LAPORAN PRAKTIKUM

MODUL IX
GRAPH AND TREE

Disusun oleh:
Maulana Ghani Rolanda
NIM: 2311102012

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

a. Mahasiswa diharapkan mampu memahami graph dan tree

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

BAB II
DASAR TEORI

Graf adalah struktur matematika yang digunakan untuk memodelkan hubungan pasangan antar objek, terdiri dari himpunan simpul (vertex atau node) dan himpunan sisi (edge) yang menghubungkan pasangan simpul. Graf dapat dibagi menjadi beberapa jenis, seperti graf berarah (digraph) di mana sisi-sisi memiliki arah, dan graf tak berarah di mana sisi-sisi tidak memiliki arah. Selain itu, ada graf berbobot di mana setiap sisi memiliki bobot atau nilai, dan graf tak berbobot yang tidak memiliki bobot pada sisinya. Konsep dasar dalam graf mencakup jalur (path), yaitu urutan simpul yang terhubung oleh sisi, siklus (cycle), yaitu jalur yang kembali ke simpul awal tanpa mengulang sisi, graf terhubung (connected graph) di mana setiap pasangan simpul terhubung oleh jalur, dan derajat (degree), yaitu jumlah sisi yang terhubung ke simpul.

Pohon adalah graf tak berarah yang terhubung dan tidak mengandung siklus, dengan definisi formal sebagai graf terhubung dengan nnn simpul dan n−1n-1n−1 sisi. Beberapa jenis pohon meliputi pohon biner, di mana setiap simpul memiliki paling banyak dua anak, pohon biner pencarian (Binary Search Tree) yang setiap simpulnya mengikuti aturan kiri (nilai lebih kecil) dan kanan (nilai lebih besar), serta pohon AVL yang selalu seimbang secara tinggi. Konsep dasar dalam pohon mencakup akar (root),

yaitu simpul tertinggi dalam pohon, daun (leaf) yang merupakan simpul tanpa anak, subpohon (subtree) yang merupakan bagian dari pohon yang lebih besar, dan tinggi (height) yang menunjukkan panjang jalur terpanjang dari akar ke daun.

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

```
PS C:\olan\vscode\Modul9> cd "c:\olan\vscode\Modul9\" ; if ($?)
 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) Yogjakarta(8)
 Purwokerto      :  Cianjur(7) Yogjakarta(3)
 Yogjakarta      :  Cianjur(9) Purwokerto(4)
PS C:\olan\vscode\Modul9>
```

Deskripsi program

Program di atas adalah sebuah program C++ yang menampilkan representasi graf berarah berbobot, yang memodelkan hubungan antar kota dengan bobot yang menggambarkan jarak atau biaya antar kota. Program ini menggunakan header `iostream` untuk input-output dan `iomanip` untuk manipulasi format output, serta namespace `std` untuk kemudahan penulisan fungsi standar. Variabel global `simpul` menyimpan nama-nama kota dalam array string, sedangkan `busur` adalah array 2D yang menyimpan bobot hubungan antar kota, di mana nilai `0` menunjukkan tidak ada koneksi langsung. Fungsi `tampilGraph` bertugas menampilkan graf dengan melakukan iterasi melalui setiap simpul asal dan tujuan, dan menampilkan kota tujuan beserta bobotnya jika terdapat koneksi. Fungsi utama `main` memanggil `tampilGraph` untuk menampilkan hasil graf ke layar. Output program ini menampilkan kota

asal diikuti oleh daftar kota tujuan yang terhubung langsung, beserta bobot hubungan tersebut dalam format yang mudah dibaca. Program ini efektif dalam memodelkan dan menampilkan hubungan berarah berbobot antar kota, memberikan gambaran visual tentang jarak atau biaya perjalanan antar kota.

