batas maksimum (`MAX_VERTICES`), program akan memberi tahu pengguna bahwa mereka tidak bisa menambahkan simpul lebih lanjut. Jika pengguna memilih untuk menambahkan bobot antar simpul, mereka akan diminta untuk mengisi bobot setiap hubungan yang mungkin antara semua pasangan simpul yang ada.

Untuk menampilkan tabel bobot antar simpul, program menggunakan fungsi `printWeights()`, yang mencetak matriks bobot dalam format yang terstruktur dengan baik. Fungsi ini menampilkan nama simpul baik di baris maupun kolom pertama dari tabel, dan kemudian mencetak bobot dari hubungan antar simpul sesuai dengan entri dalam matriks `weights`. Jika belum ada simpul yang ditambahkan, program akan memberi tahu pengguna bahwa simpul harus ditambahkan terlebih dahulu sebelum mereka bisa mengatur atau melihat bobot antar simpul. Pilihan keempat dari menu memungkinkan pengguna untuk keluar dari program dengan mengucapkan "Terima kasih!". Program ini menyediakan cara yang efektif untuk membangun dan menganalisis graph berbobot, yang dapat digunakan dalam berbagai aplikasi seperti analisis jaringan dan pemodelan hubungan.

BAB IV

KESIMPULAN

Dalam Laprak Modul 9 dari materi Struktur Data, kita telah mengeksplorasi dua struktur data yang esensial: Graph dan Tree. Graph adalah struktur data yang terdiri dari simpul-simpul (nodes) yang dihubungkan oleh busur-busur (edges), memungkinkan kita untuk merepresentasikan hubungan kompleks antara entitas. Graph dapat berbentuk graph berarah (directed graph) yang memiliki arah pada setiap busurnya, atau graph tidak berarah (undirected graph) yang tidak memiliki arah. Selain itu, graph juga bisa berbobot (weighted graph) jika setiap busur memiliki nilai (bobot) tertentu, atau tidak berbobot (unweighted graph). Graph sering digunakan dalam berbagai aplikasi seperti analisis jaringan sosial, pemodelan rute transportasi, dan jaringan komputer karena kemampuannya dalam menangani hubungan banyak-ke-banyak antar entitas. Pada modul ini, kita juga mempelajari berbagai metode representasi graph seperti matriks ketetanggaan (adjacency matrix) dan daftar ketetanggaan (adjacency list), serta algoritma dasar untuk traversal dan pencarian dalam graph.

Tree adalah bentuk khusus dari graph yang memiliki struktur hierarkis, di mana setiap simpul memiliki satu induk (kecuali simpul akar yang tidak memiliki induk) dan satu atau lebih anak. Struktur Tree digunakan untuk merepresentasikan data yang memiliki hubungan satu-ke-banyak secara alami, seperti dalam sistem file komputer, struktur organisasi, dan database. Pohon biner (binary tree) adalah jenis tree yang populer di mana setiap simpul memiliki maksimal dua anak, sedangkan pohon pencarian biner (binary search tree) memiliki properti tambahan yang memudahkan pencarian data. Pada modul ini, kita mempelajari cara membangun dan memanipulasi tree, termasuk operasi penambahan simpul, penghapusan simpul, dan traversal tree menggunakan metode pre-order, in-order, dan post-order. Memahami konsep dan operasi dasar dari graph dan tree sangat penting karena keduanya merupakan fondasi dari banyak algoritma dan struktur data kompleks dalam ilmu komputer.

BAB V

REFERENSI

- <https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>
- <https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html>
- <https://www.trivusi.web.id/2022/07/struktur-data-graph.html>