

LAPORAN PRAKTIKUM

MODUL IX GRAPH DAN TREE



Disusun oleh:
Muhammad Rifki Fadhilah
NIM: 2311102032

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

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

BAB I

TUJUAN PRAKTIKUM

1. Menunjukkan beberapa algoritma dalam Pencarian.
2. Menunjukkan bahwa pencarian merupakan suatu persoalan yang bisa diselesaikan dengan beberapa algoritma yang berbeda.
3. Dapat memilih algoritma yang paling sesuai untuk menyelesaikan suatu permasalahan pemrograman.

BAB II

DASAR TEORI

Graf dan pohon adalah struktur data penting dalam ilmu komputer dengan berbagai operasi yang memungkinkan pengelolaan dan manipulasi data secara efisien.

Pada graf, operasi dasar meliputi:

1. Penambahan Simpul (Vertex Addition): Menambah simpul baru ke dalam graf.
2. Penghapusan Simpul (Vertex Deletion): Menghapus simpul dari graf beserta semua sisi yang terhubung ke simpul tersebut.
3. Penambahan Sisi (Edge Addition): Menambah sisi baru yang menghubungkan dua simpul dalam graf.
4. Penghapusan Sisi (Edge Deletion): Menghapus sisi yang menghubungkan dua simpul.
5. Pencarian Simpul (Vertex Search): Mencari apakah suatu simpul ada dalam graf.
6. Pencarian Sisi (Edge Search): Mencari apakah suatu sisi yang menghubungkan dua simpul ada dalam graf.
7. Penelusuran Graf (Graph Traversal): Menelusuri semua simpul dalam graf menggunakan algoritma seperti BFS (Breadth-First Search) dan DFS (Depth-First Search).

Pada pohon, operasi dasar meliputi:

1. Penambahan Simpul (Node Addition): Menambah simpul baru ke dalam pohon di posisi yang sesuai.
2. Penghapusan Simpul (Node Deletion): Menghapus simpul dari pohon dan menyesuaikan struktur pohon agar tetap valid.
3. Penelusuran Pohon (Tree Traversal): Menelusuri semua simpul dalam pohon menggunakan metode seperti preorder, inorder, dan postorder.
4. Pencarian Simpul (Node Search): Mencari apakah suatu simpul ada dalam pohon.
5. Penyeimbangan Pohon (Tree Balancing): Menyeimbangkan pohon untuk memastikan operasi pencarian, penambahan, dan penghapusan berjalan efisien, seperti pada pohon AVL.
6. Pencarian Minimum/Maksimum (Find Minimum/Maximum): Menemukan simpul dengan nilai minimum atau maksimum dalam pohon biner pencarian (BST).

7. Pencarian Predecessor/Successor: Menemukan simpul yang merupakan pendahulu atau penerus dari simpul tertentu dalam pohon biner pencarian.

BAB III

GUIDED

1. Guided 1

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

```

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 D:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul9\output>

```

Deskripsi program

Program ini adalah implementasi graf yang menggunakan daftar simpul dan matriks bobot busur. Program ini menyimpan nama-nama kota sebagai simpul dalam sebuah array bernama simpul, dan jarak atau biaya perjalanan antara kota-kota tersebut dalam sebuah matriks bernama busur.

Setiap elemen dalam matriks busur mewakili bobot dari satu kota ke kota lainnya. Misalnya, jika ada nilai yang bukan nol di matriks tersebut, berarti ada jalur antara dua kota, dan nilainya menunjukkan bobot jalur tersebut.

Misalnya, `busur[0][1] = 7` berarti ada jalur dari kota pertama ke kota kedua dengan bobot 7.

Fungsi `tampilGraph` digunakan untuk menampilkan graf. Fungsi ini bekerja dengan mengiterasi setiap simpul (baris) dan kemudian mengiterasi setiap simpul lainnya (kolom) untuk memeriksa apakah ada jalur yang menghubungkan kedua simpul tersebut. Jika ada jalur, maka nama simpul tujuan dan bobotnya dicetak. Setiap simpul akan dicetak dengan nama kota diikuti dengan semua kota yang terhubung dengannya serta bobot masing-masing jalur.

Ketika program dijalankan, fungsi `main` memanggil `tampilGraph` untuk menampilkan graf. Output dari program ini adalah daftar kota-kota yang dihubungkan oleh jalur-jalur tertentu dengan jarak atau biaya yang ditentukan, sehingga memudahkan untuk melihat koneksi antar kota beserta bobot jalurnya.

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 A berhasil dibuat menjadi root.  
  