2. Guided 2

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);
```

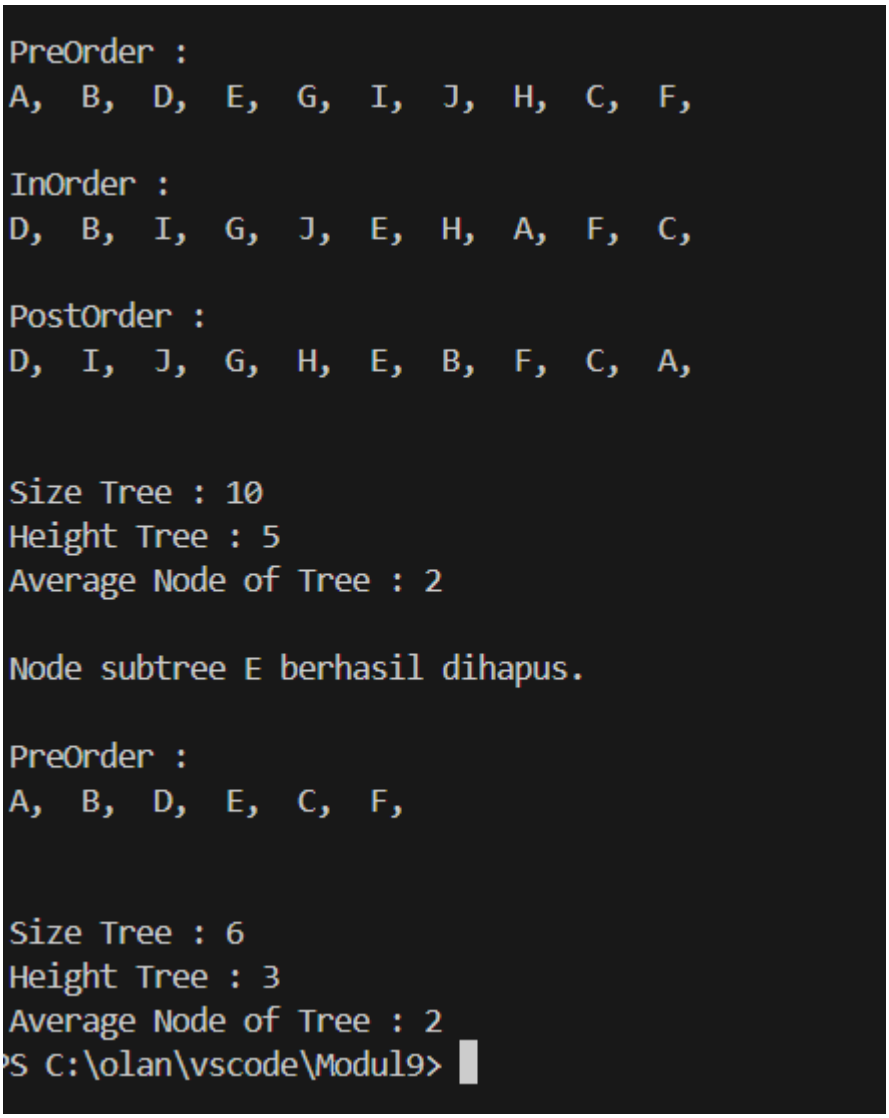
```
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// 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 C:\olan\vscode\Modul9> cd "c:\olan\vscode\Modul9\" ; if ($?)

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)
```



Deskripsi Program

Program di atas adalah sebuah program C++ yang mengimplementasikan struktur data pohon biner dengan berbagai operasi yang dapat dilakukan terhadapnya. Program dimulai dengan mendeklarasikan struktur `Pohon` yang terdiri dari data `char`, serta pointer ke kiri, kanan, dan parent. Variabel global `root` digunakan sebagai akar dari pohon, dan `baru` digunakan untuk node baru. Fungsi `init` digunakan untuk inisialisasi pohon dengan mengatur `root` menjadi NULL, sementara fungsi `isEmpty` memeriksa apakah pohon kosong.

Fungsi `buatNode` membuat node baru sebagai root jika pohon masih kosong, dan jika tidak, akan menampilkan pesan bahwa pohon sudah dibuat. Fungsi `insertLeft` dan `insertRight` digunakan untuk menambahkan anak kiri dan kanan pada node tertentu, dengan melakukan pengecekan apakah node yang dituju sudah memiliki anak kiri atau kanan. Fungsi `update` mengubah data dari node tertentu, dan `retrieve` menampilkan data dari node yang dituju. Fungsi `find` menampilkan informasi lengkap dari node yang dituju, termasuk data root, parent, sibling, dan anak kiri/kanan jika ada.

Program juga mencakup fungsi-fungsi untuk penelusuran pohon, seperti `preOrder`, `inOrder`, dan `postOrder`, yang masing-masing melakukan penelusuran pre-order, in-order, dan post-order. Fungsi `deleteTree` dan `deleteSub` digunakan untuk menghapus pohon secara keseluruhan atau subtree tertentu. Fungsi `clear` menghapus seluruh pohon dan mengatur root kembali menjadi NULL. Fungsi `size` mengembalikan jumlah node dalam pohon, sedangkan `height` mengembalikan tinggi pohon.

Program utama (`main`) mendemonstrasikan penggunaan fungsi-fungsi ini dengan membuat sebuah pohon dengan root 'A', dan menambahkan beberapa node lainnya. Program juga menunjukkan cara mengubah data node, menampilkan data node, dan mencari node tertentu. Setelah itu, program melakukan penelusuran pohon dengan metode pre-order, in-order, dan post-order. Fungsi `charateristic` menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Program ini juga mendemonstrasikan penghapusan subtree dan menampilkan kembali penelusuran pre-order serta karakteristik pohon setelah penghapusan subtree.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

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

using namespace std;

void printMatrix(const vector<vector<int>>& matrix, const
vector<string>& nodes) {
    int n = nodes.size();
    cout << setw(10) << " ";
    for (int i = 0; i < n; i++) {
        cout << setw(10) << nodes[i];
    }
    cout << endl;

    for (int i = 0; i < n; i++) {
        cout << setw(10) << nodes[i];
        for (int j = 0; j < n; j++) {
            cout << setw(10) << matrix[i][j];
        }
        cout << endl;
    }
}

int main() {
    int MaulanaGhaniRolanda_2311102012;
    cout << "Silahkan masukan jumlah simpul: ";
    cin >> MaulanaGhaniRolanda_2311102012;
    cin.ignore(); // Membersihkan buffer input

    vector<string> simpul(MaulanaGhaniRolanda_2311102012);
    vector<vector<int>> bobot(MaulanaGhaniRolanda_2311102012,
vector<int>(MaulanaGhaniRolanda_2311102012, 0));

    cout << "Silahkan masukan nama simpul" << endl;
```

```
    for (int i = 0; i < MaulanaGhaniRolanda_2311102012; i++) {
        cout << "Simpul " << i + 1 << ": ";
        getline(cin, simpul[i]);
    }

    cout << "Silahkan masukan bobot antar simpul" << endl;
    for (int i = 0; i < MaulanaGhaniRolanda_2311102012; i++) {
        for (int j = 0; j < MaulanaGhaniRolanda_2311102012; j++)
{
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << "\nMatriks Bobot:" << endl;
    printMatrix(bobot, simpul);

    return 0;
}
```

Screenshoot program

```
PS C:\olan\vscode\Modul9> cd "c:\olan\vscode\Modul9\" ;
Silahkan masukan jumlah simpul: 2
Silahkan masukan nama simpul
Simpul 1: BALI
Simpul 2: PALU
Silahkan masukan bobot antar simpul
BALI --> BALI = 1
BALI --> PALU = 2
PALU --> BALI = 2
PALU --> PALU = 1

Matriks Bobot:
                BALI      PALU
      BALI         1         2
      PALU         2         1
```

Deskripsi Program

Program di atas adalah program C++ yang memungkinkan pengguna untuk membuat graf berarah berbobot berdasarkan input pengguna, dan kemudian menampilkan matriks bobot dari graf tersebut. Program ini menggunakan header `iostream` untuk input-output, `vector` untuk menyimpan simpul dan bobot, serta `iomanip` untuk manipulasi format output. Program dimulai dengan mendefinisikan fungsi `printMatrix`, yang bertugas mencetak matriks bobot dengan format yang rapi, menampilkan nama-nama simpul di baris dan kolom pertama. Di dalam fungsi `main`, pengguna diminta untuk memasukkan jumlah simpul yang ingin dibuat. Variabel `MaulanaGhaniRolanda_2311102012` menyimpan jumlah simpul ini. Setelah itu, nama-nama simpul dimasukkan oleh pengguna melalui `getline`. Matriks bobot kemudian diisi dengan nilai yang diberikan oleh pengguna untuk setiap pasangan simpul.