Node B berhasil ditambahkan kechild kiri A  
  
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,
```

Deskripsi program

Program ini adalah implementasi dari struktur data pohon biner. Program ini terdiri dari beberapa bagian, mulai dari deklarasi struktur pohon, inisialisasi, pengecekan node, penambahan node kiri dan kanan, pembaruan data node, penelusuran (traversal) pohon, hingga penghapusan node dan subtree.

Pada awalnya, struktur pohon dideklarasikan menggunakan struct `Pohon` yang memiliki data, pointer ke anak kiri, anak kanan, dan parent. Fungsi `init` digunakan untuk menginisialisasi root pohon menjadi NULL, yang menandakan bahwa pohon masih kosong. Fungsi `isEmpty` digunakan untuk memeriksa apakah pohon kosong atau tidak.

Fungsi `buatNode` digunakan untuk membuat node baru dan menjadikannya sebagai root jika pohon masih kosong. Jika pohon sudah ada, fungsi ini akan memberikan pesan bahwa pohon sudah dibuat.

Fungsi `insertLeft` dan `insertRight` digunakan untuk menambahkan node baru sebagai anak kiri atau kanan dari node yang ditentukan. Fungsi ini juga memeriksa apakah node yang ditentukan sudah memiliki

anak kiri atau kanan, dan jika sudah, maka node baru tidak akan ditambahkan.

Fungsi ``update`` digunakan untuk mengubah data pada node yang ditentukan. Fungsi ``retrieve`` digunakan untuk menampilkan data dari node yang ditentukan. Fungsi ``find`` digunakan untuk mencari dan menampilkan data dari node yang ditentukan, termasuk informasi tentang parent, sibling, dan anak kiri atau kanan dari node tersebut.

Program ini juga menyediakan fungsi-fungsi untuk penelusuran (traversal) pohon yaitu ``preOrder``, ``inOrder``, dan ``postOrder``. Fungsi ``preOrder`` mengunjungi root terlebih dahulu, kemudian anak kiri, dan terakhir anak kanan. Fungsi ``inOrder`` mengunjungi anak kiri terlebih dahulu, kemudian root, dan terakhir anak kanan. Fungsi ``postOrder`` mengunjungi anak kiri terlebih dahulu, kemudian anak kanan, dan terakhir root.

Fungsi ``deleteTree`` digunakan untuk menghapus seluruh pohon, sedangkan fungsi ``deleteSub`` digunakan untuk menghapus subtree dari node yang ditentukan. Fungsi ``clear`` digunakan untuk menghapus seluruh pohon.

Fungsi ``size`` digunakan untuk menghitung jumlah node dalam pohon, sedangkan fungsi ``height`` digunakan untuk menghitung tinggi dari pohon. Fungsi ``characteristic`` menampilkan karakteristik dari pohon seperti ukuran, tinggi, dan rata-rata jumlah node.

Pada fungsi ``main``, program membuat node root dengan data 'A', kemudian menambahkan beberapa node anak, memperbarui data node,

menampilkan data node, mencari node, melakukan penelusuran (traversal) pohon, menampilkan karakteristik pohon, dan menghapus subtree dari node tertentu.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul: ";
    cin >> jumlah_simpul;

    vector<string> nama_simpul(jumlah_simpul);
    vector<vector<int>> bobot(jumlah_simpul,
vector<int>(jumlah_simpul));

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

    cout << "Silahkan masukkan bobot antar simpul:" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << nama_simpul[i] << "-->" << nama_simpul[j] <<
" = ";
            cin >> bobot[i][j];
        }
    }
}
```

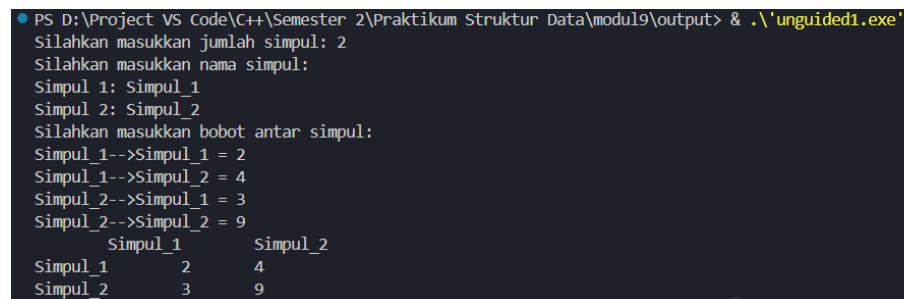
```

// Output tabel jarak
cout << "\t";
for (int i = 0; i < jumlah_simpul; ++i) {
    cout << nama_simpul[i] << "\t";
}
cout << endl;
for (int i = 0; i < jumlah_simpul; ++i) {
    cout << nama_simpul[i] << "\t";
    for (int j = 0; j < jumlah_simpul; ++j) {
        cout << bobot[i][j] << "\t";
    }
    cout << endl;
}

return 0;
}