Fungsi `printMatrix` kemudian dipanggil untuk menampilkan matriks bobot yang sudah diisi, dengan nama-nama simpul di baris dan kolom pertama. Matriks ini menampilkan jarak atau bobot antara setiap pasangan simpul yang diinputkan. Program ini efektif dalam menerima input dari pengguna, mengatur data dalam bentuk matriks, dan menampilkan hasilnya dengan format yang mudah dibaca. Dengan demikian, program ini memungkinkan pengguna untuk memodelkan graf berarah berbobot dan memvisualisasikan hubungan antar simpul dengan jelas.

2. **Unguided 2**

   Source code

```cpp
#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;
}
```

Screenshoot program

```
PS C:\olan\vscode\Modul9> cd "c:\olan\vscode\Modul9\" ; if ($?)

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 : pwt
Simpul 2 : pbg
Silahkan masukkan bobot antar simpul
pwt --> pwt = 1
pwt --> pbg = 4
pbg --> pwt = 4
pbg --> pbg = 1

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

                    pwt             pbg
          pwt         1               4
          pbg         4               1
```

Deskripsi program

Program tersebut adalah sebuah aplikasi sederhana yang memberikan pengguna kemampuan untuk memasukkan data pohon dan menampilkan matriks jarak antar simpul. Ketika program dijalankan, pengguna disambut dengan sebuah menu yang menawarkan tiga opsi: memasukkan data pohon, menampilkan matriks jarak antar simpul, atau keluar dari program.

Opsi pertama, "Masukkan data tree", memungkinkan pengguna untuk memasukkan jumlah simpul, nama simpul, dan bobot antar simpul. Input ini disimpan untuk digunakan dalam pembuatan matriks jarak antar simpul.

Opsi kedua, "Tampilkan matriks jarak antar simpul", menghasilkan tampilan matriks yang memperlihatkan bobot antar simpul berdasarkan input pengguna sebelumnya. Matriks ini mempermudah pemahaman terhadap jarak antar simpul dalam struktur pohon yang telah dimasukkan.

Pilihan terakhir, "Keluar", mengakhiri program dan keluar dari menu. Program akan terus berjalan dan menunggu input pengguna hingga opsi untuk keluar dipilih.

Fungsi inputTreeData bertanggung jawab atas pengambilan input dari pengguna, sementara fungsi displayMatrix digunakan untuk menampilkan matriks jarak antar simpul berdasarkan input yang telah dimasukkan sebelumnya.

Selama program berjalan, pengguna dapat memilih opsi yang diinginkan untuk memanipulasi data pohon dan melihat representasi visualnya dalam bentuk matriks.

BAB IV

KESIMPULAN

Kesimpulan laporan praktikum menyoroti pentingnya pemahaman konsep graph dan tree dalam ilmu komputer, serta kemampuan untuk menerapkan konsep tersebut dalam pemrograman. Praktikum ini memberikan pengalaman langsung dalam membuat kode untuk merepresentasikan dan memanipulasi struktur data yang mendasari banyak aplikasi dalam dunia nyata, seperti pemodelan jaringan, analisis data, dan optimisasi algoritma. Dengan demikian, laporan ini menjadi langkah awal yang penting dalam memahami konsep dasar yang fundamental dalam ilmu komputer.

DAFTAR PUSTAKA

Geeks For Geeks 2 Mei 2024. Tree Data Structure, Diakses pada 11 Mei 2024 dari https://www.geeksforgeeks.org/tree-data-structure/