```

Screenshoot program



```

PS D:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul9\output> & .\'unguided1.exe'
Silahkan masukkan jumlah simpul: 2
Silahkan masukkan nama simpul:
Simpul 1: Simpul_1
Simpul 2: Simpul_2
Silahkan masukkan bobot antar simpul:
Simpul_1->Simpul_1 = 2
Simpul_1->Simpul_2 = 4
Simpul_2->Simpul_1 = 3
Simpul_2->Simpul_2 = 9
      Simpul_1      Simpul_2
Simpul_1      2      4
Simpul_2      3      9

```

Deskripsi program

Program ini merupakan implementasi dari representasi graf dengan matriks berbobot. Program meminta input jumlah simpul, nama-nama simpul, dan bobot antar simpul dari pengguna. Bobot antar simpul disimpan dalam matriks bobot yang merupakan vektor dua dimensi.

Setelah menerima input, program akan menampilkan matriks bobot yang merepresentasikan graf berbobot. Matriks ini menunjukkan bobot dari setiap simpul ke simpul lainnya.

Pertama, program akan meminta pengguna untuk memasukkan jumlah simpul. Kemudian, program akan membuat vektor `nama_simpul` untuk menyimpan nama-nama simpul. Selanjutnya, program akan membuat vektor dua dimensi bobot untuk menyimpan bobot antar simpul. Matriks bobot akan diinisialisasi dengan ukuran `jumlah_simpul x jumlah_simpul`, dan setiap elemen matriks akan diisi dengan bobot antar simpul yang dimasukkan oleh pengguna.

Selanjutnya, program akan menampilkan matriks bobot sebagai representasi graf berbobot. Setiap baris dan kolom dalam matriks akan mewakili simpul, dan setiap elemen matriks akan berisi bobot dari simpul baris ke simpul kolom.

Program ini menggunakan vektor dari STL untuk menyimpan data nama simpul dan bobot antar simpul. Vektor digunakan karena memungkinkan untuk menyimpan jumlah data yang fleksibel dan memudahkan pengelolaan data dalam program.

2. Unguided 2

Source code

```
#include <iostream>
#include <vector>

using namespace std;
```

```

void inputNamaSimpul(vector<string> &nama_simpul, int
jumlah_simpul) {
    cout << "Silahkan masukkan nama simpul:" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> nama_simpul[i];
    }
}

void inputBobot(vector<vector<int>> &bobot, const vector<string>
&nama_simpul, int jumlah_simpul) {
    cout << "Silahkan masukkan bobot antar simpul:" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << nama_simpul[i] << "-->" << nama_simpul[j] <<
" = ";
            cin >> bobot[i][j];
        }
    }
}

void tampilkanTabelJarak(const vector<string> &nama_simpul, const
vector<vector<int>> &bobot, int jumlah_simpul) {
    cout << "\t";
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << nama_simpul[i] << "\t";
    }
    cout << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << nama_simpul[i] << "\t";
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << endl;
    }
}

```

```

    }
}

int main() {
    int jumlah_simpul;
    vector<string> nama_simpul;
    vector<vector<int>> bobot;

    int pilihan;

    do {
        cout << "\nMenu:\n";
        cout << "1. Input jumlah simpul\n";
        cout << "2. Input nama simpul\n";
        cout << "3. Input bobot antar simpul\n";
        cout << "4. Tampilkan tabel jarak\n";
        cout << "5. Keluar\n";
        cout << "Pilihan: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Silahkan masukkan jumlah simpul: ";
                cin >> jumlah_simpul;
                nama_simpul.resize(jumlah_simpul);
                bobot.resize(jumlah_simpul,
vector<int>(jumlah_simpul));
                break;
            case 2:
                if (jumlah_simpul == 0) {
                    cout << "Silahkan input jumlah simpul
terlebih dahulu!\n";
                } else {
                    inputNamaSimpul(nama_simpul, jumlah_simpul);
                }
            }
        }
    } while (pilihan != 5);
}

```

```

        }
        break;
    case 3:
        if (jumlah_simpul == 0) {
            cout << "Silahkan input jumlah simpul
terlebih dahulu!\n";
        } else {
            inputBobot(bobot, nama_simpul,
jumlah_simpul);
        }
        break;
    case 4:
        if (jumlah_simpul == 0) {
            cout << "Silahkan input jumlah simpul
terlebih dahulu!\n";
        } else if (nama_simpul.empty()) {
            cout << "Silahkan input nama simpul terlebih
dahulu!\n";
        } else if (bobot.empty()) {
            cout << "Silahkan input bobot antar simpul
terlebih dahulu!\n";
        } else {
            tampilkanTabelJarak(nama_simpul, bobot,
jumlah_simpul);
        }
        break;
    case 5:
        cout << "Keluar dari program.\n";
        break;
    default:
        cout << "Pilihan tidak valid. Silahkan coba
lagi.\n";
        break;
}

```

```
    } while (pilihan != 5);  
  
    return 0;  
}
```

Screenshoot program

```
Menu:  
1. Input jumlah simpul  
2. Input nama simpul  
3. Input bobot antar simpul  
4. Tampilkan tabel jarak  
5. Keluar  
Pilihan: 1  
Silahkan masukkan jumlah simpul: 2  
  
Menu:  
1. Input jumlah simpul  
2. Input nama simpul  
3. Input bobot antar simpul  
4. Tampilkan tabel jarak  
5. Keluar  
Pilihan: 2  
Silahkan masukkan nama simpul:  
Simpul 1: Simpul_1  
Simpul 2: Simpul_2  
  
Menu:  
1. Input jumlah simpul  
2. Input nama simpul  
3. Input bobot antar simpul  
4. Tampilkan tabel jarak  
5. Keluar  
Pilihan: 3  
Silahkan masukkan bobot antar simpul:  
Simpul_1-->Simpul_1 = 12  
Simpul_1-->Simpul_2 = 21  
Simpul_2-->Simpul_1 = 31  
Simpul_2-->Simpul_2 = 13
```

```

Menu:
1. Input jumlah simpul
2. Input nama simpul
3. Input bobot antar simpul
4. Tampilkan tabel jarak
5. Keluar
Pilihan: 4

      Simpul_1      Simpul_2
Simpul_1      12      21
Simpul_2      31      13

Menu:
1. Input jumlah simpul
2. Input nama simpul
3. Input bobot antar simpul
4. Tampilkan tabel jarak
5. Keluar
Pilihan: 5
Keluar dari program.

```

Deskripsi program

Program ini adalah implementasi dari representasi graf berbobot dengan menggunakan matriks berbobot. Program memungkinkan pengguna untuk menginput jumlah simpul, nama-nama simpul, dan bobot antar simpul. Selain itu, program juga dapat menampilkan tabel jarak yang merepresentasikan bobot antar simpul.

Program ini menggunakan vektor dari STL untuk menyimpan nama simpul dan bobot antar simpul. Vektor digunakan karena memungkinkan untuk menyimpan data yang dinamis dan mempermudah pengelolaan data dalam program.

Pada awal program, pengguna diminta untuk memilih menu yang tersedia. Menu-menu tersebut antara lain:

1. Input jumlah simpul: Pengguna diminta untuk memasukkan jumlah simpul yang akan digunakan dalam graf.
2. Input nama simpul: Pengguna diminta untuk memasukkan nama-nama simpul.
3. Input bobot antar simpul: Pengguna diminta untuk memasukkan bobot antar simpul.
4. Tampilkan tabel jarak: Program akan menampilkan tabel jarak yang merepresentasikan bobot antar simpul.
5. Keluar: Keluar dari program.

Program menggunakan loop `do-while` untuk menjalankan menu-menu tersebut selama pengguna belum memilih untuk keluar dari program. Setiap menu akan memeriksa apakah langkah-langkah sebelumnya telah dilakukan dengan benar sebelum menjalankan fungsinya. Jika langkah-langkah sebelumnya belum dilakukan, program akan memberikan pesan kesalahan kepada pengguna.

Program ini memungkinkan pengguna untuk mengatur graf berbobot dengan mudah dan melakukan operasi-operasi dasar terkait graf berbobot.

BAB IV

KESIMPULAN

Graf dan pohon adalah struktur data esensial dalam ilmu komputer yang menyediakan berbagai operasi untuk pengelolaan data secara efisien. Operasi dasar pada graf meliputi penambahan, penghapusan, dan pencarian simpul dan sisi, serta penelusuran menggunakan algoritma BFS dan DFS. Sedangkan pada pohon, operasi dasar mencakup penambahan, penghapusan, dan pencarian simpul, penelusuran dengan metode preorder, inorder, dan postorder, serta penyeimbangan pohon untuk menjaga efisiensi, seperti pada pohon AVL, dan pencarian nilai minimum atau maksimum dalam pohon biner pencarian. Struktur data ini sangat penting untuk aplikasi yang memerlukan representasi dan manipulasi data yang kompleks, seperti dalam jaringan komputer, pengolahan gambar, dan basis data.